Wien: 15.12.2019

ASSIGNMENT 2
CLASSIFICATION TASK

GROUP 21

SOPHIE RAIN (01425316),

LUCAS UNTERBERGER (01325438),

PETER STROPPA (01326468)

184.702

MACHINE LEARNING

INHALT

DATASETS

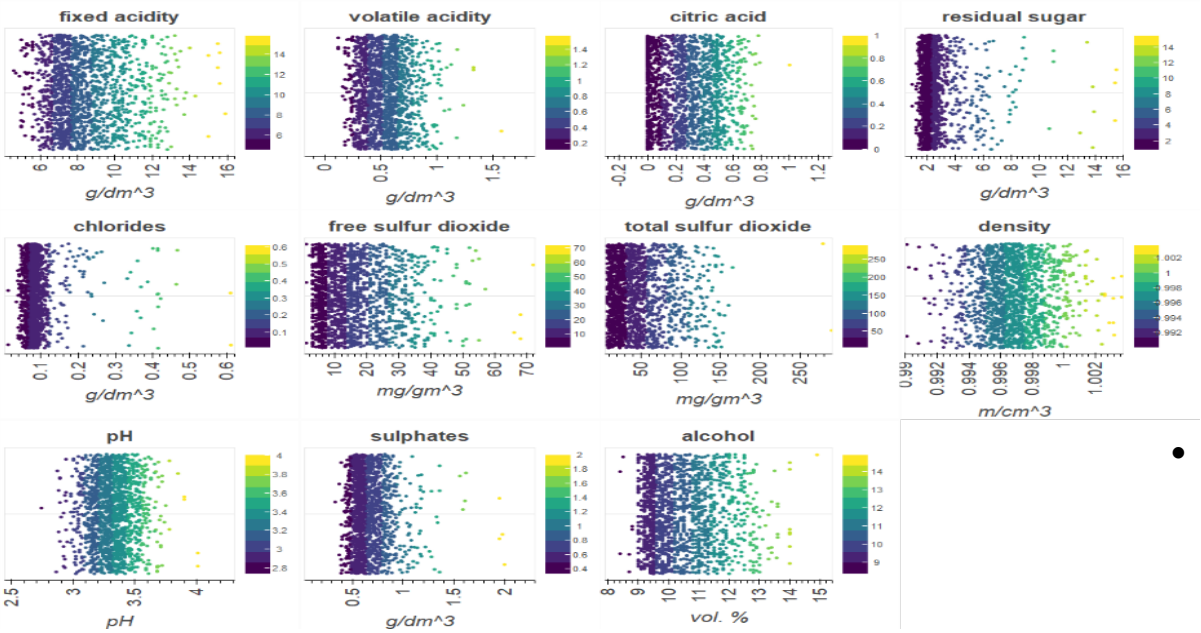
IN THIS SECTION WE GIVE A SHORT OVERVIEW ABOUT SOME GENERELL CHARACTERISTICS OF THE DATASETS

WINE

Link to the dataset: http://archive.ics.uci.edu/ml/datasets/Wine+Quality
The wine quality dataset consists of 1599 red wine samples of the Portuguese "Vinho Verde" wine. The samples were classified according to their quality. This classification is purely based upon sensory data obtained by the median of 3 evaluations made by wine experts.

A brief overview of the Red Wine Dataset:

- 11 attributes
- 10 classes (0 – 10) / only 3 – 8 are assigned
- Only numeric values (no missing values)
- Several different ranges and units according to attributed



-

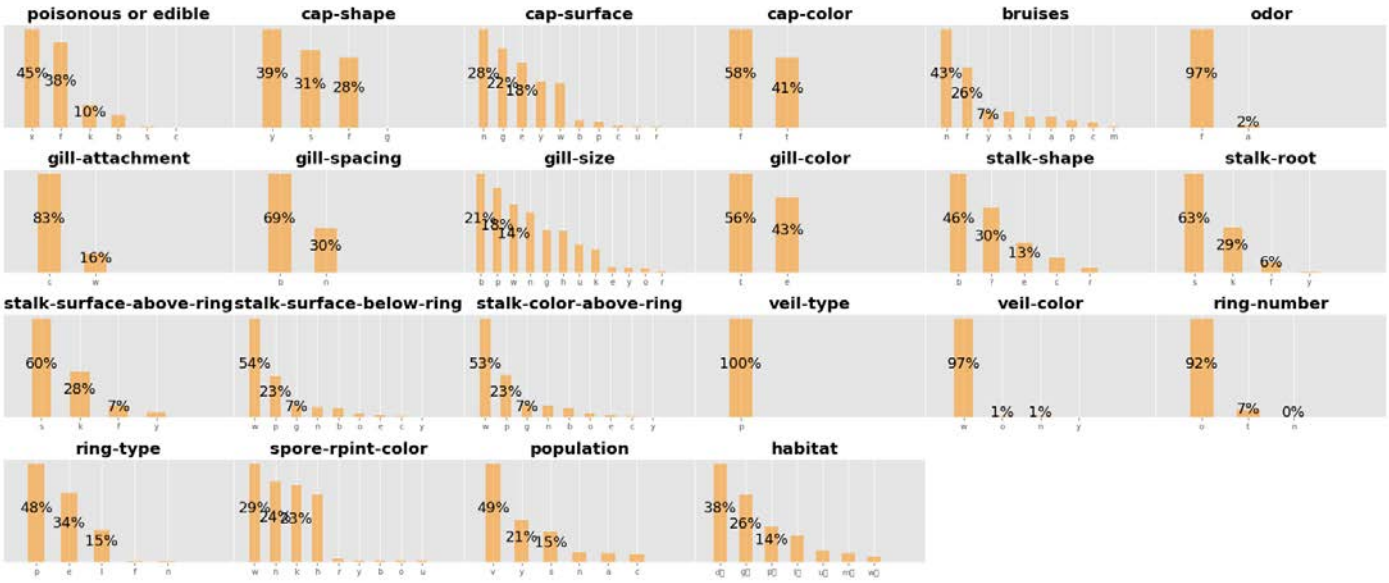
To get precise information about the attributes we studied the underlying distribution. For more in depth analysis and further information refer to our extensive analysis of the datasets in Exercise zero.

MUSHROOMS

Link to the dataset: http://archive.ics.uci.edu/ml/datasets/Mushroom
The mushroom dataset consists of 8124 samples of gilled mushrooms in the Agaricus and Lepiota Family. They are classified into 2 classes edible or poisonous. Samples of unknown edibility are classified as poisonous as well.

A brief overview of the Mushroom Dataset:

- 22 attributes (all nominal)
- 2 classes
- Only nominal values (also missing values)
- 4208 edible and 3915 nonedible



CONGRESS

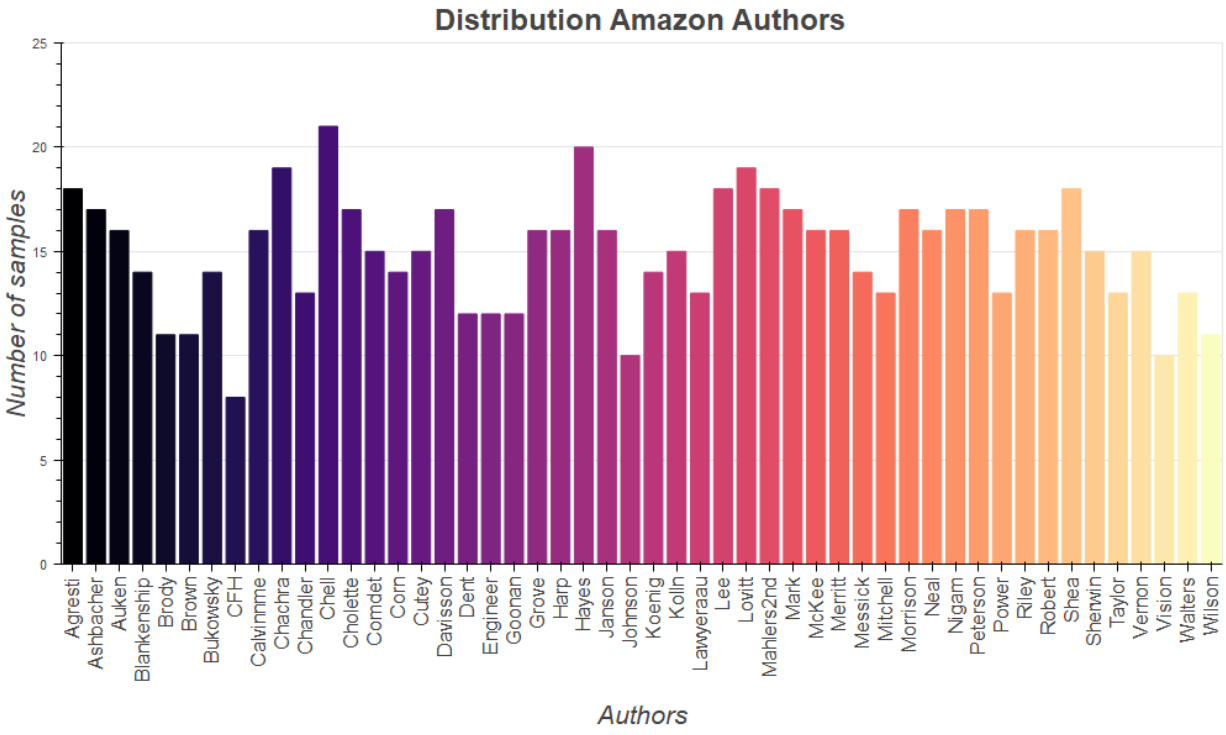
The Congress dataset, obtained from Kaggle, contains 217 samples, each one having 18 attributes. The target value is ‘class’ which has only 2 different values: ‘democrat’ or ‘republican’. The other attributes are labeled in a way that leads one to assume that they are laws or decisions that get voted on in congress. Additionally, they only contain nominal values ‘y’, ’n’ and ‘unknown’. With one exception being the attribute ‘ID’, which is a unique number for every sample throughout our training set?
Therefore, our classification task is to figure out whether the person voting belongs to the democrat or the republican faction. To do that, we are given a set of votes in favor of, or against certain laws.
In our dataset there are 137 democrats and 81 republicans, meaning that our classes are not equally distributed. There are 62,8% democrats and 37,2% republicans. We also omitted the attribute ‘ID’, since there is no useful information a classification algorithm can gain by considering it.

AMAZON

Refer to the TUWEL dataset registration page for the link to the dataset.

The dataset consists of 750 samples. Each representing a review of some product bought on Amazon. We assumed each feature to be a word or a specific expression and the corresponding value the number of occurrences of this word in the review. Hence, all the attributes are given as ordinal numbers. There are 50 classes, which are the names of the review's authors. In the training set the distribution of the classes is quite diverse (8-21). Since the role of each author is the same, there are no classes more important than others. There are no missing values in the dataset.

Due to the big number of attributes, a fine-grained analysis of the dataset is not possible or at least comes with a lot of effort (e.g. correlation).



GENERIC PROCEDURE

IN THIS SECTION WE DESCRIBE HOW WE EVALUATED OUR DATA, WHAT MEASURES, PREPROCESSING AND METHODS WE USED.

We used Python for this exercise. The machine learning package we worked with was Sklearn. Dataset manipulation was done using the pandas and Numpy packages.

COMPARISON OF DATASETS

This table shows the most important properties of our data:

Dataset	# samples	# attributes	Missing values?	# classes	classes eq. important?	Types of attributes
Amazon	750	10 000	no	50	yes	ordinal
Congress	217	18	yes (unknown)	2	yes	categorical (2 values)
Wine	1599	11	no	11	no	numeric
Mushrooms	8124	22	yes	2	no	categorical (2-10)

It shows the diversity of the datasets nicely. The number of samples ranges from 217 to 8124, the attributes from 11 to 10 000. Considering the unknown values from the congress dataset missing, we have two datasets with missing values. The column: "*classes eq. important?*" states whether the roles of the classes are symmetric. This is the case for Amazon and Congress, but not for the other two. Finally, the *'types of attributes'* column proves the diversity of our attributes.

EVALUATION TECHNIQUES AND MEASURES

We decided to work with cross-validation as evaluation method. This way we avoid overfitting and use every sample in the same way. Since we wanted an 80:20 relation in every split, we chose 5 splits.
Before splitting the data, we shuffled it. This way we wanted to make sure the order of samples does not influence our results. Splitting without sampling may result in biased learning.
We decided to use neither stratification, nor having a validation set.
The reason for not doing stratification is that we don't know the distribution of the values in the real world and did not want to spoil our learning with assumptions we don't know are correct.
We didn't consider validation sets, because for the Kaggle data its role is taken by the test sets in the competition. For the others, there is also no point of doing it, since we don't have any comparison to other groups. If we used the results of the validation set for comparison of our methods, then again, the training didn't take place independently of this set. Hence, we considered it a waste of samples. Especially for small datasets it is preferable to have bigger training data instead of validation.

As stated above, the roles of the classes in amazon and congress are interchangeable. Thus, it suffices to consider accuracy. To be thorough, we also looked at macro precision and macro recall. The situation for wine and mushroom is different. For the mushroom dataset, it is more important to avoid poisonous samples to be classified edible, than vice-versa. Consequently, the recall of 'poisonous' and the precision of 'edible' should be maximized. note, that this is not done here. The reason is the excellent performance of our algorithms (accuracy of 1!).
For the wine, there was another situation. Here it makes a difference, whether a class 4 wine is misclassified as 5 or as 9. But again, we have accuracy equal 1, thus, it wasn't relevant to check this behavior separately.

PRE-PROCESSING

In terms of pre-processing, we both used the standard methods, like encoding categorical data (One hot, Ordinal), scaling (minmax, z-score) and feature selection, and introduced our own pre-processing ideas and methods. Examples of them are a handmade ordinal encoding, handmade weighted scaling and a special approach for deciding which features to drop. The details are explained in the corresponding sections.

METHODS AND THEIR PARAMETERS

We chose the following methods.
The first one was k-Nearest Neighbors (kNN). We considered the parameters algorithm, k, and weights.
The second method was Support-Vector-Machines (SVM) with parameters kernel, coeff0, cash_size and gamma.
As the last one we took Random Forest and looked at min_samples_leaf, min_samples_split, max_features, n_estimators and n_jobs.

We chose those three, because we wanted different approaches, such as a rather simple one, one not depending on distances and one we did not have good intuition on. We expected good results of kNN for the natural language task amazon (spoiler: didn't work). For SVM we expected it to perform well for congress data (clustered). Random forest might perform best for "rule like" data, for which the other two may work poorly.

The runtime behaved differently on each dataset, as expected. For congress and wine, it took almost no time. Also, for mushrooms the runtime was neglectable (1.5 sec per split). The only dataset we encountered issues with was amazon. This didn't come as a surprise. Especially for random forest with many trees it took a long time (about 2 min/split).

METHODS

IN THIS SECTION WE GIVE INFORMATION ABOUT WHAT RESULTS WE ACHIEVED AND HOW WE HAVE DONE THAT

WINE

When analyzing the wine dataset, we found out that working with human intuition works well. Therefore, besides some standard machine learning methods, we also used our knowledge about wine to improve the results. According to the knowledge what we already had a priori, there seemed to be a strong connection between the target value quality and some of the chemical values. For example, a high acidity is just not wanted for red wines and therefor wine samples that had a high acidity, were rated poorly by the experts.

Since the dataset only consists of numerical values which unfortunately have a quite high variance in range, we focused our pre-processing on scaling those values. Feature selection was not needed in case of the wine dataset since there are only eleven. Like in the last exercise we used four different pre-processing approaches:

1) Standard dataset without pre-processing
2) Z score scaling
3) Min max scaling
4) Self-made weighted scaling method

The numbers one to four can also be seen in the result matrices. We didn't want to drop any feature, since the result of all three methods gradually benefited from a higher number of features. We could not drop any feature but knew, that some values in features are just not desirable. Therefore we developed a scaling method based on the feature correlation. For each feature, we divided the feature by their maximum, to have a range between zero and one and then multiplied by their corresponding correlation to get a somehow weighted scaling

$$\boldsymbol{If\ correlation < 0}: \quad \frac{max(feature) - feature}{max(feature) * |correlation|} \quad , \quad \boldsymbol{If\ correlation > 0}: \quad \frac{feature}{max(feature) * |correlation|}.$$

This pre-processing method outperformed all others by far and yielded the best result for all algorithms.

As mentioned in the beginning of this section, the feature values give a good idea of which class the wine will be classified as. This will be seen in the following results. Using the just descripted scaling method, we achieved a perfect score of accuracy, precision and recall equal to one for two out of the three used methods.

The results of kNN were the "worst" of all three methods. But it must be made clear, that worst in this case means, not a perfect score! When using five-fold cross validation. kNN had an impressive mean accuracy of 0.999.

The best parameters of kNN were k=1, and uniform weights. Choosing k < 5 did not have a great impact on the results, bigger k's lead to a much worse result. Below one can see the results for accuracy, precision and recall using our hand-made pre-processing method.

```
evaluation for 4, test_accuracy:    [0.997 1.    1.    1.    1.    0.999]
evaluation for 4, test_precision:   [0.999 1.    1.    1.    1.    1.   ]
evaluation for 4, test_recall:      [0.985 1.    1.    1.    1.    0.997]
```

Note on the figure:
We used cross-validation with 5 splits. Thus, the first 5 columns are the results of the respective split and the last one is the mean.
(kNN with parameters: k=1, weights=uniform, algorithm=auto, using weighted scaling as pre-processing.)

Using Support Vector Machines, the results were better than kNN. When using the weighted scaling approach with SVM we could achieve a perfect score of one for each split and all three scores. SVM achieved the best results using a polynomial kernel and polynomial degree of four. The best pre-processing was again the weighted scaling approach. Min Max scaling had quite impressive results as well, with 3 splits being one and split one and four having accuracy of over 0.99. Z score scaling could not compete at all (mean accuracy = 0.79)

```
evaluation for 4, test_accuracy:    [1. 1. 1. 1. 1. 1.]
evaluation for 4, test_precision:   [1. 1. 1. 1. 1. 1.]
evaluation for 4, test_recall:      [1. 1. 1. 1. 1. 1.]
```

Note on figure: (SVM with parameters: gamma=scale, kernel=poly and degree=4, using weighted scaling as pre-processing.)

The method Random Forest yielded perfect results as well. Opposed to the other two methods the fourth pre-processing did not yield the best result. Interestingly enough the unprocessed dataset has a perfect score. The other three pre-processing methods only achieved a perfect score in four out of five splits. The results of the pre-processed set were nearly identically good. (accuracy 0.99, precision 0, 99 and recall 0.99). We tried out a lot of parameters to find the optimal ones. The parameters max depth, min_sample_split and min_sample_leaf did not improve the results, so that finally only n_estimators and max_features, were used.

```
evaluation for 1, test_accuracy:    [1. 1. 1. 1. 1. 1.]
evaluation for 1, test_precision:   [1. 1. 1. 1. 1. 1.]
evaluation for 1, test_recall:      [1. 1. 1. 1. 1. 1.]
```

Note on figure: (Random Forest with parameter: max_features=6, n_estimators=100, not processed dataset)

To conclude, it seemed that like the mushroom dataset, also the wine dataset is quite easy to predict. As stated on the beginning we assume that this is due to the strong connection of some features with the target value. When fine tuning the parameters and using the best fitting pre-processing method, all three machine learning methods could achieve an impressive score. kNN was the only method not having a perfect score.

MUSHROOMS

Since we are dealing with strings only, we distinguished three approaches. Our first approach #1 was OneHot Encoding. Since we have >8000 samples with 32 attributes, some taking up to 7 different values our resulting dataset is of dimension 8124 x 116. Our next approach #2 was to mix Ordinal and OneHot Encoding. The idea was to ordinal encode all attributes that only come with 2 distinct values with 0 and 1 and OneHot Encode all attributes that have more values. We did this rather than ordinal encode everything, because it is difficult to rank some attributes. For example 'ring-type' has 5 different possible values: {p,e,l,f,n} where 'f' and 'n' only make up <= 1% of total occurrences each. Since we didn´t have expert information on how or whether these values are connected, we decided to OneHot encode all of them. This led to an 8124 x 111 dataset.

For kNN it turned out, that the best results are obtained for k=1, weights=distance. Larger k provided worse results.

```
evaluation for 1, test_accuracy:    [1. 1. 1. 1. 1. 1.]
evaluation for 1, test_precision:   [1. 1. 1. 1. 1. 1.]
evaluation for 1, test_recall:      [1. 1. 1. 1. 1. 1.]

evaluation for 2, test_accuracy:    [1. 1. 1. 1. 1. 1.]
evaluation for 2, test_precision:   [1. 1. 1. 1. 1. 1.]
evaluation for 2, test_recall:      [1. 1. 1. 1. 1. 1.]
```

This dataset is almost 'too easy' to classify. All methods are able to classify perfectly, after the right choice of parameters. We made sure that the results are correct, i.e. we didn´t train on the test set.

Using SVM with gamma = 'scale' and kernel=rbf we obtain perfect results as well.

Random Forests with n_estimators = 100 can also classify perfectly.

It appears that the classification task on this dataset is very easy, all methods were able to predict all test data perfectly. If shuffling isn´t done at the beginning all methods fail to classify the first and last split completely correctly. There are discrepancies with the default order of the training set, after shuffling everything works out.

CONGRESS

As already stated we omit the column 'ID'. Preprocessing nominal data often requires heuristics, since there is not always a clear-cut way to encode strings to numbers. We considered a couple of different approaches.
The first approach #1 was to use OneHot Encoding, which is feasible, because we only have 16 attributes to encode, each having 3 possible values, meaning we don´t get space issues with the resulting matrix.
#2 being Ordinal encoding, where we decided to encode 'n' - meaning no or against – with the value 0, 'y' - meaning yes – to the value 1 and 'unknown' with the value 0.5, since 'unknown' could mean that the person is neither for or against that law, therefore it makes sense to have equal distance to yes and to no.
Additionally, we tested feature selection #3. First, we encoded our dataset, this time encoding 'class' as well and then we built the correlation matrix. We considered the attributes with the absolute lowest values, closest to zero, to be obsolete, which lead us to omit the attributes 'immigration' and 'water-project-cost-sharing' (both having a value < 0.06). Intuitively this didn´t seem right at first, since especially 'immigration' feels like it should have an impact on classification. But that might as well be our bias, because we consider our regional politics. Political views could very well differ strongly, depending on where you are from, so we chose to trust the data and omit these two values.
For #4 we also considered scaling using encoding #2, by giving attributes with higher absolute correlation, bigger weights and attributes with low correlation smaller weights. This turned out to be about equally good as #3.

We managed to get good results with all three methods, SVM being the best one overall, which we also used for our Kaggle submission.

For k- Nearest Neighbors refer to the graphic below to see our test results.

```
evaluation for 4, test_accuracy:    [0.956 0.955 0.977 0.953 1.    0.968]
evaluation for 4, test_precision:   [0.967 0.944 0.971 0.944 1.    0.965]
evaluation for 4, test_recall:      [0.941 0.964 0.981 0.963 1.    0.97 ]
```

As we can see #4, ordinal encoding + scaling, has the overall best scores for kNN, with k = 5 and weights = 'distance'. Bigger k's decreased accuracy. This is already a solid result, with an accuracy of 96.8%, but we managed to improve it.

Considering Random Forests, we used n_estimators=100, which means a hundred generated random trees.

```
evaluation for 4, test_accuracy:    [0.978 0.932 0.953 0.953 1.    0.963]
evaluation for 4, test_precision:   [0.983 0.921 0.95  0.944 1.    0.96 ]
evaluation for 4, test_recall:      [0.971 0.946 0.95  0.963 1.    0.966]
```

The result is again very good, an accuracy of 96.3% with preprocessing method #4, but it is slightly worse than kNN. Using random forests, we were able to obtain better results from time to time, but in the average case random forests were not the best performing option.

In the case of Support Vector Machine, we found a setting for SVM, after time consuming parameter tests, that yielded our best result and which we also used to produce our Kaggle submission.

```
evaluation for 3, test_accuracy:    [0.933 1.    0.977 0.977 0.977 0.973]
evaluation for 3, test_precision:   [0.926 1.    0.982 0.971 0.971 0.97 ]
evaluation for 3, test_recall:      [0.935 1.    0.969 0.981 0.981 0.973]
```

As you can see, we obtain the best accuracy so far with 97.3% and our other scores are accordingly high. The parameters used were kernel='poly', coef0 = 10 and gamma='auto'. The only question that remains is whether to choose preprocessing type #3 or #4, for stability reasons we went with #3 here.

For all our methods preprocessing #3 and #4 gave the best possible results. All methods were able to yield good results after fine-tuning the parameters. Nevertheless one should note that #3 was constructed by dropping certain features that were below an 'arbitrary' value for correlation, 0.06 in this case. Still it was always one of, if not the best preprocessing typ. This dataset was small enough that all tests were computed in no time at all, even OneHot Encoding didn´t pose a computational problem. Which method to choose here largely depends on heuristics, one could also choose a worse performing one, as to avoid overfitting. But since the score is very good for each split done by cross validating, it seemed logical to go with the highest scoring one.

AMAZON

The crucial point of finding a good strategy for the Amazon dataset is handling the big number of attributes as well as possible. Also, working with human intuition does not really work. There is no point in looking through 10 000 columns and hoping to find something interesting.

However, we came up with some interesting pre-processing methods. We tried the standard approaches such as min-max scaling, z-score scaling and feature selection. Generally, they improved the performance of the methods, but apart from feature selection there were no major differences. The non-standard ways we tried were scaling using text frequency – inverse document frequency (TF-IDF) and perform a kind of 'Ordinal Encoding'. The 'ordinal encoding' is based on the following thought. Intuitively, an author stands out of others by

using specific expressions or words. That does not really depend on how often but whether those expressions are used. Hence, what we did was convert the dataset into a Boolean one. I.e. we transformed every value not equal zero into one and left the zeros how they were. This encoding led to a slight increase of accuracy for kNN and SVM.

TF-IDF is a pre-processing method coming from natural language processing. In this area it is often used for kNN. This is the reason we wanted to combine them in the amazon dataset.
TF-IDF is a specific value for each word/expression W. At first, the number of reviews n, in which W is used is computed. Then the TF-IDF value for W is log(N/(n+1)), where N is the number of reviews in the training data. Subsequently, the corresponding column is multiplied by this value. It results in upscaling words that are rare and downscaling very common words. Our intention was to increase the importance of expressions and words that are not used by every author. Since this approach did not work at all, we will not consider it in the following discussions. What probably happened, is that the words describing product names or properties get a very high scaling factor and biased the database drastically.
For every algorithm, the pre-processing that achieved the best result was feature selection.
Note: We came up with this feature selection strategy after realizing that TF-IDF was not working. Hence the ideas are somewhat controversy to TF-IDF.
We proceeded the following way. At first, we computed n as defined in tfidf. Then we considered the relative frequency n/N (N as in TF-IDF). The idea was to drop the most and the least frequent attributes. The argument of dropping the most frequent attributes is still the same as downscaling them within TF-IDF. (Must be words like 'and','the','I' etc.) After seeing that TF-IDF performed this poorly, we thought that the very rare words may be very technical terms only used for describing a special kind of product.
In a second step we had to find the best values for the lower and the upper bounds.

For the kNN method the results were very bad. No matter what pre-processing we used, we never reached a higher accuracy than 0.3. The 'best' value for k was one, even though changing it did not have much impact (as long as didn't get too big). Same for the other parameters (weights, algorithm). The best pre-processing for kNN was feature selection with a lower bound of 1.5% and an upper bound of 65%. Due to the low accuracy (and precision and recall), we did not spend much time on doing tests for kNN.

```
evalution for 1, test_accuracy:     [0.289 0.302 0.302 0.312 0.267 0.294]
evalution for 1, test_precision:    [0.349 0.27  0.339 0.323 0.28  0.312]
evalution for 1, test_recall:       [0.296 0.302 0.305 0.3   0.265 0.293]
```

Note on the figure:
The parameters used are k=1, weights=distance, algorithm=auto.

Using SVM, the results have been way better than using kNN, but still weren't good.
Same as for kNN, feature selection turned out to be the best pre-processing method.

Going further and using scaling (z-score, minmax) or our Boolean encoding did not lead to significant improvements. The linear kernel turned out to perform best. For linear kernel, there are not many other parameters to adjust.

```
evalution for 1, test_accuracy:     [0.445 0.522 0.597 0.572 0.466 0.521]
evalution for 1, test_precision:    [0.507 0.533 0.642 0.59  0.502 0.555]
evalution for 1, test_recall:       [0.445 0.505 0.585 0.575 0.47  0.516]
```

Note on the figure: SVM run with linear kernel, feature selection with lower bound 1% and upper bound 65%.

Finally, the random forest method resulted in satisfactory results. Using feature selection with 2% and 65% limits, min_samples_leaf=3, max_features=1000, n_estimators=700 we achieved mean accuracy of 0.67. The results weren't very sensitive on max_features and n_estimators but led to best results in the range between 900-1150, 680-730 respectively.

```
evalution for 2, test_accuracy:     [0.665 0.711 0.658 0.652 0.679 0.673]
evalution for 2, test_precision:    [0.694 0.73  0.732 0.656 0.67  0.696]
evalution for 2, test_recall:       [0.664 0.698 0.652 0.635 0.653 0.661]
```

Note on figure: Random forest with feature selection (2% and 65% limits), min_samples_leaf=3, max_features=1000, n_estimators=700.

Since the result is so definite, there is not much to conclude about.
The most suitable algorithm for the amazon dataset is random forest. This may seem surprising at first glance, since kNN is said to perform well in natural language tasks. But at second glance, it is reasonable for kNN to struggle with 10 000 attributes and with the fact that there are only at most 21 samples in the same class. This makes kNN unstable in this case.

CONCLUSION

CONSIDERING ALL THE RESULTS AND INFORMATION WE GAINED DURING THIS EXERCISE WE GIVE A FINAL CONCLUSION AND COMPARE THE DIFFERENT METHODS AGAINST EACH OTHER.

We managed to produce very good classification predictions for every dataset, but Amazon. The Mushroom dataset is very easy to predict and every preprocessing option turned out to be equally good. The Wine dataset only produced such high results after using our weighted scaling preprocessing method. With built in scaling methods we only achieved a score of about 93% accuracy and recall and precision being much worse. Congress has above 90%

accuracy, recall and precision for every preprocessing option, but by choosing SVM with a polynomial kernel and additional parameters we managed to locally maximize all three scores. Our custom scaling preprocessing and feature selection performed equally well here.
The Amazon dataset is the only one that proved problematic. The best result was obtained with Random forests, using feature selection. Unfortunately, our custom-built preprocessing didn't perform as well.

Best result	**kNN**	**Random Forest**	**SVM**
Wine		perfectly	perfectly
Congress			97% accuracy
Mushroom	perfectly	perfectly	perfectly
Amazon		67% accuracy	

Perfectly means 100% for all considered scores.

In the above matrix one can now determine what method worked best for which dataset. In all cases but Mushroom, preprocessing was very useful and significantly improved our results.
We conclude that Support Vector Machines and Random Forests outperformed kNN in all of our datasets. It appears that kNN has troubles dealing with many classes, which fits our intuition. Also, kNN works well on clustered data, which often can be separated with SVM as well, given the right kernel.
For heterogeneous data Random Forests provides good results, because it constructs rules rather than distance relations/majority votes.
The Amazon dataset was the hardest to produce decent results for, that may mean that the data is not expressive enough. A possible reason is that the authors used a rather English, which makes it hard to distinguish who wrote the review. Another idea would be to add some attributes, such as the area and price of the product, the time the review was written etc.

Generally speaking, some datasets profit of using human intuition and a priori knowledge.
Still, there exist cases where this leads to bias. E.g. us expecting the immigration topic to be a good indicator for someone's political orientation. In such cases it is preferable to verify intuition considering data and facts.
Further interpretations would be only educated guesses, since our results were either excellent or poor for all methods. (Amazon may be an exception.)
Overall, we noticed the results were more definite compared to the regression task.