Classification_of_GroupNameDe

Group: Firat Saritas, Simon Staehli

Description:

The goal of the this task is to predict the multiclass-labels of the feature *GroupNameDe* with three different models as well as the comparisson between those with an appropriate metric or metrics.

Data Import

First of all we need data to build the classifier. We import our data by setting the class Util with the correct path to the data. The Util class loads the data plus it sets appropriate data types for the features.

Now we drop the the feature *RealEstateTypeID* as it is exactly the same as the GroupNameDe, which would be to easy to predict with.

Pre-Processing

Drop Redundant Columns *(S, M, L) + Other

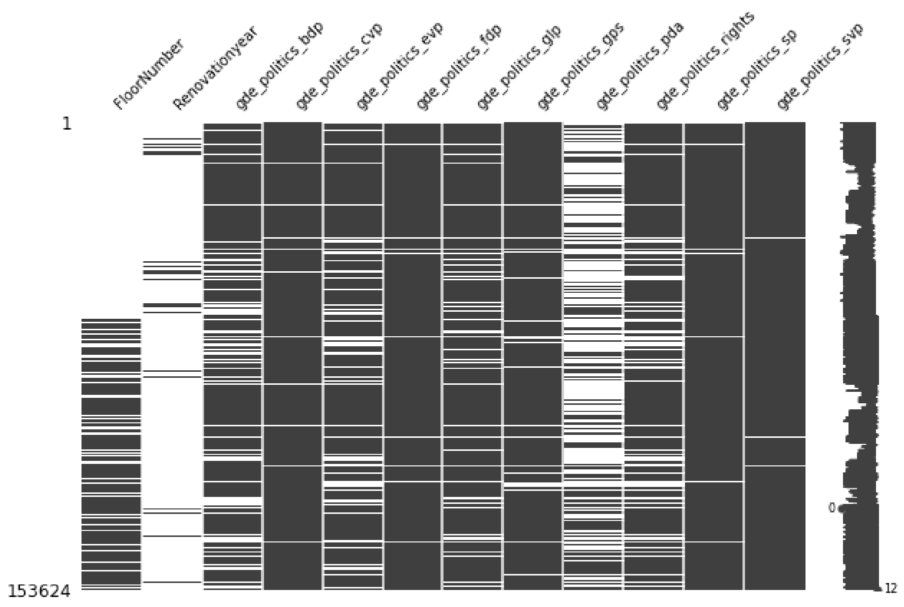
In our EDA part we detected that there are columns which are almost equal to each other or do not bring any advantage in our opinion.

For example we have the column LastUpdate which is for sure nice to have, but it has no real connection to the *GroupNameDe*.

Missing Values

Most of ML models can not handle missing values. For the sake of this we will fill them. To force equality in our notebooks we have implemented a class with several functions to fill missing values.

But first have a look on all features with missing values. We achieve this by using the matrix of the programming library missingno, which is very useful when it comes to a quick overview over the missing values.



```
Fill missing values...
FloorNumber filled.
RenovationYear filled.
Politics filled.
AreaProperty filled.
Remaining Attributes filled.

Are there any missing values left? --> False

New Shape of Dataframe: (153624, 53)
```

Create New Features

In this step we generate new features along our existing features. We built a class for this too. Following Features will be addedd to the dataframe:

AreaTotal: AreaLiving + AreaProperty added together

has_space: If AreaProperty is not zero, than an object has additional space --> 1, else --> 0

Renovation-Built: Differnce between Renovation Year and Built-Year

is_renovated: If renovation year is bigger than 0 than it is renovated

inserts_per_zip: Count of objects per Zip

Encoding of Categorical Features

For the encoding of the categorical features we use a count encodings, which means we replace it by the aggregated count per category. We will encode GroupNameDe later in the context after the upsampling.

```
Categorical Features:  ['Name', 'StateShort']
```

```
Currently processing.. Name
Currently processing.. StateShort
```

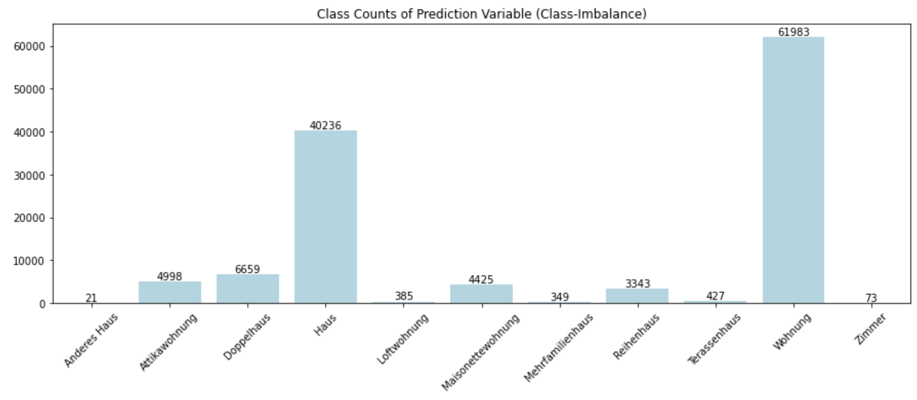
Prediction Variable

Train-Test Split for Upsampling

Now we generate a Train and a test set for the upsampling of the unbalanced classes in the target variable "GroupNameDe". As we do not need the upsampling for all our algorithms, for example DecisionTree or RandomForest, we split the data already now in two separated sets. We will use a validation set for the KNN with upsampled data and then test it agains our test set.

```
Shape X: (122899, 56)
Shape y: (30725, 56)
```

Let us have a look on the prediction variable **GroupNameDe**. Therefor we plot the aggregated count on each class of the feature.



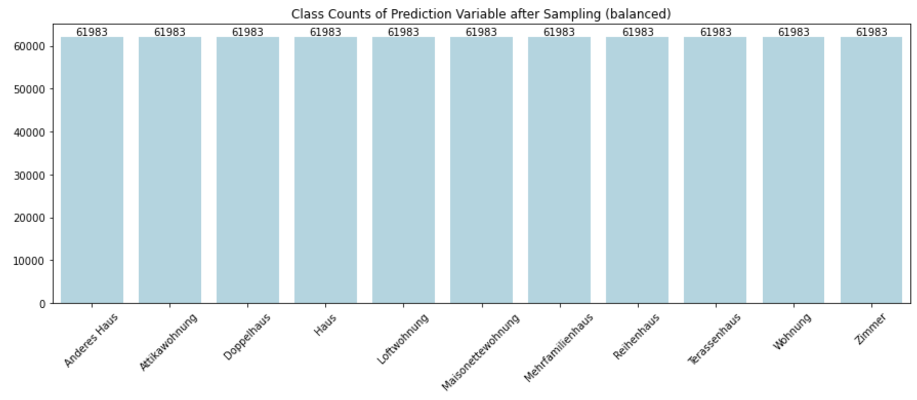
Upsampling or Downsampling?

Upsampling makes more sense, because of the underlying class counts. The lowest class count here would be around 20. This would result in a very small dataset with less information and variance in it.

```
Upsampling will produce 558914 new Rows.

Required Memory for Upsampling: 275 MB


New shape of dataframe: (681813, 56)
```



Modelling

For the prediction of the *GroupNameDe* we decided to implement and compare these three different **models**:

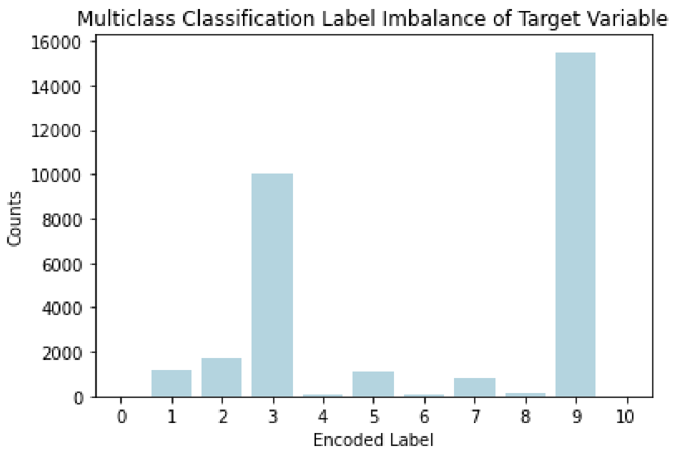
- DecisionTreeClassifier
- RandomForestClassifier
- K-Neareast Neighbour

Train-Test-Split (Train-Validation-Test Set)

For the upsampled dataset I will use a Validation Set to train the model, because of other underlying training data. After the training and the validation of the unbalanced data, we will also test it against an independent test set.

Evaluation of the Metric

Now let us have a look on the target variable. It conveys us which metrics we should use for the multiclass prediction problem.



As we can see in this plot above, there is a strong class imbalance in the testset. This conveys us to choose a metric which takes this into account.

For the chosen Metrics we took only metrics which are appropriate for classification and some of us already worked with for example in the essay for GDS or Recommender System. To compare models the time of the model training and so on would be also a metric, but as we have different data sets regarding class balance for different models, we did not include it. For example the upsampled dataset for KNN and the not upsampled datasets for the Tree Algorithms.

Decision Points for the metric:

- Class Imbalance in test set
- Unbiased, all classes should be weighted equally
- Investigate the classication power of each model (how good are our predictions for the classes?)
- Models can be compared with this metric
- How much true classes did we not hit by our classification

We took following **metrics** to compare the chosen models with each other.

Precision:

As we would like to know how good our model performs when predicting the true class, we want our model to predict the classes as precise as possible preferably without any wrong prediction of the label. Precision is a good indicator metric for this purpose. Precision answers the question: How much of the Postives were predicted? The best scores is 1, which means all positives were predicted and no wrong positve prediction was made. For us this score is very interesting because for all our models we would like a precise as possible predicition of the class with less wrong positive predictions.

$$Precision = \frac{TP}{TP + FP}$$

Recall:

Recall is the fraction of detected relevant instances over all relevant instances in a class. The best score which can be achieved by a classifier is 1, which means that all relevant predicted instaces were correct. Recall is beeing calculated over the horizontal axis, so the ratio between correct predicted classes to the ratio of correct classes which have not been detected by the algorithm.

$$Recall = \frac{TP}{TP + FN}$$

Averaging:

The Precision score and Recall score were calculated over each class. Now the question is what kind of averaging should be used in our usecase to compare the models with each other?

"Macro": As we want our model to treat each class equally we took the averaging macros, which calculates the mean along each class and weightes all of them equally. The problem with the macro averaging is, if we have one class with less data and this class was not hit correctly by the classifier than the maximum score (i.e. Precision) which can be reached out of 10 classes would be limited to 90%. But in our opinion a good classifier should also classify the less frequent classes in our dataset.

Mattheus Correlation Coefficient:

This correlation shows the balance between the correct predicted classes and the incorrect predicted classes. It reaches from -1 to 1. It is a good measure to compare models because of its class insensitivity. It puts equal weights on each classification result (TP, TN, FP, FN). If the amnount of correct predicted classes is equal to the amount of wrong predicted classes than the coefficient turns out to be 0. If the amount of corrected predicted classes ends up higher than the amount of wrong predicted classes than the coefficient turns out to be positive, same in the other direction.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Helper Functions

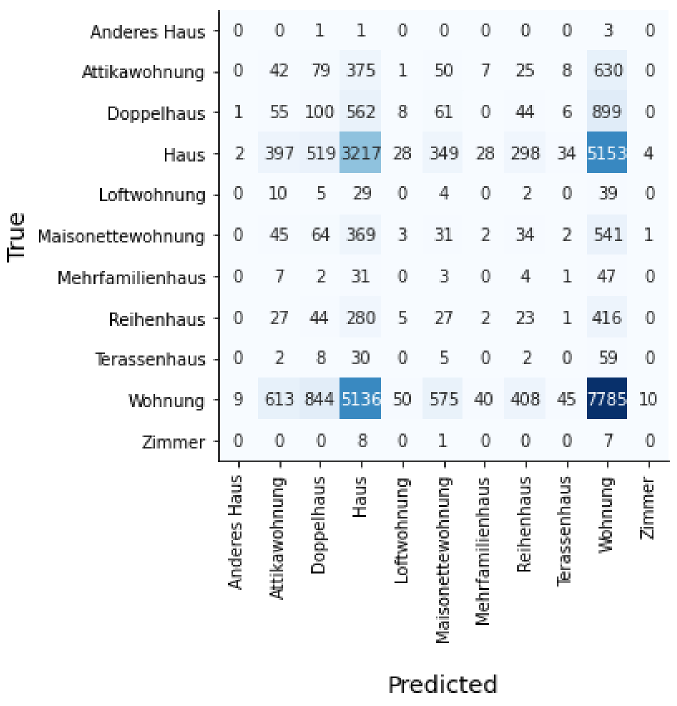
Baseline Model

To compare our built model I will use the Dummy Classifier from sklearn as baseline model.

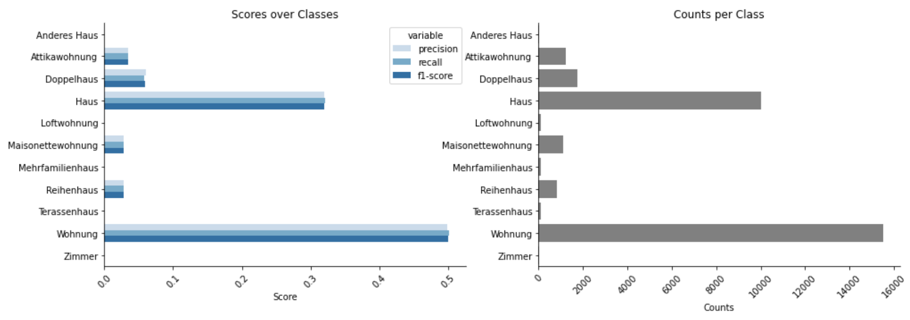
The Documentation to the DummyClassifier can be found here:

[https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html)](https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html)

As strategy I used the parameter *stratified* which uses the trainings set class distribution for the prediciton. For example if a class probability is 60%, there will be this class predicted in 6 out of 10 cases.

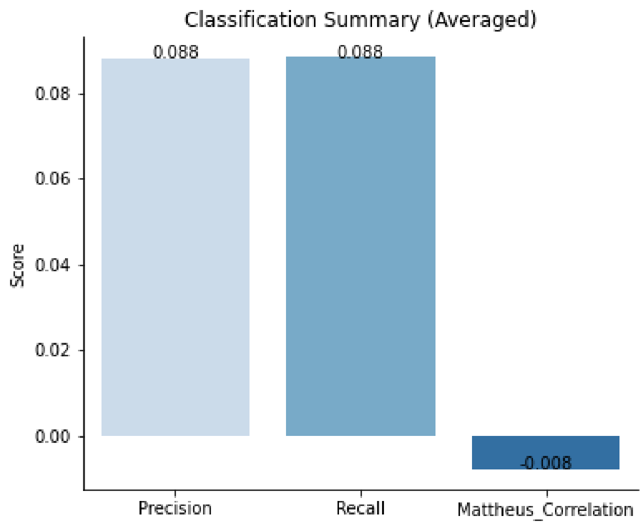


True \ Predicted	Anderes Haus	Attikawohnung	Doppelhaus	Haus	Loftwohnung	Maisonettewohnung	Mehrfamilienhaus	Reihenhaus	Terassenhaus	Wohnung	Zimmer
Anderes Haus	0	0	1	1	0	0	0	0	0	3	0
Attikawohnung	0	42	79	375	1	50	7	25	8	630	0
Doppelhaus	1	55	100	562	8	61	0	44	6	899	0
Haus	2	397	519	3217	28	349	28	298	34	5153	4
Loftwohnung	0	10	5	29	0	4	0	2	0	39	0
Maisonettewohnung	0	45	64	369	3	31	2	34	2	541	1
Mehrfamilienhaus	0	7	2	31	0	3	0	4	1	47	0
Reihenhaus	0	27	44	280	5	27	2	23	1	416	0
Terassenhaus	0	2	8	30	0	5	0	2	0	59	0
Wohnung	9	613	844	5136	50	575	40	408	45	7785	10
Zimmer	0	0	0	8	0	1	0	0	0	7	0



As we can see the sums for precision and recall are approximately the same over the prediction axis and the true axis. That is why the precision score is equal to the recall score which implies that the f1 score is same size like precision and recall. We can also spot that there were conflicts with the two major classes of "Haus" and "Wohnung" which were mixed with each other many times.

The classification summary shows us how well our model predicted over each class. we can see that our baseline classifier actually everytime not hit one of the less frequent classes like room for example. It performed ok when predicting the more frequente classes for example room.

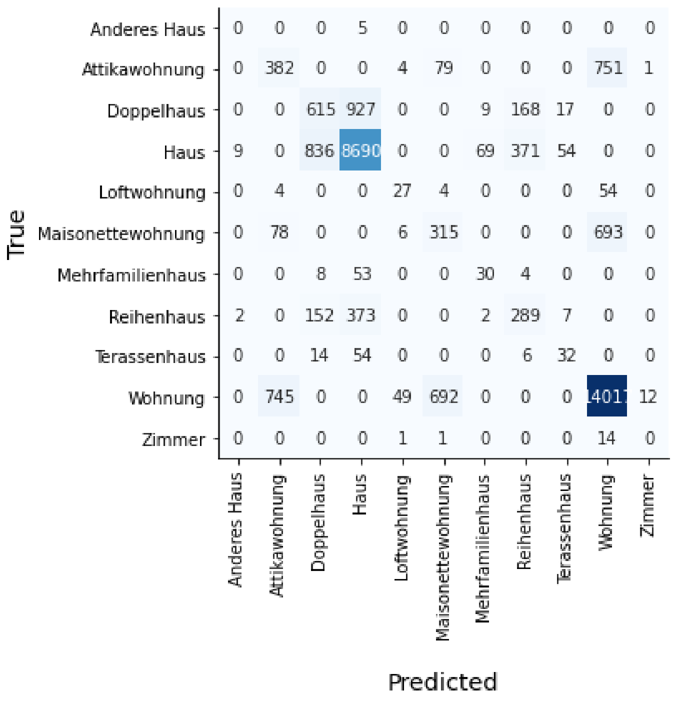


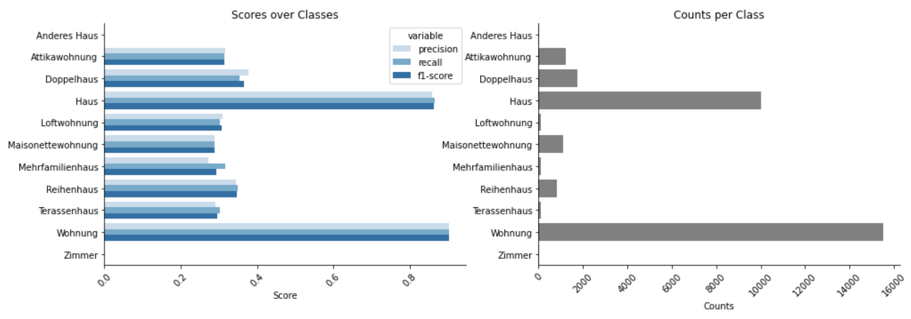
The precision of our model scored 10%, which is very low and shows us that our model does predict 90% of the time wrong for each class in average. The precision of our model is almost exactly the same as the recall score. We think this is the case because of the imbalance in the dataset which weights also the majority classes more than the other ones. This is also visible in the confusion matrix. As we can see as well is that the MCC is very bad, it almost reached the badest score possible of 0 -> Almost same amount of wrong predicted classes like correct predicted classes.

Now we have a baseline model to compare with our other models. Of course the predictions here are made simple, but now we know if our model scores less than the baseline model, it is not useful for the comparisson.

DecisionTreeClassifier

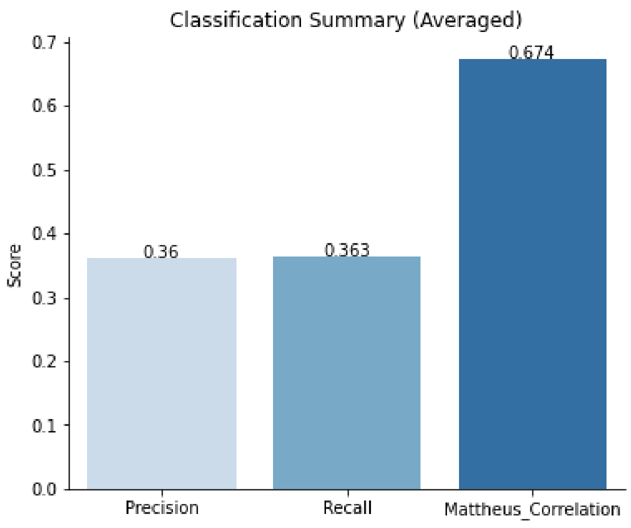
As the second model we have chosen a DecisionTreeClassifier. Let us fit the model and see what the DecisionTreeClassifier achieves when it predicts the never seen test samples.





The confusion matrix shows a clearly better result than the one from the DummyClassifier. Whereas the DummyClassifier classifed mostly at random our tree model hit way more diagonal elements in the matrix through the band. It is also visible that our classifier often mixed between the classes with suffix House and Wohnung.

The tree predicted all major classes very precisely as we can see in the report. Again the classifier had trouble with the less frequent classes particularly classes with a frequency of less than 20, which belong to those classes which are the hardest to detect for our classifier.



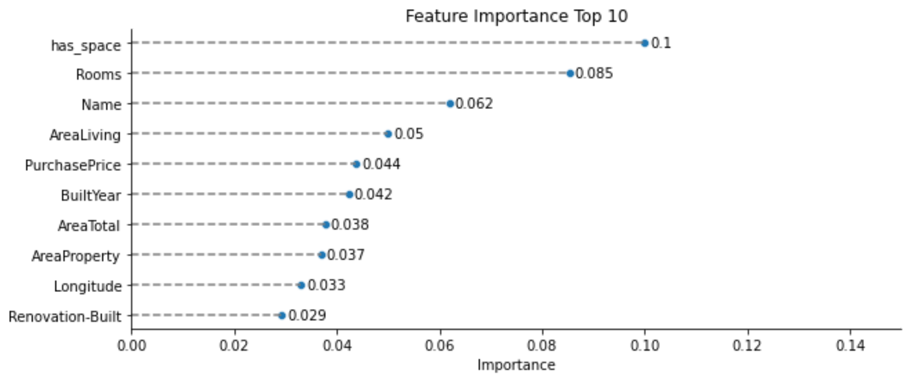
The tree scores much better regarding to the metrics. The Precision and Recall scored equal scores which is also visible in the confusion matrix: the Horizontal rows and the Vertical rows are alike. The Mattheus Coefficient of this model is approx 66% which tells us that we have hit many of the true labels in the comparisson to wrong predicted labels.

The Decision Tree is likely to overfit the data. Therefore I will adapted the parameters to underweight this behaviour and prune the tree with the parameter min_samples_split which defines the tree to perform a split by the given amount of samples. If the parameter min_samples_split kept higher than the tree is advised to stop splitting a node any further.

Best parameters found in GridSearch:

- 'ccp_alpha': 0,
- 'class_weight': 'balanced',
- 'min_samples_leaf': 1,
- 'min_samples_split': 2

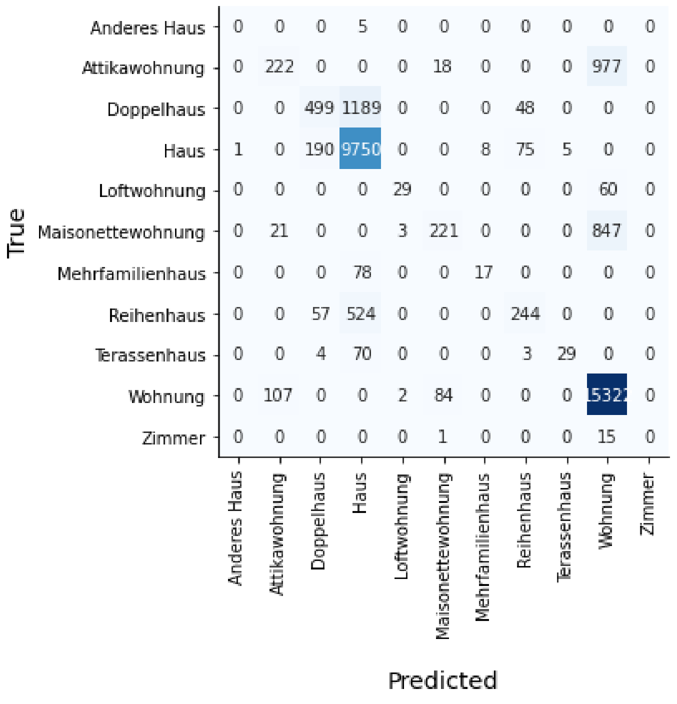
These parameters seem to be the best for having a balanced precision and recall. Of course there are more parameters we could test out, but the goal is to compare 3 models and not to accomplish the best possible classifier.



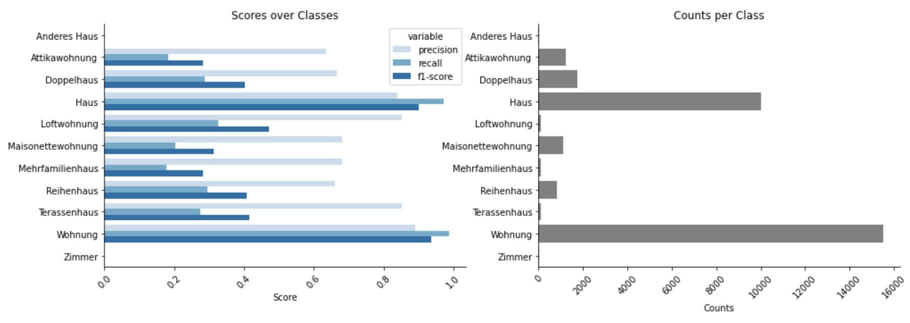
As we can see in the plot above the highest importance is shown by the feature HouseObject, which indicates a house in the dataset. It seems that a big impact is also show by the name of the website of what kind of object it is in the end.

RandomForestClassifier

Now let us see how the RandomForest performs on our testset. RandomForest is a ensemble learner of independent Decision Tree's (Forest) which we expect to be better than just an ordinary Decision Tree in this case.

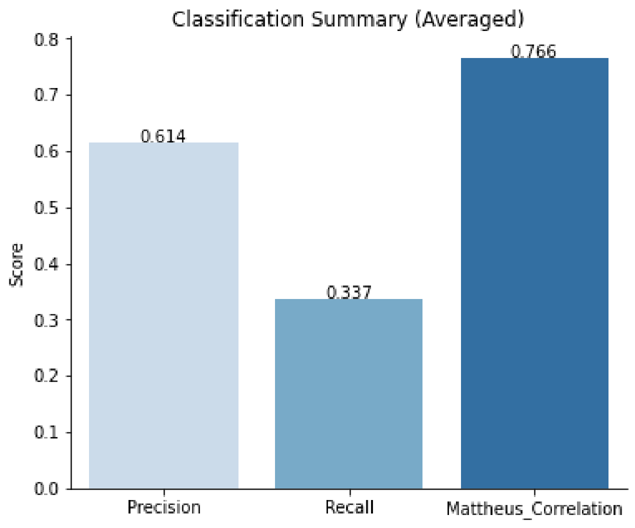


True \ Predicted	Anderes Haus	Attikawohnung	Doppelhaus	Haus	Loftwohnung	Maisonettewohnung	Mehrfamilienhaus	Reihenhaus	Terassenhaus	Wohnung	Zimmer
Anderes Haus	0	0	0	5	0	0	0	0	0	0	0
Attikawohnung	0	222	0	0	0	18	0	0	0	977	0
Doppelhaus	0	0	499	1189	0	0	0	48	0	0	0
Haus	1	0	190	9750	0	0	8	75	5	0	0
Loftwohnung	0	0	0	0	29	0	0	0	0	60	0
Maisonettewohnung	0	21	0	0	3	221	0	0	0	847	0
Mehrfamilienhaus	0	0	0	78	0	0	17	0	0	0	0
Reihenhaus	0	0	57	524	0	0	0	244	0	0	0
Terassenhaus	0	0	4	70	0	0	0	3	29	0	0
Wohnung	0	107	0	0	2	84	0	0	0	15322	0
Zimmer	0	0	0	0	0	1	0	0	0	15	0

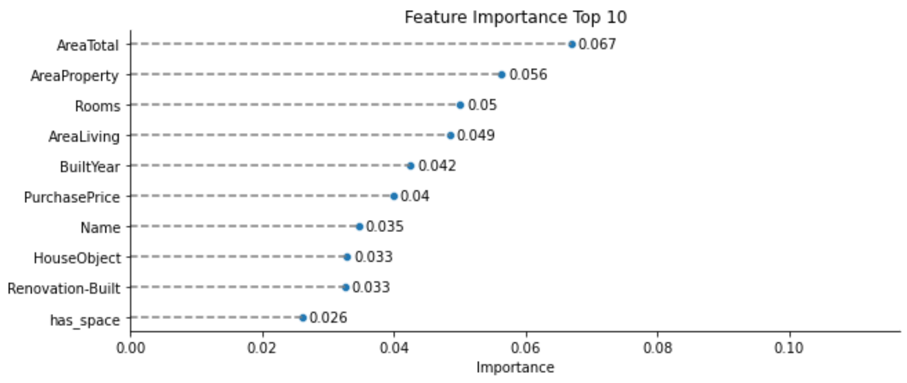


The confusion Matrix shows us that there were less prediction on wrong classes as we had with the Decision Tree. It also shows us that we have a better precision with this model regarding to the others which mostly had equal scores of Precision and Recall. This model works more precise on the test set. Again this model had trouble to predict the less frequent classes of Zimmer und Anderes Haus which have low frequencies. We tested the parameter *class_weight* without specifing balanced which had the same outcome as we had with balanced for less frequent classes.

Our class summary shows us the same as the confusion matrix showed us. High Precision than Recall, which means our model did not find actual classes. Compared to the Decision Tree which scored mostly around 30% for each metric. It seems like our model finds a large amount of the most frequent classes, which is visible on the Recall score as well.



The RandomForest shows us overall the best performances regarding of Precision and MCC. According to our precision score it classifies over 60% averaged over all classes correct in comparisson to all classification of the classes. As the performance of The Recall Score for the two major classes is high which means that the averaging was biased by the large classes, so the Decision Tree scored better than the Random Forest for the Recall.



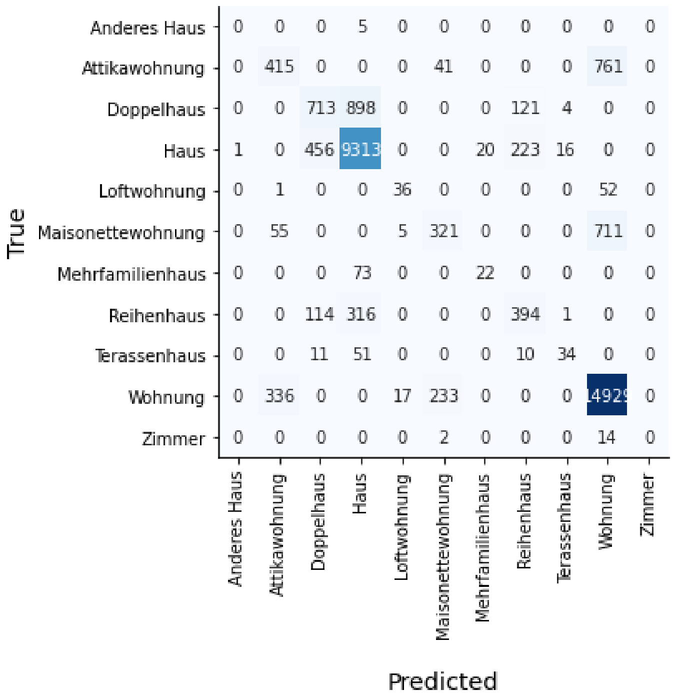
Now we will perform a GridSearch to search for the best parameter for the model. We would like to increase the Recall and Precision to balance our model, therefore we decided to chose the metric F1-SCore which is the harmonic mean of both.

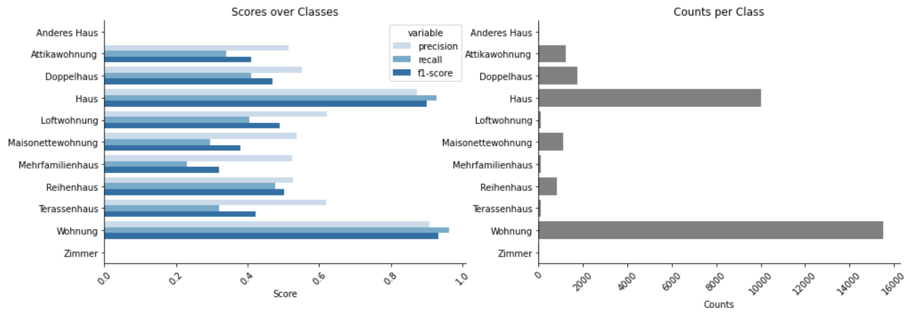
The best parameter found with the gridsearch:

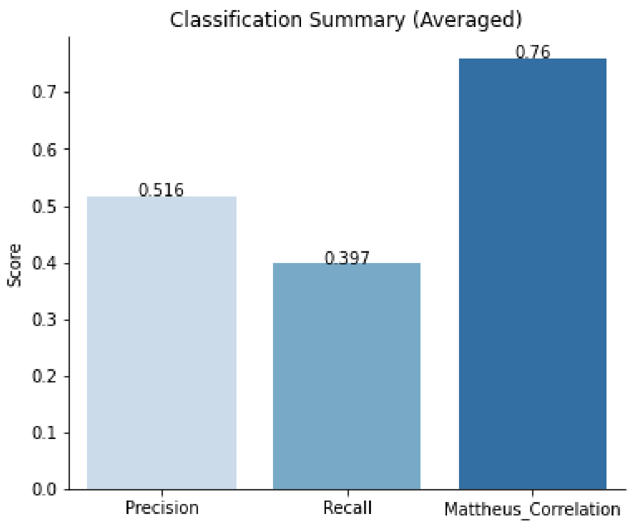
- 'class_weight': 'balanced',
- 'criterion': 'gini',
- 'min_samples_split': 10,
- 'n_estimators': 200}

Fit the new model

```
RandomForestClassifier(class_weight='balanced', min_samples_split=10,
                       n_estimators=200, n_jobs=-1)
```





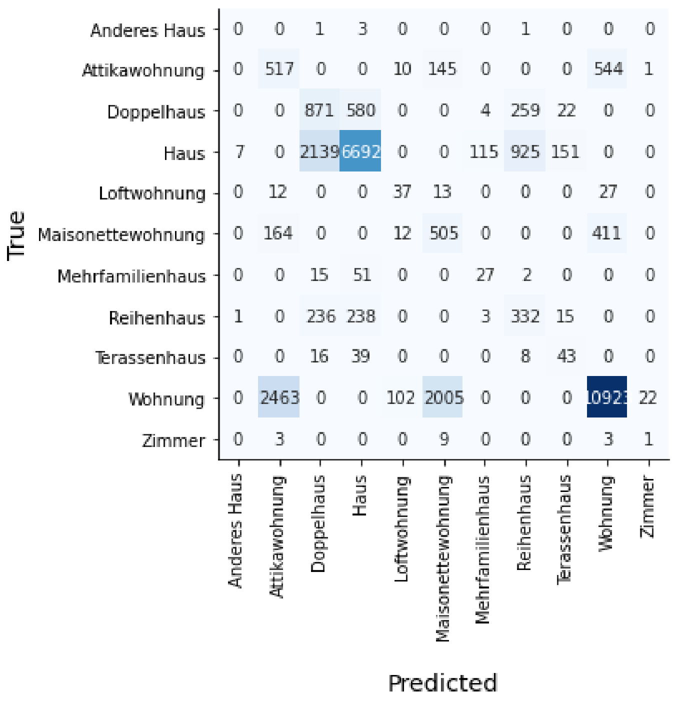


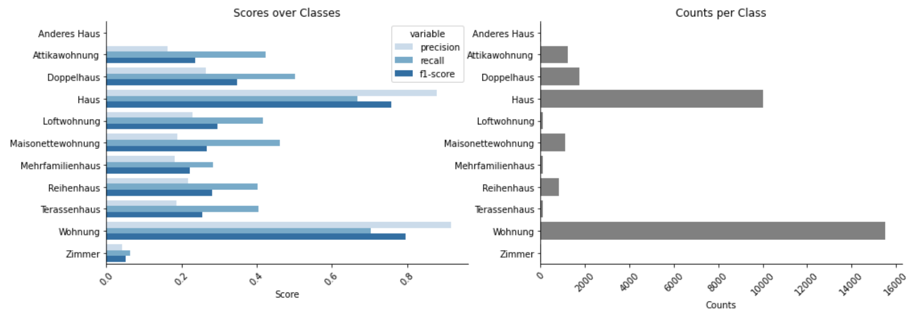
In comparisson to the first attempt, we were able to increase the overall Recall score, but decreaesed the Precision of our Model. The optimization barely decreased the MCC, which means that the ratio of "good" prediciton still overweights the "bad" predictions.

K-Nearest Neighbour (KNN)

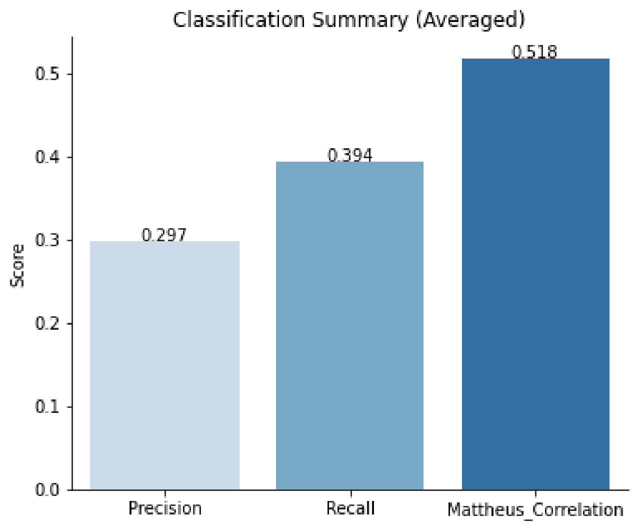
First we implement a K-Nearest Neighbour with default parameters according to its neighbours. For the Trianing of the K-Nearest Neighbour we will use the upsampled data + we will also perform a fit without the balance data. Furthermore we also use the Min-Max Scaler from SKlearn to scale the predictors properly to not overweight the distances between the neighboorhoods.

```
Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                ('kneighborsclassifier', KNeighborsClassifier(n_jobs=-
1))])
```





As we can see the KNN algorithm predicts the larger classes more precisely than it does for the smaller classes. Through the band it scores better precision than Recall.



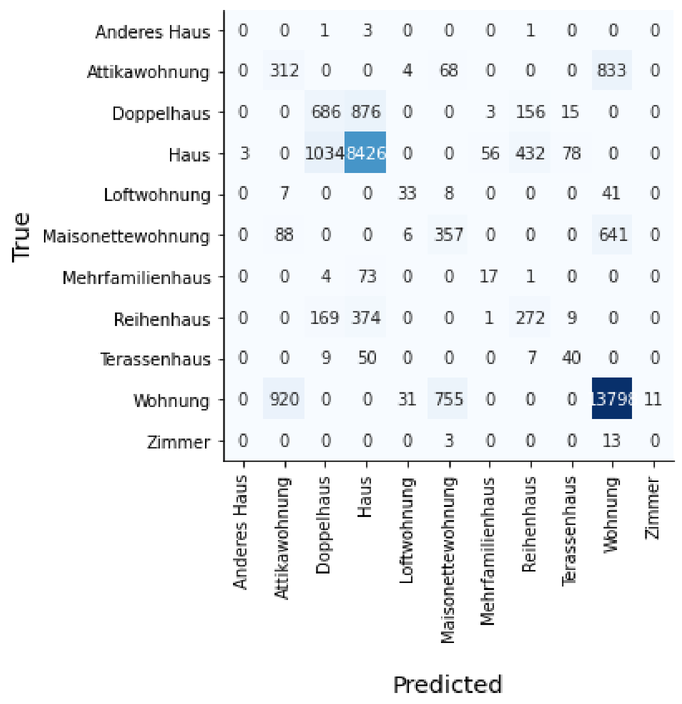
Let us now stick on the parameters and look if we can get better with the KNN to predict the classes. For that we created a GridSearch with different Neighbors and Weight for the classes.

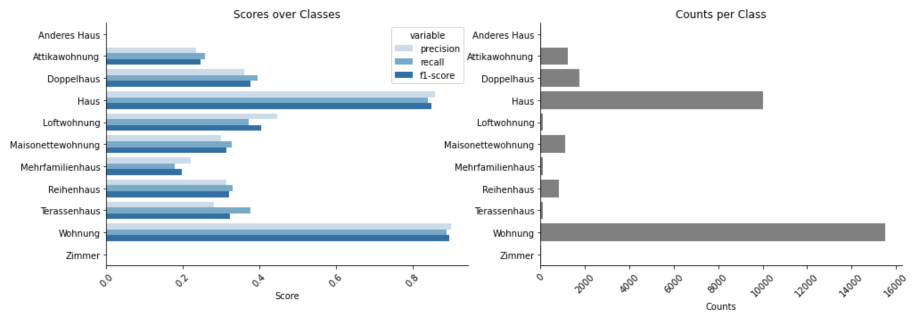
Best Parameters found by using the GridSearch:

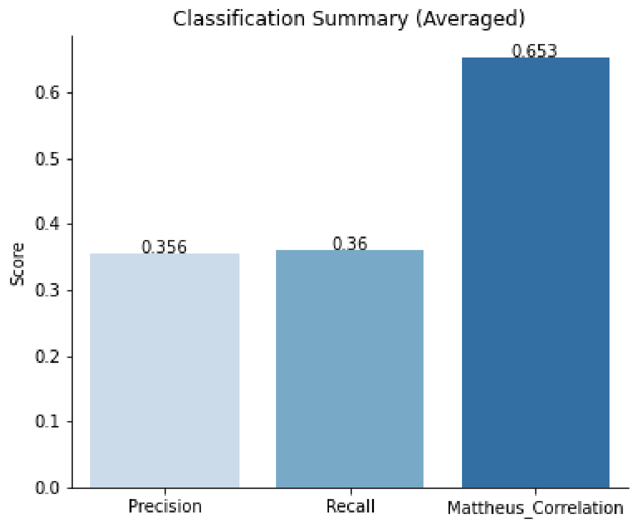
- 'knn__n_neighbors': 2,
- 'knn__weights': 'distance'

Now fit the new model with the new parameters and predict the classes within the test set.

```
Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                ('kneighborsclassifier',
                 KNeighborsClassifier(n_jobs=-1, n_neighbors=2,
                                      weights='distance'))])
```



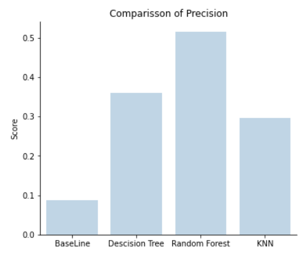


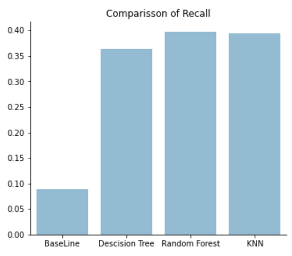


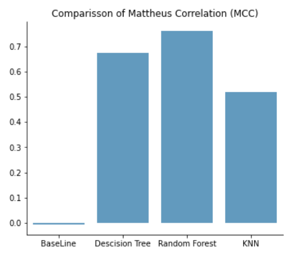
In this attempt the KNN performed a little bit better than in the first attempt, which enhanced the overall performance of the algorithm regarding to its precision and recall.

As we optimized our KNN regarding to th f1-score the averages of Precision and Recall scores are now pretty much equal to each other. Now we have four models to compare with each other. Therefore, the classification summary with the calculated Precision, Recall and MCC will indicate us, which model performed the best.

Comparisson







Precision:

First, all our models performed better than the baseline model, which is a good indicator. In the comparisson of the Precision each model did not perform specifically good, as we can spot on the scoring range from 8% to 55%. Here, the Random Forest performed the best. Though a score of 50 % means the predictions of the classifier were in 50% wrong in average. This view can be strongly biased when our classifier for example hits a group of classes to 100 % and another group of classes never. In fact this is a problem when it comes to such metrics. As we investigated this characteristics with the classification report we could see that none of the algorithms predicted the less frequent classes, not even the upsampled for the K-Nearest Neighbor, which resulted in class precision of 0. If we take into account the hardness to predict the less frequent classes the best possible class if we subtract those 2 classes would be 81%.

> Winner: RandomForestClassifier

Recall:

The same here for the models compared with our Baseline model: Our models performed a bunch better. Again the RandomForest achieved the best results, Close to the RandomForest is the KNN regarding to the 2nd and 3rd place, we have the KNN and the DecisionTree algorithms. It must be said, that the KNN took a long time to make the predictions, which is a large minus point, when predicting many datapoints. Overall our classifier did not score very good according to the Recall metric. The best classifier reached approx. 40% Recall, which means that averaged over all classes over 60% of the true classes were not detected by all of our models.

> Winner: RandomForestClassifier

Mattheus Correlation Coefficient:

We took the Correlation as an independent measure from the common classification metrics like Precision and Recall, which were actually created for binary classifications. The MCC is the only metric of those three which supports multiclass per default without any averaging. According to the formula of the MCC, more weights is given to classes with more counts in the testset. In the comparisson between all our algorithms we have the tree models performing best. This indication shows us that the RandomForest classifies the classes best. It has the highest correlation with the targets of the test set, followed by the DecisionTree.

> Winner: RandomForestClassifier

Conclusion:

The comparison of the three models should be taken with a grain of salt, as we don't know if we really got the best possible out of our models. Of course we have fed the models with the same data, but different models also require different methods of pre-processing. The winner of the three models according to our metrics is the RandomForest. In our opionion RandomForest is surely not the simplest model of all three and the algorithm is popular for its good performance straight out of the box without any adjustings. It is to mention too that we have adjusted the RandomForest's Hyperaprameters with a GridSearch get a better balance between Recall and Precision, which was in the beginning dominated by the Precision.

Sources

[1] Brownlee, Jason. „Failure of Classification Accuracy for Imbalanced Class Distributions“. Machine Learning Mastery (blog), 31. Dezember 2019. https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/ (https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/).

[2] Data Science Stack Exchange. „evaluation - Micro Average vs Macro average Performance in a Multiclass classification setting“. Zugegriffen 6. Januar 2021. https://datascience.stackexchange.com/questions/15989/micro-average-vs-macro-average-performance-in-a-multiclass-classification-settin (https://datascience.stackexchange.com/questions/15989/micro-average-vs-macro-average-performance-in-a-multiclass-classification-settin).

[3] Ferri et al. „An Experimental Comparison of Performance Measures for Classification“. Scientific. Pattern Recognition Letters. Departament de Sistemes Informàtics i Computació, Universitat Politècnica de València, València 46022, Spain, 18. Dezember 2006.

[4] „F-Score“. In Wikipedia, 1. Januar 2021. https://en.wikipedia.org/w/index.php?title=F-score&oldid=997534712 (https://en.wikipedia.org/w/index.php?title=F-score&oldid=997534712).

[5] kjytay. „What Is Balanced Accuracy?“ Statistical Odds & Ends (blog), 23. Januar 2020. https://statisticaloddsandends.wordpress.com/2020/01/23/what-is-balanced-accuracy/ (https://statisticaloddsandends.wordpress.com/2020/01/23/what-is-balanced-accuracy/).

[6] „K-Nearest Neighbour“. In Wikipedia, 2. Januar 2021. https://en.wikipedia.org/w/index.php?title=K-nearest_neighbors_algorithm&oldid=997930459 (https://en.wikipedia.org/w/index.php?title=K-nearest_neighbors_algorithm&oldid=997930459).

[7] Meli. The Definitive Guide to the Matthews Correlation Coefficient, 2020. https://www.youtube.com/watch?v=u-Ez7trpNrM&ab_channel=DataScienceBits (https://www.youtube.com/watch?v=u-Ez7trpNrM&ab_channel=DataScienceBits).

[8] „Precision and Recall“. In Wikipedia, 23. Dezember 2020. https://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=995861774 (https://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=995861774).

[9] „Receiver Operating Characteristic“. In Wikipedia, 28. Dezember 2020. https://en.wikipedia.org/w/index.php?title=Receiver_operating_characteristic&oldid=996807138 (https://en.wikipedia.org/w/index.php?title=Receiver_operating_characteristic&oldid=996807138).

[10] „Why Are Precision, Recall and F1 Score Equal When Using Micro Averaging in a Multi-Class Problem? – Simon’s Blog“. Zugegriffen 6. Januar 2021. https://simonhessner.de/why-are-precision-recall-and-f1-score-equal-when-using-micro-averaging-in-a-multi-class-problem/ (https://simonhessner.de/why-are-precision-recall-and-f1-score-equal-when-using-micro-averaging-in-a-multi-class-problem/).