MVE440/MSA220
Statistical learning for big data

Adam Wirehed

wirehed@student.chalmers.se

June 12, 2020

Contents

1 Exercise 3.2: Classification and variable selection **1**

2 Exercise 4.1: High-dimensional clustering **8**

3 Exercise 4.2: Improving variable selection in the Lasso **20**

References **24**

1 Exercise 3.2: Classification and variable selection

In this task we are suppose to analyze the given data with two different classification methods. As well as evaluate their performance and determine the best predictors.

Method

First of all I took a look at the data and observed that we are working with an unbalanced data set. The two classes are not equally represented in the training- or validation data. The amount of data corresponding to class 0 is 247 and 134 in the training- and validation data respectively. Compared to the corresponding 76 and 41 for class 1. This yield a ratio of 75-/25 % for each class in the data. However the ratio of representation is equal over the training- and validation data. To improve computational time the data is preprocessed using sci-kit learn's StandardScaler method [1]. The scaler it fitted with the training data only, but both the training and validation data is transformed. The transformation is simply subtracting the mean of the data divided by its standard deviation.

To explore the different feature importance in the data I use permutation importance for feature evaluation. Permutation importance of a feature is calculated as follows. First, a baseline metric, defined by scoring, is evaluated on a (potentially different) dataset defined by the X. Next, a feature column from the validation set is permuted and the metric is evaluated again. The permutation importance is defined to be the difference between the baseline metric and metric from permutating the feature column [2]. This will be done for both of the classifiers to observe if the methods prefer different features to each other. To evaluate if features with high importance metric are as relevant as the algorithm says, tests with a specific amount of the most important features will be done.

The classification methods will both be tested on the preprocessed data and transformed data using PCA. As with the preprocessing of the data, the PCA analysis is fitted only on the training data but the transformation is done both on training and validation data. The two classification methods tested is Lasso-regulated Logistic regression and Ridge regression. Both of the methods have their hyperparameters estimated using cross-validation with a fold split of five [1]. The logistic regression estimates its hyperparameter λ by testing values chosen from a logarithmic scale between 1e-4 and 1e4. Ridge regression chooses its hyperparameter α from a given vector of values togehter with cross-validation. By using a Lasso regularized Logistic regression the method is able to set certain parameters exactly to zero. Since our data is of the scenario $p >> n$ this could be a useful attribute. That is also the reason the algorithm was chosen for this problem. Ridge regression was chosen due to its attribute to mitigate the problem of multicollinearity in linear regression when we have a large amount of parameters. However it was also chosen for its inability to estimate coefficients as zero as this could be the reason behind the difference in performance between the methods.

For evaluating the two classification methods a validation set was given. Since the split of training and validation data was already done cross-validation will not be used for computing test scores. Instead multiple tests on the evaluation set will be done for both of the methods. The average of those tests will be used as the final test metric.

Results

Table 1: Mean accuracy score (n=10) on validation set for classification methods. Validation data used is the original validation set (preprocessed), PCA components, only using the two and 50 most important features (according to Lasso). The last column is a test using the 48 most important features after the two most important ones.

Method\Data	Original	PCA	2 feat.	50 feat.	48 feat.
Lasso-reg Logistic	0.8514	0.8000	0.7600	0.8617	0.7886
Ridge	0.8229	0.8229	0.7542	0.8229	0.8057

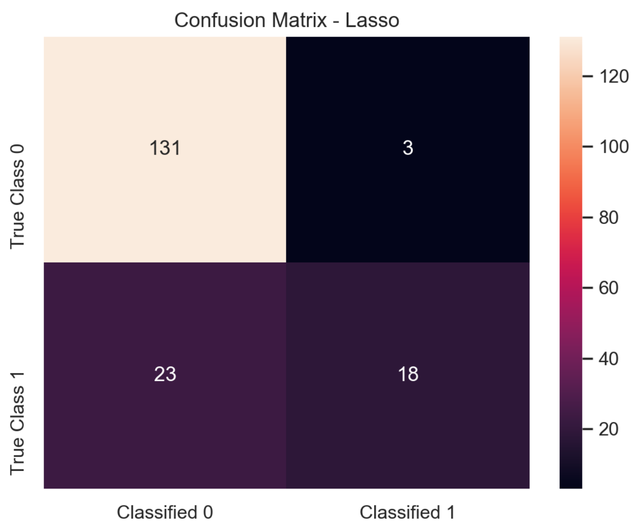

Figure 1: Confusion matrix for Lasso classifier

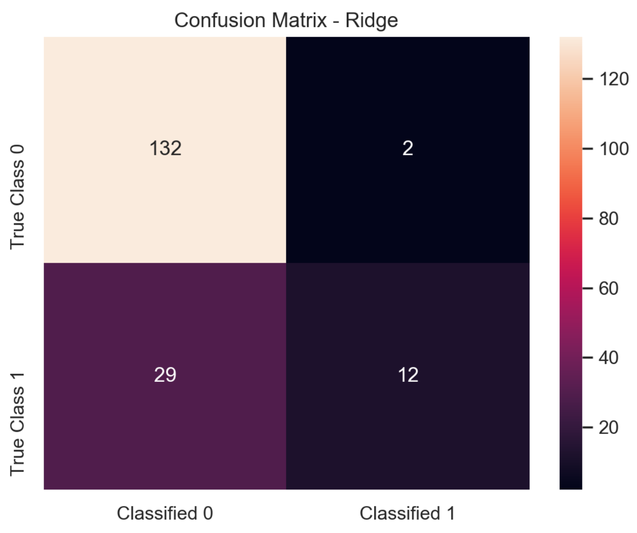

Figure 2: Confusion matrix for Ridge classifier

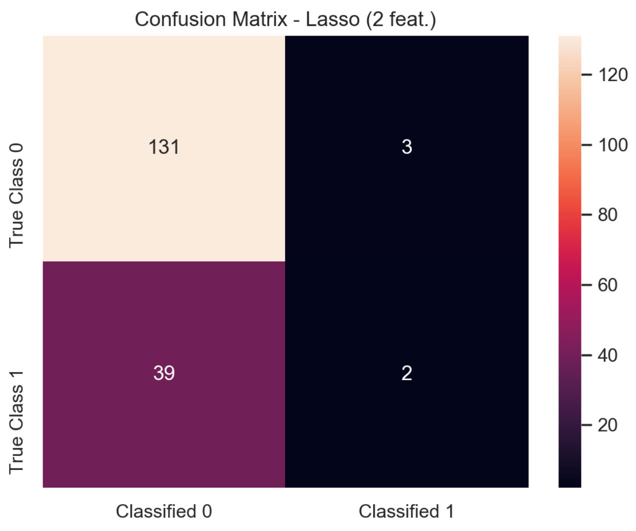

Figure 3: Confusion matrix for Lasso classifier only using the 2 most important features (according to Lasso)

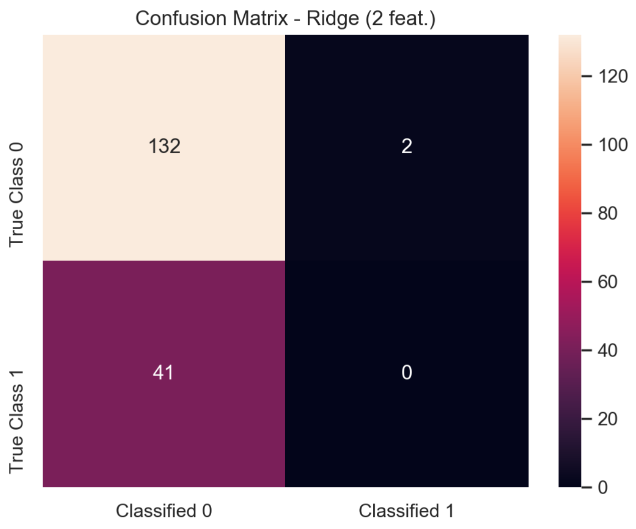

Figure 4: Confusion matrix for Ridge classifier only using the 2 most important features (according to Lasso)

Table 2: Estimated hyperparameter values for classification methods

Method\Data	Original	PCA	2 feat.	50 feat.	48 feat.
Lasso-reg Logistic (λ)	0.3594	0.0464	21.5443	21.5443	2.7826
Ridge (α)	500	1000	1	100.0	100.0

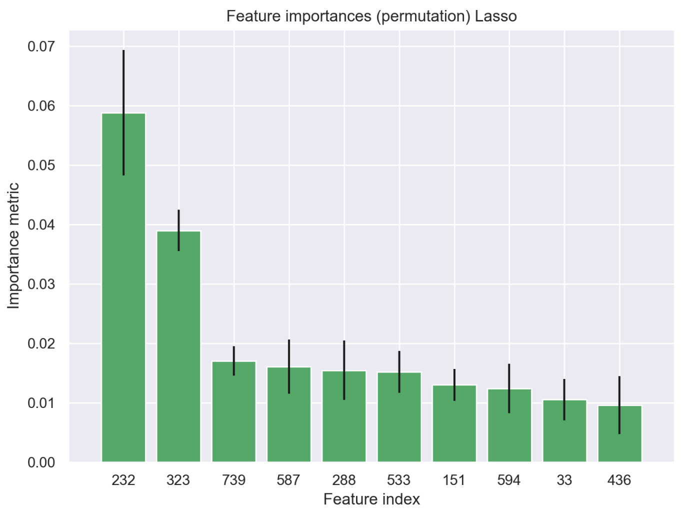

Figure 5: Feature importance metric using sci-kit learn's permutation_importance method on the Lasso classifier using the training data [1]. Green bar denotes feature importance value and black line corresponding standard deviation of the 10 most important features according to Lasso.

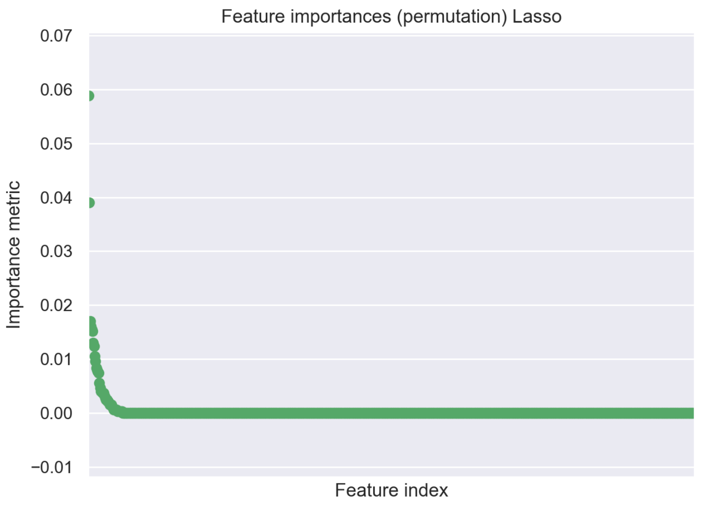

Figure 6: Feature importance metric using sci-kit learn's permutation_importance method on the Lasso classifier using the training data [1]. Y-axis indicates importance metric and x-axis the different feature index. Here all of the features and their importance metric is visualized. They are plotted in descending order of importance metric.

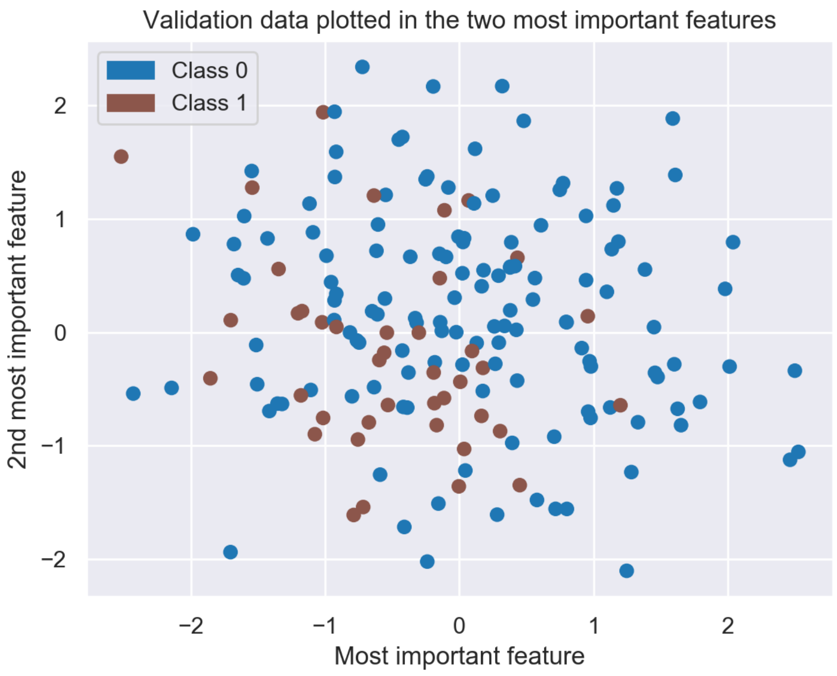

Figure 7: Data visaulized (with ground truth classes) in the two most important features (for Lasso) 232 and 323.

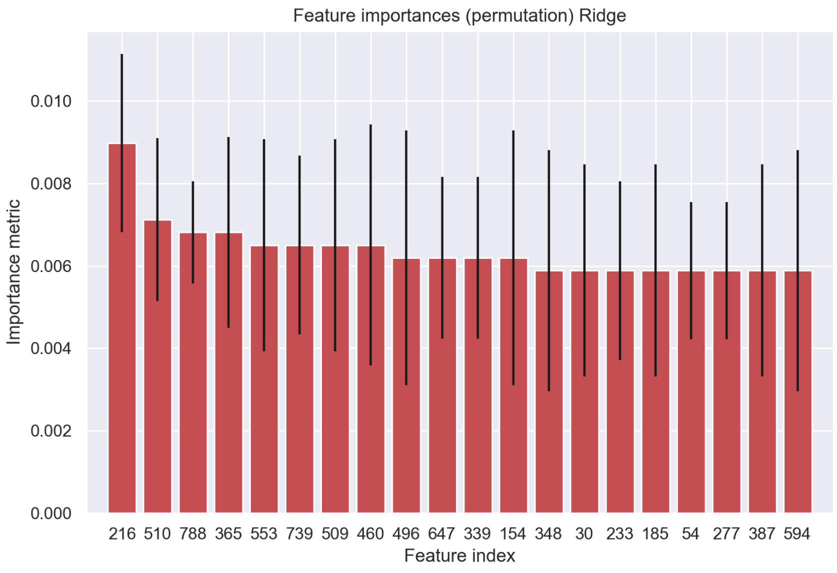

Figure 8: Feature importance metric using sci-kit learn's permutation_importance method on the Ridge classifier using the training data [1]. Red bar denotes feature importance value and black line corresponding standard deviation of the 20 most important features according to Lasso.

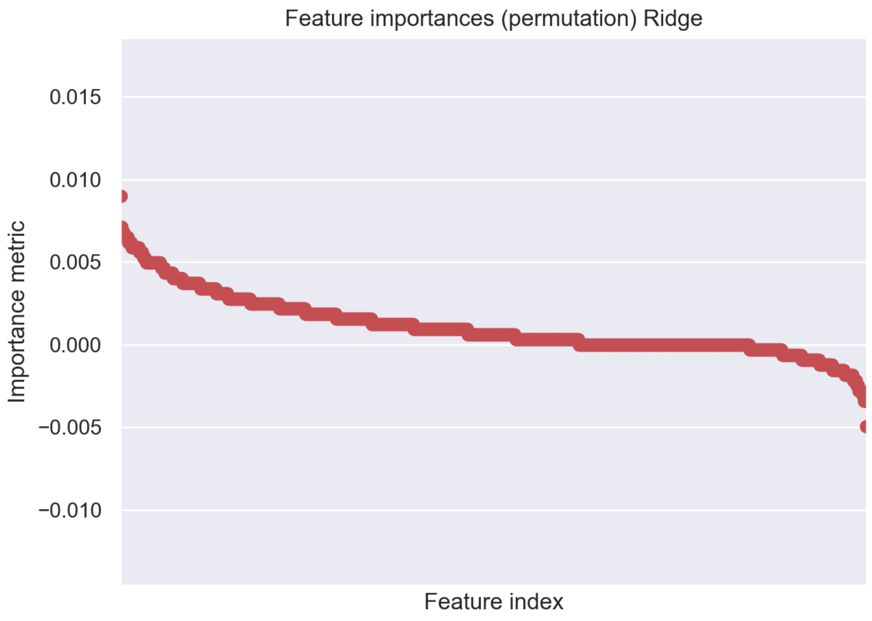

Figure 9: Feature importance metric using sci-kit learn's permutation_importance method on the Ridge classifier using the training data[1]. Y-axis indicates importance metric and x-axis the different feature index. Here all of the features and their importance metric is visualized. They are plotted in descending order of importance metric.

One further observation is that for Lasso-regulated Logistic regression, 704 out of 800 beta coefficients are estimated to be zero when using the original data. Indicating a sparse feature space The ridge regression is unable to estimate beta coefficients as exactly zero and this was as well confirmed in the code.

Discussion

By looking at table 1 we can see that the Lasso-regulated Logistic regression performed slightly better on the original data compared to Ridge. These accuracy values looks to be quite high, but since the data is unbalanced (75-/25 %) simply predicting every thing as class 0 would yield an accuracy around 0.75. And by these results around 0.85 and 0.82 our classifiers manage to extract some useful features to explain the data to obtain a higher accuracy then simply predicting everything as class 0. When we use the PCA components of the data the Lasso classifier decreases in performance. The reason behind this could be the reduction in features. Originally we have 800 features in our data. After PCA it reduces to same amount as our sample size (323). For the lasso method the new PCA dimensions might not contain enough information about the data to compared to the original. However this contradicts the results of the estimated beta coefficients. 704 out of 800 coefficients were estimated as zero for the Logistic method. We can also see that when using the 50 most important features we get similar results as when we are using all of the data (more about that later). So still having 323 features from PCA should maybe not affect the performance too much. There might be another reason behind the performance decrease when using PCA components.

The reason behind Logistic regression having better performance on this (original) data might be the models ability to estimate coefficients as exactly zero. As mentioned in the results, the Logistic method (Lasso regulated) estimated 704 out of 800 beta coefficients as zero. This indicates a sparse feature space and a low ratio of good explanatory variables.

Table 2 presents the different hyperparameter estimations from the cross-validation used in all the tests.

In Figure 5 we can see the ten features with the highest importance metric for the Lasso classifier. There are two features, 232 and 323, that seems to explain the data significantly better than the rest. By looking at figure 6 we can also see that these two features are significantly more important than the rest of the 800 features where the majority of them have importance metric 0. Which corresponds to the large amount of coefficients being estimated as zero in Lasso. There are a few other features that the Lasso method also views as important, but 232 and 323 have significantly higher values. Figure 7 visualizes the data in the feature dimension (232, 323). Here there could be a classification area with a boundary of a line going from the top left to the bottom right, where class 1 is in the bottom left area and class 0 in the top right area. But by looking at the confusion matrix for both Lasso and Ridge classifiers, figures 1 and 2, we see that a lot of the data for class 1 is mislabeled as class 0. Indicating that the diagonal line is further down to the bottom left to classify more of the data as class 0. Since the classes in the data is unbalanced it is not surprising to see that the classifiers prioritize class 0 for the decision boundaries.

Feature importance based on the Ridge classifier can be seen in figures 8 and 9. Comparing to Lasso we don't have a group of significantly more important features. In figure 8 we have high standard deviation for the importance metric implying a uncertainty in the metrics. Figure 9 shows the importance for all the features based on Ridge. Here we barley see any significant feature (maybe the left most one) and since the standard deviation is so high I won't draw any conclusions on feature importance from these results.

To test if these features are more important than others, both of the classification methods were also trained and tested with only important features. As we can see in Table 1, column 2 feat. which is the test using only feature 232 and 323, at first glance the performance is quite similar to the original data and with PCA. But since the data is unbalanced (75-/25 %) this accuracy is not very good. By looking at the confusion matrices in figures 3 and 4 we see that the classifiers is mostly predicting the data as class 0. So simply classifying everything as class 0 would yield a similar performance. Therefor only using these two features for our data does not give the classifiers enough information to obtain a similar score when using all the data. But the result using the 50 most important features (according to Lasso) we see a slight performance increase above the result using the original data for the Lasso method, and same as the original data for Ridge. But these accuracy mean values did vary around the same accuracy as for the original data for both the methods. This indicates that the chosen performance metric and corresponding computation method does indeed extract important features for explaining the data and its classes. Enough to obtain similar performance when using only the 50 using all of the data. In the last column in table 1 we see the results from the test using the 3-50th most important features (without features 232 and 323). Here we again see that without the two most significant the performance decreases, even when using a lot of important features. Implying that features 232 and 323 are essential for explaining the data, but they don't contain enough information themselves and need additional features (such as 3-50) to better explain the data.

For future research of the data, and the classifiers performance on it, the correlation between features could be explored. Since Ridge handles it much better than Lasso it would be interesting to see how much correlation is in the data, and how it affects the classifiers.

2 Exercise 4.1: High-dimensional clustering

The task for this project is to do an exploratory data analysis and find clusters in the data to be able to comprehend it better.

Method

The data was first preprocessed using sci-kit learn's StandardScaler method to improve computational time [1]. The transformation is simply subtracting the mean of the data divided by its standard deviation. To get a better understanding of the variance in the features values some features with high scaled values were visualized. Here I noticed some weird features that I will talk more about later, but since those discoveries affected the rest of the method it's important to bring that up. After scaling the data its PCA components were computed down from 728 to 50 dimensions for use in t-SNE for visualization in 2D of the data. The PCA components is used both for the clustering later and to improve the result from t-SNE. Due to the information gathered by looking at certain features, an outlier detection algorithm were used to create a new dataset for comparisons to see the implications of the outliers. The algoritmed used is the Isolation forest algorithm from sci-kit learn [1]. The IsolationForest 'isolates' observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. After this the average silhouette score, based on euclidean distance, for the three different datasets were computed (original, PCA and original without outliers). The silhouette value is a measure of how similar an object is to its own cluster. A dendogram based on hierarchical clustering, using Ward linkage, was also computed to compare to the silhouette score results. Based on the results from the silhouette score (and hierarchical clustering) the KMeans algorithm with different amount of clusters were applied to the original data and the one without outliers. The reason why the PCA components were not used will be brought up later. The clustering from KMeans will be visualized in the t-SNE dimensions. These clusters will then be used for the data as its new class labels when computing the data's five most important features (with respect to the clustering made by KMeans).

The feature importance was computed using forest of trees. In these decision trees every node is a condition of how to split values in a single feature so that similar values of the dependent variable end up in the same set after the split. The condition is based on impurity, which in case problems is information gain (entropy), while for regression trees its variance. So when I fit the data to the forest and the trees we compute how much each feature contributes to decreasing the weighted impurity. This is what feature_importances_ in scikit-learn does. The pros of this method is that the computation is fast, but it is an biased approach and has tendencies to inflate the importance of continuous features or high-cardinally categorical variables [3].

Results

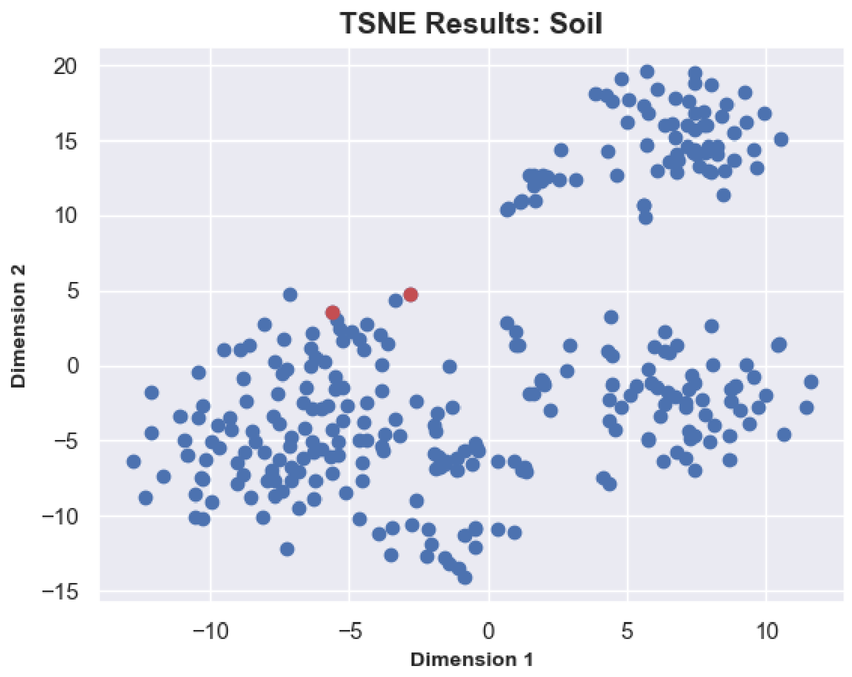

Figure 10: Data visualized in the two t-SNE dimensions with the outliers in red

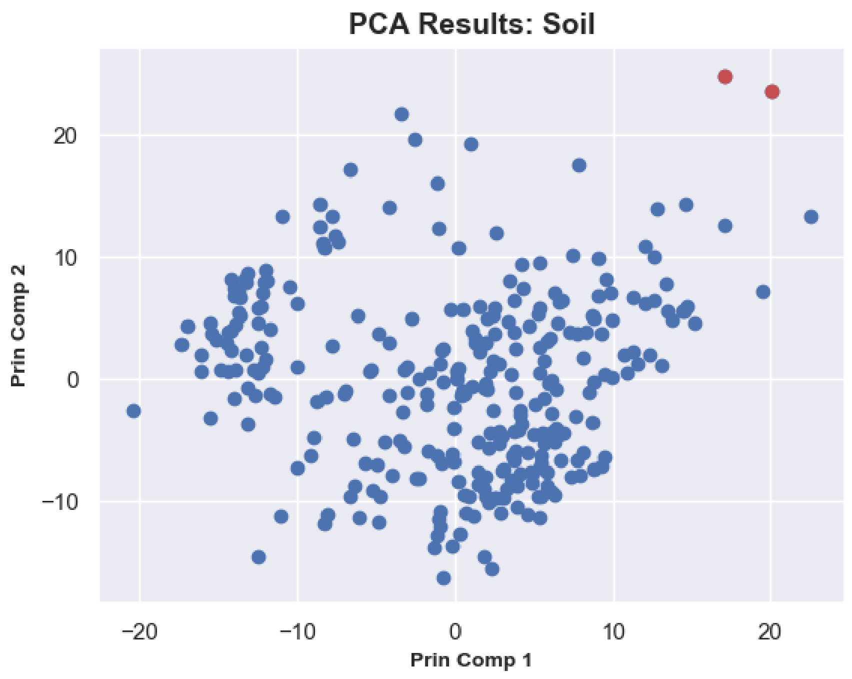

Figure 11: Data visualized in the two best PCA dimensions with the outliers in red

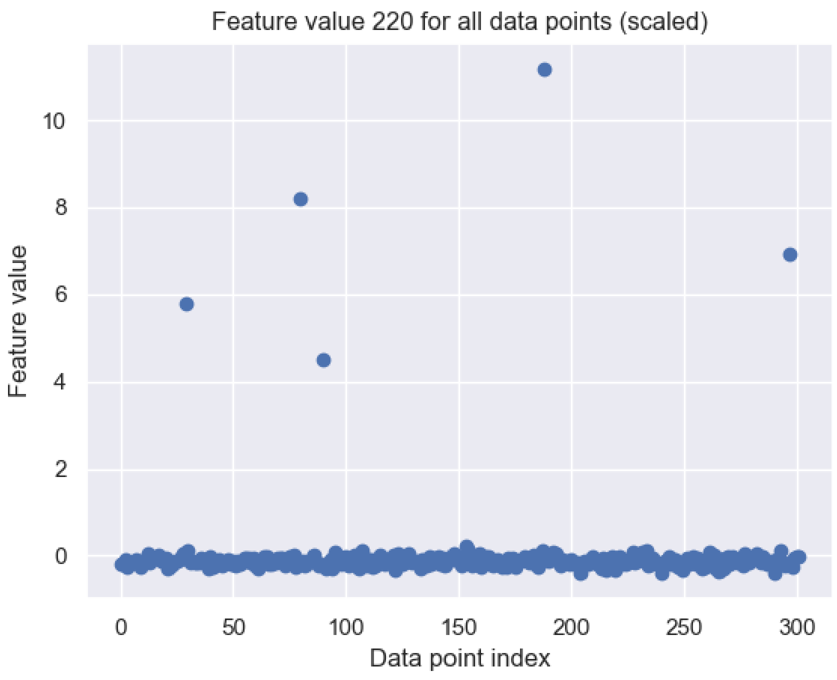

Figure 12: Feature 220 and its value (scaled) for all 302 data points

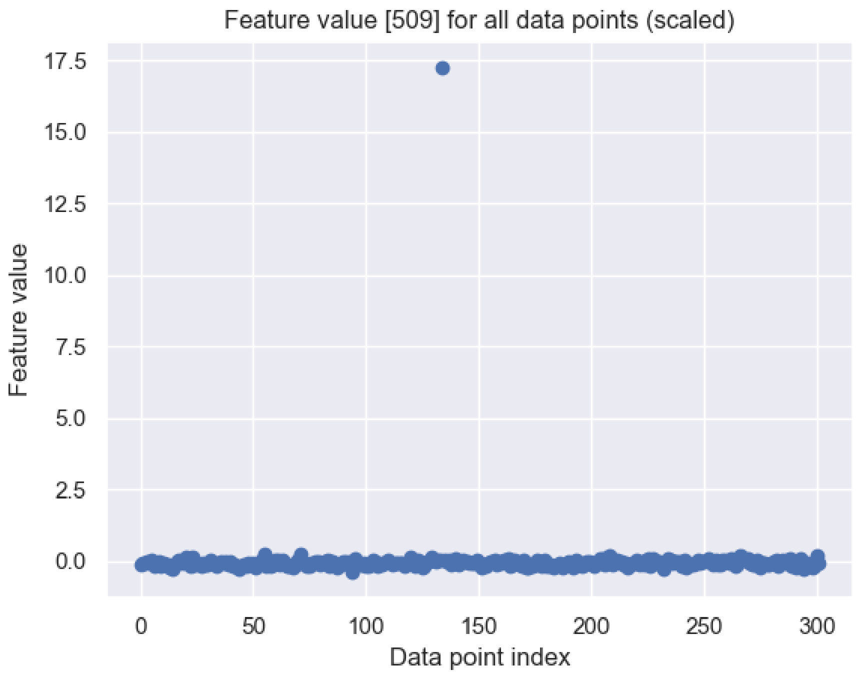

Figure 13: Feature 509 and its value (scaled) for all 302 data points

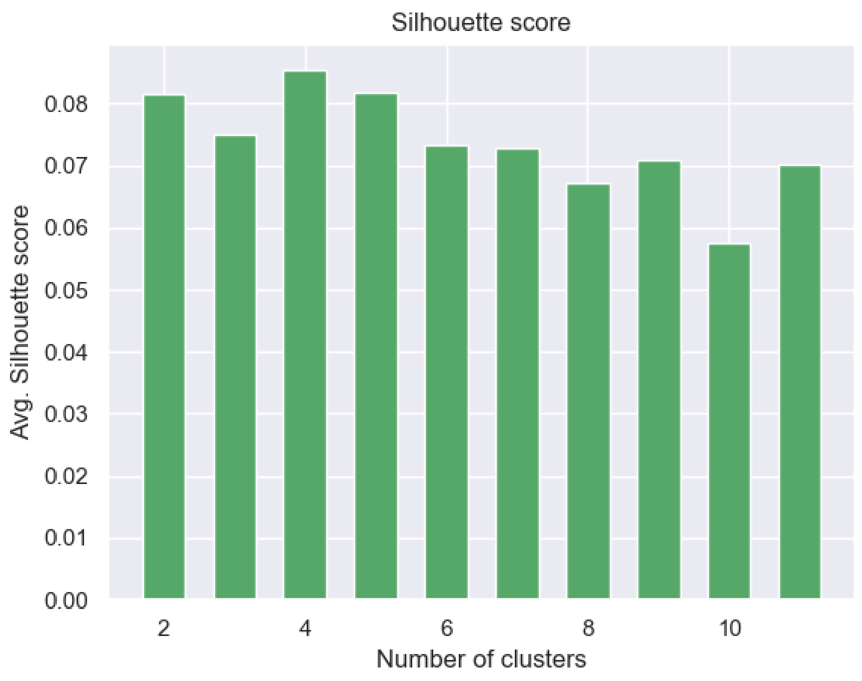

Figure 14: Silhouette score for different amount of clusters on the original (scaled) data

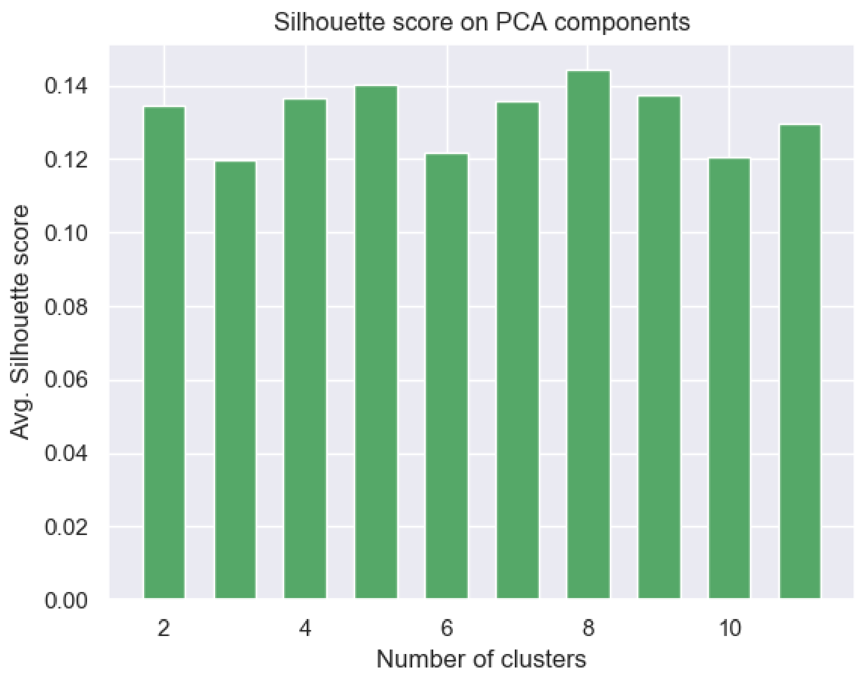

Figure 15: Silhouette score for different amount of clusters on PCA components (n$_c omponents = 50$)

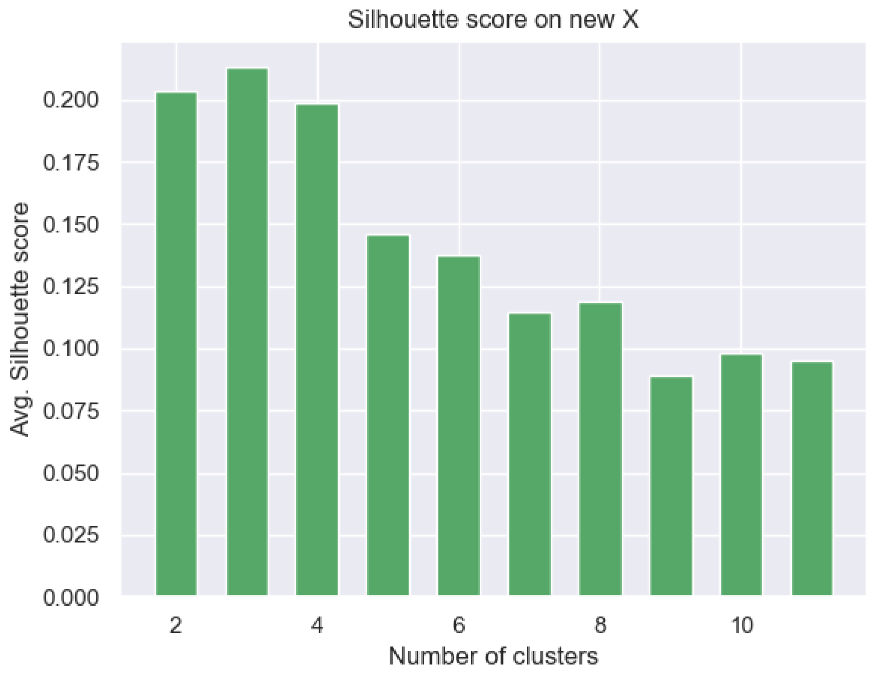

Figure 16: Silhouette score for different amount of clusters on the data without outliers

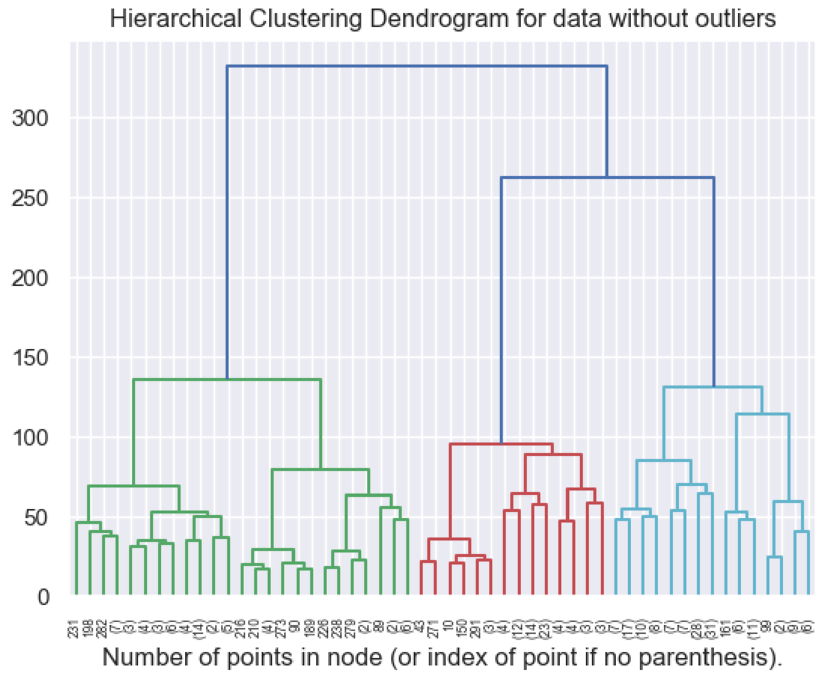

Figure 17: Dendogram of the hierarchical clustering (Ward linkage)

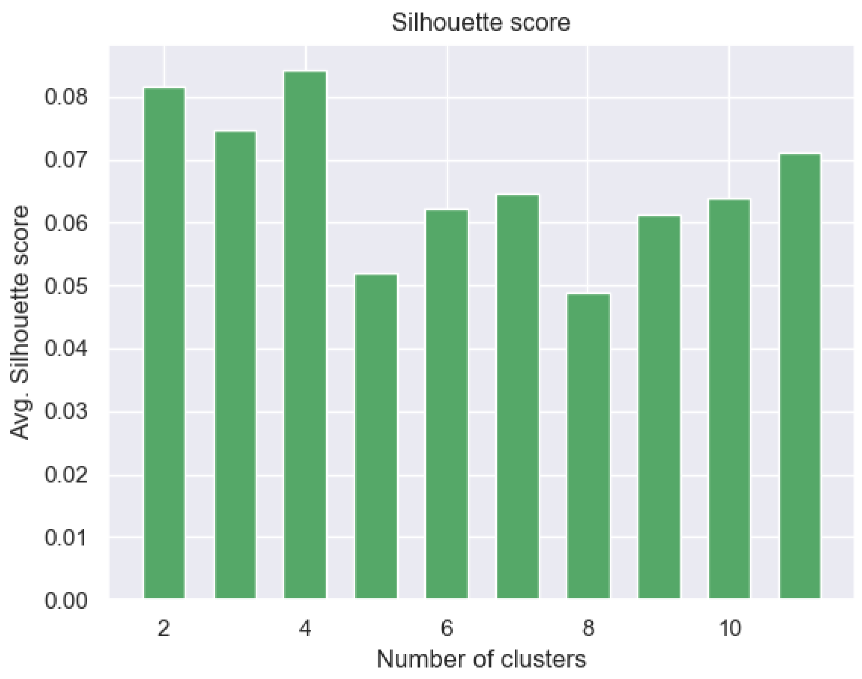

Figure 18: Silhouette score for different amount of clusters on the original (scaled) data. This is an example of a different result on the original data showcasing the variance in the result from time to time.

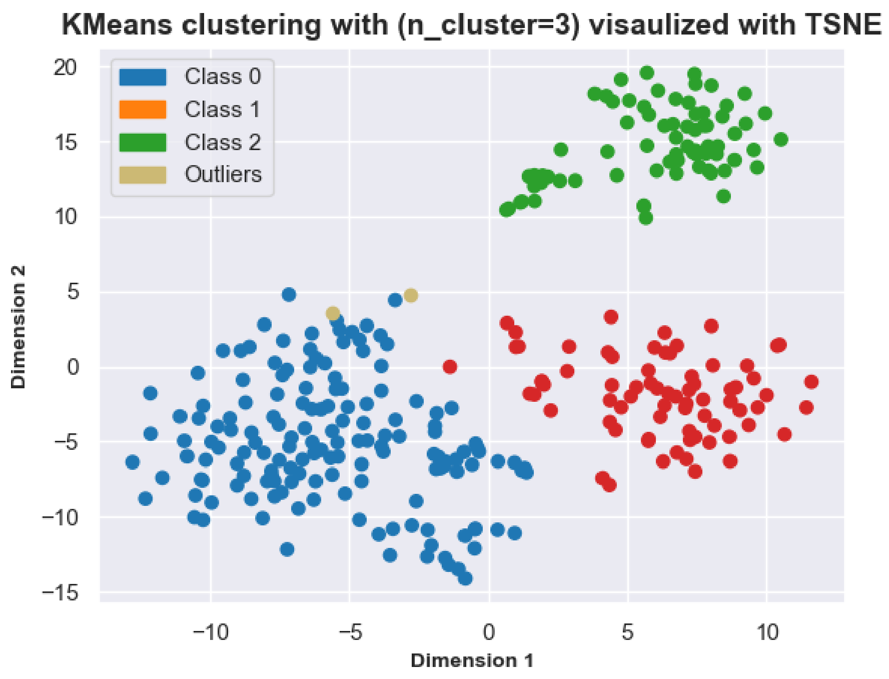

Figure 19: Clustering result from KMeans with 3 clusters (outliers in gold). Data visualized in t-SNE dimensions

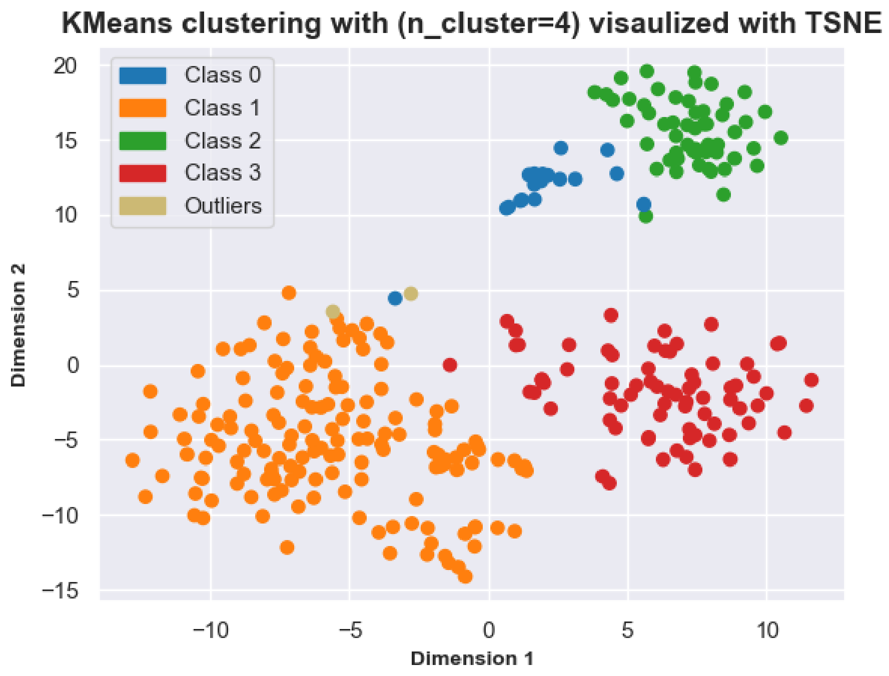

Figure 20: Clustering result from KMeans with 4 clusters (outliers in gold). Data visualized in t-SNE dimensions

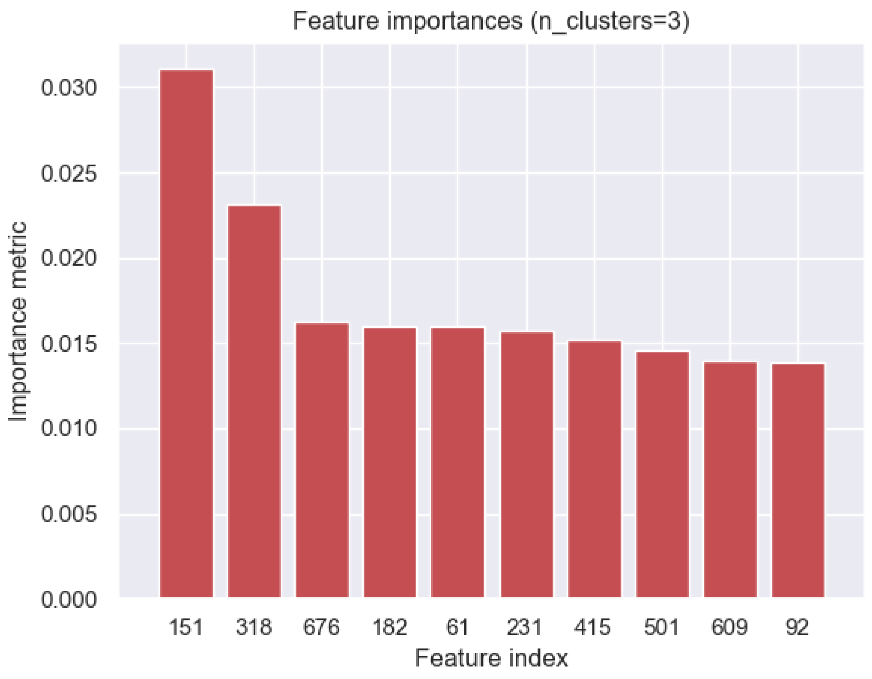

Figure 21: 10 most important features with n_cluster=3. Computed by forest of trees.

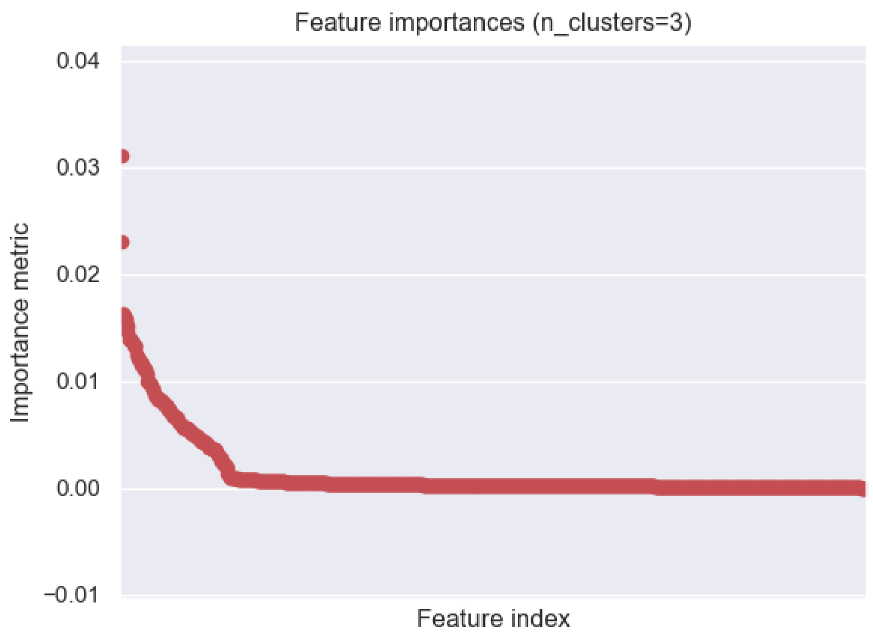

Figure 22: All features and their importance metric with n_cluster=3. Plotted in descending order. Computed by forest of trees.

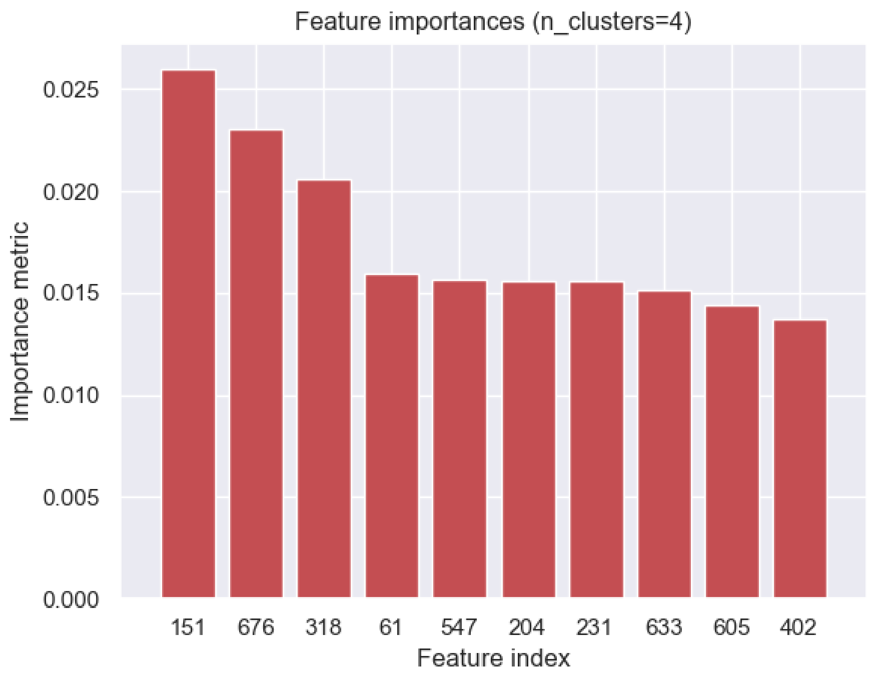

Figure 23: 10 most important features with n_cluster=4. Computed by forest of trees.

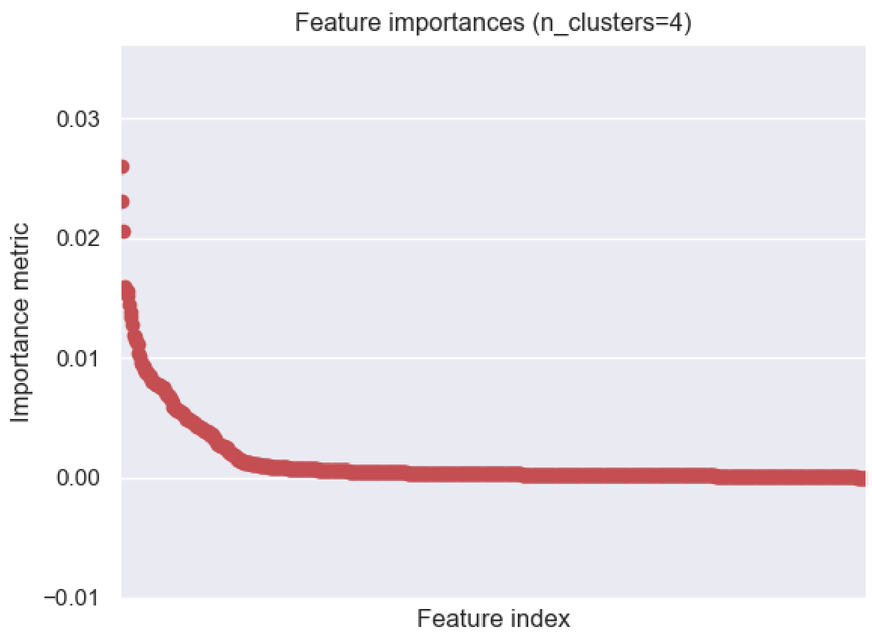

Figure 24: All features and their importance metric with n_cluster=4. Plotted in descending order. Computed by forest of trees.

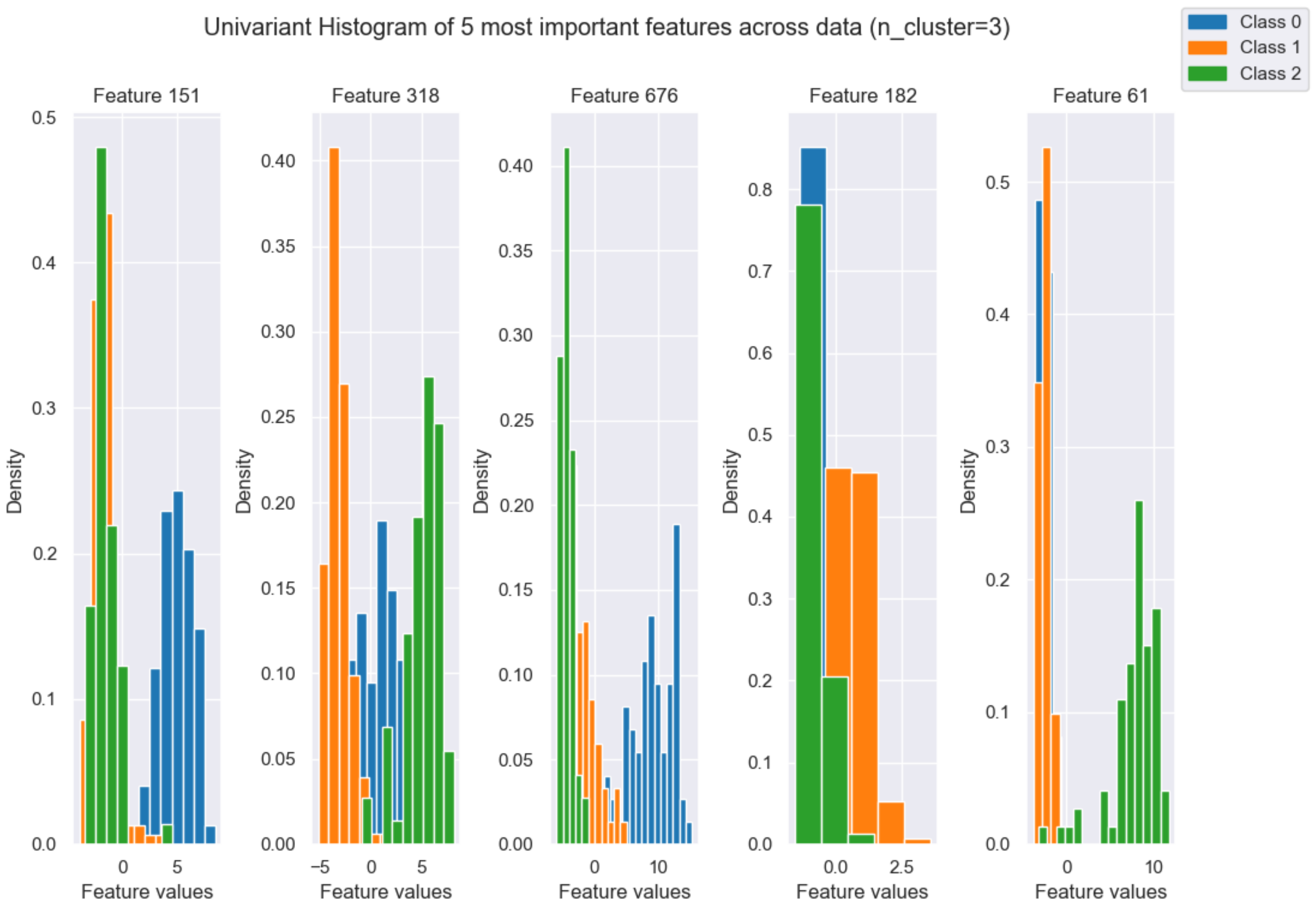

Figure 25: Univariant histogram of the five most important features based on forest of trees with n_cluster=3. Each of the classes/clusters from KMeans have its own histogram within each feature plot to visualize the difference in values for each class

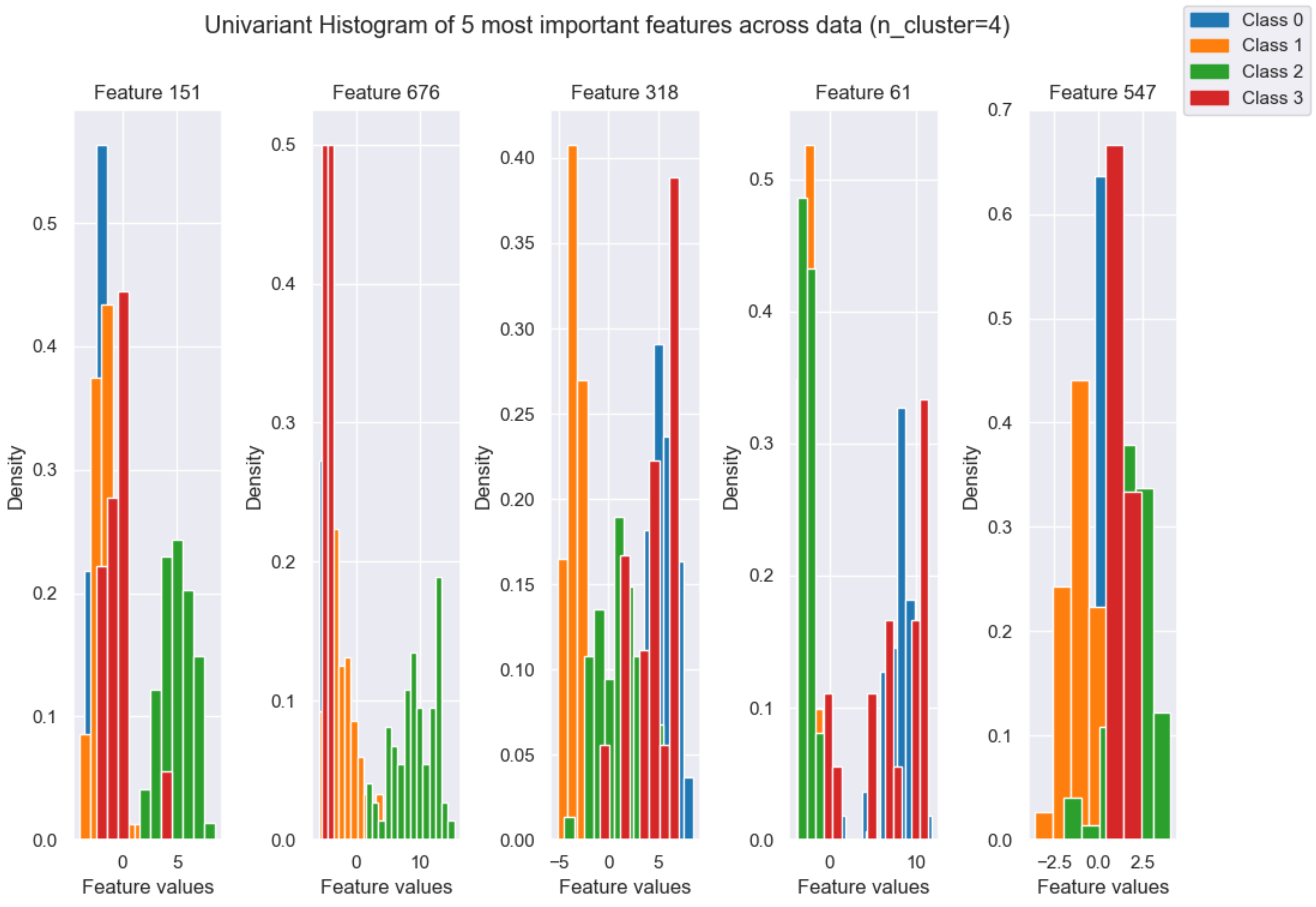

Figure 26: Univariant histogram of the five most important features based on forest of trees with n_cluster=4. Each of the classes/clusters from KMeans have its own histogram within each feature plot to visualize the difference in values for each class. In feature 676 you can see a small blue line to the left of the red. Indicating that the blue histogram (class 0) is behind the red and orange histogram. And by cross-checking with figures 19 and 25 you can confirm that.

Discussion

The initial data exploration can be seen in figures 12 and 13. Here we see that these two features have one or more outliers that could make it difficult for the clustering algorithms to estimate the amount of clusters. All the feature data is distributed around 0 with a few data points really far away. To compare these possible outliers affect on the clustering a new dataset was created without the outliers. For the results presented here the outliers were only two data points but during testing it could be between 1-6. The amount of outliers removed did however not impact the rest of the results.

The data is visualized in the t-SNE dimensions in figure 10 and in the two best PCA components in figure 11 with the two outliers in red. The silhouette score for the original data, PCA and the new dataset (without outliers) can be seen in figures 14, 15 and 16 respectively. If we start by comparing the silhouette scores for the original and new data (fig. 14, 16) by only removing two data points we get clearer results with as good drop at four clusters in the new data. For the original data we did not really have a clear drop at a certain amount of clusters, but the results here did vary from time to time. An example of that can be seen in figure 18 where the result is similar to the one on the new data. Unlike the new data, these results are not as common implying the effect as little as two outliers have on the silhouette algorithm. By looking at the silhouette scores on the PCA data we get no good estimation of the amount of clusters. It seems that To compare these results to another method a hierarchical clustering (Agglomerative) algorithm was applied to the new data. Its corresponding dendogram is visualized in figure 17. Here we clearly see that this algorithm is finding three distinctive clusters, one less than the silhouette score is presenting. Since three clusters do have the highest silhouette score, but not a clear drop after it, moving forward we are both testing the clustering for three and four clusters. The drop from three to four clusters are in the silhouette is so small that four clusters is still relevant.

The result from the KMeans clustering can be seen in figures 19 and 20. Even though t-SNE is a nonlinear dimension reduction for visualization purposes the result from the silhouette score and the dendogram indicating 3-4 clusters seems reasonable even in the t-SNE dimensions. These results all together indicate that an appropriate amount of clusters for this data is between 3-4, especially if you remove a few outliers.

The result on feature importance based on forest of trees can be seen in figures 21 to 24. By looking at figures 21 and 23 we see that both cases n_cluster=3 and n_cluster=4 share similarities of the ten most important features. It would be logical that the features separating cluster 0 and cluster 2 in figure 20 are more important in the n_cluster=4 case. Hence the differences in the result. If we take a look at figures 21, 23 and 26. Feature 676 is important in both of the scenarios, but increases significantly in the n_cluster=4 case. By looking at its feature value distribution for the different clusters (fig. 26), we can see that its distinguishing cluster 0 and 2 very well (the blue histogram is behind the red one). Implying that its increase in importance was due to its ability to separate these two clusters.

By looking at figures 22 and 24 we are seeing a lot of unrelated features with no useful information regarding the data (according to the clustering done by KMeans). If we again look at figures of feature 209 and 509 (fig. 12, 13) it seems to be the case. Especially for feature 509 we don't seem to have any useful information regarding the data. Most of the feature values seems to be normally distributed with mean zero and variance of one. And an additional outlier far away from the other values. The result from the forest of trees seems to indicate that we have a lot of similar features to that one in our data. Both of the cases have 2-3 significantly more important features and after those is seems to be a soft slope down to the unrelated features (see figures 22, 24). By looking at the five most important for each case in figures 25 and 26 each of these features seems to bring useful information about the clusters. Even though feature 182 in n_cluster=3 and feature 547 in n_cluster=4 seems to not have a lot of separation between the clusters, KMeans still makes a lot of use of them.

3 Exercise 4.2: Improving variable selection in the Lasso

In this task we explore the capabilities of the lasso and adaptive lasso to recover the set of true non-zero coefficients for small n and large n.

Method

The data used for the project was generated following the instructions for the problem as well as course project 3, *Group lasso: Wrong vs correct groups* and its corresponding guide. Here is how the data was generated more explicitly:

```
# Data generation
    X = np.random.multivariate_normal(mean=np.zeros(p), cov=np.eye(p)/m.sqrt(p), size=n)
    beta = np.zeros(p)
    beta[:5] = np.random.normal(loc=0, scale=1, size=int(p*s))
    beta = beta.reshape(-1, 1)
    sigma_e = m.sqrt(np.linalg.norm(np.dot(X, beta))*np.linalg.norm(np.dot(X, ...
        beta))/(n - 1))/gamma
    e = np.random.normal(loc=0, scale=sigma_e, size=n).reshape(-1, 1)
    SNR = np.linalg.norm(np.dot(X, beta))/np.linalg.norm(e)
    y_true = np.dot(X, beta) + e
    y_true = y_true.reshape(-1, 1)
    A_true = np.where(beta != 0)[0]
```

Where the signal-to-noise ratio is around one. The non-zero beta coefficients are sampled from a normal distribution with mean zero and variance one. Since I always use $p < n$ the beta coefficients used for estimating the weights w_i is always estimated using ordinary least squares. The weights used $\gamma = 2$ for computation. This specific value was estimated using simple testing of different values. A more structured estimation based on performance comparisons could been done, but for this project using this value seems to be good enough. P is fixed to 50 features and s is 10% which means that we always have 5 true non-zero and 45 zero beta coefficients. For simplicity during testing and exploration these five features are always the first five. The code was implemented in Python hence the adaptive lasso coefficients $\hat{\beta}_{AdaLasso}$ was estimated using an altered data matrix X' and corresponding weights w_i. The beta coefficients are also estimated using regular Lasso for comparison to Adaptive Lasso. Both Adaptive Lasso and regular Lasso used cross-validation to estimate their penalty parameters. The estimated non-zero coefficients from both adaptive and regular Lasso are then compared to the set $\mathbf{A}$ containing the true non-zero beta coefficients.

The testing is done in the following way. For each specified n the data generation and corresponding coefficient estimation is done 20 times. At each iteration, the amount of correct and incorrect non-zero coefficient estimations, for both adaptive and regular, are computed and stored. After all the 20 iterations, the ratio (correct amount divided by 5 and incorrect divided by 45) of both the methods are plotted in a graph with the ratio on the y-axis and iteration index on the x-axis. The purpose behind this metric is to get a understanding of how fluctuating the parameter estimation is. The ratio for correct estimated non-zero will show how often the methods estimate the true non-zero coefficients correctly. But then we don't know if the methods simply estimate all of the coefficients as non-zero, implying poor performance. This is the reason why the ratio of incorrect estimated non-zero coefficients are also visualized in the graph. Resulting in a good performance when the correct estimation ratio is around 1 and incorrect estimation ratio is around 0.

Results

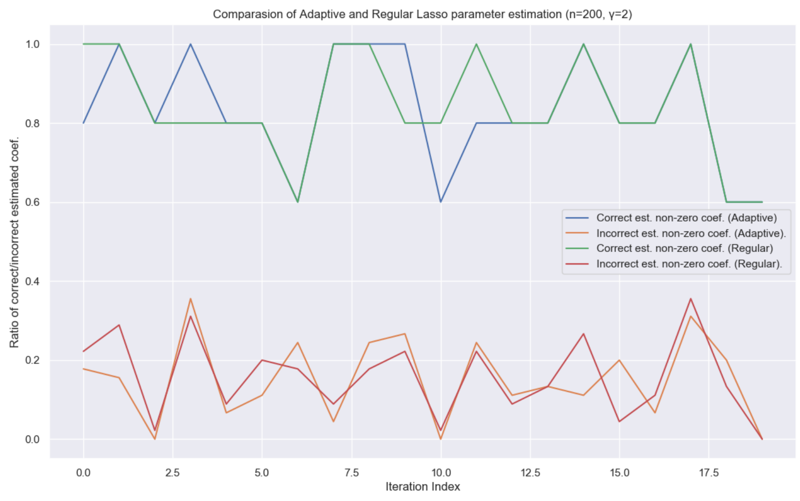

Figure 27: Beta coefficient estimation ratio of Adaptive Lasso and regular Lasso with n = 200

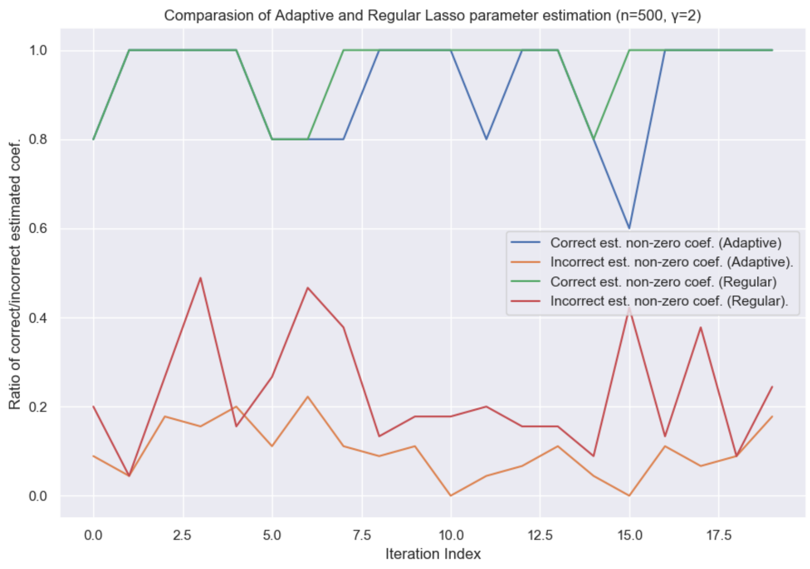

Figure 28: Beta coefficient estimation ratio of Adaptive Lasso and regular Lasso with n = 500

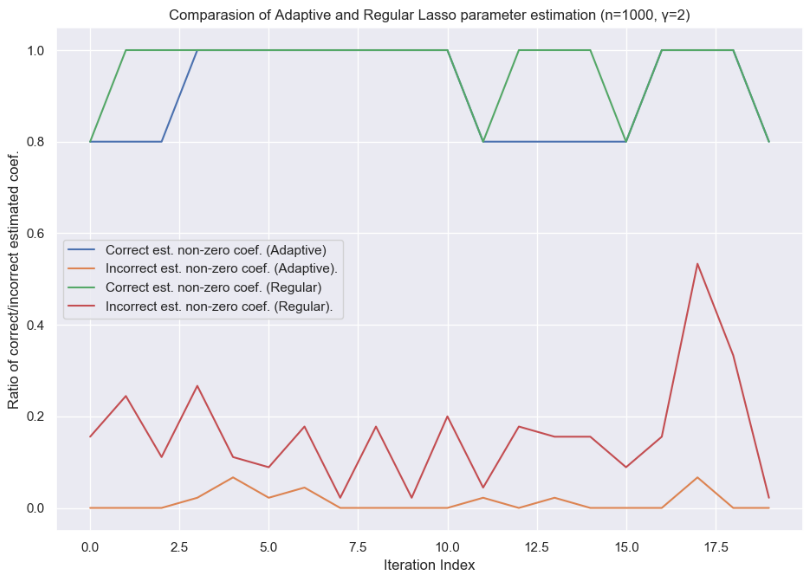

Figure 29: Beta coefficient estimation ratio of Adaptive Lasso and regular Lasso with n = 1000

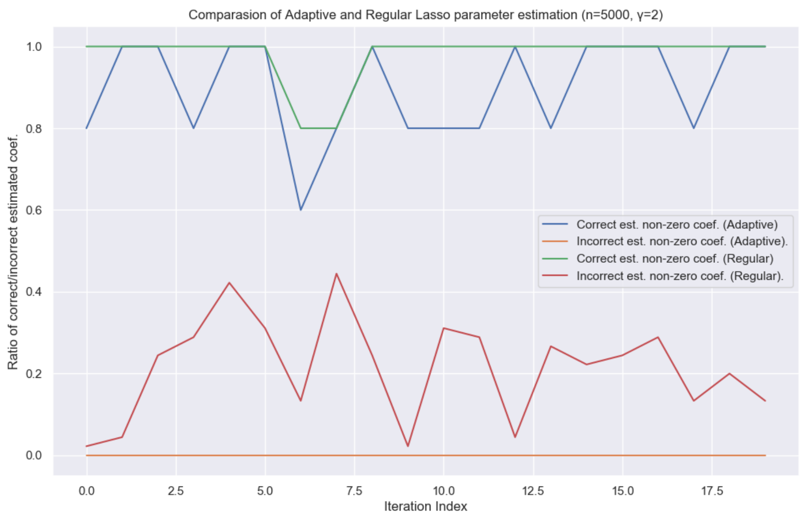

Figure 30: Beta coefficient estimation ratio of Adaptive Lasso and regular Lasso with n = 5000

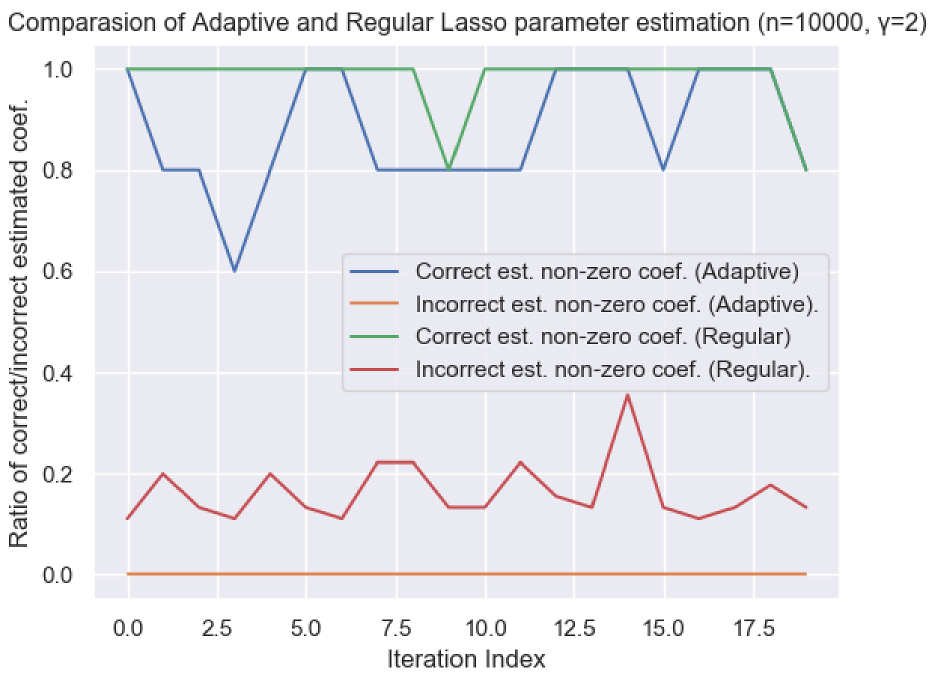

Figure 31: Beta coefficient estimation ratio of Adaptive Lasso and regular Lasso with n = 10000

Discussion

In terms of estimating the true non-zero coefficients are non-zero, from my results there is not too much of a difference between adaptive lasso and regular (blue and green line respectively), see figures 27 to 29. You could further investigate the trend that adaptive lasso is not correctly estimating all of the true non-zero coefficients at larger n. In figures 30 and 31 the adaptive lasso starts to less and less estimate all the true non-zero coefficients correctly with respect to n. It might be affected by the γ parameter used to compute the weights. The most interesting result from these tests are adaptive lassos' ability to not estimate true zero coefficients as non-zero. By looking at figure 27 we see that both adaptive and regular lasso estimate similar amounts of true zero coefficients as non-zero. But when increasing n the adaptive method yields better and better results in terms of incorrectly estimating zero coefficients as non-zero. Already at $n = 500$ in figure 28 we see a separation between the orange (adaptive) and red (regular) line. This separation increases with larger n and at $n = 5000$ the adaptive lasso method never estimates an zero coefficient as non-zero during the 20 iterations. The reason behind this better estimation is, by H. Zou, that the adaptive lasso enjoys the oracle properties [4]. This means that when our data n grows, the data dependent weights w_i for zero-coefficient predictors get inflated to infinity, and the weights for non-zero coefficient predictors converge to a finite constant. But the oracle properties do not automatically result in optimal prediction performance. Since γ strongly affects the weights the choice of the parameter also affects if adaptive lasso has the oracle property. Since the normal lasso do not inherit the oracle property we do not see this change with increasing n in my results. For this experiment with increasing n the methods ability to correctly estimate the entire set $\mathbf{A}$ seems to be consistent over all n. But since γ affects the oracle property, and that λ and γ strongly influences each other, a better method to estimate γ in this case could increase adaptive lassos' performance to correctly estimate all of $\mathbf{A}$. This by improving the convergence for the weights for nonzero coefficients.

References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[2] L. Breiman, *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. DOI: `10.1023/a:1010933404324`. [Online]. Available: `https://doi.org/10.1023/a:1010933404324`.

[3] E. Lewinson. (2019). Explaining feature importance by example of a random forest, [Online]. Available: `https://towardsdatascience.com/explaining-feature-importance-by-example-of-a-random-forest-d9166011959e` (visited on 06/07/2020).

[4] H. Zou, “The adaptive lasso and its oracle properties,” *Journal of the American Statistical Association*, vol. 101, no. 476, pp. 1418–1429, Dec. 2006. DOI: `10.1198/016214506000000735`. [Online]. Available: `https://doi.org/10.1198/016214506000000735`.