Group Project Paper - Group 2

This paper serves as a guide to understanding the jupyter notebook created by Group 2. There are several chapters explaining the main steps that were undertaken in the notebook.

1 Start

The submitted jupyter notebook begins by showing the relevant packages we imported and used throughout the paper. Next, the data set was imported, and we familiarised ourselves with it by making use of several data visualisation techniques like histograms, a correlation matrix, and a waffle chart of the class distribution. We then proceeded to handle all the missing values in our data set so we could find solutions for them.

2 Data Cleaning and Preparation

2.1 Dealing with NaN's

In the original data set, there were 19 columns containing NaN's which needed to be cleaned up so that machine learning tools could be applied later. We decided to drop all rows containing missing values in the columns "MasVnrType" and "Electrical" as this affects only a small portion of 9 out of 1460 rows and we could not assume any reasonable values as a replacement for the missing values.
For the category "LotFrontage" however we assumed that all the NaN's are equal to a value of 0, as this category refers to the linear feet of street connected to the property. We understand all NaN's in this category to mean that there is no connection of the house to the street, resulting in a length of 0.
From the correlation matrix, we learned that the column "GarageYrBlt" correlates very strongly with the column "YearBuilt". As "GarageYrBlt" contains 5.5% NaN's (which stands for no garage available). We decided to drop this column because it wasn't providing any major benefit to our data set due to the mentioned correlation. Also converting the categorical data in the column "GarageYrBlt" with one-hot encoding would have increased the size of our data set drastically, which was another reason for dropping the column.
Because of the high correlation with "GarageArea," we dropped "GarageCars" as well. A strong correlation occurs because these two features provide almost the same information. GarageArea describes the size of the garage and GarageCars describes the number of cars that fit in the garage.

2.2 Cleaning

Since the column "Central Air" contains binary data in the form of strings "Y/N", we factorized the data to 0's for all the N's and 1's for the Y's. The "ID" column also got dropped as this column does not provide any further information to our data set.
Three columns in the data set contain categorical data, but the values are numerical. For now, we only converted the contents of the column "MSSUbClass" into strings. The other two columns "OverallQual" and "OverallCond" will be dealt with later.
We transformed categorical data columns into strings to be able to create dummy variables out of the categorical data with more than two categories. We are going to use one-hot encoding later to finish this pre-processing. For now, there are still columns containing missing values which will be addressed separately in the different data sets.

3 Data Sets

Using one-hot encoding drastically increases the number of columns in the data set. That's why we approached the task of finding the best algorithm by using and comparing data sets of two different sizes. Using fewer, but only the most important, variables promises to reduce overfitting and might even increase the accuracy. We expected that the smaller data set will lead to shorter calculation times, whilst not reducing the quality of our predictions much. This however was not always the case, which will be addressed in the discussions chapter. Using a reduced data set may also prevent overfitting or at least work against it.
The main differentiation is that the smaller data set treats certain strings, and categorical columns differently than the larger data set. When running the algorithms, we further employed a pipeline for the small data set to individually select a smaller set of variables with a feature selection, which together explains the median of the outcome. This ensures that each algorithm will be calculated with a fitting smaller and larger set of variables.

3.1 Feature engineering

Quickly however on the topic of feature engineering: We considered this approach and looked into finding the median incomes in different zoning regions in the US and into finding the median income in different neighborhoods in Ames, Iowa. However, no usable data came up, which is why we didn't pursue it further.

3.2 The Smaller Data Set

We started the construction of the smaller data set by factorizing columns with categorical quality measures that followed the structure of using qualitative expressions like "poor", "fair", "average" and so on into numerical values (NaN=0, poor = 1, fair = 2, etc.). This step was taken to reduce the size of our dataset, but it came with the cost of assuming the linearity of those quality expressions. For instance, the quality measure "poor" (1) is probably not four times worse than the measurement "good" (4). This handling of the data, therefore, poses a limitation for the smaller data set. We believe however that for the sake of reducing the number of columns, is a justified measure as this process keeps all the information in one column. We applied this transformation to 13 different columns, the names of which are found in the jupyter notebook.
Afterward, we possessed the remaining missing values and proceeded to turn the remaining categorical values of the other columns into dummy variables by using one-hot encoding. The complete process can be found in the notebook. After this step, we were left with 259 columns and NaNs.
Compared to the large data set (324 columns), this set of variables is already considerably smaller. The main reduction of variables however will only be employed during the feature selection within the pipeline which we use for each algorithm (see below).

Before the feature selection demonstrations, we first split the small data set into a training and test set. Important to note is, that we stratified this splitting, meaning that the distribution of the classes in the new training and test set are following the distribution in the original data set.

3.2.1 Feature Selection

As already mentioned, we do the Feature Selection within the pipeline. This approach avoids data leakage as we only use the train data used in a specific fold of the cross validation to choose the most important variables.

Two examples of the feature selection process can be found in the jupyter notebook. These serve to visualise the process inside of the pipeline and to show the most likely candidates for the most important features in our data set. We then plotted the result to visualise our findings and to get a better understanding of the process that is going to occur later inside the pipelines.
Further explanations of the two examples can be found in the jupyter notebook.

In both, the examples, and the pipeline we use a Random Forest algorithm to select the most important features. However, there might be some differences between the selected variables due to the usage of cross validation. This is because within cross validation the feature selection will be done for each fold separately which could result in slightly different features selected than in our previous examples. However, doing the feature selection outside of the pipeline would defeat the purpose of the cross validation since there would be data leakage between the different folds.
In our second example, we applied the feature selection within a cross validation. There you can see how sensitive this selection might be. The result suggests that the features will not differ much.

3.3 Construction of the Large Data Set

The construction of the large data set was considerably simpler. Instead of factorizing categorical data into numerical data, we turned the two remaining categorical columns with numerical values, “OverallQual” and “OverallCond”, into strings. As explained above this allows non-linearity for different levels of quality e.g., 1 to 2 might have a bigger effect on the house price than from 4 to 5.
This allowed a simple use of the one-hot encoding process which resulted in a data frame with 324 columns and 0 NaNs.
We then split the data into a train and test set to work with it in the next chapter, whilst also making sure the splitting is stratified.

3.4 Oversampling

In the first chapter, we mentioned the use of a waffle chart to visualise class imbalance. We tackled this issue inside the pipeline to avoid data leakage. This is necessary since we apply cross validation. If we were to oversample our data in the beginning, there would be a data leakage and we would have an overfitting and a biased result.
In an earlier version of our notebook, we also implemented downsampling. As the data set is not that big and our results with downsampling were worse, we focused on oversampling. To oversample our data, we used SMOTE. SMOTE has the advantage that it does not simply copy-paste the data of the same class until there is the same number of rows for all classes as is done in upsampling. Instead, SMOTE creates synthetic data points from the linear combinations of other data points from the same class (Blagus & Lusa, 2013). The advantage of this is that you are left with a higher variety in your data which often results in a better ability to predict the classes of new data. This is especially useful when there are small amounts of data points available for certain classes (in our case for class 4), which is even aggravated by cross validation.

4 Algorithms

4.1 Construction of the algorithms

For running our algorithms, we decided to define a function to streamline this process.
The first element of this function is called “Classifier”. This variable incorporates the chosen classifier algorithm (e.g., Logistic Regression, Random Forest, etc.).

Next, we have “params” standing for the parameters chosen for the function to iterate through during parameter tuning in a grid search. These two variables, Classifier, and params need to be defined for each classifier algorithm individually.
The next elements “X_train”, “X_test”, “y_train” and “y_test” all relate to which training and test sets were chosen. Here, it can be decided whether to use the small or the large dataset when running the function.
"Apply_feature_selection” refers to the feature selection mentioned in Chapter 3.2. For this feature selection, we use a random forest classifier and aim for a selection of variables that explain the median of the outcome. Apply_feature_selection can be turned off, as the following two variables as well.
The 8th element of the function is called “apply_standardising” which uses a standard scaler to standardise all the data. This is necessary for some algorithms and our oversampling function.
The last element “apply_sampling” refers to the issue of oversampling which is explained in Chapter 3.4. As already mentioned, we use SMOTE to do our oversampling. We use the default distribution which means that all classes will be oversampled to the number of observations in the most represented class. Additionally, we implemented a random state to make the results replicable.
Regarding cross validation: We used the function “StratifiedKFold” to draw the folds, which has the advantage that it makes sure that the classes follow the original distribution.
The order for these steps within our newly defined function was not chosen at random. Cross validation must happen first to avoid data leakage when applying feature selection. Standardising could only ever come after the reduction of the data set (for the smaller data set) but had to come before the usage of SMOTE for oversampling as this function uses KNN, which requires stratified input data to function.
After doing those pre-processing steps within the pipeline we can finally apply our algorithm. We defined the classifier function within the pipeline and then used grid search and cross validation to perform parameter tuning on this pipeline with the defined variable “params”. Then the model was trained, the best one chosen and used to predict the classes of the test data. In the next chapter, we will discuss the output and the chosen performance metrics.

4.2 Performance metrics

When using classification models in machine learning, there are three most common metrics to assess the quality of the model: Precision, Recall, and F_1-Score (Zach, 2022). All those metrics are in the classification report of sklearn. The metric *accuracy* gives us the ratio of correct predictions out of all the predictions made (Zimmermann, 2023). This is a simple way to get a good idea of how well a given algorithm performed. *Precision* on the other hand refers to the accuracy of a positive prediction and is the ratio of true positive prediction relative to total positive predictions (Zimmermann, 2023; Zach, 2022). *The recall* is the ratio of true negatives out of the true negatives and false negatives. The *F_1-score* is an evenly weighted version of the *F*-score, which tells us more about the relationship between the data’s positive labels and those given by the classifier, which is useful when comparing different algorithms that perform similarly (Zimmermann, 2023). The latter three metrics are computed for each class. This allows us to see how the predictions performed for each class. The *macro average* and the *weighted averag*e further display the equally weighed average and the average weighted by class distribution respectively of these columns. All in all, these four metrics together give us a deep insight into the performance of our algorithms.
Our outputs also give some information on the true distribution of classes in our test set within the column *support* by providing the number of rows belonging to each of the different classes.
We also printed the measurement *best training accuracy*, which shows how well our algorithm performed on the training data. This allows us to assess potential overfitting.

To compare different algorithms fairly we implemented code that displays the *calculation time* used by each algorithm. We decided to use this measurement, as in practice the calculation time and therefore the "cost" would be one of the most important measures.
Throughout this whole paper, we are also using random states to make the algorithms replicable and therefore comparable. This is especially important when splitting the data into training and test sets, or when we make use of algorithms like Random Forest Classifier, as randomness is an integral part of their functioning.

4.3 Comparing the Algorithms and Deciding on the Best

At the end of our notebook, you can find a valuable visualisation that summarises all the algorithms' performance and allows for a quick comparison of the different classifiers. In the subplots, you can see the confusion matrices, the accuracy scores, the F_1-score, and the time it took to run the algorithm. Due to spacing issues, we do not implement the graphics here.
The best algorithm is SVM on the small data set. It has an accuracy of 85%. The class balance was handled very well. The macro and weighted average of the precision is almost the same.
The best algorithm for the large data set is the XGBoost. For most algorithms with a large data set, we had an overfitting problem but with an accuracy score of 85% we still managed to get a good result.

5 Discussion

As mentioned earlier we expected that using a smaller dataset would result in shorter run times when compared to the larger dataset. However, this was not always the case. The reason for this is the fact, that we could only employ the feature selection in the pipeline. This led to higher run times, as we had to repeat this process for every classifier and every fold within the cross validation individually. Furthermore, the feature selection is based on a Random Forest classifier, which is slower than other classifiers. We believe however that sacrificing the superior speed of the small data set in some instances is worth it, as it ensures that no data leakage could occur. Also due to the implementation of many different algorithms and grid search, we are quite confident in the performance of our best algorithms. Naturally, there is still room for improvement, but we believe that our chosen classifiers are quite reliable.

5.1 Limitations

As already mentioned in the relevant chapters the smaller data set includes some inaccuracies due to the way we dealt with some categorical data. We believe however that these steps were justified and do not make the resulting classification algorithms unusable.
For the large data set, we probably have too many variables which lead to overfitting. We purposely did this to show the difference between the two. It still could be a problem for example when estimating a multinomial logistic regression. We probably have multicollinearity in our data and therefore a biased result. Essentially, we cannot interpret the estimates as a causal relationship between features and house prices. We can only use them to predict the class of new data.

There is the risk of overfitting especially for certain classifiers (e.g., KNN) but also due to the small sample size for some classes (e.g., class 4), as these few observations become way more important and therefore more weighted in the algorithms. We tried to tackle this issue with the use of SMOTE. Yet, we still suffered from overfitting in some algorithms.

5.2 Literature

Blagus, R., & Lusa, L. (2013). SMOTE for high-dimensional class-imbalanced data. *BMC bioinformatics*, 14, 1-16.

Zach (9. Mai 2022). How to Interpret the Classification Report in sklearn (With Example). *Statology*. https://www.statology.org/sklearn-classification-report/

Zimmermann, B. (2023). Machine Learning in Finance - A Gentle Introduction with a Focus on Applications in Python