Exercise 1: Classification and variable selection

Introduction

In this analysis, we have a high dimensional dataset with binary labels (0,1). The dataset is split into training and validation; the training set contains 323 samples (N) and 800 features (P) while the validation set includes 175 examples and 800 features.

The analysis will begin by exploring the data to learn more about it; after that, we will try to answer questions such as:

1. How can we choose a suitable method to train our data with and generalize to unseen data?
2. Is there a method that performs better than the other? Why?
3. How did we select a suitable number of features that are most related to the data and help produce better performance?

Data exploration and pre-processing

- The dataset seems to have all numerical features, and there are no categorical features.
- The data contains only a small number of samples (323), a complex classifier will overfit the data. Having only 323 samples, the problem seems to tend toward linear classifier option (to be explored).
- The outcome is binary (0, 1), and therefore we need to consider methods suitable for this case.
- We have high dimensionality since P >> N, so we need to consider this fact too.

Data pre-processing

A StandardScaler used on the training and validation sets. The StandardScaler transforms the data, so it has a mean of "0" and a standard deviation of "1". The standardization is useful when we have negative values; it arranges the data in a normal distribution way, which is helpful in classification. If a variance of a feature is more than the variance of other features, that variance might take over other features in the dataset, and we don't want that to happen in our model. The shape of the distribution won't change.
Methods perform better and/or faster when features are on a similar scale and/or close to normally distributed.

Having a large number of features makes it harder to visualize the data. So, a PCA can help in visualizing the data by creating new variables (The principal components, PCs).
Figure 1 shows the two classes (0,1); we can see that the two classes mixed and is it hard to tell whether a data is linearly separable or not. A reason could be the large number of features we have compared to the sample number, and since PCA creates new features from existing ones (which might be highly correlated), it is challenging to tell exactly what is suitable in this case. So a feature selection should be performed first.
We can say that including nearly redundant variables (highly correlated and have the same effect on the target variable) can cause the PCA to overemphasize their contribution compared to others. ***Figure 1*** shows that the data form a cluster and we have some outliers.

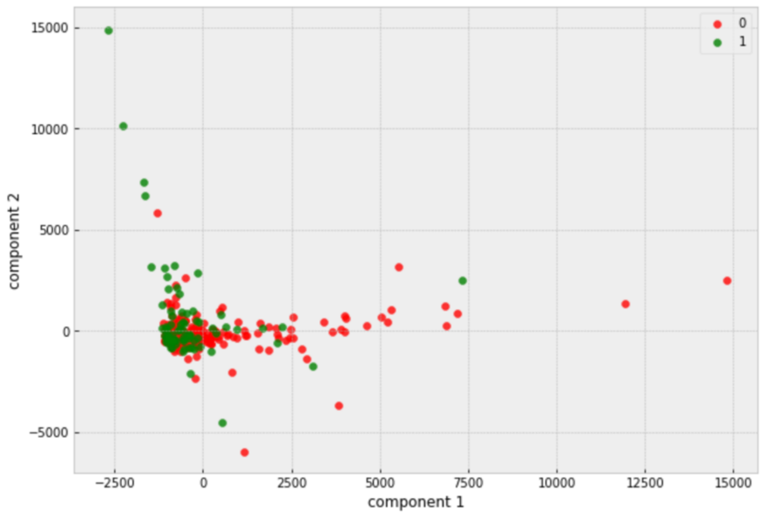


Figure 1: PCA plot for the first 2 components

Methods

From ***Figure 2*** and ***Figure 3*** below, we can see that the dataset contains highly correlated features (highly positively correlated). So we need to perform feature selection and try to minimize the effect of highly correlated features on the model and the model generalization.

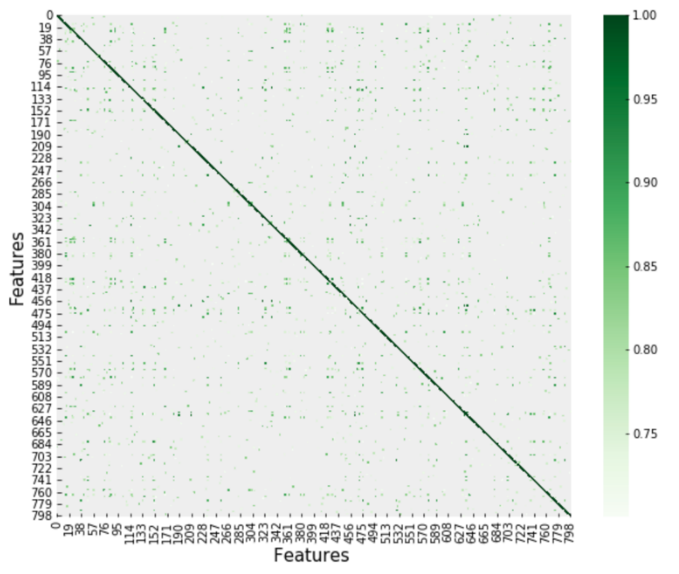


Figure 2: Features Correlation plot (Strong Positive Correlation)

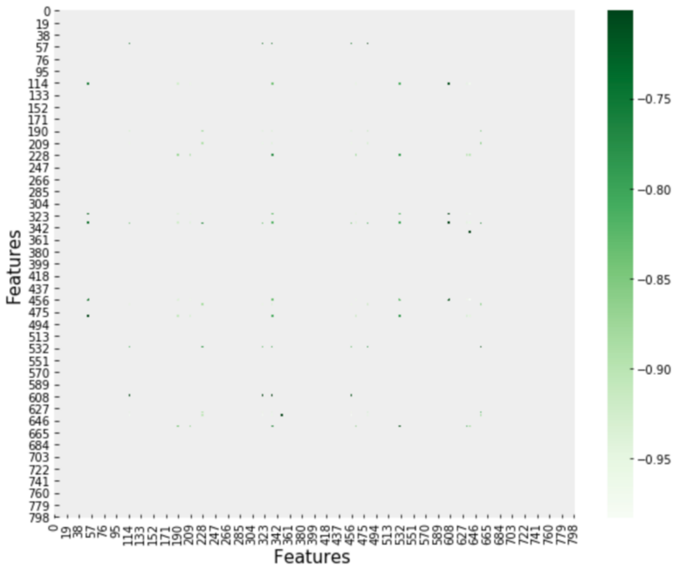


Figure 3: Features Correlation plot (Strong Negative Correlation)

The problem that we are to deal with is the classification problem, and so, choosing a method needs to be suitable for this type of problem. As mentioned above, in this case, where we have a small dataset, a complex classifier might learn the data too well, and that will cause an overfitting problem.
Dealing with a binary outcome introduce Logistic Regression as a suitable method also, the feature selection issue let us think of other advanced techniques such as Random Forest, which implements a feature selection by the feature importance. But several methods were tested on this dataset to help in choosing the suitable methods that are better in predicting outcomes (Next section: Results).

To choose the method, a cross-validation method was used in all experiments to have more accurate results. Cross-validation splits that data into training and test sets so that every point will be in a training/test position every time, and at the end, calculating the result's mean (in this case, the accuracy since it is a classification problem) and compared methods according to this result.

The data have imbalanced data i.e., 0's is the majority of classes compared to the 1's. So, to perform better, Stratification was used in all cross-validation, making sure the ratio is the same in all sets.
The evaluation also included the validation set results, where the chosen method trained using the final sets of features and tested on the unseen data to measure the prediction accuracy. Furthermore, the comparison between the two methods was explored more by using methods; Confusion matrix, Accuracy, F1 score, ROC, and AUC.
Every method has its hyperparameters that need to be carefully chosen to increase the accuracy of the model and so, e.g., in Logistic Regression, when Lasso was included to help in feature selection, a Regularization strength needed to be chosen (λ adjusts the tradeoff between having low training loss and having low weights). As the Regularization Strength decreases, the coefficients become smaller (more coefficients set to zero with more modest regularization strength).
In the Random Forest case, several hyperparameters were sets, such as maximum depth and number of trees. The hyperparameters tested (using several values) and the values that produced better accuracy used in the final method and prediction.

Results

⇒ **Training a baseline classifier**

First, I created a baseline classifier using the most common result. Using cross-validation over the training set and calculating the classification accuracy on each fold, I carried out 10 separate learning experiments (10 folds) where we pick one test set and train on the rest (9 sets in this case). After averaging the test results from those 10 experiments, I got 0.76 accuracy.
If we predict normal (the most common type) for every case, we are right about 76% of the time. The dummy performs pretty well; probably, the majority of the data is the most common type.

⇒ **Trying out some different classifiers.**

I ran several classifiers with default settings on the training data and compared the accuracy resulted from each. ***Table 1*** shows the results; it expected that a good classifier would do better than the baseline Classifier; in this case, we can say that this classifier can learn from our training data and predict unseen observations. Logistic Regression and Random Forest Classifiers have the highest scores.

Classifier	Accuracy (Cross Validation Mean)
Decision Tree Classifier (criterion=" Gini ")	0.7276
Decision Tree Classifier (criterion="entropy")	0.6844
Random Forest Classifier	0.7679
Logistic Regression	0.8140
KNeighbors Classifier	0.7339
Gaussian NB	0.7212
Linear Discriminant Analysis	0.7649
Quadratic Discriminant Analysis	0.3656

Table 1: Accuracy result for each classifier
(Using 10-fold cross-validation)

⇒ **Feature selection**

I started with sparse classification, sparse logistic regression, and applied feature selection using Lasso. Then to use more ways to compare the selection process, I applied the Recursive Feature Elimination (RFE) method on both Logistic Regression and Random Forest (5 CV and 10 CV). Then, observed the most selected features in all cases and confirm the importance of a feature.

◊ **Logistic Regression with L1 Penalty and Various Regularization Strength**

In Logistic Regression, the regularization strength used as C (Inverse of regularization strength; $C=1/\lambda$). Smaller values of C specify more robust regularization. We have three cases we need to be aware of: When C is large, the model will be very complicated (more parameters), and it won't generalize well on new data (overfitting). If C is small, the complexity is low (fewer parameters), and the model will generalize too much, which is not good. We want to find the best fit that generalizes well both on train and test set, and so selecting C value that serves this matter.

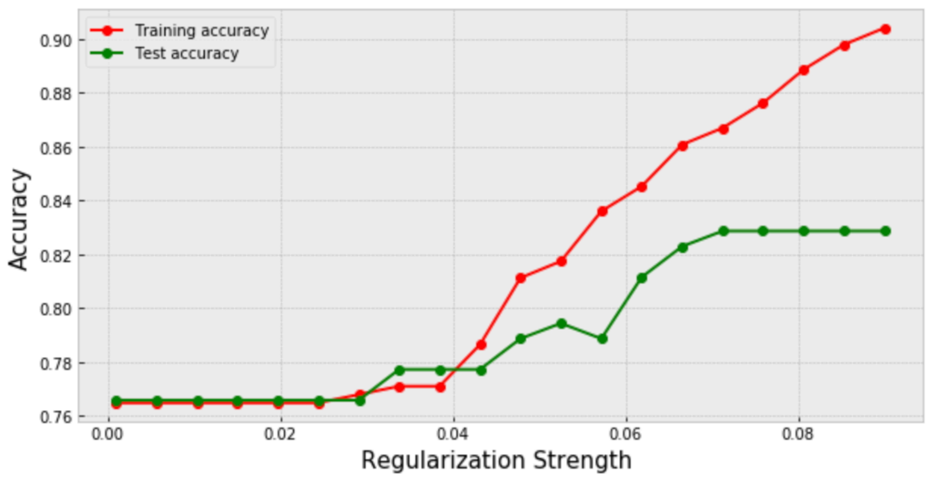


Figure 4: Regularization Strength vs Accuracy

Figure 4 shows that the accuracy is stable at the start, and it increases gradually until approximately C=0.05. After that, the accuracy dynamically increases. I beleive this is a sign of overfitting; using larger C will cause overfitting and very high accuracy for training/test data (high variance). C with about 0.05-0.06 is suitable for a good result.

- ◊ **Logistic Regression with Recursive Feature Elimination Cross-Validation (RFECV)**

RFE uses the accuracy in the model to identify which attributes (or a combination of attributes) contribute the most to predicting the outcome.
Figure 5 shows the LR with RFECV resulted in choosing 17 features out of 800 with an accuracy score = 0.9 (using the training dataset).

```
Max score: 0.9007575757575758
Optimal number of features : 17
```

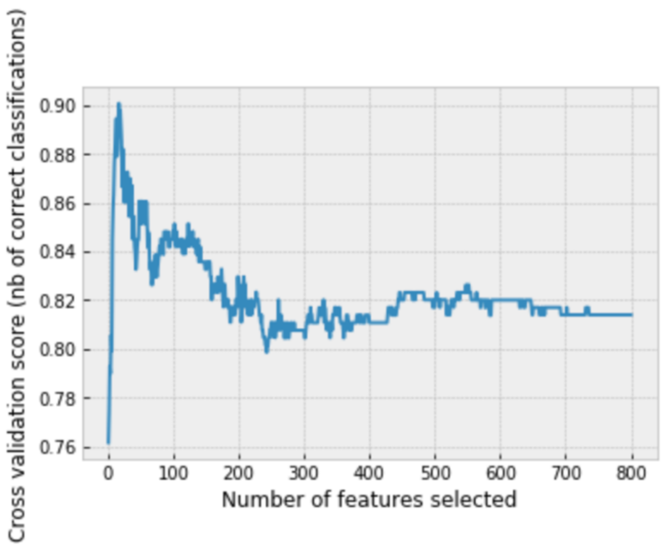


Figure 5: Number of Features VS. Cross Validation scores

◊ **Random Forest**

Figure 6 shows the hyperparameters chosen. Max depth of 6 (max depth was determined using the same process, as the max depth increase, the accuracy increases, meaning that the classifier learns more and creates more leaves. I chose a max depth that is enough to capture the main trends without overfitting), from ***Figure 6*** we can see that 100 trees seem like a right choice since we can see that the accuracy increased slightly around that and then stable.

```
Training accuracy: 0.7710227272727272
Test accuracy: 0.7714285714285715
```

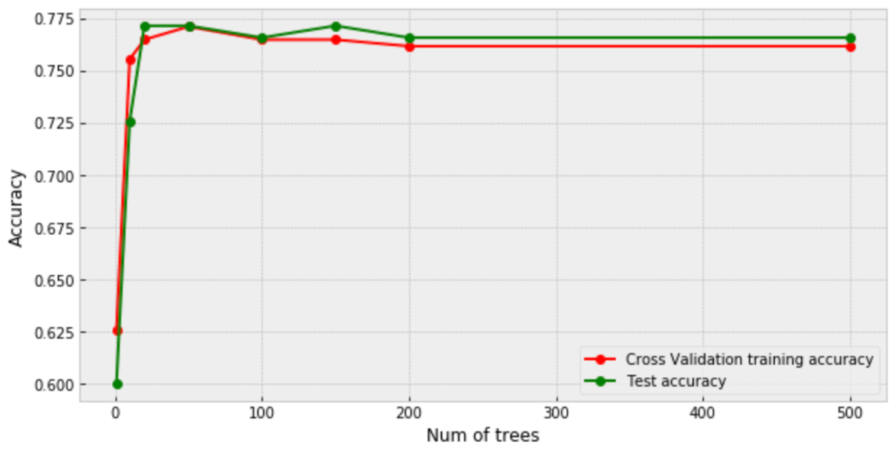


Figure 6: Number of Trees Vs Accuracy using max depth 6

To confirm the results, a GridSearchCV conducted, and the best model selected:

```
The best model: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=True, random_state=0, verbose=0,
                       warm_start=False)
The best score:  0.7678030303030303
```

⇒ **Feature importance in random forest classifiers**

In a Random Forest, the impurity decrease from each feature can be averaged, and the features ranked according to that. This is the feature importance measure in this implementation. Feature importance gives a score for each feature; the higher the score, the more important is the feature.

Figure 7 shows the features and their scores. We can see the importance of many features decrease till reaching zero in the end.

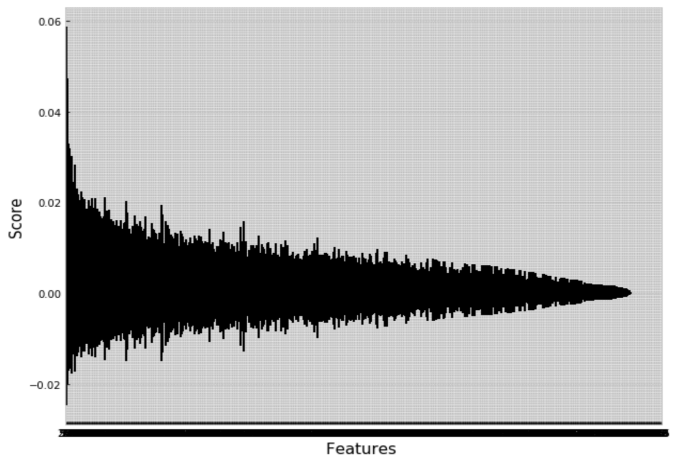
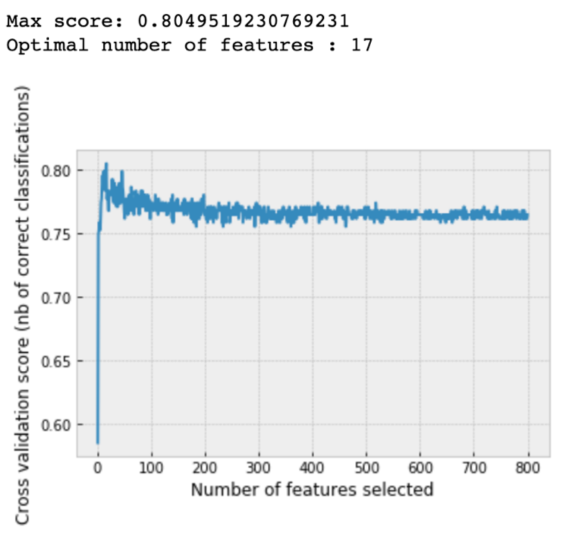


Figure 7: Random Forest Feature Importance

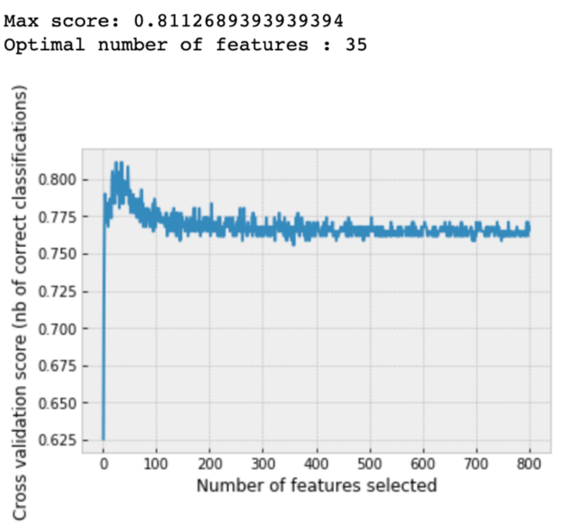
⇒ **Random Forest with Recursive Feature Elimination Cross-Validation (RFECV)**

RFECV was implemented with Random Forest using 5 and 10 folds. RFECV helps in determining the optimal number of features to be selected from the feature space.
In 5-fold RFECV, the optimal number of features is 17, while when using 10-fold RFECV, the optimal number is 35. ***Figure 8*** and ***Figure 9*** show the max score and the optimal number of features in each case.



```
Max score: 0.8049519230769231
Optimal number of features : 17
```

Figure 8: Number of Features VS. Cross Validation (5 fold) scores



```
Max score: 0.8112689393939394
Optimal number of features : 35
```

Figure 9: Number of Features VS. Cross Validation (10 fold) scores

⇒ **Methods' performance**

Features selected in all four methods:

```
LR with Lassso Number of features: 16
[33, 151, 207, 232, 288, 296, 323, 346, 396, 533, 583, 587, 594, 619, 666, 739]
```

```
LR with RFE Number of features: 17
[33, 63, 207, 232, 288, 323, 346, 396, 436, 514, 533, 594, 619, 642, 673, 739, 789]
```

```
RF with RFE (5 fold) Number of features: 17
[33, 144, 189, 232, 235, 240, 288, 323, 436, 496, 533, 587, 619, 666, 682, 709, 739]
```

```
RF with RFE (10 fold) Number of features: 35
[6, 13, 33, 72, 144, 151, 189, 191, 232, 235, 240, 251, 270, 288, 323, 346, 391, 396, 400, 436, 449, 451, 471, 491, 4
96, 511, 533, 587, 619, 666, 682, 709, 739, 755, 794]
```

We can notice that features such as 33, 151, 207, 232, 288, 323, 346, 436, 533, 594, 619, and 739 were selected more than once. LR with RFE seems to have most of these features and so to be more confident in choosing the right subset, I applied all sets in both LR and RF and observed the results.

Figure 10 shows that LR with RFE feature subset does not correlate.

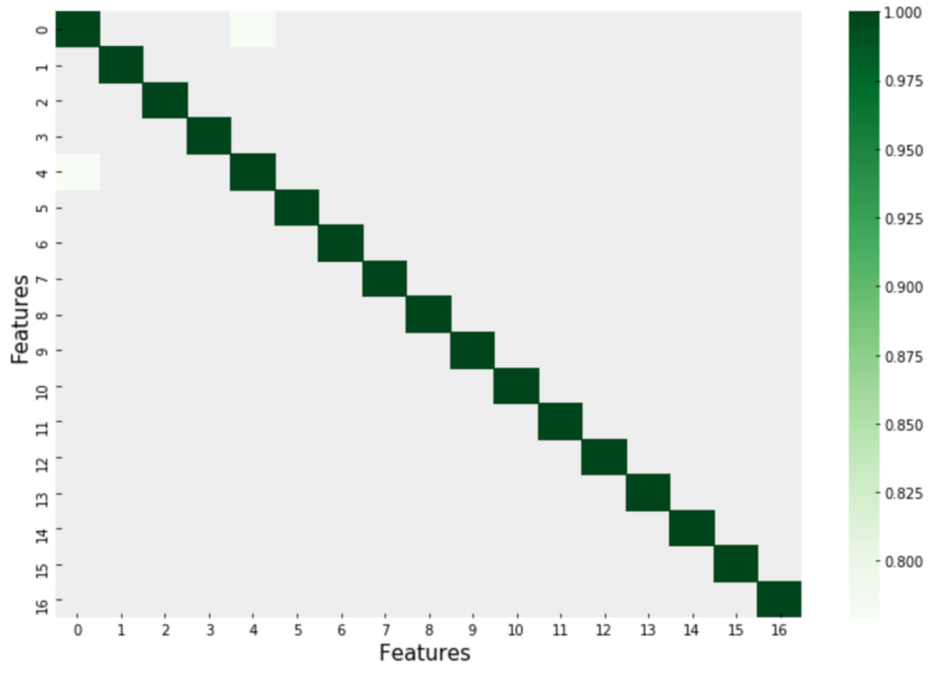


Figure 10: LR with RFE Features Correlation plot

Checking other subsets like in RF showed a few correlations between features and so the final decision was using LR with RFE subset.

⇒ **Evaluation**

Accuracy metric:

Table 2 shows the accuracy scores for Both training data and test data in each Method.

Score Classifier	Training accuracy	Test accuracy
LR	0.94	0.86
RF	0.83	0.82

Table 2: Accuracy result for each classifier

Confusion matrix:

Table 3 and ***Table 4*** dispaly the confusion matrices for LR and RF.

```
                 Confusion Matrix

 Actual/Predicted          | Actual 0        | Actual 1
----------------------------------------------------------
Predicted 0                | 123             | 13
Predicted 1                | 11              | 28
```

Table 3: Confusion matrix LR

```
                 Confusion Matrix

 Actual/Predicted          | Actual 0        | Actual 1
----------------------------------------------------------
Predicted 0                | 130             | 28
Predicted 1                | 4               | 13
```

Table 4: Confusion matrix RF

Classification Report:

Table 5 shows the classification report for both LR and RF.
Since the features don't tell what type of data we have, a score representing both precision and recall could be a good judgment.

In ***Table 5*** F1 score for LR and RF show both methods are better in finding 0 class over the 1 class.
LR better in finding 1 class compared to RF.
The accuracy of LR is higher than the accuracy of RF.

```
Classification_report LR:

              precision    recall  f1-score   support

           0       0.90      0.92      0.91       134
           1       0.72      0.68      0.70        41

    accuracy                           0.86       175
   macro avg       0.81      0.80      0.81       175
weighted avg       0.86      0.86      0.86       175
```

```
Classification_report RF:

              precision    recall  f1-score   support

           0       0.97      0.82      0.89       158
           1       0.32      0.76      0.45        17

    accuracy                           0.82       175
   macro avg       0.64      0.79      0.67       175
weighted avg       0.91      0.82      0.85       175
```

Table 5: Classification Report

ROC Curve:

Figure 11 shows that LR has a better AUC (0.93) compared to RF (0.83), meaning it can predict the 1 class better (true positive).

The curve is near the top left corner, which tells that it is a good ROC curve.

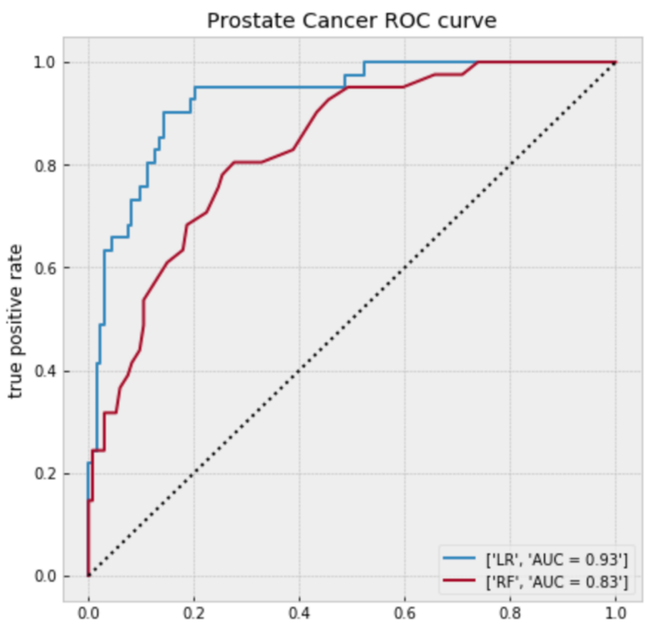


Figure 11: ROC curve for LR and RF

Discussion

- Logistic regression and Random Forest are comparable for smaller datasets with less than 1000 observations.
- LR performed better than RF in this case. The dataset has all numerical features, a small number of samples, and the target is binary. All these factors, I believe, made LR a better option or performs better than RF.
- LR showed that it could handle a large number of features compared to the sample numbers, and with the use of the right tools, we can obtain a suitable subset of features that help in generalizing the model.
- LR is less prone to overfitting.
- Classifier accuracy decreases as the number of features increases (as we saw when applying methods on data before selection). The models become overtrained very quickly.
- RF is an ensemble-based learning algorithm comprised of n de-correlated decision trees. It was built using bootstrap aggregation, where we do resample with replacement to reduce variance, and it performs feature selection implicitly.
- RF performs better when having more data and a more complex situation, like having categorical features. RF needs little or no pre-processing in this case compared to LR.
- Even though that is not clear if data is linearly separable or not from a 2d plot (I think data separable by hyperplane or separation might be caused by too few observed cases/too many variables), the results obtained indicate that there are very good features giving very good classification.

Exercise 2: High-dimensional Clustering

Introduction

In this report, we will analyze a high dimensional dataset with no labels (unsupervised learning; Clustering). The dataset contains a matrix X with n = 302 soil samples and P = 728 features.

The analysis will begin by exploring the data to learn more about it; after that, we will try to answer questions such as:

1. How many clusters/suitable numbers of clusters we can choose to group our data?
2. How can we visualize the data more easily?
3. What are the most important/relevant features that contributed to forming the clusters (up to 5 features)?

Data exploration and pre-processing

- The dataset seems to have all numerical features, and there are no categorical features.
- The data contains only a small number of samples (302) with no target value, and so a clustering algorithm needs to be applied in this case.
- The dataset contains no duplicates or null values.
- We have high dimensionality since P >> N, so we need to consider this fact.
- The data was standardized to have a mean of 0 and a variance of 1 for a more straightforward implementation of specific methods.

Methods

The dataset has a lot of features that make it hard to visualize. So, PCA and t-SNE methods used to visualize the data, and try to understand the pattern and perhaps a potential cluster behavior from the components. Furthermore, a 3D plot used to show the data in a higher dimension.

Two methods used that helped with an idea about a suitable number of clusters that we can choose to group the data, and this was helpful to implement algorithms later. The two methods are Within Cluster Scatter (Inertia; K-Means) and Silhouette Width (Further details in the result section).

To implement and try to cluster high dimensional data, K-Centroids Cluster analysis, K-Means, and Hierarchical Clustering Algorithms were applied on the dataset and helped in extracting the relevant features that we are looking for. Furthermore, PCA was implemented in conjunction with k-means because it is a suitable method for visualizing high dimensional data and help in performing clustering in smoother way.

XGBoost classifier used after confirming the optimal number of clusters for this dataset, and this classifier used to help in determining the important features that mostly identified as the most relevant features for this data.

Results

⇒ Reduce Dimensionality

◊ PCA

In this step, we will try to find a good number of components that capture the largest variance in the data.

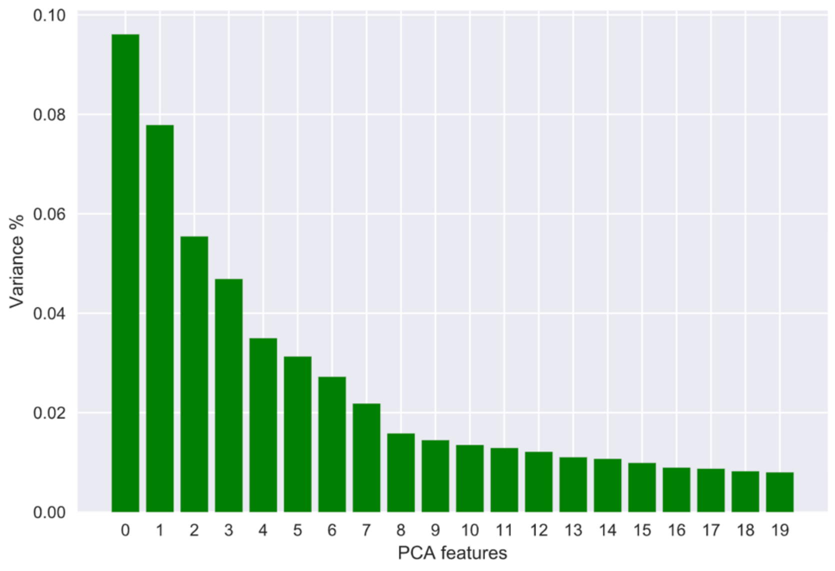


Figure 1: Scree plot showing variance decrease after the second component

Figure 1 shows that the first two components explain the majority of the variance. For this, we will plot just the first two to notice if there are any clear clusters.

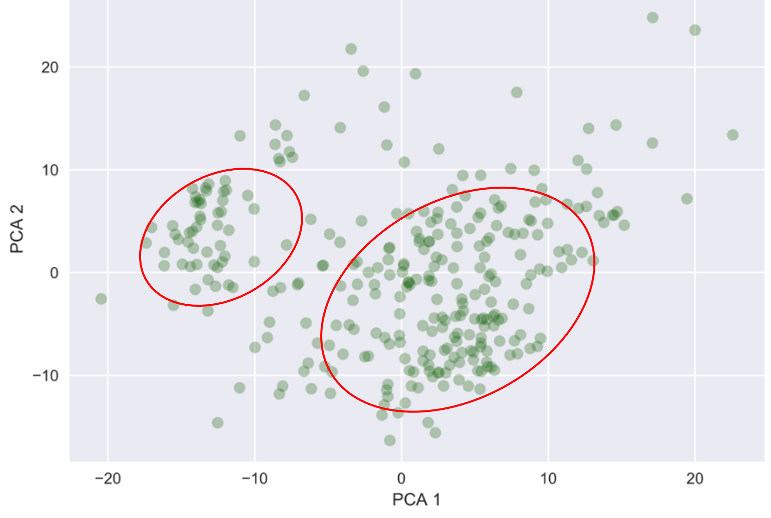


Figure 2: Scatter plot of the first twe PCA components

Figure 2 shows at least two distinct clusters. This tells us that we can group the observations in this dataset. It is essential to remember that we do not have a target label to label the groups, so we do not know exactly what the labels are.

◊ PCA with K-Means

k-means clustering used to display the top three PCA components.
First, we will try to determine the number of clusters in k-means, and this is done by measuring the sum of the squared error for each cluster (inertia), ***Figure 3*** indicates the change in the value of inertia. It decreases with more clusters, and it seems like after 3 clusters (elbow), the change in inertia is kind of no longer significant.

Figure 4 indicates the silhouette score, which is a measure of how similar observation is to its cluster compared to other clusters. A high score means that the observation is similar to its cluster and poorly matches with its neighboring clusters (highest score is at 3 clusters which agrees with elbow method).

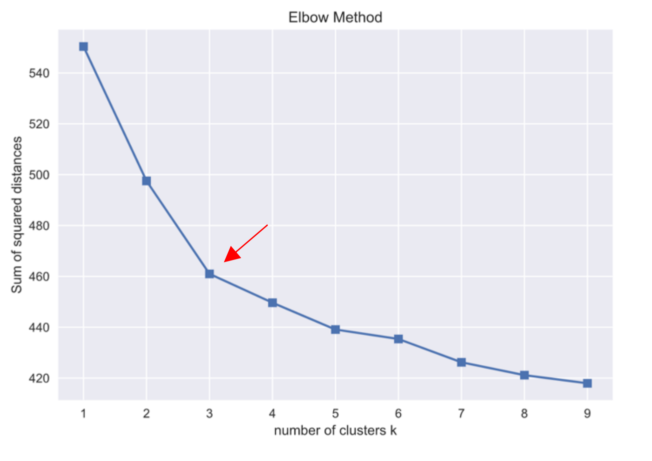


Figure 3: Scree plot shows a decrease in inertia

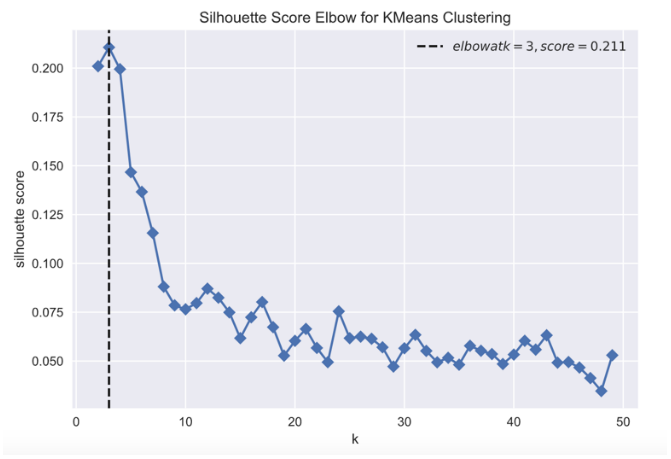


Figure 4: Silhouette Score for KMeans Clustering

Figure 5 shows that fitting the components to the k-means displays some clearly defined clusters in the data. ***Figure 6*** confirms more that 3 clusters are a suitable number of clusters using t-SNE; we can see clearly/distinct groups.

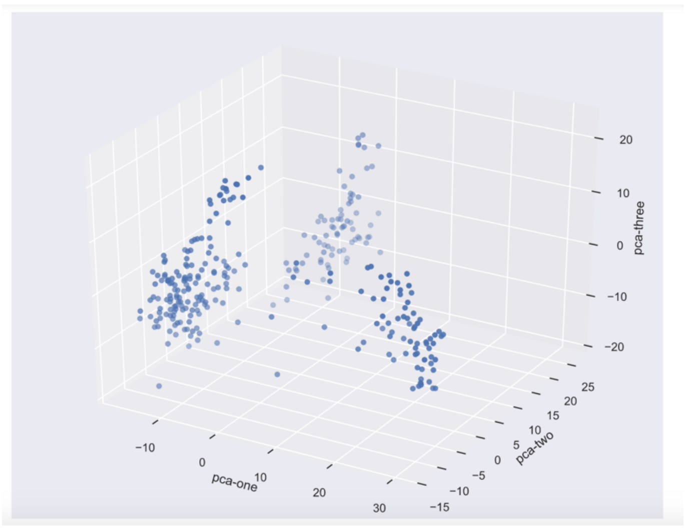


Figure 5: 3D plot of data using K-Means

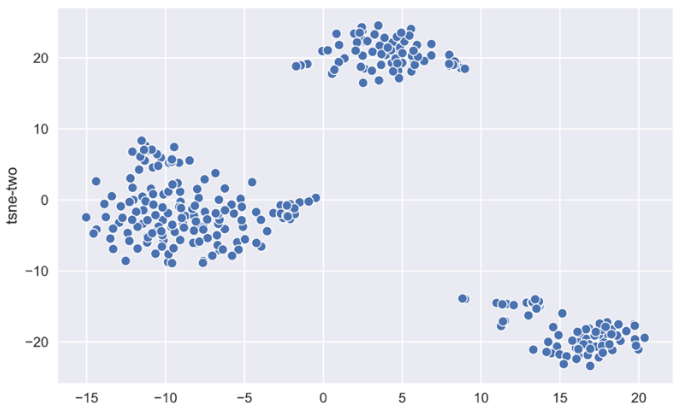


Figure 6: t-SNE visualization using two components

⇒ K-Centroids Cluster Analysis (K-Means)

Figure 8 shows the cluster's centers and how many points assigned to each cluster. Cluster 1 seems to have the majority of points followed by cluster 2 and cluster 3 with similar number of points.

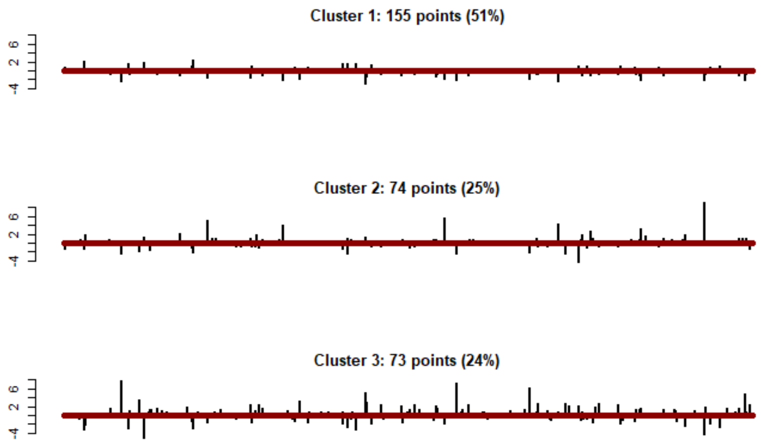


Figure 8: K-Centroids Cluster Analysis

⇒ Hierarchical Clustering

Hierarchical Clustering used to help in pointing out the most relevant features contributed in forming clusters. ***Figure 9*** shows the last three merged clusters and the size of each cluster (Ward linkage used, which minimizes the merged clusters' merged). We can notice that K-Means and Hierarchical Clustering have similar clusters sizes; two clusters have close sizes, and one precisely identical (***Figure 8 and Figure 9***).

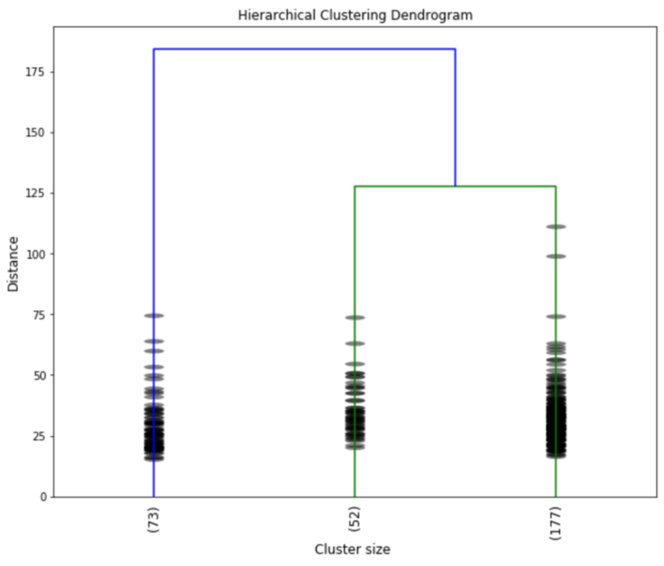


Figure 9: Hierarchical Clustering shows the last 3 mergerd clusters

◊ **Feature Importance**

Having a high dimensional data means that we have a large number of features that might not benefit the data or support the purpose of building a model that can learn and generalize to future data.
To further discover feature importance and select a group of most relevant features; after knowing the suitable number of clusters (in this case, 3) each observation grouped with its corresponding cluster label and a classification method implemented on this data.

XGBoost classifier was applied to this data and tested to see how it performs. The data split into training and test set, the classifier trained, and then the test set was used to make the prediction and calculate the accuracy of the classifier. The accuracy of the classifier was 97%, which is a very high score. Then, the classifier used to determine the most relevant features according to this scenario. The feature importance score indicates how useful the feature was in the construction of the model's boosted decision trees. The more the feature used to make key decisions in decision trees, the higher its score.

Figure 10 shows the importance score of each feature (only displays the top 13 features).

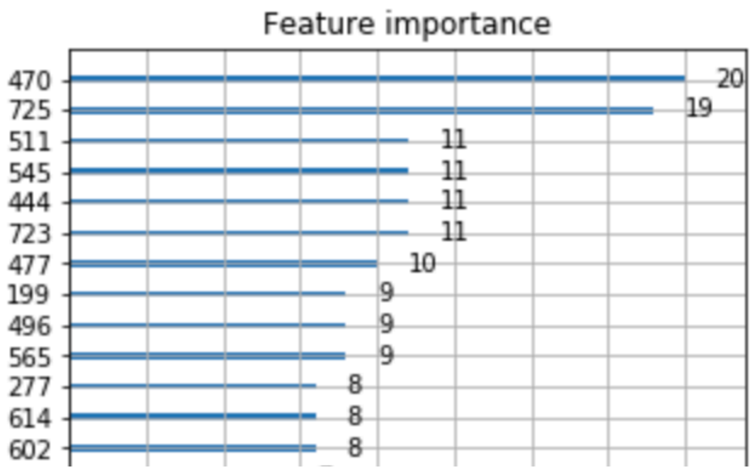


Figure 10: Feature score

Discussion

- The dataset didn’t have any label, but this doesn’t mean that we can’t discover patterns and similar characteristics between samples.
- Applying several methods to the same data helped confirm many of the results we obtained, such as the optimal number of clusters we can get from this data set, and we have a better sense of how many groups we can label the soil samples with.
- For example, it is possible to come up with a model that categorizes the soil in this data into three categories.
- PCA and k-means helped in giving us the ability to uncover hidden patterns and to think about how we can come up with a model to generalize those patterns onto future observations. They allow us to see the big picture while we pay attention to small details.
- Visualizing data using t-SNE, and 3D plots provided a better picture of data groups.
- XGBoost uses the second derivative as an approximation in addition to advanced regularization (L1 & L2), which improves model generalization. This was proved by implementing this classifier and obtaining a very high prediction accuracy.
- Combining different methods (clustering and then classification) was an interesting approach, and it shown that such approach is applicable, and it helped in reaching satisfying results.

Exercise 3: Improving variable selection in the Lasso

Introduction

In this analysis, the data was generated to work with it in a regression setting (numeric features). The data simulated was considered from a sparse linear regression model $Y = X\beta + \varepsilon$ where:

- The number of features fixed P = 50.
- Signal to noise ratio $\approx$ 1.
- The proportion of nonzero coefficients s = 10%.
- The number of samples will increase but generally N > 50.

The analysis will begin by generating the data to perform further exploration; after that, we will try to answer the question:

- What are the capabilities of the Lasso and the adaptive Lasso in recovering the set of true nonzero coefficients?

Methods

Data generation

- Data simulated from the model $Y = X\beta + \varepsilon$, where Y: responses (outcome), X: features β: coefficients, and ε: noise.
- X generated from a normal distribution with mean zero and standard deviation $\frac{1}{\sqrt{p}}$.
- β generated from normal distribution $N\left(0,\ I_p\right)$ entry-wise. (p: number of predictors/features).
- The noise (ε) generated from normal distribution $N\left(0,\ \sigma^2\ I_n\right)$. (n number of samples).

Adaptive Lasso computation

- Given the feature matrix X and the responses y.
- X' was generated from X and w; where X' = X $\mathrm{diag}(w)^{-1}$, w is the vector of weights from the ridge regression estimates.
- A standard lasso regression performed using X' and y to obtain lasso coefficients.
- The adaptive lasso coefficients calculated using the equation $\beta_{AdaLasso} = \mathrm{diag}(w)^{-1} * \beta_{Lasso}$

Both lasso and ridge regression had a hyperparameter alpha (lambda). Lambda was chosen using cross-validation, which was performed using 10 folds to obtain the best lambda value through LassoCV and RidgeCV (the best lambda gives the smallest mean error). The data fitted into these methods to obtain the required coefficients for further comparison.

For reliable conclusions, the results were based on multiple simulations (500 times), and the ***mean*** considered as representative for each measure.

Every time a dataset was simulated, a confusion matrix for the estimated coefficients compared to the true coefficients was calculated. True positives are coefficients that are nonzero in both the true and estimated coefficients. False positives are nonzero estimated coefficients but not in the true ones. Using the confusion matrix, the sensitivity, and specificity values computed.

Results

⇒ Performance Evaluation for P = 50 and N = 100

Measure (mean) /method	Lasso	Adaptive Lasso
Sensitivity	0.51	0.51
Specificity	0.98	0.98

⇒ Performance Evaluation for P = 50 and N = 500

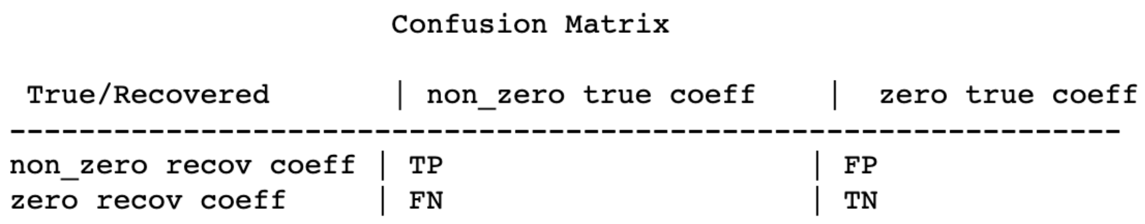
Measure (mean) /method	Lasso	Adaptive Lasso
Sensitivity	0.89	0.72
Specificity	0.77	0.99

⇒ Performance Evaluation for P = 50 and N = 1000

Measure (mean) /method	Lasso	Adaptive Lasso
Sensitivity	0.92	0.77
Specificity	0.76	0.99

⇒ Performance Evaluation for P = 50 and N = 5000

Measure (mean) /method	Lasso	Adaptive Lasso
Sensitivity	0.97	0.86
Specificity	0.74	1.0



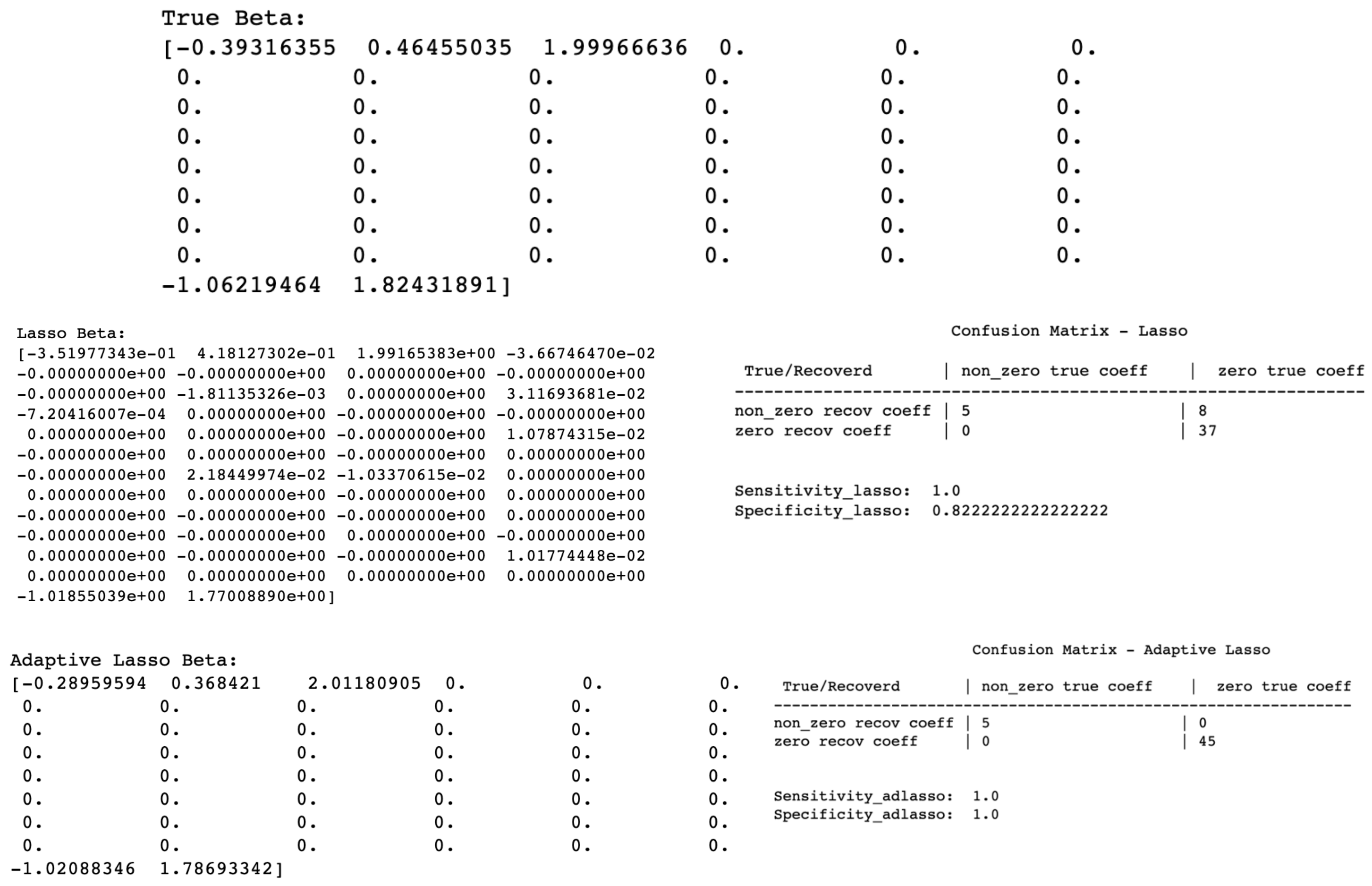
```
                    Confusion Matrix

 True/Recovered        | non_zero true coeff    |  zero true coeff
----------------------------------------------------------------
non_zero recov coeff | TP                       | FP
zero recov coeff     | FN                       | TN
```

Figure 1: The confusion matrix table

Figure 1 shows the confusion matrix model that used in doing the calculation above.

From the results above, we can notice that in the case of Lasso, when N increases, the sensitivity increases while the specificity decreases. In the adaptive lasso case, when N increases, the sensitivity increases, and the specificity is very high and reaches 1 for larger N.

Figure 2 shows a more in-depth look into what happens to the true coefficients set and the corresponding estimated coefficients set. We can see in this example (n = 5000) that for Lasso, when true coefficients are zero, the corresponding estimated coefficients are set to zero or are really small values. So, the recovered nonzero coefficients are larger than the true set.
In adaptive Lasso, we can see that the true nonzero coefficients recovered, and zero coefficients are estimated precisely zero.



```
True Beta:
[-0.39316355  0.46455035  1.99966636  0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
 -1.06219464  1.82431891]
```

```
Lasso Beta:
[-3.51977343e-01  4.18127302e-01  1.99165383e+00 -3.66746470e-02
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -1.81135326e-03  0.00000000e+00  3.11693681e-02
 -7.20416007e-04  0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  1.07874315e-02
 -0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00  2.18449974e-02 -1.03370615e-02  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00  1.01774448e-02
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -1.01855039e+00  1.77008890e+00]
```

```
                  Confusion Matrix - Lasso

 True/Recoverd        | non_zero true coeff    |  zero true coeff
----------------------------------------------------------------
non_zero recov coeff | 5                       | 8
zero recov coeff     | 0                       | 37


Sensitivity_lasso:  1.0
Specificity_lasso:  0.8222222222222222
```

```
Adaptive Lasso Beta:
[-0.28959594  0.368421    2.01180905  0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
 -1.02088346  1.78693342]
```

```
               Confusion Matrix - Adaptive Lasso

 True/Recoverd        | non_zero true coeff    |  zero true coeff
----------------------------------------------------------------
non_zero recov coeff | 5                       | 0
zero recov coeff     | 0                       | 45


Sensitivity_adlasso:  1.0
Specificity_adlasso:  1.0
```

Figure 2: The true/Estimated coefficients

Discussion

- Sensitivity measures a fraction of correct positive (coefficients that are nonzero TP) to all actual positive (TP and FN).
- A high sensitivity will flag almost every nonzero coefficient (in both true and estimated) and not generates many false-negative results (true nonzero while estimated zero).
- Specificity measures the fraction of correct negatives (coefficients that are zero TN) to all actual negative (TN and FP).
- A high specificity will correctly rule out almost every coefficient that is zero (in both true and estimated) and won't generate many false-positive results (true zero while estimated nonzero).
- As N grows, Lasso tends to have lower specificity. As we saw from the example in ***Figure 2***, Lasso has several zero true coefficients that were set to be nonzero (very small values, but not zero).
- As N grows, adaptive Lasso has a higher sensitivity and specificity, and that indicates, as we saw from the example in ***Figure 2***, the adaptive Lasso can recover the set of true nonzero coefficients.
- Adaptive Lasso retains the convexity property of the Lasso by yielding consistent estimates of the coefficients. So, adaptive Lasso can recover the true set of nonzero coefficients under average conditions than the Lasso does.
- Lasso performs variable selection because the L1 penalty is singular at the origin point, and so, lasso shrinkage produces biased estimates for the large coefficients. Adaptive Lasso simply adds weights to the coefficients to try to solve the biased issue.
- Considering the amount of samples N/P per parameter. If $P > N$ and the true model is not sparse, then the number of samples is too small to estimate the parameters accurately. If the true model is sparse so that only $K < N$ parameters are nonzero in the true model, then we can adequately estimate the coefficients.
- Even though it is hard to know which K of the P parameters are nonzero, but it turns out that adaptive Lasso performed well and better than Lasso when comparing true zero coefficients to the corresponding estimated coefficients in ***Figure 2***.