Ivo Vladislavov Petrov

Applied Data Science Coursework Report

Data-Intensive Science

December 24, 2023

Word Count: 2997 (using Overleaf)

Contents

List of Figures

Chapter 1

Introduction

This report outlines our approach for the **Applied Data Science** coursework. We have prepared the methodology to be reusable and most importantly reproducible. Because of the lack of domain knowledge, due to the unknown nature of the data, each decision was made through quantifiable statistical evidence. In the following sections, we will discuss several pre-processing and modelling techniques, which have been designed with robustness in mind.

Chapter 2

Section A

2.1 Q1 - Dataset A: Exploration, Dimensionality Reduction and Clustering

2.1.1 Part (a)

The given dataset contains 408 rows of 500 features, with no missing entries. We then observe a sample of the first 20 features to gain a better understanding of the data. As most strongly observed in Features 11, 14, 18 and 19 in Figure 2.1, we can differentiate a bimodal structure in the distribution, hinting towards the presence of 2 different clusters. However, in most cases, there is no discernible relation to the classification labels, with the slight exception of feature 19, in which class 1 dominates the left cluster. Furthermore, we can observe that many features have a lot of data points around 0. This might lead us to suspect issues of near-zero variance, which will be explored in the pre-processing part of *Q4*.

2.1.2 Part (b)

Before applying Principal Component Analysis (PCA), a necessary step is to standardise the dataset. We will not consider outliers in this case, as they will be further discussed in *Question 3*. Hence, we apply the transformation $z_{Fea_i} = \frac{X_{Fea_i} - \mu_i}{\sigma_i}$, where μ_i is the mean, and σ is the standard deviation. This transformation is necessary, as centring and scaling the data is important for correctly calculating the variance, as shown in Figure 2.2.

To visualize the data we perform PCA with only 2 components. As hypothesised in *part (a)*, we can observe the existence of 2 different clusters, which can be seen in Figure 2.3. That said, the amount of variance explained by the first 2 components amounts to only 35%, meaning that a significant portion of the variance remains unexplained.

2.1.3 Part (c)

Splitting the dataset in half, and training a K-means classifier for each half will show whether our algorithm has reached a stable set of cluster centroids. It is important to note that the split is done using stratification. If the clusters differ significantly between the two models, we can conclude that we are not capturing an underlying structure but noise in the data. The default number of clusters is 8, which is likely more than optimal considering we did not see such a structure in our PCA visualisation.

This is exactly what we can observe in Figure 2.4. The clustering is inconsistent because we

Figure 2.1: *The density plots for the first 20 features. Class distributions were stacked on top of each other to show the total distribution. most features either exhibit a bimodal structure or are highly concentrated around 0.*

are forcing the algorithm to find clusters that do not exist.

Hence, the results become quite inconsistent. It is notable that while the clusters do not look visually separated, this is due to us considering every feature for clustering, instead of only the principal components which will be discussed in *Part (e)*. We can further confirm that by observing the contingency table (Table 2.1).

2.1.4 Part (d)

Continuing from the previous discussion, we can conclude from Table 2.1 that it is highly likely that our number of clusters is unstable. The fact that there are no two clusters that exhibit anything remotely close to agreement, instead of a 1-to-1 correspondence shows that our clustering is likely to be highly unstable.

We can then use the aforementioned criteria until we reach a satisfactory number of clusters. The best convergence we achieve is at only $k = 2$ clusters. As we can see in Figure 2.5, this corresponds to precisely the 2 clusters we can identify using PCA.

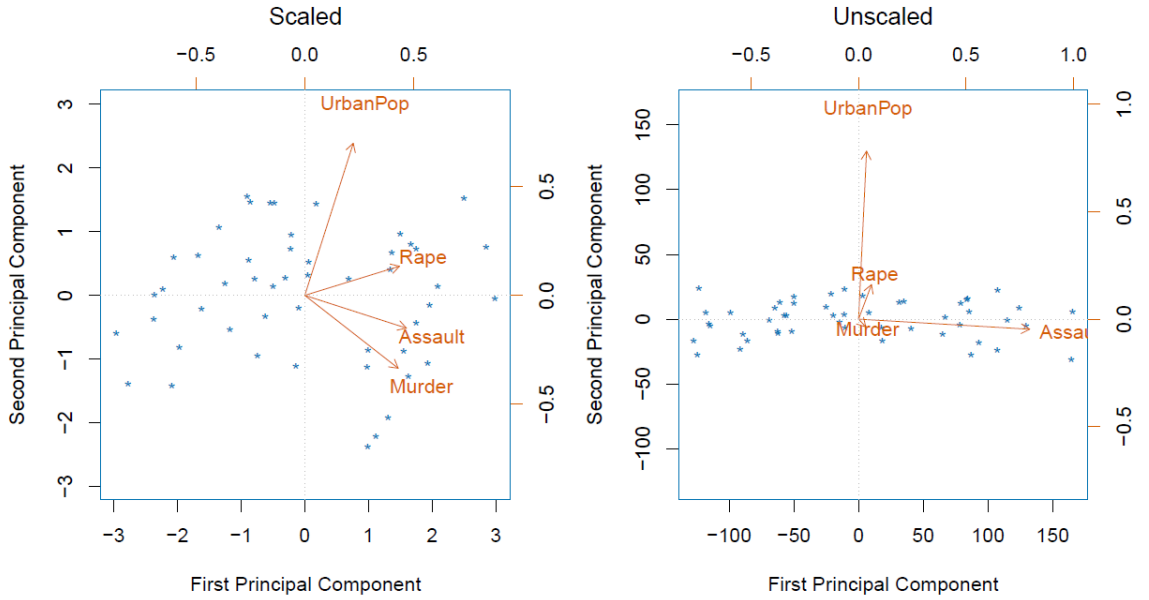


Figure 2.2: *The left plot shows correctly performed PCA, where every feature has been centred with unit variance. The right plot shows what would happen if PCA was done without normalisation. The example was taken from the ISLP textbook[1].*

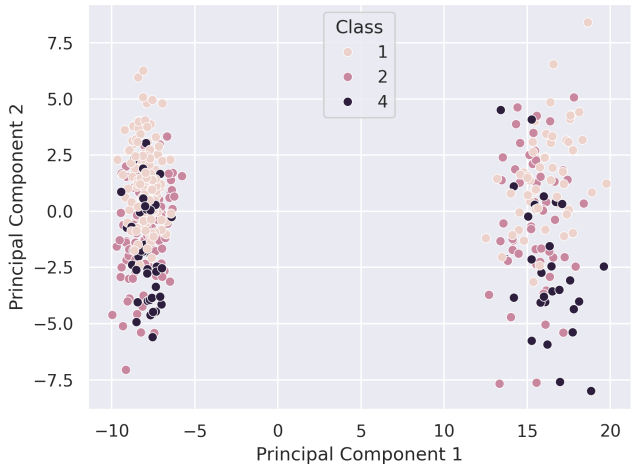


Figure 2.3: *The 2 aforementioned clusters are separated by PCA. Furthermore, while there is a mix of classes in each cluster, we find that in the lower regions of the second principle components, we cannot find points of class 1. This similarly holds in the higher value regions, where only class 1 can be found.*

2.1.5 Part (e)

While the two previous parts do not show any concrete relationship between K Means clustering and PCA, except for the obvious two clusters, we will now attempt to reason about it for $k = 3$.

We can notice in Figure 2.6 that the clusters when performing PCA first are perfectly visually separable. This can be useful if we want to emphasise the feature explainability, as we can link the components and the clusters. Furthermore, we are likely to have a more stable clustering with fewer features.

However, we further observe that when the order is reversed, two clusters significantly overlap in the PCA space. This means the best separation may lie in a component perpendicular to the defined space. Because the defined PCs cover only 35% of the variance, we are likely missing

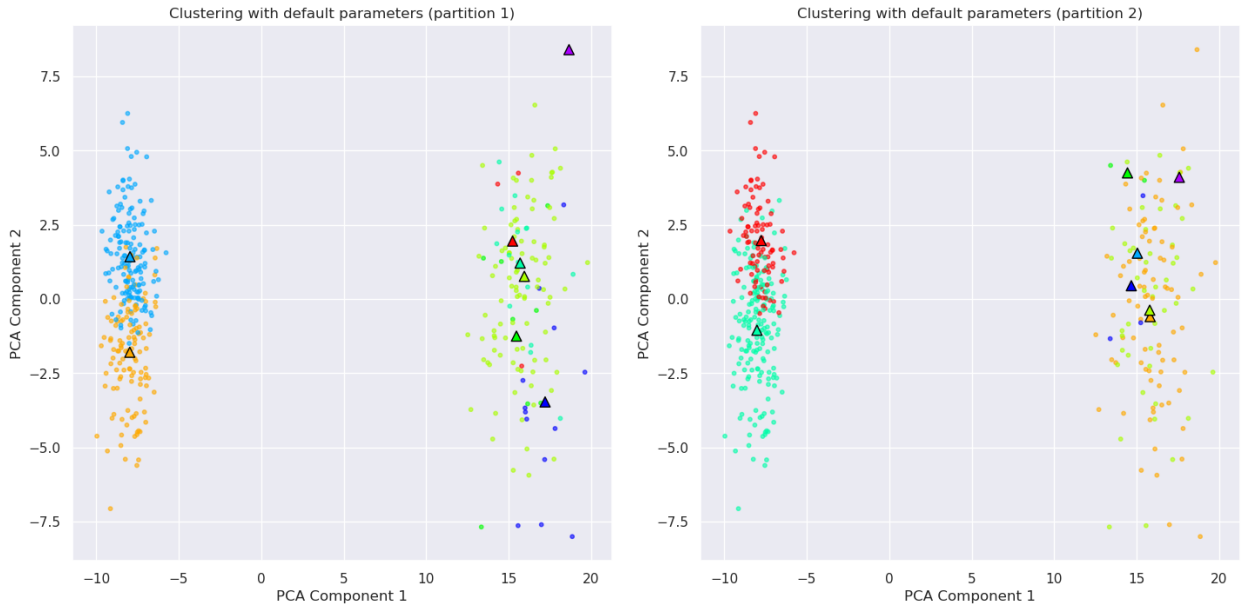


Figure 2.4: *The two different splits cause vastly different clusters. Notice the centroids on the right differ significantly. However, the two clusters shown on the left-hand side seem to overlap well.*

Cluster (partition 2)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Cluster 1	0	**2**	1	0	0	0	0	0
Cluster 2	7	0	0	0	**111**	0	0	0
Cluster 3	0	**60**	30	2	0	1	2	1
Cluster 4	0	3	**5**	0	0	0	0	0
Cluster 5	0	**9**	5	0	0	0	1	0
Cluster 6	**85**	0	0	0	69	0	0	0
Cluster 7	0	**9**	4	0	0	0	0	0
Cluster 8	0	**1**	0	0	0	0	0	0

Table 2.1: *Contingency table between 2 different clustering models. Cluster numbering is arbitrary, as it does not provide us with any information. For example, in perfect clustering, cluster 1 using the first training set might correspond to cluster 4 instead. Bolded values correspond to the maximum overlap in a given row. As can be seen, the agreement between the 2 partitions is quite poor.*

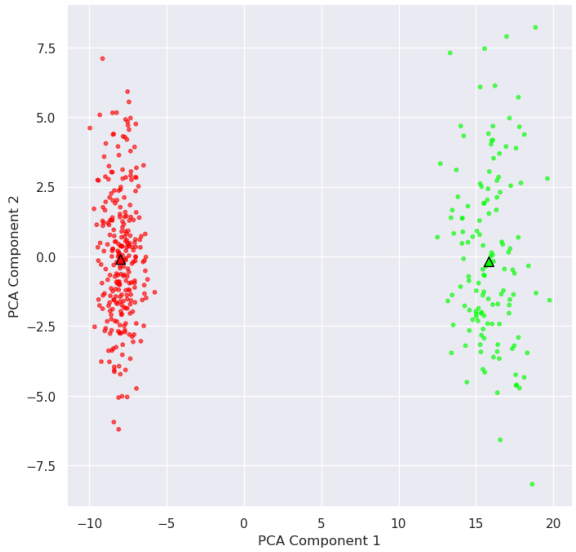
crucial information.

> **Remark** The optimal order of operations varies depending on the dataset and the domain knowledge. While K-means might perform worse in a higher-dimensional space, it is also not preferable to omit 65% of the variance. Therefore, there is likely a different optimum which we will not attempt to find.

2.2 Q2 - Dataset B: Missing Labels and Duplicated Observations

2.2.1 Part (a)

We notice in Table 2.2 that there are relatively few missing values, constituting less than 5% of the total dataset. Class 4 is less represented than the other 2, but not to a great degree.



Cluster	1	2
Cluster 1	0	**136**
Cluster 2	**272**	0

Figure 2.5: *The identified clusters are precisely what we expected - the 2 observed using PCA. The fact that we have a perfect match between the two partitions, as seen in the table on the right, can assure us that we have achieved a stable clustering.*

Figure 2.6: *If we perform PCA first (left), we obtain non-overlapping convex hulls, meaning the clusters are well-defined within the first 2 PCs. Otherwise (right), we obtain overlapping regions that show there might be other cluster structures which do not depend on the first 2 PCs.*

Class	Frequency
1	179
2	157
4	72
Missing	20

Table 2.2: *Recorded frequencies in the given dataset.*

2.2.2 Part (b)

Detecting **perfect** duplicates can be done by comparing all non-missing features of two samples. We only consider duplicates if we have a perfect agreement, excluding the labels. We then

address the pairs as follows - if the labels agree, we can simply remove either observation. Otherwise, if the labels disagree, we rewrite the entry with an undefined one, which can be addressed separately.

2.2.3 Part (c)

The obvious approach to the given problem is simply to delete rows with missing labels. However, losing data is unlikely to benefit the final model. Therefore, we discuss two methods which can be considered a better alternative:

- **Model-based imputation** - we construct a model to impute the missing labels by training it on the remaining information. Then we can predict the missing labels through this model and use them as the ground truth.
- **Semi-supervised learning** - captures many ideas of working with labelled and unlabelled data, some similar to the model-based imputation. The major difference is that some techniques focus on iteratively improving a single model, while others, such as co-training [2], use multiple models to enhance the dataset.

The latter requires a more complicated setup, meaning that it is more efficient for us to use the former method.

Model-based imputation works best if the data is missing not at random (MNAR). If the labels are missing at random (MAR), then we might not be able to well model the labels.

Definition 2.2.1 Some data is missing at random (MAR) if no clear pattern can be determined between the missing entries and the non-missing ones.
Contrary to that, we assume that data is missing not at random (MNAR) if the data is systematically related to unobserved data.

We can attempt to infer that by comparing the feature distributions of the missing data and the entire data. For that, we utilise the Kolmogorov-Smirnov test [3], defined as so:

Definition 2.2.2 For 2 samples with empirical cumulative distribution functions $F_{1,n}$, $F_{2,m}$, the Kolmogorov-Smirnov test statistic is defined as $D_{n,m} = \max_x |F_{1,n}(x) - F_{2,m}(x)|$. The corresponding p-value can be computed through a well-known distribution of said statistic.

For each feature, we measure the corresponding p-value, and we reject the hypothesis with a threshold of $\alpha = 5\%$. Three features in particular measure with a p-value lower than α - numbers 64, 343 and 420. However through visual verification, as seen in Figure 2.7, we cannot be certain this threshold is not passed only by random chance.
Therefore, we conclude our dataset has data that is likely MAR, however, we still have enough signal to use an imputation model so that no data is lost.

2.2.4 Part (d)

As our imputation model, we elect to use a Multinomial Logistic Regression, which can capture complicated relationships without much danger of overfitting. We also include an L1-regularization term to focus on the best-performing features. The results are summarized in Table 2.3 below.

Figure 2.7: *The feature distributions of the unlabeled data compared to the full dataset. It is difficult to make a concrete decision based on the given distributions due to the low sample size of only 20 points.*

Class	Original Frequency	Post-removal frequency	Post-Imputation Frequency
1	179	164	177
2	157	149	163
4	72	65	68
Missing	20	30	0

Table 2.3: *Recorded class frequencies after imputation in the given dataset. Notice that after the duplicates are removed, the frequency distribution is quite different to the original, with 15 samples of class 1 removed, in contrast to 8 for class 2. We will hence assume the second dataset as the one we compare the imputation to.*

The classes seem to be well represented, in comparison to the post-duplicate removal dataset, meaning that we can sustain our MAR assumption.

2.3 Q3 - Dataset C: Missing Data and Outliers

2.3.1 Part (a)

We can calculate that 55 data points are missing, with them being uniformly distributed between:

- **Samples** 138, 143, 231, 263 and 389.
- **Features** 58, 142, 150, 233, 269, 299, 339, 355, 458, 466, 491.

2.3.2 Part (b)

There are 2 basic types of imputation - static or model-based. The former uses the same value to replace the missing data, i.e. the median or mean of the feature. The latter uses a predictive model to guess the missing values from the other observations. An example of this is `Scikit-learn`'s `KNNImputer`.

	Static	Model-based
Pros	Efficient to compute. Likely preserves the mean/median.	Produces more believable values. Learns from available information Valid for MAR data.
Cons	Likely to artificially lower the spread. True values differ heavily from one another.	Makes assumptions on the data distribution. Might be slow to compute.

Table 2.4: *Positive and negative aspects of each method.*

Another alternative is using **Multiple Imputation**. It uses multiple methods of imputation to produce several datasets. They are evaluated separately, and we can then obtain an average score, and confidence intervals in our evaluation.

2.3.3 Part (c)

For this project, we will restrict ourselves to the aforementioned `KNNImputer`. It has all the advantages of being model-based, while also being non-parametric. A small number of neighbours **(7)** was initially chosen to incorporate features of low-represented classes. Furthermore, it also preserves the data distributions, as shown in Figure 2.8.

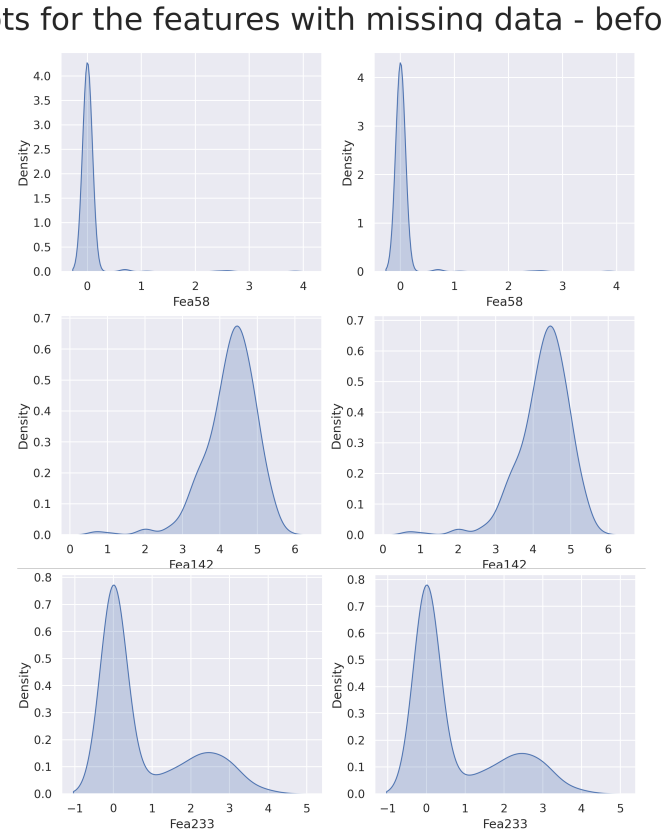


Figure 2.8: *Pre- and post-imputation distributions remain visually identical. The features were selected for demonstrative purposes.*

2.3.4 Part (d)

We can detect outliers using the Z-transform $z = \frac{x-\mu}{\sigma}$ to achieve a distribution with mean 0 and unit variance. If the feature is normally distributed, $P(|z| \geq 3) \approx 99.7\%$, meaning we can consider all values above that threshold outliers.

> **Remark** After dealing with the outliers through winsorization or removal, we need to invert the standardization, as the presence of the outliers has likely affected both the mean and the variance. Furthermore, the method is only viable on a per-feature level, meaning any point that is well-behaved within its features, but not in the full space will be considered an inlier.

Using the described method, we identify 2904 outlier values across many samples and features, which rules out removing the features/samples. If we are to use this detection method, we need to simply cap the values of the data.

2.3.5 Part (e)

As our model-based approach, we will use **Local Outlier Factor** detection. Similar to when performing K-Means clustering, we need to standardize the data, as the method uses distance-based similarity metrics. We then define a threshold above which a point will be classified as an outlier, and removed. For this task, we will set a static threshold of 1.5. This model-based approach is non-parametric, while also factoring the combination of all features, instead of considering them separately. The resulting distributions of some features can be seen in Figure 2.9.

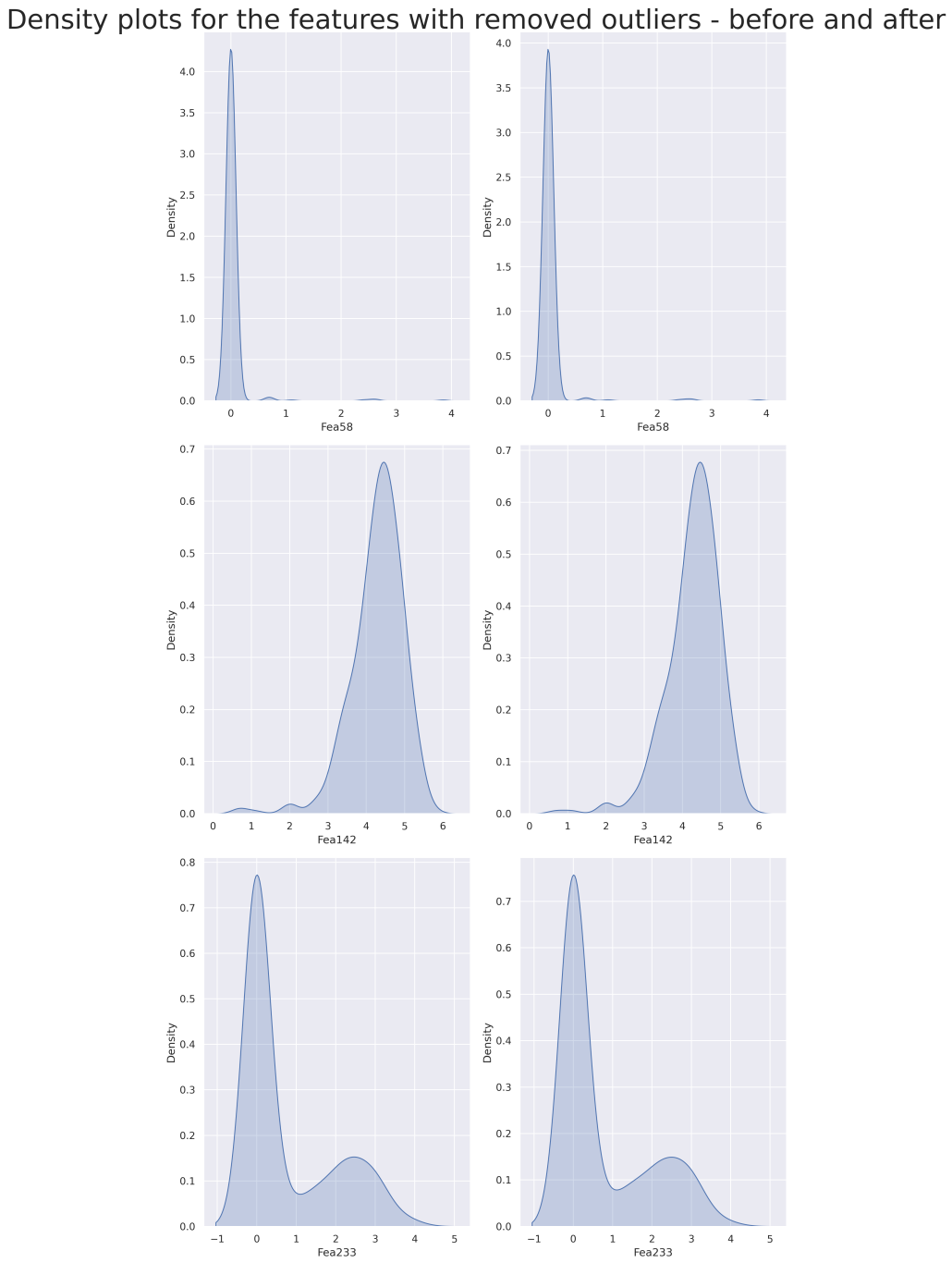


Figure 2.9: *Pre- and post-outlier removal distributions remain visually similar, despite 59 (14%) datapoints being removed. We notice that for Feature 58 (top), the small number of samples we might have considered outliers, have not been removed. This is likely because a single feature would not determine the state of being an outlier.*

Chapter 3

Section B

3.1 Q4 - Baseline Dataset: Supervised Learning and Random Forests

Throughout this section, we will discuss several classification approaches. The confusion matrices obtained from both the train and test sets can be viewed in the Appendix.

3.1.1 Part (a)

Tree models are a supervised method for classification and regression. A simple **Decision tree** uses nodes, which make decisions based on feature threshold. For a given data point, the tree makes a "decision" as to which node to head to based on the validity of the aforementioned thresholds. This process is repeated until a **leaf** node is reached, which holds a prediction - a probability/class for classification or a numerical value for regression.
Bootstrap aggregating, or **bagging** is an ensemble method that uses a collection of decision trees. First of all, N bootstrap samples are generated with replacement, and a decision tree is fit to this sample. Then, all N predictions are aggregated (via averaging or majority voting) to produce a final prediction.

A **random forest** is another ensemble method, which in addition to manipulating the sample entries, uses different sets of features. For each tree from the ensemble set, m out of n features are selected, and the tree is optimized for that particular subset. The final prediction is the average of the previous ones. Two relevant hyperparameters of a single tree within the forest are the **maximum depth**, limiting the growth of the tree, and the **number of features** for the particular tree. A heuristic that is commonly used is to select $m \approx \sqrt{n}$ features, as this would allow for a good combination of variability and information.

3.1.2 Part (b)

As a first step, we separate the data into a training and a test set with a size ratio of 4:1 in a stratified manner to preserve the class distribution. The preprocessing pipeline will be trained on the former, which includes finding the correct scaling factors. We will further utilise preprocessing techniques discussed in *Section A*. Initially, we noticed that there were no missing labels or data, and no duplicate entries. There are also relatively weak cross-correlations between the relevant features, as seen in Figure 3.1. However, we still have to take care of outliers, for which we use the `Local Outlier Factor` method, described in *Q3 (e)* with a constant threshold of 1.25. This number was chosen, as it removes less than 5% of the data and is stable (any lower value leads to significant data loss). Outliers were only removed from the training set to keep

the test set realistic.

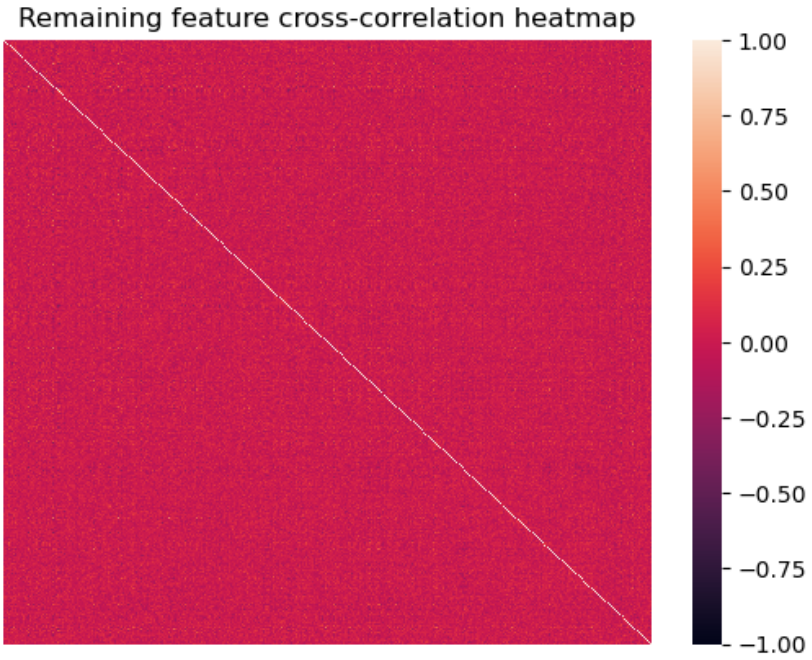


Figure 3.1: *Cross-correlation of preprocessed features. The diagonal shows the correlation of identical features. If we ignore it, the minimum and maximum correlation values are -0.42 and 0.57 respectively.*

Additionally, we need to address features which have limited information, i.e. have **Near-Zero Variance** (NZV).

Definition 3.1.1 We detect NZV features using the metric $\frac{f_{mode}}{N_{samples}}$, where f_{mode} is the frequency of the most common value of the feature. We can remove any features for which $\frac{f_{mode}}{N_{samples}} > t$ for some predefined threshold t.

Remark If we set $t = 0.9$, we remove **587** features, meaning more than half the features have little-to-no deviation.

We can then include t in our hyperparameter search if needed. Following the procedure we are left with a dataset of 477 samples with 413 features. Finally, we scale all features using the **Z-transform**, with the mean and variance derived from the training set.

3.1.3 Part (c)

Using the standard procedure, we train the **Random Forest Classifier** (RFC) on the preprocessed training set and evaluate it on the test set. We will quote the accuracy, as it is equivalent to reporting the error rate. Even with the default configuration, the classifier generalizes well with an accuracy of 94% (an error rate of 6%).

3.1.4 Part (d)

We can optimize the RFC using a measurement called the **Out of Bag** (OOB) score, which is the accuracy of the classifier aggregated across different bootstrap samples [4]. This score is a good but slightly conservative estimate of the test accuracy, as seen in Figure 3.2. We can

hence optimize the number of trees by iterating over a predefined range and using the highest OOB score, factoring in simplicity as well. We notice that we reach an optimal number of **190** trees with a test accuracy of 96%.

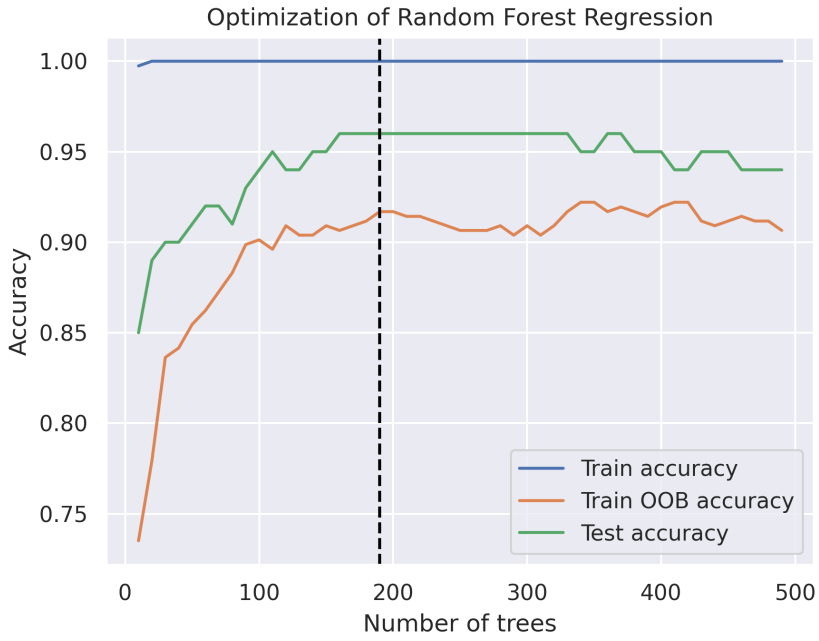


Figure 3.2: *Optimization of the random forest using OOB scores. We select the value of **190** trees due to it being the earliest point before the plateau in validation scores. Note how the training accuracy is always perfect, meaning we might be able to further improve the model through cross-validation.*

3.1.5 Part (e)

We measure the feature importance by using the corresponding metric after the RF fit. This uses only the information from the training set, so we also include a technique called **Permutation Importance** [5].

> **Definition 3.1.2 Permutation importance** is a technique used to estimate feature importance. It measures the decrease in performance when a feature is randomly shuffled.

With this in mind, we can identify the most important features, as seen in Figure 3.3 and refit the model. With the top seven features the model achieves an accuracy of 85% through OOB optimisation, as shown in Figure 3.4. Further grid search on relevant hyperparameters yields no improvement.

3.1.6 Part (f)

For this section, we will repeat the tasks using a **Support Vector Machine** (SVM). We reuse the same preprocessing pipeline, as the features have already been scaled. The model performs quite poorly with the default parameters, achieving an accuracy of 85%. Therefore, the model will be optimized through a grid search on the kernel and regularisation constant. We will also add the NZV threshold to the search, as part of the entire pipeline.

Our results also show that the optimum was found at an NZV threshold of $t = 0.9$, which is what we set as the default for the previous sections. We notice that even with extensive cross-validation, the SVM performs much worse with a limited amount of features, compared

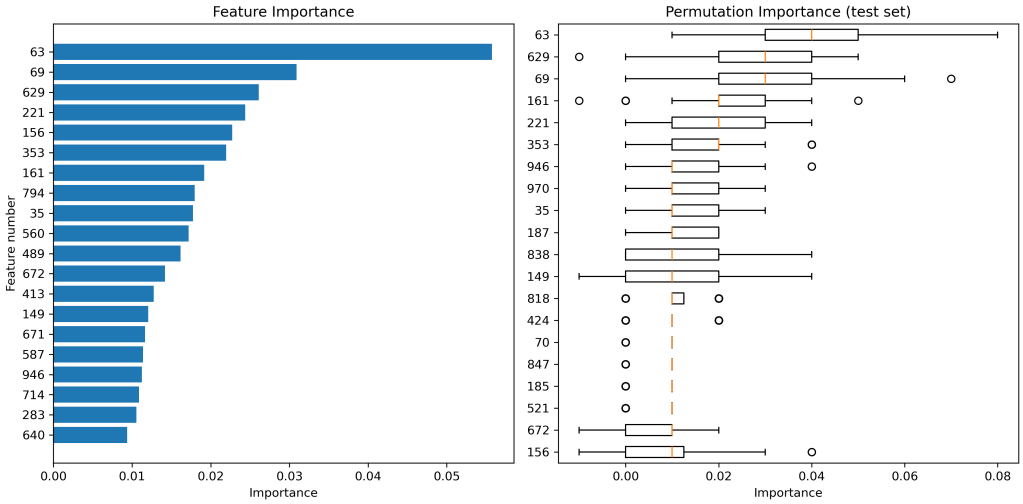


Figure 3.3: *Built-in feature importance (left) and permutation importance on the test set (right). We notice the same features show the most improvement in both cases, with small differences in placement. We select the top 7 permutation importance features as our best ones, namely features 63, 629, 69, 161, 221, 353, and 946 (0-indexing).*

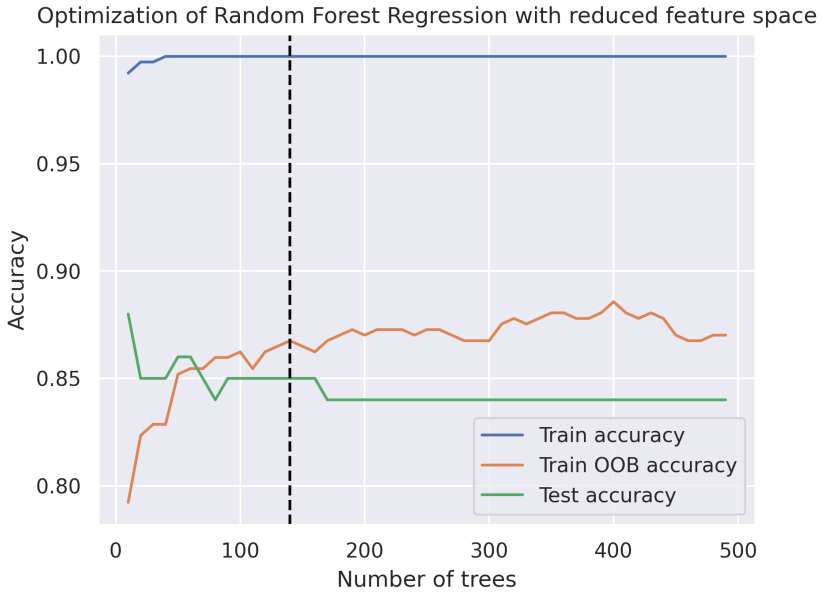


Figure 3.4: *Optimization of the random forest using OOB scores. We select the value of* ***140*** *trees due to it being the earliest point before the plateau in validation scores. The number of trees is lower than the one for the full dataset because we model less complex interactions.*

to the RFC. Using the same number of features as for the RFC, it yields an accuracy of 54%. Because the measured importances are much more balanced, as seen in Figure 3.5, the model will require a lot more information to perform well. Using the top 20 features results in an accuracy of 80%, still lower than the optimised RFC. It surpasses it only after using 30 features with an accuracy of 87%, much more than the original 7.

In conclusion, the RFC can be seen to be superior to the SVM even without rigorous cross-validation. The non-parametric nature of the model allows it more freedom to utilise the information from the features. This is also shown in how the RFC requires significantly fewer features to achieve the same accuracy as the SVM.

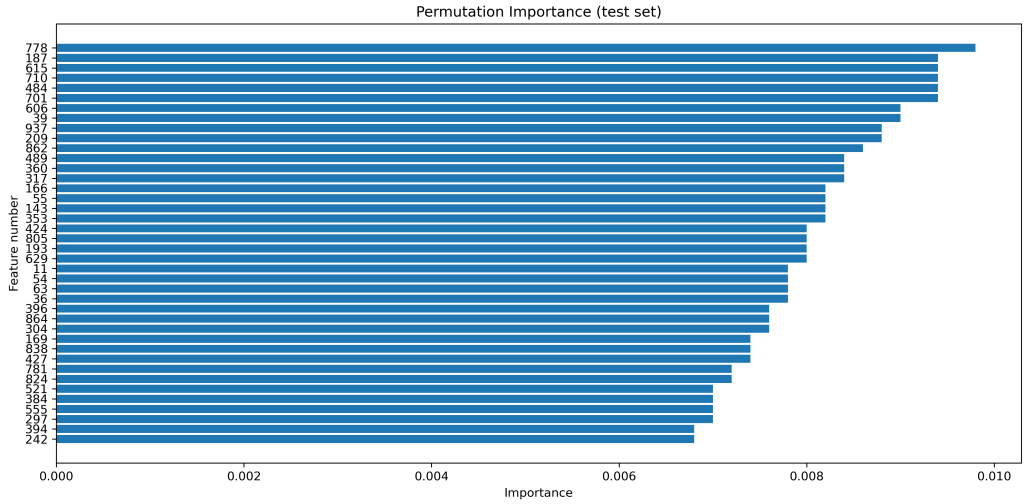


Figure 3.5: *The mean permutation importance falls off much slower, meaning many features are important to the model. Given this balance, we elect the top 20 features instead of 7. Also, note that there are significant differences in where the information is because of the model choice.*

3.2 Q5 - Baseline Dataset: Unsupervised Learning - Clustering

Before we start using the dataset, we perform the same preprocessing, as the one in *Q4*. However, we will not split the data into a train and test set, as we will not explicitly optimise the clustering algorithm.

3.2.1 Part (a)

The two techniques we will describe are the **K-means clustering** and **Hierarchical agglomerative clustering** (HAC). We have chosen them because of their simplicity, which is key in a small dataset. The former aims to minimize the mean of a distance metric from k cluster centroids to the predicted members of the clusters. In contrast, the latter builds up the clusters by merging the "nearest" ones. A major difference is that hierarchical clustering introduces a dependency between the clusters. On the other hand, there is no explicit relation between the k-means clusters, as the fit can be completely different depending on the initialization or the number of clusters.

To measure the optimal number of clusters, we will use the **Silhouette score** and the **Calinski-Harabasz index**[6]. In terms of both metrics and methods, we can see in Figure 3.6 that the optimal number of clusters is 2. We hence measure the agreement between the two clusters through a contingency table, as shown in Table 3.1.

3.2.2 Part (b)

After we obtain the class labels as our target variable, we perform the same preprocessing steps as detailed in **Q4 (b)**. For our model, we again use a **Random Forest Classifier**, for which we perform rigorous cross-validation on the number of trees, depth and number of features. We can then detect the most important features for each partition, as detailed in Figure 3.7.

We then use only the best features to reproduce the partitions. While for K-means clustering, we obtain a decent agreement with only the top 10 features, the HAC cannot mimic the results

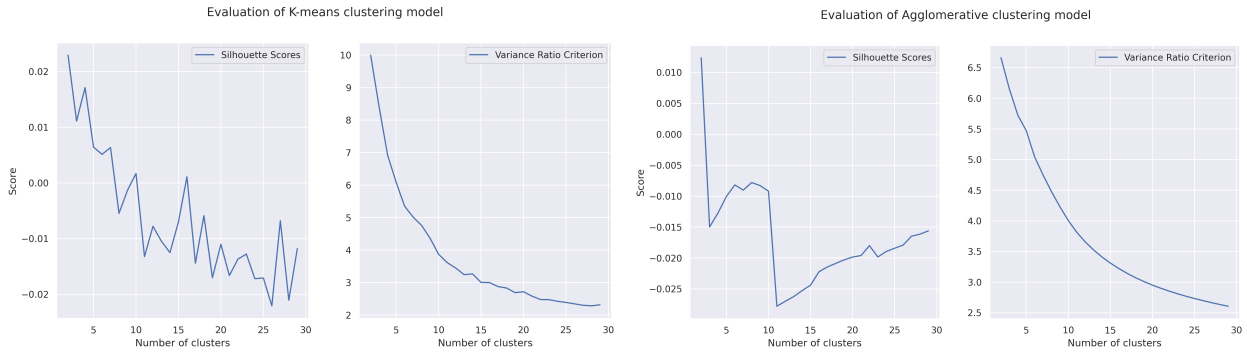


Figure 3.6: *We can see using every possible metric that both the K-means clustering and the HAC are best defined for 2 clusters. The stochastic nature of the Silhouette scores is likely due to noise in the data and clustering.*

Cluster(HAC)	**1**	**2**
Cluster 1 (KMC)	15	**160**
Cluster 2 (KMC)	**222**	80

Cluster(HAC)	**1**	**2**	**3**
Cluster 1 (KMC)	67	**112**	10
Cluster 2 (KMC)	53	19	**90**
Cluster 3 (KMC)	**120**	3	3

Cluster(HAC)	**1**	**2**	**3**	**4**
Cluster 1 (KMC)	**35**	4	1	2
Cluster 2 (KMC)	8	38	**98**	68
Cluster 3 (KMC)	45	15	0	**61**
Cluster 4 (KMC)	38	**57**	4	3

Table 3.1: *Contingency tables for different clustering settings. We can see that while agreement between the two methods is not perfect for 2 clusters, it is the best we can obtain, in comparison to the other 2 options.*

even with the top 100, as detailed in Table 3.2. What that shows is that for this particular dataset, K-means clustering is likely to be more robust.

3.2.3 Part (c)

We transform our preprocessed data using PCA with only 2 principal components. We can then plot the now 2-dimensional data with 3 different colourings, as seen in Figure 3.8 for K-means clustering, and in Figure 3.9 for HAC.

Cluster(fewer features)	KMC(10)		HAC(10)		HAC(100)	
	1	**2**	**1**	**2**	**1**	**2**
Cluster 1 (all features)	14	**161**	**225**	12	**215**	22
Cluster 2 (all features)	**283**	19	107	**133**	56	**184**

Table 3.2: *In the tables below we can see that K-means is stable even with the top 10 features (left). However, HAC performs much worse with 10 features(middle) and even with 100 (right).*

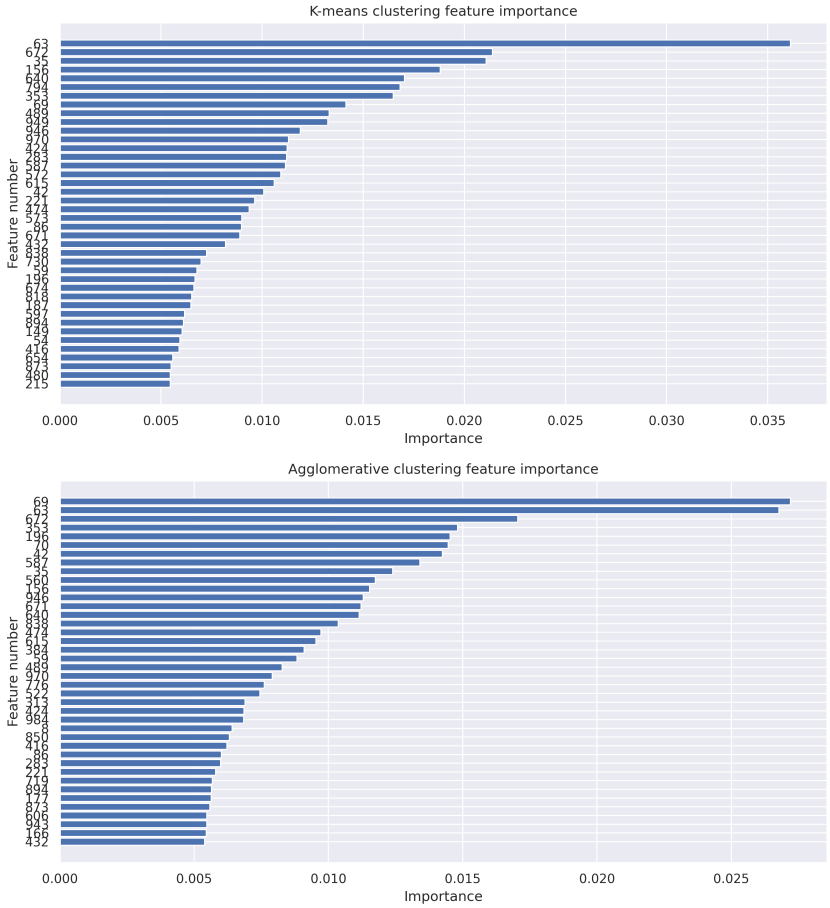


Figure 3.7: *We notice that the feature importance between the two methods is vastly different. This shows that the clustering is likely performed using different interpretations of the data.*

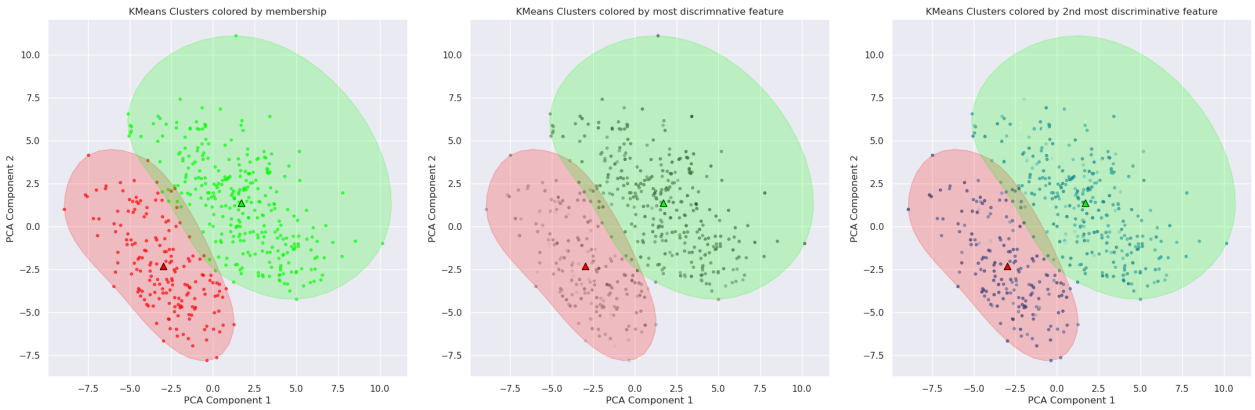


Figure 3.8: *K-means clusters are visually separable even using the PCA decomposition. It is further easily noticeable that both discriminatory features are predominantly high in one cluster while smaller in the other.*

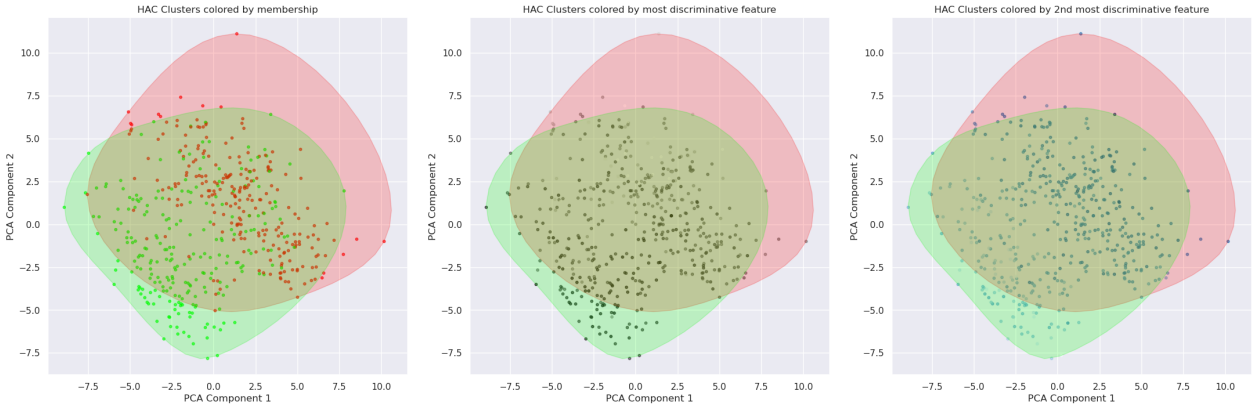


Figure 3.9: *HAC clusters are visually overlapping after doing PCA decomposition. It is therefore unclear how both discriminatory features differ between the clusters.*

Bibliography

[1] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, and Jonathan Taylor. *An Introduction to Statistical Learning: With Applications in Python*. Springer Nature, Cham, Switzerland, August 2023.

[2] Avrim Blum and Tom M. Mitchell. Combining labeled and unlabeled data with co-training. In Peter L. Bartlett and Yishay Mansour, editors, *COLT*, pages 92–100. ACM, 1998. ISBN 1-58113-057-0. URL http://dblp.uni-trier.de/db/conf/colt/colt1998.html#BlumM98.

[3] *Kolmogorov–Smirnov Test*, pages 283–287. Springer New York, New York, NY, 2008. ISBN 978-0-387-32833-1. doi: 10.1007/978-0-387-32833-1_214. URL https://doi.org/10.1007/978-0-387-32833-1_214.

[4] Wikipedia contributors. Out-of-bag error — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Out-of-bag_error&oldid=1106947153. [Online; accessed 23-December-2023].

[5] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[6] T. Caliński and J Harabasz. A dendrite method for cluster analysis. *Communications in Statistics*, 3(1):1–27, 1974. doi: 10.1080/03610927408827101. URL https://www.tandfonline.com/doi/abs/10.1080/03610927408827101.

Appendix

README.md

Appendix A

Applied Data Science - ivp24

This repository contains a solution for the M1 Applied Data Science coursework.

A.1 Table of contents

A.2 Requirements

The user should preferrably have a version of Docker installed in order to ensure correct setup of environments. If that is not possible, the user is recommended to have Conda installed, in order to set up the requirements. If Conda is also not available, make sure that the packages described in `environment.yml` are available and installed.

A.3 Setup

We provide two different set up mechanisms using either Docker or Conda. The former is recommended, as it ensures that the environment used is identical to the one used in the development in the project.

A.3.1 Using Docker

To correctly set up the environment, we utilise a Docker image. To build the image before creating the container, you can run.

```
docker build -t ivp24_ads .
```

[1] `#requirements`
[2] `#setup`
[3] `#running-the-tasks`
[4] `#frameworks`
[5] `#build-status`
[6] `#credits`

The setup image will also add the necessary pre-commit checks to your git repository, ensuring the commits work correctly. You need to have the repository cloned beforehand, otherwise no files will be into the working directory.

Afterwards, any time you want to use the code, you can launch a Docker container using:

`docker run --name <name> --rm -p 8888:8888 -ti ivp24_ads`

If you want to make changes to the repository, you would likely need to use your Git credentials. A safe way to load your SSH keys was to use the following command:

`docker run --name <name> --rm -p 8888:8888 -v <ssh folder on local machine>:/root/.ssh -ti ivp24_ads`

This copies your keys to the created container and you should be able to run all required git commands.

A.3.2 Using Conda

The primary concern when using Conda is to install the required packages. In this case **make sure to specify an environment name**. Otherwise, you risk overriding your base environment. Installation and activation can be done using the commands:

`conda env create --name <envname> -f environment.yml` `conda activate <envname>`

A.4 Running the tasks

Before running the tasks, put all the necessary data in the `/data` folder.

Each task has been solved in a separate Jupyter Notebook, found in the `/notebooks` folder. Notebooks were chosen instead of executables due to the vast amount of plots that needed to be presented.

To start Jupyter, simply run one of the following commands: - From a Docker container: `jupyter notebook --ip 0.0.0.0 --no-browser --allow-root`. After that, the command should print out URLs which can be pasted in a browser on the local system. - From a local terminal `jupyter notebok --ip=*`

The notebooks can then be executed sequentially. Note that Question 4 takes a long amount of time to run (15-20 minutes if the machine is overloaded).

A.5 Frameworks

The entire project was built on **Python** and uses the following packages: - For computation and data processing: - NumPy - Pandas - Machine Learning: - Scipy - Scikit-learn - For plotting: - matplotlib - For maintainability/documentation: - doxygen - pytest - pre-commit

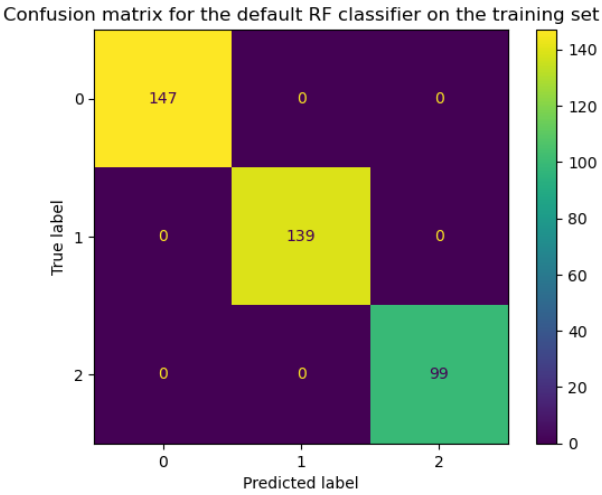
A.6 Build status

Currently, the build is complete and the program can be used to its full capacity.

A.7 Credits

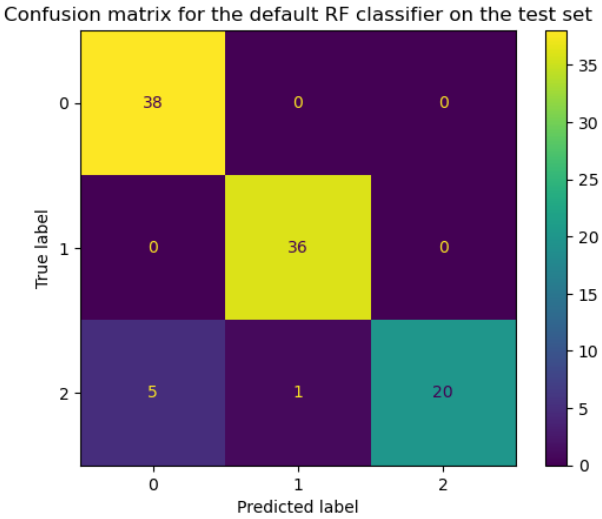
Plotting utilities were adapted from TowardsDataScience.

Confusion Matrices



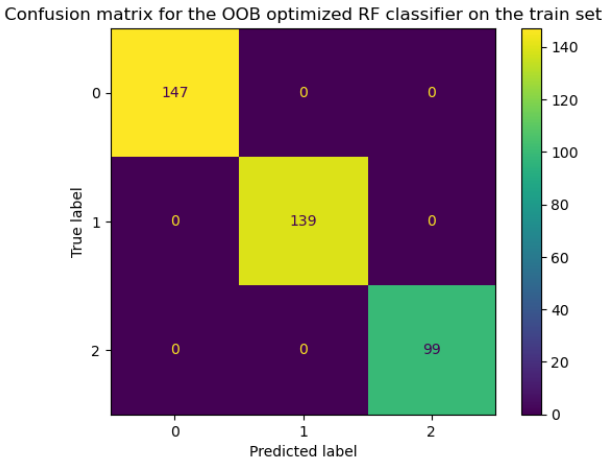
Class	Accuracy	Precision	Recall	F1-Score
1	100%	100%	100%	100%
2	100%	100%	100%	100%
3	100%	100%	100%	100%

Figure A.1: *Evaluation on the training set for default Random Forest Classifier.*



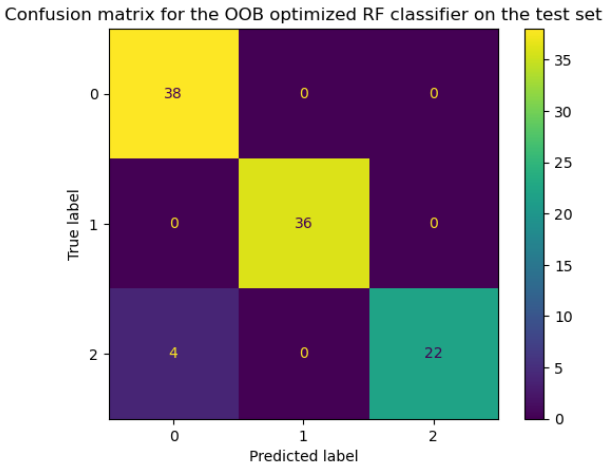
Class	Accuracy	Precision	Recall	F1-Score
1	95%	100%	88%	94%
2	99%	100%	97%	99%
3	94%	77%	100%	87%

Figure A.2: *Evaluation on the test set for default Random Forest Classifier.*



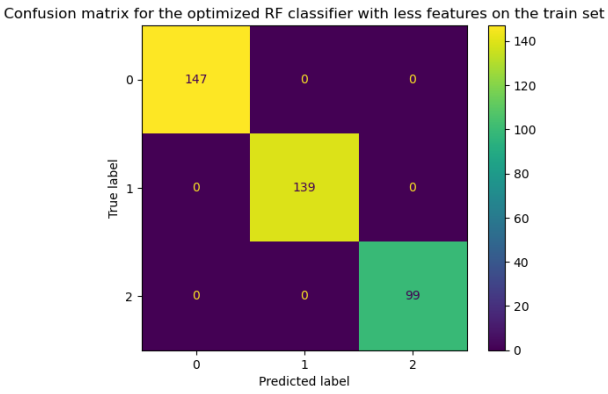
Class	Accuracy	Precision	Recall	F1-Score
1	100%	100%	100%	100%
2	100%	100%	100%	100%
3	100%	100%	100%	100%

Figure A.3: *Evaluation on the train set for the optimised Random Forest Classifier.*



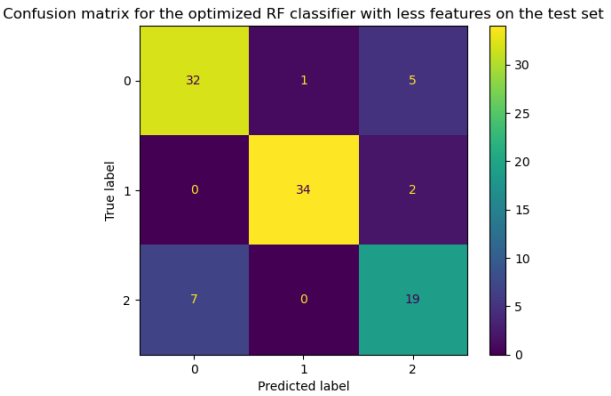
Class	Accuracy	Precision	Recall	F1-Score
1	96%	100%	90%	95%
2	100%	100%	100%	100%
3	96%	85%	100%	92%

Figure A.4: *Evaluation on the test set for the optimised Random Forest Classifier.*



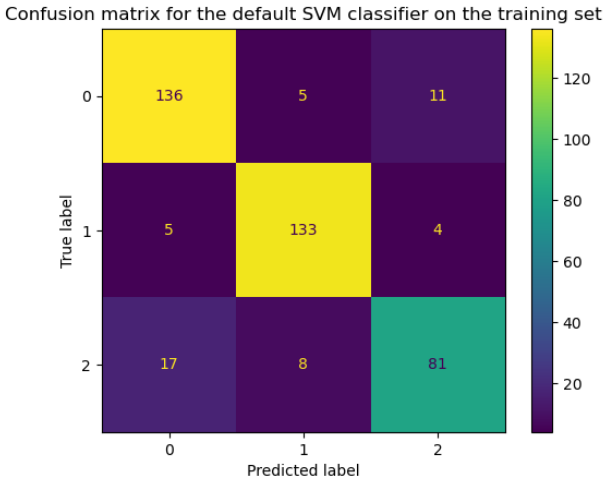
Class	Accuracy	Precision	Recall	F1-Score
1	100%	100%	100%	100%
2	100%	100%	100%	100%
3	100%	100%	100%	100%

Figure A.5: *Evaluation on the train set for the Random Forest Classifier with reduced features.*



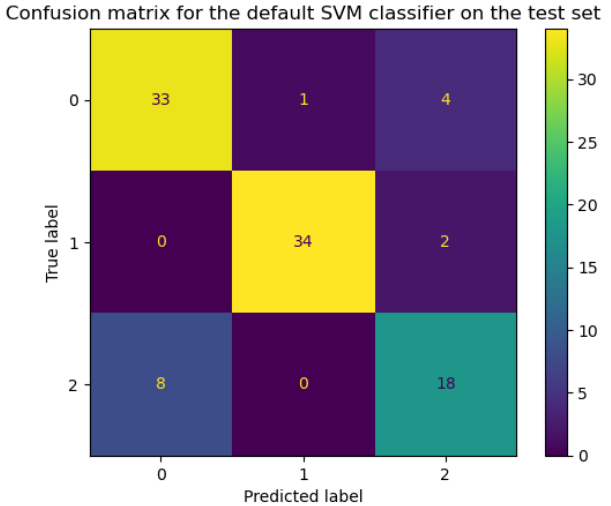
Class	Accuracy	Precision	Recall	F1-Score
1	87%	84%	82%	83%
2	97%	94%	97%	96%
3	86%	73%	73%	73%

Figure A.6: *Evaluation on the test set for the Random Forest Classifier with reduced features.*



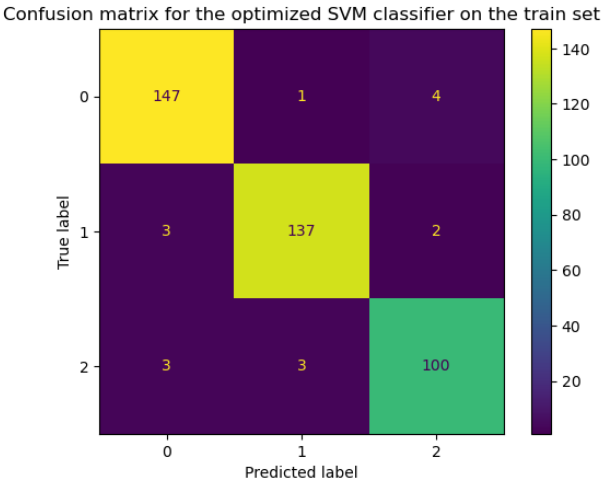
Class	Accuracy	Precision	Recall	F1-Score
1	91%	86%	89%	88%
2	95%	91%	94%	92%
3	90%	84%	76%	80%

Figure A.7: *Evaluation on the training set for default Support Vector Machine.*



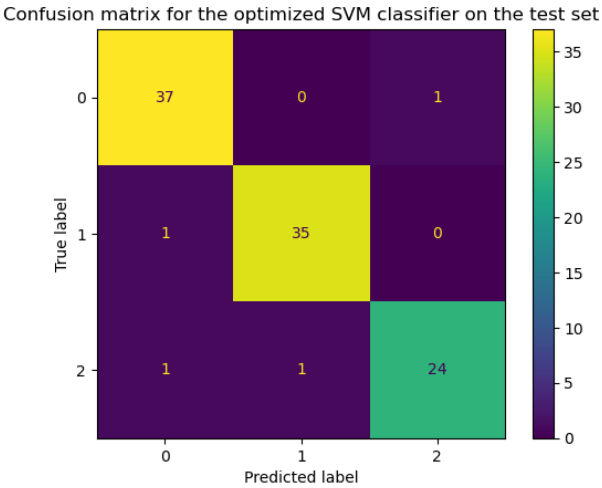
Class	Accuracy	Precision	Recall	F1-Score
1	87%	80%	87%	84%
2	97%	97%	94%	96%
3	86%	75%	69%	72%

Figure A.8: *Evaluation on the test set for the default Support Vector Machine.*



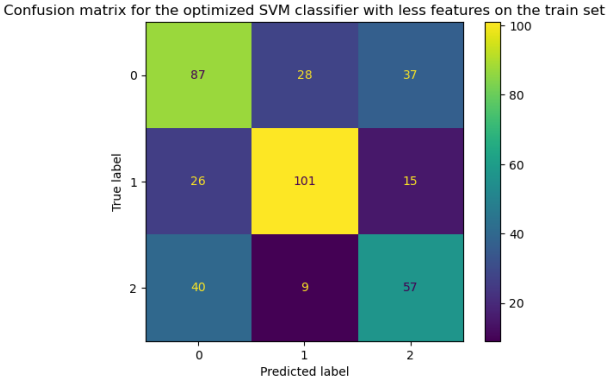
Class	Accuracy	Precision	Recall	F1-Score
1	97%	96%	97%	96%
2	98%	97%	96%	97%
3	97%	94%	94%	94%

Figure A.9: *Evaluation on the train set for the optimised Support Vector Machine.*



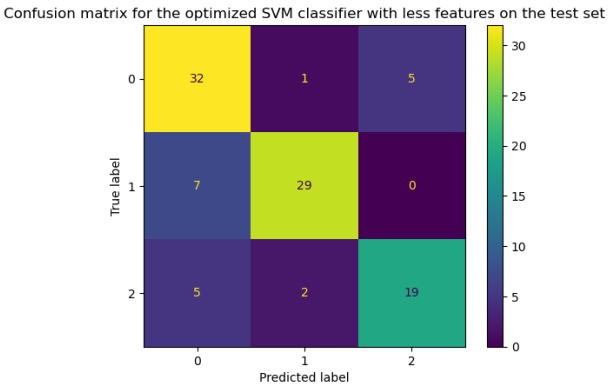
Class	Accuracy	Precision	Recall	F1-Score
1	97%	95%	97%	96%
2	98%	97%	97%	17%
3	97%	96%	92%	94%

Figure A.10: *Evaluation on the test set for the optimised Random Forest Classifier.*



Class	Accuracy	Precision	Recall	F1-Score
1	67%	57%	57%	57%
2	80%	73%	71%	72%
3	75%	52%	54%	53%

Figure A.11: *Evaluation on the train set for the Support Vector Machine with reduced features.*



Class	Accuracy	Precision	Recall	F1-Score
1	82%	73%	84%	78%
2	90%	91%	81%	85%
3	88%	79%	73%	76%

Figure A.12: *Evaluation on the test set for the Support Vector Machine with reduced features.*