Project Report

Team:
MINEheim

Data Mining 1

presented by
Anna-Lena Blinken (1818326)
Nicolas Hautschek (1816720)
Max Darmstadt (1820000)
Erik Penther (1602026)
Nicolas Fürhaupter (1819446)

HWS 2021

Contents

Chapter 1

Introduction & Procedure

1.1 Application Area and Goals

This paper represents the project report of Team MINEheim (Group 1) and documents the problem, purpose, approach, and the results of the programming project, which was completed in parallel. As mentioned in the proposal document, this project is related to the wine industry, especially the quality of wine. The main problem that needs to be solved is the fact that the quality of wine cannot yet be determined objectively. In addition, every existing approach to rate a wine's quality includes its actual consumption, which is not acceptable for rare or highly-expensive products. Wine sommeliers try to solve this problem by communicating their rating of the specified wine to the end consumer, but their taste is rather subjective and therefore not reliable. In fact, this constitutes a weak spot of the wine industry since it heavily relies on the subjective taste of a few sommeliers. As a result, there is no single source of truth about the ingredients, components or features of wine that influence its quality. In short, the main question is: What makes a good wine actually good?

Consequently, the goal of this project was to objectively predict and classify wine based on its ingredients, components, and features to enable an objective wine rating. The target variable was the quality of wine, which originally consisted of values on a scale from zero to ten. Later, the target variable was categorized into three distinct classes (class 0: low, class 1: medium, class 2: high). Due to the categorization of the target variable into classes, this project deals with a classification approach of machine learning (ML). In theory, classification is one of the most common tasks of supervised ML. It describes the classification of unknown instances of input data in one of the pre-offered categories, the classes, based on a previously trained model (Novakovic, 2017).

The solution, which was developed during this project, aims to leverage various benefits for businesses and consumers. In detail, those benefits were already addressed in the proposal document as they range from cost and time savings to an objective single source of truth and many others.

1.2 Structure and Size of the Dataset

The dataset used in this project is related to red and white variants of the Portuguese wine *"Vinho Verde"*, a unique wine product from the northwest region of Portugal. The data was originally collected by an inter-professional organization called CVRVV between 2004 and 2007 for marketing and quality assurance purposes (Cortez et al., 2009). The whole set includes 6,497 entries.

Taking a closer look at the data, 13 unique features can be identified, which describe each wine sample. Most of the features represent test results of physiochemical properties of each specific wine i.e., *fixed acidity, volatile acidity, citric acidity, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH-value, sulphates,* and *alcohol*. These attributes are given in their respective measurement unit and represented as positive numeric values. The remaining features comprise the *quality* and *type* of the wine. The quality of a wine is represented by a score metric ranging from zero to ten, where zero is a wine of low quality and ten a wine of excellent quality. Each wine was evaluated with a sensory test (blind test) by at least three assessors and then graded. The final grade was calculated with the median of these tests. The feature *type* indicates whether the wine is a red or a white wine.

When searching for null or missing values, only 34 of all samples had missing values. Concerning the value distributions of the features, it can be noticed that some of the features e.g., *alcohol*, have a slightly right-skewed distribution. Although some outliers can be found, after some investigation it turned out to be no issue for the use of the data in this project. Looking at the *type* feature it becomes apparent that there are about twice as many white wines than red wines. What can also be noticed and is a really important insight for the preprocessing is that the *quality* feature is highly imbalanced. The quality values five and six together make up about 76%, or 5,000 samples, of the whole dataset. This means that there are a lot of wines with medium quality and few of low and high quality. This problem has to be addressed in the preprocessing step.

1.3 Preprocessing

Before the models for predicting the quality of a wine were created, the data had to be preprocessed accordingly. Preprocessing is important in order to "resolve several types of problems include noisy data, redundancy data, missing data values, etc." (Kotsiantis et al., 2006, p. 116). The preprocessing of the utilized dataset for the characteristics as well as the quality of wine can be divided into three parts.

First of all, duplicates needed to be deleted and missing values must be handled in an appropriate way. The dataset of 6,497 entries did not include any duplicates. Consequently, no rows were deleted. As analyzed in section 1.2, the dataset had 34 missing values or so-called null values in seven different columns, which were imputed with the mean value.

In the second step the columns were preprocessed. Numerical values were normalized, and categorical values were converted into numerical values. According to Kotsiantis et al. normalization is a "'scaling down' transformation of the features." (Kotsiantis et al., 2006, p. 113). Therefore, all numerical features as for instance pH-values, and alcohol were normalized. The categorical value type, which determines whether the wine is red or white, was encoded to numerical values with the help of the *Ordinal Encoder* (ScikitLearn, 2021b). As the target variable quality was imbalanced, a binning of three classes, low, medium and high quality, was applied. No additional features were generated, as the dataset mainly contains chemical parameters as e.g., pH-value or sulphates, which cannot be combined to build new features. The feature selection and correlation are part of the adaption phase of the models as the feature selection differs between the applied algorithms.

In the last step, the feature and target variable were separated and stored in two different dataframes. Afterwards, the test and train split was applied with a test size of 20 % of the available data and a stratification of the target variable.

Chapter 2

Modelling

After preprocessing, this part focuses on the various ML approaches that were used. In order to ensure the comparability of the ML approaches, the *K-Nearest Neighbors (K-NN)* method was used as a baseline. As discussed in section 2.1, the K-NN approach is a rather simple, but potentially effective classification method and for this reason suitable as a baseline. Other ML approaches that were investigated are *Support Vector Machines (SVM)*, *Descision Trees*, their extension the Random Forest, and *Neural Networks*. This set up of various ML approaches was choosen for their coverage of different prediction strategies and in order to both cover symbolic and non-symbolic approaches.

The success of the different ML approaches was monitored through the Precision, Recall and F_1-Makro metric for both train and test data. All three measures are suitable for unbalanced datasets (Brownlee, 2020). Especially, the F_1-Score represents a harmonic mean between the Precision and Recall score and was therefore also used to calculate the hyperparameter tuning (Peltarion, 2021).

The procedure to evaluate the different ML approaches began with the training of the various models without any additional settings and the default values for each model. This first model, which will be referenced as 'none' in the result tables 2.1, serves as a baseline. For a quick first evaluation of the dataset, a hold-out validation was used. Afterwards, a hyperparameter tuning was performed which used the ten-fold Cross-Validation for validating the model. Based on this result, the default values for the various models were changed. To further optimize potentially low performing models, the dataset was balanced through oversampling. Depending on the corresponding model, a feature selection was conducted. Finally, a second hyperparameter tuning was carried out to optimize the altered ML model. Changes resulting from the various ML approaches were disclosed in the corresponding sections.

2.1 K-Nearest Neighbors (K-NN)

	Precision		Recall		F_1-Makro	
Adaption	Train	Test	Train	Test	Train	Test
none	0.825	0.680	0.586	0.522	0.627	0.556
+ 1. Hp Tuning	0.825	0.680	0.586	0.522	0.627	0.556
+ Balancing	1.000	0.573	1.000	0.659	1.000	0.599
+ Feature Selection	1.000	0.558	1.000	0.642	1.000	0.585
+ 2. Hp Tuning	1.000	0.558	1.000	0.642	1.000	0.585

Table 2.1: KNN - Predictions

K-Nearest Neighbors (K-NN) is a rather simple algorithm which is based on the distances between different data points to classify data (Yildirim, 2020). The initial training of the model, using the default values of the K-NN algorithm, resulted in a rather unprecise measurement. The Precision and Recall values in table 2.1 show that only around half of the data points were classified correctly. Nonetheless, these results were a usable baseline. Afterwards, a first hyperparameter tuning was executed, which resulted in almost the same exact default values. Meaning that the default values were already the most optimized combination of hyperparameters. Parameters that were tested through the hyperparameter tuning were mainly various settings for different calculation models for estimating the distance between data points and the number of relevant neighbors (*n_neighbors*) included in the prediction. As a next step to improve the prediction of the model, the training dataset was balanced through oversampling. After oversampling, the result of the model increased to a 100 % correct score on the training data, whilst the prediction for the test data was still rather poor. This indicates an overfitted model. Looking at the test data predictions in general, the Recall and F_1-Score could only be slightly increased and the Precision test data got worse (table 2.1). In order to reduce the overfitting of the model, a feature selection was used to make the data less noisy. This was done by calcualting the *Permutation Importances* of the datasets features and by calculating the *Feature Correlation*. The model was tested based on both feature selection methods and finally through a combination of both. This combination, which got the best output, still resulted in an overfitted model. Furthermore, Precision test, Recall test and the F_1-Score test score performed slightly worse. Finally, a second hyperparameter tuning concluded that the default hyperparameters for the model were the best fit. Therefore, the final scores were not changed through the second hyperparameter tuning and the model was still overfitted.

2.2 Support Vector Machine (SVM)

	Precision		Recall		F_1-Makro	
Adaption	Train	Test	Train	Test	Train	Test
none	0.497	0.510	0.393	0.387	0.396	0.387
+ 1. Hp Tuning	0.881	0.563	0.638	0.493	0.712	0.516
+ Balancing	0.893	0.529	0.890	0.671	0.887	0.557
+ Feature Selection	0.861	0.515	0.859	0.655	0.855	0.537
+ 2. Hp Tuning	0.952	0.537	0.951	0.618	0.950	0.562

Table 2.2: SVM - Predictions

Support vector machines (SVM) are computer algorithms which belong to the supervised learning methods. They learn by example to assign labels to objects and therefore can be especially useful for classification problems. To achieve this purpose, they comprise four basic concepts: the separating hyperplane, the maximum-margin hyperplane, the soft margin, and the kernel fuction. In short, SVMs draw an optimal hyperplane inbetween all of the classes that are distinguished and aim to maximize the distance between the data points and the hyperplane (maximum margin) (Noble, 2006). SVMs were chosen for this project because they are particulary efficient for high-dimensonal spaces and classification problems in machine learning (ScikitLearn, 2021c). For the SVMs, the general procedure mentioned in chapter 2 was followed. After the preprocessing of the data, the initial model was created, trained and evaluated by using a SVM. The performance of the initial model was fairly low with the F_1-Score barely reaching 40 % for both train and test data. As a next step, the first hyperparameter tuning of the attributes 'C', 'gamma', and 'kernel' resulted in an improved model with an F_1-Score of 71.2 % for the train data and 51.6 % for the test data. However, it became clear that this state of the model included a high bias towards class one (wine of medium quality) due to the unbalanced train and test dataset. The following model was based on a balanced dataset by using SMOTE to oversample the minority classes. The F_1-Score increased by roughly 4 %. Furthermore, feature selecting techniques proved as non-viable as the F_1-Score decreased and were not further inspected. Lastly, the second hyperparameter tuning resulted in the same optimal parameters and F_1-Score as before in the oversampling step. Additional experimentation with the hyperparameters led to the final model with an F_1-Score of 95 % for the train data and 56.2 % for the test data. In theory, it is possible to further increase the F_1-Score for the test data to values above 60 % by increasing the hyperparameters 'C' and 'gamma'. However, this would lead to an even higher overfit-tendency of the model. Therefore, this step was avoided. All results regarding the model evaluation can be found in table 2.2.

2.3 Decision Tree

	Precision		Recall		F_1F1-Makro	
Adaption	Train	Test	Train	Test	Train	Test
none	1.000	0.569	1.000	0.582	1.000	0.575
+ 1. Hp Tuning	0.822	0.484	0.714	0.46	0.757	0.468
+ Balancing	0.843	0.486	0.843	0.610	0.841	0.500
+ Feature Selection	0.847	0.492	0.847	0.611	0.845	0.501
+ 2. Hp Tuning	0.869	0.485	0.866	0.600	0.862	0.485

Table 2.3: Decision Tree - Predictions

The decision tree is an algorithm with a tree-like structure which can be used for classification or regression tasks. The main idea is to build a tree out of nodes in which all attributes are tested and the best split is chosen. In the end, the child nodes should represent the corresponding class of a given sample.

To start with, the first decision tree was created with the default parameters given by scikit-learn and was fitted to the train data (ScikitLearn, 2021a). What can be immediately noticed is that all Precision, Recall, and the F_1-Macro score got a value of 1, whereas the model performed significantly worse on the test data. The resulting model predicted a medium quality most of the time. This overfitting with the test data was to be expected, since decision trees tend to overfit more easily. To address this, hyperparameter tuning was applied. To find better hyperparameters, a ten-fold cross-validation grid search was applied, where the parameters *criterion, max_depth, min_samples_split,* and *min_samples_leaf* were iterated on. This led to a less overfitted model, although the test results did not really improve. To overcome the problem of only predicting medium wines, SMOTE oversampling was applied, as done earlier in the SVM. This improved the F_1-Score only marginally. Another improvement measure which was applied is to remove features from the data which do not seem to have a major impact on the model. By applying the in scikit-learn built-in feature importance methods, the feature *type* was completely removed, since it had the lowest importance compared to other features. Nevertheless, this again did not seem to improve the model by a lot. In a last attempt, a second hyperparameter tuning with the same parameter grid was applied, resulting again in no further improvement on the test F_1-Score. In conclusion, it can be observed that a decision tree will not give the best classification results and should not be considered as a prediction model in this use case. Therefore, to improve the quality of the prediction, another classification algorithm such as the random forest algorithm should be evaluated.

2.4 Random Forest

	Precision		Recall		F_1-Makro	
Adaption	Train	Test	Train	Test	Train	Test
none	1.000	0.878	1.000	0.552	1.000	0.596
+ 1. Hp Tuning	0.962	0.525	0.622	0.476	0.666	0.489
+ Balancing	0.966	0.579	0.966	0.675	0.965	0.611
+ Feature Selection	0.967	0.558	0.967	0.644	0.966	0.585
+ 2. Hp Tuning	0.966	0.577	0.965	0.667	0.965	0.607

Table 2.4: Random Forest - Predictions

The random forest algorithm, developed by the statistician Leon Breimann, is in general a combination of different decision trees (Breiman, 2001, p. 6). The first model based on the random forest algorithm was created with the default parameters of the random forest. This creation resulted in an overfitted model, which had a precision, recall and F_1-Score of 100 % within the training and a precision, recall and F_1-Score of between 55.2 % and 87.8 % in the testing. Therefore, the trained model did not make any mistakes. However, the tested model only predicted the wine of a medium quality. This might be the result of an unbalanced dataset where most entries are classified in the medium class and only a few wines are classified in the lower or higher class. The parameters of the initially created model were hypertuned (e.g, maximal depth of 13 of the tree) and were used for the other models (Hoffman, 2020). It can be depicted that the model with the hypertuned parameters did not reveal any major overfitting issues anymore as seen in the table 2.4. However, the tested model did not classify any wine as class 0 (“bad”).

Consequently, the second model included an oversampling method in order to balance the dataset. This modification resulted in an improved but overfitted model. The precision, recall and F_1-Score of the training differed between 96.5 % and 96.6 % whereas the precision, recall and F_1-Score of the testing stretches over the interval of 57.9 % and 67.5 %. Besides the remaining overfitting issue, the trained and tested model predicted every class.

In order to tackle the remaining overfitting issue, the feature importance, permutation importance and feature correlation were analyzed (Płoński, 2020). It can be depicted that dropping columns depending on the results of the *feature importance*, *permutation importance* or *feature selection* and the combination of the three, did not reveal any enhanced model. Therefore, no feature selection is applied in the next step. In table 2.4, the fourth row displays the best option of the three used techniques which is the feature importance.

Lastly, a second hyperparameter tuning was conducted to adapt the hyperparameters to the newly created model including the oversampling. This second hyperparameter tuning resulted in a slightly improved model.

2.5 Neural Networks

Preproc.	Batch size	Layers	Neurons	Epochs	F_1 Val	F_1 Test
Standard	64	3	100	30	0.406	0.434
+ Balancing	64	3	100	30	0.707	0.456
+ Balancing	64	4	300	30	0.738	0.487
+ Balancing	64	5	400	30	0.761	0.544
+ Balancing	16	7	1,000	40	0.820	0.553
+ Balancing	16	8	2,048	80	0.851	0.567

Table 2.5: Neural network – Predictions

Neural networks (NNs) are supposed to emulate the learning process of the human brain by connecting artificial neurons and giving these connections a certain weight. Adjusting these weights is the "learning" part of the algorithm (Aggarwal, 2015, p. 326).

After the preprocessing steps that were explained in chapter 1.3, the train dataset was split again to create a validation dataset, which was used during the *training* or *fitting* phase of the NN. It was used to – as the name suggests – validate the interim results of the network and adjust the hyperparameters (Tan, 2006, p. 184).

Having split the data again, it had to be converted into tensors in order to be able to be used in the NN. This had to be done for all of datasets i.e., training, validation, and test. These tensor datasets were used to create so-called data loaders which turned the data into batches which were then passed through the NN. Here a couple of different batch sizes (64, 32 and 16) were tried out in order to improve the network.

For the network architecture itself, pytorch lightning was used. Different network architectures with an increasing number of layers and neurons were tried in order to improve the performance of the model. Six different architectures were examined further, starting from three layers and 100 neurons and ending up in seven layers with 1,024 artificial neurons. The linear or fully-connected layer of PyTorch was used for the models (PyTorch, 2019, 2021). Furthermore, different activation functions (SiLU, ELU and GELU) were applied to the model but the standard model architecture with rectified linear units (ReLUs) could not be improved with other activation functions.

An overwiew of the different network architectures that were tested and their respective performances is given in table 2.5. The "neuron" column refers to the number of neurons in the first fully-connected layer (the number of neurons decreases with each layer). In the table, the validation performance refers to the F_1-Validation-Score in the last epoch. It can be seen that the F_1-Test-Score improved when the batch size was decreased while the number of layers, neurons, and epochs increased.

Chapter 3

Results

To conclude, the business use case was to predict the quality of wine with the help of its contents. In order to do that, a dataset consisting of 6,497 samples of wines with their respective physiochemical features and their quality was used. The data was unbalanced in respect to the distribution over the classes. To combat this problem and to prepare the data to be used in the algorithms, a set of preprocessing steps e.g., normalization and balancing were performed. After having preprocessed the data, different algorithms were tested. Five algorithms were used to predict wine quality.

KNN as our baseline resulted in moderate results in regard to its simplicity. The model was overfitted in the end but nevertheless produced test scores of 59.9 %. The decision tree had a mediocre performance. It still was overfitted when trying oversampling and feature selection but produced a test score of 48.5 %, although the best test score was reached without any preprocessing and the default values. Random forest performed quite well as an aggregation of different decision trees. Like all models, the random forest had a tendency to overfitting but in the end it reached the overall best test score with all preprocessing steps of 60.7 %. The support vector machine also performed reasonable as a classification algorithm with the problem of being slightly overfitted reaching a test score of 56.2 %.

Although the predictions of the neural network were decent with 56.7 % at its highest, the efforts to create the network with different architectures did not pay off. In this regard, more resources would have been needed to improve the test score. To conclude, all of the models had a tendency to overfitting. Of the five models, the random forest algorithm did the best job, while also being the only model which performed better than the baseline algorithm. Nevertheless, all models could not predict the quality of the wines with a high certainty on the basis of its contents. This is probably due to the fact that also other features besides chemical contents determine how someone perceives the quality of a particular wine.

Bibliography

Charu C Aggarwal. *Data Mining: The Textbook*. Springer International Publishing Switzerland, Cham, 2015.

Leo Breiman. Random forests. *Machine Learning*, 2001. Accessed: 2021-11-20.

Jason Brownlee. How to calculate precision, recall, and f-measure for imbalanced classification. `https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/`, 2020. Accessed: 2021-11-19.

Paulo Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Wine Quality, 2009.

Ken Hoffman. Random forest hyperparameters explained. `https://ken-hoffman.medium.com/random-forest-hyperparameters-explained-8081a93ce23d`, 2020. Accessed: 2021-11-21.

Sotiris B. Kotsiantis, Dimitris Kanellopoulos, and Panagiotis E. Pintelas. Data preprocessing for supervised leaning. *International journal of computer science*, 2006.

William S Noble. What is a support vector machine? *Nature Biotechnology*, 2006.

Jasmina et. al. Novakovic. Evaluation of classification models in machine learning. *Theory and Applications of Mathematics and Computer Science*, 2017.

Peltarion. Macro f1-score. `https://peltarion.com/knowledge-center/documentation/evaluation-view/classification-loss-metrics/macro-f1-score`, 2021. Accessed: 2021-11-19.

Piotr Płoński. Random forest feature importance computed in 3 ways with python. https://mljar.com/blog/feature-importance-in-random-forest/, 2020. Accessed: 2021-11-21.

PyTorch. Linear. https://pytorch.org/docs/stable/generated/torch.nn.Linear.html, 2019. Accessed: 2021-11-23.

PyTorch. Defining a neural network in pytorch. https://pytorch.org/tutorials/recipes/recipes/defining_a_neural_network.html, 2021. Accessed: 2021-11-23.

ScikitLearn. sklearn.tree.decisiontreeclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html, 2021a. Accessed: 2021-11-20.

ScikitLearn. sklearn.preprocessing.ordinalencoder. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html, 2021b. Accessed: 2021-11-20.

ScikitLearn. Support vector machines. https://scikit-learn.org/stable/modules/svm.html, 2021c. Accessed: 2021-11-23.

Pang-Ning Tan. *Introduction to data mining*. Pearson international Edition. Pearson, Boston Munich, 2006.

Soner Yildirim. K-nearest neighbors (knn) — explained. https://towardsdatascience.com/k-nearest-neighbors-knn-explained-cbc31849a7e3, 2020. Accessed: 2021-11-20.