



UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FELIPE COSTA FARIAS

Embarrassingly Parallel Autoconstructive Multilayer Perceptron Neural Networks

Recife
2022

FELIPE COSTA FARIAS

Embarrassingly Parallel Autoconstructive Multilayer Perceptron Neural Networks

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Área de Concentração: Inteligência Computacional

Orientador (a): Profa. Dra. Teresa Bernarda Ludermir

Coorientador (a): Prof. Dr. Carmelo José Albanez Bastos Filho

Recife

2022

Catálogo na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

F224e Farias, Felipe Costa
 Embarrassingly parallel autoconstructive multilayer perceptron neural networks / Felipe Costa Farias. – 2022.
 160 f.: il., fig., tab.

 Orientadora: Teresa Bernarda Ludemir.
 Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2022

 Inclui referências e apêndices.

 1. Inteligência computacional. 2. Redes neurais. I. Ludemir, Teresa Bernarda (orientadora). II. Título.

 006.31 CDD (23. ed.) UFPE - CCEN 2022-176

FELIPE COSTA FARIAS

“Embarrassingly Parallel Autoconstructive Multilayer Perceptron Neural Networks”

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Inteligência Computacional.

Aprovado em: 05/08/2022.

Orientador (a): Profa. Dra. Teresa Bernarda Ludermir

BANCA EXAMINADORA

Prof. Dr. Cléber Zanchettin
Centro de Informática / UFPE

Prof. Dr. Paulo Salgado Gomes de Mattos Neto
Centro de Informática / UFPE

Prof. Dr. Byron Leite Dantas Bezerra
Escola Politécnica de Pernambuco - EComp / UPE

Prof. Dr. Guilherme de Alencar Barreto
Departamento de Engenharia de Teleinformática / UFC

Prof. Dr. José Alfredo Ferreira Costa
Departamento de Engenharia Elétrica / UFRN

Dedicated to my daughter Maria Luísa Bacelar Farias.

ACKNOWLEDGEMENTS

I would like to thank God in the first place. Without him, I wouldn't be able to stay here. Furthermore, I would like to thank the State of Pernambuco and the Brazilian government for financing my academic knowledge.

To my parents Flávio Pontes Farias and Cláudia Costa Farias, my sister Flávia Costa Farias and all my family that gave me the opportunity and the necessary support for me to be where I am. Also, I would like to especially thank my wife Camila Pâmela Bacelar Barbosa Farias and my daughter Maria Luísa Bacelar Farias for understanding my absence during the work on this thesis and for always being by my side. They were indispensable for the conclusion of this process.

To Prof. Teresa Bernarda Ludermir, who gave me the opportunity to learn, to have very important discussions that contributed to this thesis, and to trust in my ideas. To Prof. Carmelo José Albanez Bastos Filho, who always supported, guided, and inspired me since my Bachelor's degree in Computer Engineering, during my Master's degree at University of Pernambuco, and now during this Doctoral degree. You both are very important to me and for the completion of this thesis, I feel very honored to have had world-class advisors like you both.

Thank you very much!

ABSTRACT

The present thesis proposes a method to automatically construct Multilayer Perceptron Artificial Neural Networks (MLP) to help non-expert users to still create robust models without the need to worry about the best combination of the number of neurons and activation functions by using specific splitting strategies, training parallelization, and multi-criteria model selection techniques. In order to do that, a data splitting algorithm (Similarity Based Stratified Splitting) was developed to produce statistically similar splits in order to better explore the feature space and consequently train better models. These splits are used to independently train several MLPs with different architectures in parallel (ParallelMLPs), using a modified matrix multiplication that takes advantage of the principle of locality to speed up the training of these networks from 1 to 4 orders of magnitude in CPUs and GPUs, when compared to the sequential training of the same models. It allowed the evaluation of several architectures for the MLPs in a very short time to produce a pool with a considerable amount of complex models. Furthermore, we were able to analyze and propose optimality conditions of theoretical optimal models and use them to automatically define MLP architectures by performing a multi-criteria model selection, since choosing a single model from an immense pool is not a trivial task. The code will be available at <<https://github.com/fariasfc/parallel-mlps>>.

Keywords: neural networks; neural networks architectures; neural networks topologies; parallelization; matrix multiplication; multi-criteria model selection.

RESUMO

A presente tese propõe um método para construir automaticamente Redes Neurais Artificiais Multilayer Perceptron (MLP) para ajudar os usuários não-especialistas a criar modelos robustos sem a necessidade de se preocupar com a melhor combinação do número de neurônios e funções de ativação, utilizando estratégias de particionamento de dados específicas, paralelização de treinamento e técnicas de seleção de modelos multicritério. Para isso, foi desenvolvido um algoritmo de particionamento de dados (Similarity Based Stratified Splitting) para produzir divisões estatisticamente semelhantes, a fim de explorar melhor o espaço de características e, conseqüentemente, treinar melhores modelos. Estas partições são usadas para treinar, de forma independente, várias MLPs com diferentes arquiteturas em paralelo (ParallelMLPs), usando uma multiplicação matricial modificada que faz uso do princípio da localidade para acelerar o treinamento destas redes de 1 a 4 ordens de magnitude em CPUs e GPUs, quando comparado ao treinamento seqüencial dos mesmos modelos. Isto permitiu a avaliação de várias arquiteturas de MLPs em um tempo muito curto para produzir um conjunto com uma quantidade considerável de modelos complexos. Além disso, pudemos analisar e propor condições de otimalidade de modelos ótimos teóricos, e usá-las para definir automaticamente arquiteturas de MLPs realizando uma seleção multi-critérios de modelos, uma vez que escolher um único modelo de um imenso conjunto não é uma tarefa trivial. O código estará disponível em <<https://github.com/fariasfc/parallel-mlps>>.

Palavras-chaves: redes neurais; arquiteturas de redes neurais; topologias de redes neurais; paralelização; multiplicação matricial; seleção de modelos multi-critérios.

LIST OF FIGURES

Figure 1 – Pareto front for each dataset.	28
---	----

LIST OF TABLES

Table 1	– Accuracy rates (%) for Multi-Layer Perceptron (MLP) architecture by dataset and number of hidden neurons. Each cell contains the averaged accuracy over 5x 10-fold cross-validations in the first row, in cases where sub-labels were created, new accuracy in the second row and Wilcoxon statistical test result in the third row – performance increase (▲), performance statistically equivalent (=).	22
Table 2	– Experiment Results comparing ordinary MLPs against our Data-driven Unraveling (DDU) proposal.	24
Table 3	– Number of losses, ties and wins regarding the models and similarities of the Similarity-Based Stratified Splitting (SBSS) over 10-fold splitting. .	24
Table 4	– Average of 10 epochs to train 10,000 models using a Central Processing Unit (CPU).	25
Table 5	– Average of 10 epochs to train 10,000 models using a Graphics Processing Unit (GPU).	26
Table 6	– Results for High Generalization Performance policies.	29
Table 7	– Results for High Robustness policies.	29
Table 8	– Results for Low Complexity policies.	30

LIST OF ABBREVIATIONS AND ACRONYMS

CNN	Convolutional Neural Networks
CPU	Central Processing Unit
DNN	Deep Neural Networks
DT	Decision Trees
GNN	Graph Neural Networks
GPU	Graphics Processing Unit
KNN	K-Nearest Neighbors
MCDM	Multi-Criteria Decision Making
MLP	Multi-Layer Perceptron
NAS	Neural Architecture Search
NIS	Negative Ideal Solution
NN	Artificial Neural Network
PIS	Positive Ideal Solution
RF	Random Forest
RL	Reinforcement Learning
RNN	Recurrent Neural Networks
SBSS	Similarity-Based Stratified Splitting
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TOPSIS	Technique for Order of Preference by Similarity to Ideal Solution

CONTENTS

1	INTRODUCTION	13
1.1	MOTIVATION	14
1.2	EXISTING SOLUTIONS	16
1.2.1	Automatic Architecture Neural Network Definition	16
1.2.2	Neural Networks Training Parallelization	16
1.2.3	Multi-Criteria Model Selection	17
1.3	OBJECTIVES	18
1.4	ORGANIZATION	18
2	MAIN CONTRIBUTIONS	20
2.1	ACADEMIC PRODUCTIONS	20
2.1.1	Survey on Network Science and Machine Learning	21
2.1.2	Analyzing the impact of data representations in classification prob-	
	lems using clustering	21
2.1.3	Clustering for Data-driven Unraveling Artificial Neural Networks . .	23
2.1.4	Similarity-Based Stratified Splitting	23
2.1.5	Embarrassingly Parallel Independent Training of Multi-Layer Per-	
	ceptrons with Heterogeneous Architectures	25
2.1.6	Have we been Naive to Select Machine Learning Models? Noisy	
	Data are here to Stay!	26
3	CONCLUSIONS AND FUTURE WORK	31
3.1	CONCLUSION	31
3.1.1	Future Works	32
	REFERENCES	34
	APPENDIX A – SURVEY ON NETWORK SCIENCE AND MA-	
	CHINE LEARNING	37
	APPENDIX B – ANALYZING THE IMPACT OF DATA REPRE-	
	SENTATIONS IN CLASSIFICATION PROBLEMS	
	USING CLUSTERING	81
	APPENDIX C – CLUSTERING FOR DATA-DRIVEN UNRAVEL-	
	ING ARTIFICIAL NEURAL NETWORKS	88

APPENDIX D – SIMILARITY BASED STRATIFIED SPLITTING: AN APPROACH TO TRAIN BETTER CLASSIFIERS	101
APPENDIX E – EMBARRASSINGLY PARALLEL INDEPENDENT TRAINING OF MULTI-LAYER PERCEPTRONS WITH HETEROGENEOUS ARCHITECTURES .	118
APPENDIX F – HAVE WE BEEN NAIVE TO SELECT MACHINE LEARNING MODELS? NOISY DATA ARE HERE TO STAY!	133

1 INTRODUCTION

Machine Learning is a research field that tries to extract knowledge from data and save it into a model that can be used to predict values from unseen data in the future. Usually, several models are created in order to find the best model possible to be used in production, solving a real-world problem. One of the most important challenges is how to select the model that we are going to use in production among several possibilities.

This knowledge is extracted during the model training process. We can resume the training process by understanding four core components of a Machine Learning system:

- Data: The data collected from a specific task that we want a model for.
- Model: Entity that is capable of storing information in its parameters and predict outputs for the task that it is trained for.
- Loss Function: A function to measure the success of the model or how close the predictions are from the targets in the data.
- Optimizer: A mechanism to adapt the model's parameters in order to minimize the loss function for a given specific Data.

If any component changes, it will directly affect the produced machine learning model. The role of the training algorithm is to use the Optimizer to update the Model's parameter in order to minimize the Loss when the Model is applied to the Data.

There are several Machine Learning models available with different complexities such as Decision Trees (DT) (QUINLAN, 1987), Random Forest (RF) (BREIMAN, 2001), K-Nearest Neighbors (KNN) (FIX; HODGES, 1989), Support Vector Machine (SVM) (CORTES; VAPNIK, 1995), Artificial Neural Network (NN) (GOODFELLOW; BENGIO; COURVILLE, 2016), Linear Regression (WEISBERG, 2005), Logistic Regression (KLEINBAUM et al., 2002), Gradient Boosting (FRIEDMAN, 2002), etc. All of these algorithms during the training process are affected by the Hyper-Parameters. We can define the Hyper-Parameters as the configurations regarding the model definition and the variables that guide the learning process. When creating a Machine Learning system, the user usually has to try a different set of hyper-parameters to find the best possible model for a given task he wants to solve. Depending on the model, the training process can be very computationally intensive and the Hyper-Parameter selection starts to become a challenge due to time restrictions.

The NN is a very common model used in Machine Learning due to its flexibility and good overall performance. There are several theoretical studies proving that NN are universal approximators (HORNIK; STINCHCOMBE; WHITE, 1989; SONTAG, 1992; LESHNO et al., 1993; PINKUS, 1999; NIELSEN, 2015) and, given enough resources, NN with a single hidden layer is capable of approximating every continuous function at some degree. Those are the main

reasons why we choose to work and optimize the NN training procedures. Training a NN is hard, but tricks can be used to enable a successful training (LIVNI; SHALEV-SHWARTZ; SHAMIR, 2014) such as (i) choosing the activation function, (ii) over-specification (larger than needed networks are easier to train), and (iii) regularization – regularizing the networks’ weights speeds up the convergence.

There are several types of Neural Networks (GOODFELLOW; BENGIO; COURVILLE, 2016) such as Multi-Layer Perceptron (MLP), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Graph Neural Networks (GNN), Deep Neural Networks (DNN), etc. The correct Neural Network family will depend on the type of data that you are trying to learn and the inductive bias used by the model.

The usually good Neural Network performance comes with the price of finding the best Hyper-Parameters. We can divide the NN Hyper-Parameters into two types: (i) architectural hyper-parameters (define the NN architecture), (ii) learning procedure hyper-parameters (define variables related to the optimization process). The most critical architecture Hyper-Parameters that define a NN architecture can be summarized by (i) number of layers, (ii) number of neurons in each layer, (iii) activation functions in each layer. There are also other very important parameters for the learning process such as the (i) learning rate, (ii) number of epochs, (iii) regularization strengths, etc. The architectural hyper-parameters are directly related to the complexity of the model.

The user usually has to find the best architecture in a multi-criteria scenario. Optimizing it regarding the model performance (low errors) and efficiency (computational complexity).

In this work, we have tried to democratize the NN architecture definition, since NN is a powerful and widely used ML technique applied to a variety of different problems, by selecting the best architecture given a hyper-parameter bounded space and a specific dataset. We proposed strategies to create minimal yet accurate MLPs in an automatic way, such that the user could rely on the machine to find the best architecture of the model for a given dataset. To do that, we have proposed some strategies that one could use to better split the dataset, to train several different MLPs in parallel, and to select robust models from an immense pool of candidates avoiding over-search (YING, 2019) using a multi-criteria decision-making process.

The remaining of this chapter is organized as follows. The Section 1.1 presents the motivation for our work. The Section 1.2 contains existing solutions related to the problems that we have investigated. The Section 1.3 we list the objectives of the thesis. The organization of this thesis is described in Section 1.4.

1.1 MOTIVATION

With the advance in the field, we can realize that the NN complexity has been growing through time, mainly after the Deep Learning advent. It is common to use models with millions/billions of parameters. Alongside with that, several researches with pruning techniques (BLALOCK et al., 2020) that are able to discard many parameters of the model and still maintain a comparable

or even better performance were published. If pruning methods are effective, it means that we are probably using more resources than needed, and it can directly impact the training algorithm, leading to under or over-fitting (BEJANI; GHATEE, 2021) issues.

Another important and recent topic is the Lottery-Ticket Hypothesis (FRANKLE; CARBIN, 2018). It states that in a randomly initialized neural network, a sub-network exists which, if isolated trained, its performance is comparable with the original network. The works (RAMANUJAN et al., 2020; MALACH et al., 2020) found that a random initialized sufficiently over-parameterized neural network contains a sub-network (without any training) that achieves competitive accuracy if compared to the trained large network.

During the last years, several papers were published about AutoML (HE; ZHAO; CHU, 2021). AutoML is a research area that explores techniques used to search for ML models (and their hyper-parameters) that better solve a specific problem. Regarding NN, AutoML can try different architectural and non-architectural hyper-parameters to find the best combination that better describes a dataset.

Several search algorithms were also assessed to perform model search/selection for NN such as Evolutionary Algorithms optimizing the NN weights, architecture and/or activation functions (SCHAFFER; WHITLEY; ESHELMAN, 1992). Usually, those methods are chosen because knowing the entire architecture hyper-parameter search space might be very expensive. Therefore, those search algorithms try to find good solutions based on a few point measurements.

If we deeply analyze any function optimization problem, we can investigate three extreme scenarios: (i) we know the formation rule (equation) that maps the search space and the optimized variable. In this scenario, we could just calculate the derivative of the function and find the critical points; (ii) we know all the measurement points mapping the search space and the optimized variable where we could pick the point that maximizes/minimizes all the measured points. This is usually very expensive to compute; And (iii) we know very few measurement points and we need to choose the most promising region to explore next. This is often a sequential approach in which the decision of the next region will depend on all the previous knowledge about the search space (all the previous measurement points are used to guide the search process). Since we don't know the mathematical equation that dictates the NN architectural hyper-parameter mapping to model performance from the scenario (i), people usually try the scenario (iii) when optimizing the architecture of the model because scenario (ii) tends to be much more time consuming, usually infeasible when assessing the models sequentially, and often suffers from over-searching problems (YING, 2019) leading to the selection of over-fitted models. Theoretically, the scenario (ii) would be the best one, since we would assess all the possible architectures. That's why we have proposed the algorithm in (FARIAS; LUDERMIR; BASTOS-FILHO, 2022a), which differ from the other mentioned approaches when performing the search in scenario (ii), but several magnitudes faster than the sequential approach would take.

As the NN are capable of learning non-linear complex and/or hidden relationships between

input-output pairs in order to make predictions, they are suited to solve real-world problems and help during the decision-making process.

Therefore, this thesis focus on the development of techniques to improve the learning process of machine learning models, NN architecture search through training parallelization, and multi-criteria model selection procedures that perform a cost-benefit analysis regarding performance and model complexity.

1.2 EXISTING SOLUTIONS

1.2.1 Automatic Architecture Neural Network Definition

The challenge to automatically find the most suited architecture for a NN given a specific dataset is a problem that attracted attention to various researchers. This is an important step during modeling with NN. It tends to be very time-consuming and non-experts have difficulties to optimize the compromise between the number of neurons and the performance of the model due to the bias-variance trade-off. In order to automatically suggest the architecture of NNs, multiple researchers proposed Constructive Neural Network (SHARMA; CHANDRA, 2010) algorithms, such as the Cascade Correlation (FAHLMAN; LEBIERE, 1989), Dynamic Node Creation (ASH, 1989) and C-MANTEC (SUBIRATS; FRANCO; JEREZ, 2012). Those works commonly try to create a NN in an incremental way such as adding or removing neurons/layers in a greedy process. One of the most common ways to define NN architectures is to use Evolutionary Algorithms to optimize the network topology and also the weights (STANLEY; MIIKKULAINEN, 2002; ZANCHETTIN; LUDERMIR; ALMEIDA, 2011; DING et al., 2013).

Recently, the field of Neural Architecture Search (NAS) emerged (REN et al., 2021). In this field, the authors are often searching algorithms with a better defined process that divides the framework into small components and use intelligent search algorithms to optimize. Another approach, is to use Reinforcement Learning (RL) (ZOPH; LE, 2016) to guide the search process.

Most of the previously cited solutions contain sequential steps to be performed that use the accumulated knowledge of the previous steps to decide the next direction in the search process. Furthermore, they usually use a single-criterion approach to optimize, which might lead to over-fitted solutions potentially caused by over-searching. In our proposal, we parallelize the architectures to speed up the training and use a multi-criteria strategy to overcome the over-searching problem.

1.2.2 Neural Networks Training Parallelization

Since the search for the best architecture implies the training of several NN, decreasing the required amount of time to train the models is very important. One of the ways to decrease the training time is through parallelization (BEN-NUN; HOEFLE, 2019; FARKAS; KERTÉSZ; LOVAS, 2020). The recent advance of computational power is prominently noticed as parallelization

of Central Processing Unit (CPU) and Graphics Processing Unit (GPU). We can classify the parallelization in two different strategies:

- **Single Machine:** The parallelization happens in a single computer in the form of GPU and multi-core or multi-thread processors with a shared memory space.
- **Multiple Machines:** In this setup, parallelization emerges from using several machines interconnected by a network. This setup comes with several challenges of synchronization, data sharding, communication overhead and other difficulties.

We can also analyze the concurrency in the entire network evaluation by dividing it into 3 classes:

- **Data Parallelism:** Mostly related to mini-batching in Stochastic Gradient Descent (SGD) optimizers and data partitioning across different processes/machines.
- **Model Parallelism:** Divides the network into different parts that are calculated by different processes.
- **Pipelining:** Uses overlapping computations of different parts of the NN. It can compute the forward phase in consecutive layers using previously calculated values from previous layers or assign each layer to a specific processor.

In this thesis, we have proposed another type of parallelism and concurrency which is the **Architecture Parallelism** in our ParallelMLP work (FARIAS; LUDERMIR; BASTOS-FILHO, 2022a). This approach allows us to use a Single Machine parallelization to concurrently train heterogeneous architectures that vary the number of neurons and activation functions.

1.2.3 Multi-Criteria Model Selection

Once several models were trained and evaluated, the task to choose the best model is usually a complicated challenge. Most of the time the model selection relies on a single-criterion that must be maximized or minimized. This can be very problematic due to the inherent presence of noise in real-world data that can lead to over-fitted choices.

Although single-criterion is probably the most common approach to select machine learning models, there are works that try to perform a multi-criteria model selection such as in (ALI; LEE; CHUNG, 2017), that proposed a multi-criteria decision making methodology using accuracy, time and consistency of each model to select the best one. It uses the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) ranking to measure the distance to the ideal classifier (TZENG; HUANG, 2011). The TOPSIS were also used in (VAZQUEZ et al., 2020) to model selection, but only used different performance metrics for a specific set.

We argue that the machine learning algorithms must be interpreted as a whole, instead of summarizing it into a single number in order to select it as the best model. That's the

reason we have proposed a multi-criteria model selection procedure that includes not only the Validation/Holdout set, but also the Training set alongside with the architectural complexity to the decision process.

1.3 OBJECTIVES

This work aims to develop methodologies to automatically define specialized MLP architectures for each dataset, without the intervention of specialists, to be used in production, predicting real world data. To do that we rely on parallel model training, and strategies to improve the learning process and select good models.

Some specific objectives of this thesis are:

- Propose data splitting approaches to better train machine learning models.
- Investigate how we can accelerate the training process given our current computational power, software and hardware resources.
- Propose techniques to automatically define MLP architectures (number of neurons and activation functions).
- Discuss specific characteristics that a good model should have.
- Run experiments to validate the proposals.

1.4 ORGANIZATION

This work is organized as a set of articles. Some chapters were included to summarize the techniques and also to present additional information.

In Chapter 2 we present our key contributions and discuss their importance to accomplish different objectives, briefly explaining each produced paper and their most important results. In Chapter 3 we give our conclusions and future work directions to improve our proposals.

The content of the Appendices from A to F are given with the entire production with details of each work. These papers defined the work developed in this thesis.

In Appendix A we present a survey relating Machine Learning, Neural Network, Deep Learning and Complex Networks/Network Science.

The Appendix B investigated how to better represent output data labels to be used in supervised training of classifiers. We have tried to group cohesive patterns into their respective clusters assigning them sub-labels.

The Appendix C extends the Appendix B and investigates how to define NN architectures in a data-driven approach using clustering to create sub-labels to facilitate the learning process and discover the number of neurons needed to compose the layers.

The Appendix D analyzes a technique responsible for splitting the data using both the output and input space information. It generates splits where the data may better represent

the task to the models during the training phase, probably leading to more realistic performance estimation when used in real-world applications.

The Appendix E shows a way to parallelize the training of heterogeneous MLP architectures. We were able to increase from 1 to 4 orders of magnitude the trainings in CPUs and GPUs by using strategies that explores the principle of locality.

Finally, the Appendix F we propose to perform model selection based on a multi-criteria decision making procedure. We have shown that it mitigates problems such as over-searching and over-fitting when selecting a model in a large pool of candidates. Problems which usually happen when a single-criterion model selection procedure is used.

2 MAIN CONTRIBUTIONS

In this chapter, we present the main contributions of this research, which resulted in 6 articles and contributed to the method development of the automatic definition of Multilayer Perceptron (MLP) Artificial Neural Network models.

- We proposed a stratified data splitting process that uses the similarity of the features to assign samples to each split. It is beneficial because it tries to cover the entire feature space during the training and tends to produce better metrics estimation for the splits.
- We suggested a modification to the default matrix multiplication applied inside the MLP layers projections. This is crucial to allow the training parallelization of individual MLPs with heterogeneous architectures and decrease the time by several orders of magnitude.
- We defined optimality conditions that theoretical optimal models are expected to have. We use them to select models as a multi-criteria decision process. It is important because it alleviates the chance of selecting over-fitted models due to over-search problems.

We first train a large set of MLPs with different architectures on different portions of the data in parallel using the ParallelMLP technique. After that, we select the best model according to a multi-criteria model ranking procedure.

2.1 ACADEMIC PRODUCTIONS

We list in this section the papers written during the course of the PhD. We have produced 6 papers, 1 was not published, 2 of them were published, and the remaining 3 are in the review process.

- Not Published
 - Survey on Network Science and Machine Learning
- Published
 - Analyzing the impact of data representations in classification problems using clustering (International Joint Conference on Neural Networks – IJCNN 2019) (FARIAS et al., 2019)
 - Clustering for Data-driven Unraveling Artificial Neural Networks (Encontro Nacional de Inteligencia Artificial e Computacional – ENIAC 2020) (FARIAS; LUDERMIR; BASTOS-FILHO, 2020a)
- Submitted – In Review Process

- Embarrassingly Parallel Independent Training of Multi-Layer Perceptrons with Heterogeneous Architectures (also in ArXiv (FARIAS; LUDERMIR; BASTOS-FILHO, 2022a))
- To be Submitted
 - Similarity Based Stratified Splitting: an approach to train better classifiers (ArXiv) (FARIAS; LUDERMIR; BASTOS-FILHO, 2020b)
 - Have we been Naive to Select Machine Learning Models?

2.1.1 Survey on Network Science and Machine Learning

In this survey, we investigated 87 papers published between 2005 to 2017. We have tried to highlight evidences on the connections between Network Science, Machine Learning/Neural Networks/Deep Learning by focusing on Complex Networks measurements, the metrics to evaluate performances, the Machine Learning/Neural Networks/Deep Learning and Network Science techniques often used and where these algorithms were applied.

Although the subject of this thesis diverged from this topic, it was very important to analyze the NN from a Network Science standpoint by representing them as a graph of interconnected nodes which we could use Network Science techniques to better understand the learning dynamics. It gave us the opportunity to better understand what happens during the learning process, since we had to think about the "information flow" of each neuron, their interaction with activation functions, and the gradient flowing back during the back-propagation in the low-level of tensor operations. With this detailed understanding of the learning dynamics, we were able to suggest different ideas in order to improve the training of NN. The first idea resulted in the IJCNN publication detailed in the next section.

2.1.2 Analyzing the impact of data representations in classification problems using clustering

The motivation of this paper was to start investigating how we could better train models by looking into the data perspective.

In this paper (FARIAS et al., 2019), we investigated how to better represent output data labels to be used in the supervised training of classifiers. We have tried to group cohesive patterns into their respective clusters assigning them sub-labels. Twelve benchmark datasets were used. The technique first creates clusters, and when appropriate, new sub-labels are generated according to Fuzzy-CMeans and Silhouette score thresholds. After that, MLPs were employed to model the 12 datasets with the cluster-generated sub-labels. We observed that, whenever the algorithm created sub-labels, the performance increased in 22 cases, remaining statistically equivalent in 14 cases, according to statistical tests with significance $p\text{-value}=0.05$.

The results of this paper are summarized in Table 1.

Table 1 – Accuracy rates (%) for MLP architecture by dataset and number of hidden neurons. Each cell contains the averaged accuracy over 5x 10-fold cross-validations in the first row, in cases where sub-labels were created, new accuracy in the second row and Wilcoxon statistical test result in the third row – performance increase (▲), performance statistically equivalent (=).

Dataset	MLP Number of Hidden Neurons					
	10	30	50	100	150	200
balance-scale	88.17	93.18	95.24	96.67	96.89	96.86
breast	95.79	95.85	95.91	95.88	95.82	95.76
			96.08	96.05	96.08	96.19
breast-cancer-wisconsin	96.88	97.13	97.19	97.37	97.40	97.65
	97.44	97.72	97.76	97.79	97.80	97.76
	▲	▲	▲	▲	▲	=
diabetes	76.92	77.05	77.28	77.67	77.13	77.46
dna	94.17	94.71	95.07	95.28	95.24	95.31
ecoli	86.73	87.48	88.28	87.78	87.92	87.96
			88.65	88.38		
iris	94.80	95.60	96.27	96.40	96.67	96.67
	96.27	96.67	96.80	97.20	97.33	97.33
	=	▲	=	▲	▲	▲
mushroom	99.92	99.99	99.99	99.99	100	100
	99.98	100	100	100	100	100
	▲	▲	▲	▲	=	=
pendigits	97.20	98.74	98.98	99.12	99.16	99.20
	97.46	98.82	99.00	99.19	99.20	99.25
	▲	=	=	▲	▲	=
pima	76.67	77.14	77.08	77.16	77.08	76.96
satimage	83.67	84.76	85.71	85.93	86.25	86.12
	84.86	85.80	86.67	86.79	87.14	87.57
	▲	▲	▲	▲	▲	▲
vehicle	78.27	79.73	80.06	80.30	80.11	80.25

Source: Author's

own elaboration.

The sub-label process was able to create better representations of the output labels, improving the MLP performance during the learning process. This performance increase may also happen due to more tight groups of data of the same sub-label, after the clustering phase.

2.1.3 Clustering for Data-driven Unraveling Artificial Neural Networks

We started this work (FARIAS; LUDERMIR; BASTOS-FILHO, 2020a) as an extension of the latest paper (FARIAS et al., 2019). Given the results obtained, we start exploring methods to automatically define the architecture of a NN.

Here we have performed an investigation on how to define NN architectures in a data-driven approach using clustering to create sub-labels to facilitate the learning process and discover the number of neurons needed to compose the layers. The model depth was also increased in order to better represent the samples the deeper their representations flow into the model. The clustering process using Gaussian Mixture Models (GMM) was firstly applied to create sub-labels. After finding the sub-labels, another GMM clustering process was applied to define the number of neurons to be individually trained in a one vs. all fashion, and finally combining them to create the layer. After that, we try to reapply the whole process but now with the updated representation of the samples to create consequent layers. We used 7 benchmark datasets and 3x10-fold cross-validation to experiment and test our hypothesis. The proposed model increased the performance in some scenarios while never decreasing it in the remaining cases with statistical significance considering $p\text{-value}=0.05$.

The results of this approach are reported in Table 2.

In this work, we have done a preliminary study on how one could define a Data-Driven Neural Network Architecture using GMMs clustering to define the width of each layer and iteratively append layers, increasing depth, aiming to find better and possibly disentangled representations, without the need to create several different NN.

We have realized that the validation metrics increased more often than test metrics, probably indicating a bad splitting procedure. Also, the training was very slow since we had to train individual neurons to combine them after. In order to tackle those two limitations, we have developed the papers (FARIAS; LUDERMIR; BASTOS-FILHO, 2020b) and (FARIAS; LUDERMIR; BASTOS-FILHO, 2022a), respectively.

2.1.4 Similarity-Based Stratified Splitting

In a tentative to remedy the issues we have raised in the last paper, we have tried to create strategies to improve the splitting procedure.

In this article (FARIAS; LUDERMIR; BASTOS-FILHO, 2020b) we proposed the Similarity-Based Stratified Splitting (SBSS). A technique responsible for splitting the data using both the output and input space information. SBSS generates splits where the data may better represent the task to the models during the training phase, probably leading to more realistic performance estimation when used in real-world applications. We have evaluated the application of SBSS in

Table 2 – Experiment Results comparing ordinary MLPs against our Data-driven Unraveling (DDU) proposal.

Model	Dataset	Train	Validation	Test
DDU	pima	77.99 (0.01)	+ 78.48 (0.02)	76.74 (0.03)
MLP	pima	78.07 (0.01)	76.71 (0.03)	76.39 (0.02)
DDU	vehicle	+ 84.12 (0.02)	+ 80.98 (0.03)	+ 80.90 (0.04)
MLP	vehicle	82.81 (0.01)	79.34 (0.02)	79.40 (0.03)
DDU	glass	66.88 (0.05)	+ 65.01 (0.05)	62.16 (0.10)
MLP	glass	66.09 (0.03)	60.83 (0.06)	62.33 (0.09)
DDU	tic-tac-toe	82.33 (0.06)	77.89 (0.06)	67.14 (0.11)
MLP	tic-tac-toe	82.46 (0.04)	77.98 (0.05)	70.79 (0.10)
DDU	iris	96.04 (0.02)	97.16 (0.03)	95.33 (0.06)
MLP	iris	+ 97.36 (0.01)	95.98 (0.03)	95.56 (0.06)
DDU	satimage	+ 91.16 (0.01)	+ 89.52 (0.01)	+ 88.86 (0.02)
MLP	satimage	89.58 (0.01)	88.17 (0.01)	88.06 (0.01)
DDU	ionosphere	93.50 (0.02)	+ 89.37 (0.03)	84.82 (0.08)
MLP	ionosphere	93.70 (0.01)	88.06 (0.03)	85.96 (0.08)

Source:

Author's own elaboration.

a 10-fold splitting called Similarity-Based Stratified 10-Fold Splitting in 22 benchmark datasets from UCI (DUA; GRAFF, 2017), on Multi-Layer Perceptrons, Support Vector Machine, Random Forests, and K-Nearest Neighbors, comparing five different similarity functions: Cityblock, Chebyshev, Cosine, Correlation, and Euclidian. According to the Wilcoxon tests, our approach consistently outperformed ordinary stratified 10-fold cross-validation in 75% of the assessed scenarios. It was also able to decrease the generalization gap of the models.

The results for each similarity function are summarized in Table 3.

Table 3 – Number of losses, ties and wins regarding the models and similarities of the SBSS over 10-fold splitting.

Model	Chebyshev			Cityblock			Euclidean			Cosine			Correlation		
	loss	tie	win	loss	tie	win	loss	tie	win	loss	tie	win	loss	tie	win
KNN	2	4	16	0	3	19	0	6	16	3	2	17	1	5	16
MLP	3	4	15	1	7	14	2	1	19	3	3	16	1	4	17
RF	0	5	17	0	4	18	0	6	16	1	8	13	0	5	17
SVM	1	5	16	0	3	19	0	5	17	1	5	16	0	6	16
Total	6	18	64	1	17	70	2	18	68	8	18	62	2	20	66
%	6.82	20.45	72.73	1.14	19.32	79.55	2.27	20.45	77.27	9.09	20.45	70.45	2.27	22.73	75

Source: Author's own elaboration.

Table 4 – Average of 10 epochs to train 10,000 models using a CPU.

	Number of Samples								
	100			1000			10000		
	Batch Size								
	32	128	256	32	128	256	32	128	256
Features	Parallel (Seconds)								
5	0.525	0.463	0.472	5.248	4.709	4.717	52.737	46.929	47.133
10	0.539	0.466	0.475	5.338	4.722	4.742	53.351	47.089	47.153
50	0.658	0.505	0.501	6.144	4.943	4.887	62.664	49.791	48.85
100	0.809	0.547	0.551	7.373	5.366	5.1	74.661	53.09	50.965
	Sequential (Seconds)								
5	13.437	6.097	5.994	112.054	56.999	48.852	1097.599	564.428	483.278
10	13.36	6.115	6.01	111.84	57.094	49.015	1097.503	564.701	484.91
50	13.884	6.571	6.467	116.303	58.993	49.546	1134.955	583.468	491.269
100	14.283	6.671	6.592	120.297	59.259	50.371	1179.405	586.267	493.744
	Parallel/Sequential (%)								
5	3.91	7.59	7.88	4.684	8.262	9.656	4.805	8.314	9.753
10	4.038	7.625	7.905	4.773	8.27	9.674	4.861	8.339	9.724
50	4.739	7.69	7.753	5.282	8.379	9.863	5.521	8.534	9.944
100	5.664	8.198	8.362	6.129	9.056	10.126	6.33	9.056	10.322

Source: Author's own elaboration.

2.1.5 Embarrassingly Parallel Independent Training of Multi-Layer Perceptrons with Heterogeneous Architectures

Since the training execution times were an issue in (FARIAS; LUDERMIR; BASTOS-FILHO, 2020a), we have tried to speed up the MLP training process.

In this paper (FARIAS; LUDERMIR; BASTOS-FILHO, 2022a), we proposed an algorithm called ParallelMLPs to train heterogeneous MLPs (with different architectures w.r.t. the number of neurons and activation functions) in an Embarrassingly Parallel way by aggregating several MLPs into a single architecture that uses a modified matrix multiplication procedure to make the gradient flow independent. We have also used strategies that maximize the principle of locality to speed up the training process. We were able to see 2-4 orders of training speed up when training 10,000 models with a different number of neurons (from 1 to 100) in the hidden layer; 10 different activation functions in simulated datasets containing 100, 1000, and 10000 samples; 5, 10, 50, and 100 features presented as batches of size 32, 128, and 256 in both CPU and GPU devices.

The main results of this paper are summarized in Tables 4 and 5

In this work, we tackled the problem of training time when using grid-search by using the parallelism of our current computing devices (CPUs and GPUs) combined with several tricks to

Table 5 – Average of 10 epochs to train 10,000 models using a GPU.

	Number of Samples								
	100			1000			10000		
	Batch Size								
	32	128	256	32	128	256	32	128	256
Features	Parallel (Seconds)								
5	0.024	0.002	0.001	0.269	0.177	0.203	2.676	1.756	2.426
10	0.025	0.002	0.001	0.276	0.178	0.206	2.745	1.767	2.439
50	0.033	0.002	0.001	0.351	0.193	0.218	3.483	1.926	2.569
100	0.043	0.002	0.001	0.449	0.215	0.233	4.438	2.128	2.75
	Sequential (Seconds)								
5	22.911	8.646	8.511	189.411	73.503	57.02	1857.653	722.915	566.592
10	22.983	8.619	8.515	188.966	73.462	56.941	1859.14	722.925	568.583
50	23.025	8.628	8.519	189.147	73.364	57.134	1858.07	719.359	567.847
100	22.993	8.581	8.503	189.015	72.849	57.129	1854.881	717.543	566.248
	Parallel/Sequential (%)								
5	0.106	0.019	0.017	0.142	0.241	0.355	0.144	0.243	0.428
10	0.11	0.019	0.017	0.146	0.243	0.362	0.148	0.245	0.429
50	0.142	0.019	0.017	0.185	0.264	0.381	0.187	0.267	0.452
100	0.186	0.018	0.017	0.237	0.294	0.408	0.239	0.297	0.486

Source: Author's own elaboration.

explore the *principle of locality*. With that, we could test several possibilities of MLPs instead of trying to create the model step by step as in the previously mentioned papers (FARIAS et al., 2019; FARIAS; LUDERMIR; BASTOS-FILHO, 2020a).

2.1.6 Have we been Naive to Select Machine Learning Models? Noisy Data are here to Stay!

Since we were able to train the MLPs very quickly with the proposed technique in (FARIAS; LUDERMIR; BASTOS-FILHO, 2022a), we start to examine how to select the best model given a large pool of complex models trained on a specific dataset.

In this work (FARIAS; LUDERMIR; BASTOS-FILHO, 2022b), we discuss the expected characteristics that theoretical optimal models should have and assessed several ways to perform model selection once we were able to train several models at the same time when using (FARIAS; LUDERMIR; BASTOS-FILHO, 2022a). In this work, we argue that the over-searching problem that often leads to the selection of over-fitted models can be alleviated if using a multi-criteria model selection procedure instead of a single-criterion, which is commonly used. We assessed the techniques in 14 UCI(DUA; GRAFF, 2017) benchmark datasets. Each dataset experiment was repeated 18 times. For each experiment, we have created 5,600 models. Here we list the

four optimality conditions that we expect our theoretical optimal models should ideally have:

- High Generalization Performance: Holdout and Test sets should have the highest performance possible;
- High Robustness: Train, Validation, Holdout and Test performances should be similar;
- Low Complexity: Models with a low number of neurons should be preferred;
- No Premature Early Stopping: Models that stopped learning at the beginning of the process probably have initialized in a bad place.

To study the relationship of those optimality conditions in an immense pool of NN models, we have used three aggregation policies:

- Individual: the models are not aggregated by architecture;
- Local: the models are aggregated by architecture within each run;
- Global: the models are aggregated by architecture considering all the runs.

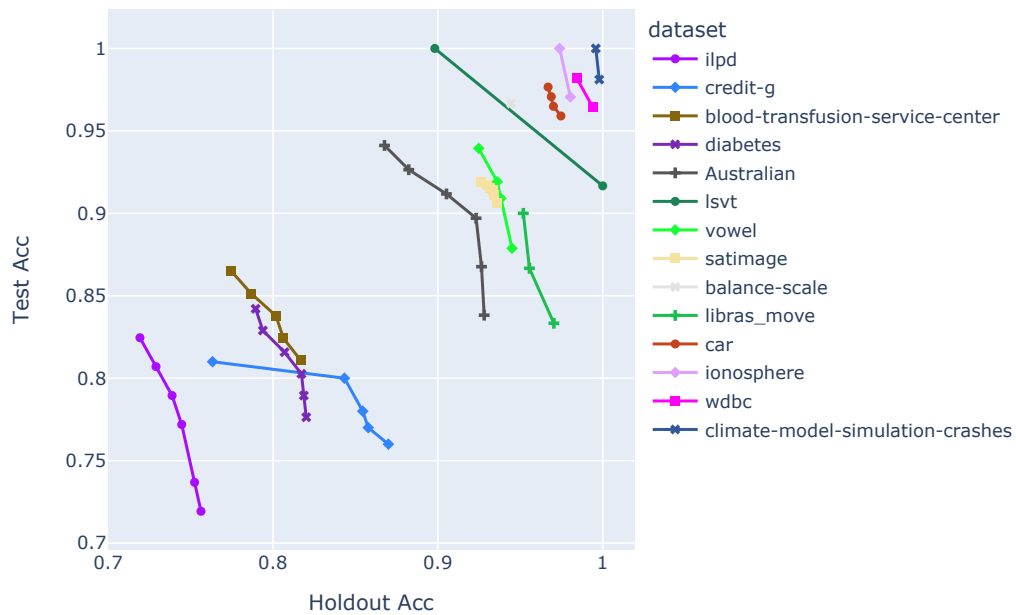
and several ranking policies in order to compare the single-criterion model selection against multi-criteria model selections using the Multi-Criteria Decision Making (MCDM) (TRIANTAPHYLLOU, 2000; ARULDOSS; LAKSHMI; VENKATESAN, 2013) algorithm called TOPSIS (TZENG; HUANG, 2011). In this algorithm, we select the model that minimizes the geometric distance from the Positive Ideal Solution (PIS) and simultaneously maximizes the geometric distance from the Negative Ideal Solution (NIS). The PIS and NIS can be understood as the best possible model (maximum metrics, one neuron, maximum stopped epoch) and the worst possible model (minimum metrics, one hundred neurons, minimum stopped epoch), respectively. After ranking the models, a Pareto Dominance filter is applied to remove the dominated solutions and select the best-ranked model among the non-dominated models.

The Figure 1 present the non-dominated models for each dataset. We can realize that if one would select the model based only on the Holdout performance, usually the Test performance is not the best possible, supporting our hypothesis that single-criterion model selection is not the best approach for model selection.

To analyze the following results, we have compared single-criterion policies (Holdout, Test) against multi-criteria policies where the first T comes from the TOPSIS algorithm, then we might have different letters meaning different things. The first T stands for Training metrics, V for Validation, H for Holdout, the last T for Test, and N for Number of Neurons included in the multi-criteria decision process.

One of the most important findings of this work is depicted in Table 6. When comparing the aggregation policies, it seems that analyzing the best architecture for a given data combination (the Individual grouping) is preferable to aggregating the architecture results and taking the

Figure 1 – Pareto front for each dataset.



Source: Author's own elaboration.

average to choose the “best architecture”. It contributes to our hypothesis that freezing the NN architecture is not the best approach. The models will experience different initialization and be exposed to slightly different training data. Therefore, they will experience different error landscapes, subjected to their learning dynamics.

The Individual, Local, and Global Holdout policies contain equivalent Test accuracies, but the Individual-Holdout also decreased the Number of Neurons. Since we did not need to perform cross-validation to group architectures within the k-fold splitting to select reasonable models, we could use the time invested into the k-fold repetitions to actually try different hyper-parameters.

We can see in Table 7 that if we compare the single-criterion Holdout with the TTVH policy – which simultaneously try to maximize not only the Holdout set metrics, but also the Training and Validation set metrics –, TTVH could select models with statistically equivalent results regarding the Test performance, while at the same time it was able to decrease the Disagreement (average absolute difference of the sets performances) if compared to the Holdout significantly, probably indicating more robust models. In order to analyze the best model on the Test set (which maximizes the Test metric), we can compare the Test vs. the TTVHT policy (which uses the Train, Validation, Holdout, and Test metrics at the same time to make a decision). We can see that the Test policy probably has selected an over-estimated model since the Train, Validation, and Holdout sets presented very poor performances. We advocate that TTVHT would be the closest to the real best model since it performs similarly in all the sets and still present an interesting Test metric, while decreasing the Disagreement.

If we include the Number of Neurons to be minimized in the decision-making process,

Table 6 – Results for High Generalization Performance policies.

Policy		# Neurons ↓	# Epochs ↑	Accuracy ↑				Disagreement ↓		
				Train	Validation	Holdout	Test	Train-Validation	Holdout-Test	All
Individual	Train	73.47	86.02	92.45	87.47	91.28	85.06	5.03	6.4	4.66
		(29.13)	(24.6)	(7.72)	(7.8)	(7.71)	(9.5)	(4.21)	(6.84)	(4.05)
	Validation	47.02	62.99	87.85	91.86	88.97	84.25	4.26	5.21	4.38
		(34.84)	(35.21)	(8.8)	(6.85)	(7.98)	(9.78)	(3.96)	(6.76)	(3.83)
	Holdout	76.41	84.79	92.08	89.94	91.76	85.54	2.77	6.37	3.94
		(27.5)	(25.13)	(7.92)	(7.79)	(7.66)	(9.29)	(2.66)	(6.53)	(3.51)
Test	42.8	63.72	87.42	87.07	87.68	90.61	2.67	4.29	3.13	
	(33.91)	(34.72)	(8.8)	(7.64)	(7.98)	(7.05)	(3.25)	(3.51)	(2.41)	
	75.5	78.21	91.22	89.05	90.9	89.65	2.83	3.52	2.67	
THT	(24.61)	(27.19)	(8.36)	(7.89)	(8.07)	(7.34)	(2.67)	(3.26)	(2.0)	
Local	Train	85.19	80.42	91.75	88.0	90.95	85.21	3.86	5.98	4.12
		(18.79)	(26.98)	(8.21)	(7.93)	(8.0)	(9.43)	(3.6)	(7.27)	(4.09)
	Validation	80.45	62.89	89.07	90.8	89.65	85.14	2.68	4.94	3.52
		(20.88)	(33.46)	(8.9)	(7.08)	(8.21)	(9.38)	(2.69)	(6.1)	(3.19)
	Holdout	86.41	78.35	91.52	89.37	91.2	85.01	2.76	6.36	3.94
		(17.5)	(27.47)	(8.01)	(7.6)	(7.71)	(9.6)	(2.76)	(7.27)	(3.97)
Test	68.52	61.08	88.61	88.13	88.77	89.05	2.28	3.46	2.48	
	(30.98)	(34.2)	(9.19)	(7.81)	(8.51)	(7.49)	(2.62)	(3.28)	(2.11)	
	82.75	68.67	90.56	88.58	90.28	88.76	3.26	3.58	2.86	
THT	(19.23)	(30.14)	(8.51)	(7.57)	(8.03)	(7.59)	(3.32)	(3.79)	(2.4)	
Global	Train	91.86	75.19	91.43	87.85	90.69	85.34	3.73	5.55	3.85
		(11.95)	(27.65)	(8.26)	(7.9)	(8.01)	(9.22)	(3.9)	(6.55)	(3.87)
	Validation	92.29	61.85	89.17	90.13	89.51	84.91	1.96	5.19	3.33
		(13.52)	(29.62)	(8.28)	(7.42)	(7.95)	(9.8)	(1.89)	(6.77)	(3.48)
	Holdout	87.36	75.85	90.99	88.78	90.83	85.48	3.08	5.49	3.67
		(23.97)	(26.96)	(8.93)	(7.87)	(8.02)	(9.18)	(2.75)	(6.39)	(3.47)
Test	85.0	62.13	89.37	88.26	89.24	87.69	2.24	3.4	2.42	
	(21.18)	(32.4)	(8.82)	(8.03)	(8.46)	(8.17)	(3.15)	(4.4)	(2.74)	
	89.36	69.11	90.04	88.45	90.21	87.44	3.59	3.9	3.18	
THT	(12.43)	(30.22)	(10.02)	(7.74)	(8.2)	(8.42)	(3.66)	(4.9)	(2.9)	

Source: Author's own elaboration.

Table 7 – Results for High Robustness policies.

Policy	# Neurons ↓	# Epochs ↑	Accuracy ↑				Disagreement ↓		
			Train	Validation	Holdout	Test	Train-Validation	Holdout-Test	All
Holdout	76.41 (27.5)	84.79 (25.13)	92.08 (7.92)	89.94 (7.79)	91.76 (7.66)	85.54 (9.29)	2.77 (2.66)	6.37 (6.53)	3.94 (3.51)
TTVH	75.98 (27.06)	82.64 (26.28)	91.49 (8.4)	91.1 (7.09)	91.5 (7.91)	85.42 (9.39)	2.26 (2.36)	6.26 (6.91)	3.84 (3.59)
Test	42.8 (33.91)	63.72 (34.72)	87.42 (8.8)	87.07 (7.64)	87.68 (7.98)	90.61 (7.05)	2.67 (3.25)	4.29 (3.51)	3.13 (2.41)
THT	75.5 (24.61)	78.21 (27.19)	91.22 (8.36)	89.05 (7.89)	90.9 (8.07)	89.65 (7.34)	2.83 (2.67)	3.52 (3.26)	2.67 (2.0)
TTVHT	76.5 (24.32)	77.75 (27.45)	91.05 (8.5)	90.47 (7.21)	91.05 (8.0)	88.88 (7.78)	2.52 (2.45)	3.42 (3.53)	2.48 (2.11)

Source: Author's own elaboration.

we can see in Table 8 that although we had some Test metric decrease, the models largely dropped the number of neurons. This indicates that the models with a high number of neurons are probably in a diminishing returns regime.

Table 8 – Results for Low Complexity policies.

Policy	# Neurons ↓	# Epochs ↑	Accuracy ↑				Disagreement ↓		
			Train	Validation	Holdout	Test	Train-Validation	Holdout-Test	All
TTVH	75.98 (27.06)	82.64 (26.28)	91.49 (8.4)	91.1 (7.09)	91.5 (7.91)	85.42 (9.39)	2.26 (2.36)	6.26 (6.91)	3.84 (3.59)
TTVHN	10.27 (9.27)	78.08 (28.3)	87.59 (7.94)	88.36 (7.13)	87.83 (7.66)	82.9 (9.96)	2.14 (1.93)	5.54 (7.19)	3.54 (3.74)
THT	75.5 (24.61)	78.21 (27.19)	91.22 (8.36)	89.05 (7.89)	90.9 (8.07)	89.65 (7.34)	2.83 (2.67)	3.52 (3.26)	2.67 (2.0)
THTN	8.75 (7.24)	78.35 (27.0)	86.78 (7.93)	85.97 (8.22)	86.7 (7.8)	86.9 (7.94)	2.37 (2.65)	2.75 (2.11)	2.2 (1.6)
TTVHT	76.5 (24.32)	77.75 (27.45)	91.05 (8.5)	90.47 (7.21)	91.05 (8.0)	88.88 (7.78)	2.52 (2.45)	3.42 (3.53)	2.48 (2.11)
TTVHTN	11.37 (9.44)	77.48 (27.79)	87.54 (7.92)	88.14 (7.13)	87.76 (7.61)	86.44 (8.35)	2.09 (1.93)	3.15 (3.58)	2.3 (1.93)

Source: Author's own elaboration.

3 CONCLUSIONS AND FUTURE WORK

3.1 CONCLUSION

In this thesis, we were able to analyze and propose optimality conditions of theoretical optimal models and use that to perform a multi-criteria model selection in an immense pool of heterogeneous MLP architectures (FARIAS; LUDERMIR; BASTOS-FILHO, 2022b), therefore automatically defining the architecture. The MLP architectures were trained in parallel (FARKAS; KERTÉSZ; LOVAS, 2020) using a splitting algorithm (FARIAS; LUDERMIR; BASTOS-FILHO, 2020b) that better explores the feature space among the different sets of the data.

This is useful because the training time was reduced, allowing to perform a grid-search in the architectural hyper-parameters of several MLPs in a reasonable time that could be prohibitive depending on the size of the dataset and the number of models we would like to assess. With that, we can reduce the time for the laborious task of model selection that the user would perform with a manual search. This also allows non-expert users to easily create good NN models in an automatic way.

We have proposed the model agnostic SBSS procedure that can increase the performance of classifiers by using input and output distribution information of the data.

Furthermore, we present a strategy to embarrassingly parallelize the training of heterogeneous MLP architectures in CPUs or GPUs. Improving the training time by several order of magnitudes, when compared to the sequential approach. The parallelization was possible due to a modification that we have proposed in the matrix multiplication operator that explores the principle of locality. This modification can also be used in other scenarios, such as in Sparse Neural Networks.

We also surprisingly found that fixing MLP parameters using a 10-fold cross-validation, as is usually done, might limit the learning capacity of the models if we are able to train several models in an acceptable time and be careful when selecting them to avoid over-fitting due to over-searching issues.

We have defined four optimality conditions that are expected for a theoretical optimal model to have. To analyze the influence of each one, we used different selection policies to emphasize the contrast between using or not a specific optimality condition during the selection procedure. The multi-criteria model selection procedure can mitigate over-searching problems since it will dilute the noise fitting usually responsible for the selection of over-fitted models. We have also had the opportunity to investigate the behavior of a very large pool of complex models to examine what we should look from the model and training procedure perspective to select good models. We are usually fooling ourselves when using the best model in a given set to be the gold standard, since real-world data usually contains inevitable noise and the model that maximizes the set performance is probably counting the noise as correct predictions.

The over-search problem and knowing more about the function that we want to optimize (architectural hyper-parameter search space) seem not to be the root problem of selecting bad models, but selecting a model based on a single criterion seems to be. Because real-world data is noisy, selecting the best model based solely on a specific set of the data is a sub-optimal procedure, since it is probably selecting a model that is also wrongly maximizing the noise term ϵ of that specific set.

3.1.1 Future Works

Throughout the analysis, we have realized that models with a very simple architecture sometimes had a very good performance. Which informs us that smaller models should be sufficient. The problem seems to be related to the initialization process. In models with a small number of neurons, we cannot afford small mistakes in the parameter initialization. If we compare to models with high complexity, we might have a few neurons that started in a very promising region and the overall performance of the model is very correlated with a few good neurons inside the whole network. This also corroborates with the Lottery Ticket Hypothesis and why pruning works. In order to investigate it, we plan to use our ParallelMLPs framework to create thousands of small models and look into the relationship between good models and their parameter initialization. We expect that very few models will have outstanding performance just because they started in a very promising region. Smaller models are also a way to increase the model robustness since it decreases the variance of the model and increase the bias, if it's not at the double-descent regime (BELKIN et al., 2019).

The ParallelMLPs framework can also be extended to perform feature selection if one uses a boolean mask at the beginning of the process. Another possibility is to use several MLP heads to train on top of an encoder simultaneously flowing the gradients of different architectures into the encoder to study how it would affect the learning dynamics of the encoder model. For Sparse Neural Networks, ParallelMLPs strategies might also be useful to not waste time with mask multiplications and increase the efficiency of the training algorithms. It allowed us to observe intriguing behaviors during the model selection procedure. We believe that using an immense pool of candidates opens a research avenue to study the distribution of machine learning models at a large scale and better understand what properties we should consider when selecting robust and accurate models.

We also plan to use ParallelMLPs to automatically find not only the number of neurons for a MLP, but we would like to create deeper networks, stacking layer-by-layer. Appending them while the model performance increases. Each layer can be a ParallelMLP training to choose the best number of neurons for that specific depth. Likewise, we would like to experiment with ensembles instead of selecting a single MLP and see if we can have better performances.

It is also worthwhile to extend the ParallelMLPs to work with Recurrent Neural Networks (YU et al., 2019) or to use a similar idea to speed up the attention mechanisms (NIU; ZHONG; YU, 2021) present in Transformers (TAY et al., 2020) architectures.

The multi-criteria model selection can also be further explored by using different normalization techniques or even other MCDM algorithms and the inclusion/exclusion of other variables that usually impact the learning dynamics of MLP models. A comparison with AutoML techniques using single-criterion and multi-criteria model selection seems to be interesting.

In order to collect more evidence on our optimality conditions and the multi-criteria against the single-criterion model selection policy to select robust models and have better performance estimation – with smaller fluctuations among the sets, we would like to partition a big dataset into Train, Validation, Holdout, and several disjoint Test subsets with a considerable amount of data to simulate the application of the model in different real-world scenarios. We expect a higher variance in the performance of the Test subsets – indicating a lack of robustness and poor performance estimation – for the single-criterion model selection policy. If a multi-criteria model selection is used, we expect smaller performance fluctuations for the Test subsets – indicating robust models and probably a better performance estimation when exposing the model to real unseen data in production. Analyzing the influence of adversarial attacks in single-criterion selected models and multi-criteria selected models (expected to be more robust) seems to be an interesting research direction.

A compelling extension of the SBSS procedure is to support regression problems given the promising results obtained for classification models.

REFERENCES

- ALI, R.; LEE, S.; CHUNG, T. C. Accurate multi-criteria decision making methodology for recommending machine learning algorithm. *Expert Systems with Applications*, Elsevier, v. 71, p. 257–278, 2017.
- ARULDOSS, M.; LAKSHMI, T. M.; VENKATESAN, V. P. A survey on multi criteria decision making methods and its applications. *American Journal of Information Systems*, Citeseer, v. 1, n. 1, p. 31–43, 2013.
- ASH, T. Dynamic node creation in backpropagation networks. In: *International 1989 Joint Conference on Neural Networks*. [S.l.: s.n.], 1989. p. 623 vol.2–.
- BEJANI, M. M.; GHATEE, M. A systematic review on overfitting control in shallow and deep neural networks. *Artificial Intelligence Review*, Springer, v. 54, n. 8, p. 6391–6438, 2021.
- BELKIN, M.; HSU, D.; MA, S.; MANDAL, S. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, v. 116, n. 32, p. 15849–15854, 2019. Available at: <<https://www.pnas.org/doi/abs/10.1073/pnas.1903070116>>.
- BEN-NUN, T.; HOEFLER, T. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 52, n. 4, p. 1–43, 2019.
- BLALOCK, D.; ORTIZ, J. J. G.; FRANKLE, J.; GUTTAG, J. What is the state of neural network pruning? *Proceedings of machine learning and systems*, v. 2, p. 129–146, 2020.
- BREIMAN, L. Random forests. *Machine learning*, Springer, v. 45, n. 1, p. 5–32, 2001.
- CORTES, C.; VAPNIK, V. Support vector machine. *Machine learning*, v. 20, n. 3, p. 273–297, 1995.
- DING, S.; LI, H.; SU, C.; YU, J.; JIN, F. Evolutionary artificial neural networks: a review. *Artificial Intelligence Review*, Springer, v. 39, n. 3, p. 251–260, 2013.
- DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Available at: <<http://archive.ics.uci.edu/ml>>.
- FAHLMAN, S.; LEBIERE, C. The cascade-correlation learning architecture. *Advances in neural information processing systems*, v. 2, 1989.
- FARIAS, F.; LUDERMIR, T.; BASTOS-FILHO, C. Clustering for data-driven unraveling artificial neural networks. In: SBC. *Anais do XVII Encontro Nacional de Inteligência Artificial e Computacional*. [S.l.], 2020. p. 567–578.
- FARIAS, F.; LUDERMIR, T.; BASTOS-FILHO, C. Similarity based stratified splitting: an approach to train better classifiers. *arXiv preprint arXiv:2010.06099*, 2020.
- FARIAS, F. C.; LUDERMIR, T. B.; BASTOS-FILHO, C. J. A. Embarrassingly parallel independent training of multi-layer perceptrons with heterogeneous architectures. *arXiv preprint arXiv:2206.08369*, 2022.

- FARIAS, F. C.; LUDERMIR, T. B.; BASTOS-FILHO, C. J. A. *Have we been Naive to Select Machine Learning Models? Noisy Data are here to Stay!* 2022.
- FARIAS, F. C.; LUDERMIR, T. B.; ECOMP, C. J. A. B.-F.; OLIVEIRA, F. Rosendo da S. Analyzing the impact of data representations in classification problems using clustering. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2019. p. 1–6.
- FARKAS, A.; KERTÉSZ, G.; LOVAS, R. Parallel and distributed training of deep neural networks: A brief overview. In: IEEE. *2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES)*. [S.l.], 2020. p. 165–170.
- FIX, E.; HODGES, J. L. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, JSTOR, v. 57, n. 3, p. 238–247, 1989.
- FRANKLE, J.; CARBIN, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.
- FRIEDMAN, J. H. Stochastic gradient boosting. *Computational statistics & data analysis*, Elsevier, v. 38, n. 4, p. 367–378, 2002.
- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016.
- HE, X.; ZHAO, K.; CHU, X. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, Elsevier, v. 212, p. 106622, 2021.
- HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks*, Elsevier, v. 2, n. 5, p. 359–366, 1989.
- KLEINBAUM, D. G.; DIETZ, K.; GAIL, M.; KLEIN, M.; KLEIN, M. *Logistic regression*. [S.l.]: Springer, 2002.
- LESHNO, M.; LIN, V. Y.; PINKUS, A.; SCHOCKEN, S. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, Elsevier, v. 6, n. 6, p. 861–867, 1993.
- LIVNI, R.; SHALEV-SHWARTZ, S.; SHAMIR, O. On the computational efficiency of training neural networks. *Advances in neural information processing systems*, v. 27, 2014.
- MALACH, E.; YEHUDAI, G.; SHALEV-SCHWARTZ, S.; SHAMIR, O. Proving the lottery ticket hypothesis: Pruning is all you need. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2020. p. 6682–6691.
- NIELSEN, M. A. *Neural networks and deep learning*. [S.l.]: Determination press San Francisco, CA, USA, 2015.
- NIU, Z.; ZHONG, G.; YU, H. A review on the attention mechanism of deep learning. *Neurocomputing*, Elsevier, v. 452, p. 48–62, 2021.
- PINKUS, A. Approximation theory of the mlp model in neural networks. *Acta numerica*, Cambridge University Press, v. 8, p. 143–195, 1999.
- QUINLAN, J. R. Simplifying decision trees. *International journal of man-machine studies*, Elsevier, v. 27, n. 3, p. 221–234, 1987.

- RAMANUJAN, V.; WORTSMAN, M.; KEMBHAVI, A.; FARHADI, A.; RASTEGARI, M. What's hidden in a randomly weighted neural network? In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2020. p. 11893–11902.
- REN, P.; XIAO, Y.; CHANG, X.; HUANG, P.-Y.; LI, Z.; CHEN, X.; WANG, X. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 54, n. 4, p. 1–34, 2021.
- SCHAFFER, J. D.; WHITLEY, D.; ESHELMAN, L. J. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In: IEEE. *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*. [S.l.], 1992. p. 1–37.
- SHARMA, S. K.; CHANDRA, P. Constructive neural networks: A review. *International journal of engineering science and technology*, v. 2, n. 12, p. 7847–7855, 2010.
- SONTAG, E. Feedback stabilization using two-hidden-layer nets. *IEEE Transactions on Neural Networks*, v. 3, n. 6, p. 981–990, 1992.
- STANLEY, K. O.; MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. *Evolutionary computation*, MIT Press, v. 10, n. 2, p. 99–127, 2002.
- SUBIRATS, J. L.; FRANCO, L.; JEREZ, J. M. C-mantec: A novel constructive neural network algorithm incorporating competition between neurons. *Neural Networks*, Elsevier, v. 26, p. 130–140, 2012.
- TAY, Y.; DEHGHANI, M.; BAHRI, D.; METZLER, D. Efficient transformers: A survey. *ACM Computing Surveys (CSUR)*, ACM New York, NY, 2020.
- TRANTAPHYLLOU, E. Multi-criteria decision making methods. In: *Multi-criteria decision making methods: A comparative study*. [S.l.]: Springer, 2000. p. 5–21.
- TZENG, G.-H.; HUANG, J.-J. *Multiple attribute decision making: methods and applications*. [S.l.]: CRC press, 2011.
- VAZQUEZL, M. Y. L.; PEÑAFIEL, L. A. B.; MUÑOZ, S. X. S.; MARTINEZ, M. A. Q. A framework for selecting machine learning models using topsis. In: SPRINGER. *International Conference on Applied Human Factors and Ergonomics*. [S.l.], 2020. p. 119–126.
- WEISBERG, S. *Applied linear regression*. [S.l.]: John Wiley & Sons, 2005.
- YING, X. An overview of overfitting and its solutions. In: IOP PUBLISHING. *Journal of Physics: Conference Series*. [S.l.], 2019. v. 1168, n. 2, p. 022022.
- YU, Y.; SI, X.; HU, C.; ZHANG, J. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , v. 31, n. 7, p. 1235–1270, 2019.
- ZANCHETTIN, C.; LUDERMIR, T. B.; ALMEIDA, L. M. Hybrid training method for mlp: optimization of architecture and training. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, IEEE, v. 41, n. 4, p. 1097–1109, 2011.
- ZOPH, B.; LE, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

APPENDIX A – SURVEY ON NETWORK SCIENCE AND MACHINE LEARNING

Survey on Network Science and Machine Learning

Farias, F. C.^{a,b}, Ludermir, T. B.^a, Bastos-Filho, C. J. A.^c

^a*Federal University of Pernambuco*

^b*Federal Institute of Pernambuco*

^c*University of Pernambuco*

Abstract

Machine Learning, Artificial Neural Networks, and Deep Learning have been presenting interesting approaches for solving complex tasks using information extracted from data. On the other hand, the application of Network Science tools has increased in many areas and can analyze data in Complex Networks representations of systems. Some algorithms in Machine Learning present internal structures that resemble a network/graph. Besides, both areas can solve complex problems. Therefore, they may share insights that can benefit the performance of each other. The representation of Artificial Neural Networks are composed of nodes linked by its weights and can have millions of parameters, increasing the difficulty to analyze them. As Network Science has been successfully used to analyze big Complex Networks, it is possible to gain insights from these models through the usage of the Network Science tools. At the same time, Machine Learning techniques may be used to improve Network Science techniques. In this paper, we surveyed the Machine Learning, Artificial Neural Networks, and Deep Learning fields that may be used together with Network Science tools, cooperating to solve complex problems. We selected 87 papers written from 2005 to 2017 from IEEE, Science Direct, and ACM platforms that are discussed in this survey. We also highlighted the elements shared between these areas and pointed future possibilities that we believe are worth further investigation.

Keywords: machine learning, neural networks, deep learning, network science, complex networks

Email addresses: felipefariax@gmail.com (Farias, F. C.), tbl@cin.ufpe.br (Ludermir, T. B.), carmelofilho@gmail.com (Bastos-Filho, C. J. A.)

1. Introduction

Computational Intelligence studies adaptive mechanisms to enable or facilitate intelligent behavior in complex and changing environments. A sub-field of this area, called Machine Learning, can be defined as a type of Computational Intelligence that enables computers or machines to learn without being explicitly programmed by the intensive use of data [1]. Machine Learning research field is growing fast due to the amount of available data created by the increasing number of devices and sensors, and the proliferation of the Internet of Things and Big Data technologies [2, 3]. Another relevant field that is attracting attention is the Deep Learning field, which may be seen as a sub-field of Machine Learning in which the process of information happens in multiple stages/layers, often learned hierarchically. The models of Deep Learning are commonly associated with a massive number of learnable parameters.

Over the last years, the need to analyze large complex graph-like structures, also called Complex Networks, attracted the attention of researches and many tools were developed from different disciplines like statistical physics, graph theory, control theory, among others. The National Research Council defined Network Science as the study of network representations of physical, biological, and social phenomena leading to predictive models of these phenomena [4]. The Network Science studies **Complex Networks**, which has as core principles its **nodes**, which are discrete entities with internal features/resources and its **links**, possibly weighted or unweighted, directed or undirected connections between the nodes, where the information/resources between them can be exchanged. The Network Science can analyze and describe complex systems by the structure (how the nodes are connected through links), behavior (what results from information exchange) and dynamics (the temporal attributes of nodes and links) of Complex Networks. There are many applications of Network Science in different areas of knowledge like computer science and engineering, social networks, business, biology, public health, Internet, epidemiology, physics.

In this paper, our goal is to discuss how Network Science and Machine Learning are correlated focusing primarily on how Network Science can benefit Machine Learning algorithms, especially the Neural Networks/Deep Learning because of their relative success in diverse areas. As they provide tools to solve complex problems, it may be useful to understand how they may act together.

1.1. Need for the study

In this paper, we perform a quantitative and qualitative analysis on several papers published since 2005 until 2017. The review appears to be necessary due to the increasing number of researches published over the years and the popularity that the two areas are presenting. This article is a survey that discusses the main ideas used in Machine Learning, Artificial Neural Networks, Deep Learning, and Network Science simultaneously to solve complex problems. Since the use of techniques related to Neural Networks are growing and obtaining outstanding results on many different areas and the Network Science also are calling the attention of many researchers, the union of these two areas may be appealing, once Neural Networks can be considered Complex Networks with many nodes and connections between them. We perform systematic research considering three scientific bases: Science Direct, ACM, IEEE.

1.2. Objective of the study

This paper evidences the connections between Network Science, Machine Learning/Neural Networks/Deep Learning by focusing on Complex Networks measurements, the metrics to evaluate performances, the Machine Learning/Neural Networks/Deep Learning and Network Science techniques often used and where these algorithms were applied.

1.3. Structure of the paper

We have organized the remainder of this work as follows: we present the methodology used to collect the papers used in the survey in Section 2; in Section 3 we describe the inclusion and exclusion criteria and perform a quantitative analysis, presenting charts containing information about each one of the discussed papers; in Section 4 we analyzed each one of the papers individually, grouping the papers whether Network Science was used to improve Machine Learning techniques or the opposite and commentaries on previously published books/overviews/surveys; in Section 5 we discussed the subject of the survey; We present a conclusion of the survey in Section 6; We list the references used in this survey in Section 7.

2. Methodology

We have applied a systematic methodology to search for papers on IEEE [5], ACM [6] and ScienceDirect [7] databases using *year* – newest to oldest – as

the sorting criteria. The search strings and the number of papers found for each database are presented in Table 1. Initially, the papers were evaluated by *title*, *keywords* and *abstract*. After this first analysis, we selected 149 papers to be entirely read and evaluated. During the reading, we selected 67 papers to remain in this survey, and we observed that we must have included other 20 interesting papers not found in the first round of search. Thus, we have considered 87 papers related to the subject of this survey.

Database	Search String	Number of Papers
IEEE	("machine learning" OR "neural network" OR "deep learning") AND ("complex network" OR "network science")	136
ACM	("machine learning" OR "neural network" OR "deep learning") AND ("complex network" OR "network science")	379
Science Direct	TITLE-ABSTR-KEY("machine learning" OR "neural network" OR "deep learning") AND TITLE-ABSTR-KEY("complex network" OR "network science")	40
TOTAL		555

Table 1: Search Strings and number of papers found in Databases.

3. Review of Literature

In this section, we describe the criteria used to include or exclude the papers, with exception from books and overviews, and we present a quantitative analysis.

3.1. Inclusions Criteria

- Studies that use Network Science for Machine Learning/Neural Networks/Deep Learning;
- Studies that combine Network Science techniques with Machine Learning/Neural Networks/Deep Learning;

- Researches on Classification, Clustering or Prediction associated with Network Science;
- Comparisons on Network Science against purely Machine Learning/Neural Networks/Deep Learning techniques;
- Studies published in conferences or journals.

3.2. *Exclusions Criteria*

- Papers with the term Complex Networks just mentioning a big network, without concerning about Network Science;
- Machine Learning/Neural Networks/Deep Learning studies without relations on Network Science;
- Papers about synchronization;
- Studies without experiments;
- Papers written in language different from English.

3.3. *Quantitative Analysis*

Among the 87 studies, we found three books [8, 9, 10], one survey on Complex Networks measurements [11], three overviews papers, [12] that covers community detection algorithms, [13] that relates Data Mining and Complex Networks and [14], which compares some algorithms for community detection implemented in the IGraph library. The 80 remaining studies are ordinary papers that relate Network Science and Machine Learning/Neural Networks/Deep Learning.

As one can observe in Figure 1, the number of papers published since 2005 has been growing. We were expecting this behavior since the research interest in both fields of Network Science and Machine Learning, separately, have been increasing.

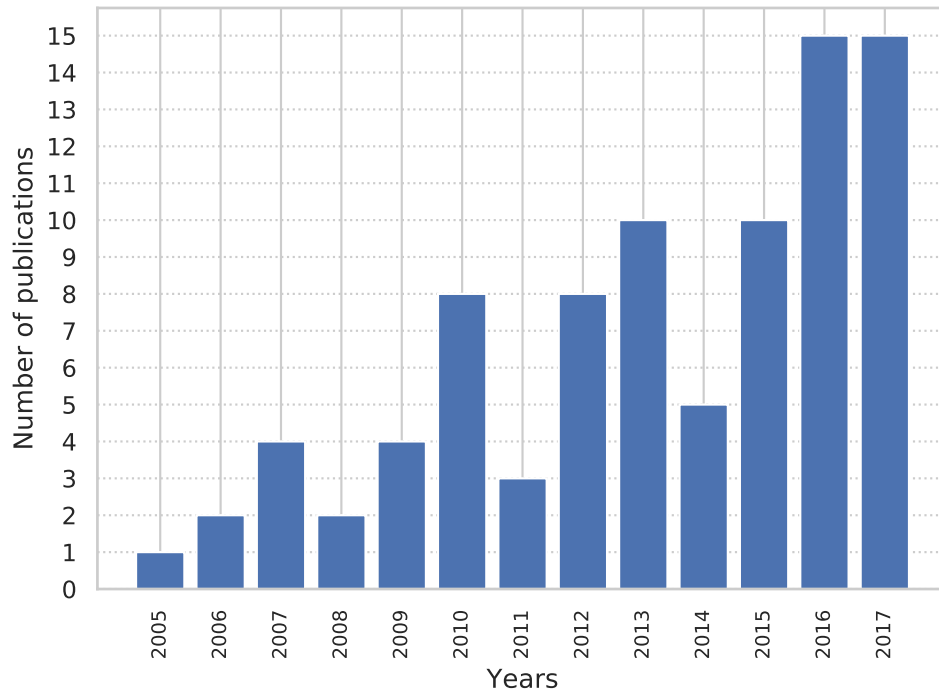


Figure 1: Histogram of papers published over the years.

Regarding the publication vehicle, Figure 2 depicts the distribution of the studies. It is interesting to observe that, even though it is a new field, there are a good amount of publications in Journals.

Machine Learning (ML) techniques may be used to solve Network Science (NS) problems (ML applied to NS end), and NS aspects and techniques also may be used to solve ML problems (NS applied to ML end). We present some information concerning the application of ML to solve an NS problem and the opposite in Figure 3. One can observe the prominence of the use of NS to solve ML problems.

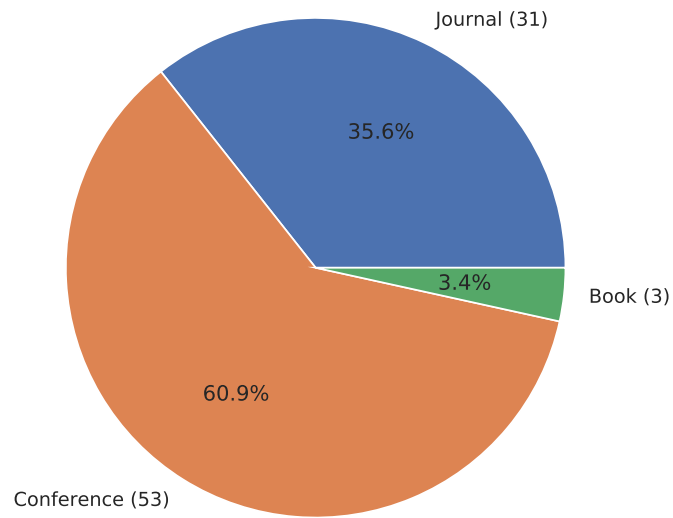


Figure 2: Distribution of studies regarding the publication vehicle.

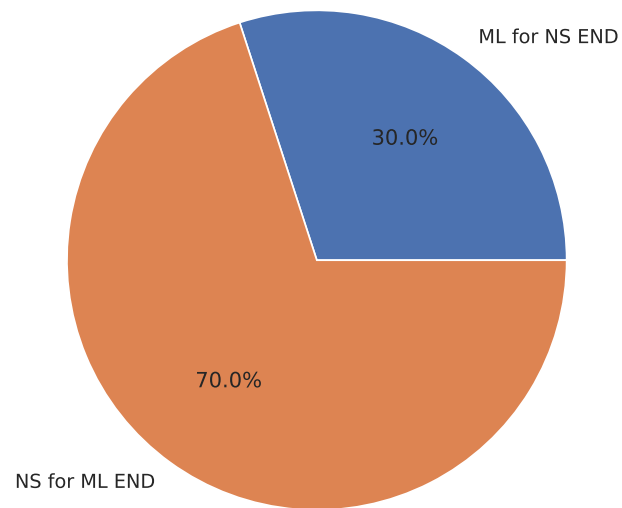


Figure 3: Application goals of the techniques.

Figure 4 details the authors that have at least two papers published on the subject of this survey. It is worth to notice that there are only five researches that have at least three papers on the subject, possibly indicating great opportunities for further researches.

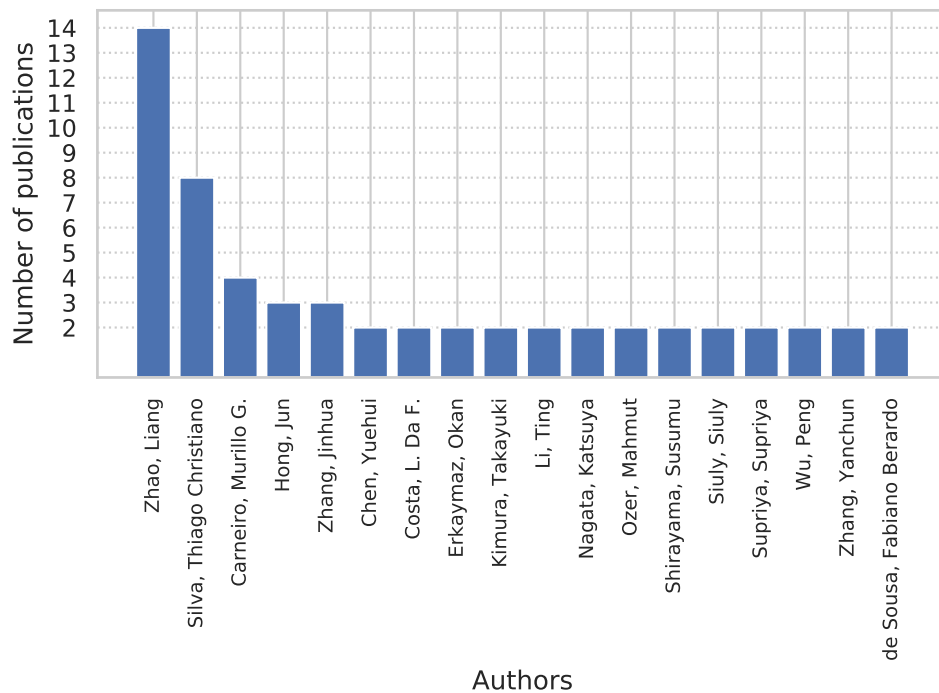


Figure 4: Number of papers per authors. Only authors with more than 1 paper published on the subject of this survey are presented.

The ML techniques, or slight modifications of the original technique, that appeared explicitly at least two times among the articles are grouped in Figure 5. One can observe that Artificial Neural Network is the most used ML technique.

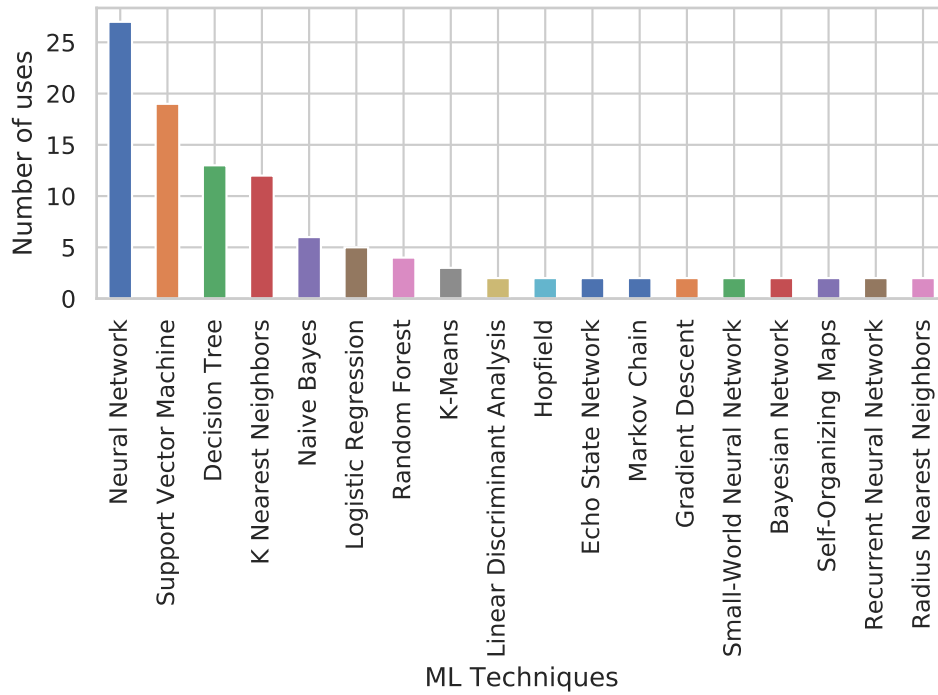


Figure 5: Number of times each Machine Learning technique appears in the references used in the survey.

The NS aspects like network topologies and techniques like Clustering Coefficients that appeared explicitly at least two times among the articles are grouped in Figure 6. Statistical measures extracted from complex network measures, e.g., Average Degree, Skewness of Degree Distribution, among others, were grouped only as Degree. The Clustering Coefficient and Degree were the most used NS measurements while the topologies highlighted were the Small-World and Scale-Free.

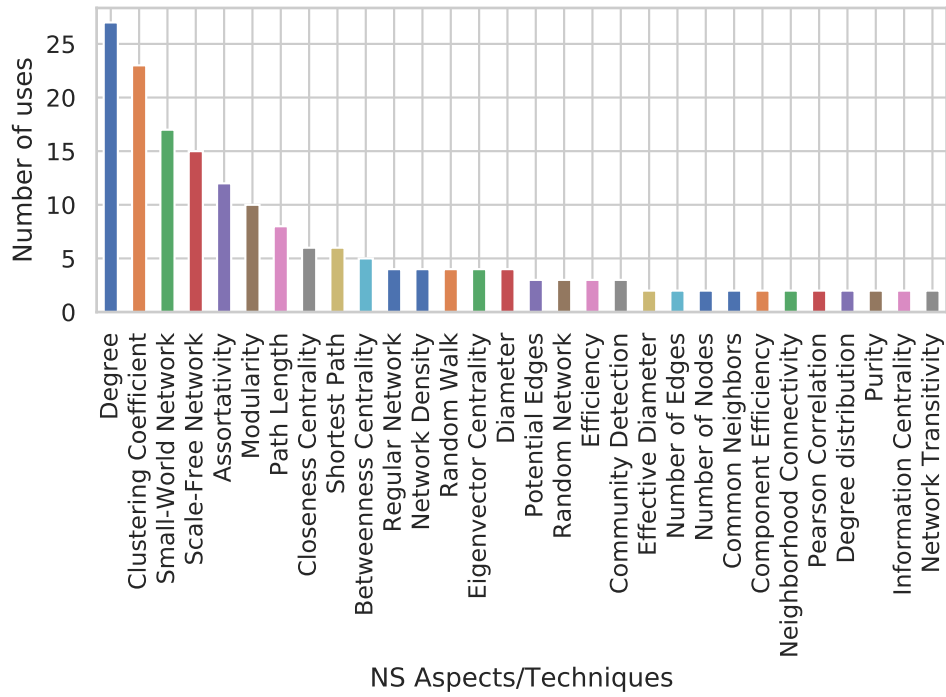


Figure 6: Number of times each Network Science concept appears in the references used in the survey.

We depict the most used evaluation metrics that appeared at least two times among the articles in Figure 7. Accuracy is the most used evaluation metric over all the works presented in this survey, followed by Precision and F1-Score.

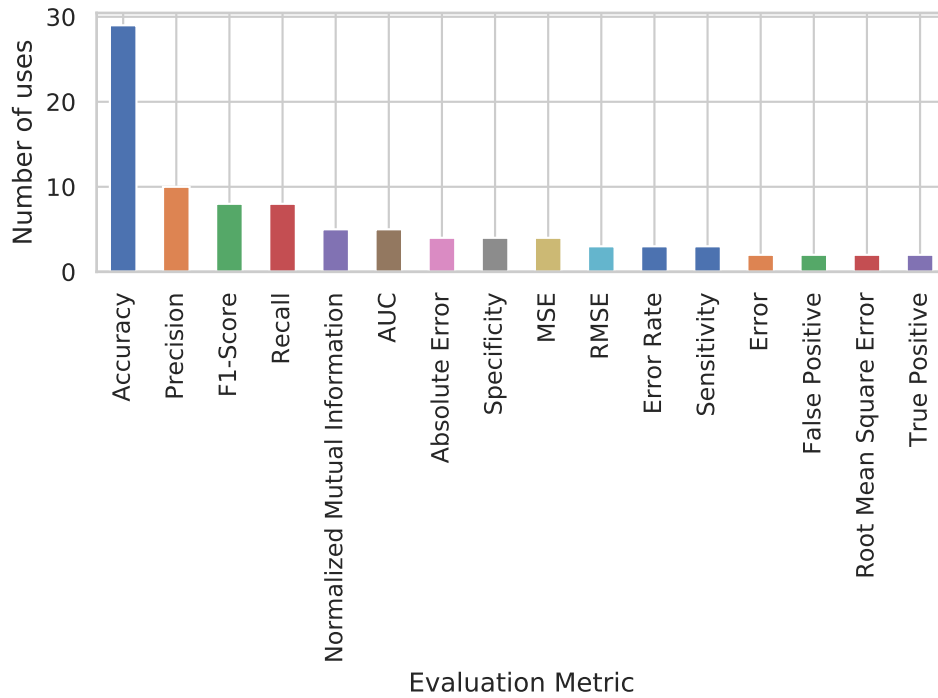


Figure 7: Number of times each Evaluation Metrics appears in the references used in the survey.

We show the most used tools/libraries that appeared among the articles in Figure 8. WEKA is an open-source collection of Machine Learning algorithms for data mining tasks [15], LIBSVM is a famous library that implements a Support Vector Machine [16]. Scikit-Learn is a Machine Learning package written in optimized C-code, and Python [17] and NetworkX is a Python package for the creation, manipulation, and study of structure, dynamics, and functions of Complex Networks [18].

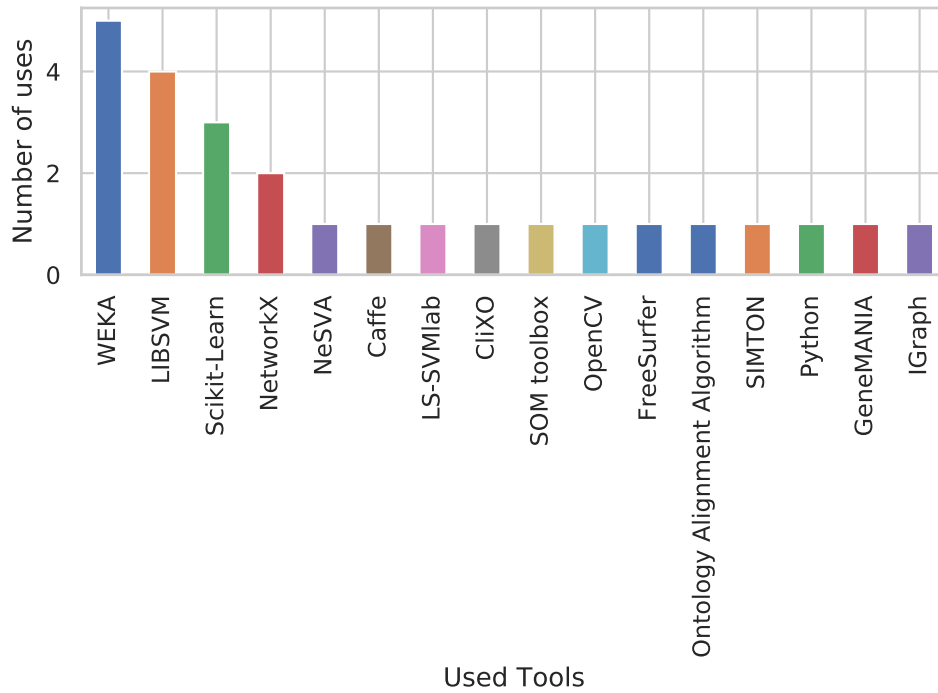


Figure 8: Number of times each computational tool appears in the references used in the survey.

4. Papers Analysis

Each one of the papers used in this survey is individually analysed in this section.

4.1. Machine Learning applied to Network Science

The papers that use Machine Learning techniques to solve problems of Network Science nature are listed in this section.

4.1.1. Communications Networks

The first paper found in this survey is [19], that used Artificial Neural Networks to control packet routing in a computer network with scale-free, randomized, and irregular topologies.

Another work on the same subject [20] used Artificial Neural Networks to dynamically route data in a scale-free network using Path Length and local information of the nodes.

A model proposed by [21] predicts Link Quality for Wireless Sensor Networks, using Matrix Factorizations and Gradient Descent to minimize a function, through the evaluation of the following coefficients: Common Neighbors, Adamic-Adar, and Resource Allocation. They also compared their approach to other techniques of unsupervised link prediction models: Common Neighbors Index, Salton Index, Preferential Attachment Index, Adamic-Adar Index, and Karz Index.

4.1.2. Complex Network Generation

We list in this section the papers that use Machine Learning models to guide the Complex Network Generation process.

The research in [22] used a Neural Network with Least Mean Square learning algorithm to generate a model of gene regulatory networks to identify and understand functions and mechanisms at the molecular level in organisms by analyzing the Average Path Length, Average Clustering Coefficient, Average Degree, Modularity and Non-Isolated Node Ratio metrics of the networks.

The work in [23] employs a Stochastic Block Model to generate random graphs to detect communities using Active Learning.

4.1.3. Link Prediction

We list in this section the papers which use approaches that try to find links between nodes of a Complex Network using Machine Learning techniques.

Intending to assess links on social networks, [24] used topological features like Number of Common Neighbors, Resource Allocation, Adamic-Adar Coefficient, Jacard's Index, Preferential Attachment, and Network Density as inputs to Machine Learning techniques such as Logistic Regression, Random Forest, Naive Bayes, and Decision Tree. In [25], the authors used Evolutionary Algorithms with non-topological features, and combinations of them, and compared their results with Decision Tree, Random Forest, Support Vector Machine and Neural Networks aiming to verify if two given nodes are linked.

In [26], the authors used Principal Component Analysis and Feature Forward Selection as a preprocessing stage to build binary classification models, using the following Machine Learning algorithms: Support Vector Machine,

Naive Bayes, K Nearest Neighbors and Decision Tree. They deployed the following measures to assess the classification: Common Neighbors, Resource Allocation, Adamic-Adar Coefficient, Jaccard's Coefficient, Preferential Attachment and Network Density.

4.1.4. *Discovering of Network Properties*

In this section, we present the papers that use Machine Learning techniques to find Network Properties/Measurements.

The relationships between the information diffusion among people using Artificial Neural Networks was studied in [27, 28]. They predict the rate and time convergence of the network using Potential Edges, Degree, Clustering Coefficient, Path Length, and Assortativity, whereas a Decision Tree reveals the statistical indicator that has a strong effect on the information diffusion.

In [29], the authors used support Vector Machine to predict communities at time $t + 1$ using evolution snapshots of a given network a time t .

An experimental study on the correlation between the complex network metrics and its correlations were performed in [30]. They evaluated the performance using the following complex network metrics: Pearson Correlation, Clustering Coefficient, Degree, Scale-Free Network, Assortativity, Betweenness Centrality, Closeness Centrality, Neighborhood Connectivity, Eigenvector Centrality, Network Density, Path Length and Diameter. To assess the correlations, they have used the algorithms K-Means, Ward, PCA, CART and Naive Bayes, using the full matrix with features and a reduced matrix with few complex network metrics in order to compare the Rand Index and the Adjusted Mutual Information.

An approach to approximate real complex network metric values using other metrics as attributes to an artificial Neural Network and a Regression Tree was proposed by [31]. Several combinations of the following metrics were used to predict the others: Eigenvector Centrality, Information Centrality, Eccentricity Centrality, Sub-graph Centrality, Walk-Based Betweenness Centrality, Betweenness Centrality, Closeness Centrality, Degree Centrality.

Deep Learning techniques, such as Search Convolutional Neural Network, node2vec, and Convolutional Neural Networks, were used in [32] to classify graphs topologies. In some experiments, the Degree, Clustering Coefficient, and Assortativity of the networks were added as attributes to the models.

A generative model based in Markov Chains were used in [33] to infer latent features of a given graph to learn Social Circles and perform a Multilabel Classification.

The identification of infection states of nodes in Complex Networks was examined in [34], that developed techniques to identify the latently infected nodes in the presence of missing infection time-and-state data. They used Naive-Bayes, Decision Tree, and Stacking as classifiers, analyzing the following Complex Networks measures: Infection Betweenness Centrality, Number of Nodes, Number of Edges, Clustering Coefficient, Degree, and Diameter.

An approach to detect communities using a distributed, sparse and non-negative symmetric Encoder-Decoder derived from a modified Autoencoder was proposed in [35].

A Framework, named Deep Aligned Autoencoder based Embedding, which learns the embedding representation of a Complex Network from its Meta Paths was proposed in [36].

An online content prediction from a group-level popularity perspective was studied in [37] which consists of two steps: (i) group users into clusters with Part Graph K Way and (ii) predict content popularity via a novel constrained tensor decomposition technique (PARAFAC), optimizing it via Gradient Descent.

In [38], the authors proposed a methodology for task-focused model selection resulting in a great increase in performance. The methodology works as follows: (i) node attributes, labels and, possibly, an edge-set are used as inputs, generating a collection of networks inferred from functions on these attributes and labels, outputting an edge-set; (ii) for each inferred network model, a classification method was built on attributes and labels for each node of interest, constraining attributes and labels available according to the network structures to produce predictions for some task (collective classification, link prediction, among others); (iii) select the appropriate network model to the tasks of interest. The deployed Machine Learning methods were: Support Vector Machine, Random Forest, K Nearest Neighbors, and Threshold Network, while the Network Science techniques mentioned were: Collective classification, Link Prediction, Intersection of attributes, Degree, Sum of Attributes (most active nodes) and Unique attribute count (most diverse nodes)

A work that estimates values of network measurements in the future based in small windows of measurements in the past was done in [39]. They have used an artificial Neural Network to predict the Shortest Path, Network Density, Degree, Clustering Coefficient, Network Transitivity, Modularity, Effective Diameter, Assortativity, and Degree distribution based on its past values.

An investigation of the distribution of distances in intrinsically high-dimensional spaces was conducted in [40] and leveraged this analysis to gain knowledge of phenomena related to the so-called dimensionality curse. They also concluded that for some real-life large-scale networks, the distribution of the incoming node degree is closely related to the derived infinite-dimensional k-occurrences function. To accomplish this study, they have used the Direct and Reverse Nearest Neighbors as Machine Learning techniques and analyzed the Degree and Hubness of the nodes and the Scale-free structure of the network.

A simulation of a chaotic trajectory tracking of a Scale-free network was performed in [41]. The trajectory tracking of dynamical complex networks were done combining Recurrent Neural Networks and slide-mode control.

Due to the lack of methods to decide whether the usage of networks abstractions are justified, a solution was proposed by [42], where they generalized standard network abstractions to multi-order graphical models. They introduced a multi-order graphical modeling framework that captures multiple variable-length pathways in networks. Their approach combines multiple higher-order Markov models into a multi-layer model consisting of De Bruijn graphs with multiple dimensions, allowing them to capture temporal correlations with multiple correlations length, simultaneously. Using PageRank, they show that correlations in sequential data can invalidate the application of graph-analytics methods. It is possible to decide when a network abstraction is justified and infer the optimal high-order graphical model that may be used to generalize network analysis techniques.

4.2. Network Science applied to Machine Learning

In this section, we present the papers that use Network Science as an aid to Machine Learning algorithms.

4.2.1. Network Science Measures in Features

The papers that extract/select features to be used as inputs to classifiers are listed in this section.

The work in [43] summarizes text using the following Network Science measures: Degree, Clustering Coefficient, Minimal Paths, Locality Index, Matching Index, Dilation, Hubs, K-cores, W-cuts and Communities as features to the following Machine Learning algorithms: Flexible-Bayes, C4.5, Support Vector Machine, Logistic Regression.

In [44], a Neural Network was used to assess the performances of a WDM network using Algebraic Connectivity, Natural Connectivity, Average Path Length, Clustering Coefficient, Diameter, and Entropy as features.

A Support Vector Machine, in combination with graph kernel, was used in [45] to classify Multiple Sclerosis Clinical Forms using Shortest Path measure.

In [46], a technique to characterize time series data using graphs, called Weight Visibility Graph, was used to model EEG data and extract the following complex network measures: Modularity and Average Weighted Degree. These two features are used as inputs for an SVM and a KNN to detect epilepsy.

A Support Vector Machine and a Linear Discriminant Analysis were used in [47] to perform epilepsy detection from EEG data, previously transformed into a Complex Network, by the analysis of the Average Weighted Degree.

A framework named Mashup was proposed in [48]. This framework analyzes the topology of multiple interaction networks from heterogeneous data sources, representing multiple networks using low dimensional vectors that can be used as inputs to a machine learning algorithm. They evaluated the Degree, Clustering Coefficient, Centrality, Betweenness Centrality, Eigenvector Centrality, Stress Centrality, Bridging Centrality, Information Centrality, Current-Flow Betweenness Centrality in a Support Vector Machine and Random Forest algorithms, and evaluated the proposal in several genetical datasets.

The paper in [49] developed a Diffusion Network and used a Neural Network to predict possible future attacks to understand the distribution and trending of historical terrorist attacks.

The classification of Electromyogram using a Neural Network with Degree, Clustering Coefficient, Transitivity, and Efficiency as inputs was presented in [50]. They also used the Weight Visibility Algorithm to transform time-series into Complex Networks.

A very well written paper [51] proposed to select the best features through the construction of a Feature Vector Graph from a dataset and analyze the Modularity measure. Preferring measures that have small distance within the same class, while choosing long distances for elements of other classes.

The characterization of the complexity of datasets in Machine Learning applications was introduced in [52]. They used the Number of Nodes, Number of Edges, Diameter, Shortest Path, Clustering Coefficient, Degree and Class Change Ratio to analyze what would be the best classifier to be used

among K Nearest Neighbors, Radius Nearest Neighbors, Decision Tree, Logistic Regression, Naive Bayes and Support Vector Machine. The goal is to investigate the adoption of measures used to evaluate complex network properties in the characterization of the complexity of datasets in machine learning applications aiming to select the best type of classifier. The Matrix data is converted into a Graph representation to apply the measures.

The sentiment analysis in text, as positive or negative opinions, taking into account the relationship between concepts regardless of the semantics of each word was conducted by [53]. The text is transformed into a Complex Network and the Degree, Clustering Coefficient, Shortest Path, Betweenness Centrality and Efficiency were extracted as features to be used as inputs to Decision Tree, RIP, and Naive Bayes.

A proposal of Network Motif Model that extracts features from relational data to present them to a Support Vector Machine was presented in [54]. They also compared the technique with Bayesian Networks.

An investigation of linear and non-linear systems control using a Small-World Neural Networks Controller was done in [55].

The classification of electroencephalogram data using Small-World Neural Networks was compared with Radial Basis Function Neural Network, standard Multi Layer Perceptron and Least Squares Support Vector Machine in [56].

An interesting work that creates features from 3D images, using Complex Network Theory, of several objects was proposed in [57]. They have used KNN and the Degrees of the nodes inside the network.

In [58], 53 different meta-path similarity measures were used to assess the proximity of entities in heterogeneous Information Networks. Using them as features to a Spectral Clustering and an SVM, they compared the results achieved in text-based datasets.

4.2.2. Analytics in Metrics

The papers that analyze Network Science metrics and use this information to support Machine Learning models are listed in this section.

An very interesting work was done in [59]. They proposed a framework that uses Network Science to perform a Supervised Data Classification considering not only the inherent attributes and class topologies but also pattern formations since the data relationships may present useful structures inside the same class data points. The authors named it a *high level* classification. They combined a *low level* classification, performed by any traditional

Machine Learning algorithm. with a *high level* classification, using complex network measures to exploit complex topological properties of data. In this work they used: Fuzzy Decision Tree, Fuzzy Support Vector Machine, Weighted K-Nearest Neighbors and Neural Networks as *low level* classifiers and the network measures: Assortativity, Clustering Coefficient, and Average Degree as *high level* classification. These three measures were chosen because they can characterize the local and global behaviors of a network. At the training phase, the *high level* classification needs a transformation of a matrix-based dataset into a graph-based dataset which is generated by applying a combination of ϵ -radius and K-Nearest Neighbors. In the classification phase, the new data point is inserted in the network of the *high level* classifier and every class evaluates the impact of the insertion and the changes that occurred in the class pattern formation. If a small or no change occurred in the previously calculated measures, then the new data point is said in compliance with that class pattern. The *low level* classifier also calculates the membership of the test instance for every class present in the problem. The output of the two classification levels can be linearly combined to perform a fuzzy classification. The combination of the outputs also considers the dataset class balance, giving more importance proportionally to the class frequency. Several artificial and real datasets were used to show that the performance of the framework may help the final classification. They suggest that the *high level* term is especially useful in complex situations of classifications.

Further research of [59] were conducted in [60, 61], where they create a network using the K-Associated Optimal Graph, that uses the Purity Measure to represent the "lass purity" of each component, and a complex network measure called Component Efficiency, performing the analysis of *low* and *high level* in an embedded way. The advantages are the reduction of the computational cost and the absence of parameters. In [60], several data clustering algorithms were compared: CHAMELEON, Expectation-Maximization, Fuzzy C-Means, Optimized K-Means, Modularity, and Particle Swarm Optimization, while in [61], the following techniques were compared with their method: Decision Tree and SVM.

A stochastic and non-linear model for competitive learning applied in data clustering, and community detection problems were proposed in [62]. They used particles biologically inspired by the competitions that occur in many social and natural systems. These particles are designed to behave like flag carriers to conquer vertices and defend its current dominated ver-

tices, emerging a competitive behavior. When a particle visits a vertex, it strengthens its domination level on that vertex and weakens the domination of its rivals in the same vertex. The expectation is that each particle dominates a community in the network. The visitation walk inside the network is guided by a combination of random (adventuring behavior) and preferential movements (defensive behavior). The particles may operate in two states: *active* (can navigate) or *exhausted* (cannot navigate). Each particle carries an energy term which is increased when it visits an already dominated vertex and decreased when it visits a vertex owned by a rival particle. If this energy drops under a threshold, the particle changes its state to *exhausted* and is teleported back to one of the vertices dominates by this particle.

The proposal in [63] also uses the idea of *low* and *high level* classification. The difference is that at the *high level* classification, Tourist Walks' dynamics (transient and cycle length), which can capture local-to-global topological properties of the data, are used instead of all the complex network measures adopted in the researches as mentioned earlier. The results were compared with: Bayesian Networks, Weighted KNN, Neural Network and Fuzzy SVM.

The work in [64] classifies patterns using Logistic Regression and K Nearest Neighbors with the following topological features as inputs: Closeness Centrality, Assortativity, and Clustering Coefficient. They also used a Social Learning Particle Swarm Optimization algorithm to (i) define the value for the coefficients for each metric, and (ii) to map the feature vectors to networks.

An approach to detect overlapping communities combining the Largest Eigenvalue of a matrix (Network Spectral Semi-Diameter), to be used in the calculation of the threshold, and epidemic spreading models was proposed in [65].

Learning in a Semi-Supervised way using Modularity measure from complex networks constructed by a K Nearest Neighbors was done by [66].

The usage of particles with Random and Deterministic movements, performed by a Random Walk, competing with each other to occupy a large number of nodes to cluster data was done by [67].

A semi-supervised data classification model based on the combination of Random and Preferential Walk of particles in a network constructed from the input dataset was proposed in [68]. This algorithm is biologically inspired in a competition-cooperation process, where the particles within the same class cooperate and the particles of different classes compete against themselves to propagate class labels to the network. An extension to detect overlapping

structures or vertices in a network was proposed in [69].

The usage of Small-World Coefficients as features was presented to Linear and Quadratic Discriminant Analysis, K Nearest Neighbors and Support Vector Machine to classify motor imagery-based Brain-Computer Interface [70].

4.2.3. *Network Science in Machine Learning Structures*

The papers that use Network Science tools to modify the internal structures of Machine Learning algorithms are presented in this section.

The influence of Regular, Small-World and Random topologies applied to the structure of a Self-Organizing Map to classify digits, analyzing the Clustering Coefficient and Mean Shortest Path of the networks and their error rate on the problem was investigated in [71]. On their experiments, the network performance is weakly controlled by its topology. They have also used Genetic Algorithms to evolve the network structure and found that the networks are more random than initial small-world topology and their connectivity distribution is more heterogeneous.

In [72], the neurons of a Hopfield network was connected using a modification proposed in this article of Watts-Strogatz rule to create a Small-World network topology. They claim that the retrieval of patterns is successful for low values of the Clustering Coefficient. During the construction of the network, many connections weights are lost, reducing the computational cost of the technique. While decreasing the Clustering Coefficient, the edges were reconnected over the network, enabling efficient retrievals.

Research conducted by [73] analyzed the Degree Distribution, Clustering Coefficient, and Sparsity and suggested that a small-world Neural Network, as diluted Hopfield Networks, may perform as well as fully-connected networks, consuming only a small fraction of connections. They calculate the overlap on the recognition of a dataset composed of 20 Chinese Characters.

An analysis of the topological features of Restricted Boltzmann Machines (RBM) and Gaussian Restricted Boltzmann Machines (GRBM) using Network Science metrics (Degree, Shortest Path and Clustering Coefficient) in [74] revealed that if the topology of an RBM or a GRBM is restricted to a scale-free, the number of parameters may be lower with virtually no performance degradation. To verify the proposal, they evaluated the techniques on several datasets and extracted the Root Mean Square Error and the Pearson Correlation Coefficient.

The usage of EEG data to classify gait, among running, walking, and standing, was assessed in [75]. They combined Common Spatial Patterns and

FastICA methods and used a Small-World Neural Network, which rewires the connections between its neurons during the training, based on a Rewiring Probability.

Self-Organizing Maps in [76] were assessed in time-series prediction with different Complex Network topologies: Small-world, Scale-Free, Regular and Random Networks. They claim that most regular topology and not to update all neurons at the same time lead to better results.

An investigation of the dynamical behavior of a Chaotic Neural Network (a modification of the Hopfield Recurrent Neural Network) where the underlying network of neurons presents community structure that resembles the ones found in biological neural networks was done by [77]. The Modularity measure was used to evaluate the quality of the communities

The structure of Small-World networks was used in [78] to reconnect the neurons of a Neural Network using the distance to give probabilities of reconnections, where near layers have more chance to reconnect than distant layers. In their simulations, the Small-World Neural Network achieved better performances than the ones that use Regular/Random connections. Also, in [79], a Small-World Neural Network, RBF Neural Network, and a Support Vector Machine, were used to classify EEG data.

The Community Discovery algorithm known as Fast GN, which tries to maximize the Modularity measure, was used in Complex Networks to determine the center and width of an RBF Neural Network, using Closeness Centrality, Node Center and Node Distance measures to predict data in [80]. They also compared their results with the combination of K-Means and Neural Network solution.

Immune Programming was used in [81] to evolve a Complex Network, based on a Scale-Free topology, and a Particle Swarm Optimization was used to fine-tuning parameters to perform a classification task. A comparison of the proposed method was performed with Flexible Neural Tree, Neural Network, and Wavelet Neural Network.

A Complex Network model [82], with Scale-Free topology, was evolved, using Genetic Algorithms, and fine-tuned, using Particle Swarm Optimization, to predict network traffic

The classification of Internet Traffic using Complex Network's Community Detection algorithm was done in [83]. They used the Modularity measure and K-Means, DBSCAN, and NFC algorithms.

A comparison of Regular, Random and Small-World connectivity in an artificial Neural Network [84], using Efficiency measures, concludes that the

Small-World topology reduces the learning error and time.

A preliminary study was conducted in [85] to evolve Artificial Neural Networks with Scale-Free topologies. They showed on one hand that connectivity has little influence on the network performance, while on the other hand, the input connectivity seems to be more effective.

An extended Echo State Network model that contains a naturally evolved state *reservoir* with Scale-Free property, Small-World effect and the hierarchically distributed structure was proposed in [86]. Another study investigating the performance of an ESN with a Small-World topology as its reservoir in time series prediction was done in [87]. They found that the ESN exhibited high performance even when reduced the number of nodes, while the standard random or fully connected ESN declined.

The topology of a power grid was simulated in [88] using Small-World networks aiming to evaluate the performance of a Recurrent Neural Network.

A Small-World Neural Network was used in [89] to control cursor movement in a three-dimensional scene using Electroencephalogram.

A Small-World Feed-forward Neural Networks was applied for diabetes diagnosis [90, 91] and had better performance than conventional Feed-forward Neural Networks.

4.2.4. Other Utilization

A Group Search Optimization (GSO) trained an artificial Neural Network with Small-World scheme to build a soft sensor model for inferring the outlet ammonia concentration in the fertilizer plant was discussed in [92].

A Multi-Agent Reinforcement Learning system combined with Complex Networks, with Scale-Free features, Simulated Annealing, Queuing Theory, and Markov Decision Processes were evolved by [93].

Given a set of entities, labeled and unlabeled, the goal of [94] was to assign class labels to unlabeled entities. In other words, the goal is to perform classification on networked entities. Modularity Kernel was proposed to exploit latent community structure of networked entities. They used a Support Vector Machine and Logistic Regression.

The dissertation in [95] proposed a learning model using an interaction network by evolving it using a Noisy Preferential Attachment. They also used an artificial Neural Network in their agents to learn classification tasks. To validate their method of evolution, they compared with Small-World, Scale-Free, and Regular Networks.

The Purity Measure was used in k-Associated Optimal Graph [96] to reduce the dimensionality of images and present them to a K-Nearest Neighbors.

A time-series prediction technique based on Network Science theory was proposed in [97]. First mapping the time series into a network representation, extracting fluctuation features, and finally constructing models with these features to predict the time series. Their approach was compared to: Grey Prediction, Exponential Smoothing, Auto-regressive Integrated Moving Average (ARIMA) and Radial Basis Function Neural Network.

The problem of insider threat event detection was addressed by [98]. They proposed a framework that uses graph mining analytics to detect anomalous events, analyzing time-series of several graph properties: Intra-community Edges, Inter-community Edges, Infomap, Degree, Connected Components, Maximum Weight.

4.3. Commentaries on previously published books/overviews/surveys

The book Machine Learning in Complex Networks [9] discuss techniques to perform Unsupervised, Semi-Supervised, and Supervised Learning using only Complex Network analysis.

The dynamics of processes executed on the network are characterized by its connectivity and its topological features which can be assessed by the extraction of network measurements. A survey including general considerations on complex network characterization, principal models, and the presentation of the main existing techniques was done in [11]. They also have analyzed correlations between some of the most traditional measurements, perturbation analysis and multivariate statistics for feature selection and network classification.

Data Mining and Complex Networks generally aim to study complex systems in order to extract information of them, an overview of these two similar areas was presented in [13]. It presents several contexts where both Complex Network Theory and Data Mining were used synergistically.

The Community Structure, also known as Clustering, is one of the essential feature extracted from graphs. They can be viewed as independent compartments of a graph that share similarities. The detection of these communities is extremely relevant in many research areas, such as Computer Science, Sociology, Biology, Physics, among others. An overview of the main Communities Detection algorithms is presented in [12].

A discussion on how Machine Learning, Data Mining, and Information Theory may support Complex Network analysis is presented in [8]. An introduction to complex networks-measures, statistical properties, and models are also provided in this study.

A comparison of several community detection algorithms implemented in the IGraph library was made in [14]. They also ranked the algorithms based on their performance extracted from several scenarios.

A fully available material discussing the Network Science area, presenting an overview of the main features, metrics, topologies and other topics regarding the field can be visited in [10].

The works presented in this session that discuss both Machine Learning, or some related area, and Network Science fields together. They are focused on (i) how to use Complex Network analysis in tasks which Machine Learning techniques were being successfully applied; (ii) how to use Machine Learning to gather information of Complex Networks. None of them targets on: (i) how the field of Network Science can be used inside existing Machine Learning techniques and vice-versa; (ii) the conduction of a Systematic Review of both areas being used side-by-side.

5. Discussions

There are many studies regarding both areas of Machine Learning and Network Science fields. Some use techniques of one field to solve problems of the other, while others use techniques of one field to support techniques of the other field aiming to facilitate the original task that one is generally used to perform by itself. Another point that deserves attention is the growth over the years of both areas, making we believe that both are prosperous fields.

The number of studies that use Network Science in Machine Learning is higher than the opposite, probably because the field of Machine Learning is older than Network Science, consequently, the number of problems evaluated by the Machine Learning community is much larger than the problems already researched in the area of Network Science, naturally leading the application of a new tool in older and well-known problems.

Some problems solved by Machine Learning field may overlap with solutions from Network Science and vice-versa, for example, Clustering and Community Detection, both areas are using its tools to solve the problem, and both with exciting performances.

As previously written, Machine Learning techniques, specially Neural Networks, have a considerable resemblance to Complex Networks studied by Network Science. The relations between Machine Learning and Network Science are shown in Figure 9. The first common feature is the NN Neurons or DT Split Nodes that we can represent them as NS Nodes, which are connected by NN Weights or DT Paths represented by weighted and non-weighted NS Links, respectively. The forward/backward phase of NNs, which activate neurons/nodes and communicate them between the Weights/Links, may be represented as the NS Network Regime. The clustering techniques, found in Unsupervised Machine Learning are also related to Community Detection belonging to Network Science techniques.

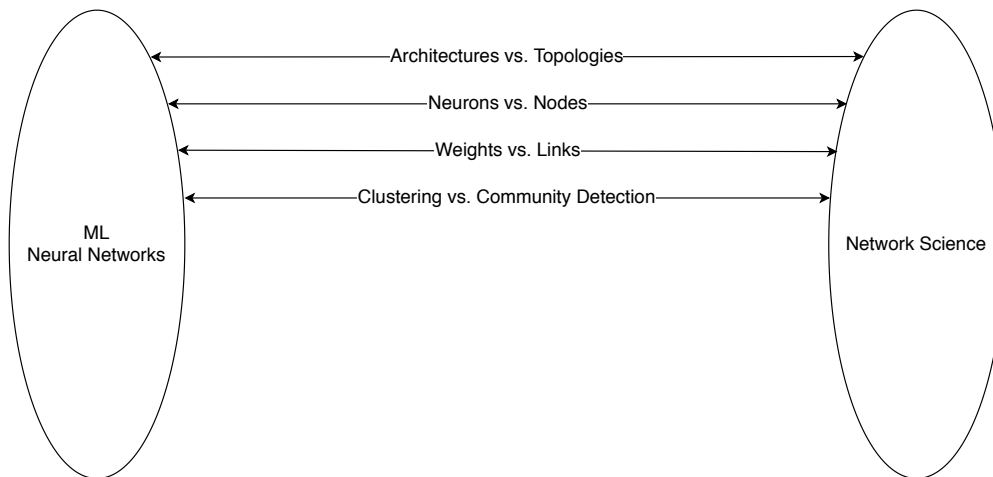


Figure 9: Current Relation between Machine Learning/Neural Networks and Network Science.

We believe that in the future there is space to increase the similarities of the two areas as depicted in Figure 10.

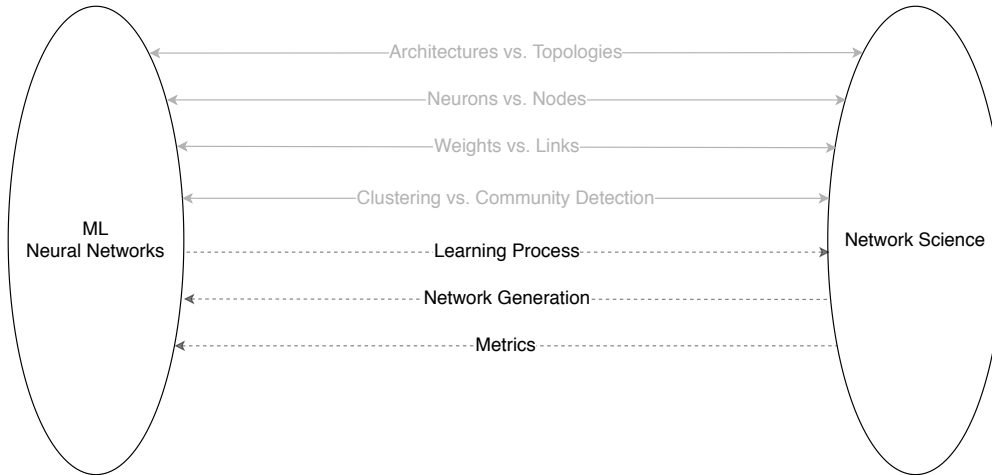


Figure 10: Possible Future Relations between Machine Learning/Neural Networks and Network Science.

One possible approach might be the Learning Process of Machine Learning techniques, such as Backpropagation, which updates parameters in order to decrease a loss function being used to generate topologies of NS optimizing some aspect of this Complex Network. The Network Generation such as Power-law and Small-world processes can be used to create complex architectures of Neural Networks. Another feature that deserves attention are the NS Metrics, which may be used during the training phase of ML algorithms, being part of the loss function to be optimized or even acting as regularizers. One old, but still present, concern in the Machine Learning community, specifically at the Artificial Neural Networks/Deep Learning field is the problem of the vanishing/exploding gradients, in which the gradient values that guide the process of learning during the back-propagation increase wildly or decrease unbridled as these gradients traverse the layers due to multiplications that the Chain Rule produces. Some strategies have been proposed to alleviate this problem. One of them that has been used successfully in some tasks is the presence of Skip-Connections or Shortcut Connections [99, 100, 101, 102]. The Artificial Neural Networks with specific Complex Network topologies that were covered in this survey generally had better results than the variants with a regular topology. A possible explanation of this behavior is that these topologies, for example, the Small-World, may benefit the network behaving like Skip-Connections, once the nodes of a specific layer can be connected to nodes of the next layers, not necessarily

being the closer layer, auxiliating the propagation of gradients through the weights of the initial layers.

6. Conclusions

In this paper, we conducted a Systematic Review on the Network Science/Complex Network and Machine Learning areas, showing how one field could help the other to better perform their tasks. Initially, 555 papers were selected, after analyzing all of them, we selected 67 papers and included 20 more papers related to this survey found in the references of these 67 papers. Thus, we used 87 pieces of research.

We have found studies that used Network Science techniques to (i) Extract Features that could be presented to classical Machine Learning techniques, (ii) analyze metrics in order to classify data and (iii) guide Machine Learning algorithms that present graph structures internally. Also, we found Machine Learning techniques applied into Network Science field in order to (iv) analyze Complex Networks structures, (v) guide the process of Network Generation, (vi) perform Link Prediction and (vii) to discover several properties of Complex Networks. We hope that this survey arouses interest in the scientific community to develop techniques that use or be inspired by exciting ideas of Network Science field to improve Machine Learning field. We also believe that good ideas of Machine Learning can be used to improve Network Science researches. We have pointed some directions which we believe that could be adopted to improve one field by using mechanisms and tools of the other.

7. References

- [1] R. C. Eberhart, Y. Shi, Computational Intelligence: Concepts to Implementations, Elsevier Science, 2011.
URL <https://books.google.com.br/books?id=Vdk1hiVdZMC>
- [2] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): A vision, architectural elements, and future directions, Future generation computer systems 29 (7) (2013) 1645–1660.
- [3] S. John Walker, Big data: A revolution that will transform how we live, work, and think (2014).

-
- [4] N. Council, D. Sciences, B. Technology, C. Applications, Network Science, National Academic Press, 2006.
 - [5] Ieee xplora digital library, <http://ieeexplore.ieee.org>, accessed: 2017-05-20.
 - [6] Acn digital library, <http://dl.acm.org>, accessed: 2017-05-20.
 - [7] Science direct, <http://sciencedirect.com>, accessed: 2017-05-20.
 - [8] M. Dehmer, S. C. Basak, Statistical and Machine Learning Approaches for Network Analysis, 1st Edition, Wiley Publishing, 2012. doi:10.1002/9781118346990.
 - [9] T. Christiano Silva, L. Zhao, Machine Learning in Complex Networks, Springer International Publishing, 2016. doi:10.1007/978-3-319-17290-3.
 - [10] A. L. Barabási, M. Pósfai, Network Science, Cambridge University Press, 2016.
URL <http://barabasi.com/networksciencebook/>
 - [11] L. da F. Costa, F. A. Rodrigues, G. Travieso, P. R. V. Boas, Characterization of complex networks: A survey of measurements, Advances in Physics 56 (1) (2007) 167–242. doi:10.1080/00018730601170527.
URL <https://doi.org/10.1080/00018730601170527>
 - [12] S. Fortunato, Community detection in graphs, Physics Reports 486 (3–5) (2010) 75–174. arXiv:0906.0612, doi:10.1016/j.physrep.2009.11.002.
URL <http://dx.doi.org/10.1016/j.physrep.2009.11.002>
 - [13] M. Zanin, D. Papo, P. A. Sousa, E. Menasalvas, A. Nicchi, E. Kubik, S. Boccaletti, Combining complex networks and data mining: Why and how, Physics Reports 635 (2016) 1–44. arXiv:1604.08816, doi:10.1016/j.physrep.2016.04.005.
URL <http://dx.doi.org/10.1016/j.physrep.2016.04.005>
 - [14] F. B. de Sousa, L. Zhao, Evaluating and Comparing the IGraph Community Detection Algorithms, in: 2014 Brazilian Conference on Intelligent Systems, IEEE, 2014, pp. 408–413. doi:10.1109/BRACIS.2014.79.
URL <http://ieeexplore.ieee.org/document/6984865/>

-
- [15] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, The weka data mining software: An update, *SIGKDD Explor. Newsl.* 11 (1) (2009) 10–18. doi:10.1145/1656274.1656278.
URL <http://doi.acm.org/10.1145/1656274.1656278>
 - [16] C.-C. Chang, C.-J. Lin, Libsvm: A library for support vector machines, *ACM Trans. Intell. Syst. Technol.* 2 (3) (2011) 27:1–27:27. doi:10.1145/1961189.1961199.
URL <http://doi.acm.org/10.1145/1961189.1961199>
 - [17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, *J. Mach. Learn. Res.* 12 (2011) 2825–2830.
URL <http://dl.acm.org/citation.cfm?id=1953048.2078195>
 - [18] A. Hagberg, P. Swart, D. S Chult, Exploring network structure, dynamics, and function using networkx, 2008.
 - [19] T. Kimura, H. Nakajima, T. Ikeguchi, A packet routing method for complex networks by a stochastic neural network, *Physica A: Statistical Mechanics and its Applications* 376 (1-2) (2007) 658–672. doi:10.1016/j.physa.2006.10.061.
 - [20] Y. Naganuma, A. Igarashi, A packet routing strategy using neural networks on scale-free networks, *Physica A: Statistical Mechanics and its Applications* 389 (3) (2010) 623–628. doi:10.1016/j.physa.2009.09.048.
URL <http://dx.doi.org/10.1016/j.physa.2009.09.048>
 - [21] Y. Zhao, S. Li, J. Hou, Link Quality Prediction via a Neighborhood-Based Nonnegative Matrix Factorization Model for Wireless Sensor Networks, *International Journal of Distributed Sensor Networks* 2015 (2015) 16:16—16:16. doi:10.1155/2015/828493.
URL <https://doi.org/10.1155/2015/828493>
 - [22] L. Liu, M. Liu, M. Ma, Construction of linear dynamic gene regulatory network based on feedforward neural network, 2014 10th International Conference on Natural Computation, ICNC 2014 (2014) 99–107doi:10.1109/ICNC.2014.6975817.

-
- [23] A. Gadde, E. E. Gad, S. Avestimehr, A. Ortega, Active learning for community detection in stochastic block models, *IEEE International Symposium on Information Theory - Proceedings 2016-Augus* (2016) 1889–1893. arXiv:1605.02372, doi:10.1109/ISIT.2016.7541627.
 - [24] M. Abufouda, K. A. Zweig, Are We Really Friends? Link Assessment in Social Networks Using Multiple Associated Interaction Networks, in: *Proceedings of the 24th International Conference on World Wide Web - WWW '15 Companion*, ACM Press, New York, New York, USA, 2015, pp. 771–776. doi:10.1145/2740908.2742468. URL <http://dl.acm.org/citation.cfm?doid=2740908.2742468>
 - [25] D. Jesenko, M. Mernik, B. Žalik, D. Mongus, Two-level evolutionary algorithm for discovering relations between nodes' features in a complex network, *Applied Soft Computing Journal* 56 (2017) 82–93. doi:10.1016/j.asoc.2017.02.031.
 - [26] A. Pecli, B. Giovanini, C. Pacheco, C. Moreira, F. Ferreira, F. Tosta, J. Tesolin, M. Dias, S. Filho, M. Cavalcanti, R. Goldschmidt, Dimensionality reduction for supervised learning in link prediction problems, *ICEIS 2015 - 17th International Conference on Enterprise Information Systems, Proceedings 1* (March 2017). doi:10.5220/0005371802950302.
 - [27] K. Nagata, S. Shirayama, Analysis method of influence of potential edge on information diffusion, *Procedia Computer Science* 4 (2011) 241–250. doi:10.1016/j.procs.2011.04.026.
 - [28] K. Nagata, S. Shirayama, Method of analyzing the influence of network structure on information diffusion, *Physica A: Statistical Mechanics and its Applications* 391 (14) (2012) 3783–3791. doi:10.1016/j.physa.2012.02.031.
 - [29] B. Ngonmang, E. Viennet, Toward Community Dynamic through Interactions Prediction in Complex Networks, in: *Sitis, SITIS '13*, IEEE Computer Society, Washington, DC, USA, 2013, pp. 462–469. doi:10.1109/SITIS.2013.81. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6727230>
 - [30] A. Garcia-Robledo, A. Diaz-Perez, G. Morales-Luna, Correlation analysis of complex network metrics on the topology of the Internet, in: *2013 10th International Conference and Expo on Emerging*

- Technologies for a Smarter World (CEWIT), IEEE, 2013, pp. 1–6.
doi:10.1109/CEWIT.2013.6713749.
URL <http://ieeexplore.ieee.org/document/6713749/>
- [31] F. Grando, L. C. Lamb, Estimating complex networks centrality via neural networks and machine learning, Proceedings of the International Joint Conference on Neural Networks 2015-Septe. doi:10.1109/IJCNN.2015.7280334.
- [32] S. Pal, Y. Dong, B. Thapa, N. V. Chawla, A. Swami, R. Ramanathan, Deep learning for network analysis: Problems, approaches and challenges, in: MILCOM 2016 - 2016 IEEE Military Communications Conference, 2016, pp. 588–593. doi:10.1109/MILCOM.2016.7795391.
URL <http://ieeexplore.ieee.org/document/7795391/>
- [33] C. Tsourakakis, Provably Fast Inference of Latent Features from Networks, in: Proceedings of the 24th International Conference on World Wide Web - WWW '15, 2015, pp. 1111–1121. doi:10.1145/2736277.2741128.
URL <http://dl.acm.org/citation.cfm?doid=2736277.2741128>
- [34] Y.-s. Lim, B. Ribeiro, D. Towsley, Classifying latent infection states in complex networks, Computational Social Networks 2 (1) (2015) 8. arXiv:1402.0013v1, doi:10.1186/s40649-015-0015-6.
URL <http://www.computationalsocialnetworks.com/content/2/1/8>
- [35] B.-J. Sun, H. Shen, J. Gao, W. Ouyang, X. Cheng, A Non-negative Symmetric Encoder-Decoder Approach for Community Detection, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management - CIKM '17, ACM Press, New York, New York, USA, 2017, pp. 597–606. doi:10.1145/3132847.3132902.
URL <http://dl.acm.org/citation.cfm?doid=3132847.3132902>
- [36] J. Zhang, C. Xia, C. Zhang, L. Cui, Y. Fu, P. S. Yu, BL-MNE: Emerging heterogeneous social network embedding through broad learning with aligned autoencoder, Proceedings - IEEE International Conference on Data Mining, ICDM 2017-Novem (2017) 605–614. doi:10.1109/ICDM.2017.70.

-
- [37] M. X. Hoang, X.-H. Dang, X. Wu, Z. Yan, A. K. Singh, GPOP: Scalable Group-level Popularity Prediction for Online Content in Social Networks, in: Proceedings of the 26th International Conference on World Wide Web, WWW '17, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 2017, pp. 725–733. arXiv:arXiv:1603.07016v1, doi:10.1145/3038912.3052626.
URL <https://doi.org/10.1145/3038912.3052626>
<http://dl.acm.org/citation.cfm?doid=3038912.3052626>
 - [38] I. Brugere, C. Kanich, T. Y. Berger-Wolf, Network Model Selection for Task-Focused Attributed Network Inference, in: 2017 IEEE International Conference on Data Mining Workshops (ICDMW), Vol. 2017-Novem, IEEE, 2017, pp. 118–125. arXiv:arXiv:1708.06303v2, doi:10.1109/ICDMW.2017.21.
URL <http://ieeexplore.ieee.org/document/8215652/>
 - [39] S. Mohamadyari, N. Attar, S. Aliakbary, On feature prediction in temporal social networks based on artificial neural network learning, in: 2017 7th International Conference on Computer and Knowledge Engineering (ICCKE), Vol. 2017-Janua, IEEE, 2017, pp. 303–307. doi:10.1109/ICCKE.2017.8167896.
URL <http://ieeexplore.ieee.org/document/8167896/>
 - [40] F. Angiulli, On the Behavior of Intrinsically High-dimensional Spaces: Distances, Direct and Reverse Nearest Neighbors, and Hubness, J. Mach. Learn. Res. 18 (1) (2017) 6209–6268.
URL <http://dl.acm.org/citation.cfm?id=3122009.3242027>
 - [41] C. J. Vega, O. J. Suarez, E. N. Sanchez, G. Chen, Trajectory tracking on complex networks via neural sliding-mode pinning control, in: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Vol. 64, IEEE, 2017, pp. 995–1000. doi:10.1109/SMC.2017.8122740.
URL <https://ieeexplore.ieee.org/document/8357436/>
<http://ieeexplore.ieee.org/document/8122740/>
 - [42] I. Scholtes, When is a Network a Network?: Multi-Order Graphical Model Selection in Pathways and Temporal Networks, in: Proceedings

- of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17, ACM, New York, NY, USA, 2017, pp. 1037–1046. arXiv:1702.05499, doi:10.1145/3097983.3098145. URL <http://doi.acm.org/10.1145/3097983.3098145>
- [43] D. S. Leite, L. H. M. Rino, Combining multiple features for automatic text summarization through machine learning, in: A. Teixeira, V. L. S. de Lima, L. C. de Oliveira, P. Quaresma (Eds.), *Computational Processing of the Portuguese Language*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 122–132.
 - [44] D. R. B. de Araujo, J. F. Martins, C. J. A. Bastos, Using Multi-Layer Perceptron and Complex Network Metrics to Estimate the Performance of Optical Networks, 2013 Sbmo/Ieee Mtt-S International Microwave & Optoelectronics Conference (Imoc).
 - [45] C. Stamile, G. Kocavar, S. Hannoun, F. Durand-Dubief, D. Sappey-Marinier, A graph based classification method for multiple sclerosis clinical forms using support vector machine, in: *Revised Selected Papers of the First International Workshop on Machine Learning Meets Medical Imaging - Volume 9487*, Springer-Verlag, Berlin, Heidelberg, 2015, pp. 57–64. doi:10.1007/978-3-319-27929-9_6.
 - [46] S. Supriya, S. Siuly, H. Wang, J. Cao, Y. Zhang, Weighted Visibility Graph With Complex Network Features in the Detection of Epilepsy, *IEEE Access* 4 (2016) 6554–6566. doi:10.1109/ACCESS.2016.2612242.
 - [47] S. Supriya, S. Siuly, Y. Zhang, Automatic epilepsy detection from EEG introducing a new edge weight method in the complex network, *Electronics Letters* 52 (17) (2016) 1430–1432. doi:10.1049/el.2016.1992. URL <http://digital-library.theiet.org/content/journals/10.1049/el.2016.1992>
 - [48] H. Cho, B. Berger, J. Peng, Compact Integration of Multi-Network Topology for Functional Analysis of Genes, *Cell Systems* 3 (6) (2016) 1–9. doi:10.1016/j.cels.2016.10.017. URL <http://dx.doi.org/10.1016/j.cels.2016.10.017>
 - [49] Z. Li, D. Sun, H. Chen, S. Y. Huang, Identifying the socio-spatial dynamics of terrorist attacks in the Middle East, in: *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*, 2016, pp. 175–180. doi:10.1109/ISI.2016.7745463.

-
- [50] P. Artameeyanant, S. Sultornsanee, K. Chamnongthai, Classification of electromyogram using weight visibility algorithm with multi-layer perceptron neural network, in: 2015 7th International Conference on Knowledge and Smart Technology (KST), 2015, pp. 190–194. doi:10.1109/KST.2015.7051485.
 - [51] G. Zhao, Y. Wu, F. Chen, J. Zhang, J. Bai, Effective feature selection using feature vector graph for classification, *Neurocomputing* 151 (P1) (2015) 376–389. doi:10.1016/j.neucom.2014.09.027.
 - [52] G. Morais, R. C. Prati, Complex Network Measures for Data Set Characterization, in: 2013 Brazilian Conference on Intelligent Systems, no. OCTOBER 2013 in BRACIS '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 12–18. doi:10.1109/BRACIS.2013.11. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6726419>
 - [53] D. R. Amancio, R. Fabbri, O. N. Oliveira Jr., M. G. V. Nunes, L. da F. Costa, Distinguishing between positive and negative opinions with complex network features, in: Proceedings of the 2010 Workshop on Graph-based Methods for Natural Language Processing, no. July in TextGraphs-5, Association for Computational Linguistics, Stroudsburg, PA, USA, 2010, pp. 83–87. URL <http://dl.acm.org/citation.cfm?id=1870490.1870503>
 - [54] C. W. Huang, C. C. Yu, C. H. Mao, H. M. Lee, Network motif model: An efficient approach for extracting features from relational data, *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics* 6 (2007) 5141–5146. doi:10.1109/ICSMC.2006.385124.
 - [55] X. Li, F. Xu, J. Zhang, S. Wang, A multilayer feed forward small-world neural network controller and its application on electrohydraulic actuation system, *Journal of Applied Mathematics* 2013. doi:10.1155/2013/872790.
 - [56] T. Li, J. Hong, J. Zhang, F. Guo, Brain-machine interface control of a manipulator using small-world neural network and shared control strategy, *Journal of Neuroscience Methods* 224 (2014) 26–38. doi:10.1016/j.jneumeth.2013.11.015. URL <http://dx.doi.org/10.1016/j.jneumeth.2013.11.015>

-
- [57] C. A. B. P. Filho, F. S. Osorio, Complex network shape descriptor for 3D objects classification, Proceedings - 2017 LARS 14th Latin American Robotics Symposium and 2017 5th SBR Brazilian Symposium on Robotics, LARS-SBR 2017 - Part of the Robotics Conference 2017 2017-Decem (2017) 1–5. doi:10.1109/SBR-LARS-R.2017.8215280.
 - [58] C. Wang, Y. Song, H. Li, Y. Sun, M. Zhang, J. Han, Distant Meta-Path Similarities for Text-Based Heterogeneous Information Networks, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management - CIKM '17, ACM Press, New York, New York, USA, 2017, pp. 1629–1638. doi:10.1145/3132847.3133029. URL <http://dl.acm.org/citation.cfm?doid=3132847.3133029>
 - [59] T. C. Silva, L. Zhao, Network-based high level data classification, IEEE Transactions on Neural Networks and Learning Systems 23 (6) (2012) 954–970. doi:10.1109/TNNLS.2012.2195027.
 - [60] M. G. Carneiro, L. Zhao, High level classification totally based on complex networks, Proceedings - 1st BRICS Countries Congress on Computational Intelligence, BRICS-CCI 2013 (2013) 507–514doi:10.1109/BRICS-CCI-CBIC.2013.90.
 - [61] M. G. Carneiro, J. L. G. Rosa, A. A. Lopes, L. Zhao, Network-based data classification : combining K -associated optimal graphs and high-level prediction, Journal of the Brazilian Computer Society 20 (14) (2014) 1–14. doi:10.1186/1678-4804-20-14.
 - [62] T. C. Silva, L. Zhao, Stochastic competitive learning in complex networks, IEEE Transactions on Neural Networks and Learning Systems 23 (3) (2012) 385–398. doi:10.1109/TNNLS.2011.2181866.
 - [63] T. C. Silva, L. Zhao, High-level pattern-based classification via tourist walks in networks, Information Sciences 294 (2015) 109–126. arXiv:1305.1679, doi:10.1016/j.ins.2014.09.048.
 - [64] M. G. Carneiro, L. Zhao, R. Cheng, Y. Jin, Network structural optimization based on swarm intelligence for highlevel classification, Proceedings of the International Joint Conference on Neural Networks 2016-Octob (2016) 3737–3744. doi:10.1109/IJCNN.2016.7727681.

-
- [65] X. Deng, Y. Wen, Y. Chen, Highly efficient epidemic spreading model based LPA threshold community detection method, *Neurocomputing* 210 (2016) 3–12. doi:10.1016/j.neucom.2015.10.142.
 - [66] T. C. Silva, L. Zhao, Semi-supervised learning guided by the modularity measure in complex networks, *Neurocomputing* 78 (1) (2012) 30–37. doi:10.1016/j.neucom.2011.04.042.
 - [67] T. C. Silva, L. Zhao, Network-based learning through particle competition for data clustering, *Proceedings of the International Joint Conference on Neural Networks* (2011) 45–52doi:10.1109/IJCNN.2011.6033198.
 - [68] T. C. Silva, L. Zhao, Network-based stochastic semisupervised learning, *IEEE Transactions on Neural Networks and Learning Systems* 23 (3) (2012) 451–466. doi:10.1109/TNNLS.2011.2181413.
 - [69] T. C. Silva, L. Zhao, Uncovering overlapping cluster structures via stochastic competitive learning, *Information Sciences* 247 (2013) 40–61. doi:10.1016/j.ins.2013.06.024.
 - [70] L. Santamaria, C. James, Use of graph metrics to classify motor imagery based BCI, 2016 International Conference for Students on Applied Engineering, ICSAE 2016 (2017) 469–474doi:10.1109/ICSAE.2016.7810237.
 - [71] F. Jiang, H. Berry, M. Schoenauer, The impact of network topology on self-organizing maps, in: *Proceedings of the first ACM/SIGEVO ...*, GEC '09, ACM, New York, NY, USA, 2009, pp. 247–253. doi:10.1145/1543834.1543869.
URL <http://dl.acm.org/citation.cfm?id=1543869>
 - [72] K. Omachi, T. Isokawa, N. Kamiura, N. Matsui, H. Nishimura, Retrieval performance of complex-valued associative memory with complex network structure, *International Conference on Emerging Trends in Engineering and Technology, ICETET* 1 (2) (2012) 40–43. doi:10.1109/ICETET.2012.26.
 - [73] P. Zheng, W. Tang, J. Zhang, A simple method for designing efficient small-world neural networks, *Neural Networks* 23 (2) (2010) 155–159.

- doi:10.1016/j.neunet.2009.11.005.
 URL <http://dx.doi.org/10.1016/j.neunet.2009.11.005>
- [74] D. C. Mocanu, E. Mocanu, P. H. Nguyen, M. Gibescu, A. Liotta, A topological insight into restricted Boltzmann machines, *Machine Learning* 104 (2-3) (2016) 243–270. arXiv:1604.05978, doi:10.1007/s10994-016-5570-z.
 URL <https://doi.org/10.1007/s10994-016-5570-z>
- [75] C. Zhang, J. Zhang, J. Hong, Classification of EEG Signals using multiple gait features based on Small-world Neural Network, *International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)* (2016) 61–66.
- [76] J. C. Burguillo, Using self-organizing maps with complex network topologies and coalitions for time series prediction, *Soft Computing* (2013) 695–705 doi:10.1007/s00500-013-1171-y.
 URL <http://link.springer.com/10.1007/s00500-013-1171-y>
- [77] F. B. de Sousa, L. Zhao, Investigation of complex dynamics in a recurrent neural network with network community structure and asymmetric weight matrix, *The 2013 International Joint Conference on Neural Networks (IJCNN)* (2013) 1–7 doi:10.1109/IJCNN.2013.6706846.
 URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6706846>
- [78] X. Li, X. Li, J. Zhang, Y. Zhang, M. Li, A new multilayer feedforward small-world neural network with its performances on function approximation, *Proceedings - 2011 IEEE International Conference on Computer Science and Automation Engineering, CSAE 2011* 3 (50905136) (2011) 353–357. doi:10.1109/CSAE.2011.5952696.
- [79] L. Ting, H. Jun, J. Zhang, EEG Classification Based on Small-world Neural Network for Brain-Computer Interface, *2010 Sixth International Conference on Natural Computation (ICNC 2010) (Icnc)* (2010) 252–256.
- [80] B. Wu, W. Ma, T. Zhu, J. Yang, Predicting mechanical properties of hot-rolling steel by using RBF network method based on complex network theory, *2010 Sixth International Conference on Natural Computation 4 (Icnc)* (2010) 1759–1763. doi:10.1109/ICNC.2010.5584387.
 URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5584387

-
- [81] P. Wu, Y. Chen, T. Xu, H. Tang, Evolving Complex Network for Classification Problems, 2009 International Conference on Computational Intelligence and Natural Computing 1 (2009) 287–290. doi:10.1109/CINC.2009.171.
URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5231129>
 - [82] P. Wu, Y. Chen, Q. Meng, Q. Z. Liu, Small-Time Scale Network Traffic Prediction Using Complex Network Models, Natural Computation, 2009. ICNC '09. Fifth International Conference on 3 (2009) 303–307. doi:10.1109/ICNC.2009.122.
 - [83] J. Cai, S. Z. Yu, Internet traffic identification using community detecting algorithm, in: Proceedings - 2010 2nd International Conference on Multimedia Information Networking and Security, MINES 2010, MINES '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 164–168. doi:10.1109/MINES.2010.43.
URL <http://dx.doi.org/10.1109/MINES.2010.43>
 - [84] D. Simard, L. Nadeau, H. Kröger, Fastest learning in small world neural networks, Physics Letters A 336 (1) (2004) 14. arXiv:0402076, doi:10.1016/j.physleta.2004.12.078.
URL <http://arxiv.org/abs/physics/0402076>
 - [85] M. Annunziato, I. Bertini, M. De Felice, S. Pizzuti, Evolving complex neural networks, in: R. Basili, M. T. Pazienza (Eds.), AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 194–205.
 - [86] Z. D. Z. Deng, Y. Z. Y. Zhang, Complex Systems Modeling Using Scale-Free Highly-Clustered Echo State Network, The 2006 IEEE International Joint Conference on Neural Network Proceedings (2006) 1–8doi:10.1109/IJCNN.2006.247295.
 - [87] Y. Kawai, T. Tokuno, J. Park, M. Asada, Echo in a small-world reservoir: Time-series prediction using an economical recurrent neural network, in: 2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob), Vol. 2018-Janua, IEEE, 2017, pp. 126–131. doi:10.1109/DEVLRN.2017.8329797.
URL <http://ieeexplore.ieee.org/document/8329797/>

-
- [88] K. Kimura, T. Kimura, Neural networks approach for wind-solar energy system with complex networks, Proceedings of the International Conference on Power Electronics and Drive Systems (2013) 1–5doi:10.1109/PEDS.2013.6526978.
 - [89] T. Li, J. Hong, J. Zhang, Electroencephalographic (EEG) control of cursor movement in three-dimensional scene based on small-world neural network, Proceedings - 2010 IEEE International Conference on Intelligent Computing and Intelligent Systems, ICIS 2010 3 (50905136) (2010) 587–591. doi:10.1109/ICICISYS.2010.5658416.
 - [90] O. ErKaymaz, M. Ozer, Impact of small-world network topology on the conventional artificial neural network for the diagnosis of diabetes, Chaos, Solitons and Fractals 83 (2016) 178–185. doi:10.1016/j.chaos.2015.11.029.
 - [91] O. ErKaymaz, M. Ozer, M. Perc, Performance of small-world feedforward neural networks for the diagnosis of diabetes, Applied Mathematics and Computation 311 (2017) 22–28. doi:10.1016/j.amc.2017.05.010.
 - [92] X. Yan, W. Yang, H. Shi, A group search optimization based on improved small world and its application on neural network training in ammonia synthesis, Neurocomputing 97 (2012) 94–107. doi:10.1016/j.neucom.2012.06.001.
URL <http://dx.doi.org/10.1016/j.neucom.2012.06.001>
 - [93] D. Dahlem, W. Harrison, Globally optimal multi-agent reinforcement learning parameters in distributed task assignment, in: Proceedings - 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2009, Vol. 2 of WI-IAT '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 28–35. doi:10.1109/WI-IAT.2009.122.
URL <http://dx.doi.org/10.1109/WI-IAT.2009.122>
 - [94] D. Zhang, R. Mao, Classifying networked entities with modularity kernels, in: Proceeding of the 17th ACM conference on Information and knowledge mining - CIKM '08, CIKM '08, ACM, New York, NY, USA, 2008, pp. 113–122. doi:10.1145/1458082.1458100.
URL <http://eprints.bbk.ac.uk/7081/>

-
- [95] S. Swarup, Artificial language evolution on a dynamical interaction network, Ph.D. thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, aAI3290395 (2007).
 - [96] T. H. Cupertino, M. G. Carneiro, L. Zhao, Dimensionality reduction with the k-associated optimal graph applied to image classification, IST 2013 - 2013 IEEE International Conference on Imaging Systems and Techniques, Proceedings (2013) 366–371doi:10.1109/IST.2013.6729723.
 - [97] M. Wang, A. L. M. Vilela, L. Tian, H. Xu, R. Du, A new time series prediction method based on complex network theory, in: 2017 IEEE International Conference on Big Data (Big Data), Vol. 2018-Janua, IEEE, 2017, pp. 4170–4175. doi:10.1109/BigData.2017.8258440. URL <http://ieeexplore.ieee.org/document/8258440/>
 - [98] P. Moriano, J. Pendleton, S. Rich, L. J. Camp, Insider Threat Event Detection in User-System Interactions, in: Proceedings of the 2017 International Workshop on Managing Insider Security Threats - MIST '17, ACM Press, New York, New York, USA, 2017, pp. 1–12. doi:10.1145/3139923.3139928. URL <http://dl.acm.org/citation.cfm?doid=3139923.3139928>
 - [99] R. K. Srivastava, K. Greff, J. Schmidhuber, Training very deep networks, in: Advances in neural information processing systems, 2015, pp. 2377–2385.
 - [100] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
 - [101] S. Xie, R. Girshick, P. Dollár, Z. Tu, K. He, Aggregated residual transformations for deep neural networks, in: Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, IEEE, 2017, pp. 5987–5995.
 - [102] G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger, Densely connected convolutional networks., in: CVPR, Vol. 1, 2017, p. 3.

8. Bio

Felipe C. Farias received the B.Sc. in Computer Engineering in 2014, in this occasion he awarded the Academic Laurea of Engineering, the M.Sc. degree in Computer Intelligence in 2016, both from University of Pernambuco (UPE). Ph.D. student of Machine Learning at Federal University of Pernambuco (UFPE). He is currently a Professor at Federal Institute of Education, Science and Technology of Pernambuco (IFPE). His research interests include Neural Networks, Deep Learning, Machine Learning, Network Science. More information at <https://scholar.google.com/citations?hl=pt-BR&user=zwRHO14AAAAJ> <http://lattes.cnpq.br/4598958786544738>

Teresa Ludermir received the Ph.D. degree in Artificial Neural Networks in 1990 from Imperial College, University of London, UK. From 1991 to 1992, she was a lecturer at Kings College London. She joined the Center of Informatics at Federal University of Pernambuco, Brazil, in September 1992, where she is currently a Professor and head of the Computational Intelligence Group. She has published over 300 articles in scientific journals and conferences, three books in Neural Networks (in Portuguese) and organized three of the Brazilian Symposium on Neural Networks. Her research interests include weightless Neural Networks, hybrid neural systems, machine learning and applications of Neural Networks. She is an IEEE senior member and a Research fellow level 1A (the higher level) of the National Research Council of Brazil (CNPq). More information at <https://scholar.google.com.br/citations?hl=pt-BR&user=w-tKJOwAAAAJ> <https://orcid.org/0000-0002-8980-6742> <http://lattes.cnpq.br/6321179>

Prof. Carmelo J. A. Bastos-Filho was born in Recife, Brazil, in 1978. He received the B.Sc. in electronics engineering and the M.Sc. and Ph.D. degrees in electrical engineering from Federal University of Pernambuco (UFPE) in 2000, 2003, and 2005, respectively. In 2006, he received the best Brazilian thesis award in electrical engineering. His interests are related to: the development of protocols and algorithms to manage and to design optical communication networks, development of novel swarm intelligence algorithms and the deployment of swarm intelligence for complex optimization and clustering problems, development and application of multiobjective optimization and many-objective optimization for real-world problems, Development and application of soft deep learning techniques, development of solutions using network sciences for big data, applications in robotics and applications in

biomedical problems. He is currently an Associate Professor at the Polytechnic School of the University of Pernambuco. He is the scientist-in-chief of the technological Park for Electronics and Industry 4.0 of Pernambuco. He is an IEEE senior member and a Research fellow level 1D of the National Research Council of Brazil (CNPq). He published over 230 full papers in journals and conferences and advised over 50 PhD and MSc candidates. More information at <http://scholar.google.com/citations?user=t3A96agAAAAJ&hl=en>

APPENDIX B – ANALYZING THE IMPACT OF DATA REPRESENTATIONS IN CLASSIFICATION PROBLEMS USING CLUSTERING

Analyzing the impact of data representations in classification problems using clustering

Felipe Costa Farias
Centro de Informática
Universidade Federal de
Pernambuco
Recife, Brazil
Instituto Federal de Educação,
Ciência e Tecnologia de
Pernambuco
Paulista, Brazil
felipefariax@gmail.com

Teresa Bernarda Luderemir
Centro de Informática
Universidade Federal de
Pernambuco
Recife, Brazil
tbl@cin.ufpe.br

Carmelo J. A. Bastos-Filho
Ecomp
Universidade de Pernambuco
Recife, Brazil
carmelofilho@ieee.org

Flávio Rosendo da Silva
Oliveira
Instituto Federal de Educação,
Ciência e Tecnologia de
Pernambuco
Paulista, Brazil
flavio.oliveira@paulista.ifpe.edu.br

Abstract— This work presents an investigation about how to better represent output data labels to be used in supervised training of classifiers. The posed hypothesis is that grouping cohesive patterns into clusters and assigning them sub-labels, may improve the classifier performance. We used 12 benchmark datasets to test our hypothesis. First, we create the clusters, and when appropriate, new sub-labels were generated, according to Fuzzy-CMeans and Silhouette score thresholds. After that, Multilayer Perceptrons were employed to model each dataset with cluster generated sub-labels, obtaining promising results. From results, we observed that in cases where the sub-labels were used, the accuracy increased with statistical significance with $p=0.05$ in 22 cases and remained statistically equivalent in 14 cases, presenting no decrease in accuracy.

Keywords—: *machine learning, data representations, fuzzy-cmeans, multilayer perceptron, sub-labels.*

I. INTRODUCTION

Machine Learning algorithms allow computers to learn how to solve complex problems by examining data. Learning processes can be of, at least, three different kinds: (i) Supervised Learning, (ii) Unsupervised Learning and (iii) Reinforcement Learning. Concerning Supervised Learning, the system to be trained needs data samples comprised of inputs to be presented to the model and the expected targets (labels) as model outputs. The process of dataset preparation, usually involves a human specialist to label all data samples, either manually or automatically according to programmatic means. This sample annotation is related, for example, to assigning patient records with sick or not sick tags for a given medical problem.

The process of label annotation may lead to at least 2 problems: (i) it is commonly a time-consuming task to manually label all data, therefore a person can become tired which may cause mistakes along the labeling task or; (ii) a given specialist may create labels that make sense to him/her, but in some cases, the machine learning algorithm could better tackle the learning task if one splits a given label into other labels which better characterize the problem decision boundaries. That kind of problem can also be caused due to

lack of expertise in the area to be modeled. For example, one specialist could label a financial dataset with two labels - buy and sell, but machine learning algorithms might learn better if the dataset had been annotated with buy, sell and hold labels, producing more concise groups of input data for each class. This work presents an investigation about the impact of creating independent sub-labels in a dataset to address this problem of how to better assign labels to a given dataset, through previous data clustering.

We organized the remainder of this work as follows: we present the background information in Section 2; in Section 3, we describe the deployed methodology, concerning clustering and classification phases; in Section 4, we present the experimental arrangement and the results; and finally, in Section 5, we present the discussion, conclusions and future works.

II. BACKGROUND

In this section, we present a brief background on machine learning and clustering algorithms, and we introduce some previous works from the literature related to our proposal.

A. Machine Learning

The Machine Learning field has many techniques that can learn the behavior of a system using data produced by this system, such as Decision Trees, Support Vector Machines, Artificial Neural Networks [1]. In this work, we have selected the last technique as our classifier, as it is one of the most successful techniques widely used to model complex systems [2]–[4].

Fig. 1 Fig. 1 shows a simple architecture of a Neural Networks, which can be seen as a system of interconnected nodes, called neurons, weighted by parameters learned from data presented at the learning phase to the model. It is inspired by the functioning of the brain, where signals are propagated from the input layer over the neurons through the layers of the model in order to activate specific output neurons on the output layer where each output neuron is responsible for a single class.

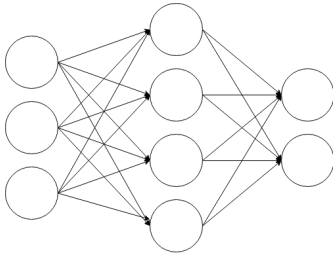


Fig. 1. Simple Neural Network with 3 inputs, 4 hidden nodes and 2 outputs.

B. Clustering

Clustering is a type of Unsupervised Learning that aims to automatically group data based on its similarities. Several techniques can be used to group data, such as K-Means, Self-Organizing Maps, Fuzzy-CMeans [5]. We have chosen the last technique to perform automatic clustering on our data.

Usually, the clustering algorithms need the number of clusters desired as a parameter of the technique. As we do not know the best number of clusters for each dataset, we have assessed a range for the number of clusters and evaluated the quality of the clustered data. There are several ways to assess the quality of the clusters found by the clustering algorithms such as Homogeneity Score, Mutual Information Score, V-measure Score, Calinski-Harabaz Score, Silhouette Score [6]. We have chosen the Silhouette Score because it relates the mean intra-cluster distance and the mean nearest-cluster distance for each sample.

C. Related Works

There are some approaches to subdivide the output space to facilitate the decision on several classifiers. This procedure is known as Class Decomposition. The goal in [7] was to expand binary problems into the multi-class classification aiming to increase the accuracy of the models. Their approach is divided into three steps: (i) separate a class into some subclasses using an unsupervised learning technique and re-label each sample as a function of the new subclasses. (ii) a supervised learning algorithm is applied to each version of these new problems. (iii) Combine and decide by voting. They used k-means in the unsupervised step with two possible values for $k=3$ and $k=5$, dividing the positive and negative classes into three and five subclasses, respectively. The C5.0 and C5.0-Boosted were used as classifiers. They found that it may increase accuracy, but it may be unfair to assume that it was caused only by the subdivision of classes as they are using several specialized classifiers, thus, applying more resources to classify the data with an approach that resembles an ensemble system.

The study in [8], a K-means algorithm is applied in imbalanced classes, clustering each large class locally, with the goal to produce sub-classes with equal sizes to ease the learning phase of supervised learning algorithms, such as Support Vector Machines (SVMs).

In [9] they applied a class decomposition via clustering in three steps: (i) decompose classes into clusters, (ii) search for

optimal class assignment configuration and (iii) a function mapping predictions to the original set of class labels. They used a Naïve Bayes and a Support Vector Machine to classify the data divided by an Expectation Maximization technique.

The work in [10] used the Clustering Inside Classes approach and shown that it can improve accuracy on linear classifiers, they evaluate the number of clusters per class and the size of the training set. Their approach used the same number of clusters for each label.

The paper in [11] proposed a technique called K Best Cluster-Based Neighbor (KB-CB-N) that integrate three different similarity measures for cluster-based classifications (distance, density, and gravity). They divided the approach into two steps: (i) apply Expectation Maximization technique to cluster the data for the member of each class and (ii) predict with their technique. They also show good improvements on the field.

A clustering based class decomposition was proposed in [12] to improve the performance of classifiers. They focus on the investigation of the effect of the k-means and hierarchical unsupervised techniques. They combined K-Means to cluster and Naïve Bayes to predict the data. They also assessed the number of clusters from 2 to 5, assigning the same number of clusters for classes in each trial.

The work in [13] uses a K-means clustering algorithm to decompose the classes to be presented to the Random Forests ensemble learning algorithm and shows that their method had significantly improved the accuracy of Random Forests.

An approach that exploits subclass information in the optimization process of the Support Vector Data Description (SVDD) was evaluated in [14]. They have evaluated it in face recognition and human action recognition, obtaining better performances. An extension of the aforementioned work, combined global and local geometric data relationships to regularize the process and applied it to image and video classification [15].

III. METHODOLOGY

In this section, we present details about the proposal, comprised of clustering and classification phases and also studied datasets.

The main idea of Divide-and-Conquer is to break problems into 2 or more sub-problems of the same type until these problems can be easily solved. These partial solutions can thus be combined to solve the whole original problem. **Fig. 2** shows a toy problem with two labels (circles and triangles) and a decision boundary of a simple classifier. As one can observe, there is no way to properly separate circles and triangles as the feature space is entangled when considering the original labels.

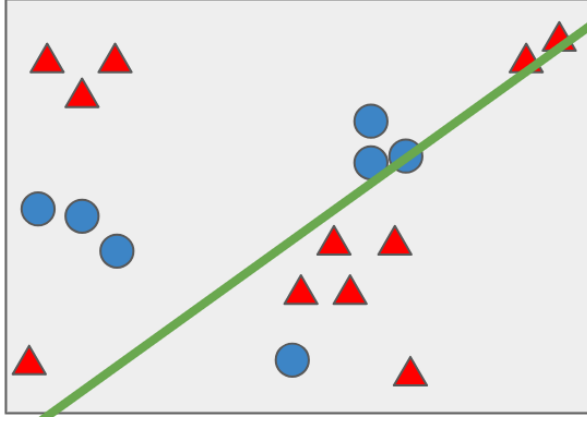


Fig 2. Example of a two labels problem and a decision boundary

Using this traditional approach, the classifier must learn how to model data of a given label, even containing patterns drastically different concerning its input space. However, if sub-labels were used as depicted in **Fig. 3**, one or more classifiers could focus on specific regions of input space. This work aims at verifying if this Divide-and-Conquer approach could make the learning process more natural and/or could produce more accurate classifiers.

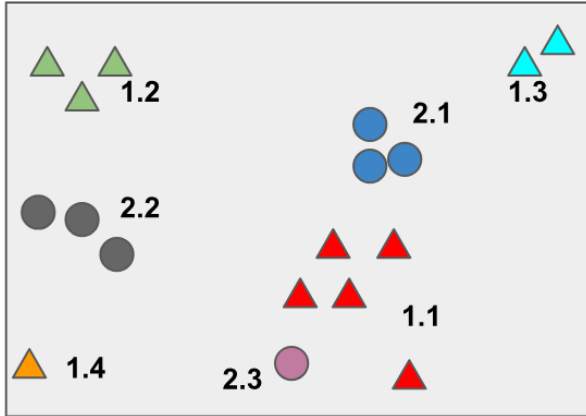


Fig. 3. Sub-labels created on the previous problem, transforming the 2 original labels into a problem with 7 labels (3 sub-labels for the circles and 4 sub-labels for the triangle)

Inspired by this idea, two phases were combined: a clustering phase (subsection A - *Clustering Phase*) and a classification phase (subsection B - *Classification Phase*). An overview of this approach can be seen in **Fig. 4**. First, an unsupervised algorithm was used to cluster data, aiming to break the problem of a complex classification into simpler ones. After that, we used a classifier to predict the final label considering the generated sub-labels as intermediate results. Finally, we extracted the evaluation metrics to be analyzed. In order to investigate the suitability of this Divide-and-Conquer approach, we used (a) several numbers of clusters, and (b) several thresholds.

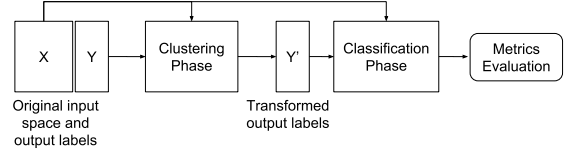


Fig. 4. Overview of proposed approach

A. Clustering Phase

A central part of the proposal comprises clustering input data and possibly creating sub-labels for each original label to match generated cohesive clusters. We sub-divided the original N output labels in $M \geq N$ sub-labels to assess this, so that the classifier may learn better the features of cohesive groups. When $M = N$ no subdivisions occurred in original labels.

In order to avoid clusters of sub-labels containing different original labels inside, patterns of different classes were clustered separately. Another argument to cluster each class individually is that the number of clusters to transform the original class into more cohesive ones, and probably easier to learn, may vary from class to class.

To assess cluster quality, the Silhouette Score was evaluated and compared with threshold levels to decide about using that cluster as a sublabel or not.

The equations used to calculate the Silhouette Score are given in (1) and (2).

$$S_i = (b_i - a_i) / \max \{a_i, b_i\} \quad (1)$$

$$S = \sum_{i=1}^N S_i / N, \quad (2)$$

where a_i is the average distance between i and all other data within the same cluster; b_i is the smallest average distance of i to all points in any other cluster, of which i is not a member; N is the total number of data samples; S_i is the Silhouette Score for data i and S is the average of the Silhouette Scores regarding all the data.

Tested number of clusters ranged from 2 to 10 and 20 thresholds between 0 and 1 were used to assess the Silhouette Score of each cluster inside each class. If this threshold level is smaller than the Silhouette Score of a cluster, this cluster is used as sublabel. Otherwise, no sublabel is created for that cluster. Also, it is worth mentioning that if threshold = 0, each cluster is always used as a sublabel.

A pseudo-code of this clustering phase is presented in **Alg. 1**.

Algorithm 1 Clustering Phase

Input: threshold: threshold of Silhouette Score, labels: list of unique labels, X: features, Y: labels

Output: Y': the possible recoded output

```

1. Y' = Y
3. for label in labels do
4.   x = X[Y==label]
5.   for k in [2 ... 10] do
6.     c = find_clusters(k, x)
7.     s = silhouette_score(c)
8.     clusters.append(c)
9.     silhouettes.append(s)
10.  best_idx = argmax(silhouettes)
11.  if silhouettes[best_idx] >= threshold:
12.    Y'[Y==label] = apply_sublabels(clusters[best_idx])

```

B. Classification Phase

A After the clustering phase, the classification was performed using the Scikit-Learn MLP implementation with standard parameters, only varying the number of hidden neurons. The difference that the MLP may experience is the recoding of the output layer, increasing the number of neurons according to the possible created sub-labels. For example, when the original labels are [0, 1, 2], totaling 3 labels that would require 3 output neurons at the output layer. After the clustering phase, the same data may have outputs recoded to [0, 0, 0, 1, 2, 2], totaling 6 virtually different sub-labels, thus needing 6 output neurons at the output layer.

C. Studied Datasets

In order to evaluate this proposal, we employed 12 public benchmark datasets. The number of instances, features, and labels for each dataset were summarized in the **Table I**. Those datasets were selected to encompass a large number of different characteristics, concerning the number of instances, features, and classes, thus allowing to better analyze the impact of the proposed approach in classification performance.

TABLE I. DATASETS

Dataset	# Instances	# Features	# Labels
balance-scale	625	4	3
breast	699	10	2
breast-cancer-wisconsin	569	30	2
diabetes	768	8	2
dna	3186	180	3
ecoli	327	7	5
iris	150	4	3
mushroom	8124	22	2
pendigits	10992	16	10
pima	768	8	2
satimage	6435	36	6
vehicle	846	18	4

IV. EXPERIMENTS AND RESULTS

To assess our hypothesis, we have simulated five times 10-fold cross-validations with 20 equally spaced thresholds between 0 and 1 to decide about using or not created clusters.

For each dataset, six possible MLP architectures with 10, 30, 50, 100, 150 and 200 hidden neurons were employed.

Experimental results were summarized in **Table II**. The first column contains the dataset analyzed. The following columns contain the results for the MLP architecture described above. Each cell may contain up to 3 lines: (i) in the first line, it can be seen the averaged accuracy without sub-labels creation; (ii) in the second line, it can be seen the averaged accuracy of threshold level with greatest averaged accuracy; and (iii) the third line shows absolute difference between the proposed method and the method without the sub-labels method. When there are no second or third lines, it means that the best threshold level (with the highest averaged accuracy) created no sublabel for any original label, keeping the same output labels as the original ones. The results show that in the cases where sub-labels were used, the absolute accuracy rate increased in 34 cases, remaining the same in 2 and have not decreased in any case. The average value for the threshold level was 0.24.

TABLE II. ACCURACY RATES (%) FOR MLP ARCHITECTURES BY DATASET AND NUMBER OF HIDDEN NEURONS

Dataset	10	30	50	100	150	200
balance-scale	88.17	93.18	95.24	96.67	96.89	96.86
breast	95.79	95.85	95.91 96.08 0.17	95.88 96.05 0.17	95.82 96.08 0.26	95.76 96.19 0.43
breast-cancer-wisconsin	96.88 97.44 0.56	97.13 97.72 0.59	97.19 97.76 0.57	97.37 97.79 0.42	97.40 97.80 0.40	97.65 97.76 0.11
diabetes	76.92	77.05	77.28	77.67	77.13	77.46
dna	94.17	94.71	95.07	95.28	95.24	95.31
ecoli	86.73	87.48	88.28 88.65 0.37	87.78 88.38 0.6	87.92	87.96
iris	94.80 96.27 1.47	95.60 96.67 1.07	96.27 96.80 0.53	96.40 97.20 0.8	96.67 97.33 0.66	96.67 97.33 0.66
mushroom	99.92 99.98 0.06	99.99 100 0.01	99.99 100 0.01	99.99 100 0.01	100 100 0	100 100 0
pendigits	97.20 97.46 0.26	98.74 98.82 0.08	98.98 99.00 0.2	99.12 99.19 0.07	99.16 99.20 0.04	99.20 99.25 0.05
pima	76.67	77.14	77.08	77.16	77.08	76.96
satimage	83.67 84.86 1.19	84.76 85.80 1.04	85.71 86.67 0.96	85.93 86.79 0.86	86.25 87.14 0.89	86.12 87.57 1.45
vehicle	78.27	79.73	80.06	80.30	80.11	80.25

Wilcoxon test with a statistical significance of 0.05 was employed to assess the quality of the proposed method. Results of Wilcoxon test are presented in **Table III** where ▲ means that the accuracy of sub-labels is greater than the accuracy without sub-labels with 0.05 significance level. The = symbol means that there is no statistical evidence that accuracies are different with a 0.05 significance level. The absence of second or third lines means that the best threshold level (with the highest averaged accuracy) created no sublabel for any original label, keeping the same output labels as the original ones. The accuracy increased with statistical significance in 22 cases, remaining statistically equivalent in 14 cases while there was no decrease in any case.

In cases of datasets balance-scale, diabetes, dna, pima, and vehicle, no subclasses were created because the best threshold found was not sufficient to create more than 1 cluster. When

sub-labels were effectively used, almost all the MLP architectures in that dataset had used subdivisions. This probably indicates that data was better represented in the new sub-labels space than in original labels. This is the case of breast, breast-cancer-wisconsin, ecoli, iris, mushroom, pendigits and satimage datasets.

TABLE III. WILCOXON TEST BY DATASET

Dataset	10	30	50	100	150	200
balance-scale	88.17	93.18	95.24	96.67	96.89	96.86
breast	95.79	95.85	95.91	95.88	95.82	95.76
			96.08	96.05	96.08	96.19
breast-cancer-wisconsin	96.88	97.13	97.19	97.37	97.40	97.65
diabetes	76.92	77.05	77.28	77.67	77.13	77.46
dna	94.17	94.71	95.07	95.28	95.24	95.31
ecoli	86.73	87.48	88.28	87.78	87.92	87.96
iris	94.80	95.60	96.27	96.40	96.67	96.67
mushroom	99.92	99.99	99.99	99.99	100	100
pendigits	97.20	98.74	98.98	99.12	99.16	99.20
pima	76.67	77.14	77.08	77.16	77.08	76.96
satimage	83.67	84.76	85.71	85.93	86.25	86.12
vehicle	78.27	79.73	80.06	80.30	80.11	80.25

Chosen sub-labels that appeared in at least one of experimental runs given the best threshold levels are presented in *Table IV*. It is possible to realize that the number of created sub-labels (clusters) was different for each class. For example, in the satimage dataset, the original labels 2 and 3 were not subdivided into other labels in any situation, probably because these classes are well defined and cohesive.

TABLE IV. CHOSEN SUB-LABELS

Dataset	Chosen Sub-labels
balance-scale	
breast	[[0, 0, 0, 1, 1], [0, 0, 1, 1]]
breast-cancer-wisconsin	[[0, 0, 1, 1, 1], [0, 0, 1, 1]]
diabetes	
dna	
ecoli	[[0, 1, 2, 3, 4], [0, 1, 2, 3, 3, 4], [0, 1, 2, 2, 3, 4]]
iris	[[0, 0, 1, 1, 2, 2], [0, 0, 1, 1, 2, 2, 2]] [[0, 1, 1, 2, 2, 2], [0, 1, 1, 2, 2], [0, 0, 1, 1, 2, 2, 2]]
mushroom	[[0, 0, 0, 0, 1, 1]]
pendigits	[[0, 1, 2, 3, 4, 5, 5, 6, 7, 8, 9]]
pima	
satimage	[[0, 0, 1, 1, 2, 3, 4, 5, 5], [0, 0, 1, 1, 2, 3, 4, 4, 5, 5]]
Vehicle	

Fig. 5 and *Fig. 6* shows, for each MLP architecture, the sum and the average of the absolute differences between the accuracies with and without sublabel usage.

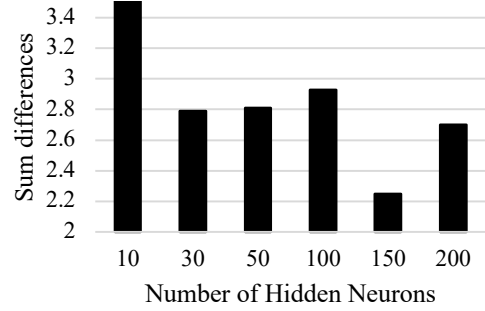


Fig. 5. Sum of absolute differences and the sum of the averages of the absolute differences between results with and without the sub-labels creation.

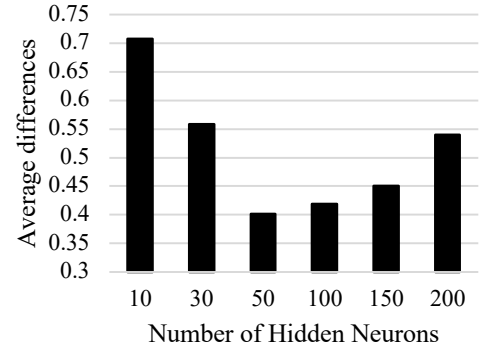


Fig. 6. Average of absolute differences and the sum of the averages of the absolute differences between results with and without the sub-labels creation.

As can be seen, the highest sum and average are perceived in the architecture with the lowest number of hidden neurons, suggesting that the use of sub-labels probably facilitates the learning process, optimizing learning capability usage. The number of hidden neurons on the larger architectures may already have been more than enough to handle the problems, in their original label representation.

V. CONCLUSIONS

Considering Divide-and-Conquer strategy inspiration, this work evaluated the suitability of creating, possibly different number of sub-labels, for each original label/class, based on clustering metrics to improve model feature learning and thus classification performance.

It is possible to observe that in the breast, breast-cancer-wisconsin, ecoli, iris, mushroom, pendigits and satimage datasets the use of sub-labels improved accuracy. This may suggest that a better representation of the output labels improved MLP learning. This performance increase may also happen due to more tight groups of data of the same sublabel, after clustering phase.

Accuracy improvements were more significant on models with a smaller number of hidden neurons, probably indicating that the use of sub-labels optimized model use of learning units.

The process of clustering inherently uses distance metrics, which may not be entirely appropriate for some types of dataset. The usage of another method for grouping similar labels, which does not use distance metrics, may have more success in defining the sub-labels.

Considering the promising results, it is reasonable to extend this study further. Future works shall comprise: (i) using different classifiers, such as Radial Basis Function Networks, which may help to better segment clusters; (ii) analyzing other cluster metrics, other than silhouette, to infer cluster quality; (iii) developing a method to automatically choose the threshold of employed metric to decide about considering or not a cluster as sublabel; (iv) using other cluster techniques without distance metrics; (v) increase the number of neurons in the hidden layer to better understand the trend of the improvements; (vi) evaluate the procedure on deep learning tasks; and (vii) compare results with other Machine Learning techniques, such as SVM.

REFERENCES

1. R. C. Eberhart and Y. Shi, *Computational Intelligence: Concepts to Implementations*. Elsevier Science, 2011.
2. Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data*, vol. 4. AMLBook New York, NY, USA:, 2012.
3. J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.
4. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
5. R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Trans. Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.
6. F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in {P}ython," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
7. N. Japkowicz, "Supervised learning with unsupervised output separation," *Int. Conf. Artif. Intell. Soft Comput.*, pp. 1–5, 2002.
8. J. Wu, H. Xiong, P. Wu, and J. Chen, "Local Decomposition for Rare Class Analysis," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007, pp. 814–823.
9. R. Vilalta, M.-K. Achari, and C. F. Eick, "Class decomposition via clustering: a new framework for low-variance classifiers," in *Third IEEE International Conference on Data Mining*, 2003, pp. 673–676.
10. D. Fradkin, "Clustering inside classes improves performance of linear classifiers," *Proc. - Int. Conf. Tools with Artif. Intell. ICTAI*, vol. 2, no. May, pp. 439–442, 2008.
11. Z. Said Abdallah and M. M. Gaber, "KB-CB-N classification: Towards unsupervised approach for supervised learning," *IEEE SSCI 2011 Symp. Ser. Comput. Intell. - CIDM 2011 2011 IEEE Symp. Comput. Intell. Data Min.*, pp. 283–290, 2011.
12. S. Banitaan, A. B. Nassif, and M. Azzeh, "Class Decomposition Using K-Means and Hierarchical Clustering," *2015 IEEE 14th Int. Conf. Mach. Learn. Appl.*, no. December, pp. 1263–1267, 2015.
13. E. Elyan and M. M. Gaber, "A fine-grained Random Forests using class decomposition: an application to medical diagnosis," *Neural Comput. Appl.*, vol. 27, no. 8, pp. 2279–2288, Nov. 2016.
14. V. Mygdalis, A. Iosifidis, A. Tefas, and I. Pitas, "Kernel subclass support vector description for face and human action recognition," in *2016 First International Workshop on Sensing, Processing and Learning for Intelligent Machines (SPLINE)*, 2016, pp. 1–5.
15. V. Mygdalis, A. Iosifidis, A. Tefas, and I. Pitas, "Semi-supervised subclass support vector data description for image and video classification," *Neurocomputing*, vol. 278, pp. 51–61, 2018.

APPENDIX C – CLUSTERING FOR DATA-DRIVEN UNRAVELING ARTIFICIAL NEURAL NETWORKS

Clustering for Data-driven Unraveling Artificial Neural Networks

Felipe Costa Farias^{1,2}[0000–0001–7411–5562], Teresa Bernarda
Ludermir¹[0000–0002–8980–6742], and Carmelo J. A.
Bastos-Filho³[0000–0002–0924–5341]

¹ Universidade Federal de Pernambuco, Centro de Informática, Recife PE, Brasil
tbl@cin.ufpe.br

² Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco, Paulista PE,
Brasil

felipefariax@gmail.com

<http://www.springer.com/gp/computer-science/lncs>

³ Universidade de Pernambuco, Recife PE, Brasil
carmelofilho@ieee.org

Abstract. This work presents an investigation on how to define Neural Networks (NN) architectures adopting a data-driven approach using clustering to create sub-labels to facilitate the learning process and to discover the number of neurons needed to compose the layers. We also increase the depth of the model aiming to represent the samples better, the more in-depth it flows into the model. We hypothesize that the clustering process identifies sub-regions in the feature space in which the samples belonging to the same cluster have strong similarities. We used seven benchmark datasets to validate our hypothesis using 10-fold cross validation 3 times. The proposed model increased the performance, while never decreased it, with statistical significance considering the p-value < 0.05 in comparison with a Multi-Layer Perceptron with a single hidden layer with approximately the same number of parameters of the architectures found by our approach.

Keywords: Neural networks · Data-driven architecture · Sub-labels · Clustering · Representation learning.

1 Introduction

Computational Intelligence studies adaptive mechanisms to facilitate intelligent behavior in complex and dynamic environments. It is often applied in situations where heuristics are insufficient to solve a problem associated with uncertainty or stochastic behavior. Machine Learning is a branch of Computational Intelligence that allows computers to learn by experience (data) without being explicitly programmed [3]. The research's attention has grown due to the amount of available data and the proliferation of technologies, such as Internet of Things and Big Data [6]. Consequently, we have experienced the proposal of sophisticated models to learn from datasets with a large number of examples.

There are several types of Machine Learning algorithms. In Supervised Machine Learning, the algorithms during the training phase try to process inputs X and create an association with known outputs Y . One of the most used families of algorithms to perform this task is Artificial Neural Networks (ANN) [5]. Several works investigate the automatic construction of ANN architectures. This problem is known as Neural Architecture Search (NAS). The NAS approaches aim to decrease human intervention during the modeling process of an ANN. It offers mechanisms to propose ANN architecture automatically. However, most of the strategies presented in the literature does not consider the characteristics of the data involved in the problem to build the ANN architecture.

Among the strategies for solving the NAS problem, we observed the application of global optimizers such as Genetic Algorithms (GA) [11]. We highlight three works that use Genetic Algorithms and Reinforcement Learning to perform this task. The NeuroEvolution of Augmenting Topologies (NEAT) [14, 15] uses GAs to find the structure and weights of ANNs, encoding these attributes as part of the individual deployed in the evolutionary process. The strategy of Reinforcement Learning was used in [16]. Despite the good results, they have used 800 NVidia K-40 GPUs for 28 days, needing 22,400 GPU hours of processing time. The same strategy was proposed in [17]. However, they used the concept of transferability to allow learning from a simple dataset and applying it to a more complex dataset. Again, the processing time is a big challenge. They used 500 NVidia P100 GPUs for four days, totaling 2,000 GPU hours. In order to explore the architecture space based on the current network and reusing its weights, a Reinforcement Learning meta-controller was used to grow the network depth/layer width in [1]. They used 5 GPUs during 2 days, training 450 networks. These methods can achieve reasonable results, but they are often time-consuming.

In a different branch, we have the work [9] that focuses on searching for the architecture composition progressively, starting from the simplest candidates to the more complex ones. This proposal is based on *Blocks* constructing *Cells* that will compose entire *Networks*. A Binary Particle Swarm Optimization (BPSO) algorithm was used in [10] to define the architecture of an ANN that has no regular layers. These proposal are interesting but does not explicitly take into account the distribution information regarding the dataset. It is important to observe this information since the model will use samples of the dataset to train and the neurons should behave plausibly with respect to its inputs. For example, the neurons should neither explode or vanish its activations for all the dataset but have specific activation patterns for each label.

Even though the NAS has good results, these methods try several architectures to find which one is the best through a search algorithm, often using several GPUs for several hours. Differently from NAS approaches, in this work we investigate how to design a data-driven ANN Architecture using clustering to discover the number of neurons of a given layer and iteratively increase the depth of this ANN without the need to initializing and training many different architectures. We only start with the inputs and apply our strategy to create the subsequent

hidden and output layers. The layer that will be created is based on the current data representation that this layer will have as inputs. We use the dataset distribution in order to start with weights that activates more to specific labels and less to all the other labels. We have focused on tabular datasets since several real-world applications share this data representation. To investigate this problem, We have used benchmark datasets to evaluate our hypothesis. Our goals are to infer (i) the width (number of neurons) through clustering of the feature space to find sub-labels that share strong similarities, creating specific neurons that activate to specific sub-labels; and (ii) the depth (number of layers) of an ANN regarding the specificity of each dataset. The main idea concerns on each layer representing the samples in a more straightforward manner to the next ones. We aim to disentangle the representations into a space that is easier for the classifier on the last layers of the model to recognize the patterns. We name this process as Clustering for Data-driven Unraveling (DDU) Artificial Neural Networks.

The remainder of this work was organized as follows. We present the background information in Section 2. We describe the proposed methodology to data-driven evolve a ANN architecture in Section 3. We show the experimental arrangement and the results in Section 4. Finally, we present the discussion, conclusions, and future works in Section 5.

2 Our proposal

In this section, we present the proposal's details, comprising the creation of sub-labels and the definition of the layers.

Since some samples of a specific label can lay near the same region while other samples of the same label may be in other regions, it may harm the learning process. We have used clustering techniques to find the essential regions of interest. The clustering process aims to create sub-labels based on the proposal presented in [4]. The primary goal is to enhance the learning process. We use the clusters to map each one of the sub-labels to a single neuron in the forward layer and compose the entire layer. We add layers composed of neurons from clustering in an iterative manner. Fig. 1 presents an example of a possible data-driven defined architecture of a ANN through the processing of our proposal.

The *width* of each layer is defined by the number of neurons that compose a specific layer. We find the number of layers by applying the clustering process. On the other hand, the *depth* of the DDU ANN is the number of layers. The number of layers is also determined automatically in the appending new layers process. The appending process is applied iteratively until we reach a stop criterion. Fig. 2 shows the steps that compose our proposal.

Given the inputs $x_i \in X$ and the outputs $y_i \in Y$, we use GMMs (Gaussian Mixture Models) clustering algorithm to create sub-labels as shown in [4] using the X as inputs. This process create the clusters regarding each label separately and apply the prediction of each created cluster to all the data in order to calculate the scores discussed in the next paragraph. We generate GMMs with

4 Farias et al.

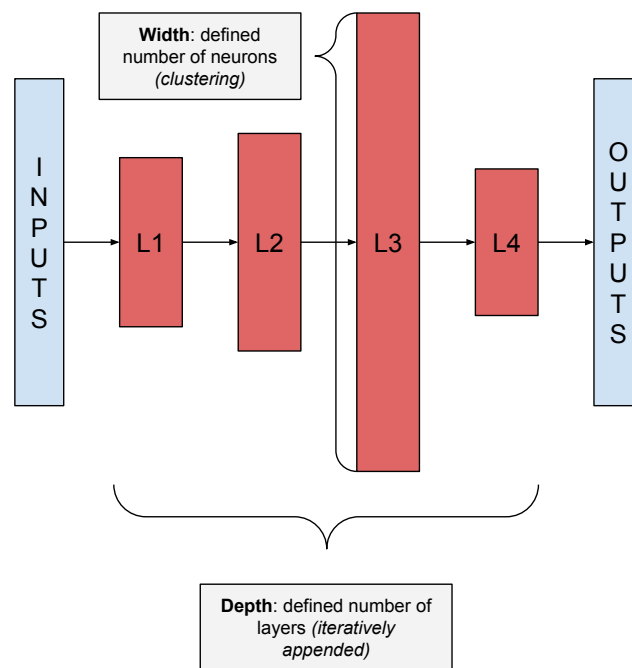


Fig. 1. Example of architecture found by the Data-driven Unraveling technique. The inputs are the features of the samples, while the outputs are its labels.

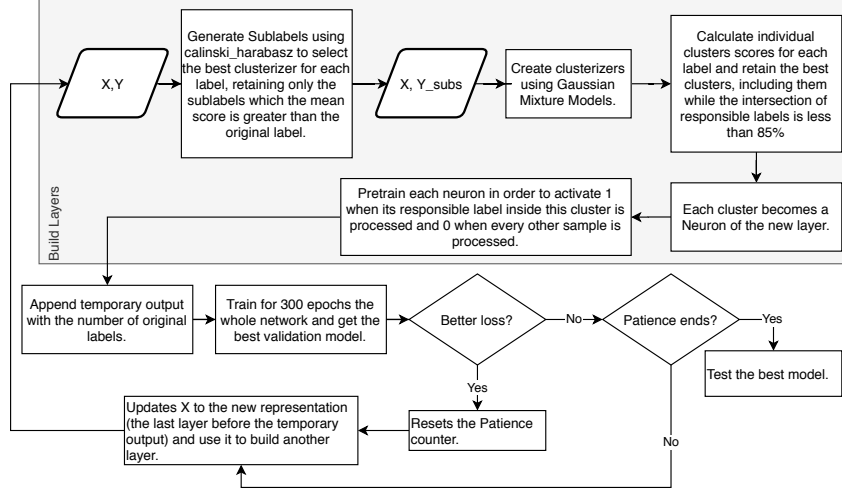


Fig. 2. Diagram of the proposed methodology.

the number of clusters in $[2, 3, 4, 5]$ repeating the process for 2 times, totaling 8 GMMs generation. We decided to deploy the GMM technique since it presented the highest Calinski-Harabasz score [2] when compared to Silhouette Score for related. We also expect that the samples of the same label could converge to the same spatial region as the samples are processed by layers and we believe that it could be approximated by Gaussian distributions due to the Central Limit Theorem.

We defined purity and belongingness metrics, combining them into a single score assigned to each cluster. Since each cluster is assigned to a specific label or sub-label (responsible label), the score is related to that specific responsible label inside the cluster found. Purity is defined as the percentage of a specific label regarding all the elements inside this cluster. Belongingness is the percentage of a specific label within the cluster regarding all samples of this label. For each responsible label, we assign the cluster with maximum score. The rationale relies on the idea that the maximum score value returns all the samples of a single label (belongingness=100%) in a cohesive region without other labels inside (purity=100%). The score is minimum when it has a low number of an assigned label inside this cluster, regarding all the feature space (low belongingness), and the cluster has a significant number of samples not belonging to the assigned responsible label (low purity). As we need to define a single indicator in our methodology to assess the clustering process, we decided to combine these two metrics as follows:

$$score = kappa * purity + (1 - kappa) * belongingness \quad (1)$$

$kappa$ is automatically calculated using the following strategy to give the same importance to the two metrics:

$$kappa = k_p \quad (2)$$

$$k_b = (1 - k_p) \quad (3)$$

$$k_p * \bar{p} = k_b * \bar{b} \quad (4)$$

where k_p is the *kappa* regarding the purity and k_b is the *kappa* regarding the belongingness, \bar{b} is the average belongingness and \bar{p} is the average purity for each label.

It is easy to observe that Eq. (4) can be rewritten as

$$kappa = 1 - \frac{1}{(\frac{\bar{b}}{\bar{p}} + 1)} \quad (5)$$

After defining the sub-labels, we evaluate if the sublabeled scenario is better than the original one. For each sub-label, we evaluate if the average score regarding the clusters with this responsible sub-label has increased compared to the averaged score of the clusters responsible for its original label that derived the sub-label. In this process, we may have chosen some, all, or none of the created sub-labels.

If we have chosen some sub-labels, we adjust the Y to use the proposed sub-labels, treating them as different labels because it has shown different features to be considered for the same class. By doing this, we believe we can create more concise groups of samples, facilitating the classifier's learning process.

After the generation of possible sub-labels, we apply the GMMs clustering algorithm with the number of clusters in $[L, L + 1, L + 2, \dots, L + 7]$ for 2 times, totaling 16 GMMs generation. We analyze each one of the clusters independently of the GMM generation process. We calculate their scores and assign their responsible labels. Then, we sort them, for each label, from the highest to lowest score. After this, for each label (or sub-label), we select the clusters until the intersection of the elements of the responsible label inside the specific cluster - we save the indexes of the responsible labels handled by each cluster - is less than 85% of the union of the responsible labels already retained by the clusters already chosen for this specific label (or sub-label).

Each one of the retained clusters is mapped into a neuron and pre-trained for 5 epochs with a higher learning rate of 0.01 trying to activate 1 for the responsible label elements inside this cluster and 0 otherwise. Our goal is to make this neuron trigger specifically for activations of this label (or sub-label) when the model processes a sample near that region of clustering. So we applied a traditional SGD training with this modified outputs. This process finishes with a layer created based on the clusters and the weights pre-trained to activate in the responsible labels. We append this layer with a SELU [8] function activation since it appears to self-normalize the network, into the model, and append a temporary output layer with the number of labels of the original problem together with a LogSoftmax activation.

We train the model for 300 epochs and select the model with the lowest validation loss. If the global loss in this validation set decreases, we reset the patience parameter and do the same process to add another layer using the activations of the last layer (before the temporary output) as inputs to the new layer. If the loss increases, we increment the patience counter and stops the depth growing when the patience counter reaches 3.

We present the pseudo-code describing the algorithm to perform the DDU method in Algorithm 1.

Algorithm 1 Algorithm for DDU

Require: X, Y

```

1: C = []
2: patience = 0
3: while patience < 3 do
4:   current_y = Y
5:   Y_subs = generate_sublabels(X)
6:   if better_performance(Y_subs) then
7:     current_y = Y_subs
8:   end if
9:   clusters = create_GMMs(X)
10:  calculate_scores(clusters, Y_subs)
11:  sort(clusters) // from highest to lowest score
12:  for label in labels: do
13:    for cluster in clusters[label]: do
14:      if responsible_label_intersection < 85% then
15:        C.append(cluster)
16:      end if
17:    end for
18:  end for
19:  Pretrain neurons regarding C
20:  Append temporary output
21:  Train entire model for 300 epochs
22:  loss = model(validation_split)
23:  if loss decreased then
24:    best_model = model
25:    patience = 0
26:  else
27:    patience++
28:  end if
29:  X = model.represent(X)
30: end while
31: return best_model.

```

3 Experiments and Results

We select seven benchmark datasets to validate our proposal. We present the datasets in Table 1.

Table 1. Datasets used as benchmarks

Dataset	# Instances	# Features	# Labels
Glass	214	9	6
Ionosphere	315	34	2
Iris	150	4	3
Pima	768	8	2
Satimage	6430	36	6
Tic-tac-toe	958	9	2
Vehicle	846	18	4

We have used a Stratified 10-Fold and repeated it three times, resulting in 30 trials. In the process of the Stratified 10-fold, we divided the entire dataset into ten mutually exclusive splits. We maintained the proportion of labels in each subset. The first run uses the first split for test, whereas the other nine are used for Train/Validation. In the second run, the split number 2 is used for test, whereas the other nine are used for train/Validation. We repeat the process until the last run uses the split number 10 for test and the first nine splits are used for train/Validation. We also have saved the indexes that were presented in each fold to be used during the comparison with a different model: Multilayer Perceptrons (MLPs) with only one hidden layer with the number of weights approximately the same of the architecture generated by our proposal, regarding the samples presented in that arrangement.

We have used the PyTorch [12] and Scikit-Learn [13] libraries to implement our proposal. We have chosen the ADAM optimizer [7] with a learning rate of 0.001 for the training of the entire model with the Negative Log-likelihood loss and 0.1 for the pre-training of each neuron given their related clusters with the Mean Squared Error loss.

Table 2 compares the results of the proposed technique (DDU) and MLPs with only one hidden layer with the number of weights - and not the number of neurons - approximately equal to each one of the architectures created by the DDU. We have used this strategy to compare the neuron arrangement's importance in different layers, with approximately the same number of parameters. For each dataset, the average accuracy of 30 runs for each set (train, validation, and test) is given, and its standard deviation appears between parenthesis. We also have applied the Wilcoxon statistical test between the two techniques comparing the same dataset, highlighting the text where it presented a $p - value < 0.05$.

We split the data into Train, Validation and Test sets, and we expect these three splits to share the same sample distribution. The Validation Set is often

Table 2. Results in Train, Validation and Test sets regarding DDU and MLP models for each dataset

Model	Dataset	Train	Validation	Test
DDU	pima	77.99 (0.01)	+ 78.48 (0.02)	76.74 (0.03)
MLP	pima	78.07 (0.01)	76.71 (0.03)	76.39 (0.02)
DDU	vehicle	+ 84.12 (0.02)	+ 80.98 (0.03)	+ 80.90 (0.04)
MLP	vehicle	82.81 (0.01)	79.34 (0.02)	79.40 (0.03)
DDU	glass	66.88 (0.05)	+ 65.01 (0.05)	62.16 (0.10)
MLP	glass	66.09 (0.03)	60.83 (0.06)	62.33 (0.09)
DDU	tic-tac-toe	82.33 (0.06)	77.89 (0.06)	67.14 (0.11)
MLP	tic-tac-toe	82.46 (0.04)	77.98 (0.05)	70.79 (0.10)
DDU	iris	96.04 (0.02)	97.16 (0.03)	95.33 (0.06)
MLP	iris	+ 97.36 (0.01)	95.98 (0.03)	95.56 (0.06)
DDU	satimage	+ 91.16 (0.01)	+ 89.52 (0.01)	+ 88.86 (0.02)
MLP	satimage	89.58 (0.01)	88.17 (0.01)	88.06 (0.01)
DDU	ionosphere	93.50 (0.02)	+ 89.37 (0.03)	84.82 (0.08)
MLP	ionosphere	93.70 (0.01)	88.06 (0.03)	85.96 (0.08)

used as a proxy to decide when to stop the learning process of a model because we assume that it would share the distribution/behavior of the unseen test data. Our technique presented better accuracies in the Validation Set, which was not necessarily reflected in the Test Set. It maybe has occurred due to our process of K-Fold splitting. As we have used the Stratified K-Fold, we expected that the train, validation and test splits should have approximately the same feature distribution. Still, it probably has not happened since it guarantees the same percentage of labels in each split, but each split may still not be statistically equivalent in the feature space distribution. Also, the split was based on the original labels. That probably could generate completely different sub-labels during the process once the feature space distribution information was not used at the splitting process, as it is based solely on the output labels. Considering that in each trial, the same indexes were presented to both techniques, the DDU appears to have a better knowledge extraction since the validation accuracy improved and is statistically significant, and the training accuracy not necessarily accompanied. Maybe a better way to split the data should be a stratification based in regions that each sample occupies in the feature space. In this way, inside the same region/cluster, we should stratify the samples into the K splits.

As one can see in Table 3, the DDU generated some architectures, including ones with more than one hidden layer, that show better accuracies in different splits. It is worth remembering that the number of neurons of the MLPs was calculated to match the number of parameters (weights) found by DDU in each run. As it shows, meaning that not only the learning capacity (stored in weights) matters but also the arrangement of neurons in different layers influences the results.

The architectures with the minimum, median, and maximum number of neurons/layers generated for each dataset by the DDU are presented in Table 3. The column **Architecture** shows the complete NN architecture starting from the input layer to the output layer. The column **Weights** presents the number of weights existent in that specific architecture. The columns **Train**, **Val.** and **Test** show the accuracy for each split.

Table 3. Minimum, Median and Maximum number of weights in found Architectures

Dataset	Architecture	Weights	Train	Val.	Test
glass	[9 9 6]	216	64,58	60,42	50,00
glass	[9 12 12 6]	405	67,36	65,31	71,43
glass	[9 11 14 16 13 13 10 6]	1125	72,92	73,47	71,43
ionosphere	[34 2 3 2]	1236	92,83	88,61	80,00
ionosphere	[34 5 4 2]	1354	92,41	88,61	94,29
ionosphere	[34 5 8 6 2 6 3 5 2]	1481	96,20	89,87	82,86
iris	[4 3 3]	37	97,03	97,06	93,33
iris	[4 4 5 3]	67	96,04	97,06	100,00
iris	[4 9 6 7 6 5 3]	235	97,03	100,00	100,00
pima	[8 2 2]	84	77,22	78,61	76,62
pima	[8 5 3 4 3 5 2]	168	77,22	76,88	79,22
pima	[8 6 7 5 5 2 7 5 3 4 2]	308	76,88	79,77	77,63
satimage	[36 10 11 6]	1832	89,91	88,80	88,02
satimage	[36 13 12 15 6]	2190	90,90	89,15	88,18
satimage	[36 11 12 15 13 14 18 13 13 12 13 12 13 12 11 6]	4014	93,32	90,81	90,67
tic-tac-toe	[9 5 8 2]	182	73,99	75,00	71,88
tic-tac-toe	[9 7 6 6 5 7 3 2]	314	87,62	82,41	72,92
tic-tac-toe	[9 8 8 3 3 5 5 5 5 3 4 4 4 3 4 2]	431	88,08	86,57	47,92
vehicle	[18 12 4]	588	79,82	78,53	78,82
vehicle	[18 13 12 4]	762	82,84	81,15	86,90
vehicle	[18 12 15 9 8 4]	959	83,19	81,15	89,29

We highlight that the generated architectures do not necessarily create shapes that always increase or decrease the number of hidden neurons. It might increase, decrease, and increase again if the representation became locally worse at a given depth. The layers near the output tend to have a low number of hidden neurons, probably indicating that less resource is needed to treat the problem at that depth. It happens since representations of the samples were facilitated by the possible disentanglement of the feature space done by all the previous layers.

Regarding the Validation split, the accuracy increased as the architecture used more weights and/or layers in most cases. It occurs since the samples are processed through the layers, thus probably creating better representations which the final output layer can classify with fewer efforts. On the other hand,

the layers near the input tend to have a larger number of neurons due to the high entanglement of the features at the early stages.

4 Conclusions

In this paper, we presented a preliminary study on how to define a Data-Driven Neural Network Architecture without creating several Networks. To assess this, we have used GMMs clustering to define each layer's width and iteratively appending layers, increasing depth, aiming to find better and possibly disentangled representations, easing the model learning process.

As presented in Table 2, the performance has increased statistically (Wilcoxon significance test with $\alpha = 0.05$) on two occasions in the Test split while never decreased it. It also corroborates with our initial hypothesis that clustering the feature space may reveal the number of necessary neurons representing the samples in a more organized way. The quantity of neurons seems correlated with the high non-linearity entanglement at the first layers and with the possible low entanglement at the last layers of the model. If we consider the Validation split, that is considered as a proxy to the accuracy in Test split, our approach was better in 5 cases out of 7.

Our proposal has shown some exciting results, and our hypothesis could be evaluated in these initial experiments. For future works, we intend to assess the technique in more benchmark datasets and adapt the method to work with datasets usually used in Deep Learning tasks such as MNIST and CIFAR10. We also want to perform more evaluations using the data-driven found architectures to evaluate if a pruning process would perform successfully or if it would fail, leading us to believe that the DDU is capable of creating effective architectures with a small number of useless neurons/weights. Another valid investigation is to split the data stratified by region and by labels inside each region. The evaluation of other Clustering Techniques and strategies to retain the cluster may impact the performances. Also, more tests on hyperparameters values need to be done to have more robust conclusions about its impact on the learning process. We also plan to compare the technique with other approaches, such as the NEAT.

References

1. Cai, H., Chen, T., Zhang, W., Yu, Y., Wang, J.: Efficient architecture search by network transformation. In: Thirty-Second AAAI conference on artificial intelligence (2018)
2. Caliński, T., Harabasz, J.: A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods* **3**(1), 1–27 (1974)
3. Eberhart, R.C., Shi, Y.: *Computational Intelligence: Concepts to Implementations*. Elsevier Science (2011)
4. Farias, F.C., Bernarda Ludermir, T., Bastos-Filho, C.J.A., Rosendo da Silva Oliveira, F.: Analyzing the impact of data representations in classification problems using clustering. In: 2019 International Joint Conference on Neural Networks (IJCNN). pp. 1–6 (July 2019)

12 Farias et al.

5. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep learning, vol. 1. MIT press Cambridge (2016)
6. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems* **29**(7), 1645–1660 (2013)
7. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014)
8. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. In: *Advances in neural information processing systems*. pp. 971–980 (2017)
9. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. pp. 19–34 (2018)
10. Liu, J., Gong, M., He, H.: Nucleus neural network: A data-driven self-organized architecture. *arXiv preprint arXiv:1904.04036* (2019)
11. Mitchell, M.: An introduction to genetic algorithms. MIT press (1998)
12. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., dAlché Buc, F., Fox, E., Garnett, R. (eds.) *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc. (2019)
13. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011)
14. Stanley, K.O., Clune, J., Lehman, J., Miikkulainen, R.: Designing neural networks through neuroevolution. *Nature Machine Intelligence* **1**(1), 24–35 (2019)
15. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. *Evolutionary computation* **10**(2), 99–127 (2002)
16. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016)
17. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 8697–8710 (2018)

APPENDIX D – SIMILARITY BASED STRATIFIED SPLITTING: AN APPROACH TO TRAIN BETTER CLASSIFIERS

Highlights

Similarity Based Stratified Splitting: an approach to train better classifiers

Felipe C. Farias, Teresa B. Ludermir, Carmelo J. A. Bastos-Filho

- Split the data using input and output distribution information;
- An approach to train better classifiers;
- Classifier agnostic and low-cost proposal to increase models' performance.

Similarity Based Stratified Splitting: an approach to train better classifiers

Felipe C. Farias^{a,b}, Teresa B. Ludermir^b and Carmelo J. A. Bastos-Filho^c

^aUniversidade Federal de Pernambuco, Centro de Informática, Recife PE, Brasil

^bInstituto Federal de Educação, Ciência e Tecnologia de Pernambuco, Paulista PE, Brasil

^cUniversidade de Pernambuco, EComp, Recife PE, Brasil

ARTICLE INFO

Keywords:

k-fold

cross-validation

machine learning

samples similarity

classifiers

data splitting

ABSTRACT

We propose a Similarity-Based Stratified Splitting (SBSS) technique, which uses both the output and input space information to split the data. The splits are generated using similarity functions among samples to place similar samples in different splits. This approach allows for a better representation of the data in the training phase. This strategy leads to a more realistic performance estimation when used in real-world applications. We evaluate our proposal in twenty-two benchmark datasets with classifiers such as Multi-Layer Perceptron, Support Vector Machine, Random Forest and K-Nearest Neighbors, and five similarity functions Cityblock, Chebyshev, Cosine, Correlation, and Euclidean. According to the Wilcoxon Sign-Rank test, our approach consistently outperformed ordinary stratified 10-fold cross-validation in 75% of the assessed scenarios.


1. Introduction

Machine Learning systems use data to extract knowledge. The goal is to store information in the internal parameters to analyze future unseen data. Some learning paradigms have been used in machine learning. In Supervised learning, the inputs and desired outputs (targets) are presented to the model. In Unsupervised learning, no output is given—the model clusters data based on the similarity among the elements. Conversely, in the Reinforcement learning paradigm, the model acts in an environment and evaluates if the last actions led to a better environment metric, receiving a reward or a punishment. Modeling a supervised learning system requires a dataset $D = \{(x_i, y_i) | x_i \in X, y_i \in Y\}$, where X is the input in the feature space, and Y is the output, which may be labeled for classification or real values for regression applications.

We may define a Machine Learning system as a combination of three elements: (i) a model M , which is a mathematical function mapping the domain X into image Y , processing the input X using its internal parameters W ; (ii) an Error/Cost/Loss Function $L = \text{error}(M(X), Y)$, which evaluates the performance of M in D ; and (iii) an Optimizer (O), which minimizes the function L . The training phase of a Learning System tries to find the best W for a model M using data D to estimate the parameters W , minimizing the error L using O . In other words, $W = \text{argmin}(L(M(X), Y))$ with W estimated by $O(L)$.

A typical approach to conceive a Supervised Learning System needs to have data X_{train} (training set) presented to the model to estimate the parameters W and different data X_{test} (validation set or test set), which is not used to modify the models' parameters W , but to evaluate its performance, serving as an estimation of real-world data during the inference phase. Several strategies have been developed to split data better (Kohavi et al., 1995). The error considering the training set should never be used alone as a model's performance estimator since some problems during the training phase may arise. The most common problems are over-fitting and under-fitting (Bishop, 2006; Russell & Norvig, 2020). To mitigate these problems, we can split the data into two or more subsets. We briefly discuss some split data methods in the next paragraphs.

One of the most common split data methods, probably due to its simplicity, is the Holdout Splitting. Holdout randomly divides the original dataset into a training set from which the algorithm produces the model M and a test set on which the performance of M is evaluated (Russell & Norvig, 2020). A common choice is to use 75% of the

 felipefariax@gmail.com (F.C. Farias); tbl@cin.ufpe.br (T.B. Ludermir); carmelofilho@ieee.org (C.J.A. Bastos-Filho)
ORCID(s): 0000-0001-7411-5562 (F.C. Farias); 0000-0002-8980-6742 (T.B. Ludermir); 0000-0002-0924-5341 (C.J.A. Bastos-Filho)

samples to the training set and 25% to the test set. It is desired that the training and test sets contain different samples and follow approximately the same distribution, which is not always the case.

In the K-Fold cross-validation, the data is divided into k subsets. Then k rounds of learning are performed. On each round, $1/k$ of the data is used as the test set, while the remaining samples are used as the training data. The average test set performance of the k rounds should be calculated. Popular values for k are 5 and 10 (Breiman & Spector, 1992; Wong, 2015; Russell & Norvig, 2020). An interesting analysis of K-Fold cross-validation estimates can be found in (Wong & Yang, 2017; Jung, 2018; Wong & Yeh, 2019).

Leave One Out is K-Fold cross-validation when k assumes the original dataset number of samples (Russell & Norvig, 2020). Consequently, it uses a test set with just one sample and the training set with all others. The process is repeated until every single sample belongs to the test set once. Then, the model's performances in the test sets are also averaged.

If the dataset is imbalanced, a stratified splitting may be recommended since it divides the dataset, maintaining the class proportion in each of the subsets created. It is even worse when there is a small number of samples to be trained.

Cross-validation is used to estimate the model generalization to an independent dataset. It is commonly applied to learning systems to predict its future performance or how accurate it would be when used in the real world, where the model never received the data during the training phase. It can also be used to (i) stop the training phase at a point which if the model was trained below or above that quantity, and it could present under-fitting or over-fitting behaviors, respectively; (ii) compare the performance of different models submitted to the same data; (iii) select the best model over several runs when there is a stochastic component intrinsic to the model; (iv) choose the classifiers that will compose an ensemble system.

Since it is a common approach to use cross-validation to obtain the best model to run on real-world problems, we should expect that our data during the training phase could have approximately the same distribution of data received in real-world applications. Besides, several algorithms rely on the fact that the samples in each subset are Independent and Identically Distributed (Haykin & Haykin, 2009; Japkowicz & Shah, 2011). These are reasonable assumptions since the process of learning means to model the distribution of the training data as close as possible, expecting that future data seen in the real-world follow the approximated distribution. None of those mentioned earlier strategies grants it or even have robust mechanisms to partially induce this assumption since they use the labels or the output distribution to split the data, ignoring all the input space distribution information. The DUPLEX algorithm (Snee, 1977) uses the input space information to divide the data into two disjoint sets with statistical similarity and cover almost the same region of input space. It uses the Euclidean distance between all pairs of samples to place the most distant samples in the same set, alternating from the train set and test set. As it was created to solve regression problems, it does not consider the output distribution of the labels nor the stratification process.

Our research question is posed as follows:

RQ: Can we use input and output space information during the data splitting process to cover all the regions where samples exist, aiming to maintain approximately the same statistical properties, and to generate better data to create better classifiers?

Consider a real-world intelligent system. The most common approach to train an Artificial Neural Network in this task is to (i) split the data into train and validation sets; (ii) define an upper bound limit of epochs for the model to be trained, and (iii) define a patience threshold for early stopping the training before the limit of epochs based on how many times the model presented consecutive performance decreases. At the end of each epoch, considering the training set, the validation set is used to assess the model's performance. As the validation set is not used to change the model's parameters, it acts as a proxy set of the data that the model will be presented in real-world operation. It is natural to expect that higher performances in the validation set consequently generate better models in real-world usage, leading us to use the model that shows the smallest error in the validation set. In order to happen what we expect, it is a necessary condition that the validation set has samples drawn from approximately the same distribution of the real-world data that the model will see in the future. In other words, our splitting process should look not only to the output space but also the input distribution of samples plays an essential role in the model's performance. In this work, we propose a low-cost method to increase the model's performance by better selecting the training data to be presented using similarity functions to place similar samples in different splits.

We organized the remainder of this work as follows. We present the background information, describe the proposed methodology to split the data using similarity functions, and provide experiment details in Section 2. We show the results and discussions in Section 3. Finally, we present conclusions and future works in Section 4.

2. Material and Methods

2.1. Our Proposal

In our proposal, we consider the output space and the distribution of inputs/features space to create the splits to validate our hypothesis. We call it Similarity-Based Stratified Splitting (SBSS). Even though our technique is not limited to a specific data splitting strategy, we focused our evaluations on Stratified 10-Fold. We used its folds to compose the train and test set, the latest mimicking the model's real-world usage.

To illustrate our hypothesis, in Figure 1, we show a dataset with 20 samples divided into two labels.

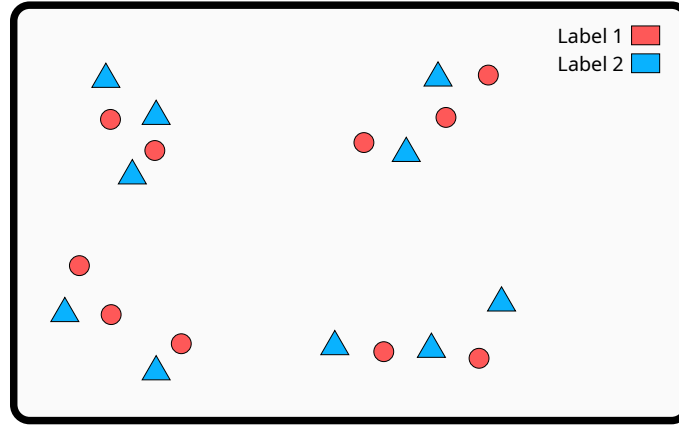


Figure 1: Example of dataset with 2 labels.

If we use a conventional stratified split process that does not consider the input space distribution, at worst case, we can create two bad splits, as shown in Figure 2. If we use Split 1 as the training set and Split 2 as the test set, the model will probably perform poorly since the samples lie in a region that the model was not exposed to data and did not learn how to separate the samples at this sub-space. The splitting process is also used to select the hyper-parameters of a model in a proxy subset of the data not presented in the training phase. This proxy subset should follow approximately the same distribution of the real data that the model will be presented when inferring in production. If the samples are not carefully chosen, we may not have consistent performance metrics compared to the real-world deployed scenario, even when the validation/test set presents high-performance measurements.

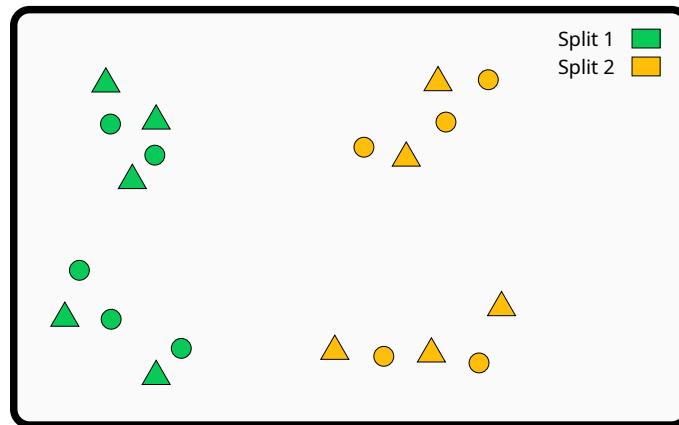


Figure 2: Stratified splits ignoring the feature space.

We try to place similar elements belonging to the same label in different splits to maintain the input and output distribution over all the splits approximately equal. First, we create N splits and we calculate the similarity matrix of samples' features. For each label, we find the pivot sample (the sample with the largest similarity to all other samples of the same label) and the next $N-1$ most similar samples. Then we shuffle these picked samples to guarantee the stochastic behavior and append each sample to each split. After, we remove the picked samples and repeat the process

until there is no sample left in the dataset. This process is summarized in Algorithm 1. The Python code is also available in ¹. In Figure 3, we have an expected scenario of our approach outcome. The distributions of both splits contain more relevant data/information to be learned by the model than the ones in Figure 2.

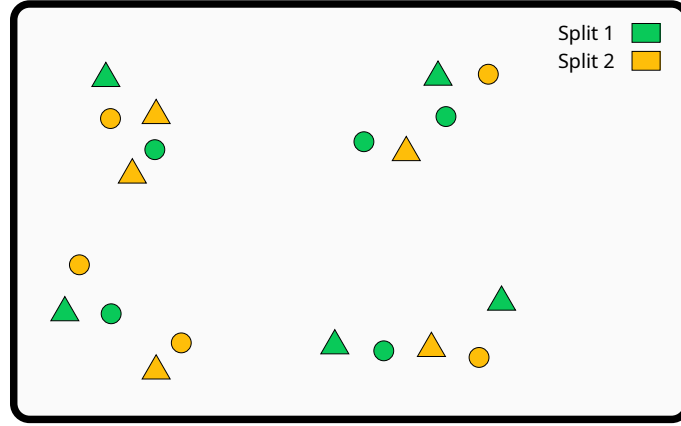


Figure 3: One expected splitting scenario of our approach.

One could argue that this approach has no value since we are not randomly picking samples and the real-world usage of the model the outcomes are random; therefore, we could not assume that it will follow the same distribution. We believe that if this happens remarkably, probably the dataset does not represent the task's population. In this case, more data should be collected before start modeling in order to represent the task to be learned better. Once Learning Systems recognizes patterns in the data, it will probably fail to recognize these patterns when using the real-world model if it does not have similar patterns to learn. For example, an application of object recognition trained with only one color of a specific object may have difficulties to detect colored objects of the same nature and vice-versa. The model would probably perform better if the same ratios of one color and colored objects are presented during the training and validation phase. As this information may not be explicit, one could use the similarities between the objects' colors to drive this behavior. Also, perhaps the real-world application is not generating random outcomes. The samples can be drawn from a distribution that classical statistical probability distributions have difficulties handling or representing graphically. It could give us the notion of distribution that we would like them to have.

Note that any similarity/distance function can be used. We have assessed five similarity functions, namely (i) Cityblock, (ii) Chebyshev, (iii) Euclidean, (iv) Cosine, (v) Correlation to show this. The equations of each function are presented in Eq. 1-5.

$$Chebyshev = \max |u_i - v_i| \quad (1)$$

$$Cityblock = \sum_i |u_i - v_i| \quad (2)$$

$$Euclidean = \sqrt{\sum_i (u_i - v_i)^2} \quad (3)$$

$$Cosine = 1 - \frac{u \cdot v}{\|u\|_2 \|v\|_2} \quad (4)$$

$$Correlation = 1 - \frac{(u - \bar{u}) \cdot (v - \bar{v})}{\|(u - \bar{u})\|_2 \|(v - \bar{v})\|_2} \quad (5)$$

where u and v are two input vectors; i is the dimension index; $|\cdot|$ is the absolute value and $\|\cdot\|_2$ refers to a L2-norm.

2.2. Classifiers

We have assessed different families of classifiers to show the transversality of our proposal. We deployed the following algorithms: (i) a K-Nearest Neighbors (KNN) (Cover & Hart, 1967) (ii) Random Forest (RF) (Breiman,

¹ <https://github.com/felipefariax/sbss>

Algorithm 1 Algorithm for Similarity Based Stratified K-Fold Splitting**Require:** X, Y, k // Normalized Inputs, Outputs, number of splits

```

1: splits = [[]*k] // Create a list with K sublists
2: split_labels = []
3: similarity_matrix = similarity(X, X) // Calculate the similarity between all the samples
4: for label in labels: do
5:   while exist_samples_dataset(X, Y, label) do
6:     //Index of the most similar sample to all samples not already picked of this label. i.e. largest similarity sum
       to other samples of the same label
7:     pivot_idx = get_pivot_sample_idx(X, Y, label)

8:     picked_samples.append(pivot_idx)

9:     //Get the next N-1 most similar samples
10:    for split in N-1: do
11:      closest_idx = get_most_similar_sample(X, Y, label, picked_samples)
12:      picked_samples.append(closest_idx)
13:    end for

14:    shuffle(picked_samples)

15:    // Filling splits
16:    for i = 0...k-1 do
17:      splits[i].append(picked_samples[i])
18:      split_labels.append(label)
19:    end for

20:    X, Y = remove_samples_from_dataset(X, Y, picked_samples)
21:  end while
22: end for

23: // Contains the index of each sample belonging to each split as a “k” by “nb_samples” matrix. Each col has indexes
   of the samples belonging to approximately the same region in the same label regarding the original dataset.
24: return splits, split_labels

```

2001) classifier with 100 trees; (iii) Support Vector Machine (SVM) (Cortes & Vapnik, 1995) and (iv) Multilayer Perceptron (MLP) (Haykin & Haykin, 2009) with 20 hidden neurons without a validation set to early stop the model, 300 epochs and 0.001 as the learning rate. All the other hyperparameters were used with the default values available in the Scikit-Learn (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg et al., 2011) Python library. As we intend to show that SBSS can improve the classifiers’ performance, we have not done much hyperparameter experimentation, meaning that we could achieve even better performances if the hyperparameters were carefully chosen. We have chosen these classifiers due to their different nature bases such as instance-based, decision tree, hyperplane separation, and regression-based methodologies.

2.3. Datasets

We assessed the proposed algorithm in several situations, such as many features and labels, a low number of samples, and dataset imbalance. We have used 22 datasets from UCI (Dua & Graff, 2017) presented in Table 1. We calculated the Imbalance of each dataset by adapting the suggestion in (Romano, 2016) according to Eq. 6, resulting in 0 when the dataset is balanced and 1 otherwise.

Table 1
Datasets used as benchmarks

Dataset	# Features	# Labels	# Samples	Imbalance
balance-scale	4	3	625	0.17
blood-transfusion-service-center (btsc)	4	2	748	0.21
car	6	4	1728	0.40
diabetes	8	2	768	0.07
tic-tac-toe	9	2	958	0.07
ilpd	10	2	583	0.14
vowel	12	11	990	0.00
australian	14	2	690	0.01
climate-model-simulation-crashes (cmssc)	18	2	540	0.58
vehicle	18	4	846	0.00
credit-g	20	2	1000	0.12
wdbc	30	2	569	0.05
ionosphere	34	2	351	0.06
satimage	36	6	6430	0.04
libras move	90	15	360	0.00
hill-valley	100	2	1212	0.00
musk	167	2	6598	0.38
lsvt	310	2	126	0.08
madelon	500	2	2600	0.00
cnae-9	856	9	1080	0.00
dbworld-bodies	4702	2	64	0.01
arcene	10000	2	200	0.01

$$1 - \frac{\sum_{i=1}^k \frac{c_i}{n} \log(\frac{c_i}{n})}{\log(k)} \quad (6)$$

where n is the number of samples; k is the number of labels, and c_i is the number of samples in label i .

2.4. Experiments

We have simulated ten experiments applying SBSS to 10-Fold cross-validation, which we called Similarity-Based Stratified 10-Fold (SBSF), totaling 100 executions – 10 simulations of 10 splits, each split being used as the testing set once. As we have 22 datasets, five similarity measures, and four classifiers, we have done 44,000 SBSF and 10,000 ordinary 10-fold simulations. Although several metrics (Seliya, Khoshgoftaar & Van Hulse, 2009) can be used, we have used the average accuracy of the 10-fold averaged splits to compare our approach against the original stratified 10-fold splitting. We have applied the Wilcoxon Signed-Rank test to assess if our approach significantly increases the classifier's performance with $\alpha = 0.05$. We use nine folds to compose the train set and one fold as the test set.

3. Results and Discussions

In this section, we evaluate and discuss the results of our experiments briefly, comparing the scenarios with and without applying the SBSF split strategy.

We present in Table 2 the training and test set average accuracy. We present average accuracy, and the average standard deviation of 10 evaluations of 10-fold applied to SBSF and original 10-fold stratified splitting inside the parenthesis. We also show the averaged accuracy difference between SBSF and 10-fold. We can see that the Correlation similarity yields the best test accuracy, while the Euclidean had the worst accuracy among the similarity functions used in the SBSF strategy, even though it is still more significant than the 10-fold strategy. Also, the accuracy increase in the test set was more prominent than in the training set, probably indicating a better generalization of the model since

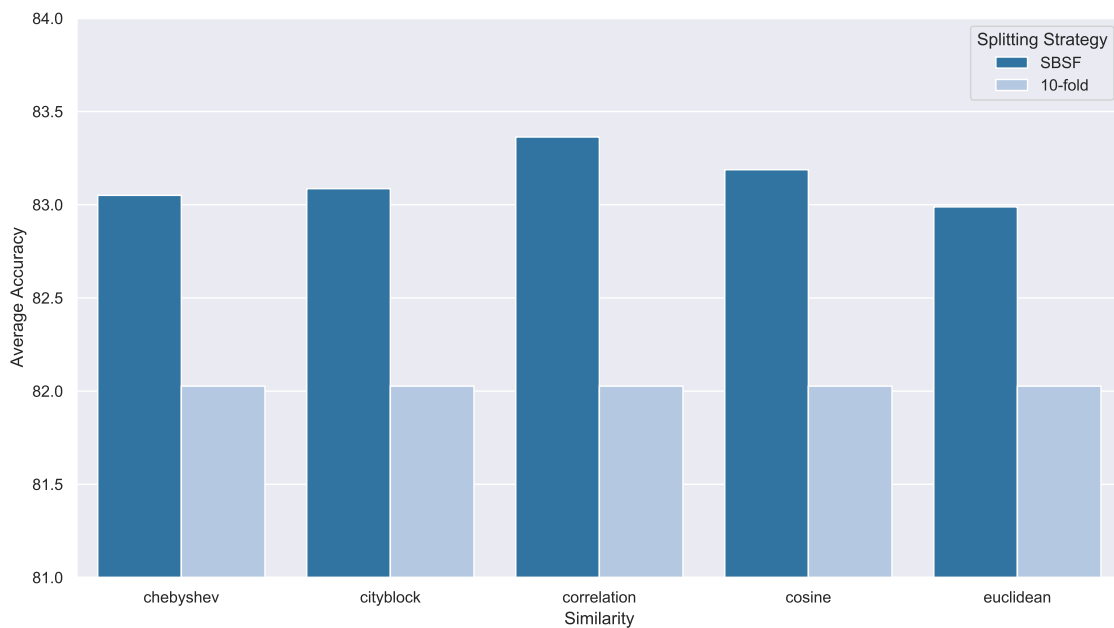
Table 2

Similarity accuracy averaged over all datasets and models.

Similarity	Train			Test		
	10-fold	SBSF	Difference (SBSF-10-fold)	10-fold	SBSF	Difference (SBSF-10-fold)
Chebyshev	90.129 (1.16)	90.492 (1.104)	0.363 (-0.056)	82.027 (4.326)	83.050 (3.758)	1.023 (-0.568)
Cityblock	90.129 (1.16)	90.378 (1.085)	0.249 (-0.075)	82.027 (4.326)	83.086 (3.644)	1.059 (-0.682)
Euclidean	90.129 (1.16)	90.385 (1.104)	0.256 (-0.056)	82.027 (4.326)	82.988 (3.678)	0.961 (-0.648)
Cosine	90.129 (1.16)	90.380 (1.102)	0.251 (-0.058)	82.027 (4.326)	83.188 (3.440)	1.161 (-0.886)
Correlation	90.129 (1.16)	90.536 (1.101)	0.407 (-0.059)	82.027 (4.326)	83.363 (3.574)	1.336 (-0.752)
Average	90.129 (1.16)	90.434 (1.099)	0.305 (-0.061)	82.027 (4.326)	83.135 (3.619)	1.108 (-0.707)

it tends not to have high-density regions that would give more importance due to a bad data splitting. The standard deviation was also reduced in SBSF.

The averaged accuracy for all datasets and models for each similarity compared to the 10-fold strategy is presented in Figure 4. It is easy to notice that the Correlation presented the most remarkable performance among all similarities.

**Figure 4:** Averaged accuracy over datasets and models for each SBSF similarity compared to 10-fold strategy.

The boxplots comparing absolute accuracy for each similarity and model, regarding all datasets are shown in Figure 5. The median values in SBSF with Correlation similarity was always more significant than the 10-fold strategy for every model. The same occurs with other similarity/model scenarios.

In Figure 6, the boxplots depict the differences between the accuracies of the SBSF and 10-fold. In general, the similarities had few negative differences, with some of them treated as outliers.

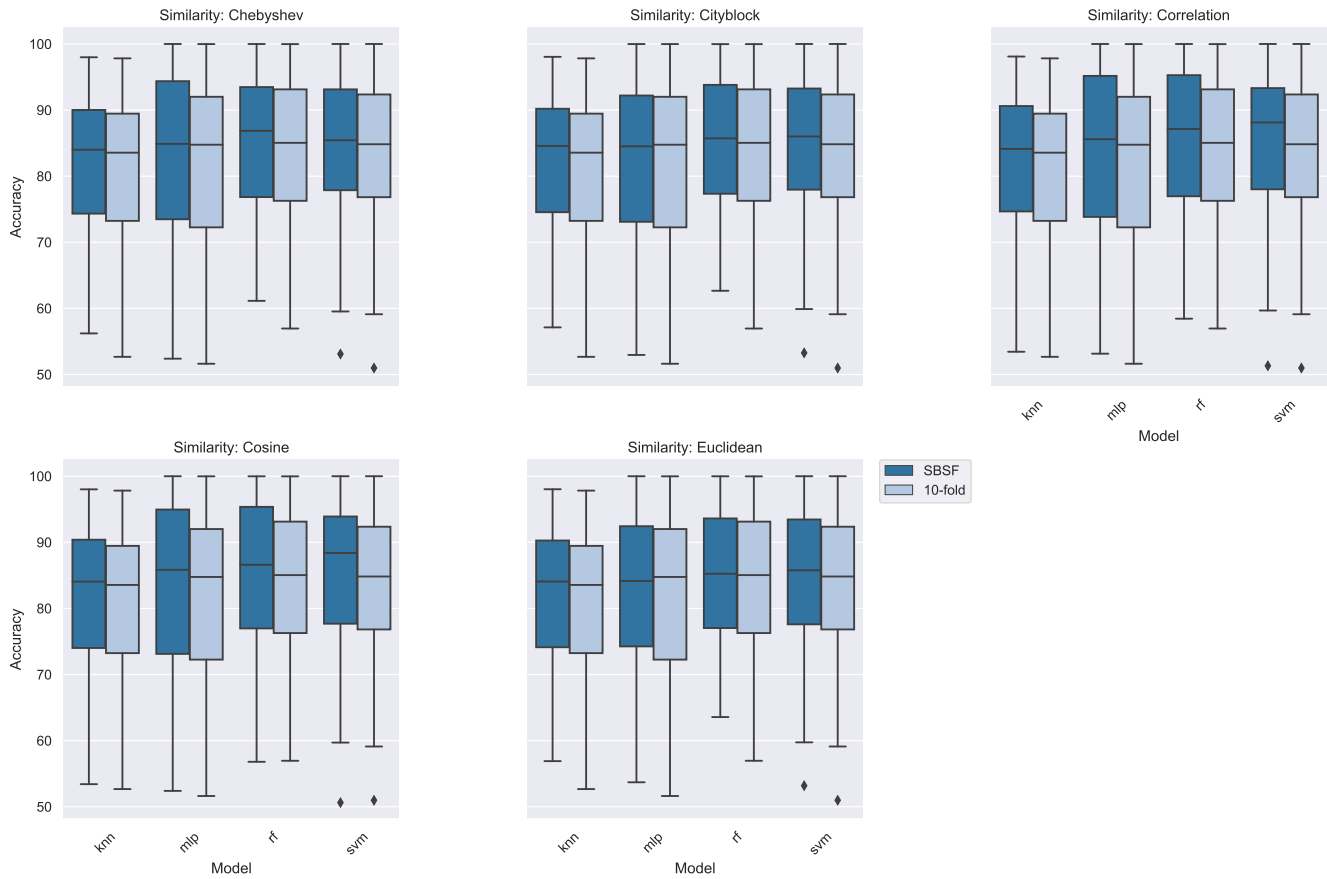


Figure 5: Boxplots with accuracies for each similarity and model of SBSF compared to 10-fold strategy.

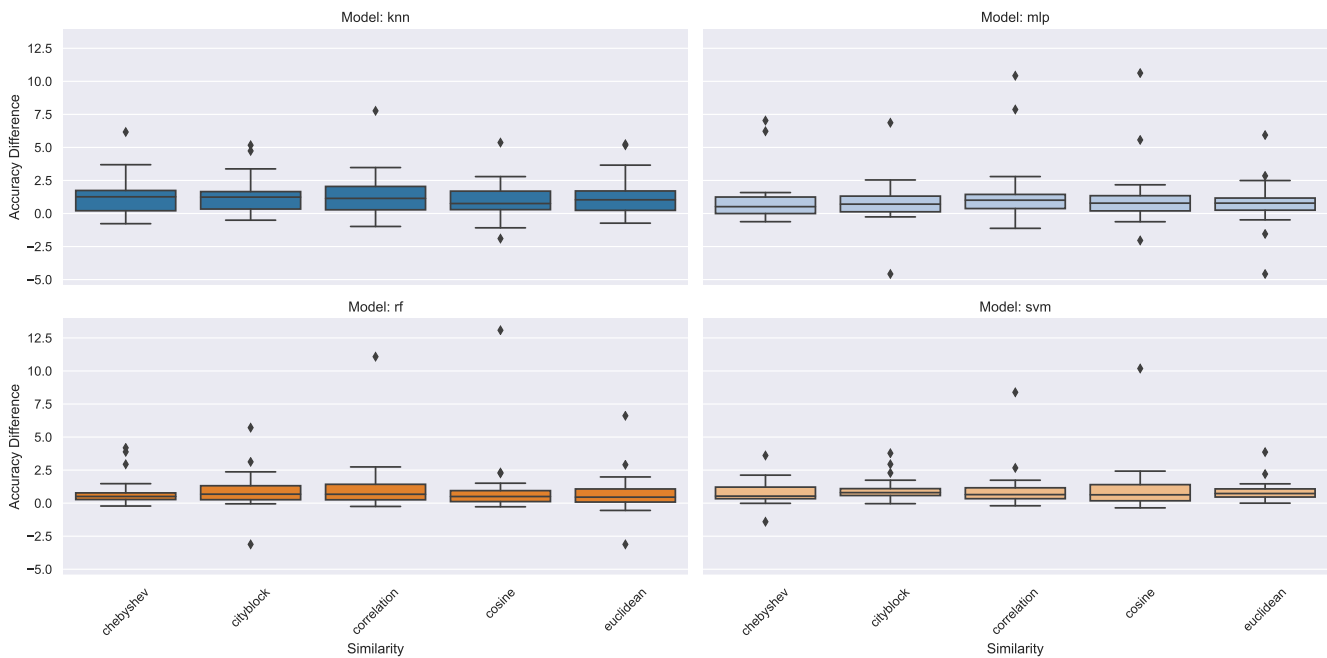


Figure 6: Boxplots with accuracies differences SBSF - 10-fold.

Table 3

Averaged difference of accuracy between train and test sets for each splitting strategy.

Strategy	Difference Train-Test
SBSF + Chebyshev	7.442
SBSF + Cityblock	7.292
SBSF + Correlation	7.173
SBSF + Cosine	7.192
SBSF + Euclidean	7.397
10-fold	8.103

Table 4

Number of losses, ties and wins regarding the models and similarities of the SBSF over 10-fold splitting.

Model	Chebyshev			Cityblock			Euclidean			Cosine			Correlation		
	loss	tie	win	loss	tie	win	loss	tie	win	loss	tie	win	loss	tie	win
KNN	2	4	16	0	3	19	0	6	16	3	2	17	1	5	16
MLP	3	4	15	1	7	14	2	1	19	3	3	16	1	4	17
RF	0	5	17	0	4	18	0	6	16	1	8	13	0	5	17
SVM	1	5	16	0	3	19	0	5	17	1	5	16	0	6	16
Total	6	18	64	1	17	70	2	18	68	8	18	62	2	20	66
%	6.82	20.45	72.73	1.14	19.32	79.55	2.27	20.45	77.27	9.09	20.45	70.45	2.27	22.73	75

In Table 3, we can see that the average difference between the accuracy in the training - test set for SBSF is less than the 10-fold. It may indicate that the training and test set distributions are closer in the SBSF strategy than ordinary 10-fold. It can also indicate that SBSF intrinsically regularizes the training as the more significant this difference, the greater the chance of overfitting.

In Table 4, the scores of SBSF vs. 10-fold are presented for each model and each similarity. The MLP is probably the most sensible technique to its hyperparameters between the assessed models. Specifically the learning rate, number of neurons, and number of epochs, which we have fixed as 0.001, 20, and 300, respectively. As we have assessed this simple architecture regardless of the dataset, which has several numbers of samples/features, this probably led to the most significant number of losses without any hyperparameter exploration. The similarity Cityblock appears to be the best similarity since it got the highest number of wins (79.55%). However, due to the highest accuracy of Correlation similarity shown in Table 2, we have chosen the Correlation similarity with 75% of wins, 22.73% of ties, and only 2.27% of losses with an average increase in test set accuracy of 1.336% for the next analysis. The Correlation similarity probably had better performances because the vectors' mean of u and v are subtracted, which led to smaller magnitudes of the vectors, facilitating the calculation of L2-norms better-extracting similarities information among the samples.

In the following subsections, we present a detailed comparative analysis of each model using the Correlation similarity with SBSF and ordinary 10-fold.

3.1. K-Nearest Neighbors

The train and test accuracies with their respective differences (SBSF-10-fold) of SBSF with Correlation similarity in KNN, presented in Table 5, increased the test accuracy in 17 datasets, remaining the same in 4 and losing in 1 case when compared with 10-fold splitting according to Wilcoxon statistical test. An increase in the training set did not necessarily accompany the test set's increase with the same magnitude. For example, in the vehicle dataset, the test accuracy increased 2.071% while the training stayed almost the same. The standard deviation also decreased in SBSF. The average difference of train-test accuracies of SBSF and 10-fold is $1 - (87.028 - 81.222) / (86.322 - 79.791) = 11.1\%$ lower than the ordinary 10-fold.

3.2. Multi-layer Perceptron

Table 6 shows the accuracies of train and test set with their differences of SBSF with Correlation similarity in MLP. The test accuracy increased in 17 cases, persisted the same in 4, and lost in 1. As in KNN, the difference in

Table 5

Average accuracy for KNN with SBSF using Correlation similarity. Highlighted values are higher according to Wilcoxon Significance test.

Dataset	Train			Test		
	SBSF	10fold	Difference	SBSF	10fold	Difference
australian	88.851 (0.447)	87.902 (0.614)	0.949 (-0.167)	86.397 (2.603)	84.536 (4.562)	1.861 (-1.959)
arcene	92.38 (1.044)	92.178 (1.025)	0.202 (0.019)	88.526 (6.447)	85.05 (8.313)	3.476 (-1.866)
balance-scale	89.789 (0.864)	86.99 (0.962)	2.799 (-0.098)	85.8 (2.952)	82.624 (4.115)	3.176 (-1.163)
btsc	81.862 (0.774)	81.827 (0.829)	0.035 (-0.055)	77.041 (2.751)	76.912 (3.787)	0.129 (-1.036)
car	98.997 (0.211)	98.914 (0.257)	0.083 (-0.046)	96.661 (1.292)	95.672 (1.638)	0.989 (-0.346)
cmssc	94.797 (0.321)	93.889 (0.419)	0.908 (-0.098)	94.113 (1.771)	92.815 (2.277)	1.298 (-0.506)
cnae-9	91.249 (0.746)	91.118 (0.718)	0.131 (0.028)	84.759 (3.019)	84.185 (2.875)	0.574 (0.144)
credit-g	81.686 (0.58)	81.653 (0.682)	0.033 (-0.102)	74.34 (3.131)	74.04 (3.914)	1.3 (-0.783)
dbworld-bodies	66.022 (2.165)	60.118 (2.712)	5.904 (-0.547)	62.6 (6.114)	54.833 (5.858)	7.767 (0.256)
diabetes	82.386 (0.655)	82.079 (0.7)	0.307 (-0.045)	75.658 (3.738)	73.841 (4.877)	1.817 (-1.139)
hill-valley	72.873 (0.679)	72.833 (0.763)	0.04 (-0.084)	53.442 (3.784)	52.673 (4.487)	0.769 (-0.703)
ilpd	79.23 (1.005)	78.237 (0.931)	0.993 (0.074)	67.123 (4.262)	65.147 (4.379)	1.976 (-0.117)
ionosphere	87.245 (0.582)	87.809 (0.748)	-0.564 (-0.166)	84.5 (4.427)	85.211 (5.436)	-0.711 (-1.009)
libras move	87.093 (1.046)	85.006 (1.199)	2.087 (-0.153)	77.967 (6.01)	75.083 (7.749)	2.884 (-1.739)
lsvt	90.528 (1.877)	89.021 (1.491)	1.507 (0.386)	82.667 (10.143)	83.647 (10.737)	-0.98 (-0.594)
madelon	73.864 (0.656)	73.963 (0.592)	-0.099 (0.064)	56.865 (3.122)	57.135 (3.106)	-0.27 (0.016)
musk	98.865 (0.069)	98.842 (0.083)	0.023 (-0.014)	98.102 (0.42)	97.825 (0.593)	0.277 (-0.173)
satimage	94.055 (0.141)	93.877 (0.159)	0.178 (-0.018)	91.309 (0.877)	90.88 (1.08)	0.429 (-0.203)
tic-tac-toe	85.021 (0.67)	84.932 (0.688)	0.089 (-0.018)	83.758 (3.596)	83.466 (3.344)	0.292 (0.252)
vehicle	82.02 (0.792)	82.025 (0.753)	-0.005 (0.039)	72.183 (3.848)	69.788 (4.101)	2.395 (-0.253)
vowel	98.053 (0.259)	98.038 (0.306)	0.015 (-0.047)	96.263 (1.834)	94.192 (2.459)	2.071 (-0.625)
wdbc	97.742 (0.284)	97.825 (0.294)	-0.083 (-0.01)	96.821 (2.141)	96.854 (2.452)	-0.033 (-0.311)
Average	87.028 (0.721)	86.322 (0.769)	0.706 (-0.048)	81.222 (3.558)	79.791 (4.188)	1.431 (-0.63)
Losses/Ties/Wins	2L/8T/12W			1L/5T/16W		

test and training accuracy was not proportional. For example, in dbworld-bodies dataset, the test accuracy increased 10.424% while the training accuracy only increased 1.423%. The standard deviation of the accuracies also had a large decrease. As we have not used a validation set to stop the training or choose the best MLP model, we can realize that the difference between accuracies obtained in the training and testing sets of 10-fold is higher than in SBSF. It is probably a sign of over-fitting in 10-fold splitting since the training error is much lower than the test error. We believe that, as we have better sampled the dataset through SBSF, a better knowledge extraction was performed. Thus, SBSF can act as an intrinsic regularizer difficulting the over-fitting, as the train-test accuracy difference of SBSF in this experiment is, on average, $1 - (85.958 - 81.8)/(85.53 - 80.253) = 21.205\%$ lower than the ordinary 10-fold.

3.3. Support Vector Machine

Regarding SVM in Table 7, the usage of SBSF also increased the test set accuracy in 16 datasets, remaining the same in 6 and with no losses compared with 10-fold splitting. One can observe an increase in the test set performance without not necessarily having an increase in the training set, as it was the case with previous classifiers. For example, in the arcene dataset, the test accuracy increased 1.734% while presenting a statistically significant decrease in the training set, which was 0.283 smaller than with 10-fold. The SBSF strategy performed worse in 4 cases regarding the training set (arcene, car, credit-g, and ionosphere) while showing statistically significant increases in the same datasets at the test set, except in ionosphere. The train-test accuracy difference of SBSF in this experiment is, on average, $1 - (89.458 - 84.38)/(89.029 - 83.259) = 11.993\%$ lower than the ordinary 10-fold.

3.4. Random Forest

Assessing the impact of SBSF to RF, we can see from Table 8 that the test set accuracy increased in 17 datasets, remaining the same in 5 and with no losses regarding the 10-fold splitting. The corresponding increase behavior in the training and testing sets of previous classifiers also applies to RF. The train-test accuracy difference of SBSF in this

Table 6

Average accuracy for MLP with SBSF using Correlation similarity. Highlighted values are higher according to Wilcoxon Significance test.

Dataset	Train			Test		
	SBSF	10fold	Difference	SBSF	10fold	Difference
australian	89.98 (0.932)	90.188 (1.053)	-0.208 (-0.121)	88.0 (2.292)	86.652 (4.127)	1.348 (-1.835)
arcene	85.175 (22.52)	84.122 (22.788)	1.053 (-0.268)	74.895 (18.009)	72.1 (16.379)	2.795 (1.63)
balance-scale	97.409 (0.896)	97.031 (1.053)	0.378 (-0.157)	96.5 (2.276)	95.406 (2.682)	1.094 (-0.406)
btsc	80.763 (0.707)	79.979 (0.769)	0.784 (-0.062)	80.23 (2.273)	78.943 (3.204)	1.287 (-0.931)
car	97.394 (0.92)	97.222 (1.021)	0.172 (-0.101)	96.503 (1.537)	96.134 (1.83)	0.369 (-0.293)
cmssc	99.73 (0.345)	99.265 (0.643)	0.465 (-0.298)	96.132 (2.185)	94.722 (2.718)	1.41 (-0.533)
cnae-9	99.897 (0.045)	99.889 (0.046)	0.008 (-0.001)	92.287 (2.158)	91.898 (2.279)	0.389 (-0.121)
credit-g	87.772 (1.943)	87.996 (1.846)	-0.224 (0.097)	73.66 (3.781)	72.74 (4.031)	0.92 (-0.25)
dbworld-bodies	100.0 (0.0)	98.577 (0.695)	1.423 (-0.695)	99.4 (1.897)	88.976 (12.706)	10.424 (-10.809)
diabetes	79.863 (0.736)	79.674 (0.964)	0.189 (-0.228)	77.276 (3.709)	76.886 (4.195)	0.39 (-0.486)
hill-valley	63.855 (4.764)	64.449 (5.4)	-0.594 (-0.636)	62.725 (5.963)	63.848 (7.206)	-1.123 (-1.243)
ilpd	75.774 (1.069)	74.704 (0.928)	1.07 (0.141)	73.211 (4.147)	71.496 (4.169)	1.715 (-0.022)
ionosphere	99.15 (0.387)	99.161 (0.374)	-0.011 (0.013)	91.912 (4.341)	92.048 (4.345)	-0.136 (-0.004)
libras move	66.163 (11.839)	60.639 (10.532)	5.524 (1.307)	60.533 (12.558)	52.667 (10.987)	7.866 (1.571)
lsvt	99.972 (0.062)	99.991 (0.028)	-0.019 (0.034)	86.917 (8.747)	85.462 (9.689)	1.455 (-0.942)
madelon	59.112 (10.967)	59.407 (11.629)	-0.295 (-0.662)	53.546 (4.572)	53.023 (4.181)	0.523 (0.391)
musk	99.997 (0.009)	99.997 (0.008)	0.0 (0.001)	99.997 (0.01)	99.986 (0.03)	0.011 (-0.02)
satimage	85.368 (1.583)	85.015 (1.53)	0.353 (0.053)	84.611 (1.787)	84.062 (1.64)	0.549 (0.147)
tic-tac-toe	91.989 (2.371)	92.149 (2.525)	-0.16 (-0.154)	86.579 (3.627)	86.285 (4.408)	0.294 (-0.781)
vehicle	77.016 (2.689)	76.824 (2.159)	0.192 (0.53)	74.39 (4.133)	73.099 (4.117)	1.291 (0.016)
vowel	55.879 (7.081)	56.663 (7.298)	-0.784 (-0.217)	53.152 (7.265)	51.626 (8.279)	1.526 (-1.014)
wdbc	98.813 (0.302)	98.719 (0.32)	0.094 (-0.018)	97.143 (1.98)	97.505 (1.993)	-0.362 (-0.013)
Average	85.958 (3.28)	85.53 (3.346)	0.428 (-0.066)	81.8 (4.511)	80.253 (5.236)	1.547 (-0.725)
Losses/Ties/Wins	1L/12T/9W			1L/4T/17W		

experiment is, on average, $1 - (99.701 - 86.051)/(99.637 - 84.803) = 7.982\%$ lower than the ordinary 10-fold.

The test accuracy of each classifier applied to each dataset using SBSF with Correlation similarity is summarized in Table 9. The RF presented better results in 8 of 22 cases. Since Decision Trees are used internally to divide spaces, and SBSF acts improving these spatial representations, most improvements were achieved.

Considering MLP, SVM, KNN and RF techniques and all the similarities scores, SBSF significantly increased the test accuracy in 330 cases (75%), remained statistically similar in 91 (20.68%), and decreased in 19 cases (4.32%).

3.5. Generalization Gap

The generalization gap can be understood as the ability of the model to make good predictions on unseen data. It can be measured as the difference between the Train and Test performances Jiang, Krishnan, Mobahi & Bengio (2018). We present the generalization gap for each model, dataset, and splitting strategy are displayed in Table 10. Bold values indicate smaller differences.

Smaller differences mean that the Train and Test accuracy are closer. When the gap is larger, it might be an indication of under/over-fitting of the model. The SBSF approach consistently delivers smaller differences.

4. Conclusion

We believe that every model, including research and/or industrial applications, could benefit from this strategy to prepare the data to be learned, generating better models with increased real-world usage performance. After the model is trained, the real-world data to be presented should follow approximately the same distribution of the prepared training set. If this happens, probably the model could increase its performance if it uses the SBSS.

The SBSF showed statistically significant performance increases in the test set in 75% of the cases. It also was able to decrease the generalization gap. We believe that the a-posteriori distribution of the input space regions can be

Table 7

Average accuracy for SVM with SBSF using Correlation similarity. Highlighted values are higher according to Wilcoxon Significance test.

Dataset	Train			Test		
	SBSF	10fold	Difference	SBSF	10fold	Difference
australian	87.843 (0.347)	87.626 (0.567)	0.217 (-0.22)	86.368 (2.152)	85.449 (3.993)	0.919 (-1.841)
arcene	92.573 (0.895)	92.856 (1.09)	-0.283 (-0.195)	78.684 (6.468)	76.95 (8.617)	1.734 (-2.149)
balance-scale	92.689 (0.253)	91.561 (0.354)	1.128 (-0.101)	92.133 (1.141)	90.464 (1.546)	1.669 (-0.405)
btsc	78.03 (0.206)	77.594 (0.405)	0.436 (-0.199)	77.5 (0.872)	76.781 (1.827)	0.719 (-0.955)
car	97.973 (0.202)	98.13 (0.16)	-0.157 (0.042)	96.778 (1.173)	96.846 (1.312)	-0.068 (-0.139)
cmssc	97.57 (0.298)	96.864 (0.379)	0.706 (-0.081)	93.566 (1.271)	92.722 (1.723)	0.844 (-0.452)
cnae-9	99.313 (0.126)	99.298 (0.146)	0.015 (-0.02)	91.852 (2.351)	91.389 (2.624)	0.463 (-0.273)
credit-g	82.409 (0.504)	82.567 (0.541)	-0.158 (-0.037)	77.11 (2.853)	76.13 (3.204)	0.98 (-0.351)
dbworld-bodies	100.0 (0.0)	98.577 (0.695)	1.423 (-0.695)	92.6 (9.962)	84.214 (15.234)	8.386 (-5.272)
diabetes	80.553 (0.521)	80.233 (0.555)	0.32 (-0.034)	78.184 (3.807)	76.979 (4.407)	1.205 (-0.6)
hill-valley	53.554 (0.403)	53.295 (0.486)	0.259 (-0.083)	51.317 (2.686)	50.982 (3.326)	0.335 (-0.64)
ilpd	71.93 (0.0)	71.355 (0.084)	0.575 (-0.084)	71.93 (0.0)	71.356 (0.762)	0.574 (-0.762)
ionosphere	95.918 (0.417)	96.075 (0.466)	-0.157 (-0.049)	93.588 (2.622)	93.394 (4.226)	0.194 (-1.604)
libras move	91.511 (1.284)	89.917 (0.84)	1.594 (0.444)	82.867 (5.571)	81.306 (5.883)	1.561 (-0.312)
lsvt	88.796 (1.07)	87.187 (1.573)	1.609 (-0.503)	83.0 (9.131)	82.558 (9.329)	0.442 (-0.198)
madelon	95.768 (0.207)	95.767 (0.249)	0.001 (-0.042)	59.681 (2.677)	59.115 (2.859)	0.566 (-0.182)
musk	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)
satimage	91.148 (0.127)	90.951 (0.149)	0.197 (-0.022)	90.172 (0.938)	89.879 (1.083)	0.293 (-0.145)
tic-tac-toe	92.91 (0.464)	92.994 (0.388)	-0.084 (0.076)	89.884 (2.31)	89.54 (2.963)	0.344 (-0.653)
vehicle	83.381 (0.671)	81.655 (0.757)	1.726 (-0.086)	77.963 (3.11)	75.297 (4.098)	2.666 (-0.988)
vowel	95.807 (0.406)	95.872 (0.417)	-0.065 (-0.011)	93.717 (2.015)	92.687 (2.271)	1.03 (-0.256)
wdbc	98.395 (0.222)	98.258 (0.23)	0.137 (-0.008)	97.464 (2.04)	97.663 (1.97)	-0.199 (0.07)
Average	89.458 (0.392)	89.029 (0.479)	0.429 (-0.087)	84.38 (2.961)	83.259 (3.784)	1.121 (-0.823)
Losses/Ties/Wins	4L/5T/13W			0L/6T/16W		

better explored, and this information can be incorporated into the model through a careful data splitting process used during the training phase. It is a low-cost strategy to increase models' performance by only changing how the training data is presented. We expect this approach to deploying models in the academy and industry scenarios with better performances.

As future works, we intend to evaluate SBSS in other classifiers, including investigating their hyperparameters. We also plan to assess the proposal with different splitting strategies, such as Holdout, with and without stratification. An analysis of regression models with SBSS is also relevant. Other similarities functions may also benefit the SBSS strategy. Also, we intend to use the average of similarities instead of the sum in the algorithm that may benefit the performance.

References

- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning* volume 4. springer. doi:10.1117/1.2819119. arXiv:0-387-31073-8.
- Breiman, L. (2001). Random forests. *Machine learning*, 45, 5–32.
- Breiman, L., & Spector, P. (1992). Submodel Selection and Evaluation in Regression. The X-Random Case. *International Statistical Review / Revue Internationale de Statistique*, 60, 291. doi:10.2307/1403680.
- Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20, 273–297. URL: <http://link.springer.com/10.1007/BF00994018>. doi:10.1007/BF00994018.
- Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13, 21–27.
- Dua, D., & Graff, C. (2017). UCI machine learning repository. URL: <http://archive.ics.uci.edu/ml>.
- Haykin, S., & Haykin, S. (2009). *Neural Networks and Learning Machines*. Number v. 10 in Neural networks and learning machines. Prentice Hall.
- Japkowicz, N., & Shah, M. (2011). *Evaluating learning algorithms: a classification perspective*. Cambridge University Press.
- Jiang, Y., Krishnan, D., Mobahi, H., & Bengio, S. (2018). Predicting the generalization gap in deep networks with margin distributions. *arXiv preprint arXiv:1810.00113*, .

Table 8

Average accuracy for RF with SBSF using Correlation similarity. Highlighted values are higher according to Wilcoxon Significance test.

Dataset	Train			Test		
	SBSF	10fold	Difference	SBSF	10fold	Difference
australian	100.0 (0.0)	99.998 (0.005)	0.002 (-0.005)	88.574 (2.773)	87.101 (3.817)	1.473 (-1.044)
arcene	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	85.263 (6.047)	83.0 (8.066)	2.263 (-2.019)
balance-scale	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	85.7 (2.874)	82.96 (3.31)	2.74 (-0.436)
btsc	93.435 (0.228)	93.434 (0.321)	0.001 (-0.093)	74.743 (3.332)	73.959 (4.652)	0.784 (-1.32)
car	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	98.737 (0.835)	98.472 (1.033)	0.265 (-0.198)
cmssc	99.996 (0.013)	100.0 (0.0)	-0.004 (0.013)	93.283 (1.063)	92.574 (1.584)	0.709 (-0.521)
cnae-9	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	93.167 (2.235)	92.611 (2.221)	0.556 (0.014)
credit-g	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	76.96 (3.019)	76.25 (3.845)	0.71 (-0.826)
dbworld-bodies	100.0 (0.0)	98.577 (0.695)	1.423 (-0.695)	97.2 (5.583)	86.119 (15.418)	11.081 (-9.835)
diabetes	99.999 (0.005)	100.0 (0.0)	-0.001 (0.005)	76.987 (4.363)	76.352 (4.249)	0.635 (0.114)
hill-valley	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	58.442 (4.273)	56.955 (3.93)	1.487 (0.343)
ilpd	100.0 (0.0)	99.998 (0.006)	0.002 (-0.006)	72.246 (3.949)	70.509 (4.883)	1.737 (-0.934)
ionosphere	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	93.059 (3.5)	93.309 (4.202)	-0.25 (-0.702)
libras_move	99.996 (0.012)	100.0 (0.0)	-0.004 (0.012)	84.367 (6.373)	83.333 (6.231)	1.034 (0.142)
lsvt	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	85.25 (8.986)	83.968 (9.292)	1.282 (-0.306)
madelon	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	71.492 (2.538)	71.615 (2.929)	-0.123 (-0.391)
musk	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	99.994 (0.019)	99.98 (0.046)	0.014 (-0.027)
satimage	99.999 (0.001)	99.999 (0.002)	0.0 (-0.001)	92.092 (0.79)	91.712 (1.068)	0.38 (-0.278)
tic-tac-toe	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	95.947 (2.034)	95.919 (1.967)	0.028 (0.067)
vehicle	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	75.5 (3.478)	75.251 (3.452)	0.249 (0.026)
vowel	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	97.99 (1.476)	97.515 (1.557)	0.475 (-0.081)
wdbc	100.0 (0.0)	100.0 (0.0)	0.0 (0.0)	96.125 (2.335)	96.204 (2.303)	-0.079 (0.032)
Average	99.701 (0.012)	99.637 (0.047)	0.064 (-0.035)	86.051 (3.267)	84.803 (4.093)	1.248 (-0.826)
Losses/Ties/Wins	0L/21T/1W			0L/5T/17W		

Jung, Y. (2018). Multiple predicting K-fold cross-validation for model selection. *Journal of Nonparametric Statistics*, 30, 197–215. doi:10.1080/10485252.2017.1404598.

Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai* (pp. 1137–1145). Montreal, Canada volume 14.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12, 2825–2830.

Romano, S. (2016). A general measure of dataset imbalance. <https://stats.stackexchange.com/questions/239973/a-general-measure-of-data-set-imbalance>. Accessed: 2020-07-26.

Russell, S., & Norvig, P. (2020). *Artificial intelligence: a modern approach*. Pearson series in artificial intelligence (fourth edition ed.). Hoboken: Pearson.

Seliya, N., Khoshgoftaar, T. M., & Van Hulse, J. (2009). A study on the relationships of classifier performance metrics. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, (pp. 59–66). doi:10.1109/ICTAI.2009.25.

Snee, R. D. (1977). Validation of regression models: methods and examples. *Technometrics*, 19, 415–428.

Wong, T.-T. (2015). Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. *Pattern Recognition*, 48, 2839–2846.

Wong, T. T., & Yang, N. Y. (2017). Dependency Analysis of Accuracy Estimates in k-Fold Cross Validation. *IEEE Transactions on Knowledge and Data Engineering*, 29, 2417–2427. doi:10.1109/TKDE.2017.2740926.

Wong, T.-T., & Yeh, P.-Y. (2019). Reliable Accuracy Estimates from k-fold Cross Validation. *IEEE Transactions on Knowledge and Data Engineering*, 4347, 1–1. doi:10.1109/tkde.2019.2912815.

Table 9

Average test accuracy for all classifiers using Correlation similarity with SBSF. Best absolute accuracy for each dataset are highlighted.

Dataset	RF	MLP	KNN	SVM
australian	88.574 (2.773)	88.0 (2.292)	86.397 (2.603)	86.368 (2.152)
arcene	85.263 (6.047)	74.895 (18.009)	88.526 (6.447)	78.684 (6.468)
balance-scale	85.7 (2.874)	96.5 (2.276)	85.8 (2.952)	92.133 (1.141)
btsc	74.743 (3.332)	80.23 (2.273)	77.041 (2.751)	77.5 (0.872)
car	98.737 (0.835)	96.503 (1.537)	96.661 (1.292)	96.778 (1.173)
cmssc	93.283 (1.063)	96.132 (2.185)	94.113 (1.771)	93.566 (1.271)
cnae-9	93.167 (2.235)	92.287 (2.158)	84.759 (3.019)	91.852 (2.351)
credit-g	76.96 (3.019)	73.66 (3.781)	74.34 (3.131)	77.11 (2.853)
dbworld-bodies	97.2 (5.583)	99.4 (1.897)	62.6 (6.114)	92.6 (9.962)
diabetes	76.987 (4.363)	77.276 (3.709)	75.658 (3.738)	78.184 (3.807)
hill-valley	58.442 (4.273)	62.725 (5.963)	53.442 (3.784)	51.317 (2.686)
ilpd	72.246 (3.949)	73.211 (4.147)	67.123 (4.262)	71.93 (0.0)
ionosphere	93.059 (3.5)	91.912 (4.341)	84.5 (4.427)	93.588 (2.622)
Libras move	84.367 (6.373)	60.533 (12.558)	77.967 (6.01)	82.867 (5.571)
lsvt	85.25 (8.986)	86.917 (8.747)	82.667 (10.143)	83.0 (9.131)
madelon	71.492 (2.538)	53.546 (4.572)	56.865 (3.122)	59.681 (2.677)
musk	99.994 (0.019)	99.997 (0.01)	98.102 (0.42)	100.0 (0.0)
satimage	92.092 (0.79)	84.611 (1.787)	91.309 (0.877)	90.172 (0.938)
tic-tac-toe	95.947 (2.034)	86.579 (3.627)	83.758 (3.596)	89.884 (2.31)
vehicle	75.5 (3.478)	74.39 (4.133)	72.183 (3.848)	77.963 (3.11)
vowel	97.99 (1.476)	53.152 (7.265)	96.263 (1.834)	93.717 (2.015)
wdbc	96.125 (2.335)	97.143 (1.98)	96.821 (2.141)	97.464 (2.04)
Average Score	86.051 (3.267) 8	81.8 (4.511) 7	81.222 (3.558) 1	84.38 (2.961) 6

Felipe C. Farias received the B.Sc. in Computer Engineering in 2014, in this occasion he awarded the Academic Laurea of Engineering, the M.Sc. degree in Computer Intelligence in 2016, both from University of Pernambuco (UPE). Ph.D. student of Machine Learning at Federal University of Pernambuco (UFPE). He is currently a Professor at Federal Institute of Education, Science and Technology of Pernambuco (IFPE). His research interests include Neural Networks, Deep Learning, Machine Learning, Network Science. More information at <https://scholar.google.com/citations?hl=pt-BR&user=zwRHO14AAAAJ> <http://lattes.cnpq.br/4598958786544738>

Teresa B. Ludermit received the Ph.D. degree in Artificial Neural Networks in 1990 from Imperial College, University of London, UK. From 1991 to 1992, she was a lecturer at Kings College London. She joined the Center of Informatics at Federal University of Pernambuco, Brazil, in September 1992, where she is currently a Professor and head of the Computational Intelligence Group. She has published over 300 articles in scientific journals and conferences, three books in Neural Networks (in Portuguese) and organized three of the Brazilian Symposium on Neural Networks. Her research interests include weightless Neural Networks, hybrid neural systems, machine learning and applications of Neural Networks. She is an IEEE senior member and a Research fellow level 1A (the higher level) of the National Research Council of Brazil (CNPq). More information at <https://scholar.google.com.br/citations?hl=pt-BR&user=w-tKJOwAAAAJ> <https://orcid.org/0000-0002-8980-6742> <http://lattes.cnpq.br/6321179168854922>

Prof. Carmelo J. A. Bastos-Filho was born in Recife, Brazil, in 1978. He received the B.Sc. in electronics engineering and the M.Sc. and Ph.D. degrees in electrical engineering from Federal University of Pernambuco (UFPE) in 2000, 2003, and 2005, respectively. In 2006, he received the best Brazilian thesis award in electrical engineering. His interests are related to: the development of protocols and algorithms to manage and to design optical communication networks, development of novel swarm intelligence algorithms and the deployment of swarm intelligence for complex optimization and clustering problems, development and application of multiobjective optimization and many-objective optimization for real-world problems, Development and application of soft deep learning techniques, development of solutions using network sciences for big data, applications in robotics and applications in biomedical problems. He is currently an Associate Professor at the Polytechnic School of the University of Pernambuco. He was the scientist-in-chief of the technological Park for Electronics and Industry 4.0 of Pernambuco. He is currently the director for science parks and graduate studies of Pernambuco. He is an IEEE senior member and a Research fellow level 1D of the National Research Council of Brazil (CNPq). He published over 230 full papers in journals and conferences and advised over 50 PhD and MSc candidates. More information at <http://scholar.google.com/citations?user=t3A96agAAAAJ&hl=en>

Table 10

Absolute difference between Train and Test accuracies. Bold values indicate smaller differences.

Dataset	KNN		MLP		SVM		RF	
	SBSF	10-fold	SBSS	10-fold	SBSS	10-fold	SBSS	10-fold
australian	2.454	3.366	1.98	3.536	1.475	2.177	11.426	12.897
arcene	3.854	7.128	10.28	12.022	13.889	15.906	14.737	17
balance-scale	3.989	4.366	0.909	1.625	0.556	1.097	14.3	17.04
btsc	4.821	4.915	0.533	1.036	0.53	0.813	18.692	19.475
car	2.336	3.242	0.891	1.088	1.195	1.284	1.263	1.528
cmssc	0.684	1.074	3.598	4.543	4.004	4.142	6.713	7.426
cnae-9	6.49	6.933	7.61	7.991	7.461	7.909	6.833	7.389
credit-g	7.346	7.613	14.112	15.256	5.299	6.437	23.04	23.75
dbworld-bodies	3.422	5.285	0.6	9.601	7.4	14.363	2.8	12.458
diabetes	6.728	8.238	2.587	2.788	2.369	3.254	23.012	23.648
hill-valley	19.431	20.16	1.13	0.601	2.237	2.313	41.558	43.045
ilpd	12.107	13.09	2.563	3.208	0	0.001	27.754	29.489
ionosphere	2.745	2.598	7.238	7.113	2.33	2.681	6.941	6.691
libras-move	9.126	9.923	5.63	7.972	8.644	8.611	15.629	16.667
lsvt	7.861	5.374	13.055	14.529	5.796	4.629	14.75	16.032
madelon	16.999	16.828	5.566	6.384	36.087	36.652	28.508	28.385
musk	0.763	1.017	0	0.011	0	0	0.006	0.02
satimage	2.746	2.997	0.757	0.953	0.976	1.072	7.907	8.287
tic-tac-toe	1.263	1.466	5.41	5.864	3.026	3.454	4.053	4.081
vehicle	9.837	12.237	2.626	3.725	5.418	6.358	24.5	24.749
vowel	1.79	3.846	2.727	5.037	2.09	3.185	2.01	2.485
wdbc	0.921	0.971	1.67	1.214	0.931	0.595	3.875	3.796
Average	5.805	6.485	4.158	5.277	5.078	5.770	13.650	14.834

**APPENDIX E – EMBARRASSINGLY PARALLEL INDEPENDENT TRAINING
OF MULTI-LAYER PERCEPTRONS WITH HETEROGENEOUS
ARCHITECTURES**

Embarrassingly Parallel Independent Training of Multi-Layer Perceptrons with Heterogeneous Architectures

Felipe C. Farias

Teresa B. Ludermir

Center of Informatics

Federal University of Pernambuco

Recife, PE, Brazil

FELIPEFARIAX@GMAIL.COM

TBL@CIN.UFPE.BR

Carmelo J. A. Bastos-Filho

E-Comp

University of Pernambuco

Recife, PE, Brazil

CARMELOFILHO@IEEE.ORG

Editor:

Abstract

The definition of a Neural Network architecture is one of the most critical and challenging tasks to perform. In this paper, we propose ParallelMLPs. ParallelMLPs is a procedure to enable the training of several independent Multilayer Perceptron Neural Networks with a different number of neurons and activation functions in parallel by exploring the *principle of locality* and parallelization capabilities of modern CPUs and GPUs. The core idea of this technique is to use a Modified Matrix Multiplication that replaces an ordinal matrix multiplication by two simple matrix operations that allow separate and independent paths for gradient flowing, which can be used in other scenarios. We have assessed our algorithm in simulated datasets varying the number of samples, features and batches using 10,000 different models. We achieved a training speedup from 1 to 4 orders of magnitude if compared to the sequential approach.

Keywords: neural networks, parallelization, scatter add, gpu, supervised learning

1. Introduction

Machine Learning models are used to solve problems in several different areas. The techniques have several knobs that must be chosen before the training procedure to create the model. The choice of these values, known as hyper-parameters, is a highly complex task and directly impacts the model performance. It depends on the user experience with the technique and knowledge about the data itself. Also, it is generally difficult for non-experts in modelling and the problem to be solved due to a lack of knowledge of the specific problem. The user usually has to perform several experiments with different hyper-parameter values to maximize the model's performance to find the best set of hyper-parameters. Depending on the size of the model and the data, this can be very challenging.

The hyper-parameter search can be performed manually or using search algorithms. The manual hyper-parameter search can be done by simply testing different sets of hyper-parameters. One of the most common approaches is to use a grid search. The grid-search process assesses the models by testing all the possible combinations given a set of hyper-

parameters and their values. Search algorithms can also be applied by using knowledge of previous runs to test promising hyper-parameter space during the next iteration, such as evolutionary and swarm algorithms, Gaussian processes Bergstra et al. (2015). In this paper, we focus on the hyper-parameter search for Artificial Neural Networks since it is an effective and flexible technique due to its universal approximator capabilities Leshno et al. (1993) applied to tabular datasets since it is a prevalent type of data used in the real world by several companies.

The best set of hyper-parameters could be chosen if the technique could be assessed using all the possible values for each variable. However, this is usually prohibitive due to the model training time. Two paths can be taken to accelerate the process: (i) reducing the training time of each model or (ii) assessing the model in the most promising points of the hyper-parameter space. The second point can be challenging because it is hard to perform a thorough search using a few points.

GPUs are ubiquitous when training Neural Networks, especially Deep Neural Networks. Given the computational capacity that we have today, we can leverage the parallelization power of CPUs that contain several cores and threads and GPUs with their CUDA Nickolls et al. (2008) cores to minimize the training time. The parallelization is commonly applied during matrix multiplication procedures, increasing the overall speed of the training process. However, if the data and the model that is being used do not produce big matrices, GPUs can decrease the training time if compared to CPUs due to the cost of CPU-GPU memory transfers Gregg and Hazelwood (2011). Also, the GPUs might not saturate their usage when using small operations. If we aggregate several MLPs as a single network with independent sub-networks, the matrices become more extensive, and we can increase the GPU usage, consequentially increasing the training speed. In this paper, we (i) designed a single MLP architecture that takes advantage of caching mechanisms to increase speed in CPUs or GPUs environment, called ParallelMLPs from now on, that contains several independent internal MLPs and allow us to leverage the parallelization power of CPUs and GPUs to train individual and independent Neural Networks, containing different architectures (number of hidden neurons and activation functions) simultaneously.

2. Background

In this section, we present important aspects of hyper-parameter search, computer organization, training speed of machine learning models.

2.1 Hyper-parameter Search

Hyper-parameters can be defined as the general configurations to create the models and control the learning process. They might be discrete, such as the (i) number of layers, (ii) number of neurons, (iii) activation functions, or continuous such as (i) learning rate and (ii) weight regularization penalty, among others. There are several ways to perform a hyper-parameter search. The most common and simple one is the Manual Search. The user can run a grid search to try all the possible combinations given a set of possibilities for every hyper-parameter being searched. In Bergstra and Bengio (2012), the authors claim that performing a Random Search is more efficient than running a grid search. We believe

that the random search is more critical for continuous values since it is more difficult to find the correct answer for that specific hyper-parameter. Nevertheless, a grid search might be the complete assessment of the best discrete hyper-parameter settings for the discrete ones. More knowledge regarding the function space leads to a better choice of the optimized point. In other words, if more architectures are tested, more information is gained about the hyper-parameter that maximizes the performance on a given task.

2.2 Computer Organization and Model Training Speed

Several factors can impact the computational speed of an algorithm. Consequentially, there are several ways to improve the efficiency of an algorithm. One of the most important is through the Big-O analysis Cormen et al. (2022). However, once the algorithm is optimized regarding Big-O, usually hardware efficiency is related to the algorithm speed. The computer architecture is made up of several components. The Von Neumann model von Neumann (1993) describes the architecture of digital computers. In this architecture, the computer must have (i) a Central Processing Unit (CPU) that controls everything, containing an Arithmetic and Logic Unit (ALU), a program counter (PC), and processor registers (ii) primary memory to store data and instructions, external mass storage, and input/output mechanisms. All the communication happens in a shared bus. This shared bus can be a bottleneck as it can limit the throughput between CPU and memory. Most modern computers operate with multi-core processors containing several cache levels between the CPU and RAM Edelkamp and Schrödl (2012) to mitigate this bottleneck. Algorithms aware of memory hierarchy management and computer organization can have severe speedups during execution Vanhoucke et al. (2011).

The ParallelMLPs training algorithm was designed to mainly take advantage of *principle of locality* Ali and Syed (2013). When using several models as a single model (i.e. fusing several matrices), we store this data contiguously, which can be very useful for caching mechanisms to exploit the spatial locality, pre-fetching from memory the input data and model parameters' neighbourhood. Since we are presenting a specific batch for several models simultaneously, the temporal locality can also be exploited once this data might be retained for a certain period. The instructions can also be cached efficiently since, in big matrix multiplication, the instructions will be repeated several times.

The CPU and GPU processors are an order of magnitudes faster than memory transfer operations (RAM or VRAM). When training sequentially, the processors can waste several cycles waiting for specific operations such as batch creation (RAM access, CPU to GPU transfers, CUDA global memory to CUDA shared memory). When training 10,000 models sequentially for 10 epochs in 1,000 samples, we will have to randomly access memory (bad caching locality properties) to create the batches $10,000 * 10 * 1,000 = 100,000,000$ times. Several cache misses can happen during batch fetching. When we use ParallelMLPs, we increase the chance of cache hit since we are using the current batch on 10,000 models simultaneously. Probably the batch is in the cache closest to the processor during the tensor operations. At the same time, we decrease the number of memory access in case of cache misses to $10 * 1,000 = 10,000$ at maximum. Of course, everything described here depends on the computer it is being applied to due to different memory layouts and the number of cores.

2.3 Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) is a prevalent and powerful machine learning model Goodfellow et al. (2016). There are some challenges to training it since the architecture definition is not an easy task and is very dependent on the user experience. The training process achieves the parameter adjustment of an MLP. It can be divided into two phases, the forward phase. We project the input space into a hidden/latent space, usually applying a non-linear function to a weighted sum of the inputs, projecting this latent space into the output space again to have the predictions. These predictions are compared against the targets that the model is trying to learn through a loss function that measures how good the models' predictions are. We perform the backward phase with the error by taking the partial derivatives of each parameter w.r.t. the error to update the parameters, hopefully decreasing the prediction errors in the following forward phase.

3. Methodology

In order to facilitate the methodology explanation, we will write the tensors in capital letters. Their respective dimensions can be presented as subscripts such as $W_{[3,4]}$ meaning a two-dimensional tensor with three rows and four columns.

An example of a MLP architecture with 4 inputs, 3 hidden neurons and 2 outputs (4 – 3 – 2) can be seen in Figure 1. The weights are also explicit with notation w_{ij} meaning a connection from neuron j in the current layer to neuron i in the next layer. The first weight matrix with shape $[3, 4]$ projects the input representation (4 dimensions) into the hidden representation (3 dimensions). In contrast, the second matrix with shape $[2, 3]$ projects the hidden representation (3 dimensions) into the output representation (2 dimensions).

The training procedure of an MLP is usually performed by the Backpropagation algorithm Werbos (1982). It is divided into two phases. We perform several non-linear projections in the forward phase until we reach the final representation in the backward phase. Then, we can calculate the error using a loss function to backpropagate them using the gradient vector of the error w.r.t. the parameters to update the parameters of the network in order to minimize its error.

The forward calculation can be described as two consecutive non-linear projections of the type $H = X \times W_1^T$. and $Y = H \times W_2^T$. We need two different weight matrices w_1 with shape $[3, 4]$ and w_2 with shape $[2, 3]$. We also would want two bias vectors, but we will not use them to facilitate the explanations and figures.

The forward phase in our example can be visualized as a three-step procedure.

1. Input to hidden matrix multiplication, $H_{batch, hid} = X_{batch, in} \times W_{hid, in}^T$.
2. Hidden activation function application, $H'_{batch, hid} = \sigma(H_{batch, hid})$.
3. Hidden activated to output matrix multiplication, $Y_{batch, out} = H'_{batch, hid} \times W_{out, hid}^T$.

To train the model, we need to apply a loss function to compare the numbers in step 3 against the targets which the model is trying to learn. After the loss calculation, the gradient of each parameter w.r.t. the loss is estimated and used to update the parameters of the network.

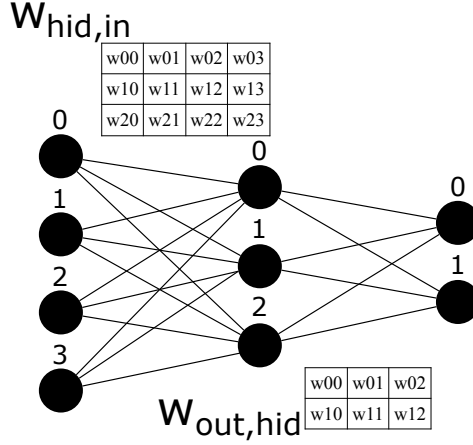


Figure 1: Simple MLP with 4 inputs, 3 hidden neurons, and 2 outputs and its weight matrices.

Since MLP architectures usually do not saturate GPUs, we can change the architecture to create MLPs that share the same matrix instance but have their own independent set of parameters. It allows them to train in parallel. Suppose we want to train two MLPs in the same dataset with architectures $MLP_1 = 4 - 1 - 2$ and $MLP_2 = 4 - 2 - 2$. To leverage the GPU parallelization, we can fuse both architectures as a single architecture (ParallelMLPs), in the form $4 - 3 - 4$. In Figure 2 we can see both internal networks represented as a single architecture (ParallelMLPs) with different colours. The number of inputs does not change; the number of hidden neurons is summed, while the number of output neurons is multiplied by the number of independent MLPs we want to train. The layout of ParallelMLP was designed to take advantage of temporal and spatial locality principles in caching Ali and Syed (2013) and overall parallelization mechanisms.

In order to maintain the internal MLPs independent of each other, we cannot follow the same steps previously described because, in that case, we would mix the gradients of all the internal models during the backpropagation. The matrix multiplication can be understood as two consecutive operations: (i) element-wise vector multiplication (rows * columns) and (ii) reduced sum of the previous vectors. We need to change how we perform the matrix multiplication in step 3. The main idea is to divide the matrix multiplication into two procedures: (i) matrix element-wise multiplication and (ii) summation. However, the summation needs to be carefully designed such that we do not reduce-sum the entire vector but different portions of the axis. We can use broadcast techniques implemented in every tensor library, and a special case of summation called Scatter Add to make these two procedures memory efficient. We will refer to this procedure as Modified Matrix Multiplication (M3). Ahn et al. (2005).

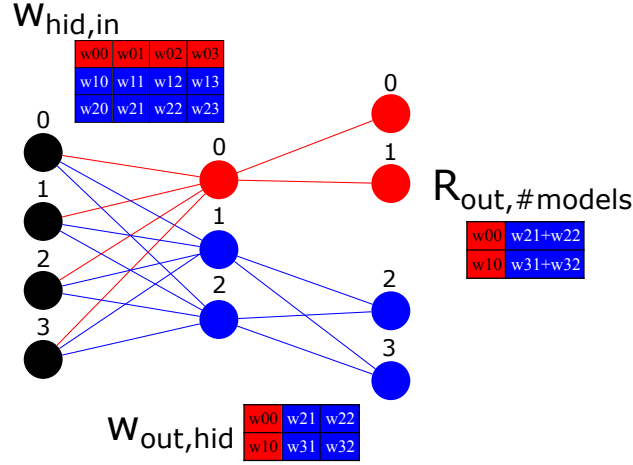


Figure 2: Two independent MLPs are represented as a single MLP with four inputs, 1 and 2 hidden neurons, and two outputs. The weight matrices are also highlighted to understand the parameter mapping from the single MLP architecture to its two internal MLP architectures. The red colour is related to the 4 – 1 – 2 MLP, while the blue colour is related to the 4 – 2 – 2 MLP. $R_{out,\#models}$ contain the result after applying the Scatter Add operation.

This M3 procedure might be useful to handle sparse NN. Most of the time, sparsity is treated with masking. Masking tends to be a waste of resources since the original amount of floating point calculations are still being done, and additional masking floating point operations are being added to the process.

The Scatter Add ($\phi(D, S, I)$) operation takes a dimension D to apply the operation and two tensors as inputs. The source tensor S contains the numbers that we want to sum up and the indices tensor I , which informs which elements in the source tensor must be summed and stored in the result tensor. When applied to two-dimensional tensors as $\phi(1, S, I)$, the result tensor R (initialized as zeros) can be calculated as:

- $R[I[i, j], j] = R[I[i, j], j] + S[i, j]$ if dimension = 0
- $R[i, I[i, j]] = R[i, I[i, j]] + S[i, j]$ if dimension = 1

A very simple example of the scatter add operation would be when:

- $D = 1$,
- $S_{[1,6]} = [[1, 2, 3, 4, 5, 6]]$,
- $I_{[1,6]} = [[0, 1, 1, 2, 2, 2]]$,
- $R_{[1,3]} = \phi(1, S, I)_{[1,3]} = [[1, 5, 15]]$,

with I informing how to accumulate values in the destination tensor R , with the first element (0) accumulating only the first element of S , the second destination element (1) accumulating the second and third values of S , and the latest element (2) accumulating the fourth, fifth and sixth element of S . This operation is implemented in parallel in any popular GPU tensor library. In our architecture represented in Figure 2, in order to have the two separated outputs, one for each internal MLP, we would have to sum across the lines using a matrix I as follows:

$$I_{[2,3]} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

It would generate an output matrix $[2, 2]$ where each line is related to each internal/individual MLP.

The Scatter Add operation is responsible for keeping the gradients after the loss function application not mixed during the backpropagation algorithm, allowing us to train thousands of MLPs simultaneously in an independent manner.

To summarize, the steps to perform the parallel training of independent MLPs are:

1. Input to hidden matrix multiplication, $H_{batch,hid} = X_{batch,in} \times W_{hid,in}^T$.
2. Hidden activation function application, $H'_{batch,hid} = \sigma(H_{batch,hid})$.
3. Hidden activated element-wise multiplication with multi-output projection matrix, $S_{batch,out,hid} = H'_{batch,1,hid} \odot W_{1,out,hid}$ (broadcasted element-wise multiplication)
4. Scatter Add to construct the independent outputs, $Y_{batch,model,out} = \phi(1, S_{batch,out,hid}, I_{batch,out,hid})$

After that, a loss function is applied to calculate the gradients and update all the internal MLPs parameters independently.

We can go further and use not only a single activation function that should be applied to all the internal MLPs, but several of them by repeating the original number of architectures as many times as we have for activation functions. Since this is often not enough to use all the GPU resources, one can also have repetitions of the same architecture and activation function. In order to use several activation functions, the last step must be modified such that we can apply different activation functions to different portions of the matrix O . This can be done using a tensor split operation, applying each activation function iteratively in different continuous portions of the representations, and finally concatenating the activated representations again.

4. Experiments

Simulations were performed in order to compare the speed of the Parallel training against the Sequential approach.

4.1 Computing Environment

A Machine with 16GB RAM, 11GB NVIDIA GTX 1080 Ti, and an I7-8700K CPU @ 3.7GHz containing 12 threads were used to perform the simulations. All the code was written using PyTorch Paszke et al. (2019).

4.2 Model Architectures

We have created architectures starting with one neuron at the hidden layer until 100, resulting in 100 different possibilities. For each architecture, we have assessed ten different activation functions: *Identity*, *Sigmoid*, *Tanh*, *ReLU*, *ELU*, *SeLU*, *GeLU*, *LeakyReLU*, *Hardshrink*, *Mish*. It increases the number of independent architectures to $100 * 10 = 1,000$. We have repeated each architecture 10 times, totalling $1,000 * 10 = 10,000$ models. It is worth mentioning that this design is not limited to these circumstances. One could create arbitrary MLP architectures such as 3 different networks with 3, 19, and 200 hidden neurons and still would be able to leverage the speedup from ParallelMLPs.

4.3 Datasets

We have created controlled training datasets with 100, 1,000, and 10,000 samples. With 5, 10, 50, and 100 features. Giving a combination of 12 different datasets. For all the simulations, 12 epochs were used, ignoring the first two epochs as a warm-up period, and 32 as the batch size.

4.4 Training Details

All the samples are stored in GPU at the beginning of the process to not waste much time of GPU-CPU transfers. It favours the Sequential processing speed more than the Parallel since the former would have 10,000 more CPU-GPU transfers throughout the entire experiment.

The data is used only as train splits because this phase is where the gradients are calculated, and the two operations of forward and backward are used. Therefore, the training split processing is much more expensive than validation and test splits.

5. Results and Discussion

The following tables contain the average training time of 10 training epochs (forward and backward steps, no validation, and no test loops) when varying the number of samples (columns) and the number of features (rows) for strategies: (i) Parallel (using ParallelMLPs), (ii) Sequential (training one model at the time), and (iii) the percentage of ParallelMLPs training times against the Sequential strategy (Parallel/Sequential). The CPU and GPU results are presented in Table 5 and Table 5, respectively.

Suppose one is training for 100 epochs of the previously mentioned 10,000 models in a dataset with 10,000 samples and 100 features with 32 as the batch size. In that case, CPU-Sequential can take more than 32 hours ($1179.405 * 100 / 3600 = 32.76$), while CPU-Parallel only 2 hours ($100 * 74.661 / 3600 = 2.07$) would be necessary to perform the same training. The same case with batch size of 256 samples, we would have approximately

EMBARRASSINGLY PARALLELMLPs

	Number of Samples								
	100			1000			10000		
	Batch Size								
	32	128	256	32	128	256	32	128	256
Features	Parallel (Seconds)								
5	0.525	0.463	0.472	5.248	4.709	4.717	52.737	46.929	47.133
10	0.539	0.466	0.475	5.338	4.722	4.742	53.351	47.089	47.153
50	0.658	0.505	0.501	6.144	4.943	4.887	62.664	49.791	48.85
100	0.809	0.547	0.551	7.373	5.366	5.1	74.661	53.09	50.965
	Sequential (Seconds)								
5	13.437	6.097	5.994	112.054	56.999	48.852	1097.599	564.428	483.278
10	13.36	6.115	6.01	111.84	57.094	49.015	1097.503	564.701	484.91
50	13.884	6.571	6.467	116.303	58.993	49.546	1134.955	583.468	491.269
100	14.283	6.671	6.592	120.297	59.259	50.371	1179.405	586.267	493.744
	Parallel/Sequential (%)								
5	3.91	7.59	7.88	4.684	8.262	9.656	4.805	8.314	9.753
10	4.038	7.625	7.905	4.773	8.27	9.674	4.861	8.339	9.724
50	4.739	7.69	7.753	5.282	8.379	9.863	5.521	8.534	9.944
100	5.664	8.198	8.362	6.129	9.056	10.126	6.33	9.056	10.322

Table 1: Average of 10 epochs to train 10,000 models using a CPU.

14 hours and 1.5 hours for CPU-Sequential and CPU-Parallel, respectively. In this case, the CPU-Parallel is 15.8 for the first scenario and 9.3 for the second scenario times faster than CPU-Sequential. The CPU-Parallel experiments took between 3.9% and 10.3% of the CPU-Sequential time, considering all the variations we have performed. As one can see, the CPU speed improves when using larger batch sizes probably to better exploration of the *principle of locality*.

If we analyze the same experiments in GPUs, more than 51 hours ($1854.881 \times 100 / 3600 = 51.5$) would be necessary for GPU-Sequential, and 7.4 minutes when using GPU-ParallelMLPs for the 32 batch experiment and 15.5 hours for GPU-Sequential and 4.6 minutes for the 256 batch size. It gives us a speed improvement on GPU-Parallel of 417.6 and 202.17 times, respectively, when compared to GPU-Sequential. The GPU-Parallel experiments range from 0.017% to 0.486% of the GPU-Sequential time for all the assessed experiments. As one can see, the GPU speed improves when using larger batch sizes probably to better exploration of the *principle of locality* and also a better parallelization in the GPU kernel.

At first glance, the GPU training time should be faster than the CPU training time. When comparing GPU-Sequential against the CPU-Sequential, the GPU slowness can be characterized by the high number of function/kernel calls to perform high-speed operations (small matrix multiplications). The single-core of a CPU is optimized to perform a specific computation very quickly, whereas the single-core of a GPU will be much slower due to its meagre clock rate. However, GPUs contain more cores than the CPU, and they can run the computation in parallel. It is often the scenario in which GPUs will outperform CPUs. As

	Number of Samples								
	100			1000			10000		
	Batch Size								
	32	128	256	32	128	256	32	128	256
Features	Parallel (Seconds)								
5	0.024	0.002	0.001	0.269	0.177	0.203	2.676	1.756	2.426
10	0.025	0.002	0.001	0.276	0.178	0.206	2.745	1.767	2.439
50	0.033	0.002	0.001	0.351	0.193	0.218	3.483	1.926	2.569
100	0.043	0.002	0.001	0.449	0.215	0.233	4.438	2.128	2.75
	Sequential (Seconds)								
5	22.911	8.646	8.511	189.411	73.503	57.02	1857.653	722.915	566.592
10	22.983	8.619	8.515	188.966	73.462	56.941	1859.14	722.925	568.583
50	23.025	8.628	8.519	189.147	73.364	57.134	1858.07	719.359	567.847
100	22.993	8.581	8.503	189.015	72.849	57.129	1854.881	717.543	566.248
	Parallel/Sequential (%)								
5	0.106	0.019	0.017	0.142	0.241	0.355	0.144	0.243	0.428
10	0.11	0.019	0.017	0.146	0.243	0.362	0.148	0.245	0.429
50	0.142	0.019	0.017	0.185	0.264	0.381	0.187	0.267	0.452
100	0.186	0.018	0.017	0.237	0.294	0.408	0.239	0.297	0.486

Table 2: Average of 10 epochs to train 10,000 models using a GPU.

we are increasing the size of matrices to be multiplied in GPU-ParallelMLP, a considerable amount of speed can be delivered compared to CPU-ParallelMLP.

It is essential to mention that the GPU memory consumption of the 10,000 parallel models using 100 features and batch size of 256 (the worst case scenario for our experiments w.r.t. memory allocation) was less than 4.8GB, meaning that (i) simpler GPUs can be used and still take advantage of our approach, and (ii) is probably possible to improve the speed if using more models in parallel to make a better usage of the GPU memory.

As we can perform a very efficient grid-search in the discrete hyper-parameters space that will define the network architecture, it is much easier for the user to select a suitable model since several number of neurons and activation functions can be trained in parallel, mainly for beginners in the field who have difficulties to guess the best number of neurons and activation function, since the hyper-parameter definition is highly dependent on the user experience. Also, researchers that use any search method that proposes an MLP architecture in a specific problem can now train the models in parallel. The ParallelMLPs can be applied for both classification and regression tasks. We believe this M3 operation can be optimized and lead to even more speed improvements if a specialized CUDA kernel could be written.

6. Conclusion

We have demonstrated how to parallelize a straightforward core operation of Neural Networks by carefully choosing alternative operations with a high degree of parallelization in modern processors (CPUs or GPUs).

The ParallelMLPs algorithm described in 3 was able to accelerate the training time of several independent MLPs with a different number of architectures and activation functions from 1 to 4 orders of magnitude by simply proposing an efficient memory representation layout that fuses several internal MLPs as a single MLP and using the M3 strategy to perform the matrix projection in an independent way. It allows us to explore better the *principle of locality* and the parallelization in modern processors (CPUs and GPUs). The technique can be helpful in several areas to decrease the training time and increase the number of model assessments. Since we are able to train thousand of networks in a reasonable time, we can investigate the distribution of models for a specific dataset in a large scale. Also, it might be a good way to represent sparse NN.

7. Future Works

We believe the ideas proposed in this paper can inspire other researchers to develop parallelization of other necessary core operations/modules such as Convolutions, Attention Mechanisms, and Ensemble fusion of heterogeneous MLPs.

In future works, we would like to investigate if the M3 operation can be used from the second transformation until the last layer to train MLPs with more than one hidden layer since only during the first transformation (from input to the first hidden layer) all the previous neurons are sum-reduced instead of a sparse version of them. We can see an example of this idea into Figure 3.

An interesting work would be to perform feature selection using ParallelMLPs by repeating the MLP architecture and creating a mask tensor to be applied to the inputs before the first input to hidden projection. We also plan to perform model selection in the large pool of trained MLPs in a specific dataset. Also, we plan to automatize the number of neurons and the number of layers. After we finish the ParallelMLP training, we can (i) remove the output layer or (ii) use the output layer as the new representation of the dataset to be the input of a new series of ParallelMLP training. It is also possible to use the original features concatenated with the previously mentioned outputs like residual connections Szegedy et al. (2017). After each ParallelMLP training, we can pick the best MLP to create the current layer, continuously increasing the number of layers until no more improvements are perceived. A further investigation is needed to verify if similar ideas could be used for convolutional and pooling layers since they are basic building blocks for several Deep Learning architectures. We also would like to investigate what happens if an architecture containing a backbone representing the input space into a latent space and MLP at the end, such as Chen et al. (2020) to perform the classification would be replaced by a parallel layer with several independent set of outputs, but sharing the backbone’s parameters. One hypothesis is that the backbone would be regularized by different update signals from a heterogeneous set of MLP heads. This technique can also be similar to a Random Forest Breiman (2001)

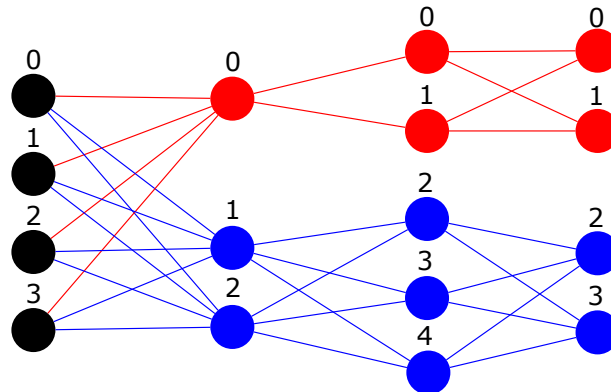


Figure 3: Two independent two hidden layers MLPs represented as a single MLP by ParallelMLP. The first with architecture in red being $4 - 1 - 2 - 2$ and the second in blue $4 - 2 - 3 - 2$. The weight matrices were removed to ease the readability of the figure.

of MLPs where during the training phase, the inputs could be masked depending on the individual network to mimic bagging or mask specific features, or even simulate a Random Subspace Ho (1998). Our technique can be used to find the best random initialized model given that Malach et al. (2020) was able to found good sub-networks without any training – extending the idea of Lottery Ticket Hypothesis Frankle and Carbin (2018). There is space to parallelize even more hyper-parameters such as batch size, learning rate, weight decay, and initialization strategies. One straightforward way is to use boolean masks to treat each case or hooks to change the gradients directly, but it might not be the most efficient way. A handy extension of our proposition would be to automatically calculate the hyper-parameter space to saturate the GPU.

Acknowledgments

We would like to acknowledge the Centro de Tecnologias Estratégicas do Nordeste (CETENE) for providing computational resources (FACEPE APQ-1864-1.06/12).

References

- Jung Ho Ahn, Mattan Erez, and William J Dally. Scatter-add in data parallel architectures. In *11th International Symposium on High-Performance Computer Architecture*, pages 132–142. IEEE, 2005.
- Amjad Ali and Khalid Saifullah Syed. Chapter 3 - an outlook of high performance computing infrastructures for scientific computing. In Atif Memon, editor, *Advances in Computers*, volume 91 of *Advances in Computers*, pages 87–118. Elsevier, 2013. doi: <https://>

- doi.org/10.1016/B978-0-12-408089-8.00003-3. URL <https://www.sciencedirect.com/science/article/pii/B9780124080898000033>.
- James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.
- James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015.
- Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.
- Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.
- Stefan Edelkamp and Stefan Schrödl. Chapter 8 - external search. In Stefan Edelkamp and Stefan Schrödl, editors, *Heuristic Search*, pages 319–365. Morgan Kaufmann, San Francisco, 2012. ISBN 978-0-12-372512-7. doi: <https://doi.org/10.1016/B978-0-12-372512-7.00008-0>. URL <https://www.sciencedirect.com/science/article/pii/B9780123725127000080>.
- Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- Chris Gregg and Kim Hazelwood. Where is the data? why you cannot debate cpu vs. gpu performance without the answer. In *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*, pages 134–144. IEEE, 2011.
- Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.
- Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993. ISSN 0893-6080. doi: [https://doi.org/10.1016/S0893-6080\(05\)80131-5](https://doi.org/10.1016/S0893-6080(05)80131-5). URL <https://www.sciencedirect.com/science/article/pii/S0893608005801315>.
- Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pages 6682–6691. PMLR, 2020.
- John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53, 2008.

- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.
- Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.
- J. von Neumann. First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993. doi: 10.1109/85.238389.
- Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

**APPENDIX F – HAVE WE BEEN NAIVE TO SELECT MACHINE LEARNING
MODELS? NOISY DATA ARE HERE TO STAY!**

Have we been Naive to Select Machine Learning Models? Noisy Data are here to Stay!

Felipe C. Farias

Teresa B. Ludermir

Centro de Informatica

Universidade Federal de Pernambuco

Recife, PE, Brazil

FELIPEFARIAX@GMAIL.COM

TBL@CIN.UFPE.BR

Carmelo J. A. Bastos-Filho

E-Comp

University of Pernambuco

Recife, PE, Brazil

CARMELOFILHO@IEEE.ORG

Editor:

Abstract

The model selection procedure is usually a single-criterion decision making in which we select the model that maximizes a specific metric in a specific set, such as the Validation set performance. We claim this is very naive and can perform poor selections of over-fitted models due to the over-searching phenomenon, which over-estimates the performance on that specific set. Furthermore, real world data contains noise that should not be ignored by the model selection procedure and must be taken into account when performing model selection. Also, we have defined four theoretical optimality conditions that we can pursue to better select the models and analyze them by using a multi-criteria decision-making algorithm (TOPSIS) that considers proxies to the optimality conditions to select reasonable models.

Keywords: neural networks, model selection, parallelization, over-fitting, over-searching, noise

1. Introduction

Machine Learning methods have been successfully applied to several areas because the algorithms can learn complex patterns from data. During the model development phase, we usually try to find the best model to solve a specific task. It is common to assess several different kinds of models and different combinations of hyper-parameters. After that, we need to select a model, or a set of models in the case of Ensembles, to deploy it to production to predict things in the real world.

This paper investigates practical ways to select individual models to put in production for real-world usage. We call this problem model selection. The book (Hastie et al., 2009) states that the model selection goal is to estimate the performance of different models to choose the best one. One common approach is to use all the trained models as an ensemble. However, we are often interested in choosing a single model instead of several models to be used as an ensemble during operation.

We can define over-fitting as a phenomenon that prevents the model from generalizing well on unseen data. In this case, the model can perfectly describe the Train set but have inferior performance at the Test set. In (Dietterich, 1995), Dietterich claims that over-fitting is a phenomenon that emerges as we work too hard to find the best fit to the training data because there is a risk of fitting the noise by memorizing peculiarities of that training data instead of learning the general predictive rule.

In (Caruana et al., 2000), the authors claim that MLP models with excess capacity (number of hidden neurons) generalize well if training with early stopping. An over-fitting overview was done in (Ying, 2019) where the author discusses potential causes and solutions. To reduce the effects of over-fitting, they suggest the application of 4 perspectives to mitigate the over-fitting problem: (i) use of early stopping, (ii) network reduction, (iii) data expansion, and (iv) regularization. In our proposal, we are explicitly using the early stopping and regularization in the form of weight decay.

From the results of the paper (Zhang et al., 2021), one can confirm how powerful artificial Neural Networks (ANN) are and how important it is to use procedures to mitigate the over-fitting potential. They have found that ANN with sufficient parameters can perfectly fit even random labels. However, when using the trained model in the Validation set, the performance was terrible (as expected). The gap/disagreement between the random noise set and the correctly labeled set inspired our current proposal.

If we analyze from the perspective of (Ng et al., 1997), the fact that real data is noisy makes model selection even harder. Consider an example in which we are performing model selection using a single-criterion approach to maximize the Holdout performance. Our Holdout set contains 100 samples, of which 80 samples are correctly labeled without noise, but 20 samples are noisy/wrong. Theoretically, suppose we have models A and B predicting 80% of the time correctly. In that case, it might be the case that A is better than B in the Test set because A correctly predicted the 80/80 healthy and 0/20 problematic samples. In comparison, B correctly predicted 60/80 healthy and 20/20 problematic samples, simply (over-)fitting the noise. Since we usually do not have the information of which points are correctly labeled or not. We can analyze the expectations. Now let us imagine that model A correctly predicted all the samples according to the current labeling (80/80 and 20/20) and B correctly predicted only 80% (80/80 and 0/20), i.e., model A is over-estimated on the Holdout set. The most common approach is to select model A since it maximizes the single criterion. However, if we look into the Test set, model B will probably have a better performance because it does not fit the same noise present in the Holdout set. Since the over-estimated model would be selected due to noise in the Holdout set, we expect that it will correctly predict only 80% of the correctly labeled data and present a bad performance on the noisy data in the test set. If model B is chosen, we would expect that it would correctly label roughly 100% of the correctly labeled data and still have approximately lousy performance in the noisy portion of Test data. It is important to create strategies to better handle this phenomenon.

Although single-criterion is probably the most common approach to select machine learning models, there are works that tries to perform a multi-criteria model selection such as in Ali et al. (2017), that proposed a multi-criteria decision making methodology using accuracy, time and consistency of each model to select the best one. It uses the TOPSIS Tzeng and Huang (2011) ranking to measure the distance to the ideal classifier.

The TOPSIS were also used in (Vazquez et al., 2020) to model selection, but only used different performance metrics for a specific set. A fuzzy approach to work alongside with TOPSIS was proposed in Akinsola et al. (2019). None of the mentioned multi-criterion proposals look into the training data to perform the model selection. We argue that the performance equilibrium between the individual sets can be a proxy measurement to the model robustness.

As far as we know, this is the first work investigating an immense number of NN architectures trained for several tabular datasets. The extensive regime of models allowed observing interesting behaviors during model selection that usually does not happen in small regime.

The objective of this paper is to compare the commonly used single-criterion against the multi-criterion model selection procedure. This paper is organized as follow. In Section 2 we present our contribution and define optimality conditions of theoretical optimal models. In Section 3 we explain the experimental setup. In Section 4 we introduce and discuss the results. The conclusions are given in Section 5. Finally, in Section 6 we present future directions and opportunity of investigations on how to improve our contributions.

2. Methodology

This section explains our methodology to study model selection on a large scale. The primary objective is to generate a partition of the data to analyze the effect of the model selection procedure in the model put in production.

2.1 Data Splitting

The data splitting procedure is depicted in Figure 1. The first step is to divide the data into 10-folds. After that, the last fold is used as a Fixed Test set (blue box) among all the runs for a specific dataset. Another fold is used as a Holdout set (green box). Each of the remaining eight folds will be used once as a Validation set (red box), and the other seven will compose the Train set (gray box). It allows us to run nine times with different splits for the sets. We repeat the experiment twice, totaling 18 independent runs for each dataset.

It is worth mentioning that both Holdout and Fixed Test were never used during model training (neither was used in early stopping). Therefore, both sets can be seen as proxies for future unseen data, and can be used to estimate the models' performance when used in production.

2.2 Models

We have used the ParallelMLPs (Farias et al., 2022) approach to create several different MLP architectures and analyze them simultaneously. The ParallelMLP was arranged to create architectures with 1 to 100 neurons in the hidden layer, using seven different activation functions (700 architectural possibilities) and eight repetitions (each repetition will use a different validation set). It creates a total of $100 \times 7 \times 8 = 5,600$ MLPs. We use the validation set to perform early stopping with the patience of 10 epochs.

With this strategy, seven splits are used as train splits. Since we have eight architectural repetitions (8 sub-networks are identical, they only differ by their random initialization), we

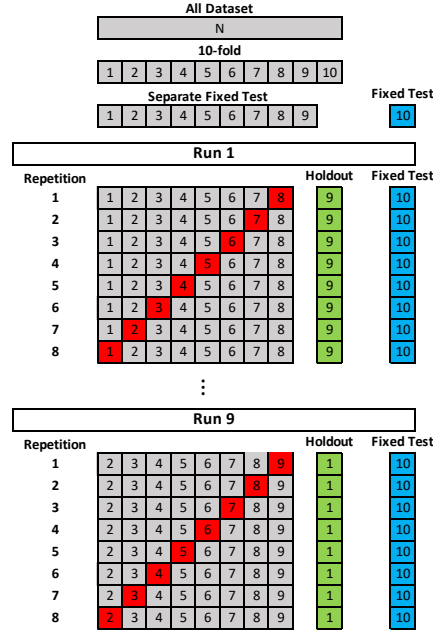


Figure 1: Description of the sets constructions.

round-robin the validation (early-stopping) split such that each one of the eight repetitions uses a different validation split. In order to accomplish this round-robin, we sample from training and validation splits (8 splits in total). We also use a train boolean mask that maps which dataset indices must be used as training samples for each sub-network to ignore those points during training dynamically. In practical terms, it means that the expected adequate batch size will proportionally have $7/8$ of the original batched samples, e.g., if $batch_size = 32$, it means that the expected number of training samples that each model will consider in each batch during back-propagation is $7/8 * 32 = 28$ samples, ignoring $1/8 * 32 = 4$ samples in expectation since it is marked for validation only. During the validation phase, we need to invert the training mask to obtain the validation mask. This procedure allows us to mimic the training of 8 independent trials varying which split is used to early stop the model. In our specific case, 700 different architectures are being trained 8 times, using a different validation set.

The model selection is a procedure responsible for selecting the best architecture, i.e., the one that maximizes the model performance on a specific dataset. The most common approach is using k-fold cross-validation and selecting the model with the best average performance on the validation set. In our understanding, this approach results in acceptable model choices, but it limits the learning process because the architecture is fixed. Consequently, the number of parameters must be the same.

As a Machine Learning system is composed by 4 components: (i) **model** – including architectures – to predict the targets; (ii) **loss function** to measure the success of the model;

(iii) **optimization** algorithm to adjust the parameters of the model; and (iv) the **data**. If one of these components changes during the learning process, we cannot guarantee that we will have the same performance – this is probably why k-Fold is a natural choice for ML hyper-parameter optimization. Even the initialization randomness can be a decisive factor for successfully learning a task once bad initial conditions difficult the learning dynamics of the model. Regarding the data, if we change the order in which the batches are chosen, we might have different learning dynamics. If we are performing a hyper-parameter search using a Tree or Gradient Boosting methods such as XGBOOST (Chen and Guestrin, 2016), LightGBM (Ke et al., 2017), and even simple Random Forests (Breiman, 2001), we usually optimize the number of estimators. However, each estimator can grow to a specific depth, leading to different architectures and parameters even if we keep the same hyper-parameters. It is also observed when Support Vector Machines (Cortes and Vapnik, 1995) (SVM) are used since the same hyper-parameters might create SVMs with different support vectors. This behavior of using a different number of parameters, even if we use the same hyper-parameters, mostly happens due to the learning dynamics given the initial conditions and the samples chosen to train the models, leading to different error landscapes, affecting the learning process. That is why we believe that forcing a NN to have a fixed number of parameters that will store the model’s knowledge is not a good idea since the model’s initialization and the splitting. Even the order of batches being presented will guide the model to different landscapes.

To the best of our knowledge, this is the first paper to investigate model selection using a large number of ANN. It was possible because we have used the ParallelMLPs algorithm to train several independent MLPs in an acceptable time window. Therefore, by looking into this regime, we could investigate further the relationship between different properties of NN trained on the same datasets. Due to the regime of a large number of trained models, we had the opportunity to investigate properties such as over-fitting and over-search on a large scale.

2.3 Optimality Conditions

We propose four optimality conditions based on desired properties that we believe the selected ANN models should have:

- High Generalization Performance
- High Robustness
- Low Complexity
- No Premature Early Stopping

We argue that the more conditions a model meet, the more confident we are about its future performance when deployed in production to be used in the real world.

2.3.1 HIGH GENERALIZATION PERFORMANCE

How should one select a model from several possible candidates? Selecting the best Holdout model would be the best approach?

When performing model selection, we are trying to collect the model that will perform the best during real-world operation. It means that we want a model with a high generalization. Therefore, the model should have good and close performance for the Holdout and Test sets.

Often only the Validation set is used to perform model selection. As we try to select the best model among several possibilities, we may incur into a problem if our objective is solely to maximize the Validation performance. In (Ng et al., 1997) the authors suggest that selecting the “apparent best” model is problematic because datasets are noisy, and the “apparent best” model that maximizes the performance is learning the noise in the data. They suggested an approach to select models based on a percentile of the possibilities called *percentile-cv*. However, it is a costly approach since it is based on a Leave-One-Out.

To analyze this optimality condition, we can define the Data Formation Rule Equation as

$$Y = M(X) \tag{1}$$

where the data (Y) can be explained by a prediction using the theoretical optimal Model (M) applied to the input data (X). However, real-world data very often contains noise, that’s why we add the ϵ component to account for noise/aleatoric uncertainty (Hüllermeier and Waegeman, 2021).

$$Y = M(X) + \epsilon \tag{2}$$

The noise can be related to the attribute or the targets/labels of the dataset. Therefore, for real-world data, we are usually influenced by $\epsilon > 0$ and $\epsilon \ll M$, otherwise the dataset would probably contain too much noise to be helpful. For an interesting survey on dataset noise, one can read the (Gupta and Gupta, 2019) work. Since the ϵ is irreducible due to its stochastic behavior, and therefore the model should not explain, we should avoid the model to learn ϵ . Consequently, if we have an incredible performance on a specific set of a dataset, it probably means that our model also has learned the noise of that set instead of learning what should be learned (model close to the theoretical optimal model). Usually, the more a model learns noise, the less generalizable it tends to be. We say that the single-criterion decision-making is naive because it assumes that the model should be able to explain the entire set, disregarding the actual irreducible error. When performing model selection, we really should be looking for a model that maximizes the metric performance of Y due to the $M(X)$ and not the ϵ term.

The NN must have the highest performance possible to solve a specific task when analyzing unseen data. The performance is usually measured only by looking into the Validation/Test set. Since we are using the Holdout and Test set as unseen data w.r.t. the Training and Validation sets, **the models must have the highest performance possible in those two sets**, on top of this assumption, we add that the metrics in both Holdout and Test sets should agree by having similar values, assuming they were pretty split without

any biases. We have used a Stratified 10-fold and the Similarity-Based Stratified Splitting (SBSS) (Farias et al., 2020), both designed to achieve equivalent statistics for their folds.

2.3.2 HIGH ROBUSTNESS

The Training and Validation set are one of the most important piece for the model to train, since the former updates the model’s parameters and the latter is applied to early stopping in order to avoid over-fitting. Since they play such a relevant role, why do we ignore their information during the model selection procedure? One can argue that Training set performance will be highly overestimated because the parameters are being optimized directly to fit them. However, we have found that, inside the immense pool of MLP candidates that we were able to create, we still can find similar models w.r.t the Test set performance, but with the Train and Validation metrics close to each other that does not seem to be over-estimated as we would expect.

This condition is an extension of the previous optimality condition, but now looking into the training data (Train and Validation sets). It is helpful to analyze an extreme case in which we have the universe of points (all possible points) for a given task, such as all possible movements in a game. Suppose we train a model on this universe that perfectly fits the task (without over-fitting), finding the theoretical optimal model. In that case, the optimal decision boundaries were correctly found. Therefore we would have correctly predicted 100% on all four sets (Train, Validation, Holdout, and Test). Suppose we randomly sub-sample the universe data and use only 80% of the original data. In that case, the model could converge to the same parameters and still have perfect predictions on the four sets (which is very unlikely) or converge to a point where it can correctly predict 90% of the cases. Since we randomly sub-sampled the data, we expect statistical properties to be preserved. Therefore, all four sets should have roughly 90% of the time correct predictions. In other words, **all the Train, Validation, Holdout, and Test sets should have similar performances**. This equilibrium probably increases the robustness of the model and the trustfulness w.r.t. their estimated performance metrics.

Another way to analyze this condition is to consider two models, A and B, where the Train, Validation, Holdout, and Test metrics for model A is 0.99, 0.94, 0.92, 0.85 and model B 0.86, 0.85, 0.85, 0.85. Which one should be selected? Model B is probably more robust than A since model A presents some behaviors usually observed in over-fitted models (very high Training/Validation performance). Since we usually do not look into the Train and Validation metrics, we would probably select model A if only looking into the Validation/Holdout performance metric. That is why we defend the importance of considering more than a single-criterion to create better decisions.

When we perform model selection using a single set as a single-criterion decision given a large number of models, we are probably selecting a model that maximizes that metric by correctly predicting the corrected labeled points, but also “correctly” predicting the noise on that data.

To better understand this optimality condition, we can decompose and specialize the Equation 2 by each individual set as follows:

$$Y_{train} = M(X_{train}) + \epsilon_{train} \quad (3)$$

$$Y_{validation} = M(X_{validation}) + \epsilon_{validation} \quad (4)$$

$$Y_{holdout} = M(X_{holdout}) + \epsilon_{holdout} \quad (5)$$

$$Y_{test} = M(X_{test}) + \epsilon_{test} \quad (6)$$

It is worth noticing that the model component M is shared among all the sets since we are using the same model to explain the data.

We can define the model predictions as $\hat{Y} = M(X) = f(X|\theta)$ where θ are the parameters of the model. When training a NN on Y_{train} , we are trying to find the model's parameters θ such that $f(X_{train}|\theta)$ minimizes the loss function $L = loss(\hat{Y}_{train}, Y_{train} + \epsilon_{train})$. Specifically for the Train set we have:

$$\theta = \underset{\theta}{\operatorname{argmin}} \operatorname{loss}(f(X_{train}|\theta), Y_{train} + \epsilon_{train}) \quad (7)$$

Since the data explanation equations are explained by the model M and the ϵ noise, there is a compromise between both terms. We theoretically should choose the model M that explains (100%) of set Y . However, in practice, each set Y will have a percentage explained by M and the remaining percentage by its own ϵ when we select the model M with a single criterion that maximizes the performance metric or minimizes a loss function on a single set, e.g., the Train set – explained by Equation 3. It means that the selected model will probably incorporate the ϵ_{train} of that set, which is different from all the other ϵ . This procedure will probably produce an over-fitted model. If we are willing to minimize the loss between Y_{train} and \hat{Y}_{train} , we can state that $(Y_{train} - \hat{Y}_{train}) \rightarrow 0$, which is to say that $\hat{Y} = M(X_{train}) - \epsilon_{train}$. The model M learned how to compensate for the noise ϵ_{train} of the training data. Therefore M can fully explain Y_{train} , also minimizing the loss between Y_{train} and \hat{Y}_{train} as can be seen in the following set of equations:

$$Y_{train} = M(X_{train}) + \epsilon_{train} \quad (8)$$

$$M(X_{train}) = f(X_{train}|\theta) - \epsilon_{train} \quad (9)$$

$$Y_{train} = (f(X_{train}|\theta) - \epsilon_{train}) + \epsilon_{train} \quad (10)$$

$$Y_{train} = f(X_{train}|\theta) \quad (11)$$

The problem occurs when the previously learned model M is applied to unseen data in the test set. The part less affected by ϵ_{test} in the Test set (explained mainly by the theoretical optimal model M term) will perform poorly due to the over-fitted model on the Training set, presenting a high generalization error as follows

$$Y_{test} = M + \epsilon_{test} \quad (12)$$

$$Y_{test} = (f(X_{test}|\theta) - \epsilon_{train}) + \epsilon_{test} \quad (13)$$

To alleviate the problem of over-fitting, we usually use the Validation set constraining the model learning to minimize the loss function to explain the Train set in Equation 14

subjected to also minimize the loss in the Validation set simultaneously as in Equation 15, but without using the Validation data to change the model's parameters.

$$\theta = \operatorname{argmin} \operatorname{loss}(f(X_{\text{train}}|\theta), Y_{\text{train}} + \epsilon_{\text{train}}) \quad (14)$$

subject to:

$$\theta = \operatorname{argmin} \operatorname{loss}(f(X_{\text{validation}}|\theta), Y_{\text{validation}} + \epsilon_{\text{validation}}) \quad (15)$$

When we perform model selection in a multi-criteria approach, e.g., maximizing all the sets, we are implicitly taking into account the noise on every set and trying to decrease the influence of each noise ϵ during the selection procedure such that we can find a model M which mostly explains all the sets used in the multi-criteria decision. It will select models with worse results, but we claim that they are closer to reality since most of the explanation for each set comes from the shared model M instead of a specific noise for a single set. In other words, we are trying to select the model that simultaneously optimizes the performance in the Training, Validation, and Holdout sets. We can understand it as finding M that minimizes $(Y_{\text{train}} - \hat{Y}_{\text{train}})$, $(Y_{\text{validation}} - \hat{Y}_{\text{validation}})$, $(Y_{\text{holdout}} - \hat{Y}_{\text{holdout}})$ simultaneously. In that case we are giving more importance to M term (since it is shared) relative to each set ϵ : $M + \epsilon_{\text{train}} + M + \epsilon_{\text{validation}} + M\epsilon_{\text{test}} = 3M + \epsilon_{\text{train}} + \epsilon_{\text{validation}} + \epsilon_{\text{test}}$. That is why we believe the selected model when optimizing several sets is closer to the theoretical optimal model M since it dilutes the influence of different ϵ across the sets.

The separation into several disjoint subsets to be presented to the NN during the learning process might also be an interesting research avenue to avoid over-fitting and improve model selection since we will probably decrease the influence of individual subset noises during the learning and selection process. However, we will leave that as future work.

2.3.3 LOW COMPLEXITY

The complexity of the model is related to its representational capacity. When increasing the number of parameters in the model, it is natural to understand that it can store more information. However, we usually need more data to avoid over-fitting. If fewer parameters are used, less data is needed to train the model successfully. It is less prone to over-fit while decreasing the bias and increasing the difficulty for the model to fit the data. We can also analyze it through the lenses of the *bias-variance tradeoff* (Hastie et al., 2009). Generally, when model complexity increases, the variance increases, and the bias decreases. Ideally, the model should have low variance and low bias. The best model should have the lowest complexity and the highest performance. It is usually hard to achieve due to the bias-variance trade-off. **The model should have the lowest number of neurons as possible.**

2.3.4 NO PREMATURE EARLY STOPPING

Model initialization is very important to the training dynamics. We can analyze it using the same data, learning algorithm, and the number of epochs, only varying the random initialization of the model. We can divide the initialization into four categories: (i) the model starts in a sweet spot, and no further training is needed (early stopping on epoch 0)

to predict the data (this is very unlikely) correctly; (ii) the model starts in a good region and a few epochs would be sufficient to optimize it, causing an early stopping in the beginning, because of the fast convergence due to a good initial guess (also unlikely); (iii) the model starts in a decent/moderate region and can be optimized for several epochs with late or no early stoppings; (iv) the model starts in an awful place that difficult the learning process causing early stoppings at the beginning of the training phase for various convergence issues such as vanishing/exploding gradients; The third case is more likely to happen since the training using Stochastic Gradient Descent algorithms tends to converge to a minimum due to the error decreasing (and therefore the gradients) throughout the number of epochs.

If a model starts in a bad region (due to the random initialization), it might not be able to be adequately adjusted. Suppose we randomize a model and assess its performance in a dataset without the training phase. In that case, the chances that this model produces bad predictions is much larger than hitting a good spot and correctly predicting the data. Suppose we train using this bad randomly initialized model. In that case, it will probably stop training earlier than a good initialized model unless it randomly started very close to its convergence point in the parameters space (which is also unlikely). Therefore, we expect the **best model to be produced by a more extended training session (without premature early stoppings)**.

We believe that a multi-criteria model selection is better than a single-criterion because when we perform a multi-objective optimization, we usually will have a Pareto Front containing several non-dominated solutions. The solution that we will select using a multi-criteria method will probably not correspond to any extreme case in any dimension, which would be the natural choice in a single-criterion approach. Therefore, we expect that it alleviates the chance of selecting a model that is also fitting the noise of a specific set, consequently avoiding over-fitted models.

In order to select the best model among the 5,600 independently trained (in parallel) MLPs, we have analyzed several model selection policies related to the optimality conditions proposed in this paper.

2.4 Selection Policies

Here we describe the strategies we have used to select the models.

2.4.1 AGGREGATION POLICIES

To rank the models, we can aggregate them by the architecture, locally or globally, or treat them individually. In order to compare the aggregation at different levels, we proposed three aggregation policies to study each optimality condition. Each aggregation group has different approaches on how to aggregate metrics in order to find the best architecture:

- **Individual:** Rankings are treated as individual models' ranks in each run for each dataset, without any aggregation. For each run and dataset, we compare 5,600 individual models to find the one that maximizes the objectives the most.
- **Local:** Rankings are averaged by architecture (8 repetitions of the same architecture, defined by the number of neurons and activation function), containing a mutually ex-

clusive validation set for early stoppings. For each run and dataset, we are comparing between $5,600/8 = 700$ different architectures.

- **Global:** In this aggregation, we averaged the rankings of each eight architectural repetitions of all the 18 runs, leading to $18 * 8 = 144$ architectures trained in different folds but sharing the same Fixed Test set. Once we find the best architecture given all the runs w.r.t. their rankings, we select the model among its eight architectural repetitions for each run. This strategy allows us to understand if we have a "best architecture" regarding a specific dataset and an applied ranking.

2.4.2 RANKING POLICIES

In order to rank models, we proposed several ranking policies that use different strategies to study our optimality conditions.

Even though policies that use the Test metric should be avoided in real life because we would be selecting a model based on the Test set, it serves as an upper bound or an approximation of the theoretical optimal model to compare with and analyze the relationship between other policies that do not include the Test set metric as part of the decision criteria.

- **Single Sets**

Here, we pick the models that maximize the performance metric on a specific set.

- **Train:** Selects the model with the best performance on the train set. Even though it is not a common approach, we can use it to study over-fitting.
- **Validation:** Selects the model with the best performance on their respective validation set that was used to early stop the model training.
- **Holdout:** Selects the model with the best performance on the holdout set.
- **Test:** Selects the model with the best performance on the Test set for comparison purposes.

Local Validation is the most common approach, where the average performance of the same model architecture is calculated on the validation set. In our case, we calculate the average of the eight repetitions trained using a mutually exclusive validation set. Once we have defined the architecture, we select the model with the best performance on its validation set.

We argue that ML models contain several important intrinsic aspects usually neglected when performing the model selection. Summarizing the model and its idiosyncrasies by naively treating model selection as a single-criterion decision task when we only look into a specific set performance metric such as Validation or Holdout accuracy is probably an oversimplified and sub-optimal approach. Therefore, we propose that model selection should be a multi-criteria procedure task and use a combination of properties to rank the models. We have tried to maximize the *performance metric* and the *number of epochs*, and minimize the *number of neurons*, simultaneously. We used Multi-Criteria Decision-Making (MCDM)

algorithms (Triantaphyllou, 2000; Aruldoss et al., 2013; Salabun et al., 2020) to rank the models. Specifically, a straightforward and known algorithm is the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) (Tzeng and Huang, 2011). In this algorithm, we will select the model that minimizes the geometric distance from the Positive Ideal Solution (PIS) and simultaneously maximize the geometric distance from the Negative Ideal Solution (NIS). We can understand the PIS and NIS as the best possible model (maximum metrics, one neuron, maximum stopped epoch) and the worst possible model (minimum metrics, 100 neurons, minimum stopped epoch), respectively. After ranking the models, we apply a Pareto Dominance criteria filter to remove dominated solutions and select the best-ranked model among the non-dominated Pareto solutions. We relied on the PyMCDM library (Shekhovtsov, 2022) to use the TOPSIS algorithm.

Since the MCDM methods create a single scalar rank based on several variables, it will intrinsically incorporate a trade-off or compromise analysis during its execution. Consequentially, we will not be selecting the best performance for a single optimized variable, alleviating the issue of over-searching. The MCDM methods can use different weight scheme combinations and the vectors' normalization.

To improve the paper's readability, we are naming TOPSIS policies after the first letter of what we include in its multi-criteria.

- **High Generalization Performance**

Both Holdout and Test performances should be maximized simultaneously.

- **THT**: TOPSIS maximizing Holdout and Test metrics.

- **High Robustness**

The Train and Validation sets also need to be maximizes.

- **TTVH**: TOPSIS maximizing Train, Validation, and Holdout metrics
- **TTVHT**: TOPSIS maximizing Train, Validation, Holdout, and Test metrics

- **Low Complexity**

We additionally provide the number of neurons to be minimized in the TOPSIS.

- **TTVHN**: TOPSIS maximizing Train, Validation, Holdout, and minimizing the Number of Neurons.
- **TTVHTN**: TOPSIS maximizing Train, Validation, Holdout, Test, and minimizing the Number of Neurons.

- **No Premature Early Stopping**

We additionally provide the Number of Epochs to be maximized (E) in the TOPSIS. For comparison purposes, we also tried to minimize the Number of Epochs (B – Begin of the training).

- **TTVHNE**: TOPSIS maximizing Train, Validation, Holdout, Number of Epochs, and minimizing the Number of Neurons.
- **TTVHNB**: TOPSIS maximizing Train, Validation, Holdout, and minimizing the Number of Neurons and Number of Epochs (Begin of the train) to comparisons purposes.

- **TTVHTNE**: TOPSIS maximizing Train, Validation, Holdout, Test, Number of Epochs, and minimizing the Number of Neurons.
- **TTVHTNB**: TOPSIS maximizing Train, Validation, Holdout, Test, and minimizing the Number of Neurons and Number of Epochs (Begin of the train) for comparisons purposes.

The policies are using the number of neurons as a tiebreaker (models with fewer neurons are preferred) in case models have the same metric being ordered.

As the splits are sampled so that the statistical characterization should be the same, we claim that the best model is not the model that only maximizes the Test set performance metric. However, it also needs a slight disagreement regarding the performance along the Train, Validation, and Holdout sets. If we select the model solely based on the Test performance, the model might be over-fitting in the Test set because its performance might be much higher than on the other sets. Therefore, we are considering good models that balance the performance on all the individual sets. We suggest using the TTVH policy if we want to maximize the accuracy and TTVHN policy to still have good accuracy but focusing on smaller models since both of the policies does not look into the Test set and the No Premature Early Stopping criteria seems to need further adjustments regarding the weight that this condition should be given.

3. Experiments

3.1 Computing Environment

A Machine with 16GB RAM, 11GB NVIDIA GTX 1080 Ti, and an I7-8700K CPU @ 3.7GHz containing 12 threads were used to perform the simulations. All the code was written using PyTorch (Paszke et al., 2019).

3.2 Number of Neurons

We have created MLPs containing from 1 to 100 neurons in their hidden layer. It gives us 100 different architectures.

3.3 Activation Functions

We have used seven activation functions (Identity, GELU, LeakyReLU, ReLU, SeLU, Sigmoid, Tanh). Combined with the number of neuron variations, we get $100 * 7 = 700$ different architectures.

3.4 Splitting Strategy

We have assessed our proposal with the Similarity-Based Stratified Splitting (SBSS) (Farias et al., 2020) and without it – an ordinal stratified 10-fold – with one fold being a fixed test split, another one a fixed holdout split, and the remaining eight splits being used as training and validation splits. We have used 10 splits because it is one of the most common k-fold setup used in machine learning.

3.5 Datasets

We assessed the proposed selection policies and optimality conditions in several situations, such as many features and labels, a low number of samples, and dataset imbalances. We have used 14 datasets from UCI (Dua and Graff, 2017) listed in Table 1. We calculated the Imbalance of each dataset by adapting the suggestion in (Romano, 2016) according to Eq. 16, resulting in 0 when the dataset is balanced and 1 otherwise.

$$Imbalance = 1 - \frac{\sum_{i=1}^k \frac{c_i}{n} \log(\frac{c_i}{n})}{\log(k)} \quad (16)$$

where n is the number of samples; k is the number of labels, and c_i is the number of samples in label i .

Dataset	# Features	# Labels	# Samples	Imbalance
balance-scale	4	3	625	0.17
blood-transfusion-service-center	4	2	748	0.21
car	6	4	1728	0.40
diabetes	8	2	768	0.07
tic-tac-toe	9	2	958	0.07
ilpd	10	2	583	0.14
vowel	12	11	990	0.00
australian	14	2	690	0.01
climate-model-simulation-crashes	18	2	540	0.58
vehicle	18	4	846	0.00
credit-g	20	2	1000	0.12
wdbc	30	2	569	0.05
ionosphere	34	2	351	0.06
satimage	36	6	6430	0.04
libras move	90	15	360	0.00
lsvt	310	2	126	0.08

Table 1: Datasets used as benchmarks

We selected datasets with different properties such as number of features, labels, samples and imbalance levels. This creates different challenges for each dataset that are useful to simulate several situations which our proposal may be exposed when working with real world data.

4. Results and Discussion

In this section we present the results and discussions of our experiments.

Each point in Figure 2 is the average of 8 models with the same architecture (combination of the number of neurons and activation function) in a specific run for each dataset. As expected, we can observe a high correlation in the distribution of Train, Holdout, and Test accuracies.

In Figure 3 we have created the Pareto Front using the Holdout and Test sets performances considering all the models for each dataset. Pareto Front is a concept that tries to

MULTI-CRITERIA MACHINE LEARNING MODEL SELECTION

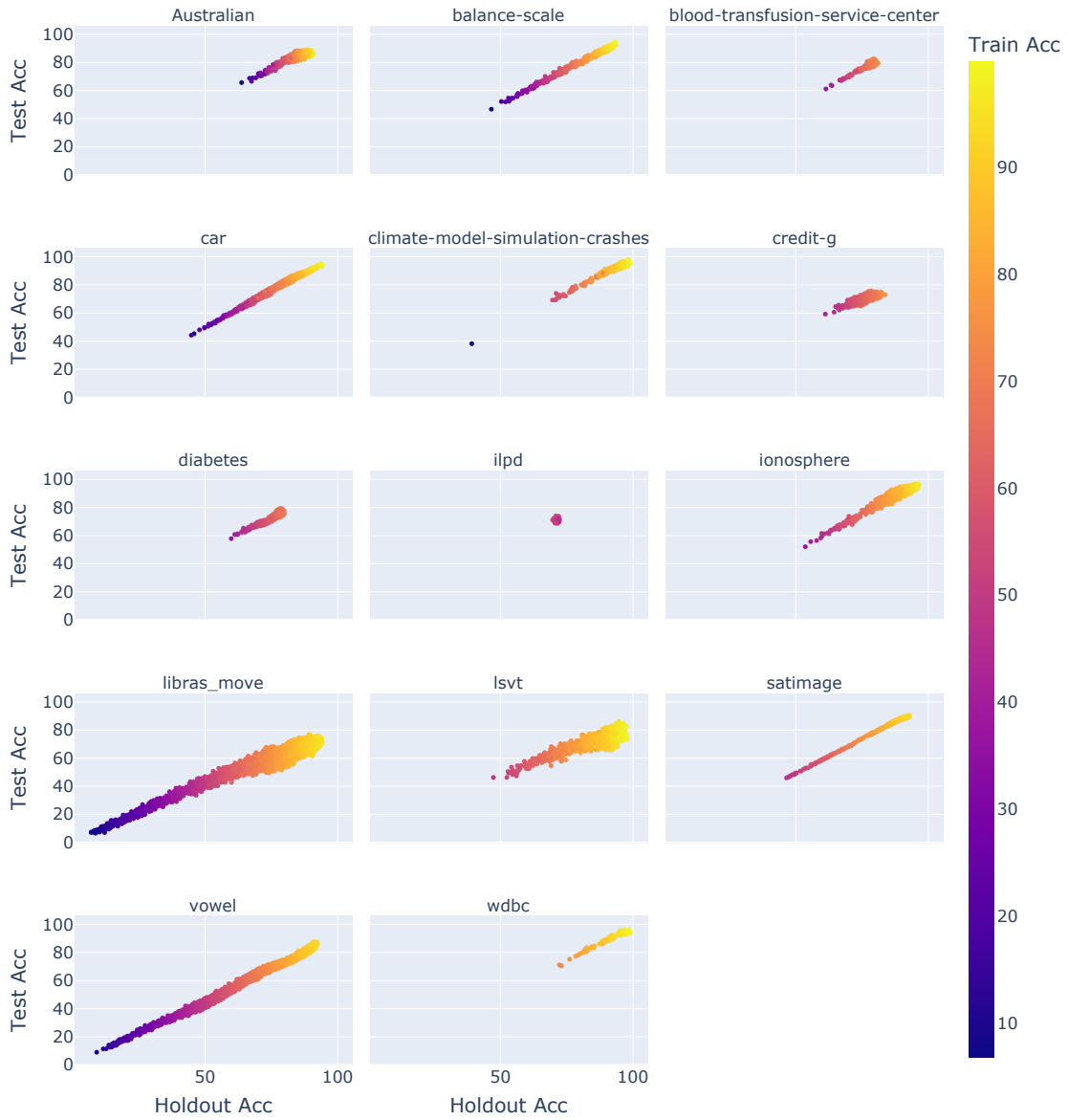


Figure 2: Average Train, Holdout, and Test accuracies grouped by run, dataset, and architecture.

capture trade-offs between variables being used in a multi-objective optimization process. Since we are using two variables in this plot (Holdout and Test accuracies), we create a set of models such that there are no other models that can improve some variable without reducing others. Neither the Holdout nor the Test set participates during the model training. Therefore, from the model standpoint, those two specific sets can be considered as two proxies of real data that the model will probably see during the real operation.

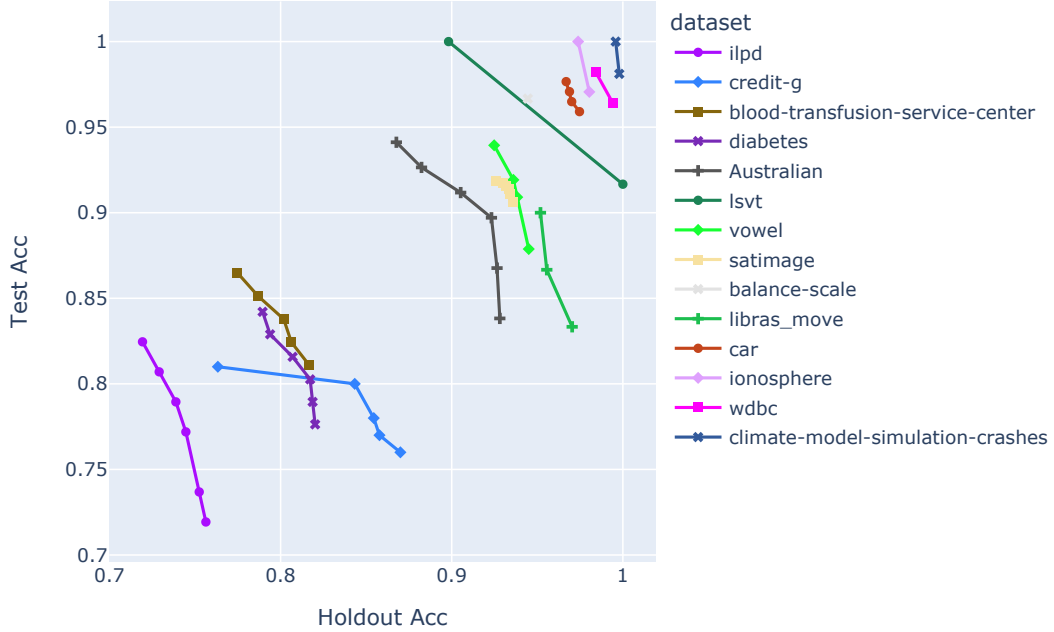


Figure 3: Pareto front for each dataset.

As we can see, if the model that performs best in the Holdout set is used, it is not guaranteed that the model will have the best performance in the Test set. Based on this finding, we can state that selecting a model that only maximizes the Holdout set will not give us the best model to be used as a production model.

4.1 SBSS vs. Stratified 10-fold

A comparison between a 10-fold SBSS and an ordinary Stratified 10-fold is presented in Table 2. It is easy to realize that the 10-fold SBSS consistently outperforms the ordinary Stratified 10-fold splitting. Therefore, the subsequent analysis will be done only using SBSS.

The following results tables contain the average and standard deviations in parenthesis for each Policy for the selected models w.r.t. the Number of Neurons (# Neurons - lower is better), Number of trained Epochs due to Early Stopping (# Epochs - higher is better), Accuracies for Train, Validation, Holdout, and Test sets (higher is better), and the Dis-

MULTI-CRITERIA MACHINE LEARNING MODEL SELECTION

Splitting Strategy	Accuracy \uparrow			
	Train	Validation	Holdout	Test
10-fold SBSS	83.62 (12.72)	83.68 (11.24)	84.28 (11.29)	80.97 (12.45)
Stratified 10-fold	73.94 (16.62)	76.63 (14.72)	76.48 (14.42)	74.87 (15.73)

Table 2: Accuracy average and standard deviation (in parenthesis) for all the 5,600 generated models, 18 runs and 14 datasets, totaling 1,411,200 models for each Splitting Strategy.

agreement (smaller is better) as the average of the absolute individual difference between Train-Validation, Holdout-Test and All (averaged absolute difference for each combination of paired sets). Wilcoxon statistical significance test tables contain three symbols to represent: (\blacktriangle) row distribution significantly larger than column; (\equiv) row distribution not significantly different from the column; (∇) row distribution significantly smaller than the column. If the variable is Accuracy, policies that produce larger values are better. On the other hand, if the variable is Disagreement, smaller values are better. The Summary contains the sum of $\blacktriangle(1)$, $\equiv(0)$, $\nabla(-1)$ for each policy in rows.

4.2 High Generalization Performance Results

The results for each aggregation and ranking Policy are presented in Table 3 and are useful for understanding the High Generalization Performance condition. Also, the Wilcoxon Statistical Tests regarding the Test performance can be seen in Table 4.

We can analyze the architectural generalization by examining the three grouping strategies with the Test policy. The average number of neurons for Test policy with groupings Individual, Local and Global are 42.8, 68.5, 85; while the accuracies are 90.61, 89.05, 87.69, with Individual (win) – Local, Local (win) – Global. If we use Holdout instead of Test policy, we have the number of neurons 76.41, 86.41, 87.36 and Test accuracy 85.54, 85.01, 85.48. The Holdout Wilcoxon tests for the three groups are all equivalent. The Global grouping (for each dataset, we have chosen a single architecture) led to similar performances if we used the Local grouping (for each run, we choose the best model given the best-averaged architecture) or the Individual grouping (each run will have to treat each model individually, without any architecture grouping). However, Individual grouping could provide equivalent Test performances but with smaller networks.

It seems that analyzing the best architecture for a given data combination (the Individual grouping) is preferable. It contributes to our hypothesis that freezing the NN architecture is not the best approach. The models will experience different initializations and be exposed to slightly different training data. Therefore, they will experience different error landscapes, subjected to their learning dynamics.

We did not need to train several models to select good ones. The Individual–Holdout does not aggregate performances by the architecture, i.e., it does not use the concept of “best architecture” for a given dataset, differently from the Local–Holdout and Global–Holdout. Still, the Individual–Holdout is performing equivalently to the Local–Holdout and

FARIAS, LUDERMIR AND BASTOS-FILHO

Policy		# Neurons ↓	# Epochs ↑	Accuracy ↑				Disagreement ↓		
				Train	Validation	Holdout	Test	Train-Validation	Holdout-Test	All
Individual	Train	73.47 (29.13)	86.02 (24.6)	92.45 (7.72)	87.47 (7.8)	91.28 (7.71)	85.06 (9.5)	5.03 (4.21)	6.4 (6.84)	4.66 (4.05)
	Validation	47.02 (34.84)	62.99 (35.21)	87.85 (8.8)	91.86 (6.85)	88.97 (7.98)	84.25 (9.78)	4.26 (3.96)	5.21 (6.76)	4.38 (3.83)
	Holdout	76.41 (27.5)	84.79 (25.13)	92.08 (7.92)	89.94 (7.79)	91.76 (7.66)	85.54 (9.29)	2.77 (2.66)	6.37 (6.53)	3.94 (3.51)
	Test	42.8 (33.91)	63.72 (34.72)	87.42 (8.8)	87.07 (7.64)	87.68 (7.98)	90.61 (7.05)	2.67 (3.25)	4.29 (3.51)	3.13 (2.41)
	THT	75.5 (24.61)	78.21 (27.19)	91.22 (8.36)	89.05 (7.89)	90.9 (8.07)	89.65 (7.34)	2.83 (2.67)	3.52 (3.26)	2.67 (2.0)
Local	Train	85.19 (18.79)	80.42 (26.98)	91.75 (8.21)	88.0 (7.93)	90.95 (8.0)	85.21 (9.43)	3.86 (3.6)	5.98 (7.27)	4.12 (4.09)
	Validation	80.45 (20.88)	62.89 (33.46)	89.07 (8.9)	90.8 (7.08)	89.65 (8.21)	85.14 (9.38)	2.68 (2.69)	4.94 (6.1)	3.52 (3.19)
	Holdout	86.41 (17.5)	78.35 (27.47)	91.52 (8.01)	89.37 (7.6)	91.2 (7.71)	85.01 (9.6)	2.76 (2.76)	6.36 (7.27)	3.94 (3.97)
	Test	68.52 (30.98)	61.08 (34.2)	88.61 (9.19)	88.13 (7.81)	88.77 (8.51)	89.05 (7.49)	2.28 (2.62)	3.46 (3.28)	2.48 (2.11)
	THT	82.75 (19.23)	68.67 (30.14)	90.56 (8.51)	88.58 (7.57)	90.28 (8.03)	88.76 (7.59)	3.26 (3.32)	3.58 (3.79)	2.86 (2.4)
Global	Train	91.86 (11.95)	75.19 (27.65)	91.43 (8.26)	87.85 (7.9)	90.69 (8.01)	85.34 (9.22)	3.73 (3.9)	5.55 (6.55)	3.85 (3.87)
	Validation	92.29 (13.52)	61.85 (29.62)	89.17 (8.28)	90.13 (7.42)	89.51 (7.95)	84.91 (9.8)	1.96 (1.89)	5.19 (6.77)	3.33 (3.48)
	Holdout	87.36 (23.97)	75.85 (26.96)	90.99 (8.93)	88.78 (7.87)	90.83 (8.02)	85.48 (9.18)	3.08 (2.75)	5.49 (6.39)	3.67 (3.47)
	Test	85.0 (21.18)	62.13 (32.4)	89.37 (8.82)	88.26 (8.03)	89.24 (8.46)	87.69 (8.17)	2.24 (3.15)	3.4 (4.4)	2.42 (2.74)
	THT	89.36 (12.43)	69.11 (30.22)	90.04 (10.02)	88.45 (7.74)	90.21 (8.2)	87.44 (8.42)	3.59 (3.66)	3.9 (4.9)	3.18 (2.9)

Table 3: Results for High Generalization Performance policies.

		Individual			Local			Global			Summary
		Holdout	Test	THT	Holdout	Test	THT	Holdout	Test	THT	Test Accuracy ↑
Individual	Holdout	≡	▽	▽	≡	▽	▽	≡	▽	▽	-6
	Test	▲	≡	▲	▲	▲	▲	▲	▲	▲	8
	THT	▲	▽	≡	▲	▲	▲	▲	▲	▲	6
Local	Holdout	≡	▽	▽	≡	▽	▽	≡	▽	▽	-6
	Test	▲	▽	▽	▲	≡	▲	▲	▲	▲	4
	THT	▲	▽	▽	▲	▽	≡	▲	▲	▲	2
Global	Holdout	≡	▽	▽	≡	▽	▽	≡	▽	▽	-6
	Test	▲	▽	▽	▲	▽	▽	▲	≡	▲	0
	THT	▲	▽	▽	▲	▽	▽	▲	▽	≡	-2

Table 4: Wilcoxon Statistical Significance Test comparisons for High Generalization policies for different aggregations w.r.t. Test Accuracy.

Global-Holdout, according to Table 4. We could select good models even when we have not used 10-fold cross-validation to aggregate architectures and decide the “best architecture”. Therefore, instead of repeating eight times the same architecture, we could always use different architectures using the same amount of memory and computational processing and increase the diversity of models by a factor of 7 (instead of $5,600/8 = 700$, we would have 5,600 unique architectures). As we did not need to find the “best architecture”, we could use the time to run 10-fold cross-validation in other ways, such as varying non-architectural hyper-parameters, e.g., the learning rate or the maximum number of epochs. Using different validation sets for the same architectures in a large pool of ANN might contribute to still finding good models. The training data change in the same run, and a specific combination of folds composing the training data might create better learning dynamics, producing exciting models.

When the Individual-Test policy is used, the Test accuracy is much higher than the Holdout accuracy. At the same time, the Train and Validation accuracies are minor if compared to the Test accuracy. Likewise, if the Individual-Holdout policy is used, the Test accuracy is smaller than the Holdout accuracy. Should we expect the model’s performance in the real world to be closer to the Holdout or Test set? When we select the model that indiscriminately maximizes the Holdout or the Test metric (Holdout and Test policies), we are probably overestimating one metric and underestimating the other while potentially over-fitting the model. It can also be seen as a possible sign over-searching (Ying, 2019). Over-searching is a common problem when the hypothesis space of ML algorithms grows, increasing the over-fitting probability (Thornton et al., 2013). Over-searching is also a counterintuitive problem because we are trying to find the best hyper-parameters of the model, and the more points we know about the error landscape (more models trained and tested), the more we increase the chance of facing over-searching issues. In other words, the more information we have about our function to be optimized, the more we increase the chance of selecting a bad set of hyper-parameters. We argue that over-searching can be alleviated using a multi-criteria approach to select the models. We also advocate that the Disagreement between the Holdout-Test sets deserves attention when selecting or estimating the model’s performance in the real world.

From Table 4, we can realize that Individual-THT produces smaller NN and is better than both Local-THT and Global-THT. It also decreases the Disagreement between Holdout-Test and All if we compare them within the Individual group. It is worth mentioning that we have not tried to minimize the Disagreement measurements during the THT policy explicitly. However, this behavior had emerged while trying to maximize both Holdout and Test metrics.

Since the performance of the Individual-THT policy for Holdout and Test sets has the smallest Disagreement-Holdout-Test and also Disagreement-All, if compared to Individual-Holdout and Individual-Test policies, we could say that the real-world performance estimations would be more reliable than picking the model by individually maximizing the performance of each set using Individual-Holdout or Individual-Test (that is probably over-estimated in the optimized set and underestimated in the other).

It is also important to highlight that even though THT is deciding based on the Holdout and Test set metrics, the Individual-THT also increased both Train and Validation accuracy if compared against deciding only by maximizing the Test function. It contributes to

our hypothesis that the models that have similar sets performance would be more robust, mentioned in section 2.3.2.

Although seen as a bad practice due to the potential over-fitting, the Train policy is probably not catastrophic in our case due to the early stopping usage. Interestingly, the Train and Holdout approach contains very similar metrics. It might be because the Train set contains more data (70%) than the Holdout set (10%); therefore, it estimates better, and we are mitigating over-fitting by regularization and early stopping during training.

Since the results using Individual Aggregation are reasonable, we will not analyze the Local and Global aggregations in the subsequent results.

4.3 High Robustness

The results to understand the High Robustness Optimality Condition are presented in Table 5. Also, the Wilcoxon Statistical Tests regarding the Test performance can be seen in Table 6, and the Disagreement All is presented in Table 7.

Policy	# Neurons ↓	# Epochs ↑	Accuracy ↑				Disagreement ↓		
			Train	Validation	Holdout	Test	Train-Validation	Holdout-Test	All
Holdout	76.41	84.79	92.08	89.94	91.76	85.54	2.77	6.37	3.94
	(27.5)	(25.13)	(7.92)	(7.79)	(7.66)	(9.29)	(2.66)	(6.53)	(3.51)
TTVH	75.98	82.64	91.49	91.1	91.5	85.42	2.26	6.26	3.84
	(27.06)	(26.28)	(8.4)	(7.09)	(7.91)	(9.39)	(2.36)	(6.91)	(3.59)
Test	42.8	63.72	87.42	87.07	87.68	90.61	2.67	4.29	3.13
	(33.91)	(34.72)	(8.8)	(7.64)	(7.98)	(7.05)	(3.25)	(3.51)	(2.41)
THT	75.5	78.21	91.22	89.05	90.9	89.65	2.83	3.52	2.67
	(24.61)	(27.19)	(8.36)	(7.89)	(8.07)	(7.34)	(2.67)	(3.26)	(2.0)
TTVHT	76.5	77.75	91.05	90.47	91.05	88.88	2.52	3.42	2.48
	(24.32)	(27.45)	(8.5)	(7.21)	(8.0)	(7.78)	(2.45)	(3.53)	(2.11)

Table 5: Results for High Robustness policies.

Policy	Holdout	TTVH	Test	THT	TTVHT	Summary Test Accuracy ↑
Holdout	≡	≡	▽	▽	▽	-3
TTVH	≡	≡	▽	▽	▽	-3
Test	▲	▲	≡	▲	▲	4
THT	▲	▲	▽	≡	▲	2
TTVHT	▲	▲	▽	▽	≡	0

Table 6: Wilcoxon Statistical Significance Test comparisons for High Robustness policies w.r.t. Test Accuracy.

We argue that the difference of the performance metrics for each set is inversely proportional to its robustness. Let us compare the policies that do not look into the Test set (TTVH and Holdout). We can see that TTVH could select models with statistically equivalent results regarding the Test performance, as depicted in Table 6. At the same time, from Table 7 it was able to significantly decrease the Disagreement if compared to the Holdout policy. Therefore, instead of looking only into the Holdout performance metric, we should

MULTI-CRITERIA MACHINE LEARNING MODEL SELECTION

Policy	Holdout	TTVH	Test	THT	TTVHT	Summary All Disagreement ↓
Holdout	≡	▲	▲	▲	▲	4
TTVH	▽	≡	≡	▲	▲	1
Test	▽	≡	≡	▲	▲	1
THT	▽	▽	▽	≡	▲	-2
TTVHT	▽	▽	▽	▽	≡	-4

Table 7: Wilcoxon Statistical Significance Test comparisons for High Robustness policies w.r.t. All Disagreement.

use the TTVH approach since it produces statistically equivalent performance metrics and still decreases the Disagreement between the sets’ performances. This probably indicates a more robust selection.

We have generally been taught that the best model is the one that maximizes the Test performance metric. This is evident if we consider how state-of-the-art methods are traditionally benchmarked: very often only analyzing if the Test performance metric is better than previous methods applied to the same dataset. We argue that the performance metrics of the other sets must also be involved for a complete evaluation. If we recall the Equation 6, $Y_{test} = M(X_{test}) + \epsilon_{test}$, it might be the case that we select the model that maximizes $M(X_{test}) + \epsilon_{test}$ on the Test data, providing a probably over-estimation for the model performance during production, instead of what we really want: the model that maximizes only the $M(X_{test})$ term, since when using the model in production, we will probably have a $\epsilon_{production}$ different from our ϵ_{test} .

Comparing the policies that include the Test metric in the decision, the Test policy should be the upper bound policy regarding the Test metric if we use the . However, as we mentioned, it is probably over-estimated due to inherent noise in the Test set. Therefore we need to be cautious about taking this as the desired model. To corroborate this idea, we can see from the THT policy that the Train and Validation metrics improved compared to the Test policy. However, we have not explicitly optimized for that. On the other hand, the TTVHT model (which also optimizes the Train and Validation metrics) would probably be our target model since it tries to maximize the performance over the Train, Validation, Holdout, and Test sets. As a consequence of the better performance equilibrium in the TTVHT and THT compared to the Test policy, we can also see that the Disagreement was significantly decreased in Table 7.

If we compare the TTVHT to the TTVH approach, we can see that the Number of Neurons, Train, Validation, and Holdout set performances are similar. Let us consider only the subset of models which deliver similar performance to the TTVH values (similar to freezing the previously mentioned metrics). We have a group of models that varies the Test metric. The TTVHT would be an approximation to this group’s best model, but TTVH could not select it. It probably means that we can include other attributes that describe the learning dynamics of the model in the TOPSIS decision process, such as the difference between the initial and final performances and weight regularization values, to create better model selectors. It opens many possibilities to improve the multi-criteria model selection procedure.

4.4 Low Complexity

To analyze the Low Complexity Optimality Condition, we are trying to minimize the number of neurons in a model. The results for each policy containing and without the Number of Neurons as decision criteria are presented in Table 8. The Wilcoxon statistical test results of the same set of policies regarding the Test and Disagreement are respectively presented in Table 9 and Table 10

Policy	# Neurons ↓	# Epochs ↑	Accuracy ↑				Disagreement ↓		
			Train	Validation	Holdout	Test	Train-Validation	Holdout-Test	All
TTVH	75.98 (27.06)	82.64 (26.28)	91.49 (8.4)	91.1 (7.09)	91.5 (7.91)	85.42 (9.39)	2.26 (2.36)	6.26 (6.91)	3.84 (3.59)
TTVHN	10.27 (9.27)	78.08 (28.3)	87.59 (7.94)	88.36 (7.13)	87.83 (7.66)	82.9 (9.96)	2.14 (1.93)	5.54 (7.19)	3.54 (3.74)
THT	75.5 (24.61)	78.21 (27.19)	91.22 (8.36)	89.05 (7.89)	90.9 (8.07)	89.65 (7.34)	2.83 (2.67)	3.52 (3.26)	2.67 (2.0)
THTN	8.75 (7.24)	78.35 (27.0)	86.78 (7.93)	85.97 (8.22)	86.7 (7.8)	86.9 (7.94)	2.37 (2.65)	2.75 (2.11)	2.2 (1.6)
TTVHT	76.5 (24.32)	77.75 (27.45)	91.05 (8.5)	90.47 (7.21)	91.05 (8.0)	88.88 (7.78)	2.52 (2.45)	3.42 (3.53)	2.48 (2.11)
TTVHTN	11.37 (9.44)	77.48 (27.79)	87.54 (7.92)	88.14 (7.13)	87.76 (7.61)	86.44 (8.35)	2.09 (1.93)	3.15 (3.58)	2.3 (1.93)

Table 8: Results for Low Complexity policies.

Policy	TTVH	TTVHN	THT	THTN	TTVHT	TTVHTN	Summary Test Accuracy ↑
TTVH	≡	▲	▽	▽	▽	▽	-3
TTVHN	▽	≡	▽	▽	▽	▽	-5
THT	▲	▲	≡	▲	▲	▲	5
THTN	▲	▲	▽	≡	▽	▲	1
TTVHT	▲	▲	▽	▲	≡	▲	3
TTVHTN	▲	▲	▽	▽	▽	≡	-1

Table 9: Wilcoxon Statistical Significance Test comparisons for Low Complexity policies w.r.t. Test Accuracy.

Policy	TTVH	TTVHN	THT	THTN	TTVHT	TTVHTN	Summary All Disagreement ↓
TTVH	≡	▲	▲	▲	▲	▲	5
TTVHN	▽	≡	▲	▲	▲	▲	3
THT	▽	▽	≡	▲	▲	▲	1
THTN	▽	▽	▽	≡	▽	≡	-4
TTVHT	▽	▽	▽	▲	≡	▲	-1
TTVHTN	▽	▽	▽	≡	▽	≡	-4

Table 10: Wilcoxon Statistical Significance Test comparisons for Low Complexity policies w.r.t. All Disagreement.

The models decreased the average accuracy when adding the number of neurons in the multi-criteria equation. However, it largely dropped the number of neurons necessary to

encode the knowledge of each dataset. It is essential to create simpler models since they tend to generalize better because they make fewer assumptions about the data it is trying to learn.

It is worth mentioning that TOPSIS allows us to set different weights for each objective, even though we have used the same weight for all objectives. Tuning this specific TOPSIS parameter will lead to different results based on the importance we are willing to give for each objective.

4.5 No Premature Early Stopping

We have included the number of epochs in the TOPSIS decision-making process to analyze the importance of the Early Stopping epoch. The ranking policies end with E (stopping close to the End of training) or B (stopping close to the Begin of the training).

As we can see from Table 11, the ranking policies that maximize the number of trained epochs (ends with E) consistently outperform their counterparts that minimize the number of trained epochs (ends with B) for all the Aggregation and Ranking policies. It shows that how long the model was trained is also important to consider during the model selection. Even though when directly comparing TTVHN vs. TTVHNE (that includes the maximization of trained epochs), we can see from Table 12 that the performance on the Test set is equivalent as well as the Disagreement is not significantly different. Since no statistically better results were found by maximizing the Number of trained Epochs, but it still affects the model selection when comparing B vs. E policies, it might be the case that we need to tweak better the weight for this specific criteria to mostly avoid premature early stopping (B) instead of looking for late or no early stopping at all (E), which can also contain models that have not converged yet at the final epochs. This probably would be better used as a filter to avoid premature early stopping (B), but not be simultaneously maximized.

Policy	# Neurons ↓	# Epochs ↑	Accuracy ↑				Disagreement ↓		
			Train	Validation	Holdout	Test	Train-Validation	Holdout-Test	All
TTVHN	10.27	78.08	87.59	88.36	87.83	82.9	2.14	5.54	3.54
	(9.27)	(28.3)	(7.94)	(7.13)	(7.66)	(9.96)	(1.93)	(7.19)	(3.74)
TTVHNB	13.8	17.94	83.41	85.37	84.01	80.48	2.68	4.63	3.34
	(12.23)	(20.68)	(7.61)	(7.33)	(7.38)	(10.26)	(2.17)	(5.87)	(3.24)
TTVHNE	10.77	89.83	87.53	87.79	87.66	82.96	2.2	5.33	3.41
	(9.38)	(24.46)	(7.96)	(7.08)	(7.66)	(9.91)	(2.06)	(6.87)	(3.64)
THTN	8.75	78.35	86.78	85.97	86.7	86.9	2.37	2.75	2.2
	(7.24)	(27.0)	(7.93)	(8.22)	(7.8)	(7.94)	(2.65)	(2.11)	(1.6)
THTNB	11.17	16.53	81.83	83.49	82.49	82.63	2.39	3.08	2.44
	(8.83)	(20.6)	(8.28)	(8.61)	(8.14)	(9.74)	(2.7)	(3.03)	(2.04)
THTNE	8.49	89.79	86.69	85.53	86.61	86.14	2.53	2.62	2.19
	(6.9)	(24.18)	(8.26)	(8.19)	(7.89)	(8.12)	(2.84)	(2.69)	(1.88)
TTVHTN	11.37	77.48	87.54	88.14	87.76	86.44	2.09	3.15	2.3
	(9.44)	(27.79)	(7.92)	(7.13)	(7.61)	(8.35)	(1.93)	(3.58)	(1.93)
TTVHTNE	11.89	89.55	87.53	87.85	87.7	85.9	2.04	3.39	2.39
	(10.47)	(24.41)	(7.93)	(7.19)	(7.66)	(8.3)	(2.11)	(4.05)	(2.23)
TTVHTNB	14.91	19.09	83.78	85.7	84.38	83.37	2.59	3.3	2.62
	(12.97)	(21.61)	(7.08)	(6.92)	(6.89)	(8.09)	(2.4)	(3.41)	(2.04)

Table 11: Results for No Premature Early Stopping policies.

Policy	TTVHN	TTVHNB	TTVHNE	THTN	THTNB	THTNE	TTVHTN	TTVHTNE	TTVHTNB	Summary Test Accuracy \uparrow
TTVHN	\equiv	\blacktriangle	\equiv	∇	\equiv	∇	∇	∇	\equiv	-3
TTVHNB	∇	\equiv	∇	∇	∇	∇	∇	∇	∇	-8
TTVHNE	\equiv	\blacktriangle	\equiv	∇	\equiv	∇	∇	∇	\equiv	-3
THTN	\blacktriangle	\blacktriangle	\blacktriangle	\equiv	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	8
THTNB	\equiv	\blacktriangle	\equiv	∇	\equiv	∇	∇	∇	∇	-4
THTNE	\blacktriangle	\blacktriangle	\blacktriangle	∇	\blacktriangle	\equiv	∇	\equiv	\blacktriangle	3
TTVHTN	\blacktriangle	\blacktriangle	\blacktriangle	∇	\blacktriangle	\blacktriangle	\equiv	\blacktriangle	\blacktriangle	6
TTVHTNE	\blacktriangle	\blacktriangle	\blacktriangle	∇	\blacktriangle	\equiv	∇	\equiv	\blacktriangle	3
TTVHTNB	\equiv	\blacktriangle	\equiv	∇	\blacktriangle	∇	∇	∇	\equiv	-2

Table 12: Wilcoxon Statistical Significance Test comparisons for No Premature Early Stopping policies w.r.t. Test Accuracy.

Policy	TTVHN	TTVHNB	TTVHNE	THTN	THTNB	THTNE	TTVHTN	TTVHTNE	TTVHTNB	Summary All Disagreement \downarrow
TTVHN	\equiv	\equiv	\equiv	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	6
TTVHNB	\equiv	\equiv	\equiv	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	6
TTVHNE	\equiv	\equiv	\equiv	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	6
THTN	∇	∇	∇	\equiv	\equiv	\equiv	\equiv	\equiv	∇	-4
THTNB	∇	∇	∇	\equiv	\equiv	\blacktriangle	\equiv	\equiv	∇	-3
THTNE	∇	∇	∇	\equiv	∇	\equiv	\equiv	\equiv	∇	-5
TTVHTN	∇	∇	∇	\equiv	\equiv	\equiv	\equiv	\equiv	∇	-4
TTVHTNE	∇	∇	∇	\equiv	\equiv	\equiv	\equiv	\equiv	∇	-4
TTVHTNB	∇	∇	∇	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	\blacktriangle	\equiv	2

Table 13: Wilcoxon Statistical Significance Test comparisons for No Premature Early Stopping policies w.r.t. All Disagreement.

5. Conclusions

In this work, we propose to use several criteria of a machine learning model, in this specific case, Neural Networks, to perform model selection (multi-criteria model selection) instead of using the most common approach of a single criterion. We have shown that fixing the number of Neurons of an ANN, primarily done in ANN training methodology, for a specific dataset does not produce the best possible models. Using a flexible architecture usually leads to better results. Also, we empirically demonstrated that over-fitting and over-searching could be mitigated by performing a multi-criteria model selection procedure instead of a single criterion when deciding from a very large pool of candidates because it alleviates the metrics maximization on a specific set that the model usually is also explaining the noise. When a multi-criteria approach is used, we are probably diluting the risk of noise fitting through all the sets since we expect the noise should not be modeled, and using more data from different sets that agree with each other during the model selection, we increase our confidence on the model assessment or performance estimation.

We have also defined optimality conditions that we desire to have in theoretical models and empirically demonstrated that those conditions seem important during model selection. The oracle or any model selected solely based on a single performance for a set is usually over-optimistic and probably over-fitted on that specific set due to the over-searching issue and deserves attention if one would consider them as the best targets during model selection. We believe this work might lead to different research directions since it gives a different perspective and justifies why model selection that only maximizes specific performance metrics tends not to be the best approach.

6. Future Works

We would like to evaluate our policies within AutoML algorithms, which usually only perform model selection based on a single objective. In this process, it might also be interesting to build deep neural networks layer-by-layer using ParallelMLPs to train several candidates and use TOPSIS to rank and choose the current layer, appending layer by layer. Using other learning dynamic attributes and tweaking the weights for each criterion might produce better model selections. We intend to try different MCDM methods with different variables and weights and vector normalization schemes to investigate if we can select better models. In order to collect more evidence on our optimality conditions, we would like to analyze adversarial attack influence for models using single-criterion and multi-criteria model selections. With respect to the No Premature Early Stopping condition, a further investigation is needed since it has influence on the decision process but apparently we were not able to fully capture it. This probably can be converted into a filter to avoid premature early stopped models, but without emphasizing longer training, since good models can still have good performance but converging faster than other late converged models.

Acknowledgments

We would like to acknowledge the Centro de Tecnologias Estratégicas do Nordeste (CETENE) for providing computational resources (FACEPE APQ-1864-1.06/12) and to thank FACEPE, CNPq and CAPES (Brazilian Research Agencies) for their financial support.

References

- JET Akinsola, O Awodele, SO Kuyoro, and FA Kasali. Performance evaluation of supervised machine learning algorithms using multi-criteria decision making techniques. In *Proceedings of the International Conference on Information Technology in Education and Development (ITED)*, pages 17–34, 2019.
- Rahman Ali, Sungyoung Lee, and Tae Choong Chung. Accurate multi-criteria decision making methodology for recommending machine learning algorithm. *Expert Systems with Applications*, 71:257–278, 2017.
- Martin Aruldoss, T Miranda Lakshmi, and V Prasanna Venkatesan. A survey on multi criteria decision making methods and its applications. *American Journal of Information Systems*, 1(1):31–43, 2013.
- Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- Rich Caruana, Steve Lawrence, and C Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*, 13, 2000.
- Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

- Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.
- Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)*, 27(3):326–327, 1995.
- Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL <http://archive.ics.uci.edu/ml>.
- Felipe Farias, Teresa Ludermir, and Carmelo Bastos-Filho. Similarity based stratified splitting: an approach to train better classifiers. *arXiv preprint arXiv:2010.06099*, 2020.
- Felipe Costa Farias, Teresa Bernarda Ludermir, and Carmelo Jose Albanez Bastos-Filho. Embarrassingly parallel independent training of multi-layer perceptrons with heterogeneous architectures, 2022. URL <https://arxiv.org/abs/2206.08369>.
- Shivani Gupta and Atul Gupta. Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science*, 161:466–474, 2019.
- Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.
- Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110(3):457–506, 2021.
- Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.
- Andrew Y Ng et al. Preventing” overfitting” of cross-validation data. In *ICML*, volume 97, pages 245–253. Citeseer, 1997.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- Simone Romano. A general measure of dataset imbalance. <https://stats.stackexchange.com/questions/239973/a-general-measure-of-data-set-imbalance>, 2016. Accessed: 2022-07-04.

- Wojciech Sałabun, Jarosław Watróbski, and Andrii Shekhovtsov. Are mcda methods benchmarkable? a comparative study of topsis, vikor, copras, and promethee ii methods. *Symmetry*, 12(9):1549, 2020.
- Andrii Shekhovtsov. Pymcdm. <https://gitlab.com/shekhand/mcda>, 2022.
- Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 847–855, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450321747. doi: 10.1145/2487575.2487629. URL <https://doi.org/10.1145/2487575.2487629>.
- Evangelos Triantaphyllou. Multi-criteria decision making methods. In *Multi-criteria decision making methods: A comparative study*, pages 5–21. Springer, 2000.
- Gwo-Hshiung Tzeng and Jih-Jeng Huang. *Multiple attribute decision making: methods and applications*. CRC press, 2011.
- Maikel Yelandi Leyva Vazquezl, Luis Andy Briones Peñafiel, Steven Xavier Sanchez Muñoz, and Miguel Angel Quiroz Martinez. A framework for selecting machine learning models using topsis. In *International Conference on Applied Human Factors and Ergonomics*, pages 119–126. Springer, 2020.
- Xue Ying. An overview of overfitting and its solutions. In *Journal of Physics: Conference Series*, volume 1168, page 022022. IOP Publishing, 2019.
- Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.