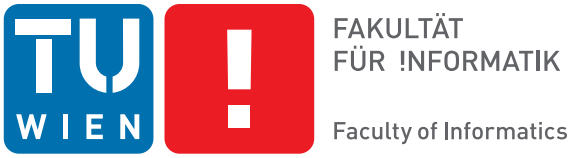

Exercise 1 - Classification

Machine Learning

Group 6

Diogo Braga (E12007525)
Enrico Coluccia (E12005483)
Matthias Kiss (01218325)

November 15, 2020

Contents

Chapter 1

Introduction

1.1 Handling the Exercise as a Group

We decided to split the workload for this exercise between the three group members in a way that everyone worked on at least one data set from start to finish and each group member had the chance to go through the whole process from a preliminary data analysis to preprocessing and finally applying different classifiers and adjusting their parameters. Code was shared on github. Therefore each member's contributions were available to all others which allowed for some consistency in handling the data sets and different classifiers.

This approach might lead to some redundancy in this report for the analysis of each data set and probably some inconsistencies in the comparison between the different data sets. However we feel like the aim of the exercise was that each group member gets the chance to play around with the different classifiers and to get a feeling for their performance on one data set as well as for the importance of each step in the process of tackling a classification problem from start to finish. This way we feel like each group member was able to have a good learning experience with this exercise.
We all decided to use python in combination with the scikit-learn library for the training of the different classifiers.

1.2 Report Structure

The report presents 4 different chapters, one for each data set. As in the provided notebooks, the process that we followed was the same for all the data sets. First we analyze and explore the data, trying to find some correlations. Then we go through the data preparation and pre-processing to get better quality data. Lastly we go through classification tasks, trying to find the best classifier and trying different solutions to improve the model which are hyper parameter optimization, holdout and k-fold cross validation.

Chapter 2

Breast Cancer

2.1 Data Set Information

There are two classes (binary classification): 'no-recurrence-events' and 'recurrence-events', that describe whether the patient's cancer reappeared after the treatment. The target is imbalanced (201 instances against 81). The other 9 attributes contain general information about the patients themselves as well as more specific information about their individual cancer diagnoses. Using this information the goal is to classify whether a patient will have breast cancer again, or not. There are a total number of 282 instances.

Description of Attributes

- Class: Describes if a patient had recurrent tumors. Nominal quantity.
- Age: Age is listed in intervals of 10 years.
- Menopause: This is another nominal quantity.
- Tumor-size: Greatest diameter in (mm) of a tumor. Interval quantity.
- Inv-nodes: Number of lymph-nodes in close proximity of the tumor. Interval quantity.
- Node-caps: Indicates whether there are metastases in the surrounding lymph nodes.
- Deg-malign: Describes how bad the cancer is. This is an ordinal quantity.
- Breast: Tumor in left or right breast, nominal quantity.
- Breast-quad: Location of tumor in breast, nominal quantity.
- Irradiate: If the patient underwent radiation therapy, nominal quantity.

2.2 Data Pre-processing

In this section we described the pre-processing applied on data. A first necessary pre-processing was to clear the whole dataset from the quotation marks and to replace the '?' with NaN to mark missing values.

We didn't use feature selection since the number of features is very low. The data set doesn't present any outliers.

Missing Data

One of the most important things at the beginning of the data approach is to check for missing data. With the help of a heat map, it was possible to visualize the features with missing values (see fig. 2.2).

```
age           0
menopause     0
tumor-size    0
inv-nodes     0
node-caps     8
deg-malig     0
breast        0
breast-quad   1
irradiat      0
Class         0
dtype: int64
```

Figure 2.1: Missing values sum

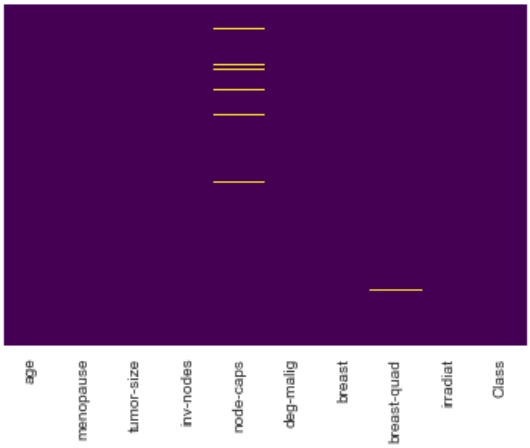

Figure 2.2: Missing values heatmap

Hence, we have a total of 8 missing values for "node-caps" and 1 missing value for "breast-quad" (see fig. 2.1).

The approach we followed to face the presence of those missing values was to use the function *fillna* on the related columns with mode of each column. We used the mode since both attributes are categorical.

Data Encoding

To perform machine learning models on our data, like those described later, we must ensure all input and output variables to be numeric. Hence, in order to fit the model we had to encode our data to numbers in several ways.

The two techniques we used are label/integer encoding and one hot encoding. In particular:

- **Label Encoding** to encode ordinal variables such as "age", "inv-nodes", "tumor-size"

- **One Hot Encoding** to encode non ordinal variables such as "breast-quad", "breast", "menopause".
- **Binary** encoding for the Target Class

Scaling

Since we're going to use algorithms that are sensitive to feature scaling (e.g. KNN and SVM with distances) we used several methods to ensure a proper scaling.

Finally, we chose StandardScaler as the best performing method in our tests. There is no significant effects of the scaling to our results.

2.3 Models Prediction

Classifiers

First we randomly applied the holdout method to divide the data in training/test set. The percentages we chose are 70% for training set and 30% for the test set.

We ran different classifiers from several classification algorithms, in order to choose the three best classifiers on our data. To perform it, standard parameters were used.

At the end of this evaluation, the three classifiers we chose are:

- Support Vector Machine
- K-Nearest Neighbors
- Decision Tree

The results are presented in table 2.1. We choose to do a *per Class* evaluation since we are interested in the performance on the *recurrence* class (health prediction)

	Class	**recall**	**precision**	F_1-score	**Accuracy**
SVM	no-recurrence	0.73	0.97	0.83	
	recurrence	0.71	0.19	0.30	**0.73**
K-Nearest Neighbors	no-recurrence	0.75	0.77	0.76	
	recurrence	0.44	0.42	0.43	**0.66**
Decision Tree	no-recurrence	0.77	0.82	0.79	
	recurrence	0.50	0.42	0.46	**0.69**

Table 2.1: Classifiers' per Class evaluation recall, precision and F-score and accuracy.

About algorithms' running time, the slowest one was the K-Nearest Neighbors with 0.01 seconds, then we have Decision Tree and SVM with 0.001 seconds. This happens because we don't have a huge number of features in our data set and therefore these algorithms (e.g. based on distance, features), that could have a large computational effort, have a reasonable run time in this case.

Hyperparameter optimization

Once we had analyzed the different algorithms with the standard settings we tried to tune the hyperparameters to find possible changes in the general performances.

Even in this phase, we used a random training/test sets split with holdout method.

To optimize those parameters the Grid-Search method was used. Basically, it builds a model for each parameter combination possible, then it iterates through every parameter combination and stores a model for each combination.

For Support Vector Machine the grid search lead to these results:

- C: 1
- gamma: 1

For K-Nearest Neighbors the grid search lead to these results:

- k: 2
- p: 2
- weights: uniform

For Decision Tree the grid search lead to these results:

- criterion: entropy
- max_depth: 50
- max_features: log2
- min_samples_leaf: 2
- splitter: random

	Accuracy	**Prev. Accuracy**
SVM	0.744	0.73
K-Nearest Neighbors	0.756	0.66
Decision Tree	0.72	0.69

Table 2.2: Classifiers' performance after the hyperparameter optimization.

Hence, we could conclude that hyperparameter optimization lead to a general improvement of the classifiers' performances. The algorithm with the best improvement was K-Nearest Neighbors with a gain of 0.96 point on accuracy.

The *Grid Search* algorithm took around 1 second to perform all the parameters permutations.

K-Fold Cross Validation

After some research it was concluded that the use of K-Fold Cross Validation can bring advantages, specifically in avoiding overfitting. Hence, we performed the CV with k=10 to our three classifiers to compare the results with the holdout method. This will give us a better indication of how well the models will perform on unseen data.

	accuracy mean	std	**holdout acc.**
SVM	0.73	0.079	0.73
K-Nearest Neighbors	0.7	0.081	0.7
Decision Tree	0.69	0.06	0.66

Table 2.3: Classifiers' CV performance comparison.

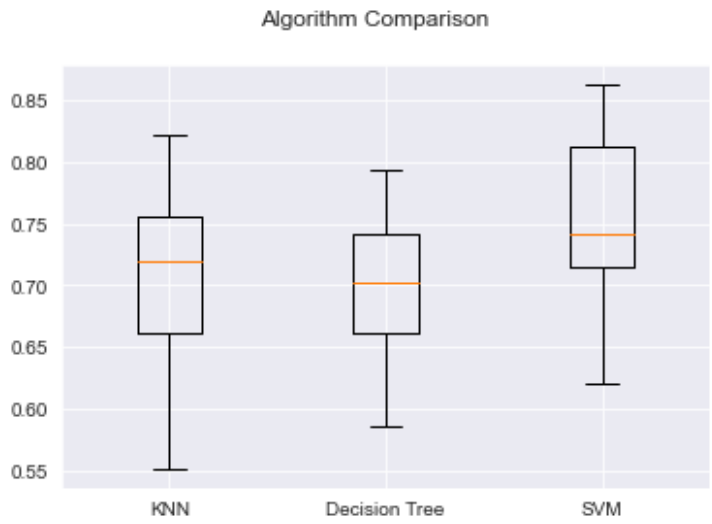

Figure 2.3: CV Algorithm Comparison

Chapter 3

Wine Quality

This data set includes two subsets, one for white wines and one for red wines. Each of the two sets includes 11 attributes that describe the chemical properties of different wines. There are 6497 different wines in total, of which 4898 are white wines and 1599 are red wines. The class attribute is each wine's quality on a scale from 1 to 10, however no wine was rated lower than 3 or higher than 8.

The goal of the classification is to assign the right quality to each wine depending on its chemical makeup. As a very first pre-processing step the two data sets were combined and the additional attribute 'type' was introduced to distinguish between red wines and white wines. This was done to compare the quality of predictions for all wines together as well as the two types of wine separately.

3.1 Dataset Information

- **type**: red wine or white wine, nominal quantity
- **fixed acidity**: grams of tataric acid per 100ml, ratio quantity
- **volatile acidity**: grams of acetic acid per liter of wine, ratio quantity
- **citric acid**: grams of citric acid per liter, ratio quantity
- **residual sugar**: grams of sugars per liter, ratio quantity
- **chlorides**: grams of sodium chloride per liter, ratio quantity
- **free sulfur dioxide**: milligrams per liter, ratio quantity
- **total sulfur dioxide**: milligrams per liter, ratio quantity
- **density**: grams per cubic centimeter, ratio quantity
- **pH**: pH value, ratio quantity
- **sulphates**: grams of potassium sulfate per liter, ratio quantity

- **alcohol**: alcohol content in vol.%, ratio quantity
- **quality**: class attribute, score between 0 and 10 (values are actually only between 3 and 8), ordinal quantity

3.2 Data Pre-processing

As already mentioned in the beginning of this chapter the first pre-processing step was to combine the red wine and white wine data sets into one data set including both types of wine. We thought it might be interesting to see, if the quality of red wines and white wines depends on similar criteria. Therefore we compare the predictions for the combined data set with the predictions when just one type of wine is considered.

This means there was a total of three resulting data sets: both wine types, only white wine, only red wine. From here on all three data sets are treated equally and pre-processing as well as all classifiers are used identically on all three sets to allow for a consistent comparison.

This data set does not have any missing values. And all attributes are ratio quantities, so no encoding was needed for them. The only exception is the newly introduced 'type' attribute for the data set containing both types of wine. The type attribute was label-encoded, such that white wine was assigned the value '0' and red wine was assigned the value '1'.

Exploratory Data Analysis

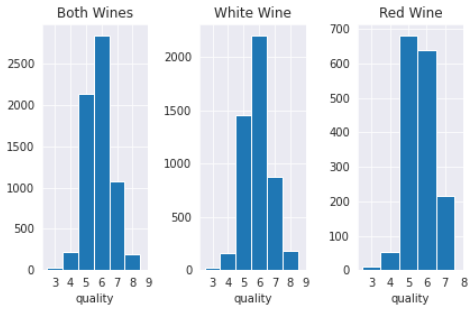

Figure 3.1: Distribution of the class values for all three data sets.

The target class in this data set has 7 values, that are wine ratings from 3 to 8. In figure 3.1 we can see the distribution of the target class values for all wines and both types separately. This indicates that it is most likely to get wines with a rating of 6 and rather unlikely to get very high or low ratings. This can lead to problems, when splitting test and training set because it can happen that one class value is not included in either set. Therefore we choose a test set of 30% of all values. This way it is more likely to have all class values in both sets.

Feature Selection

As it was already stated in the description that came with the data set, some of the chemical properties might not be independent of each other. Therefore feature selection was performed on the three data sets (both wines, red wine and white wine). For this the ExtraTreesClassifier was used. The result was that for each data set indeed some columns were removed.

One example that makes some intuitive sense without being familiar with the chemistry of wines is that for each data set 'fixed acidity' was removed and only volatile acidity was kept. It seems to make sense that those two attributes are closely related not only chemically but also to the overall perceived acidity when tasting and rating the wines.

Scaling

As we will later see the KNearest Neighbors classifier was chosen as one of the three classifiers for this data set. Therefore scaling was performed due to the importance of it to this classifier.

For the scaling we chose the standardScaler, which calculates the mean value and standard deviation for each attribute and then scales by subtracting the mean value and then dividing by the standard deviation. This leads to a distribution with mean = 1 and std = 0 for each attribute.

Outlier Detection

Outlier detection was performed using the mathematical Z-score function. However as already seen in the visual analysis of the data we performed no large outliers were present and therefore no outliers were deleted.

3.3 Models Prediction

In order to choose three classifiers to examine more closely first a list of 8 classifiers was prepared: Logistic Regression, Gaussian Naive Bayes, KNearest Neighbors, SVM rbf, SGD Classifier, Decision Tree, Random Forest and Multi-layer Perceptron Classifier.

For each of these classifiers default settings were used to find the best three for the three wine quality data sets. The selection was performed by calculating the average accuracy for each classifier for 15 simulations with different test/training set splits of the data. This way we hoped to remove the influence of each specific training set and test set pair on the results of the classifiers.

This process yielded the results shown in table 3.3 for the three data sets. According to those results Random Forest, KNearest Neighbors and Decision Tree were chosen as the best classifiers with respect to all three data sets. We chose those three classifiers despite the SVM rbf classifier being ranked second for the red wine data set to have a consistent basis for the comparison of the hyper parameter optimization presented in the next chapter.

An interesting point to mention here is that also the average run time of each classifier was calculated and is shown in table 3.3. Here the slowest algorithm was the Random Forest, that took approximately 10 times as long as the second slowest algorithm, KNearest Neighbors, and approximately 30 times slower than the fastest algorithm, Decision Tree. This makes sense, when considering, that for the random forest a lot of decison trees have to be computed and evaluated. The KNearest Neighbors, despite its simple concept also needs some computational effort to calculate the distance between each point in the data set, which scales like N^2 using the standard brute force approach but can be reduced to $Nlog(N)$ using more refined techniques. The standard decision tree is the fastest of the algorithms, which scales like $Nlog(N)$.

	average accuracy			**average run time**		
classifiers	**both**	**red**	**white**	**both**	**red**	**white**
Random Forest	0.663	0.673	0.665	0.635	0.254	0.490
KNearest Neighbors (1)	0.604	0.615	0.606	0.064	0.022	0.056
Decision Tree	0.583	0.600	0.584	0.019	0.004	0.018
SVM rbf	0.554	0.606	0.554	1.022	0.090	0.621
Multi-layer Perceptron	0.558	0.595	0.554	8.664	4.289	8.580
Logistic Regression	0.539	0.589	0.529	0.341	0.060	0.297
SGD Classifier	0.502	0.530	0.478	0.055	0.016	0.050
Gaussian Naive Bayes	0.483	0.581	0.454	0.002	0.002	0.002

Table 3.1: Comparison of accuracy and run time for different classifiers after 15 random splits into test and training set for the three data sets.

Hyperparameter Optimization

	accuracy			**prev. accuracy**		
classifiers	**both**	**red**	**white**	**both**	**red**	**white**
Random Forest	0.666	0.692	0.679	0.652	0.683	0.676
KNearest Neighbors	0.665	0.673	0.647	0.663	0.583	0.613
Decision Tree	0.579	0.640	0.583	0.573	0.594	0.582

Table 3.2: Comparison of accuracy before and after hyperparameter optimization for Random Forest, KNearest Neighbors and Decision Tree.

For the hyperparamter optimization the three best classifiers determined in the previous chapter were compared for the three data sets. For each classifier a set of parameters was compiled that seemed to be the most influential on the performance of the classifiers, while also not using too many parameters as to keep the computation time reasonably low. This test used one random test/training set split for each run. In table 3.3 the results for each classifier and each data set are shown. The best parameters for each set are the following: **Random Forest**:
Both Types: criterion=gini, max_depth=None, max_features=auto, min_samples_leaf=1

Red Wines: criterion=gini, max_depth=None, max_features=auto, min_samples_leaf=1
White Wines: criterion=entropy, max_depth=None, max_features=auto, min_samples_leaf=1
KNearest Neighbors:
Both Types: n_neighbors: 75, p=1, weights=distance
Red Wines: n_neighbors=100, p=1, weights=distance
White Wines: n_neighbors=50, p=1, weights=distance
Decision Tree:
Both Types: criterion=gini, max_depth=50, max_features=sqrt, min_samples_leaf=1, splitter=best
Red Wines: criterion=entropy, max_depth=100, max_features=None, min_samples_leaf=1, splitter=best
White Wines: criterion=gini, max_depth=50, max_features=auto, min_samples_leaf=1, splitter=best

Here it is interesting to note that for random forest and decision tree classifiers entropy was chosen once for each over the gini index.

K-Fold Cross Validation

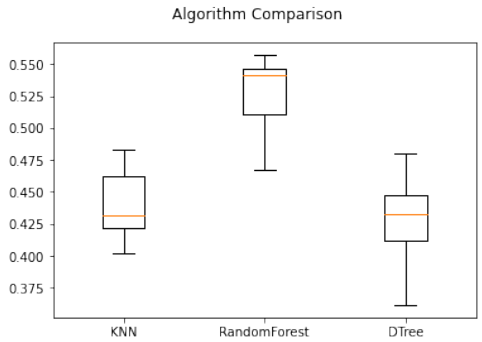

Figure 3.2: KFold Cross Validation for both wine types, comparing all three classifiers.

In order to get some statistical data about the performance of the three classifiers from the previous section we also performed the K-Fold Cross Validation for this data set with 10 splits. The results can be seen in table 3.3 as well as figure 3.2.

	accuracy mean			**accuracy std.**		
classifiers	**both**	**red**	**white**	**both**	**red**	**white**
Random Forest	0.527	0.575	0.526	0.030	0.059	0.045
KNearest Neighbors	0.440	0.495	0.455	0.025	0.06	0.034
Decision Tree	0.426	0.475	0.412	0.035	0.044	0.029

Table 3.3: Comparison of accuracy before and after hyperparameter optimization for Random Forest, KNearest Neighbors and Decision Tree.

Chapter 4

Congressional Voting

4.1 Data Set Information

The *Kaggle* competition offers 2 main data sets, one for training and one for testing. In order to know the best model to apply to the test data, a portion of the training data set is used to test the models. Therefore, within the training data set, there is a large portion to train the models, and a small portion to test which of the models performs best. After that, the best model is trained with the entire training data set, so that it is then possible to perform the predict on the test data set. The split in the training data set does not happen in this way if K-Fold Cross Validation is used.

The goal of this data set is to predict the party of a congress member. The data set contains 18 columns and 218 occurrences. It is a binary classification problem, with the most common target being "democrat" with 135 occurrences.

4.2 Data Pre-processing

In this section, it is going to be specified the pre-processing applied, in a way that the data can be more useful to predict the target class. Bearing in mind that the features are already few, the feature selection is not shown. It was tested to remove the column that has more missing values, but the performance of the models decreased. This is understandable because, with few features, all of these have great influence on the target. Since the data is binary, scaling methods are also not necessary because there would be no advantages in terms of enhancing the models.

Missing Data

First, with an observation of the data, the presence of a variable *unknown* was noticed, which represents the missing data. For this to be identified as missing data by the applied methods, the variable has to be interpretable, and for that reason we replace all these occurrences with *np.nan*. The y and n values were also changed to, respectively, 1 and 0, in order to allow the application of the necessary methods for predicting. After that, we observe the amount of missing data per feature, and we identified that the feature

export-administration-act-south-africa has a greater amount of missing data compared to the other features. In figure 4.1 it is possible to see the missing data per feature. We tested the predictor by excluding this column, but it was not favorable. The missing values were later replaced by the mode associated with each class.

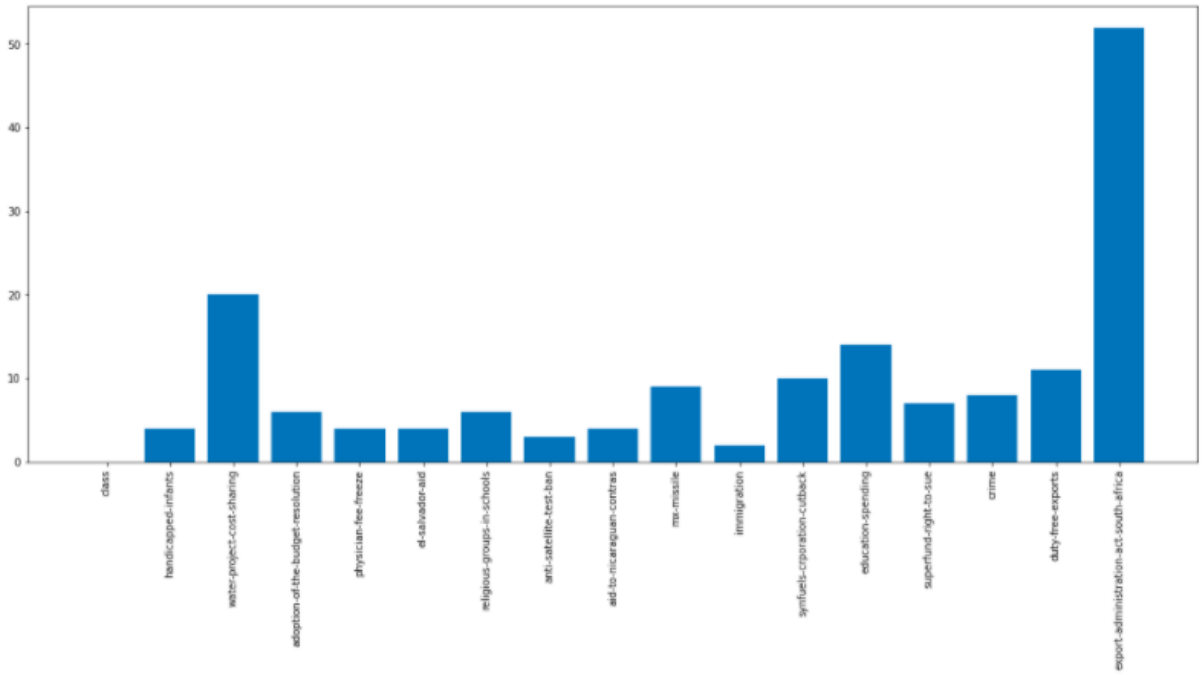

Figure 4.1: Missing data per feature.

Exploratory Data Analysis

In order to get a better understanding of the distribution of the data in this set we performed an exploratory data analysis to see if there are any apparent interesting features of the data. In figure 4.2 we show the relationship between votes of republicans and democrats on freezing physician fees. It can be seen that this topic is almost entirely split along party lines and therefore already a good indicator on which party a congress member belongs to.

Outliers Detection

We chose to use mathematical methods to find some outliers, specifically the function *Z-Score*. With threshold equal to 3, no outlier was identified and the models showed good results, which leads us to conclude that there are no outliers of great importance.

K-Fold Cross Validation

First, a simple method of train test split was used, but after some research it was concluded that the use of *K-Fold Cross Validation* can bring advantages, specifically in

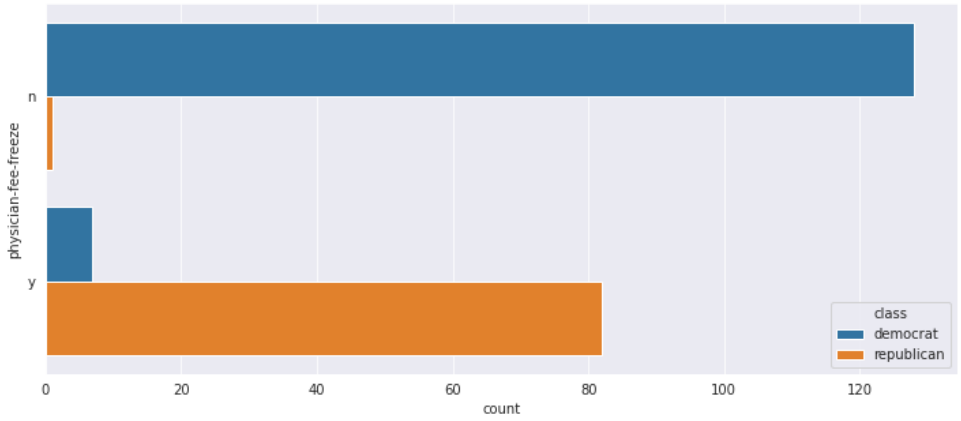

Figure 4.2: Votes per party on the topic of physician fee freeze.

avoiding overfitting. It can happen that the test train split does not occur randomly and a subset of the data has only a certain state, which would lead to overfitting in relation to the training data and a poor generalization of the model. By default we use 5 splits.

4.3 Models Prediction

After performing the pre-processing of the data in order to enhance the models, it is time to perform the fit and predict the data in the best model, the final objective of the problem. The best model was chosen based on the presented accuracy.

Classifiers

We tried 6 different types of models:

- Linear Classifiers: Logistic Regression and Naive Bayes Classifier
- Nearest Neighbor
- Support Vector Machines
- Decision Trees
- Random Forest
- Neural Networks

Table 4.1 shows the performance of the predictor in the training data set associated with each classifier, with the default hyper parameters and *5-fold cross validation* on the splitting. It can be concluded that, in general, all classifiers present good results. This is because the data has well-defined trends, making the task easier for classifiers.

Classifier	Accuracy (%)
Multi-layer Perceptron Classifier	97.26
Random Forest	96.78
Logistic Regression	96.32
SVM rbf	95.42
Decision Tree	95.39
SGD Classifier	94.97
Gaussian Naive Bayes	94.46
KNearest Neighbors	92.65

Table 4.1: Accuracy of the classifiers in the training dat aset.

Hyperparameter Optimization

At this point, the classifiers were tested with the default hyperparameters. Therefore, after choosing the best performing classifier, *Grid* and *Random Search* were applied in order to find the hyperparameters that improve the model's performance the most. The one with the best accuracy was the one chosen to make the final fit in all training data. This fit was also performed with Cross Validation.

Test Data

After choosing the model and training it in all the data, the necessary pre-processing of the training data is applied to the test data, specifically the same feature selection and scaling. The entire process is completed with the test data predictor. After that, the ids are added and a *csv* file with the results is exported.

Chapter 5

Amazon Commerce Reviews

5.1 Data Set Information

The *Kaggle* competition process is performed in the same way as explained in the previous data set. The goal of this data set is to predict who wrote the reviews. The data set contains 10001 columns without a description, which makes it difficult to understand their meaning and to make decisions at the human level to improve data quality. There are 750 occurrences, which is not a significant number for the number of columns that exist and it would be better to have more data, but this is a common situation in real life problems, since obtaining data and labels takes time and money. There are 50 different hypotheses, the most usual being "McKee" with 19 occurrences.

5.2 Data Pre-processing

In this section, the pre-processing applied is going to be specified, in a way that the data can be more useful to predict the target class.

One of the most important steps at the beginning of the data approach is checking for missing data. With the help of a heat map, it was possible to conclude that there was **no missing data**, which is not common, but it is beneficial for the credibility of the data. Other important thing is about the IDs of the occurrences, which are removed from the dataframe because they are merely identifiers, having no influence on the target's predictor. Including these in the prediction process would be a mistake. The data is all numeric, therefore, no encoding was necessary.

Exploratory Data Analysis

Since there was no information regarding the features, we tried to visualize the distribution of the target class, in order to understand, for example, if there was any label with a substantially higher occurrence that could influence the result. In the figure 5.1 it is possible to see that the data are fairly evenly distributed, which is good to conclude that there are no trends that can lead to errors in the prediction.

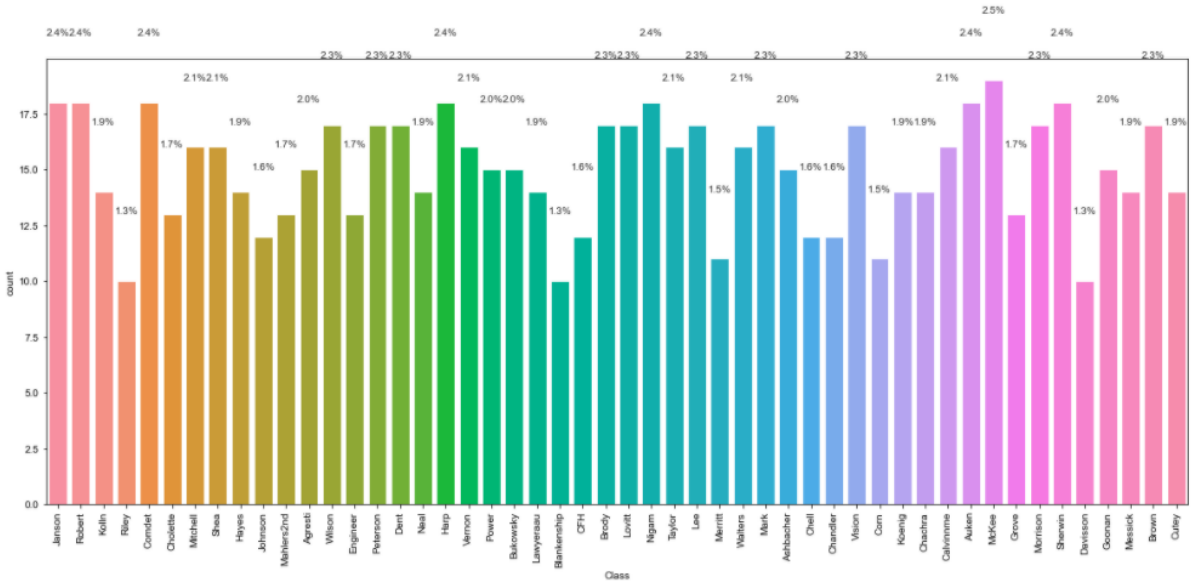

Figure 5.1: Distribution of the target class.

Feature Selection

Since we were in the presence of a large number of features, the hypothesis arose that part of them did not have a direct influence on the prediction. In fact, some of them can even be redundant so it is not interesting to keep so many columns, which would increase the computation time. The ideal is to keep the features that are really relevant to the prediction, keeping computing costs low. Due to these observations, the feature selection appears as a possible pre-processing method to be applied.

We tried 4 different methods:

- SelectPercentile (chi2)
- SelectFromModel (LinearSVC)
- SelectFromModel (LogisticRegression)
- SelectFromModel (ExtraTreesClassifier)

Of these 4, different parameters were tested, thus obtaining a different number of selected features, and the one that showed the best performance in our tests was SelectFromModel with ExtraTreesClassifier. At the end of the chapter the results obtained in the tests are presented, and it is possible to visualize the influence of these methods.

Scaling

Some machine learning algorithms are sensitive to feature scaling and can achieve better results after applying them. In algorithms like logistic regression and neural networks that use gradient descent as an optimization technique require data to be scaled, because

of the difference in ranges of features will cause different step sizes for each feature. In distance-based algorithms like KNN and SVM, distances between data points are used to determine their similarity, so it is also important to smooth the process, scaling the data before feeding it to the model. On the other hand, tree-based algorithms are insensitive to scaling.

We tried 4 different methods:

- Standardization (StandardScaler)
- Standardization (RobustScaler)
- MinMaxScaler
- Normalization

The one that showed the best performance in our tests was StandardScaler. At the end of the chapter the results obtained in the tests are presented, and it is possible to visualize the influence of these methods.

Outliers Detection

Given the large amount of data, it is difficult to perform a visual analysis of all columns in order to find outliers, so we chose to use mathematical methods in the sense that, if evident outliers exist, we can exclude some. We use the mathematical function *Z-Score*, and vary the threshold to visualize the influence on the tests.

K-Fold Cross Validation

First, a simple method of train-test-split was used, but after some research it was concluded that the use of K-Fold Cross Validation can bring advantages, specifically in avoiding overfitting. It can happen that the test train split does not occur randomly and a subset of the data has only a certain state, which would lead to overfitting in relation to the training data and a poor generalization of the model. By default we use 5 splits.

5.3 Models Prediction

After performing the pre-processing of the data in order to enhance the models, it is time to perform the fit and predict the data in the best model, the final objective of the problem. The best model was chosen based on the presented accuracy.

Classifiers

We tried 6 different types of models:

- Linear Classifiers: Logistic Regression and Naive Bayes Classifier
- Nearest Neighbor

- Support Vector Machines
- Decision Trees
- Random Forest
- Neural Networks

To analyze which is the best pre-processing, we tried some combinations and obtained the accuracy of the best classifiers. Table 5.1 shows this process. It is possible to see that the best combination is *SelectPercentile(chi2, 25)* and *StandardScaler*, therefore, this is the pre-processing chosen to continue the problem.

Feature Selection	**Scaling**	**Best Classifier**	**Accuracy**
SelectPercentile(chi2, 5)	StandardScaler	Logistic Regression	71.17%
SelectPercentile(chi2, 25)	StandardScaler	Logistic Regression	75.96%
SelectPercentile(chi2, 50)	StandardScaler	Logistic Regression	74.45%
SelectPercentile(chi2, 25)	RobustScaler	Logistic Regression	63.88%
SelectPercentile(chi2, 25)	MinMaxScaler	MLP	71.29%
SelectPercentile(chi2, 25)	Normalization	RandomForest	60.27%
SelectFromModel (LinearSVC)	StandardScaler	Logistic Regression	63.33%
SelectFromModel (LogisticRegression)	StandardScaler	Logistic Regression	63.33%
SelectFromModel (ExtraTreesClassifier)	StandardScaler	Logistic Regression	74.93%

Table 5.1: Accuracy based on combinations of pre-processing methods.

Hyperparameter Optimization

At this point, the classifiers were tested with the default hyperparameters. Therefore, after having the best performing classifier, *Grid* and *Random Search* were applied in order to find the hyperparameters that improve the model's performance. The one with the best accuracy was the one chosen to make the final fit in all training data, being this fit also performed with Cross Validation.

Test Data

After choosing the model and training it in all the data, the necessary pre-processing of the training data is applied to the test data, specifically the same feature selection and scaling. The entire process is completed with the test data predictor. After that, the ids are added and a *csv* file with the results is exported.

Chapter 6

Conclusion

Completing this project related to supervised classification problems, the group thinks that the work was carried out successfully, investigating and learning new methods, as well as obtaining good results.

It was interesting to see that, for example, for the breast-cancer and wine data set, despite their different size and one being a binary classification, while the other had 6 class values, K-Nearest Neighbors and Decision Trees were among the best performing classifiers. It was also shown that the hyperparameter optimization lead to better results in every case compared to the default values of each classifier. This shows that in any case such a optimization needs to be performed and there is no classifier that should always be used with standard parameters. This analysis has also shown that pre-processing on data is really important, above all on large data sets. Methods such as feature selection, feature scaling and outlier detection/removal are fundamental to improve an algorithm's overall performance. It is also useful to perform a good preliminary data exploration to get an overview on data distribution and/or correlation.

Regarding the structure of the *Kaggle* exercises, it was very interesting to deal with the different training and test data sets, and having to use the training data to reach conclusions to implement in the test data. This opinion is due to the fact that this type of problems appears as usual in the real world, which is good for the development of us as programmers.