INSTITUTE OF LOGIC AND COMPUTATION

MACHINE LEARNING - EXERCISE 1

Group 8

Member:
Peter HOLZNER, 01426733
Alexander LEITNER, 01525882
Mario HITI, 01327428

Submission: November 15, 2020

Contents

1 Introduction

2 Classifications techniques

2.1 K-Nearest Neighbors

K-Nearest Neighbors, or KNN for short, is one of the simpler machine learning algorithms. When training a KNN classifier, it essentially only stores the training data set. During the prediction step, it then trys to find out where the test sample fits into the saved/trained data set by computing which k samples are the closest matches (k-nearest neighbors).

The difficult parts are then how to best define this notion of distance between samples and how to prepare the features, especially when non-numerical features are involved. Firstly, one can consider different distance measures, most commonly used are Minkowsky metrics such as the Manhattan or Euclidean distance. When $k > 1$, then the output is calculated by voting between the k-nearest samples where the voting weight of each sample can also be influenced by its distance to the sample that is being infered, otherwise voting weight can be assigned uniformly.

Decision boundaries are formed between regions of differently assigned classes. These boundaries tend to be smoother and less erratic when k is high, and result in a model with slower inference, but can lead to underfitting by not capturing the minute differences between samples. Lower ks can be thought of as more complex models, which can lead to overfitting the training set, and are less compute intensive to compute. Looking for the correct k to use is then the major task when fitting a KNN model and is highly data dependent.

One of its major pitfalls in practice is that the KNN classifier becomes slower and slower (for inference) as more training samples are added and as more features are considered.

2.2 Multi-layer Perception

A multi-layer perception (MLP) has the structure of a single layer perception, but with one or more so-called hidden layers. The inputs and weights are used to work out the activation for any node (weighted sum and transfer/activation function). The output of the first hidden layer nodes can be then fed into the next (hidden) layer of nodes, or in case of a single hidden layer, into the output layer. The output layer is not directly connected to the input layer, hence MLPs are also often called feed-forward Neural Networks.

For each Hidden Layer or Output Layer node, a weighted sum s is calculated,

$$s = \sum \omega \cdot x \tag{1}$$

$$f(s) = \frac{1}{1 + e^{-s}} \tag{2}$$

where ω is the weight, x is the input and s is the weighted sum. The resulting sum of each node is then transformed into the activation number via the transformation rule (often called the activation function).

The idea of back propagation is to use the output error (difference between the actual and predicted classes) to adjust the weights of the inputs at the output layer. We can also calculate the error at the previous layer, and use it to adjust the weights there. This process can repeat through any hidden layer $\Rightarrow$ "back propagation". A key to the success of MLPs in learning complex decision boundaries is the use of non-linear transformation functions, such as the "sigmoid" ("logitisc" in sklearn) function above - although there are other popular functions such as "tanh" and "relu".

2.3 Random Forrest

The last classifier we tested was the Random Forest Classifier (RFC). RFC is a type of ensemble classifier that consists of multiple Decision Trees, where each tree is only given subset of all features.

The final results of the RFC is then aggregated from the individual trees ("majority voting"). The expectation is that overfitting can be avoided this way.

A simple Random Forest in Pseudocode[1]: Training:

1. Assume number of cases in the training set is N. Then, a sample of these N cases is taken at random but with replacement (bootstrapping).
2. If there are M input variables (or features), a number $m < M$ is specified (subset of features) such that at each node, m variables are selected at random out of the M. The best split on these m is used to split the node. The value of m is held constant while the forest is grown.
3. Each tree is grown to the largest extent possible and there is no pruning.

Prediction:

1. Let each tree produce its prediction output.
2. Aggregate the individual prediction into the final result of the Random Forest (i.e. majority vote for classification, average for regression)

2.4 Run-time Comparison

Figure 1 shows the run-time behaviour of the different algorithms using the polish bankruptcy dataset. For KNN the time to make a prediction does scale linear with the number of neighbours while the time to fit the model is significantly smaller. For RFC the runtimes are flipped, i.e. fitting takes significantly longer than prediction.

MLP has a more complex behaviour because some hidden layer sizes converge faster and stop before the maximum amount of iterations is reached. Once the model is fitted the time to make predictions is insignificant. Overall KNN is the fastest algorithm but will be slower if the same model is used to make repeated predictions.

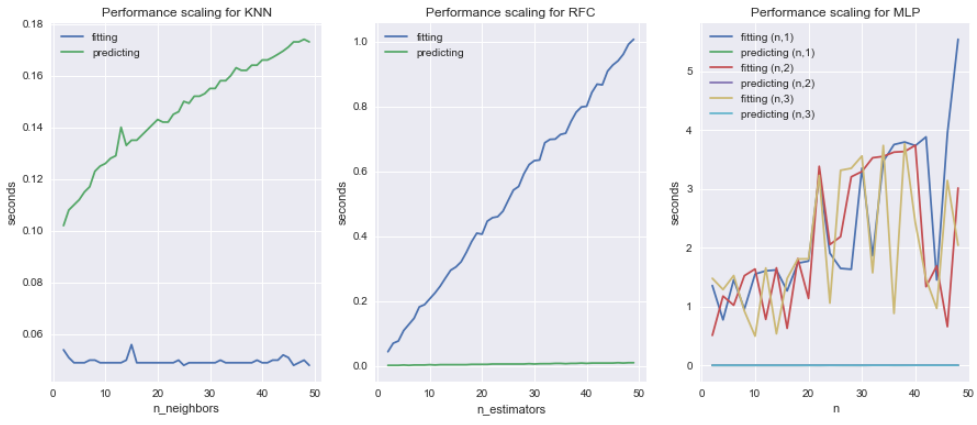


Fig. 1: Runtimes for KNN, RFC and MLP depending on the number of neighbours, estimators and hidden layer sizes

[1]adapted from Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, ISBN: 9781492032618

3 Amazon

3.1 Data set characteristics

It's a high dimensional data-set in case of the number of columns 10000. The target column for the classification is named "Class" and it has 50 different names. The other columns have only numeric values. To see the different distributions from the classes we split it in two different plots.

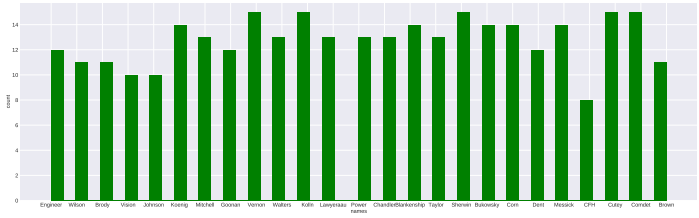


Fig. 2: distribution of the first part of the Class

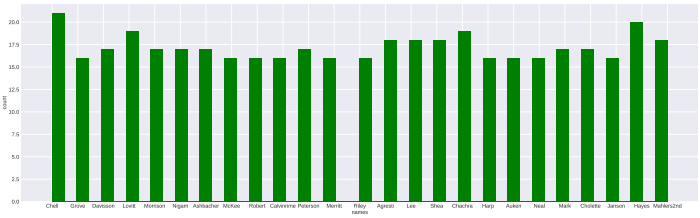


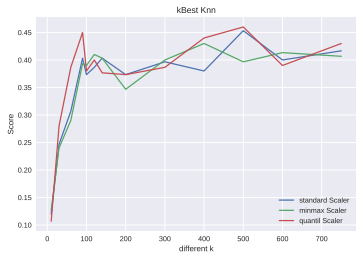
Fig. 3: distribution of the second part of the Class

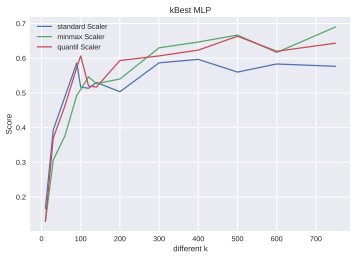
The mean value of this distribution is 15 and the max value of a class is 22. So there are roughly no out-layers in the number of one class. For further inspections we will reduce the number of columns with the k-Best and the PCA method from sklearn and see which method works better. Then we try to find the best classification model to work with this data-set.

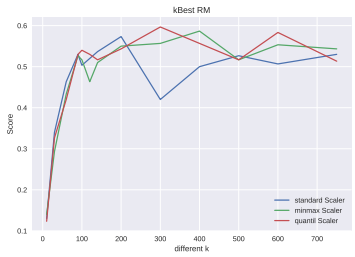
3.2 Data prepossessing

3.2.1 Feature Selection

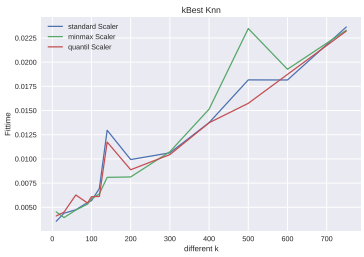
Use the k-Best method to reduce the number of columns and work with that.

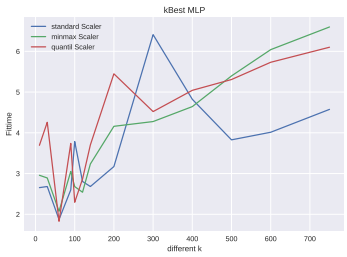


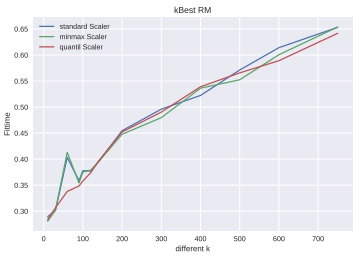




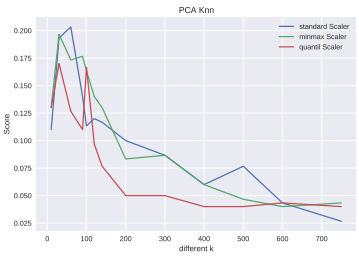
We tested the features (all columns without the Class column) of the train data-set with different values of k for the k-Best method from sklearn and plot it vs the score of it's individual classifier with default parameters. All three classifiers have the same tendency that at a certain value of k the score did not improve anymore. The MLP classifier achieve the highest score 60%.



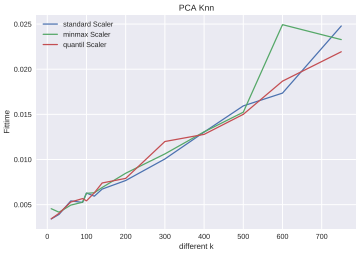


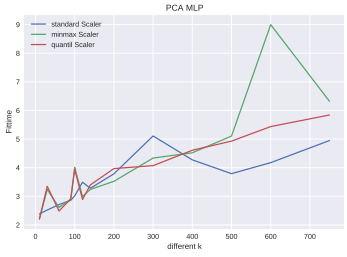


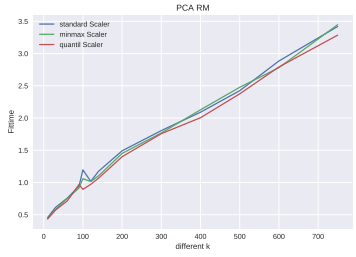
For the train-time vs the different k value there is a linear behavior for the Knn and the Random Forrest classifier. For the MLP classifier it stops when the given tolerance is reached and that's the reason why the train-time looks like this.
In comparison the PCA method:



In comparison to the k-Best method the PCA method performs for all three different classifier worse and for increasing k values the score decreases dramatically to a low score.







There are basically no differences in training time between the PCA and the k-Best method.

3.3 Comparison of different classifiers

For the following test we used the k value for the k-Best method to prepare the data-sets. We used the k value which achieved the best score from the previous section.

3.3.1 k-nearest neighbors classifier

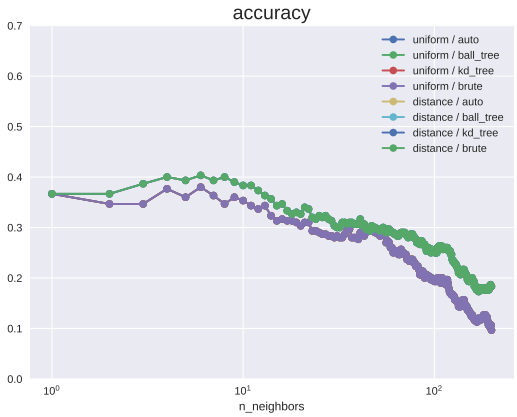
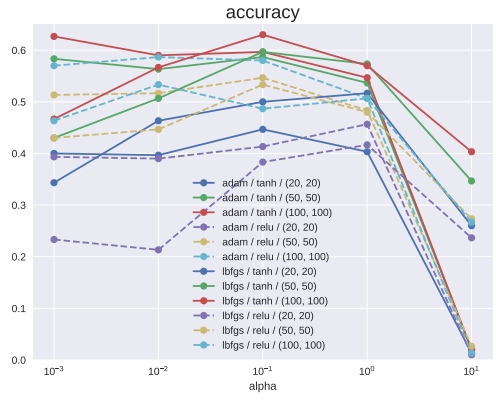


Fig. 4: accuracy score with different parameters

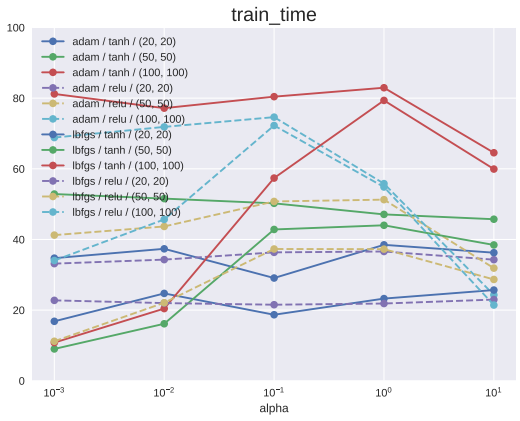
The first two different parameters are uniform and distance. For the uniform parameter all points in each neighbourhood are weighted equally and for the distance parameter it weights points by the inverse of their distance. The reason why we chose this parameter was to see if there are any differences between the uniform and the distance parameter. The uniform parameter performs a bit better than the distance parameter. All in all the best accuracy is about 0.4 and there are basically no differences between the different parameters. The other scores like F1, recall and precision are in the same range about 0.4.

	score	F1 score	precision	recall	runtime
KNN	0.473	0.53743	0.455699	0.489984	0.0923s

3.3.2 MLP-classifierer



(a) accuracy score vs alpha



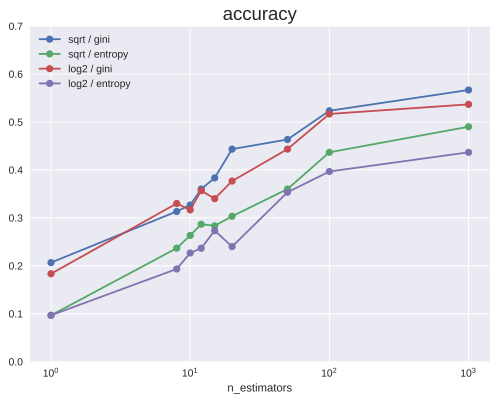
(b) train-time vs alpha

On the left side the accuracy vs alpha with different parameters for the classifier is plotted. The highest score is achieved with a relatively low value of alpha and the highest number of hidden-layer (100, 100). The "tanh" solver performs better than the other solvers. The other scores like "F1", "recall" or "precision" are also in the same range than the accuracy score.

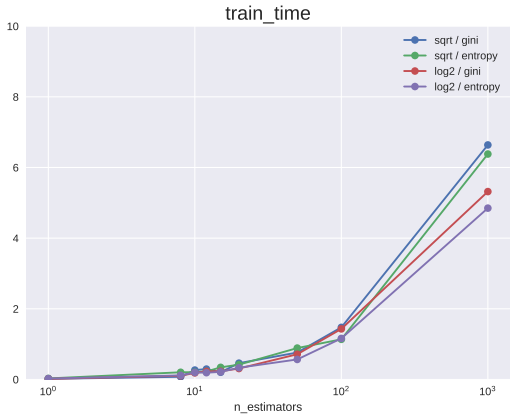
	score	F1 score	precision	recall	runtime
MLP	0.603	0.60244	0.652	0.609921	67.998622s

For the train-time plot each increase of the number of hidden-layer size also increases the train-time. The "tanh" solver has the best performance.

3.3.3 Random Forrest classifier



(a) accuracy score vs number of trees



(b) train-time vs number of trees

For the Random Forrest classifier the accuracy grows nearly linear with the number of trees (n estimators). The parameter "gini" achieved the best score for that.
For the train-time there is no dramatical increase like for the MLP classifier as before and all different parameters perform approximately the same.

	score	F1 score	precision	recall	runtime
RFC	0.576667	0.562676	0.613419	0.595786	5.628368s

3.4 Best parameter results

For the multi class classification to get the F1 score, the precision and recall we calculate it with the sklearn prepared method and used the parameter "macro" to calculate the average to get only a scalar for these scores. The best score produces the MLP classifier. KNN is the best model in case of computation efficiency because it produced a reasonable result with the fastest way from the two other classifier.

	score	F1 score	precision	recall	Runtime
MLP	0.65	0.6362	0.67969	0.653198	67.998s
RFC	0.576667	0.562676	0.613419	0.595786	5.628368s
KNN	0.473	0.53743	0.455699	0.489984	0.0923s

4 Polish companies bankruptcy

4.1 Data set characteristics

The "polish companies bankruptcy" data set describes financial characteristics of various polish companies in five different years and classifies them as bankrupt or solvent/not bankrupt. Each year contains approximately 5.000 - 10.000 instances with 64 different attributes each. Depending on the year only 5-10% of all companies become insolvent thus making the classes highly imbalanced. For the following analysis only the fifth year was considered as it contains highest amount of bankrupt companies relative to the sample size.

4.2 Data preprocessing

4.2.1 Feaure Selection

While this is a big data set with a rather high amount of dimensions (compared to the other sets in this exercise), the amount of computational power needed to analyze the complete set is still within the capabilities of an average PC. Furthermore a correlation analysis has shown, that almost every attribute has at least some correlation with the predicted class. Therefore no feature selection steps or removal of columns are performed.

4.2.2 Scaling

The data set describes a very wide range of companies in terms of size. Therefore, values of some attributes have a very high standard deviation and some extreme outliers being several orders of magnitude bigger than the average as seen in Figure 7. Therefore scaling or normalization as a pre-processing step is expected to have a positive impact on the prediction. For this data-set effects of no scaling, min-max-scaling and normalization with l0- and l1-norm are studied

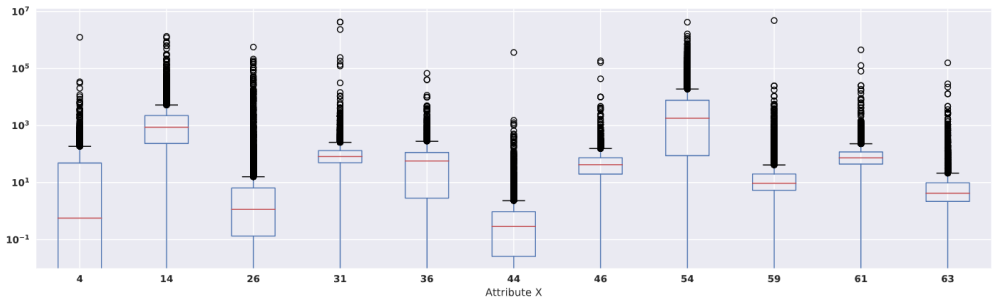


Fig. 7: Boxplot with outliers of several selected attributes. Note that the y-Axis uses a logarithmic scale. Some outliers are several orders of magnitude bigger than the average

4.2.3 Missing values

About 1,46% of all values are missing which makes imputation necessary. For this data-set the effects of imputation with a constant and imputation with the mean value of a given feature is studied.

4.3 Rating different classifiers

In order to compare different classifiers a common metric is needed. However due to classes being heavily imbalanced towards solvency the often used accuracy score $ACC = \frac{TP+TN}{samples}$ is not suitable

because $TP >> TN$. Therefore a cost matrix from the view point of a potential investment bank has been devised:

	predicted bankruptcy	predicted solvency solvent
solvent	-1 (TP)	100 (FP)
bankrupt	1 (FN)	0 (TN)

The cost values can be interpreted as follows:

- TP: the model made a correct prediction. Investing in this company generates a small amount of money (negative cost)
- FP: the model erroneously predicted a company that will go bankrupt as solvent. The complete investment is lost (high cost)
- FN: the model predicted a solvent company as bankrupt. We lose a small amount of potential revenue (small cost)
- TN: This company will go bankrupt. No investment shall be made. (neutral cost)

This cost matrix assumes that every company requires the same investment and will generate the same cost if it goes bankrupt. In reality bigger companies with higher investments will generate a higher cost if the prediction is wrong. However for this exercise we assume that the cost matrix is identical for every company.

In a final step the cost is normalized to allow for comparison of data sets of different sizes. The resulting score is a value ≤ 1 with $score = 1$ representing a perfect prediction. Therefore any result between 0 and 1 one can be considered as a profit gain for an investment bank and any model rated < 0 would lead to an overall loss.

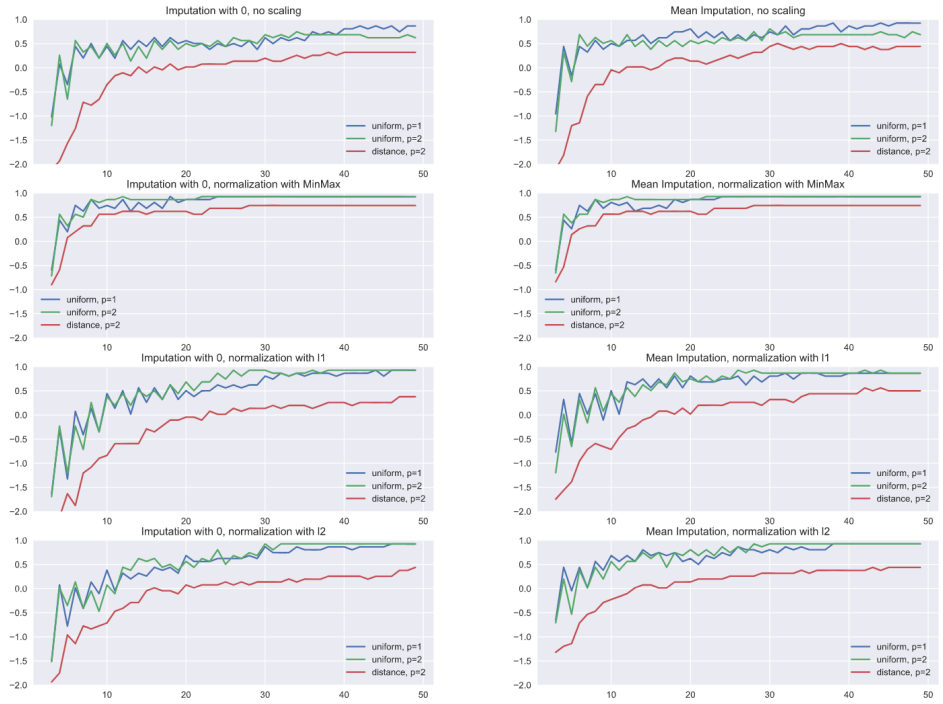
4.4 Comparison of different classifiers

4.4.1 k-nearest neighbors classifier

As seen in Figure 8 the KNN-classifier manages to make very good predictions ($score > 0.9$) for higher k. The highest score of was achieved using a uniform distribution, k=26, mean imputation and normalization with the L1-norm. Generally all combinations of parameters and pre-processing techniques tend to converge towards a good score. Every scaling does have a positive impact on the score with min-max scaling being the best. In all cases a uniform weighting of neighbours achieves a better result than a weighting based on manhatten distance (p=1) or euclidian distance (p=2).

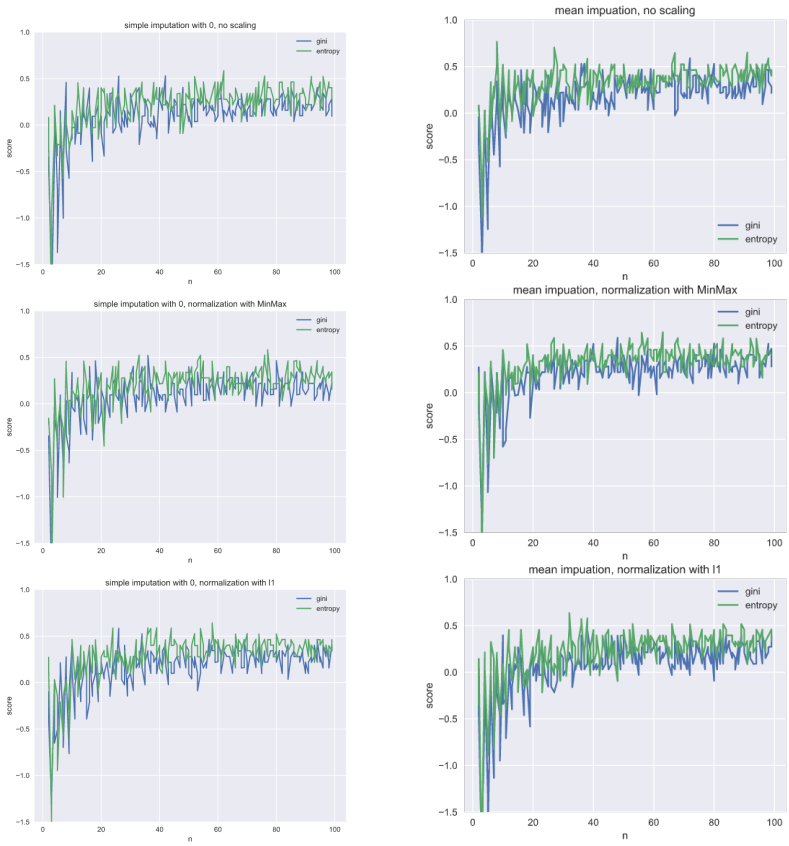
4.4.2 Random Forest Classifier

As seen in Figure 9 the RFC classifier can make predictions with a positive score but those scores rarely reach values > 0.5. Furthermore predictions tend to be quite unstable. This is especially prevalent for low n (n = number of decision trees). Due to the amount of noise it is very hard to see the effect of pre-processing. There is no significant difference between constant imputation and mean imputation but data normalization using the L1-norm does lead to slightly better results. Furthermore the "entropy" criterion does lead to better result than "gini" in most cases.



(a) Constant Imputation (b) Mean Imputation

Fig. 8: comparision of different preprocessing tequnices and knn weights. The values on the x-Axis represent the numbers of nearest-neighbours



(a) Constant Imputation (b) Mean Imputation

Fig. 9: Comparison of different pre-processing techniques. The values on the x-Axis represent the numbers of decision trees

4.4.3 Multi-layer Perceptron Classifier

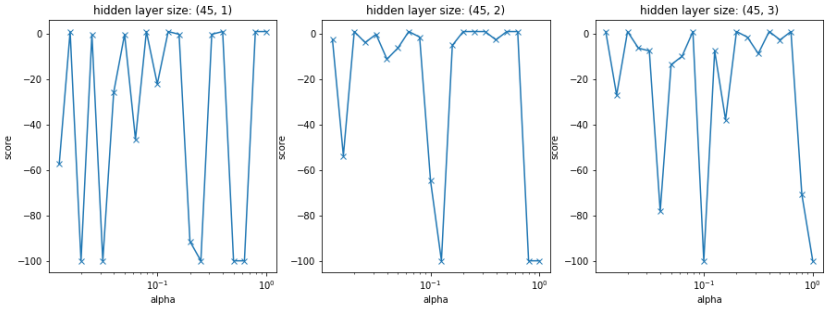


Fig. 10: Score for different learning rates and alphas

Compared to KNN and RFC the MLP-classifiert managed to achieve the highest score (< 0.94) however it also exhibits the highest instability as shown in Figure 10. Generally predictions are either very good (> 0.8) or very bad ($<< 0$) with very few results in between. This behaviour was observed across multiple hidden layer sizes and learning rates and not just the parameters represented in Figure 10. No correlation between input parameters and bad scores was found and fluctuations appear seemingly random. Due to this high instability of results no discussion pre-processing effects is made.

4.5 Results and conclusions

	accuracy	score	F1 score	precision	recall	runtime
k-nearest neighbors	0.9362	0,9303	0,1374	0.9000	0.0743	0,142s
random forest	0.9334	0.6864	0.1060	0.6363	0.0578	0,664s
multi-layer perceptron	0.9328	0.8674	0.0480	0.7500	0.0247	3,811s

Table 1: Summary of best results in terms of calculated score for each classifier

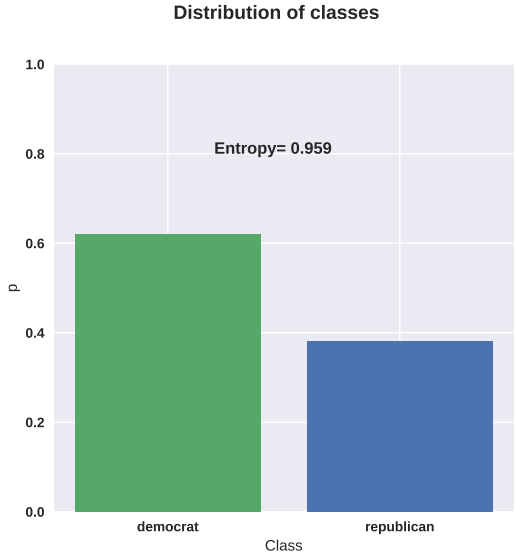
KNN achieved the best results in terms of score, runtime and stability. MLP scored slightly below KNN but has the worst stability. Results for RFC are rather unstable as well but do not result in extreme drops in performance at random times such as MLP but is fluctuating within a certain band-with. Overall scores are generally worse than KNN.

Table 1 shows the best result for each classifier. Generally better models (in terms of score) have good accuracy and precision but very low recall and F1 values. This is because classifiers are rated by their cost function described in Section 4.3 which puts a very high cost on FP predictions. Therefore models which make conservative predictions, by only rating a company solvent if the level of confidence is very high, are considered better. However this leads to a high amount of FN predictions which result in low recall and F1 scores. No model with high accuracy and recall rate has been found.

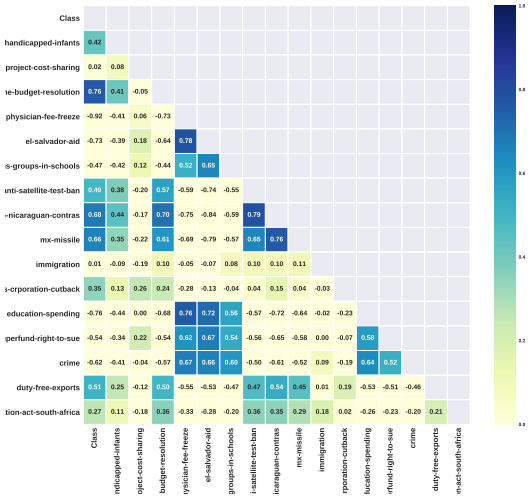
5 Voting the Congress

5.1 Data set characteristics

The data-set has a low number of rows and a medium in columns. The features are nominal values and the target class contains only 2 names "Republican" and "Democrats".



(a) distribution from the target



(b) correlation heat-map of the features

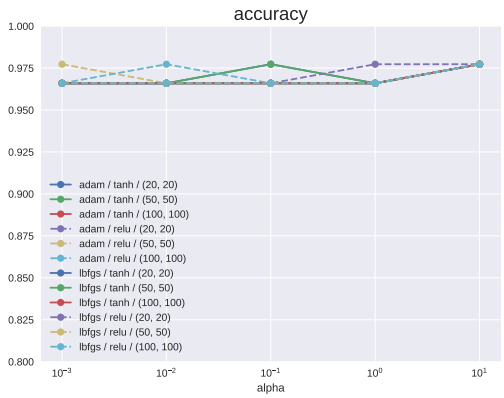
The distribution of the class is almost even distributed with an entropy of 0.959. The heat-map on the other side shows the absolute correlation between the single feature columns and the target class. The correlation from some columns is up to 0.7.

5.2 Data prepossessing

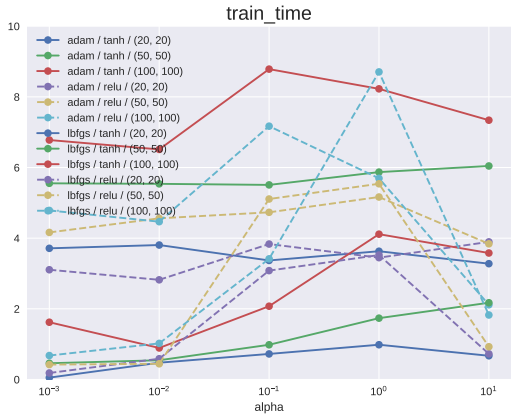
The features are nominal values like "n" for no, "y" for yes and "unknown". To prevent for scaling we rewrite the names into even distributed numbers like "n" to -1, "unknown" to 0 and "y" to 1. We decided that the "unknown" value in the data-set is not a missing value because of the scaling technique that we chose and the results are good enough for that strategy.

5.3 Comparison of different classifiers

5.3.1 MLP-classifierer classifier

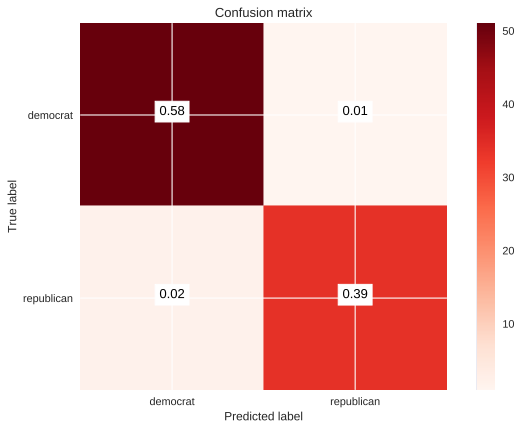


(a) accuracy score vs alpha

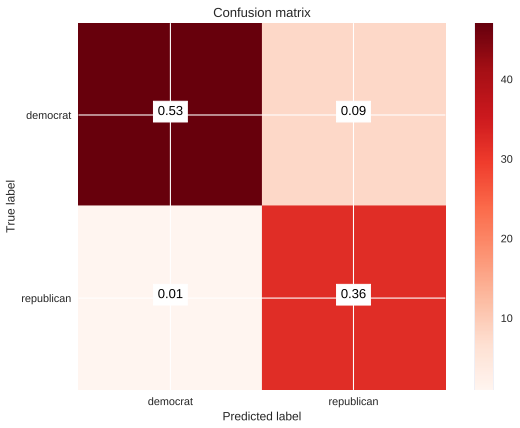


(b) train-time vs alpha

With all different combinations of the parameters it achieved a reasonable good score. The differences are only less than 5%. By increasing the number of hidden-layer size there is no improvement in case of achieving a better score. Only the train-time increases with the higher number of hidden-layer size.



(a) confusion matrix of the MLP-classifier

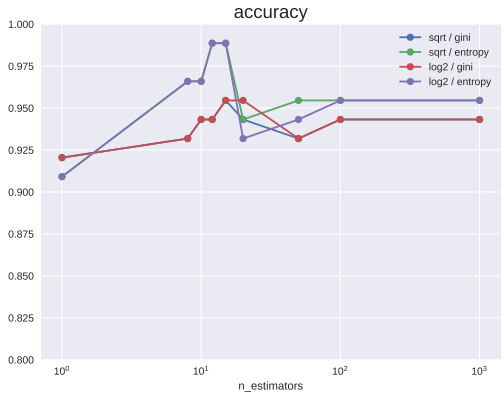


(b) confusion matrix of the RFC-classifier

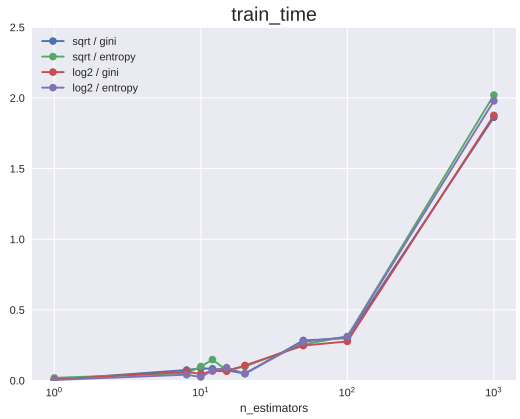
	score	F1 score	precision	recall	runtime
MLP	0.977273	0.976975	0.976975	0.976975	0.103837s

The best score from 0.977273 achieved more than one parameter settings, but the classifier gets this score also with the lowest number of hidden-size of (10, 10) and therefore also the shortest runtime.

5.3.2 Random Forrest-classifierer



(a) accuracy score vs number of trees



(b) train-time vs number of trees

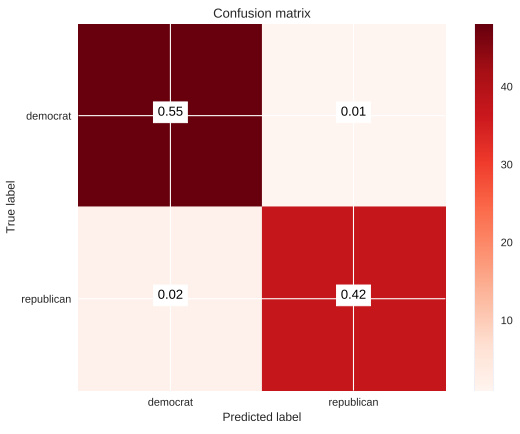
The score is slightly lower than the classifier before. For the train-time there is a significant increase from 100 trees to 1000 trees but the score does not increase much.

	score	F1 score	precision	recall	runtime
RFC	0.988636	0.988456	0.9900	0.987179	0.069584s

The score from the RFC is a bit higher and also the other scores. The confusion matrix also shows that the results are better than the MLP results and also it needed less time to train it for the classification.

5.3.3 KNN-classifierer

(a) accuracy score vs the number of neighbors



(b) confusion matrix of the RFC-classifier

	score	F1 score	precision	recall	runtime
KNN	0.965909	0.965368	0.966842	0.964155	0.002082s

5.4 Results

	score	F1 score	precision	recall	runtime	Computation efficiency
MLP	0.977273	0.976975	0.976975	0.976975	103.837ms	9.414
RFC	0.988636	0.988456	0.9900	0.987179	69.584ms	14.207
KNN	0.965909	0.965368	0.966842	0.964155	2.082ms	463.933

Again the best computation efficiency is achieved with the KNN classifier. The best score with the RFC with 2% higher score than the other classifier.

6 Heart disease

Link: Heart Disease Data Set @ UCI ML Repository

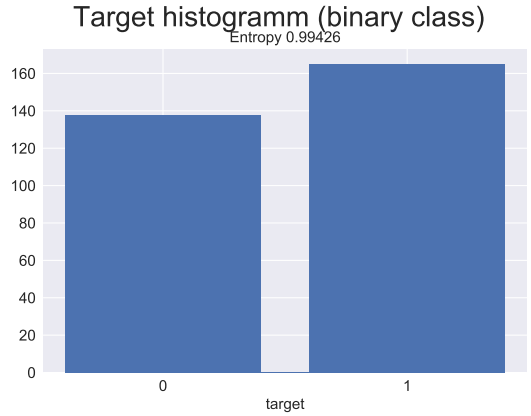
The Heart Disease data set contains various medical measurements from Cleveland residents. The goal is to be able to detect different heart diseases, indicated by five different classes ranging from 0 (no disease) to 4 - whether values from 1 to 4 indicate different diseases or just severity is unclear.

6.1 Data set characteristics

The particular data set that we use has already been preprocessed (categorical data has been numerically/binary encoded, e.g. "male":1, "female":0 for sex) and stripped of any data that clearly identifies the studied people. The original data contains 74 different attributes of which 14 have been identified as relevant by various publications. Also, additional raw, partially or differently processed data sets, for different locations, are also available on the UCI ML Repository. However, we focused our efforts on the processed.cleveland.data set.

It only contains 303 samples of different patients and is therefore both low dimensional and small in sample size. The histrograms for the target can be seen in Fig. 16a. One immediately notices the uneven distribution and the particularly low number of instances with class 4. As a Machine Learning practitioner, one now has the option to slightly reframe the task into a binary classification problem if the uneven distribution and low number of samples prove to be hurdles for training an accurate model. This however did not prove to be the case here. The most sensible grouping is then to unify all incident classes 1 to 4 into a single group. The resulting data set then has a more even distribution of classes, as can be seen in 16b.

(a) Histogram of classes for multi class classification.



(b) Histogram of classes for binary classification.

Features			
numerical		categorical	
name	range	name	range
age	29 - 77	sex	0, 1
trestbps	94 - 200	cp	0, 1, 2, 3
chol	126 - 564	fbs	0, 1, 2
thalach	71 - 202	restecg	0, 1, 2
oldpeak	0 - 6.2	exang	0, 1
-	-	slope	0, 1, 2
-	-	ca	0, 1, 2, 3, 4
-	-	thal	3, 6, 7 $\rightarrow$ 0, 2, 1

Table 2: Features in the Heart Disease data set."

6.2 Data preprocessing

As mentioned, the data set was already processed quite heavily beforehand so almost every categorical feature has already been encoded into numerical values, e.g. features such as "sex", the type of chest pains "cp", the slope type of the peak exercise ST segment "slope" were encoded into integers of order $\mathcal{O}(1)$, so $1, 2, 3, 4, ...$, and are therefore already of similar range. The only feature that was significantly different was "thal", because it was encoded as shown in Table 2 and according to our research, a "reversable defect" (7) was a sign of healthier tissue than a "fixed defect" (6). 3 previously stood for normal or no defect. We changed the encoding as indicated in Table 2.

The numerical features have different ranges, as can be seen in 2, and we therefore chose to employ a scaler (sklearn StandardScal) during the data preprocessing and will compare preprocessing with and without use of the scalar.

The were also 5 samples that contained missing values in the columns "thal" or "ca". We investigated three different strategies how to deal with them, (1) simply dropping them as they only represent $\approx 1\%$ of all samples, (2) using a MLP-Imputer to impute the values and (3) assigning fixed values to them (0 for "ca" and 1 for "thal"). However, as they all performed very similarly, we will only show results for approach (1) as it is the simplest.

6.3 Comparison of different classifiers

We will compare the different classifiers based on accuracy and f1-score, because we are looking for a balanced model and don't really know too much about the impacts of our models prognosis. A customized cost matrix would need input from various professionals in the medical field to properly gauge the impact of the different fail-to-predict cases. However, if we had to, that would be our recommendation for making a model production and real-world ready.

We will also not dive further into either the training or inference times needed for the different models, since it doesn't matter so much for such a small data set, both in dimensionality and number of samples.

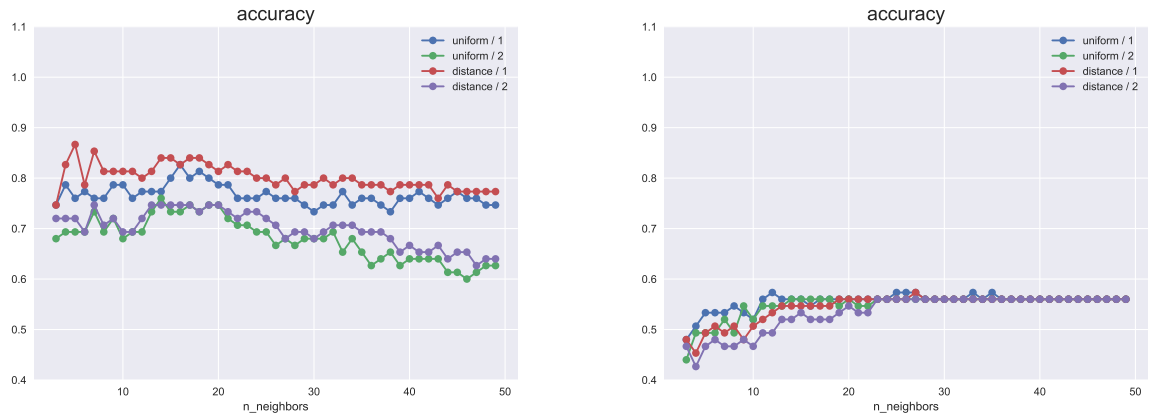
The hyperparameters we used for each classifier are shown in Table 3.

KNN		RFC		MLP	
hyper-parameter	tested values	hyper-parameter	tested values	hyper-parameter	tested values
number of neighbors	$3, 4, ..., 49$	number of estimators	[1,8,10,12,15, 20, 50 , 1e2, 1e3]	learning rate alpha	[1e-5, 1e-4, 1e-3, 1e-2, 1e-1,1,10,1e2]
weights	uniform, distance	max features per split	sqrt, log2	hidden layer sizes	[(50,50), (5,5), (5,50), (50,5)]
p	1, 2	criterion	gini, entropy	activation function	tanh, relu

Table 3: Hyperparameters.

6.3.1 K-nearest Neighbors classifier

In Fig. 17, we can clearly see the impact of using scaled features has on the KNN classifier. The takeaway - unsurprisingly - being that KNNs can really only function properly without adequately scaled data, i.e. data that has similar ranges per feature.



(a) Accuracy scores for KNNs trained with scaled features (StandardScaler).

(b) Accuracy scores for KNNs trained without scaled features.

Fig. 17: Comparsion between different KNN classifiers with and without scaled features as input.

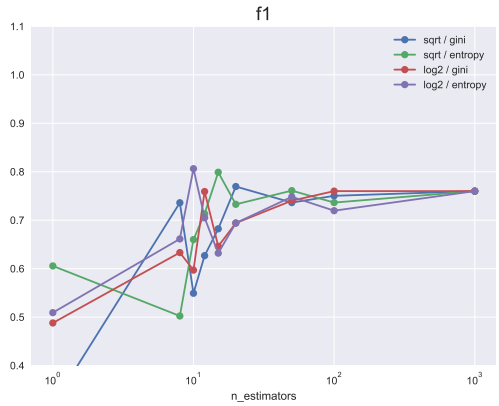
In Fig. 17, we especially note the fact that models using $p = 1$, so Manhattan distance, perform better than those using Euclidean distance measures. Additionally, when using a KNN with neighbors weighted by distance, it also slightly outperforms those weighted uniformly. The KNN classifiers we tested performed the worst for this task overall.

6.3.2 Random Forest classifier

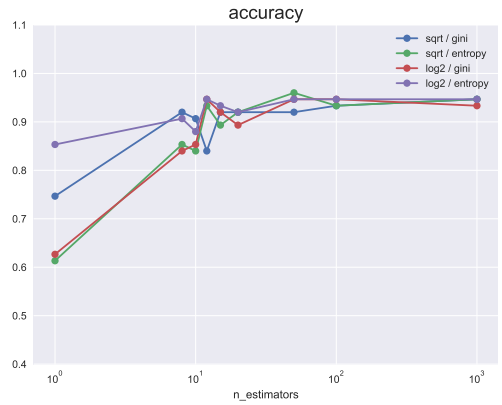
The RF classifiers were not affected by unscaled data at all and there was no clear trend visible among the different criterions and number of features considered for a split. We noticed though, that the sweet spot for the number of estimators in the forest is likely to be in the range of 12 to 20 (or 50) as these results repeated themselves across different trys, also with different (random) splits of the data set (into train- and test-set). For reference, see Fig. 18. As mentioned before,

this data set is very small and low dimensional so pruning the RFCs further for better inference performance was not of concern to us.

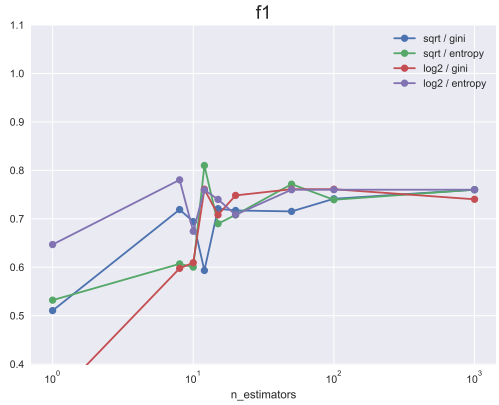
(a) Accuracy scores for RFCs trained with scaled features (StandardScaler).



(b) F1 scores for RFCs trained with scaled features (StandardScaler).



(c) Accuracy scores for RFCs trained without scaled features.



(d) F1 scores for RFCs trained without scaled features.

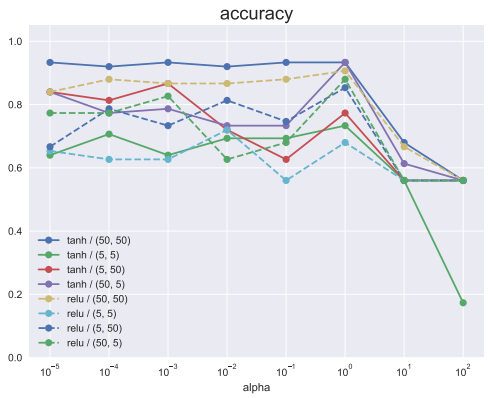
Fig. 18: Comparsion between different RFC classifiers with and without scaled features as input.

6.3.3 Multilayer Perceptron classifier

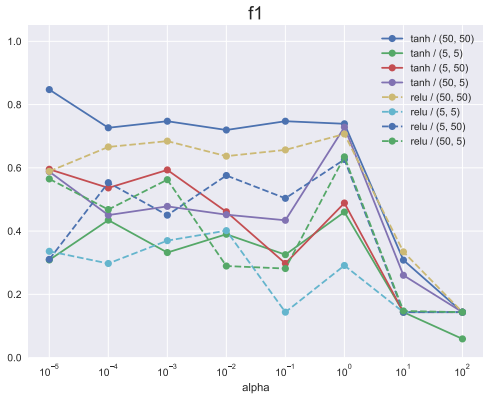
The MLP classifier behaves similarly to the KNN classifier, in that a properly scaled set of features is needed, compare Fig. 19a to Fig. 19c. There does not seem to be too much of a difference between the different activation functions, although tanh delivered the best result among the tested MLPs. What all classifiers have in common, is that they work better with a regularization alpha ¡ 1, as they deliver worse results with the bigger alphas. MLPs with bigger layers (more bubbles) performed better in this task. Those using 2 layers with 50 bubbles each performed better and they indeed delivered similar results for various alphas. Here, the clear winner was then decided by the better F1 score as the MLP with the hyperparameter set (alpha=1e-5, hidden layers = (50,50), activation = tanh).

The final selection was then conducted by looking at the confusion matrices for the best RF classifier in Fig. 20b and the best MLP classifier in Fig. 20a. The RF classifier was unable to classify even a single instance of class 4 correctly as it could not distinguish between class 3 and 4. The MLP classifier also had some trouble distinguishing the two classes, but was able to at least identify 3/5 of the samples of class 4 in the test set correctly. We conclude by "giving the win" to

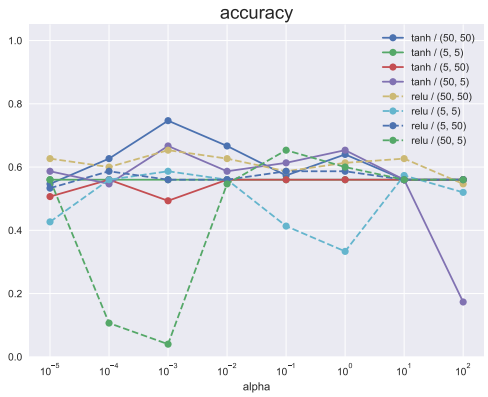
the MLP classifier in this case, as although it reached a slightly lower accuracy score, it was overall able to distinguish between all 5 classes and was only every wrong by +/- 1 class.



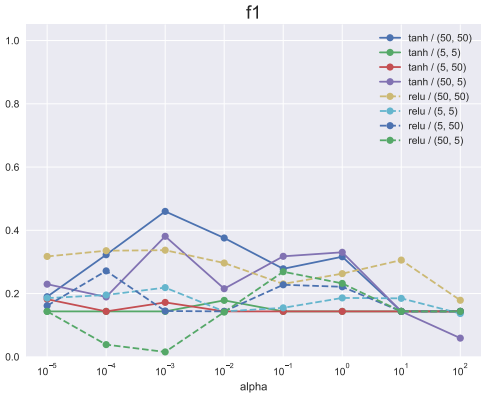
(a) Accuracy scores for MLPs trained with scaled features (StandardScaler).



(b) F1 scores for MLPs trained with scaled features (StandardScaler).

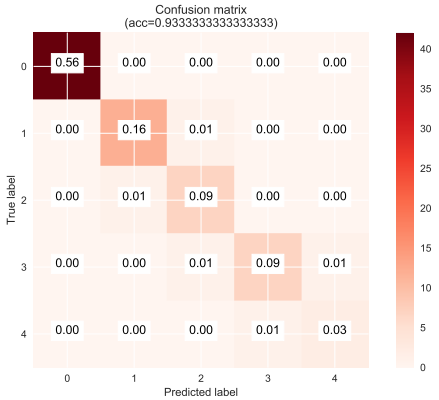


(c) Accuracy scores for MLPs trained without scaled features.

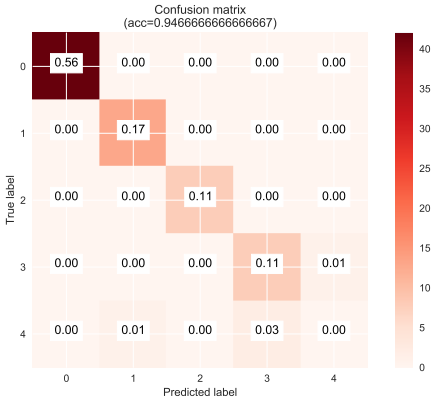


(d) F1 scores for MLPs trained without scaled features.

Fig. 19: Comparsion between different MLP classifiers with and without scaled features as input.



(a) Best scoring MLP classifier with scaled data.



(b) Best scoring RFC classifier with scaled data.

Fig. 20: Confusion matrices of the two best candidates.