Assignment 1

Group 1

Jonas Kompauer, 11776872
Lukasz Sobocinski, 12123563
Florian Lackner, 11704916

1. Experiment design

Preprocessing

All the datasets needed to be preprocessed to clean the data for the classification algorithms.

Experiments and optimisation

Every classification algorithm has parameters that can be tuned to improve the fit. Therefore, as the first step we needed to find optimal ones for each case. To do it, the GridSearchCV method from sklearn library was used. It performs an exhaustive search over specified parameters' ranges and looks for ones with the highest score. The score is obtained from the cross validation method and accuracy is used as the metric.

For each algorithm and classifier, we ran GridSearchCV (using 5 or 10-fold CV) to find optimal parameters. We decided to do it on the whole dataset, and then fit a new model on the training data, and test it on a test set. So, first of all, we used Cross Validation to evaluate which parameters are the best, and then we used the holdout method to evaluate its performance. We considered that it might violate the principle of not using the test data for training; however, here we will use it only for parameters finding, not for the classification. Moreover, CV is still used, so we have train and test splits. We tried to find the optimal parameters using only the training data, but it is very unstable and it depends on the train-test split that was made.

Then, we want to run the same experiment, but on the data that is not scaled. Theoretically, unscaled data should be a problem for Naive BAyes and Decision Tree, but it is important for kNN as the classifier is measuring distances.

We decided to use the following three classifiers for each dataset:

kNN

In k-NN classifier we use a nearest Neighbor Algorithm which determines the class of a given input based on the k nearest Neighbors, in this Assignment we could tune the following parameters:

- n_neighbors - number of neighbors.
- weights - if a decision should be based on majority voting or should the weight of the neighbour change with the distance from the point.
- p - which distance measurement needs to be used. p=1 uses Manhattan distance and p=2 stands for euclidean.

For GridSearchCV, we decided to use n_neighbors in range 1-40, and try each combination of weight and p. It can be observed that the weights parameter has completely no impact on the response. Euclidean distance is slightly superior than Manhattan for small and big k neighbours. On every plot we can observe that the accuracy quickly improves from 1 to 4 neighbours and then slowly decreases with the increase of the n_neighbors. The accuracy is quite high, which means that the boundary between classes can be drawn precisely. Therefore, it is important to have low bias. This might be the reason why our k is small. Big k would underfit the data.

Naive Bayes

For the Naive Bayes we used Gaussian Naive Bayes where only one parameter needed to be tuned - var_smoothing. It artificially adds a user-defined value to the distribution's variance from the training set. It raises the importance of samples that are further away from the distribution mean. Unfortunately, scikit learn does not allow to tune sigma and theta parameters.

GridSearchCV (using 10-fold CV) was used to find the optimal parameter. 100 parameters placed evenly between log(1) and log(-10) on a log scale were checked.

Decision Tree

In the Decision Tree we decided to tune these parameters:

max_depth - maximal depth of the tree

min_samples_split - the minimum number of samples contained in a node required to split it. The parameter is calculated as: number of samples/number of all samples.

criterion - the measure used to compute the best split.

splitter - “best” means that the feature with the best split quality will be chosen. If it is 'random', it means that the feature will be selected randomly, but the probability of the feature being selected is higher the higher the quality of the feature.

ccp_alpha - it is used for Minimal Cost-Complexity pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. If the parameter is equal to 0, no pruning is performed.

As the first step, we decided to run GridSearchCV (using 10-fold CV) to find the optimal combination of the parameters. We decided to analyse max_depth in range 1-21, min_samples_split in range 0.001-0.3 with 0.01 step and check it with all the combinations of the rest of parameters.

Evaluation

To evaluate effectiveness we decided to use accuracy, precision and recall. Moreover, we created the confusion matrix for each one of them. These parameters allowed us to see how algorithms perform and what their strong and weak sides are. Finding for each dataset and classifier the best parameter (in a given range) enables us to compare them with each other.

To assess the efficiency of the classifiers, we calculated times needed to fit and predict the model.

2. Dataset Diabetes

https://www.openml.org/d/42608

Dataset introduction

The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. It consists of 9 columns. Eight of them are features and include data from patients. The outcome column describes, if the patient has diabetes or not.

Data Exploration and Preprocessing

There are missing values and unfortunately they are not marked as “missing”, but a zero was used for those values. Because it only has eight features we used it to test our selfmade framework. Furthermore, each feature is numeric, so it is not necessary to encode them. For Preprocessing we only had to scale each feature.

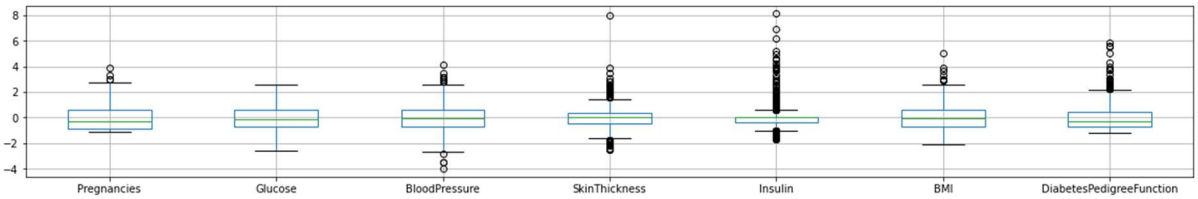


Fig 1.2. Boxplots of each Feature in the dataset

After analysing the data, we found out that each feature of the set is normally distributed. You can see boxplots of each feature in Fig. 1.1. Some of them have a skew because for example there are more women with 0 pregnancies than women with 15 pregnancies.

Carry out the classification

kNN

On Fig 1.2, there are plots of accuracy versus the number of neighbours for each combination of p and weights, but they are almost the same, so the weights do not have any impact on the accuracy.

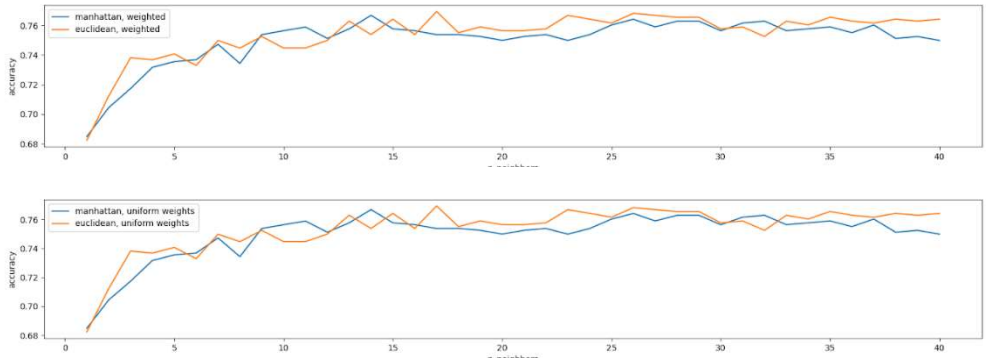


Fig 1.2. Plots of accuracy vs number of neighbours using different weights and p parameters in kNN. Graph on the top shows the relation for decision based on majority voting, one below depicts the relation for weighted approach. Both plots are the same, because this parameter does not change the accuracy.

Parameter and results with scaling: n_neighbors = 30, p = 2 (euclidean distance), weights - does not matter which one we choose, we used majority voting

Accuracy for best parameters computed by CV: 0.775

Accuracy on the test set: 0.727

Parameter and results without scaling: n_neighbors = 30, p = 1 (manhattan distance), weights - does not matter which one we choose, we used majority voting

Accuracy for best parameters computed by CV: 0.750

Accuracy on the test set: 0.714

Naive Bayes

The results of the naive Bayes approach are quite similar to the kNN, but as it can be seen further down, the time for testing and predicting is much smaller. In Fig. 1.3. one can see that there is a significant difference between the runs with scaling and without scaling. Although, the result is almost the same, getting the best parameter is very different. With the scaled data it is much easier to find a reasonable value for var_smoothing. Finding it without scaling the data, the peak of the accuracy is hard to find.

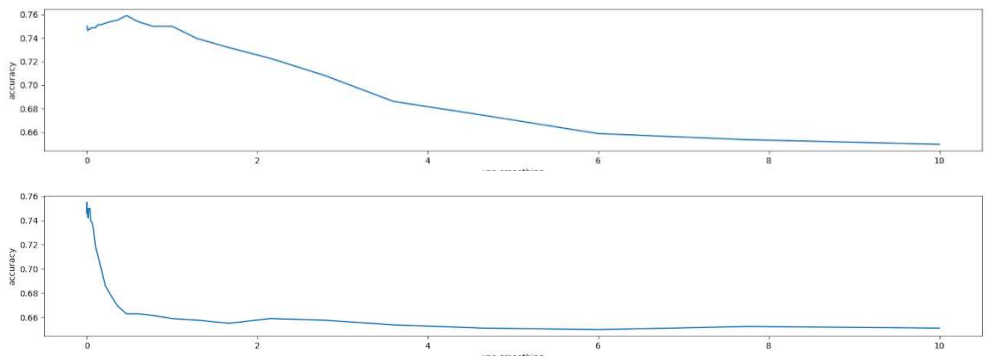


Fig 1.3. Plots of accuracy vs var_smoothing. Graph on the top shows it with scaling the data upfront, the other one shows it without scaling.

Parameter and results with scaling: var_smoothing = 0.464

Accuracy for best parameters computed by CV: 0.76

Accuracy on the test set: 0.734

Parameter and results without scaling: var_smoothing = 0.001

Accuracy for best parameters computed by CV: 0.76

Accuracy on the test set: 0.708

Decision Tree

Although the Decision Tree classifier performed a bit better than the other classifier, the results are not significant, because of the small dataset. Also the scaling factor is not very important, as one can see in Fig. 1.4.

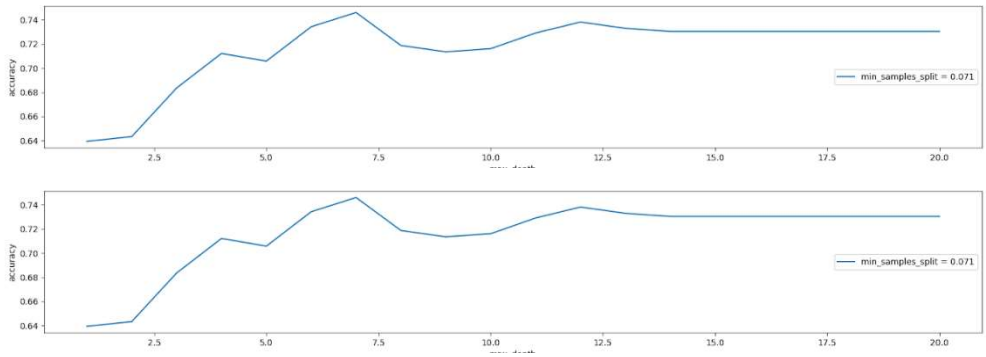


Fig 1.4. Plots of accuracy vs var_smoothing. Graph on the top shows it with scaling the data upfront, the other one shows it without scaling.

Parameter and results with scaling: criterion = 'entropy',, max_depth = 6, min_samples_split = 0.151, random_state = 123, splitter = 'best'

Accuracy for best parameters computed by CV: 0.762

Accuracy on the test set: 0.708

Parameter and results without scaling: criterion = 'entropy', max_depth = 6, min_samples_split = 0.15099999999999997, random_state = 123, splitter = 'best'

Accuracy for best parameters computed by CV: 0.762

Accuracy on the test set: 0.740

Evaluate and analyse the performance

As it can be seen in the table, the three classifiers perform very similarly and reach an accuracy of about 0.72.

	kNN		Naive Bayes		Decision Tree	
	scaled	not scaled	scaled	not scaled	scaled	not scaled
accuracy	0.727	0.714	0.734	0.708	0.708	0.740
precision	0.630	0.625	0.651	0.600	0.576	0.640
recall	0.537	0.463	0.519	0.500	0.630	0.593

Also the confusion matrices are very similar, which probably came from the small dataset

scaling								
kNN	true	false	Bayes	true	false	Tree	true	false
positive	29	17	positive	28	15	positive	34	25
negative	25	83	negative	26	85	negative	20	75

no scaling								
kNN	true	false	Bayes	true	false	Tree	true	false
positive	25	15	positive	27	18	positive	32	18
negative	29	85	negative	27	82	negative	22	82

There are big differences in the execution time of the classifiers. The Time for predicting the result is normally smaller than the time for the testing, but at the kNN classifier, it is inverted.

Classifier	Time fitting in seconds	Time predicting in seconds
kNN	0.0099	0.0407
Naive Bayes	0.0023	0.0008
Decision Tree	0.0042	0.0005

3. Dataset Speed Dating

https://www.openml.org/d/40536

Dataset introduction

The “Speed Dating” dataset represents data about 8379 Speed Dates and lots of information for each participant of a speed Date. There are 121 features which range von age to interest to expectations on partners. Approximately half of the features have numeric values and the other half are nominal values with either intervals/ranges or Strings and a lot of features have missing values. The target is to find out if both partners agree on a match.

Data Exploration and Preprocessing

The dataset is large and as mentioned above has a lot of missing values, so we decided to preprocess the missing values for the numeric feature with the mean value of all the non-missing values from this feature. For the missing nominal values we took the most frequent value of all other values to fill it. This is done for the test and train set separately to not include any information from the test set in the train set. Additionally, we scaled the numeric values using a Z-score standardisation, to ensure that there is no impact given by the value range which could have unwanted effects. This is mostly used for the kNN Algorithm as it is the only one (of our chosen algorithms) relying on distances. After filling the missing values we have another preprocessing step, namely the encoding of the nominal values, for which we just chose the One Hot Encoding.

Carry out the classification

kNN

The Figure 2.1 below shows the difference between Manhattan and Euclidean distance used. Here we compare the difference between scaled feature values (left plot) and non-scaled feature values (right plot). First difference we notice is that the euclidean distance is better when scaled whereas the manhattan distance produces better accuracy for not scaled values. The left plot peaks at 6 neighbors while the right plot has also good accuracy with 6 neighbors but peaks at 23. When using scaling we get a better accuracy but its negligible as, the accuracy with scaling is 0.842 (using 6 neighbours and euclidean distance) , while without scaling it is 0.841 (using 23 neighbors and the manhattan distance). This is quite surprising as kNN should benefit from scaling because it uses distances and therefore if we normalize it, it should become more accurate.

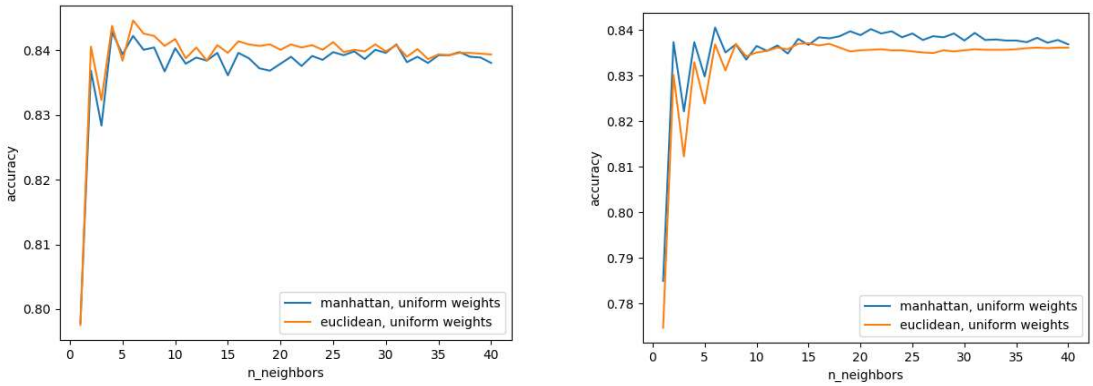


Fig 2.1 Plots of accuracy vs number of neighbours using different weights and p parameters in kNN. Graph on the left shows the accuracy when the features are scaled, the right plot shows the accuracy when the features are not scaled.

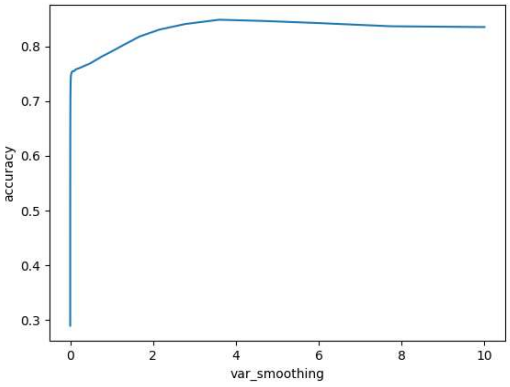
In the end we got the following parameters:

n_neighbors = 6; weights = majority voting; distance = euclidean distance

This data was generated using a 10-fold CV, we also use holdout for comparison with a 80% train-set and 20% test-set where we got slightly better results with a scaling accuracy of 0.852 and an accuracy without scaling of 0.841, so here is scaling slightly better but still not much.

Naive Bayes

We, again, compare the accuracy between scaled and not scaled numeric values, even if it should not really matter, because Naive Bayes does not operate on distances. We see in Figure 2.2 on the left side the plot with scaled variables. The accuracy peaks slowly increases until var_smoothing=3.59, with accuracy value of 0.848. The right plot peaks at 0.77 and does not change much with increasing var_smoothin, the highest accuracy value is 0.835. So we see that there is no significant difference between scaling and not scaling.



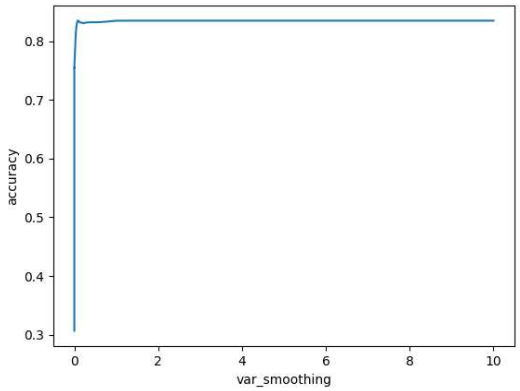


Fig 2.2 Plots of accuracy vs var_smoothin in NaiveBayes. Graph on the left shows the accuracy when the features are scaled, the right plot shows the accuracy when the features are not scaled.

In the end we got the following parameters:

var_smoothing = 3.5938

This data was, again, generated using a 10-fold CV, we also use holdout with a 80% train-set and 20% test-set where we got slightly better results with a scaling accuracy of 0.859 and an accuracy without scaling of 0.841, so here is scaling slightly better but still not much.

Decision Tree

When classifying with the Decision Tree it is hard to find the best parameters, as there are some of them with float values, so to find good values we had to test a lot. For the Speed Dating dataset using Gini impurity was always superior in our tests so we decided to stick to it.The max depth for the tree was tested in a range from 1 to 20 as shown in Figure 2.3 on the left. The accuracy peaks at max depth 7 and after that Oscillates but never surpasses the accuracy of max depth 10. This data was gathered by setting "min_sample_split" to 0.03 so the number of samples required to split an internal node is 0.03 * (Number of samples). This appears to be the best value according to our test for this dataset. In other dataset (Breast cancer and diabetes) this parameter is lower because the dataset size is also lower, so a higher fraction is needed. The right plot of Figure 2.3 shows how accurate the prediction is with respect to the "min_sample_split" parameter. We see that for up to a fraction of 0.12 the max depth has different accuracies but for every value higher than that the accuracy is for all of the three tested depths the same.

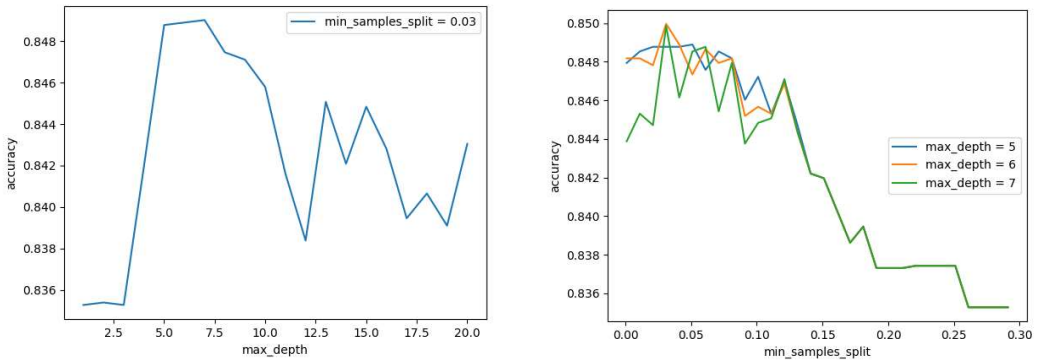


Fig 2.3 Plot of the maximum depth in relation to the accuracy on the left. On the right, plot accuracy in relation to the min_sample_split parameter given a max_depth.

We decided to use Minimal Cost-Complexity Pruning to get better results by pruning large cost complexity subtrees, but that only led to worse results, only using a value of 0 leads to good results, all higher values decrease only the accuracy. Pruning probably loses too much information in this context.

With this we get the best parameters we found:

criterion: 'gini', max_depth: 7, min_samples_split: 0.03, splitter: 'random'

With the accuracy for best parameters computed by CV: 0.842

When testing with holdout we have an accuracy of 0.859.

Results

For all classifiers we also timed the fitting and the prediction process, we got following results:

Classifier	Time fitting in seconds	Time predicting in seconds
kNN	0.0386	0.5724
Naive Bayes	0.0848	0.0397
Decision Tree	0.2387	0.0234

We see that the kNN is fastest in training the algorithm but takes up to 20x more time than the other algorithms. This probably is because of the high dimensions/ large number of features in this dataset, as kNN does not scale well with

higher dimensions. Naive Bayes is fast in both fitting and predicting as it does not struggle as much as the other algorithms with larger dimensions. The Decision Tree takes up to 0.2 seconds more time to fit the Tree as it has to consider the large number of features, but after that it is the fastest algorithm in prediction as we do not have to calculate more things but just have do check where we have to go down in the decision tree based on known input features.

Apart from speed, every algorithm performed more or less the same, considering accuracy in the table below we also see the precision and recall for each classifier:

	kNN		Naive Bayes		Decision Tree	
	scaled	not scaled	scaled	not scaled	scaled	not scaled
accuracy	0.852	0.841	0.859	0.841	0.859	0.857
precision	0.635	0.545	0.624	0.506	0.656	0.604
recall	0.176	0.044	0.29	0.277	0.250	0.303

We see that if we do not scale the data we lose points in kNN and Naive Bayes on all three performance measurements. Additionally we conclude that we have more false negative predictions as the recall value is much lower than the precision value, this can also be seen in the sparse Matrix of all classifiers for this data set:

Scaled values:

kNN	true	false	Bayes	true	false	Tree	true	false
positive	47	27	positive	78	47	positive	67	35
negative	220	1382	negative	189	1362	negative	200	1374

Non Scaled Values:

kNN	true	false	Bayes	true	false
positive	12	10	positive	74	72
negative	255	1399	negative	193	1337

Interesting is the fact that, for kNN, the false negative rate gets lower when scaling the values, and also the true negative. The true positive and false negative predictions both increase. For Naive Bayes the only significant values that change when scaling is that the true negative predictions increase and the false positive decrease.

In the end we can say that Naive Bayes would probably be the best classifier, out of our chosen three, because it is fast in fitting and predicting and has good effectiveness.

4. Dataset Purchase

https://www.kaggle.com/c/184702-tu-ml-ws-21-location

Dataset introduction

The “Purchase” dataset represents data about 10000 customers and information about what they are purchasing. Each of the 600 binary attributes represents one product and if the customer bought it or not. The target attribute consists of 100 classes where each customer is categorized into one class.

Data Exploration and Preprocessing

The dataset is quite simple and good to process, because there aren’t any missing values or duplicate rows in it. Furthermore, as it only consists of binary type attributes (so either 0 or 1) there is no need to preprocess them as their

value range is the same. The important difference to our other datasets is that there are multiple classes for the target and not just binary outcome, there exists 100 different classes we can predict, this has to be considered when comparing the accuracy between datasets.

Carry out the classification

kNN

On the Figure 2.1 below, the Plots show the accuracy with respect to the number of neighbors used for the kNN. Each Plot shows the difference between the Manhattan and the euclidean distance, but in both Plots they are the same, this is because the Dataset contains only binary values of 0 and 1 for each feature, making both distance approaches equal. Also it does not matter if we use a majority voting approach (left Plot) or a weighted approach (right plot). The accuracy with respect to the number of neighbours and peaks at 39.

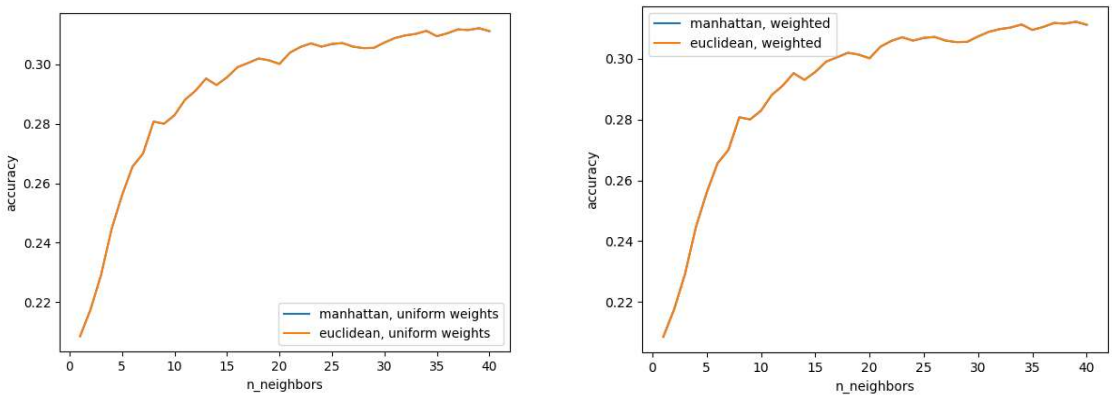


Fig 3.1 Plots of accuracy vs number of neighbours using different weights and p parameters in kNN. Graph on the left shows the accuracy in relation to the number of neighbours based on majority voting, the right plot shows the same as the left only based on a weighted approach.

In the end we got the following best parameters for kNN and the Purchase dataset:

n_neighbors = 39; weights - majority voting, distance=euclidean distance

With these parameters we got an accuracy of 0.315 when using a 10-fold CV. When testing it with hold out we got a similar result, namly an accuracy of 0.314.

Naive Bayes

For the Naive Bayes we only had to figure out one parameter, namely “var_smoothing”. Figure 3.2 shows the changing of accuracy with respect to the “var_smoothing” parameter. The accuracy increases up to a value of 0.215, more smoothing would only result in lower accuracy.

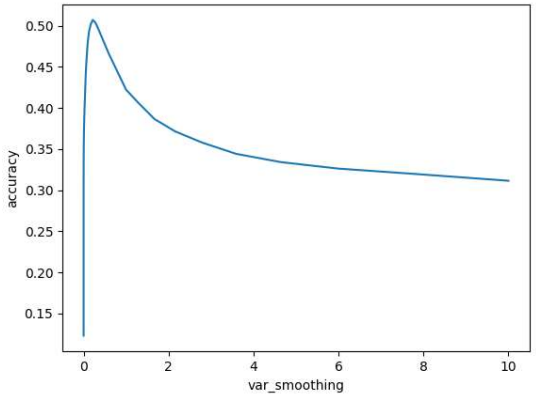


Fig 3.2 Plots of accuracy vs var_smoothin in NaiveBayes for purchase dataset

With this parameter and using a 10-fold CV we got an accuracy of 0.507, which is lower as in the other dataset but because we do not have just a binary outcome but multiple target classes.

When testing it using holdout and a 80% train set and 20% test set the accuracy decreased a small amount, to 0.506

Decision Tree

For the Speed Dating dataset using Gini impurity was always superior in our tests for the Decision Tree so we decided to stick to it.The max depth for the tree was tested in a range from 1 to 20 as shown in Figure 2.3 on the left. The accuracy peaks at max depth 9 and after that slowly decreases. This data was gathered by setting “min_sample_split” to 0.001. This appears to be the best value according to our test for this dataset, as we have a large amount of data. The right plot of Figure 2.3 shows how accurate the prediction is with respect to the “min_sample_split” parameter. We see that the accuracy is only decreasing with increasing min_sample_split value because larger subtrees would need more nodes to split.

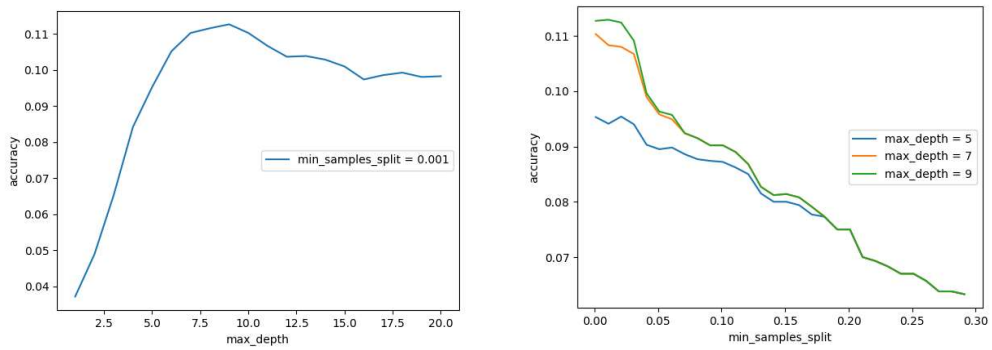


Fig 3.3 Plot of the maximum depth in relation to the accuracy on the left. On the right, plot accuracy in relation to the min_sample_split parameter given a max_depth.

As for the other Datasets we decided to try to use Minimal Cost-Complexity Pruning to get better results by pruning large cost complexity subtrees, but that only led to worse results.

We end up with the With this we get the best parameters we found:

criterion: ‘gini’, max_depth: 9, min_samples_split: 0.001, splitter: 'random'

With the accuracy for best parameters computed by CV: 0.113

When testing with holdout we have an accuracy of 0.117.

Results

For all classifiers we also timed the fitting and the prediction process, we got following results:

Classifier	Time fitting in seconds	Time predicting in seconds
kNN	0.0256	8.1833
Naive Bayes	0.1568	1.3986
Decision Tree	0.6924	0.0500

We see that the kNN is fastest in training the algorithm but in predicting the slowest because of the large number of features. This is also the reason why the Decision Tree takes the longest with fitting, but once the tree is built it predicts really fast.

Apart from speed, every algorithm performed more or less the same, considering accuracy in the table below we also see the precision and recall for each classifier:

	kNN	Naive Bayes	Decision Tree
accuracy	0.506	0.314	0.117
precision	0.499	0.376	0.095
recall	0.491	0.283	0.099

5. Dataset Breast Cancer

https://www.kaggle.com/c/184702-tu-ml-ws-21-breast-cancer/overview

Dataset introduction

Dataset consists of the data collected from analysis of cytological features of women' breast based on digital scans of them. Features are extracted from the image using a computer program called Xcyt. It computes 10 features of every cell in the sample, and calculates the mean, standard deviation and extreme value for the cells. Target variable "class" indicates if the data was measured for a breast with cancer. Therefore, in the dataset, there are 30 features, 1 target variable and the ID of the measurement.

Data Exploration and Preprocessing

We have started with exploration of the dataset. Firstly, we checked if there are any missing values or duplicated rows, but there were none of them. Also, features are of quantitative data type, so 1-to-N Coding is not needed in this case. The next step was to plot histograms and box plots of each variable to see how they are distributed and if there are any outliers. The distribution of most of the variables looks similar to the Gaussian distribution. It is good, as most of the ML algorithms assume it. Almost all the features are right skewed. There are some outliers, however they are not that significant. Thus, we decided not to do anything about them. Moreover, some variables are correlated and we should take it into account.

Data was shuffled and split into train and test data sets (in 80% to 20% proportion). Random state was set to a constant for all the tests to ensure that our results can be reproduced. Then, the unnecessary column ID was dropped and the Dataframes were scaled for kNN to work correctly (with Z-score standardisation). We did the preprocessing on the train and test data separately, to avoid any interference between them.

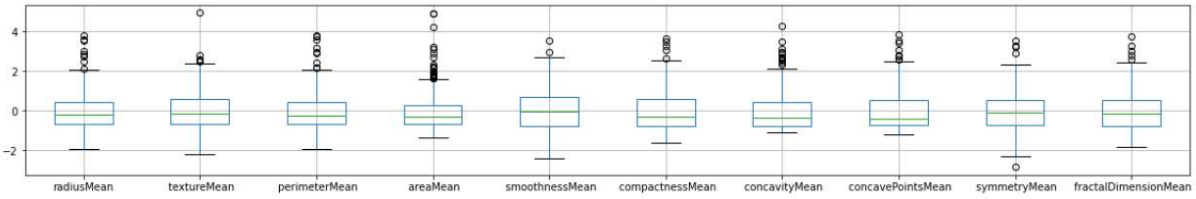


Fig 4.1. Box plots of features describing mean from the model

Carry out the classification

k-NN

On Fig. 4.2, there are plots of accuracy versus the number of neighbours for each combination of p and weights. For k between 7 to 18 Manhattan seems to be better.

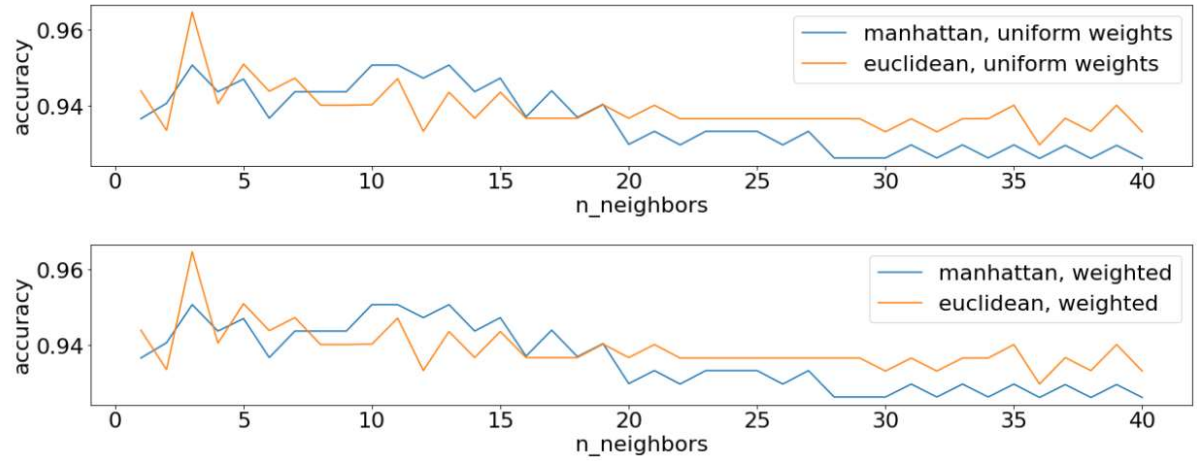


Fig 4.2. Plots of accuracy vs number of neighbours using different weights and p parameters in kNN for scaled data. Graph on the top shows the relation for decision based on majority voting, one below depicts the relation for weighted approach. Both plots are the same, because this parameter does not change the accuracy.

In the end we got the following parameters:

n_neighbors = 4, weights- majority voting (does not matter which one we choose) and euclidean distance.

Accuracy for best parameters computed by CV for scaled data: 0.965, for unscaled data: 0.933

Accuracy using holdout method for scaled data: 0.982, for unscaled data: 0.930.

We could observe that the algorithm works better for scaled data. It is reasonable, as it operates on the distance, therefore the data should be scaled.

Naive Bayes

The best accuracy was found for var_smoothing=0 and unscaled data (Fig. 4.3). For scaled data optimal var_smoothing is at 0.278.

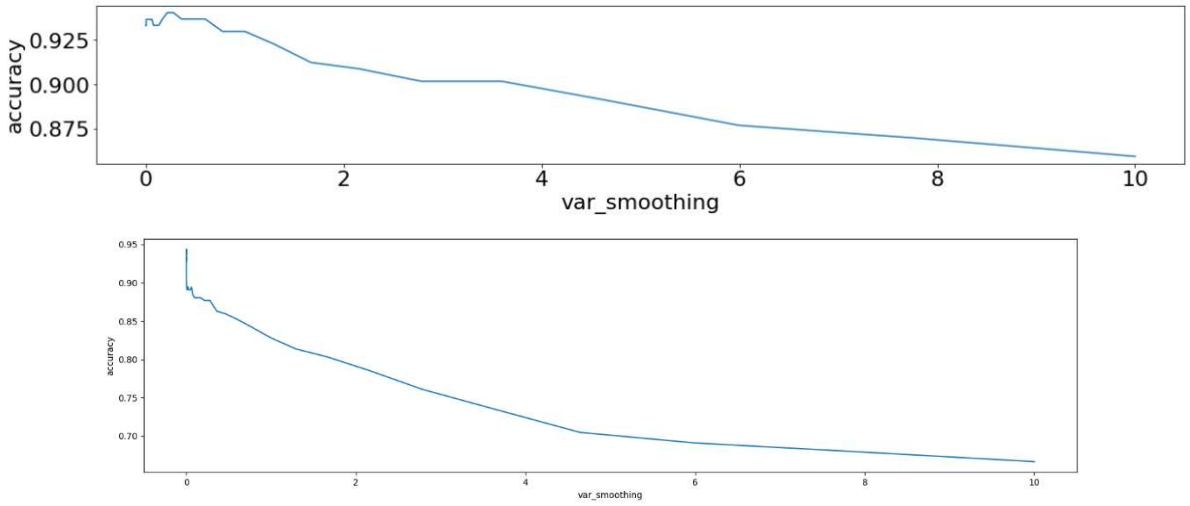


Fig 4.3. Plots of accuracy vs var_smoothing for Naive Bayes. Plot above corresponds to scaled data, one below to unscaled data.

Accuracy for best parameters computed by CV for scaled data: 0.970, for unscaled data: 0.944

Accuracy using holdout method for scaled data: 0.930, for unscaled data: 0.947.

We expected that the accuracy will be the same for both cases, because Naive Bayes should not depend on scaling. However, it was not a case. Moreover, it should be noted that the accuracy on the test set is strongly affected by the training-test split.

Naive Bayes algorithm makes the assumption that variables in the model are not correlated. However, in this case it is not true. Therefore, we decided to apply PCA in the data preprocessing to remove this problem. Of course optimal var_smoothing needed to be recomputed. Unfortunately, the accuracy was worse: 0.895. We also tried to delete columns that are strongly correlated to others manually, but it also did not improve the fit.

Decision Tree

The optimal parameters were: criterion:

'gini', max_depth: 9, min_samples_split: 0.071, splitter: 'random'.

With the accuracy for best parameters computed by CV: 0.940

Accuracy on the test set was very unstable, what needs to be analysed. First of all, we plotted how the accuracy changes according to changing max_depth and min_samples_split with random splitter and entropy criterion. To enable easier comparison a seed was fixed for all the runs.

On Fig. 4.4, it can be observed that after reaching max_depth of 9 the accuracy does not change. It means that the tree never reaches more than this number of levels. It is caused by the min_samples_split parameter, which restricts creating new leaves. Those parameters work together to do pre-pruning avoiding overfitting and making the model relatively simple.

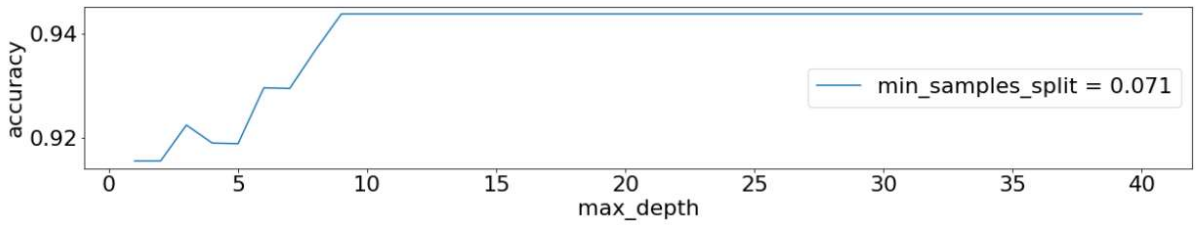


Fig 4.4. Plots of accuracy vs max_depth of Decision Tree for scaled data.

On Fig. 4.5, it can be observed how accuracy changes together with min_samples_split for some selected max_depth parameters. When min_samples_split is increased too much, too few new leaves are created and model underfits, which makes the accuracy worse.

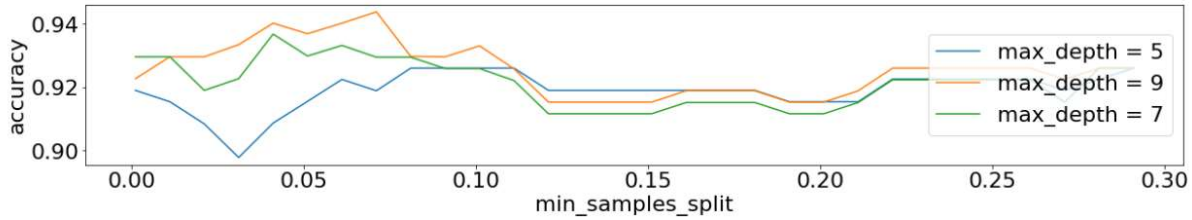


Fig 4.5. Plots of accuracy vs min_samples_splt for Decision Tree for 3 different max_depths: 5, 7 and 9 for scaled data.

We had problems with the stability of the result. To overcome it, we decided to use Minimal Cost-Complexity Pruning. We tried to optimise it for different values of cpp_alpha, but the accuracy only got worse (Fig. 4.6). Maybe pruning does not help because our tree is already quite small due to the pre-pruning techniques that we used.

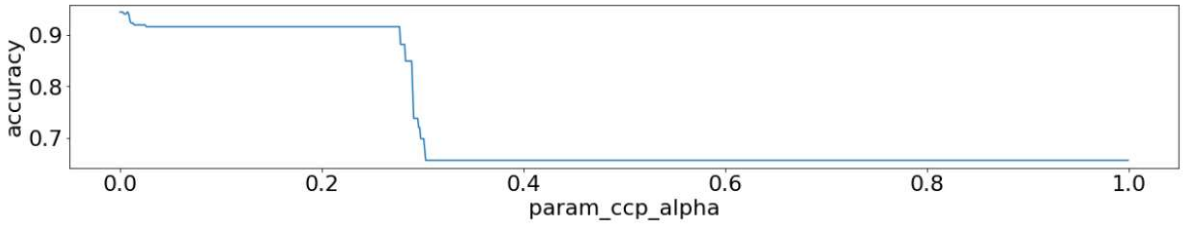


Fig 4.6. Plots of accuracy vs cpp_alpha for Decision Tree for scaled data.

Accuracy results were still unstable, so to obtain the accuracy we decided to compute the mean of 10 random train-test splits. It was: 0.906 for scaled data and 0.912 for unscaled data.

For these 3 tests, the training set of the data was used. However, for Kaggle competition we used the whole dataset to improve prediction.

Evaluate and analyse the performance

For all classifiers, we calculated how much time does fitting and predicting take:

Classifier	Time fitting in seconds	Time predicting in seconds
kNN	0.002	0.002
Naive Bayes	0.001	0.002
Decision Tree	0.001	0.003

It can be observed that values are very low, but it is caused by the fact that the dataset is relatively small. It is difficult to draw conclusions from such low numbers; however, it can be observed that Naive Bayes is the fastest. Moreover, in kNN fitting is a very quick process and prediction park takes more time.

The second step was to calculate performance measures for best parameters in each algorithm:

	kNN		Naive Bayes			Decision Tree	
	scaled	not scaled	scaled	not scaled	PCA	scaled	not scaled

accuracy	0.982	0.930	0.930	0.947	0.895	0.906	0.912
precision	0.947	0.889	0.937	0.941	0.833	0.782	0.810
recall	1	0.889	0.833	0.889	0.833	1	0.944

Overally, the accuracy achieved was very high. In the best case, for kNN only one point was classified incorrectly. It means that it is difficult to assess reliability, as one or two wrongly classified samples change the outcome significantly. Also, we have to take into account that train-test split is done randomly so the outcome varies slightly depending which seed we use. It also should be noted that in this case, the most important measure is recall. In my opinion, in this case False Negatives are worse, because it may

However, it can be observed that kNN performed on scaled data has the best accuracy (it was also the best on Kaggle competition). Moreover, it was the most stable considering different test-train splits. Maybe, it is the best algorithm for this dataset, because it doesn't make any assumptions about the data, which contains some correlations and not-normal distributions. Moreover, its perfect recall is great for this use case.

In the case of Naive Bayes, it was the fastest algorithm in terms of time (but still the difference was very low), but the accuracy was significantly lower than one from kNN. It may be caused by the fact that two assumptions of the classifier are not met: features should be uncorrelated and their distribution should be normal. To overcome it, we tried to remove collinearity using the PCA method, but it did not improve the fit accuracy.

The Decision Tree is also worse. Maybe it is caused by the fact that it is unstable and even a small change in the dataset can cause a significant change in the tree structure.

As the last step we also provided confusion matrices:

For scaled values:

kNN	true	false	Bayes	true	false	Tree	true	false
positive	18	1	positive	15	1	positive	18	0
negative	0	38	negative	3	38	negative	5	34

For not scaled values:

kNN	true	false	Bayes	true	false	Tree	true	false
positive	16	2	positive	16	1	positive	17	4
negative	2	37	negative	2	38	negative	1	35

6. Discussion and conclusions

In the table, there is a summary of the optimal parameters and the best performance measures for each of the classifiers used for each dataset:

kNN	scaling	n_neigbours	p	weights	accuracy	precision	recall
Diabetes	yes	30	2	distance	0.727	0.630	0.537
Speed dating	yes	6	2	uniform	0.852	0.635	0.176
Purchase	no	39	1	distance	0.314	0.376	0.284
Breast cancer	yes	4	1	uniform	0.982	0.947	1

Naive Bayes	scaling	var_smoothing	accuracy	precision	recall

Diabetes	yes	0.464	0.734	0.651	0.519
Speed dating	yes	3.593	0.859	0.624	0.292
Purchase	no	0.215	0.506	0.498	0.491
Breast cancer	no	0	0.947	0.941	0.889

Decision Tree	scaling	max_depth	criterion	min_sample _split	splitter	accuracy	precision	recall
Diabetes	no	9	gini	0.071	random	0.740	0.640	0.593
Speed dating	yes	7	gini	0.03	random	0.859	0.656	0.25
Purchase	no	9	gini	0.001	random	0.117	0.095	0.099
Breast cancer	yes	9	gini	0.071	random	0.912	0.810	0.944

Effectiveness

The first thing that we observed is that it is very important to choose optimal tuning parameters for a classifier as it has a huge impact on the fit. Some parameters were consistent along with the datasets but some were completely different. Therefore, it is important to tune the parameter for each dataset individually.

In terms of effectiveness, the difference between classifiers was not that significant for Diabetes, Speed Dating and Breast Cancer datasets. For two smaller datasets (Diabetes and Breast Cancer), the performance measures obtained from the holdout method were less stable, as the fit changed a lot with differences in the chosen training data. The results for the Purchase dataset were significantly worse because the target variable could take a value of one of 100 variables, so the classification was more difficult. Moreover, in this case, we could observe that Naive Bayes outperformed other algorithms. It should be also noted, that the Decision Tree did not perform that well for the algorithms and was very unstable. Naive Bayes and kNN seemed to be a better choice for our datasets.

We also got interesting results from a comparison of scaled and not scaled data. Scaling improved the results for almost every case. It was more visible for kNN, as it uses distance measurement. However, for others, the improvement was also visible. Only for the Purchase dataset, it made results worse even for kNN, but it may be caused by the fact, that most of the features there are binary and scaling does not work well for them.

Moreover, we observed that Cross Validation is much better than the holdout method. It makes the result much more stable, which is crucial for performance evaluation, especially for the Decision Tree.

Efficiency

We have also measured the efficiency of classifiers. In most of the cases, Naïve Bayes was the most efficient classifier. The Decision Tree training took a lot of time compared to others; however, it was the fastest for testing. It means that it would be suitable for applications where the classifications need to be performed very quickly. K-NN does not need time for training, but the classification itself takes a lot of time, making the algorithm the least efficient of these 3. Those results were similar for each dataset.