Machine Learning Project

Authors (Cosimo Faeti, Riccardo Galarducci).
Master Degree (Data Science & Business Informatics, Data Science & Business Informatics).
Emails: c.faeti@studenti.unipi.it, r.galarducci@studenti.unipi.it
ML course (654AA), Academic Year: 2022/2023
Date: 06/07/2023
Type of project: **B**

Abstract

This report presents the development and performance comparison of multiple models. The validation technique employed involved a systematic approach, including preliminary experimental trials, grid search with different levels of granularity, and a 5-fold cross-validation, followed by the final model assessment. Using this approach, we identified the optimal model for the ML-CUP task, which turned out to be a Support Vector Machine (SVM) with a rbf kernel function.

1 Introduction

The aim of this project is to conduct an a comprehensive analysis and comparison of different models, namely Random Forest, Support Vector Machine (SVM), and Multi-Layer Perceptron (MLP). The report is divided into two distinct parts, each addressing specific tasks.
In the first part, we focus on a classification task using benchmark Monk datasets, which is achieved through empirical trials. The second part shifts attention towards a multivariate regression task using the ML-CUP dataset. This involves a process consisting of preliminary experimental trials, parameter fine-tuning, and an in-depth comparison among the models.

2 Method

The code was developed in Python, using Jupyter Notebook as IDE. To overcome the limitations of local machines and expedite the workflow, tools like Google Colab and Kaggle were employed. The code relies on several Python libraries for data handling and graphical representation, namely Pandas, NumPy, TensorFlow, Matplotlib and Seaborn. Furthermore, specific Python libraries were utilized for building machine learning models, including Keras for MLP, TensorFlow for Random Forest, Scikit-Learn for SVM, and KerasTuner for GridSearch.

In the preprocessing phase, we utilized *one-hot encoding* on the Monk datasets. This transformation was applied to convert categorical variables into a binary representation, facilitating further analysis. For the ML-CUP dataset, no preprocessing step was deemed necessary due to the inherent nature of the data, which are composed of continuous attributes with mean close to 0 and standard deviation around 1.

To familiarize ourselves with the models, we conducted multiple empirical trials on the Monk datasets, taking into accounts its intrinsic characteristics. The dataset was already divided into

a design set and test set.
For the ML-CUP, we partitioned the data into a design set (85%) and an internal test set (15%). In the case of Random Forest and SVM, we initiated a preliminary experimental phase consisting of empirical trials where we kept certain parameters fixed while exploring the models. However, for MLP we went a step further and trained the model multiple times to evaluate the impact of initialization strategy on model performance. To ensure a robust assessment, we employed 5-fold cross-validation, considering that the validation split can also influence the results. We completed the tuning of the hyperparameters with a GridSearch approach using the HoldOut method which allows us to explore a range of hyperparameter at different levels of granularity, striking a balance between computation time and focusing on finer range of hyperparameters. Before evaluating the models on the internal test set using a model trained on the design set, we conducted model validation through 5-fold cross-validation. This approach helped us identify and select the best-performing model to make predictions on the blind test set.

3 Experiments

3.1 Monk Results

The three Monk problems consist of binary classification tasks. While for the first two problems there is no noise in the training sets, M3 has 5% of misclassified training examples. Due to this fact, we add regularization to the models only in M3, while aiming for 100% accuracy on training sets in the first two problems.

Multi Layer Perceptron. The most suitable architecture for MLP turned out to be the same in all three tasks. It consists of two hidden layers, each composed of two units. The output layer has only one unit with *sigmoid* activation function.
Regarding the activation function in the hidden layers, *elu* yielded better results compared to *relu* in all tasks.
The small number of units in MLPs makes them highly dependent on the weights initialization strategy. $HeUniform$ initialization proved to be the most robust in M1 and M3, while $RandomUniform$ initialization returned the most consistent results in M2.
We tried two different learning algorithm, *SGD* and *Adam*, and experimented with different configurations of learning rate (η) and momentum. Adam showed better results on M1 and M2 while SGD performed better on M3. Moreover, we regularized the model on M3 by adding weight decay (λ) equal to 0.01 and an early stopping callbacks that monitored the validation loss, stopping the training if it no longer decreased for more than 10 epochs (learning curves and accuracies are shown in Figure 1). These techniques lead to better performance on the test set, as shown in Table 1.

Random Forest. To train Random Forest, we used data without encoding because of its ability to handle categorical data. We experimented different hyperparameter configurations, including *num_trees*, *max_depth*, *min_examples* in a node, *num_attributes*.
In M1 and M2 we used a *min_examples* in a node equal to 1 to perfectly fit the training

data. Building a high number of trees is not worth it in both M1 and M2 since we can achieve benchmark performances with 300 and 200 of trees, respectively. In M3, we tried increasing the threshold for the *min_examples* in a node to avoid overfitting the noise.

Support Vector Machine. While training SVM, we experimented with different kernels and values of the C regularization hyperparameter. Specifically, in M3 we aimed to find the most suitable value of C that would yield the best results on the validation folds to prevent overfitting. The model with the best validation results had an rbf kernel function and C equal to 1.

The results of MLP, SVM and Random Forest are shown in Table 1, 2, and 3, respectively.

Task	**Hyperparameters**				**Loss**		**Accuracy**	
	optimizer	η	λ	α	TR	TS	TR	TS
Monk 1	Adam	0.04	0	0	0	0	100%	100%
Monk 2	Adam	0.05	0	0	0	0	100%	100%
Monk 3	SGD	0.05	0	0.01	0.23	0.13	94%	96%
Monk 3+reg	SGD	0.01	0.01	0.7	0.14	0.17	95%	97%

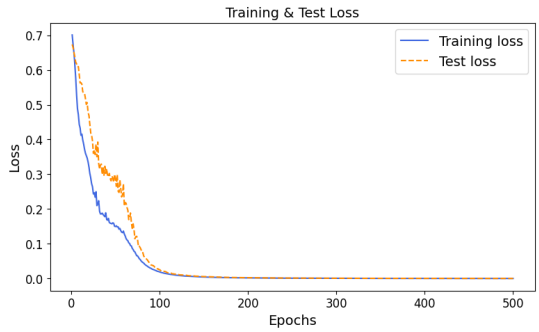
Table 1: MLP

Task	**Hyperparameters**				**Loss**		**Accuracy**	
	kernel	C	degree	gamma	TR	TS	TR	TS
Monk 1	poly	2	3	scale			100%	100%
Monk 2	poly	15	2	scale			100%	100%
Monk 3	rbf	1		scale			93%	97%

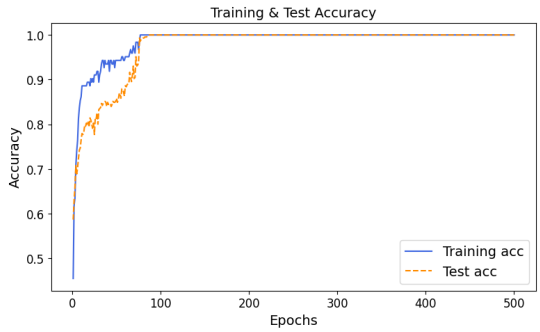
Table 2: SVM

Task	**Hyperparameters**				**Loss**		**Accuracy**	
	num_trees	max_depth	min_examples	num_attributes	TR	TS	TR	TS
Monk 1	300	10	1	$\sqrt{p}$	0	0	100%	100%
Monk 2	200	15	1	$\sqrt{p}$	0	0	100%	77%
Monk 3	200	30	10	$\sqrt{p}$	0	0	93%	97%

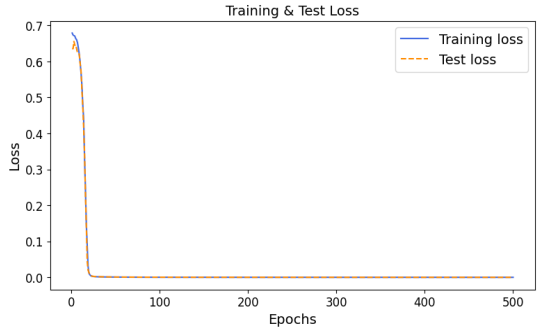
Table 3: Random Forest



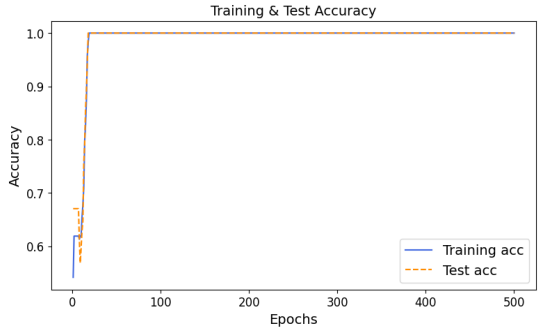
(a) Monk 1 Loss



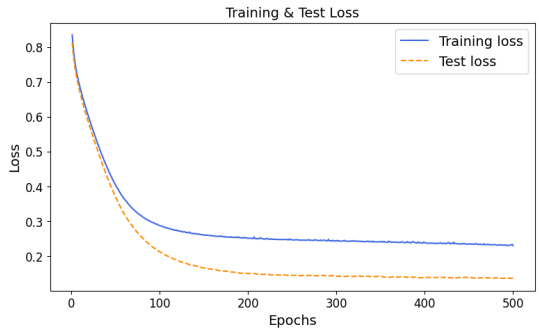
(b) Monk 1 Accuracy



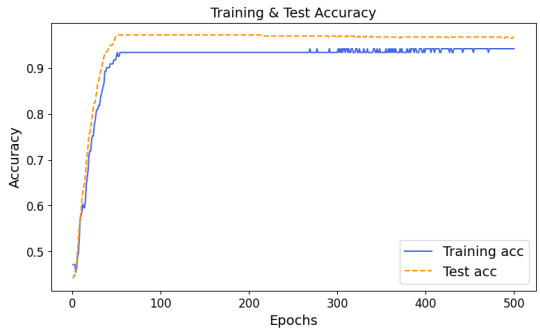
(c) Monk 2 Loss



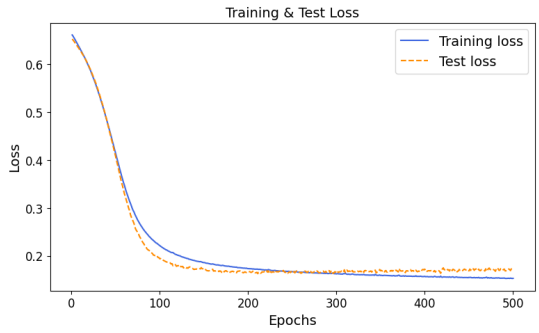
(d) Monk 2 Accuracy



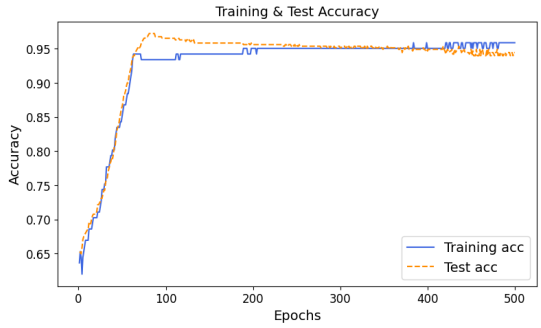
(e) Monk 3 Loss



(f) Monk 3 Accuracy



(g) Monk 3 (regularized) Loss



(h) Monk 3 (regularized) Accuracy

Figure 1: MLP Loss & Accuracy for Monk Problems

3.2 Cup Results

Validation Schema. The data is divided into a design set, which constitutes 85% of the total data for model selection and validation, and a test set, consisting of the remaining 15% of the data for model assessment.
We followed a systematic approach to select the best model using the following steps:

1. **Preliminary experimental phase**. Only for MLP, we conducted preliminary trials to assess the impact of different hyperparameters on the model performance. The aim was to identify the eventual presence of irrelevant hyperparameters and determine the values that consistently outperformed others;

2. **Coarse-grained GridSearch**. Performing a GridSearch with larger intervals on a coarse range of values for the hyperparameters. This step narrowed down the search space, focused on promising ranges, and controlled computational costs. Holdout method was used for model validation, with 20% of the design examples. For MLP, each trial was executed three times to obtain more consistent results across different initialization. Additionally, a callback was implemented during MLP training to monitor the validation loss and stop training when it no longer decreased for more than 20 epochs, avoiding overfitting and reducing computational time. To expedite the process, the coarse GridSearch for MLP was divided into four sub-blocks, each using a different combinations of optimizer and regularization method (Adam&weight_decay, SGD&weight_decay, Adam&Dropout, SGD&Dropout). The wall computing time can be seen in Table 9;

3. **Fine-grained GridSearch**. For the best model of each class retrieved in step 2, a more detailed GridSearch was conducted on a narrower range of values. This step yielded the best hyperparameter configuration for each class of models. The wall computing time can be seen in Table 9. The search space and best hyperparameter configurations for MLP, SVM, and Random Forest are summarized in Tables 4, 5, and 6, respectively;

4. **Cross Validation**. A 5-Fold Cross Validation was performed on the best model for each class to obtain mean and standard deviation of model performance. The results are shown in Table 7;

5. **Model Assessment**. The best SVM and Random Forest model were retrained using the design set and then tested on the internal test set. For the best MLP, a validation set of 20% of design set examples was used to stop the training and avoid overfitting using the same callback exploited in the GridSearch. Subsequently, the MLP model was tested on the intenal test set. The model with the lowest Mean Euclidean Error (MEE) was selected for prediction on the blind test set. The results are presented in Table 7.

Resources. The notebooks were executed using both Google Colab, which provide 12GB RAM and dinamically adjust the GPU usage limits, and Kaggle, which grants access to 12GB RAM and 30 hours per week of GPU.

Hyperparameter	**Range of Values**	**Best Values**
units	[16, 32, 64, 256]	32
activation	relu, elu	elu
kernel_initializer	RandomUniform, GlorotUniform	RandomUniform
num_hidden_layers	[2, 3]	3
learning_rate (η)	[0.001, 0.01, 0.1]	0.01
momentum (α)	[0, 0.5, 0.9]	0.9
nesterov	True/False	True
weight_decay (λ)	[0.0001,..., 0.01]	0.002
optimizer	Adam, SGD	SGD
dropout rate	[0, 0.1, 0.3, 0.5]	
batch size	[16, 64, 256]	64

Table 4: Grid Search search space MLP

Hyperparameter	**Range of Values**	**Best Values**
kernel	poly, linear, rbf	rbf
C	[0.001,...,100]	4
gamma	scale, auto	auto
epsilon	[0.01,...,10]	0.22

Table 5: GridSearch search space SVM

Hyperparameter	**Range of Values**	**Best Values**
num_trees	[100,...,1000]	500
min_examples	[1,...,10]	2
max_depth	[5,...,30]	20
categorical_algorithm	CART, RANDOM	CART
num_candidate_attributes_ratio	[-1.0 ($\sqrt{p}$), 0.2, 0.5]	0.2
winner_take_all	True/False	True

Table 6: GridSearch search space Random Forest

Model	**MEE**		
	Val mean ± stdev	Design	Test
MLP	1.4037 ± 0.0750	1.3783	1.5282
SVM	1.4461 ± 0.0606	1.2771	1.4672
Random Forest	1.4679 ± 0.0782	0.6379	1.5198

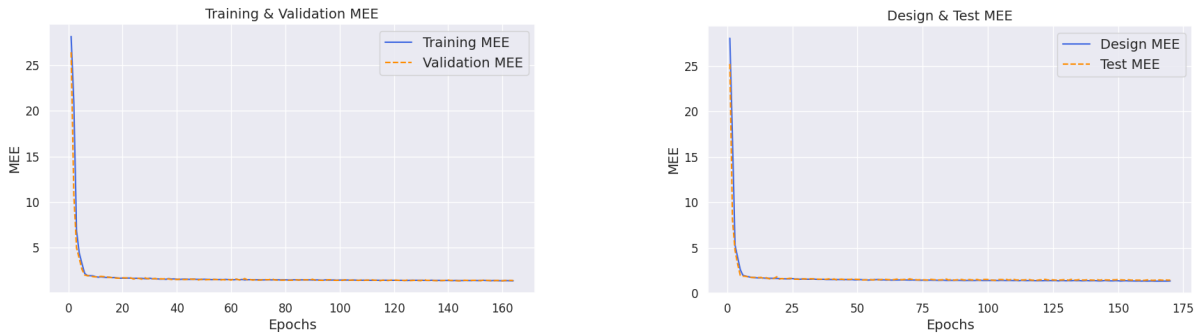
Table 7: Performance of the best model for each class on CUP dataset

3.3 Final Model

For each class of model, we have assessed the best model on the internal test set. The learning curve of MLP on validation and internal test sets is reported in Figure 2.
The results of the models are shown in Table 7. As observed, the best performances are achieved by **SVM** and are considerably better than MLP and Random Forest.
The hyperparameter configuration of the final SVM is reported in Table 8.
The 95% confidence interval of the model error is $MEE \approx 1.4461 \pm 0.003076$.

Model	**Hyperparameter**			
	kernel	C	gamma	epsilon
SVM	rbf	4	auto	0.22

Table 8: Hyperparameter configuration best model (SVM)



(a) MLP Train&Val MEE

(b) MLP Design&Test MEE

Figure 2: MLP learning curves

3.4 Discussion

Regarding **MLP**, the outcomes of the GridSearch for both the SGD (Table 10) and Adam (Table 11) optimization algorithms clearly indicate a strong preference for weight decay as a regularization technique, as opposed to dropout. In addition, SGD prefers a smaller number of units and a larger batch size while Adam has an inverse behaviour preferring a larger number of units and a smaller batch size.

The results of the GridSearch for **SVM** (Table 12) suggest a preference in non-linear kernels, specifically *rbf*, as well as a moderate level of regularization and tolerance to error, indicated by C and *Epsilon* hyperparameters, respectively.

Looking at the results of **Random Forest** GridSearch (Table 13), the models tend to favor a large number of trees (1000) combined with a minimal requirement of data points in the nodes, and a preference for the splitting criterion based on very few number of candidate attributes (0.2). The choice of the *algorithm* and of the strategy (*winner_take_all*) is instead irrelevant

for the performances of the model.

Ultimately, analysing the results showed in Table 7, we observe that Mean Euclidean Error (MEE) on the internal test set for MLP and Random Forest falls outside the interval of the performance measure of the model estimated through 5-fold cross validation. However, for SVM, the test results are closer to the mean of its validation performance.
In addition, a graphical observation provide valuable insights that contribute to our conclusion. It is evident from the graphs (Appendix A) that the different models exhibit varying degrees of capturing the test target values of x and y. Furthermore, it is evident that all three models exhibit a higher sensitivity to the noise examples found within the x target variable, as opposed to the y target variable.

4 Conclusions

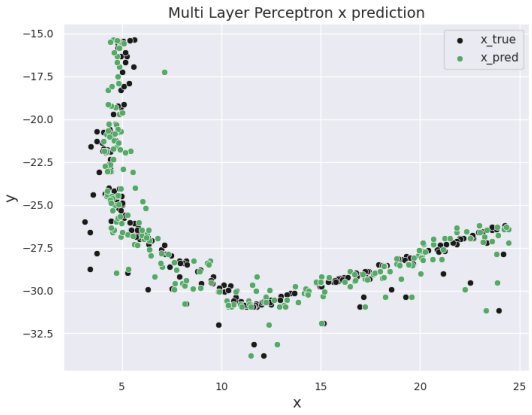
The SVM model with *rbf* kernel turned out to be the most suitable model for the MLP cup. This conclusion was drawn based on its superior performance on the internal test set compared to MLP and Random Forest, as well as its consistency with the error range obtained through cross-validation. Additionally, SVM required significantly less time for hyperparameter tuning compared to MLP.

BLIND TEST RESULTS: NoPt_ML-CUP22-TS.csv

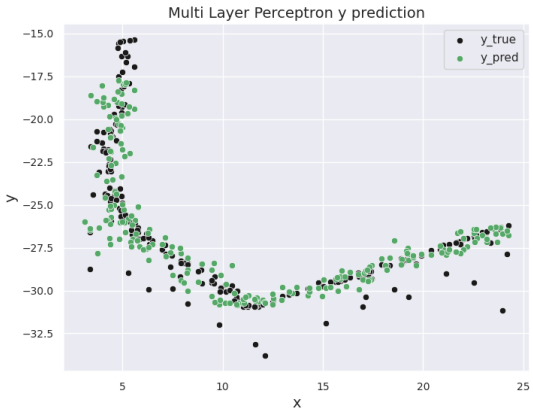
Acknowledgments

We agree to the disclosure and publication of our names, and of the results with preliminary and final ranking.

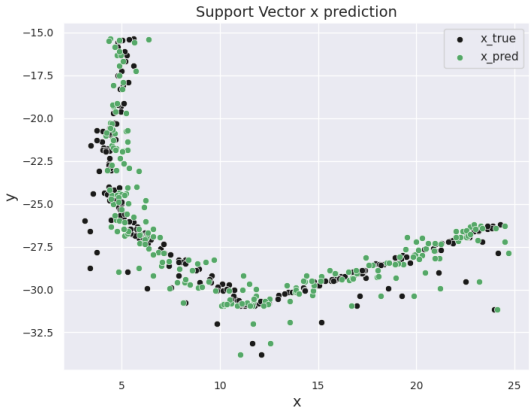
Appendix A



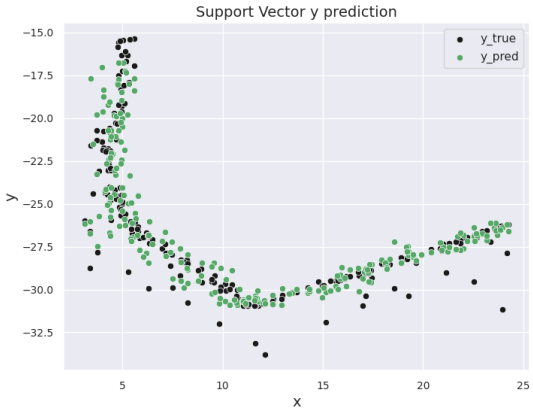
(a) MLP pred x



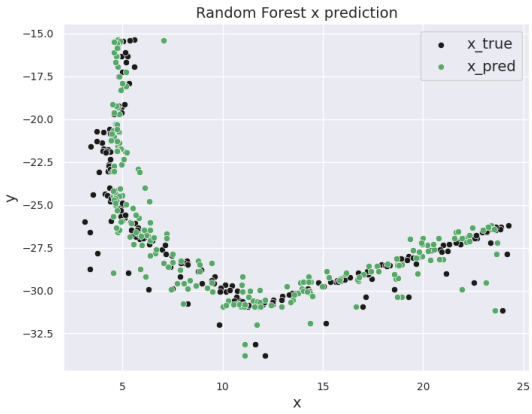
(b) MLP pred y



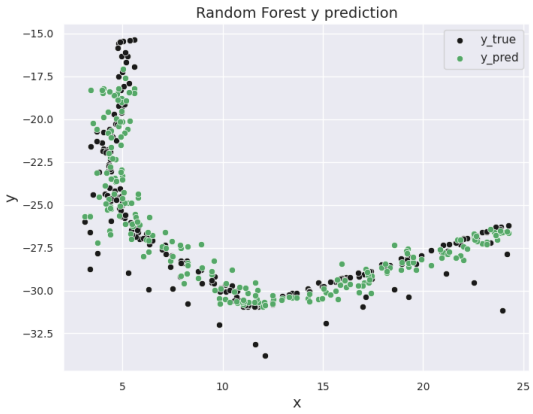
(c) SVM pred x



(d) SVM pred y



(e) Random Forest pred x



(f) Random Forest pred y

Figure 3: Predictions on internal test set

Model	**Grid Search**	
	coarse grain	**finer grain**
MLP (SGD&weight_decay)	8h 21min 46s	31min 31s
MLP (SGD&dropout)	4h 32min 14s	
MLP (Adam&weight_decay)	7h 1min	41min 49s
MLP (Adam&dropout)	11h 16min 22s	
Random Forest	34min 36s	6min 47s
SVM (linear&rbf kernels)	2min 31s	23min 56s
SVM (poly kernel)	3min 27s	

Table 9: Wall Computing Time

Rank	**Hyperparameters**							
	n_layers	units	learning_rate	momentum	weight_decay	nesterov	batch_size	dropout_rate
1°	3	16	0.01	0.9	0.001	True	64	
2°	3	16	0.01	0.9		True	64	0
3°	3	16	0.01	0.9	0.01	True	64	
4°	3	16	0.01	0.9	0.01	False	64	
5°	3	64	0.01	0.9	0.0001	True	64	

Table 10: MLP SGD (Weight Decay & Dropout) Coarse-grain GridSearch results

Rank	**Hyperparameters**						
	n_layers	units	activation	kernel_initializer	weight_decay	batch_size	dropout_rate
1°	2	256	elu	RandomUniform	0.001	16	
2°	3	64	elu	GlorotUniform	0.0001	16	
3°	2	256	elu	RandomUniform	0.0001	16	
4°	2	256	elu	RandomUniform		16	0
5°	2	256	elu	RandomUniform		16	0

Table 11: MLP Adam (Weight Decay & Dropout) Coarse-grain GridSearch results

Rank	**Hyperparameters**			
	kernel	C	epsilon	gamma
1°	rbf	10	0.01	auto
2°	rbf	10	0.1	auto
3°	rbf	10	0.01	scale
4°	rbf	10	0.1	scale
5°	rbf	10	1	auto

Table 12: SVM Coarse-grain GridSearch results

Rank	**Hyperparameters**					
	min_examples	algorithm	max_depth	candidate_attributes_ratio	n_trees	winner_take_all
1°	2	CART	15	0.2	1000	False
2°	2	CART	15	0.2	1000	True
3°	2	Random	15	0.2	1000	False
4°	2	Random	15	0.2	1000	True
5°	2	CART	30	0.2	500	False

Table 13: Random Forest Coarse-grain GridSearch results