UNIVERSITY OF CAMBRIDGE

DiS MPHIL

Applied Data Science

Coursework Submission

A REPORT PRESENTED

BY

STEVEN DILLMANN

Departments
Department of Applied Mathematics and Theoretical Physics
Department of Physics (Cavendish Laboratory)
Institute of Astronomy

Degree
MPhil Data Intensive Science

Module
M1 Applied Data Science

Supervision
Dr Steven Gratton
Tom Edinburgh

ST JOHN'S COLLEGE
UNIVERSITY OF CAMBRIDGE
17TH DECEMBER 2023

List of Figures

List of Tables

List of Listings

Contents

Total LaTeX Word Count: 2879/3000 (excluding Figures, Tables, Bibliography and Appendix)

1 Section A

1.1 Question 1: Exploration, Dimensionality Reduction and Clustering

1.1.1 Question 1 (a)

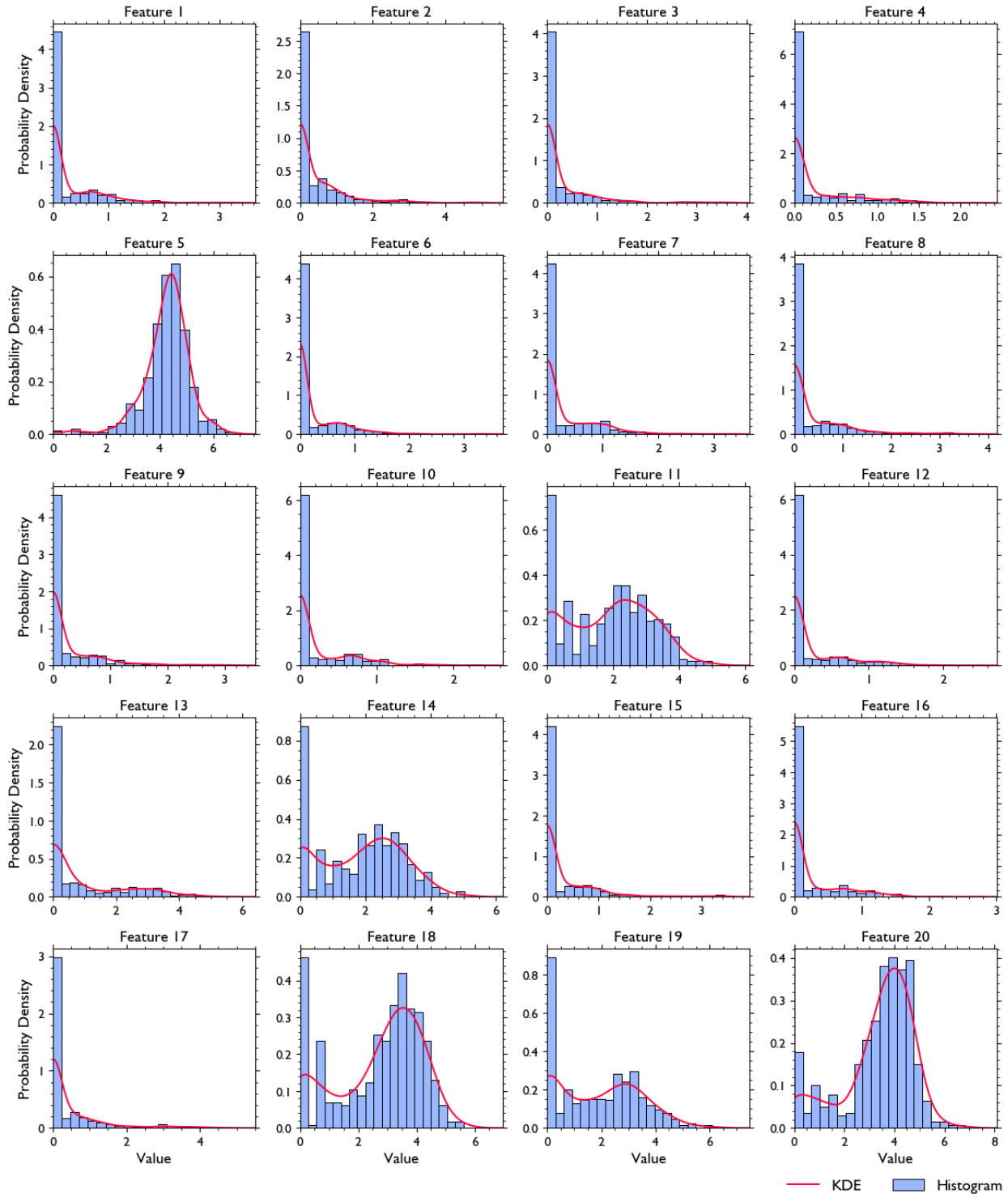


Figure 1: Density plots for the first 20 features of the `A_NoiseAdded.csv` dataset, i.e. normalised histograms with 20 bins overlayed by the kernel density estimates

Figure 1 shows selected feature density plots [2, 3] and their kernel density estimates [4]. Many features show zero-inflated distributions suggesting a sparse dataset. However, feature 5 follows a

Gaussian-like distribution. Features 11, 14, 18, 19 and 20 exhibit two different bumps, which might suggest that there are two distinct subpopulations in the data.

1.1.2 Question 1 (b)

Figure visualises the first two principal components from the principal component analysis (PCA) [5]. Figure 2 shows two distinct clusters varying strongly along the first PC. These may be the two distinct subpopulations explaining the two different bumps visible in some features in Figure 1.

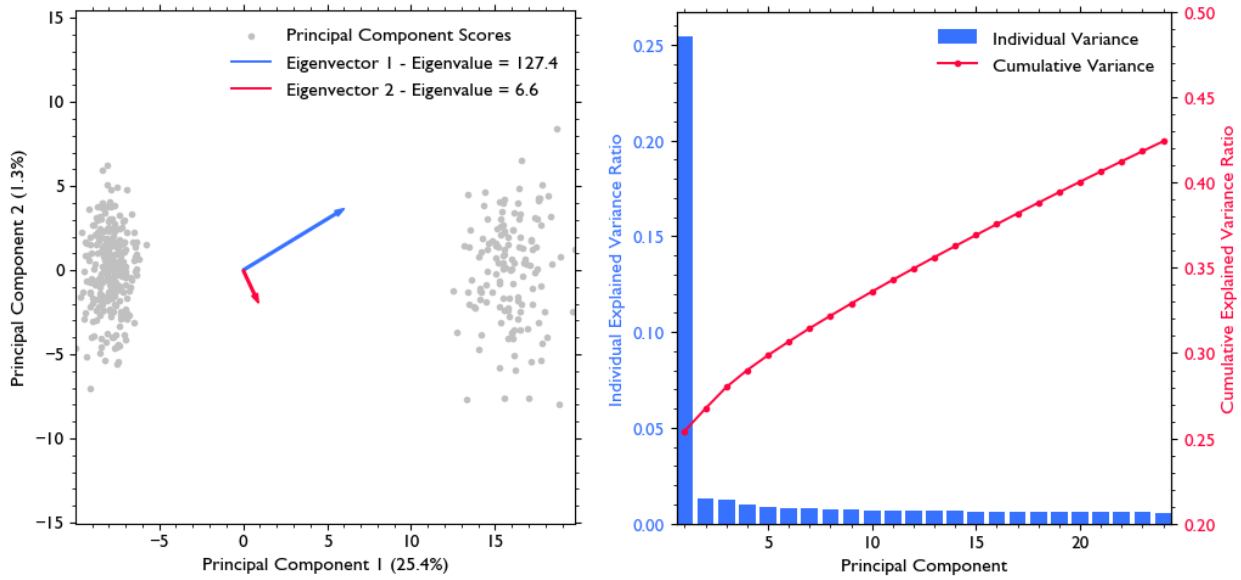


Figure 2: Two-dimensional principal component space of the features in the `A_NoiseAdded.csv` dataset (left) and the individual and cumulative explained variance ratios of the first 24 PCs (right). PCA is a dimensionality reduction technique that transforms a high-dimensional set of features to low-dimensional features while retaining important trends and patterns. The first PC covers a 25.4% of the variance explained, significantly more than all other PCs with the second covering only 1.3%.

1.1.3 Question 1 (c)

The low-dimensional PCs are used to cluster the dataset due to the reasons outlined in Question 1 (e). We split the data into two training sets, apply k-means clustering [6] to one and map the other onto the learned clusters (Model 1). We repeat the same reversed (Model 2). We run k-means with the default `scikit-learn` parameters [7, 8]). The k-means algorithm partitions the data into k clusters with k cluster centroids. Each data point is assigned to the nearest centroid. During training, Lloyd's algorithm [9] iteratively updates the centroid positions and cluster assignments by minimising the within-cluster sum-of-squares (WCSS) [10]. Once training is completed, the cluster centroids and assignments remain fixed. Unseen data can thus be mapped onto the learned clusters because during prediction k-means assigns each new data point to the nearest centroid learned from the training data.

We cluster the combined training set with both models, align their clusters with the function `centroid_matcher` [11] in Listing 1 for better comparison in Table 1.

The models agree on most assignments (diagonal entries in Table 1). However, there is a considerable number of off-diagonal entries representing the number disagreements. This is further discussed in Question 1 (d). We compute the Adjusted Rand Index (ARI) [12] to quantify the clustering agreement. The ARI of 0.51 suggests a moderate agreement.

```python
def centroid_matcher(centroids1, centroids2):
    """Matches centroids from two sets of clusters based on the minimum distance
    between them in a greedy one-to-one matching.

    Args:
        centroids1 (numpy.ndarray): Centroids from the first set of clusters.
        centroids2 (numpy.ndarray): Centroids from the second set of clusters.

    Returns:
        match_ix1 (list): List of indices of the matched centroids from the
        first set of clusters.
        match_ix2 (list): List of indices of the matched centroids from the
        second set of clusters.

    References:
        Marek Slipski, Armin Kleinboehl, Steven Dillmann, David M Kass, Jason
        Reimuller, Mark Wronkiewicz, and Gary Doran. The Cloudspotting on Mars
        Citizen Science Project: Seasonal and spatial cloud distributions
        observed by the mars climate sounder. Icarus, page 115777, 2023.
    """
    # Initialise lists to store the matched indices
    match_ix1 =[]
    match_ix2 =[]
    # Calculate the distance matrix between the centroids
    distances = distance_matrix(centroids1, centroids2)
    for index1 in range(len(centroids1)):
        # Find the index of the closest centroid in the second set of clusters
        index2 = np.argmin(distances[index1,:])
        # Append the indices to the lists of matched indices
        match_ix1.append(index1)
        match_ix2.append(index2)
        # Set the matched centroid distances to infinity to avoid rematching
        distances[index1,:] = np.inf
        distances[:,index2] = np.inf
    return match_ix1, match_ix2
```

Listing 1: Function that matches centroids from two sets of clusters based on the minimum distance between their centroids with a greedy one-to-one mapping. Different cluster labels from from two clustering results may refer to the same clusters. Therefore, we align the labels to better compare the clustering results.

Table 1: Contingency table comparing the clustering from Model 1 and 2 on the data for $k = 8$. There is a good agreement for most clusters. However, there is zero agreement for Cluster 6. The ARI is 0.51.

		Model 2 Clusters							
		0	1	2	3	4	5	6	7
Model 1 Clusters	0	56	0	0	0	0	0	50	0
	1	0	24	0	0	0	0	0	0
	2	0	0	29	0	0	0	0	1
	3	0	0	0	36	0	0	39	0
	4	0	0	0	0	25	0	0	0
	5	0	0	5	0	0	22	0	0
	6	43	0	0	0	23	0	0	0
	7	0	3	2	0	0	0	0	50

1.1.4 Question 1 (d)

Cluster Stability Analysis

The cluster assignments change considerably depending on the initialisation with the `random_state` parameter in Table 2. Different initialisations converge to different local optima, especially when the clusters are not well separated, which is the case for $k = 8$. The optimal number of clusters is $k = 2$ as it maximises a number of cluster quality scores in Figure 4. Choosing a higher k forces data points to be assigned to multiple separate subclusters, when they should be part of the same bigger cluster. Different random initialisations divide the space into different subclusters leading to a variety of different centroid positions in Figure 3. This also leads to a variety of different cluster sizes with some being very small, e.g. cluster 3 for random state 14.

Table 2: Contingency tables for different random state parameter models showing the instability of cluster assignments and variety of cluster sizes.

(a) Random State 14 vs. 24. The ARI is 0.48.

Random State 14 \ Random State 24	0	1	2	3	4	5	6	7
0	31	0	0	0	0	0	0	31
1	0	41	0	0	12	0	0	0
2	0	0	54	0	0	0	20	0
3	0	0	0	8	0	0	0	0
4	0	0	0	14	23	0	0	0
5	0	11	0	0	0	27	0	0
6	0	0	0	0	0	0	28	0
7	0	0	46	0	0	0	0	62

(b) Random State 15 vs. 25. The ARI is 0.44.

Random State 15 \ Random State 25	0	1	2	3	4	5	6	7
0	26	0	12	0	0	5	0	0
1	0	62	0	0	47	0	0	0
2	0	0	0	0	0	1	0	24
3	0	0	0	25	0	0	0	0
4	0	0	0	23	53	0	0	0
5	0	0	28	0	0	21	0	11
6	0	30	0	0	0	0	32	0
7	8	0	0	0	0	0	0	0

(c) Random State 16 vs. 26. The ARI is 0.53.

Random State 16 \ Random State 26	0	1	2	3	4	5	6	7
0	54	0	0	0	0	20	0	0
1	0	44	0	4	11	0	0	0
2	0	0	36	0	0	0	0	19
3	0	2	0	29	0	0	12	0
4	0	1	0	0	24	0	0	0
5	0	0	0	0	0	28	0	0
6	0	0	0	0	0	0	9	0
7	45	0	0	0	0	0	0	70

(d) Random State 17 vs. 27. The ARI is 0.47.

Random State 17 \ Random State 27	0	1	2	3	4	5	6	7
0	55	0	0	0	0	19	0	0
1	0	32	0	20	7	0	0	0
2	0	0	40	0	0	0	0	15
3	0	0	0	17	13	0	13	0
4	0	24	0	1	0	0	0	0
5	0	0	0	0	0	28	0	0
6	0	0	0	0	0	0	9	0
7	43	0	0	0	0	0	0	72

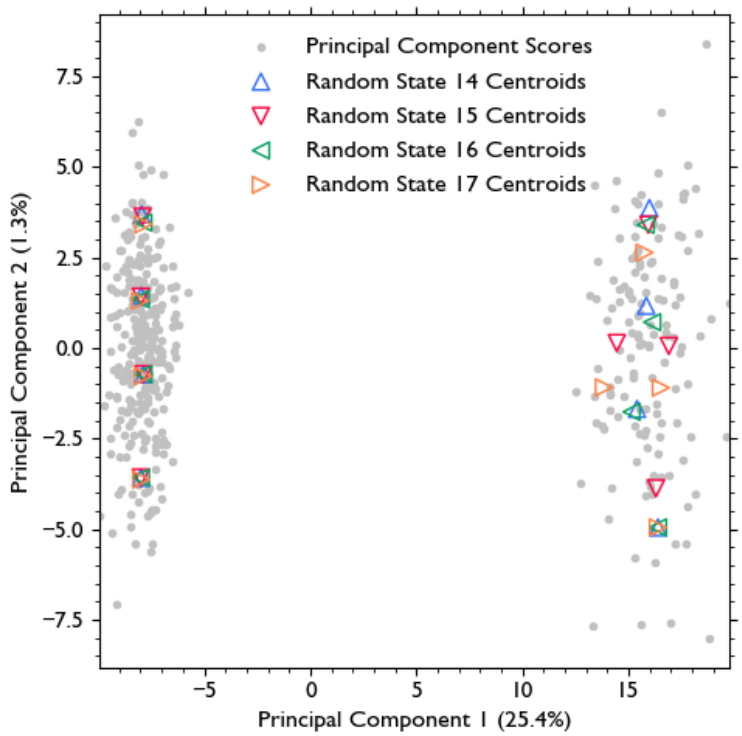


Figure 3: Cluster centroids from different random centroid initialisations in the principal component space.

Hyperparameter Optimisation for k-means

We repeat the clustering with two new models (Model 3 and 4) with $k = 2$. We find the optimal number of clusters with different cluster quality metrics [13, 14, 15] in Figure 4.

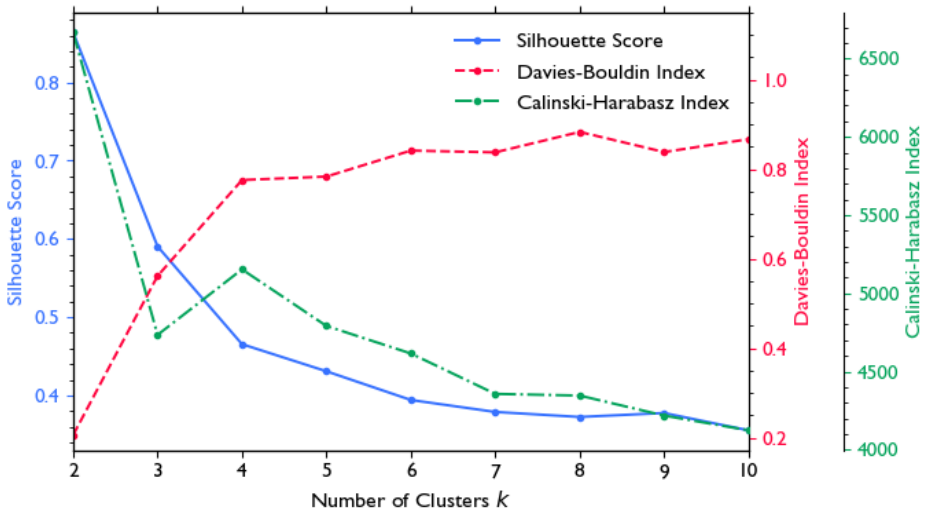


Figure 4: Different cluster quality scores for different number of clusters k. The Silhouette score measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation), which is to be maximised. The Calinski-Harabasz index is the ratio of between-clusters dispersion to within-cluster dispersion, which is to be maximised. The Davies-Bouldin index is the ratio of within-cluster distances to between-cluster distances, which is to be minimised. All scores give an optimal number of clusters of $k = 2$. This chosen k is also supported by the visual inspection of two distinct clusters in Question 1 (b).

Table 3 compares the cluster predictions on the combined training set for $k = 2$. The clusterings perfectly match (ARI of 1.0) and are stable for various initialisations because they are very well-separated for $k = 2$.

Table 3: Contingency table comparing the clustering from both models on the data for $k = 2$. The ARI is 1.0 for various random state initialisations.

		Model 4	
		0	1
Model 3	0	272	0
	1	0	136

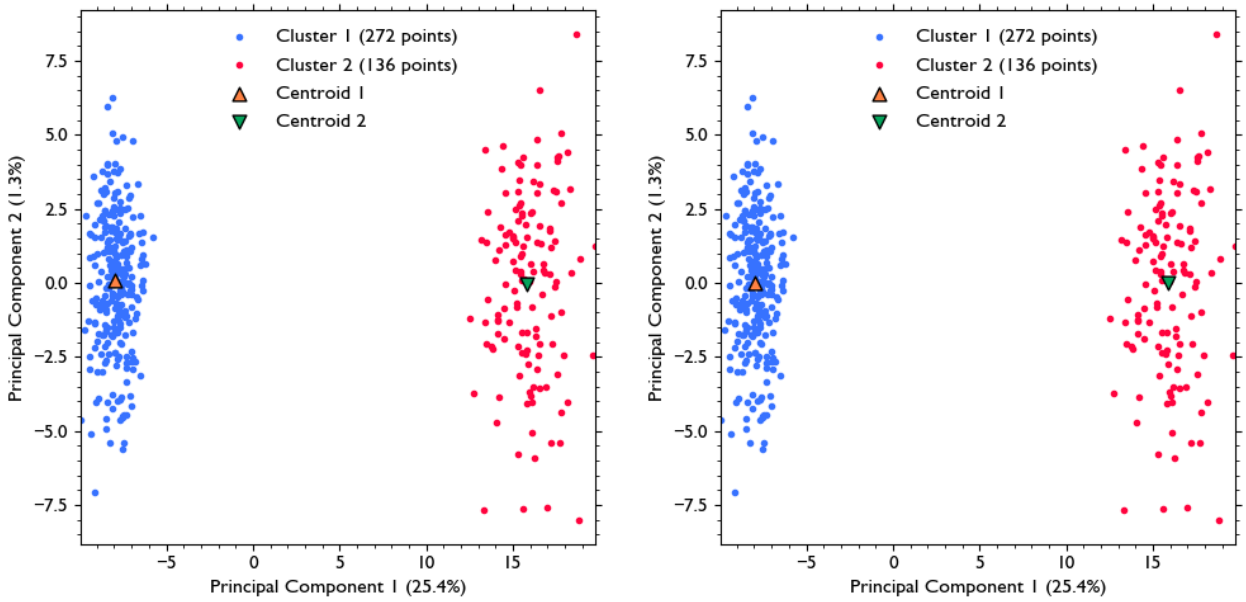
1.1.5 Question 1 (e)

We identify the clusters found in Question 1 (d) in Figure 5a. For comparison, we perform k-means clustering with $k = 2$ on the original dataset first and then apply PCA afterwards.

PCA followed by k-means vs. k-means followed by PCA

With 500 features, we are faced with the curse of dimensionality. High-dimensional datasets often suffer from increased sparsity, redundant or irrelevant features (e.g. the zero-inflated features) acting as noise in the data. This can negatively impact the performance of distance-based algorithm like k-means, because Eulidean distance calculations get disproportionately affected by zero-inflated features making the distances less meaningful. Using PCA before clustering effectively reduces the dimensionality while retaining most of the variability [16] making k-means more effective. It effectively reduces the noise, sparsity and redundancy in the data by extracting the most informative features [17, 18] resulting in more meaningful and well-separated clusters. It

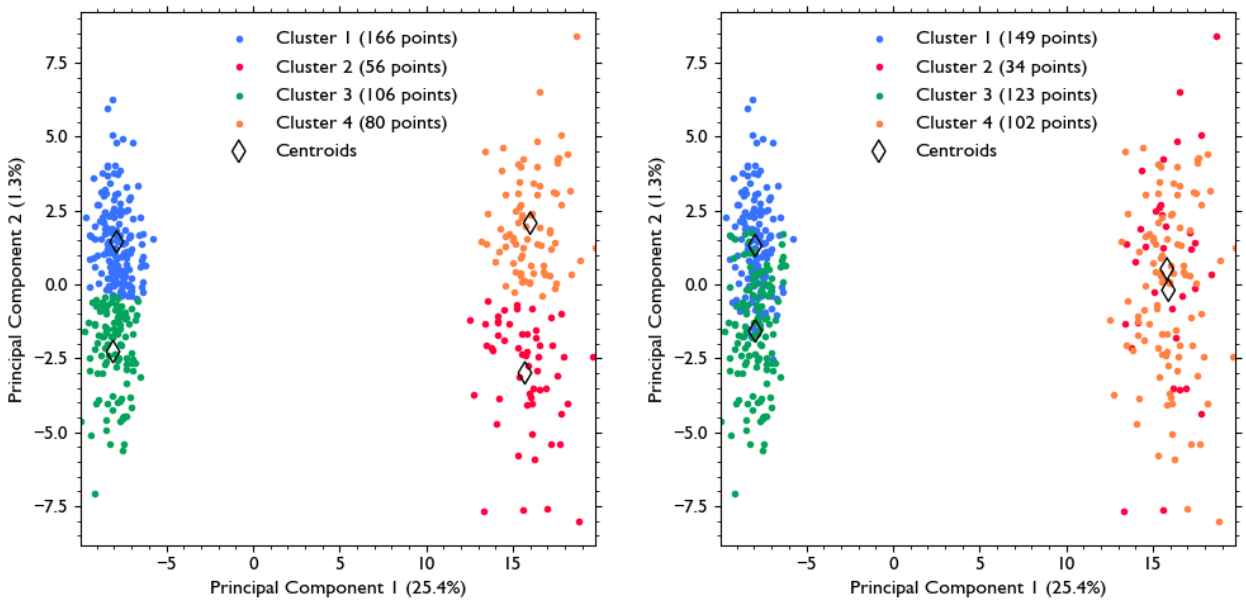
also helps increase the visualisation and interpretability. It is therefore generally recommended to use PCA first before applying k-means.



(a) PCA followed by k-means clustering. **(b)** k-means clustering followed by PCA.

Figure 5: Principal component space colour-coded by the cluster assignments for $k = 2$.

The difference between both approaches is not evident for $k = 2$ in Figure 5, because the data is very well separated into two subpopulations. Therefore, we repeat the comparison between both approaches for $k = 4$ in Figure 6. As expected, using PCA before k-means leads to more meaningful and well-separated clusters.



(a) PCA followed by k-means clustering. **(b)** k-means clustering followed by PCA.

Figure 6: Principal component space colour-coded by the cluster assignments for $k = 4$. All clusters in (a) show a good separation, while clusters 2 and 4 show very poor separation in (b) with the centroids being very close to each other.

1.2 Question 2: Missing Labels and Duplicated Observations

1.2.1 Question 2 (a)

The label frequencies in the dataset are summarised in Table 4.

Table 4: Absolute and relative frequency of labels in the `B_Relabelled.csv` dataset. There are 20 missing labels.

Label	Absolute Frequency	Relative Frequency
1	179	41.8%
2	157	36.7%
4	72	16.8%
No Label	20	4.7%
Total	428	

1.2.2 Question 2 (b)

Popular strategies to handle duplicates and their implications are outlined in Table 5.

Table 5: Strategies for handling duplicated samples in data and their associated problems.

Strategy	**Implications**
Identify and remove duplicates	May lead to a loss of valuable information and less data.
Keep the first or last duplicate entry	Sensible strategy if they are exact duplicates. However, when duplicates have conflicting labels or may be mislabelled, this strategy is not ideal.
Aggregate the duplicate entries	Sensible strategy if the label is a continuous variable (regression). However, this strategy is not suitable for categorical labels (classification).

Given the conflicting labels between duplicates and the possibility of wrong labels, we suggest a more involved strategy:

1. If duplicates have the same label, assume they are not mislabelled and drop one of them.
2. If they have conflicting labels, assume the duplicates may be mislabelled. Drop one duplicate and assign `NaN` as a preliminary label.
3. Use the prediction strategy outlined in Question 2 (d) for missing labels and replace the preliminary `NaN` labels with the predicted label (find k-nearest neighbours and use majority voting to predict the label).

Table 6 summarised the duplicated observations and assigned label. The label prediction strategy is further detailed in Question 2 (d).

Table 6: Summary of duplicated observations and their respective labels including the final assigned label from the duplicates handling strategy.

Duplicate 1	Duplicate 2	Label 1	Label 2	Preliminary Label	Final Label
Sample101	Sample30	2	4	NaN	**2**
Sample107	Sample46	1	1	1	1
Sample146	Sample74	1	1	1	1
Sample173	Sample100	4	1	NaN	**4**
Sample193	Sample44	1	1	1	1
Sample198	Sample83	1	4	NaN	**1**
Sample249	Sample188	1	1	1	1
Sample253	Sample66	1	1	1	1
Sample260	Sample28	1	2	NaN	**2**
Sample291	Sample220	2	4	NaN	**1**
Sample297	Sample117	4	4	4	4
Sample305	Sample210	1	4	NaN	**1**
Sample311	Sample175	2	2	2	2
Sample352	Sample351	2	1	NaN	**2**
Sample359	Sample120	1	2	NaN	**2**
Sample382	Sample119	4	4	4	4
Sample389	Sample344	1	1	1	1
Sample396	Sample384	2	1	NaN	**1**
Sample409	Sample147	1	2	NaN	**1**
Sample424	Sample166	1	1	1	1

1.2.3 Question 2 (c)

Handling observations with missing labels

Two approaches for dealing with missing labels are:

- **Imputation.** This involves filling in missing labels with the mean/median of the available data (static) or predicted labels from k-nearest neighbours or regression (model-based).
- **Omission.** This involves removing any observations with missing labels.

Their pros and cons are outlined in Table 7.

Missing at random (MAR) vs. Missing not at random (MNAR)

Let Y be a dataset with observed ($Y_{observed}$) and missing ($Y_{missing}$) values. Denote the missingness indicator R [19], where $R_{ij} = 1$ if Y_{ij} is observed (not missing) and $R_{ij} = 0$ if Y_{ij} is missing:

- **MAR.** The probability of missingness depends only on observed data, not on missing data [20], i.e.:
$$P(R_{ij} = 0|Y_{observed}, Y_{missing}) = P(R_{ij} = 0|Y_{observed})$$
- **MNAR.** The probability of missingness depends on missing data itself, not (only) the observed data [20], i.e.:
$$P(R_{ij} = 0|Y_{observed}, Y_{missing}) \neq P(R_{ij} = 0|Y_{observed})$$

For MAR, the missing data is related to observed factors, so we can account for the bias. For MNAR, the value of the missing data is related to the reason for it being missing [21].

Table 7: Pros and cons of imputation and omission to handle observations with missing labels.

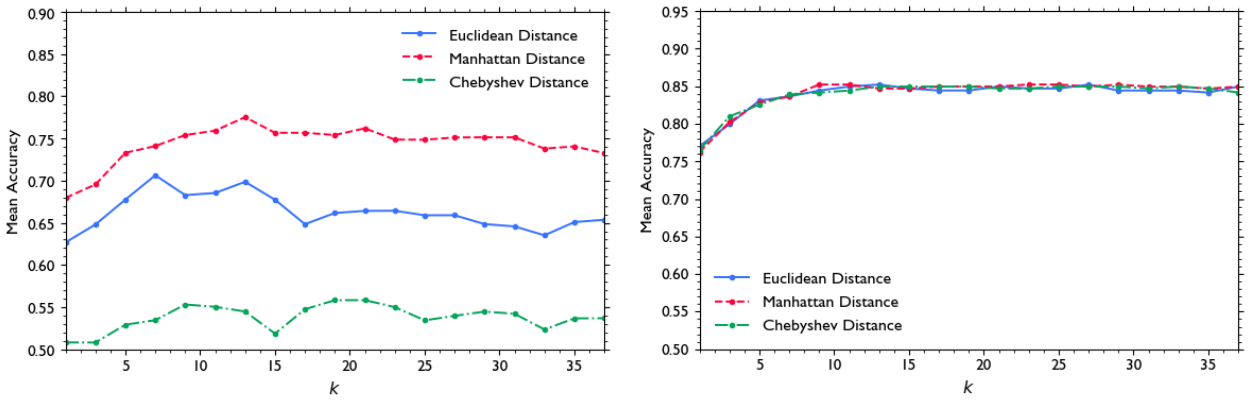
Approach	Pros	Cons
Imputation	*Maintains Dataset.* By enabling the use of a complete dataset, it avoids the loss of potentially informative observations. *Enhances Model Performance.* By enabling the use of a complete dataset, it can improve the training and performance of models.	*Complexity.* Some model-based imputation methods can be computationally expensive. *Bias.* Assumes that the labels are missing at random (MAR) and introduces a risk of skewing the dataset if it fails to correctly represent the true distributions.
Omission	*Simplicity.* Simply removing observations with missing labels is straightforward to implement. *No Assumptions on Missing Data.* When removing observations, no estimates of the missing values are made and no specific assumptions are made about the reasons for the missingness.	*Loss of Data:* Removing observations can lead to the loss of valuable information and a reduced dataset size. *Bias.* Removing observations with labels that are missing not at random (MNAR) can introduce bias into the model.

1.2.4 Question 2 (d)

We predict the missing labels in Table 4 and the additional ones from Question 2 (b) with the k-nearest neighbours (kNN) algorithm [22].

Dimensionality Reduction before kNN?

Considering the curse of dimensionality, we use observations without missing data to evaluate the performance between two approaches: (i) kNN on the original features or (ii) kNN on low-dimensional PCs. Given the class imbalances in Table 4, we use `StratifiedKFold` with 10 folds for cross-validation [23]. The performances of the kNN models are evaluated with different k and distance metrics in Figure 7.



(a) Performance of kNN on the original feature space. **(b)** Performance of kNN on the 2D PCA feature space.

Figure 7: Mean accuracy performances for different k and distance metrics.

For (i), the performances vary considerably across different distance metric with the highest accuracy being 78%. In approach (ii), the performances across different distance metrics are very similar with the highest accuracy of 85% achieved for the Manhattan distance and $k = 23$.

Hyperparameter Optimisation for PCA and kNN

We achieve a great performance improvement by applying kNN to only two PCs. We further optimise our model by finding the optimal combination of number of PCs n_{pca} and k with a grid-search style approach. We significantly improve our prediction performance with the highest accuracy of 91% achieved with the Manhattan distance, $n_{pca} = 7$ and $k = 3$.

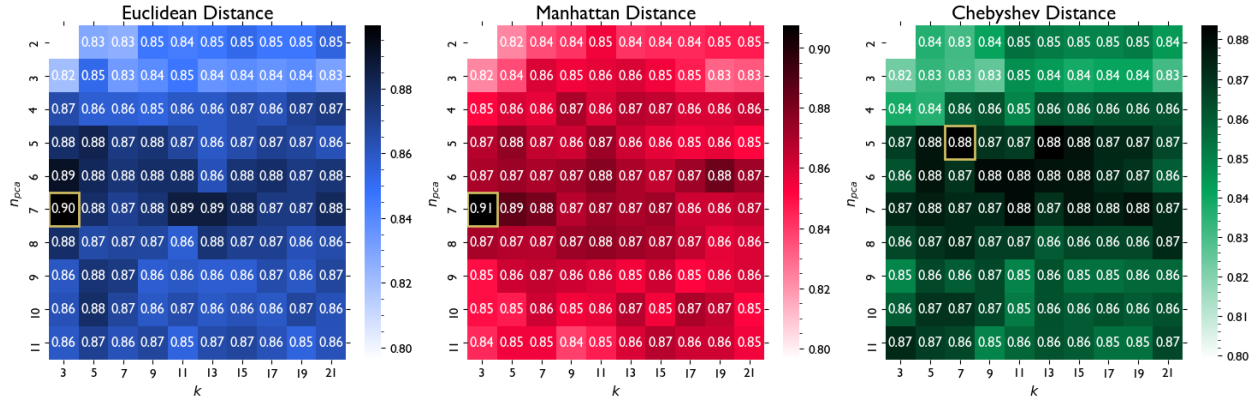


Figure 8: Mean accuracy for different hyperparameter combinations of n_{pca}, k and distance metrics. We only use odd values of k to break potential ties in the majority voting process of kNN [1].

Performance Evaluation (non-missing labels)

Using $n_{pca} = 7$ and $k = 3$ with the Manhattan distance, we apply the model on the full set of data with non-missing labels. The performance is summarised in Figure 9 and Table 21. We achieve an overall accuracy of 95%.

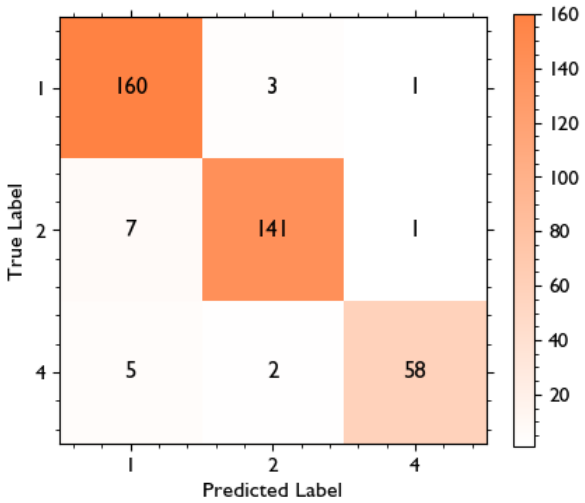


Figure 9: Confusion matrix for the classification prediction of the model on the non-missing labels data.

Table 8: Classification report for the classification prediction of the model on the non-missing labels data. The overall prediction accuracy of the model is 95%.

Class	Precision	Recall	F_1 Score	Support
1	0.93	0.98	0.95	164
2	0.97	0.95	0.96	149
4	0.97	0.89	0.93	65
Macro Average	0.95	0.94	0.95	378
Weighted Average	0.95	0.95	0.95	378

Classification Prediction (missing labels)

Using the optimised model, we predict the missing labels in our dataset. The results for the originally missing labels are shown in Table 9. The results for the duplicate samples with conflicting labels are highlighted in Table 6 in Question 2 (b).

Table 9: Predicted classification labels for the samples with missing labels.

Sample	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Prediction	1	2	1	2	4	4	2	1	2	1	2	1	1	2	1	1	2	2	2	1

The updated label frequencies for the final dataset are summarised in Table 10.

Table 10: Absolute and relative frequency of labels in the updated dataset.

Label	Absolute Frequency	Relative Frequency
1	178	43.6%
2	162	39.7%
4	68	16.7%
Total	408	

1.3 Question 3: Missing Data and Outliers

1.3.1 Question 3 (a)

There are 55 missing values (5 samples in 11 features) as summarised in Table 11.

Table 11: Features with missing values and the associated samples.

Affected Features	58	142	150	233	269	299	339	355	458	466	491
Affected Samples	138	143	231	263	389	(same across affected features)					

1.3.2 Question 3 (b)

Two approaches for imputing missing values are:

- **Mean/median/mode Imputation (static).** Fill in missing values with the mean/median (continuous data) or mode (categorical data) of the affected features across all samples.
- **kNN Imputation (model-based).** Find the k-nearest samples based on non-affected features and fill in missing values by averaging (continuous data) or majority voting (categorical data) the neighbouring values for the affected feature.

The advantages of using multiple imputation [24] are the following:

- **Accounts for Uncertainty:** It reflects the uncertainty of the imputation process by creating multiple datasets with different imputations.
- **Reduces Bias:** It averages results over multiple imputed datasets making its estimates more reliable and robust compared to single imputation.
- **Improves Accuracy:** It gives more accurate estimates than single imputation, especially when the missingness is complex.

1.3.3 Question 3 (c)

Imputer Model Selection

We choose the `KNNImputer` [21] over variations of the `IterativeImputer` [25] due to the following reasons:

- **Computational Cost:** The `KNNImputer` imputes the 55 missing values in < 0.1 s, while the default `IterativeImputer` takes ~ 2 min.
- **Hyperparameter Tuning and Simplicity:** Tuning `KNNImputer` only involves a single hyperparameter k. Tuning `IterativeImputer` requires testing different models like Bayesian Ridge or Random Forest Regression [26, 27] in addition to parameter tuning. Given the high computational cost, tuning the `IterativeImputer` becomes impractical.
- **Imputation Results:** The imputation results of our tuned `KNNImputer` are similar to the default `IterativeImputer` results.

Hyperparameter Optimisation

We tune the `KNNImputer` based on the strategy outlined in Table 12. The optimal hyperparameter is $k = 46$ from Figure 10.

Table 12: Hyperparameter optimisation strategy to find the optimal k for the `KNNImputer`.

Step	Description
Step 1	Exclude the samples with missing values and partition the dataset into $K = 10$ folds for cross-validation.
Step 2	In each fold, simulate missing values in the affected features, then impute these values using `KNNImputer` for a given k. Compute the Mean Squared Error (MSE) between the imputed values and the true values. Repeat this for each fold and average the MSE across all K folds.
Step 3	Repeat the process for a range of k values. Select the k with the lowest average MSE as the optimal hyperparameter.

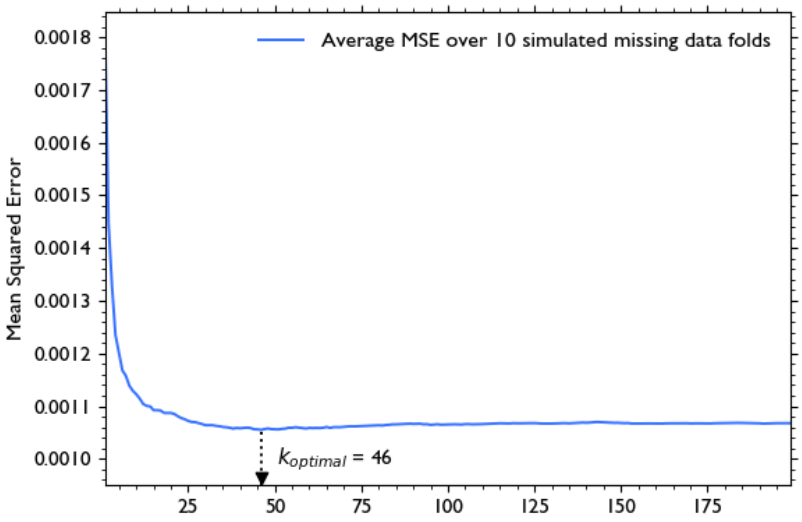


Figure 10: Average imputation MSE over 10 simulated missing data folds for different k.

Imputation Results

The imputation results are shown in Table 13. We compare the original and imputed distributions by calculating the p-value with the Kolmogorov-Smirnov (KS) test [28, 29] and in Figure 11. All

p-values are ~ 1.0, hence the imputation preserves the original distributions.

Table 13: Imputed values for the features and observations affected by missing values including the p-value from the KS test comparing the original and imputed distributions.

Feature	58	142	150	233	269	299	339	355	458	466	491
Sample138	0.000	4.216	0.039	0.725	0.166	0.000	1.913	0.000	0.913	0.200	0.000
Sample143	0.000	4.447	0.039	0.857	0.050	0.000	2.008	0.000	0.938	0.270	0.000
Sample231	0.015	4.204	0.015	0.793	0.241	0.000	2.034	0.000	1.033	0.238	0.000
Sample263	0.024	4.205	0.000	0.634	0.169	0.000	1.921	0.000	1.133	0.324	0.000
Sample389	0.015	4.397	0.000	0.490	0.159	0.000	2.100	0.000	0.988	0.223	0.000
p-value	~ 1.0	~ 1.0	~ 1.0	~ 1.0	~ 1.0	~ 1.0	~ 1.0	~ 1.0	~ 1.0	~ 1.0	~ 1.0

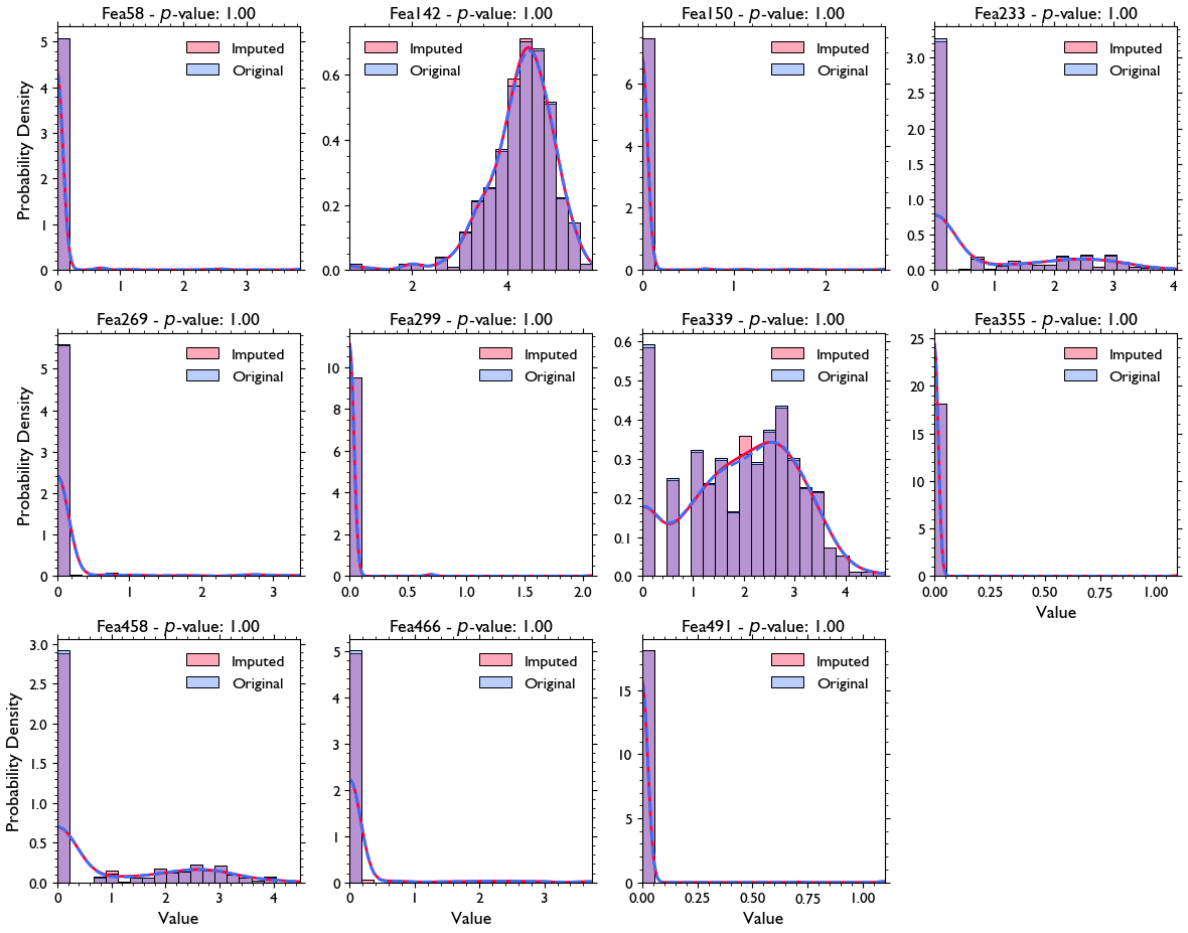


Figure 11: Density plots of the original and imputed distributions for the features affected by missing values.

1.3.4 Question 3 (d)

Standardisation Method Selection for Outlier Detection

We choose the Interquartile Range (IQR) method [30] for outlier detection due to the reasons outlined in Table 14.

Table 14: Justification for choosing the IQR Method for outlier detection.

Criterion	Justification
Robustness	The IQR method is more robust against extreme values as it focuses on the middle 50% of the data, reducing the influence of extreme outliers that can significantly affect calculations like the mean and standard deviation in the Z-score method.
Type of Distributions	Unlike the Z-score method the IQR method does not rely on the assumption of normal distribution. This makes it more suitable for our datasets that exhibit skewed or non-normal feature distributions.

The IQR method works in the following way:

1. Calculate the IQR with $IQR = Q3 - Q1$. $Q1$ represents the value below which 25% of the data falls. $Q3$ represents the value below which 75% of the data falls.
2. Compute the lower and upper bounds with $Q1 - threshold \cdot IQR$ and $Q3 + threshold \cdot IQR$.
3. Any data point below the lower or above the upper bound is considered an outlier.

We choose a conservative value of $threshold = 3$ to mitigate the issue of changing the distributions of potentially informative features too aggressively (low KS test p-values) in Question 3 (e).

Outlier Detection Results

Table 15 summarises the identified outliers.

Table 15: Total identified outliers from the IQR method in our dataset including the number of affected features and samples.

Total Outliers	Affected Features	Affected Samples
6042	365	408

1.3.5 Question 3 (e)

Outlier Correction Strategy

We use the `KNNImputer` for the imputation of outliers as outlined in Table 16. The outlier correction process is visualised in Figure 12.

Outlier Correction Results

We compare the original and outlier-corrected distributions with the KS test p-values for all features in Figure 13. We compare the before-and-after distributions for features 5 to 16 in Figure 14. Most features have a p-value ~ 1.0 suggesting that the outlier correction does not change most distributions of potentially informative features (e.g. 5, 11, 14). The features with lower p-values are zero-inflated features (e.g. 7, 8) that included some non-zero outliers before outlier-correction.

Table 16: Outlier correction strategy using the `KNNImputer`.

Step	Description	Justification
Step 1	Identify outliers using the IQR method from Question 3 (d) and impute the outlier values using the tuned `KNNImputer` from Question 3 (c).	We use the `KNNImputer` rather than the `IterativeImputer` due to the reasons outlined in Question 3 (c).
Step 2	Re-check for remaining outliers using the IQR method and impute them again. We ensure that only originally identified outliers are subject to imputation.	Recalculating the IQR may label new data points as outliers. The restriction to only impute originally identified outliers prevents loss of valuable information and reduces bias from the outlier correction method.
Step 3	Repeat Steps 1 and 2 until the total number of outliers converges.	Applying `KNNImputer` once leaves a small number of outliers, because the k-nearest neighbours of an outlier may include other outliers themselves. Hence, we introduce Step 2 and 3.

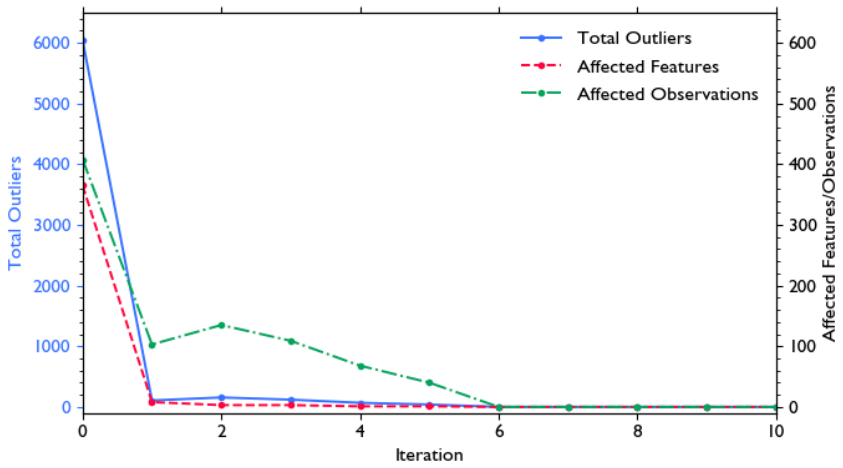


Figure 12: Number of total outliers, affected features and samples for different iterations in the outlier correction process. It converges to 0 outliers after 7 iterations.

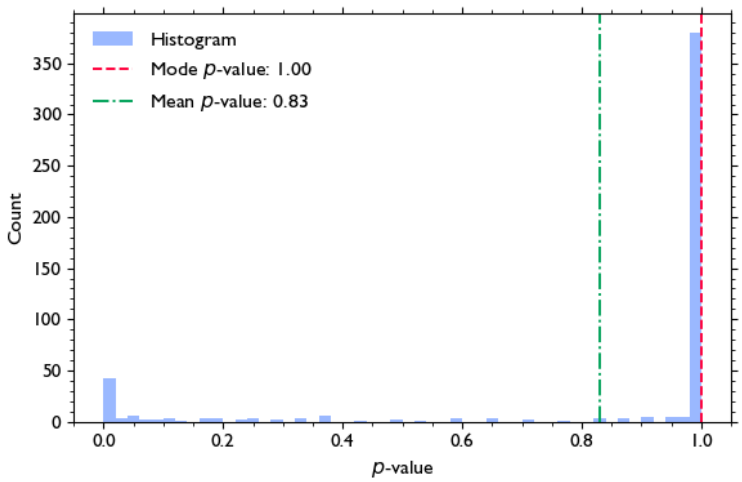


Figure 13: Distribution of p-values from the KS test comparing the original and outlier-corrected distributions

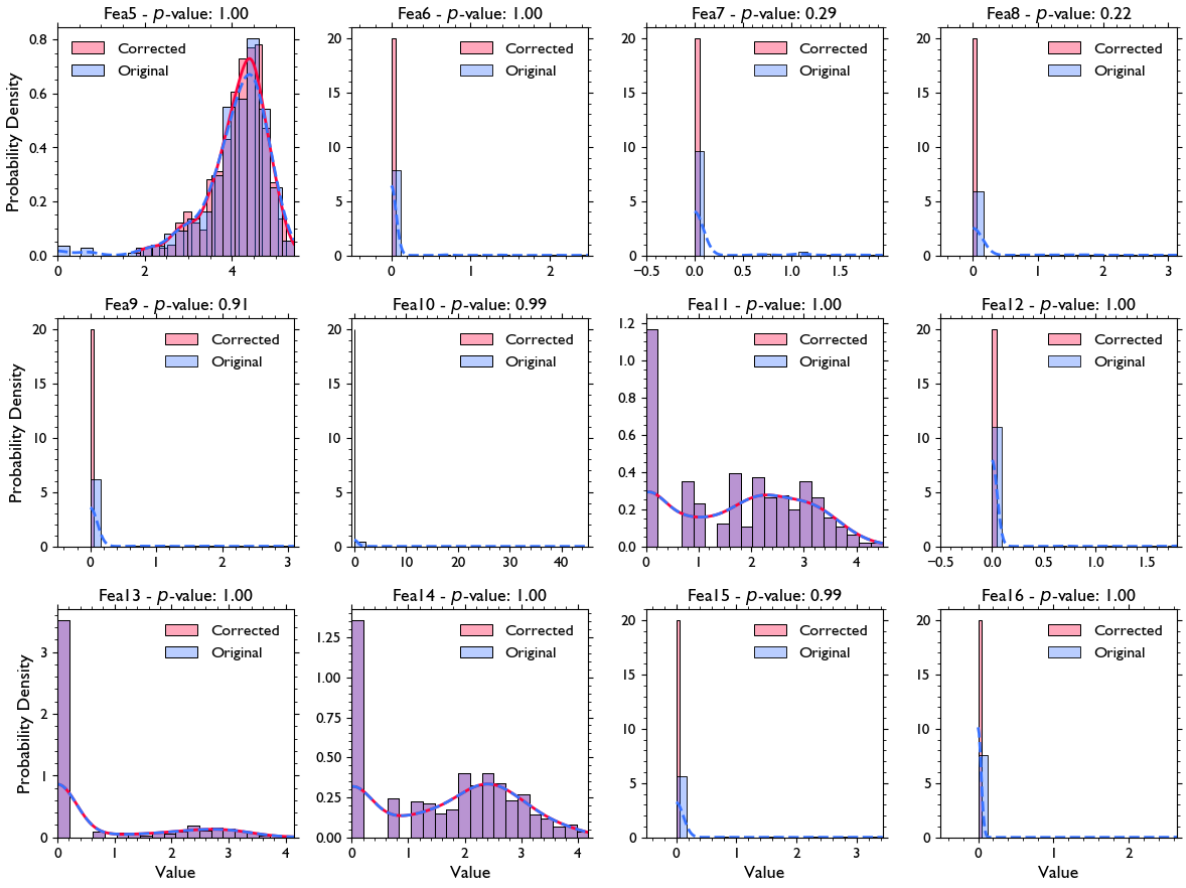


Figure 14: Density plots of the original and outlier-corrected distributions for features 5 to 16.

2 Section B

2.1 Question 4: Supervised Learning and Random Forests

2.1.1 Question 4 (a)

We highlight the differences between different tree-based classifiers [31, 32, 27] in Table 17.

Table 17: Differences between Decision Trees, Bagging and Random Forests for classification.

Aspect	Decision Tree	Bagging	Random Forest
Idea	Divides the feature space into separate segments. In each of these segments, every data point is assigned the same predicted class label based on majority voting [33].	An ensemble method that trains multiple decision trees on bootstrap samples giving multiple predictions. Majority voting across all predictions is then used to determine the class label.	An extension of bagging, where only a randomly selected subset of features is used for the partitioning at every node.
Pros	Simplicity and high interpretability.	Reduces variance of high-variance features [33], thus less prone to overfitting.	Reduces the correlation between individual trees and thus improves generalisation.
Cons	Prone to overfitting leading to high variance.	Individual trees may become highly correlated [33].	Lower interpretability.

The Gini index [34] helps decide which attribute to partition at each node in a classification tree and is given by

$$G = 1 - \sum_{j=1}^{n} p_{mj}^2$$

where $\hat{p}_{mj}$ is the proportion of samples belonging to class j in segment m. At each node, G is calculated for each resulting segment. These are then used to calculate a weighted average with respect to the segment size giving a measure of impurity after partitioning. The partition with the lowest weighted G is chosen as the best partition at that node.

Two hyperparameters for a RF single tree are:

- `max_depth`: Specifies how deep the tree can develop.
- `max_features`: Specifies the number of randomly selected features considered for partitioning at each node.

A heuristic is to use $\texttt{max\_features} = \sqrt{\text{number of features}}$ for classification problems, because this forces the RF to only use a random subset of features at each partition, reducing model variance and improving generalisation.

2.1.2 Question 4 (b)

Data Cleaning and Preprocessing

First, we apply the dataset checks in Table 18. Based on these results, we apply the preprocessing steps in Table 19.

Table 18: Dataset checks before preprocessing.

Check	Comment
Number of features	1000
Number of samples	500
Frequency of label 1	190 (38.0%)
Frequency of label 2	178 (35.6%)
Frequency of label 3	132 (26.4%)
Number of samples with missing label	0
Number of duplicate samples	0
Number of features with missing values	0
Number of non-numeric features	0
Number of highly correlated feature pairs (>0.9)	4
Number of near-zero variance features (<1%)	185
...of which are features with zero variance	42
...of which are features with all values being zero	42
Total number of outliers (IQR method from Question 3)	14182
Number of outlier-affected features	700
Number of outlier-affected samples	500

Table 19: Preprocessing steps and results.

Step	**Result**
1. Outlier correction with method from Question 3	Imputed 14182 outliers after 6 iterations.
...remaining total outliers	0
2. Re-check and drop any near-zero variance features	Removed 688 features.
...number of near-zero variance features	0
3. Re-check for remaining highly correlated features	None identified.
...remaining highly correlated feature pairs	0
Final number of features	312
Final number of samples	500

We later apply Random Forest (RF) and Adaboost [35] classifiers to the preprocessed dataset. The large amount of near-zero variance features are in fact features with all values being zero after the outlier correction. Removing outliers often helps reduce noise, improve robustness and model performance. However, creating this large amount of non-informative features should be carefully reviewed with respect to the impact on the predictive power of the classifier. We achieve performance improvements by correcting for outliers and dropping any remaining zero-inflated features for both classifiers. Removing such non-informative or any remaining highly correlated features helps enhance generalisation, reduce redundancy, increase computational efficiency.

Data Splitting

We split our dataset into a training (70%), validation (15%) and test (15%) set for training the Adaboost classifier. For the RF classifier, we use the internal out-of-bag (OOB) error [32] for validation, so we combine the training and validation set (85%) for training. Due to the slight imbalance of labels we use the `stratify` parameter to maintain the proportion of labels in splits.

2.1.3 Question 4 (c)

We apply both classifiers with the default hyperparameters and summarise their test set performances in Table 20 and Figure 15.

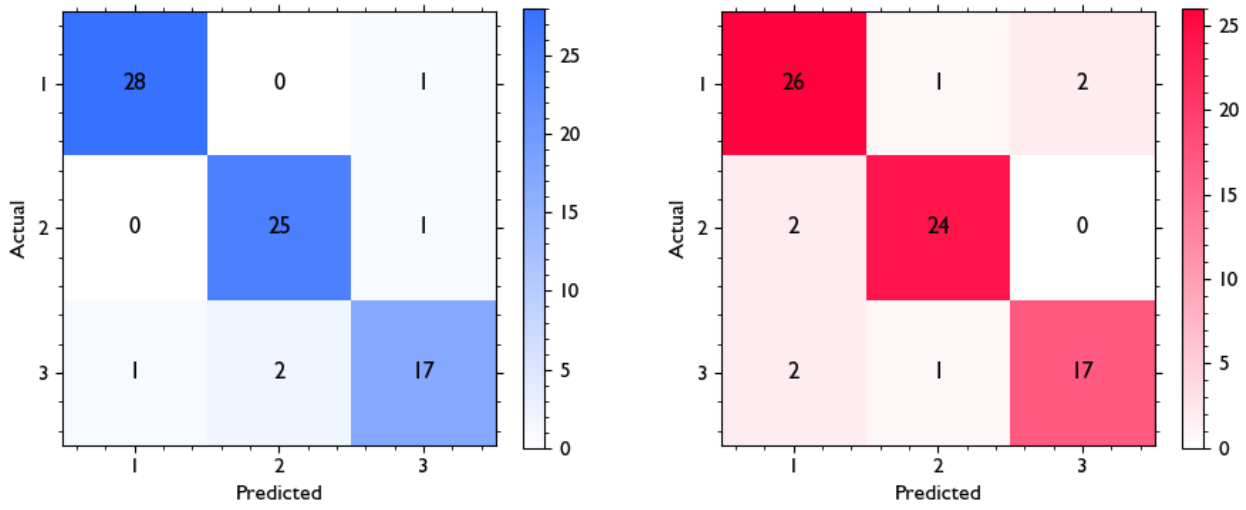
Table 20: Classification reports for the Random Forest and Adaboost classifiers with default hyperparameters based on the test set performance.

(a) Random Forest (default): Test set classification accuracy of 93% (classification error of 7%).

Class	Precision	Recall	F_1 Score	Support
1	0.97	0.97	0.97	29
2	0.93	0.96	0.94	26
3	0.89	0.85	0.87	20
Macro Average	0.93	0.93	0.93	75
Weighted Average	0.93	0.93	0.93	75

(b) Adaboost (default): Test set classification accuracy of 89% (classification error of 11%).

Class	Precision	Recall	F_1 Score	Support
1	0.87	0.90	0.88	29
2	0.92	0.92	0.92	26
3	0.89	0.85	0.87	20
Macro Average	0.89	0.89	0.89	75
Weighted Average	0.89	0.89	0.89	75



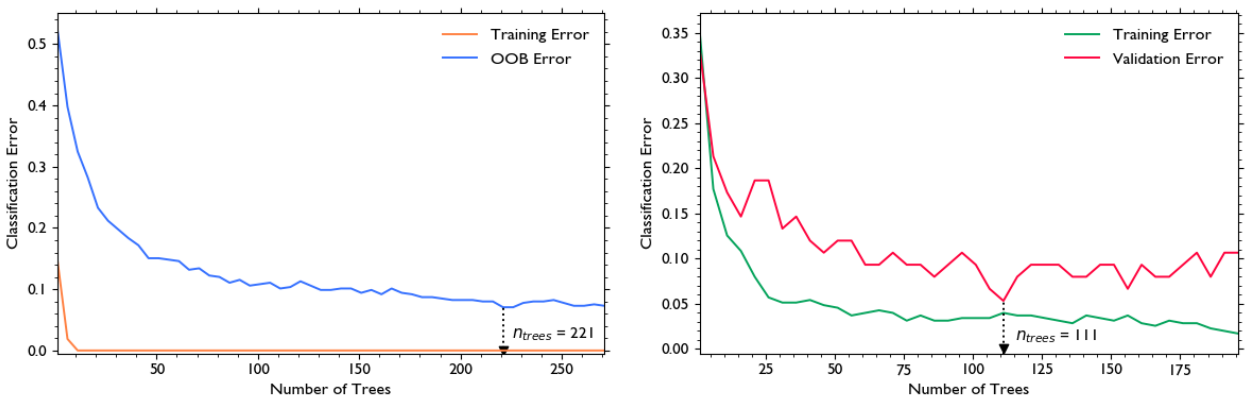
(a) Random Forest classifier predictions (default).

(b) Adaboost classifier predictions (default).

Figure 15: Confusion matrices for the Random Forest and Adaboost classifiers with default hyperparameters based on the test set performance.

2.1.4 Question 4 (d)

We optimise both classifiers with respect to the number of trees as shown in Figure 16. We apply both classifiers with the optimised hyperparameters and summarise their test set performances in Table 22 and Figure 20.



(a) Random Forest classifier training and OOB error.

(b) Adaboost classifier training and validation error.

Figure 16: Classification error of both classifiers for different number of trees.

Table 21: Classification reports for the Random Forest and Adaboost classifiers with optimised hyperparameters based on the test set performance.

(a) Random Forest (optimised): Test set classification accuracy of 95% (classification error of 5%).

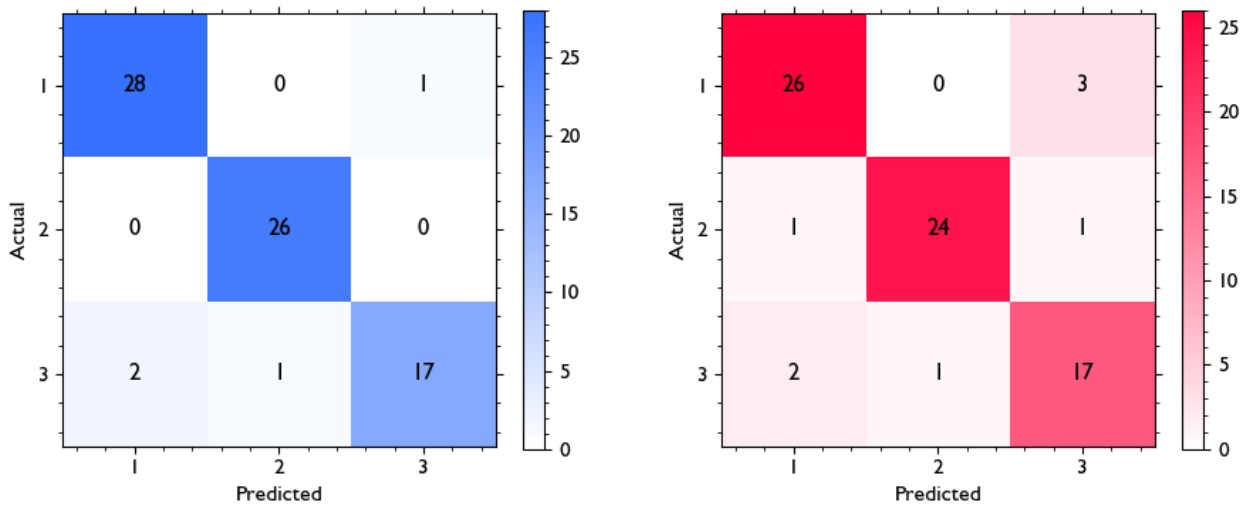
Class	Precision	Recall	F_1 Score	Support
1	0.93	0.97	0.95	29
2	0.96	1.00	0.98	26
3	0.94	0.85	0.89	20
Macro Average	0.95	0.94	0.94	75
Weighted Average	0.95	0.95	0.95	75

(b) Adaboost (optimised): Test set classification accuracy of 89% (classification error of 11%).

Class	Precision	Recall	F_1 Score	Support
1	0.90	0.90	0.90	29
2	0.96	0.92	0.94	26
3	0.81	0.85	0.83	20
Macro Average	0.89	0.89	0.89	75
Weighted Average	0.90	0.89	0.89	75

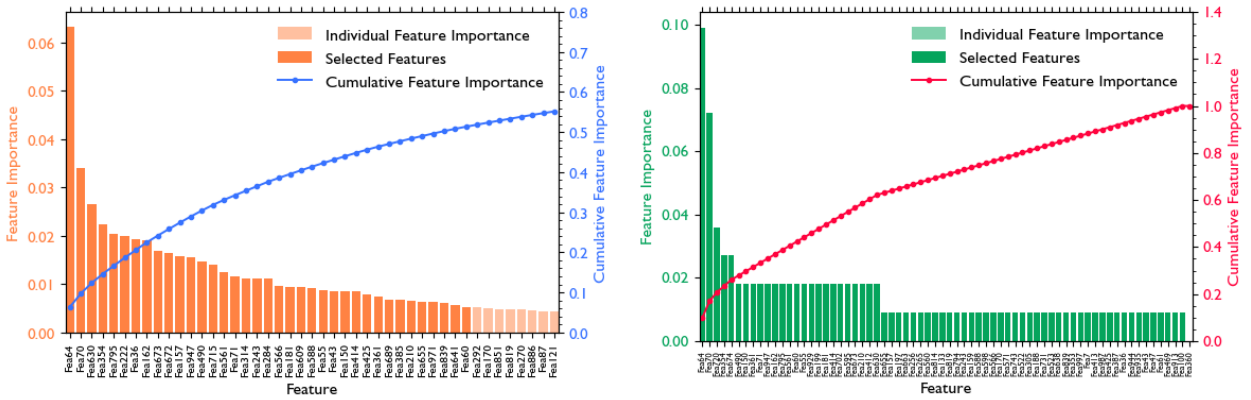
2.1.5 Question 4 (e)

We determine the feature importances from both models and list their importance in descending order in Figure 18. For RF, the feature importance relates to how much a feature reduces the impurity of the partitions it is used in (average across trees), hence higher feature importances are more important for the model prediction. For Adaboost, the feature importance is related to the contribution of each feature to each individual tree in the ensemble (weighted by the trees' performance). With the chosen subsets shown in Figure 18, we optimise the models in Figure 19.



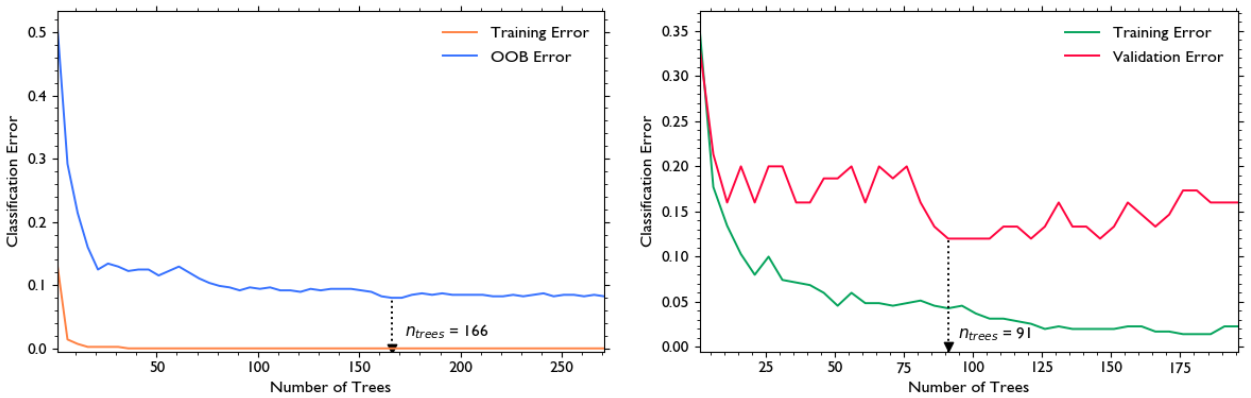
(a) Random Forest classifier predictions (optimised). **(b)** Adaboost classifier predictions (optimised).

Figure 17: Confusion matrices for the Random Forest and Adaboost classifiers with optimised hyperparameters based on the test set performance.



(a) Random Forest classifier feature importances. **(b)** Adaboost classifier feature importances.

Figure 18: Individual and cumulative feature importances for the Random Forest and Adaboost classifiers. The chosen subset of features is highlighted. The subset of the most important features to retrain the classifiers is chosen based on a knee point criterion for the RF classifier (select all features up to the point where the relative change in the cumulative importance curve is less than 1%). For the Adaboost classifier, most features have an importance of zero, so we choose all non-zero features.



(a) Random Forest classifier training and OOB error. **(b)** Adaboost classifier training and validation error.

Figure 19: Classification error for different number of trees (trained on most important features).

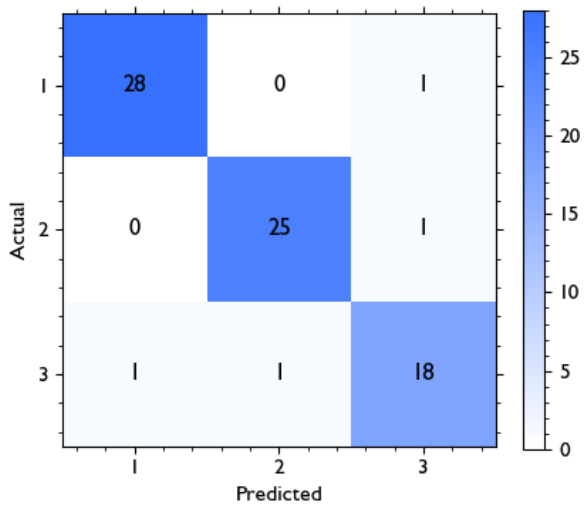
Table 22: Classification reports for the Random Forest and Adaboost classifiers with optimised hyperparameters based on the test set performance (trained on the most important features).

(a) Random Forest (important features, optimised): Test set classification accuracy of 95% (classification error of 5%).

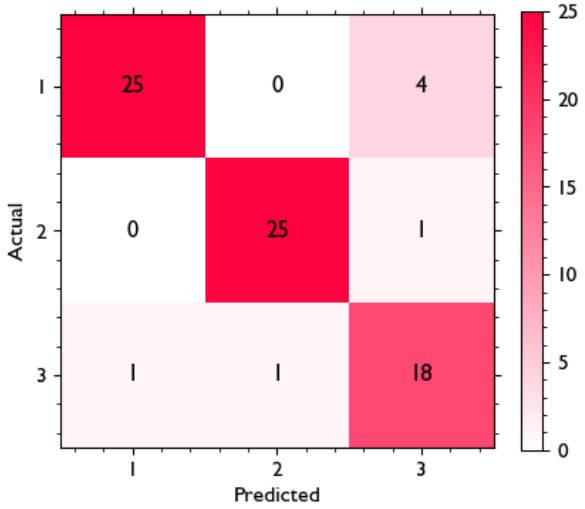
Class	Precision	Recall	F_1 Score	Support
1	0.97	0.97	0.97	29
2	0.96	0.96	0.96	26
3	0.90	0.90	0.90	20
Macro Average	0.94	0.94	0.94	75
Weighted Average	0.95	0.95	0.95	75

(b) Adaboost (important features, optimised): Test set classification accuracy of 91% (classification error of 9%).

Class	Precision	Recall	F_1 Score	Support
1	0.87	0.93	0.90	29
2	1.00	0.96	0.98	26
3	0.84	0.80	0.82	20
Macro Average	0.90	0.90	0.90	75
Weighted Average	0.91	0.91	0.91	75



(a) Random Forest classifier predictions (important features, optimised).



(b) Adaboost classifier predictions (important features, optimised).

Figure 20: Confusion matrices for the Random Forest and Adaboost classifiers with optimised hyperparameters based on the test set performance (trained on the most important features).

2.1.6 Question 4 (f)

The principle of a RF classifier is based on bagging as described in Question 4 (a). By contrast, AdaBoost is a boosting technique that combines multiple decision trees (weak learners) into a single strong learner but without bootstrapping. The idea is to train a sequence of learners that focus on correcting the residuals from previous learners thus reducing bias [33]. The results of the models are compared throughout Question 4 and summarised in Tables 23 and 24. Overall, the RF achieved higher test set accuracies in all cases.

Comparison for Question 4 (c) - Default Hyperparameters

The RF classifier performs better with default parameters than Adaboost possibly due to the following reasons:

- RF is generally more robust to overfitting, especially with a lot of features like in our case. Adaboost is more sensitive to noise which can lead to biased models.
- RF better captures complex and non-linear relationships especially when using default parameters for both classifiers.

Comparison for Question 4 (d) - Optimised Hyperparameters

The optimal number of trees for RF is found to be 221, which is quite different to the default of 100. This suggests that we fine-tuned the RF effectively, which explains the improved accuracy of 95%. The optimal number of trees for Adaboosr is 111, which is similar to the default value, hence we achieve no performance improvements.

Comparison for Question 4 (e) - Most Important Features

We reduce the complexity of the RF while maintaining the accuracy suggesting that we effectively selected the most important features. For Adaboost, we achieve a performance improvement because focusing on the most informative features mitigates its sensitivity to noise and increases robustness.

Table 23: Contingency tables comparing the test set results for the Random Forest and Adaboost classifiers.

(a) Default parameters.

		Adaboost		
		1	2	3
RF	1	26	1	2
	2	2	25	0
	3	2	0	17

(b) Optimised parameters.

		Adaboost		
		1	2	3
RF	1	27	0	3
	2	1	25	1
	3	1	0	17

(c) Most important features.

		Adaboost		
		1	2	3
RF	1	26	0	3
	2	0	26	0
	3	0	0	20

Table 24: Test set accuracy comparisons for the Random Forest and Adaboost classifiers.

Case	RF Accuracy	Adaboost Accuracy
Default parameters	93%	89%
Optimised parameters	95%	89%
Most important features	95%	91%

2.2 Question 5: Unsupervised Learning - Clustering

2.2.1 Question 5 (a)

Comparison of k-means vs. GMM

We apply a k-means and Gaussian Mixture Model (GMM) [36] to cluster the preprocessed dataset from Question 4. The differences of the cluster models are summarised in Table 25.

Table 25: Differences of the k-means and the Gaussian Mixture Model for clustering.

Aspect	k-means	Gaussian Mixture Models
Cluster Model	Non-probabilistic model that assigns each data point to the nearest cluster centroid.	Probabilistic model that assign each point a probability of belonging to clusters.
Cluster Assignment	Assigns each data point to one specific cluster (hard assignment).	Provides the probability of each data point belonging to each cluster (soft assignment).
Cluster Shapes	Assumes that clusters are spherical with equal variance.	Assumes that the data within clusters follow a Gaussian distribution. More flexible in handling clusters of different sizes and shapes.

Hyperparameter Optimisation

The main hyperparameters for both models is the number of clusters k. As in Question 1, we first apply PCA before finding the optimal number of clusters using different cluster quality metrics in Figure 21 and 22. However, the do not agre on the optimal k, hence we introduce a combined custom score ρ given by

$$\rho = S_{norm} + CH_{norm} - DB_{norm}$$

where S_{norm}, CH_{norm}, DB_{norm} are the Silhouette, Calinski-Harabasz and Davies-Bouldin scores respectively normalised to the same scale between 0 and 1. The maximum combined score ρ is achieved at $k = 3$ for both models.

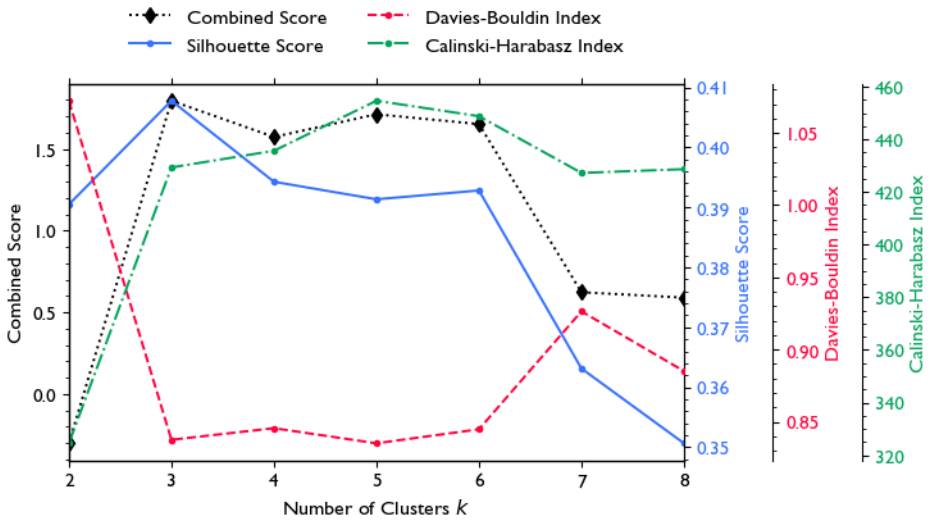


Figure 21: Silhouette, Calinski-Harabasz, and Davies-Bouldin scores for the k-means clustering model at different numbers of clusters k.

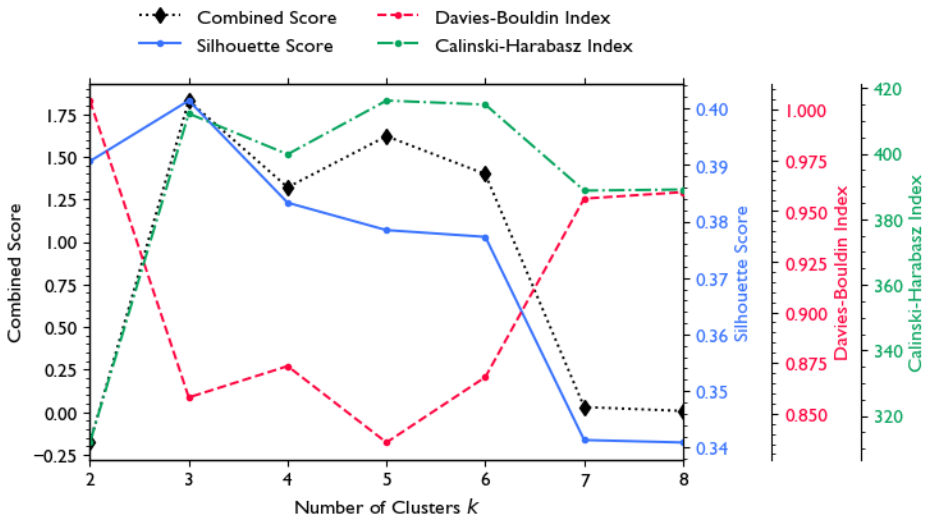


Figure 22: Silhouette, Calinski-Harabasz, and Davies-Bouldin scores for the GMM clustering model at different numbers of clusters k.

Clustering Results

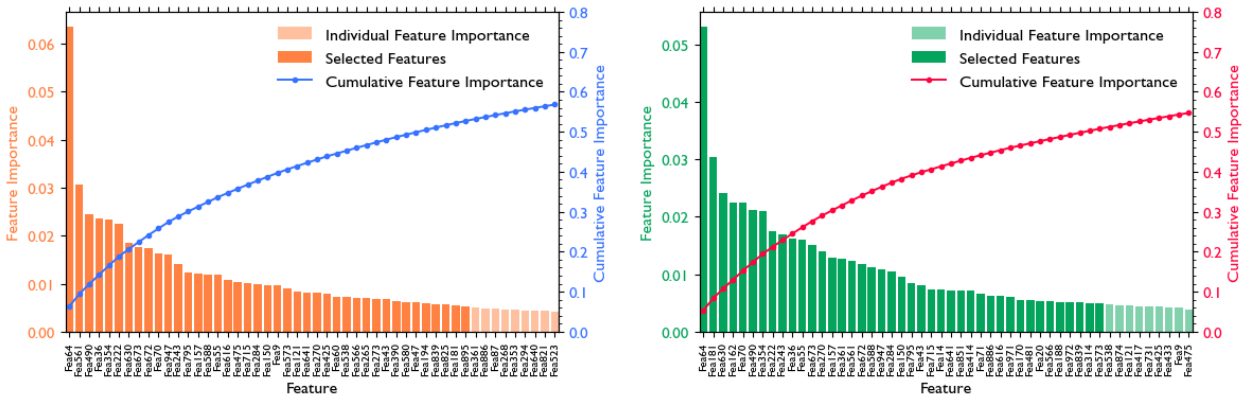
We apply both models with $k = 3$ on all features (scaled for better distance calculations) and compare the results in Table 26. The cluster assignments agree for most data points. However, there is a relatively large amount of 77 points assigned to cluster 0 by the GMM, while k-means assigns them to cluster 2. The GMM clustering also assigns 21 points to cluster 2 rather than cluster 1 like k-means does. In the high-dimensional space, GMM achieves a better cluster separation with a Silhouette score of 0.021 compared to 0.008 for k-means. This is due to the mitigation of the curse of dimensionality for distance-based clustering and noise reduction as explained in Question 1. The ARI of 0.52 suggests a moderate clustering similarity.

Table 26: Contingency table comparing the cluster outputs between the k-means and GMM models. The ARI is 0.52.

		GMM		
		0	1	2
k-means	0	117	1	3
	1	1	144	21
	2	77	0	136

2.2.2 Question 5 (b)

We train a RF classifier to identify the most discriminative features. The chosen feature subsets are highlighted in Figure 23. We apply the same clustering models using these feature subsets and compare the results to those using all features in Table 27 and Figures 24 and 25. For both models, using the most discriminative features gives results that resemble the 'true' clusters more closely due to the advantages of dimensionality reduction and feature selection as explained in Question 1. In the lower-dimensional feature space, k-means improves the cluster separation to a Silhouette score of 0.017 because distance calculations become more meaningful. For k-means, the most significant change is the assignment of 37 points to cluster 0 rather than cluster 2 as evident from the boundary area between these two clusters in Figure 24. The ARI of 0.56 for GMM suggests that the cluster assignments have changea notably, most evident from the assignment of more points to cluster 1 rather than cluster 2 in the top left corner in Figure 25.



(a) Most discriminative features for k-means.

(b) Most discriminative features for GMM.

Figure 23: Individual and cumulative features importances from the RF classifier trained on the original clustering results. For better comparison we choose the same number of features for both models based on whichever knee point gives the most features. The knee point criterion is the same as in Question 4.

Table 27: Contingency tables comparing cluster results using the most discriminative features against the ones obtained using all features.

(a) Clustering with k-means. The ARI is 0.74.

		discriminative		
		0	1	2
all	0	119	1	1
	1	3	163	0
	2	37	6	170

(b) Clustering with GMM. The ARI is 0.56.

		discriminative		
		0	1	2
all	0	144	4	47
	1	4	141	0
	2	4	29	127

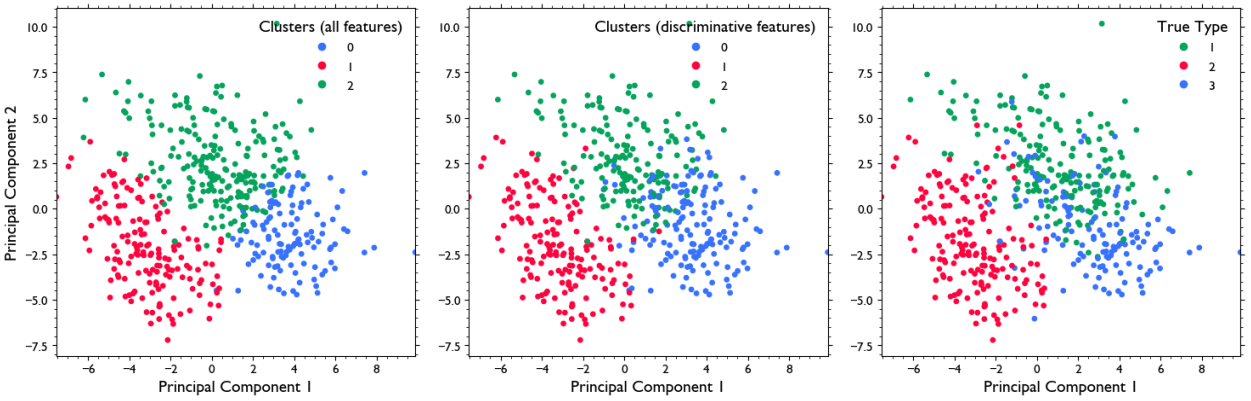


Figure 24: Clustering with k-means. Cluster results using all features (left) against the ones obtained using the most discriminative features (center, Silhouette score of 0.33). The true type classes are shown for comparison only (right). Note that the original PCA space is used here in all plots for better comparison.

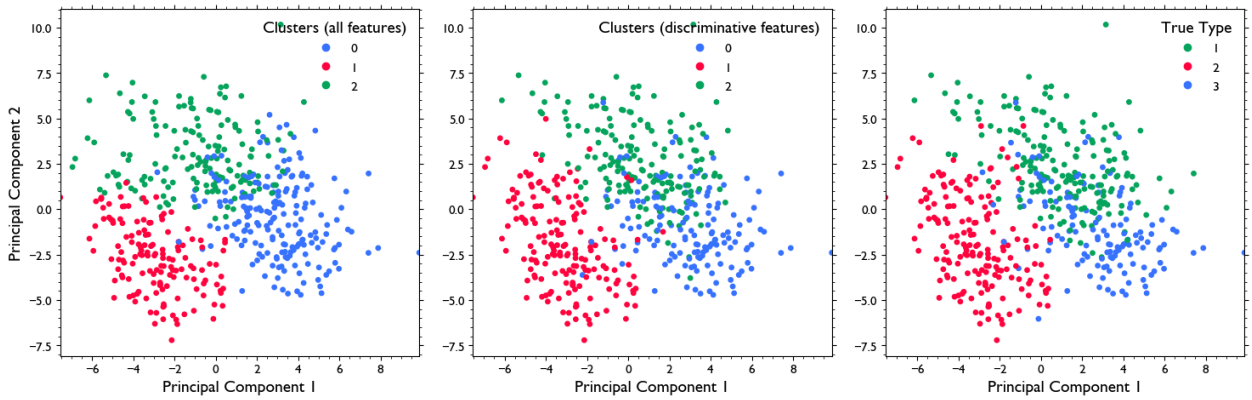
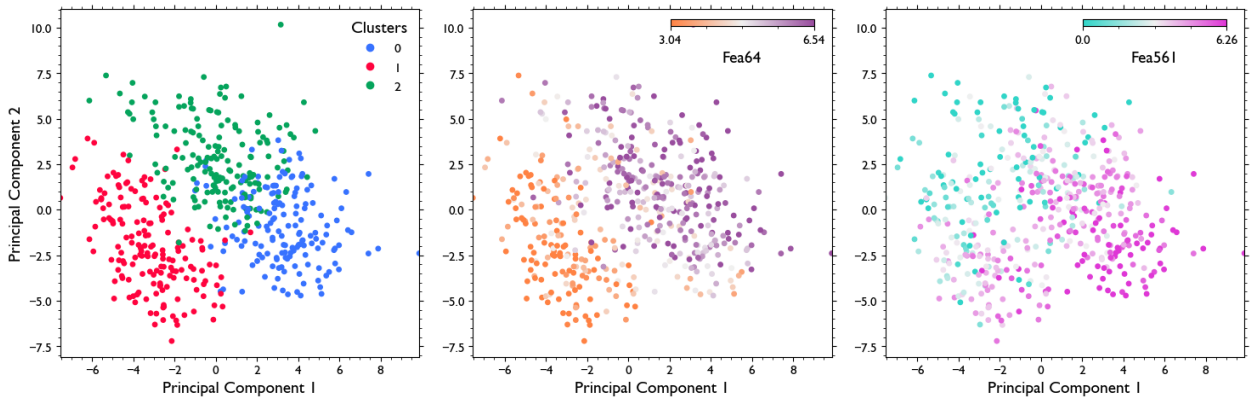


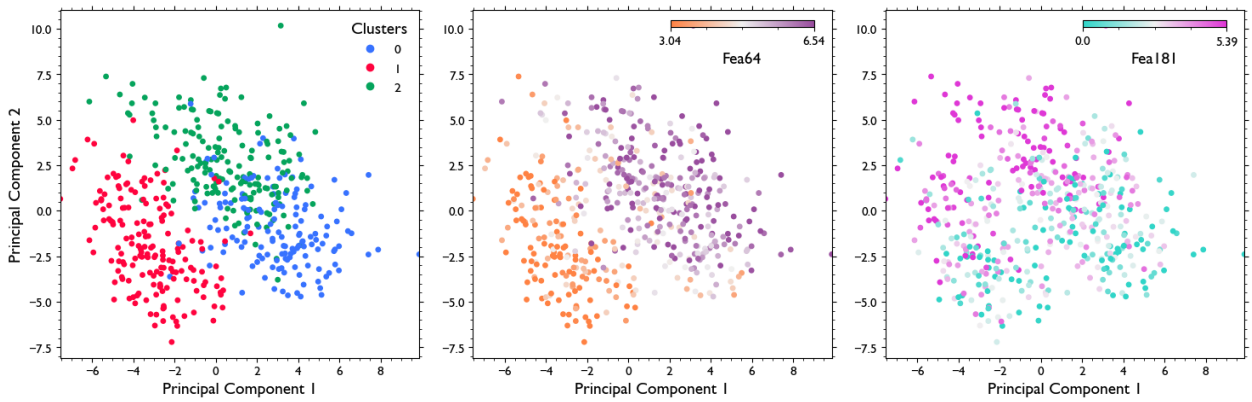
Figure 25: Clustering with GMM. Cluster results using all features (left) against the ones obtained using the most discriminative features (center). The true type classes are shown for comparison only (right). Note that the original PCA space is used here in all plots for better comparison.

2.2.3 Question 5 (c)

We visualise the clusters in the original 2D PCA space and colour-code them by cluster membership and the two most discriminative features in Figure 26. The most discriminative feature value increases from the bottom left to the top right according to the colour gradient. This feature is very informative as it shows a clear separation of two subpopulations corresponding to cluster 1 (orange) vs. cluster 0 + cluster 2 (purple). The second discriminative features show gradients from the top right to bottom left and vice versa. This explains the existence of a third subpopulation as it separates clusters 0 and 2 (turquoise vs. pink).



(a) Clustering with k-means.



(b) Clustering with GMM.

Figure 26: Cluster results using the most discriminative features (left) colour-coded by the value of the most (center) and second most (right) discriminative feature.

Bibliography

[1] Mohammed J Islam, QM Jonathan Wu, Majid Ahmadi, and Maher A Sid-Ahmed. Investigating the performance of naive-bayes classifiers and k-nearest neighbor classifiers. In 2007 international conference on convergence information technology (ICCIT 2007), pages 1541–1546. IEEE, 2007.

[2] John D Hunter. Matplotlib: A 2d graphics environment. Computing in science & engineering, 9(03):90–95, 2007.

[3] Michael L. Waskom. seaborn: statistical data visualization. Journal of Open Source Software, 6(60):3021, 2021.

[4] Emanuel Parzen. On estimation of a probability density function and mode. The annals of mathematical statistics, 33(3):1065–1076, 1962.

[5] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin philosophical magazine and journal of science, 2(11):559–572, 1901.

[6] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[7] 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, 12:2825–2830, 2011.

[8] Per-Erik Danielsson. Euclidean distance mapping. Computer Graphics and image processing, 14(3):227–248, 1980.

[9] Stuart Lloyd. Least squares quantization in pcm. IEEE transactions on information theory, 28(2):129–137, 1982.

[10] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. An introduction to statistical learning, volume 112. Springer, 2013.

[11] Marek Slipski, Armin Kleinböhl, Steven Dillmann, David M Kass, Jason Reimuller, Mark Wronkiewicz, and Gary Doran. The cloudspotting on mars citizen science project: Seasonal and spatial cloud distributions observed by the mars climate sounder. Icarus, page 115777, 2023.

[12] Lawrence Hubert and Phipps Arabie. Comparing partitions. Journal of classification, 2:193–218, 1985.

[13] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics, 20:53–65, 1987.

[14] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. Communications in Statistics-theory and Methods, 3(1):1–27, 1974.

[15] David L Davies and Donald W Bouldin. A cluster separation measure. IEEE transactions on pattern analysis and machine intelligence, (2):224–227, 1979.

[16] R Indhumathi and S Sathiyabama. Reducing and clustering high dimensional data through principal component analysis. International Journal of Computer Applications, 11(8):1–4, 2010.

[17] Chris Ding and Xiaofeng He. K-means clustering via principal component analysis. In Proceedings of the twenty-first international conference on Machine learning, page 29, 2004.

[18] N Tajunisha and V Saravanan. An increased performance of clustering high dimensional data using principal component analysis. In 2010 First International Conference on Integrated Intelligent Computing, pages 17–21. IEEE, 2010.

[19] Roderick JA Little and Donald B Rubin. Statistical analysis with missing data, volume 793. John Wiley & Sons, 2019.

[20] Donald B Rubin. Inference and missing data. Biometrika, 63(3):581–592, 1976.

[21] Gustavo EAPA Batista, Maria Carolina Monard, et al. A study of k-nearest neighbour as an imputation method. His, 87(251-260):48, 2002.

[22] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. IEEE transactions on information theory, 13(1):21–27, 1967.

[23] Michael W Browne. Cross-validation methods. Journal of mathematical psychology, 44(1):108–132, 2000.

[24] Donald B Rubin. Multiple imputation. In Flexible Imputation of Missing Data, Second Edition, pages 29–62. Chapman and Hall/CRC, 2018.

[25] Ian R White, Patrick Royston, and Angela M Wood. Multiple imputation using chained equations: issues and guidance for practice. Statistics in medicine, 30(4):377–399, 2011.

[26] Michael E Tipping. Sparse bayesian learning and the relevance vector machine. Journal of machine learning research, 1(Jun):211–244, 2001.

[27] Leo Breiman. Random forests. Machine learning, 45:5–32, 2001.

[28] KOLMOGOROV AN. Sulla determinazione empirica di una legge didistribuzione. Giorn Dell'inst Ital Degli Att, 4:89–91, 1933.

[29] Nickolay Smirnov. Table for estimating the goodness of fit of empirical distributions. The annals of mathematical statistics, 19(2):279–281, 1948.

[30] John W Tukey et al. Exploratory data analysis, volume 2. Reading, MA, 1977.

[31] J. Ross Quinlan. Induction of decision trees. Machine learning, 1:81–106, 1986.

[32] Leo Breiman. Bagging predictors. Machine learning, 24:123–140, 1996.

[33] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. The elements of statistical learning: data mining, inference, and prediction, volume 2. Springer, 2009.

[34] Corrado Gini. Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche.[Fasc. I.]. Tipogr. di P. Cuppini, 1912.

[35] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences, 55(1):119–139, 1997.

[36] Richard O Duda, Peter E Hart, et al. Pattern classification and scene analysis, volume 3. Wiley New York, 1973.

A README.md file

A copy of the `README.md` file is attached to this document and can be found on the next pages.

M1 Applied Data Science Coursework Submission (sd2022)

Description

This project is associated with the submission of the coursework for the M1 Applied Data Science Module as part of the MPhil in Data Intensive Science at the University of Cambridge. The Coursework Instructions can be found under Instructions.md and the problems to be answered can be found under DIS_MPhil_M1_Coursework.pdf. The associated project report can be found under M1 Coursework Report.

Table of Contents

Installation and Usage

To get started with the code associated with the coursework submission, follow these steps:

Requirements

- Python 3.9 or higher installed on your system.
- Conda installed (for managing the Python environment).
- Docker (if using containerisation for deployment).

Steps

You can either run the code locally using a `conda` environment or with a container using Docker. The Jupyter Notebooks associated with the different Questions are located in the `sd2022/src` directory:

- Question 1 : Answer 1 Notebook
- Question 2 : Answer 2 Notebook
- Question 3 : Answer 3 Notebook
- Question 4 : Answer 4 Notebook
- Question 5 : Answer 5 Notebook

The Jupyter Notebooks will run faster locally on a high-spec computer (recommended).

Local Setup (Using Conda) [RECOMMENDED]

1. **Clone the Repository:**

   Clone the repository to your local machine with the following command:

   ```
   $ git clone https://gitlab.developers.cam.ac.uk/phy/data-intensive-scie
   ```

   or simply download it from M1 Applied Data Science Coursework (sd2022).

2. **Navigate to the Project Directory:**

   On your local machine, navigate to the project directory with the following command:

   ```
   $ cd /full/path/to/sd2022
   ```

   and replace `/full/path/to/` with the directory on your local machine where the repository lives in.

3. **Setting up the Environment:**

   Set up and activate the `conda` environment with the following command:

   ```
   $ conda env create -f environment.yml
   $ conda activate sd2022_m1_env
   ```

4. **Install ipykernel:**

   To run the notebook cells with `sd2022_m1_env`, install the ipykernel package with the following command:

```
python -m ipykernel install --user --name sd2022_m1_env --display-name
```

5. **Open and Run the Notebook:**

   Open the `sd2022` directory with an integrated development environment (IDE), e.g. VSCode or PyCharm, select the kernel associated with the `sd2022_m1_env` environment and run the Jupyter Notebooks (located in the `sd2022/src` directory).

Containerised Setup (Using Docker)

1. **Clone the Repository:**

   Clone the repository to your local machine with the following command:

   ```
   $ git clone https://gitlab.developers.cam.ac.uk/phy/data-intensive-scie
   ```

   or simply download it from M1 Applied Data Science Coursework (sd2022).

2. **Navigate to the Project Directory:**

   On your local machine, navigate to the project directory with the following command:

   ```
   $ cd /full/path/to/sd2022
   ```

   and replace `/full/path/to/` with the directory on your local machine where the repository lives in.

3. **Install and Run Docker:**

   You can install Docker from the official webpage under Docker Download. Once installed, make sure to run the Docker application.

4. **Build the Docker Image:**

   You can build a Docker image with the following command:

   ```
   $ docker build -t [image] .
   ```

   and replace `[image]` with the name of the image you want to build.

5. **Run a Container from the Image:**

   Once the image is built, you can run a container based on this image:

```
$ docker run -p 8888:8888 [image]
```

This command starts a container from the `[image]` image and maps port `8888` of the container to port `8888` on your local machine. The Jupyter Notebook server within the container will be accessible on JupyterLab at http://localhost:8888.

6. **Access and Run the Notebook:**

   After running the container, you'll see logs in the terminal containing a URL with a token. It will look similar to this:

   ```
   http://127.0.0.1:8888/lab?token=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   ```

   Navigate to http://localhost:8888 and enter the token `XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX`. Once you accessed JupyterLab, run the Jupyter Notebooks(located in the `sd2022/src` directory) with an `ipykernel` (Python 3).

   Note: Make sure that not other Jupter Notebook Servers are running. Otherwise, you might encounter 'Invalid credentials' issues when entering the token. Close any running Jupter Notebook Servers. To stop a running server, use `Ctrl + C` in the terminal where you launched JupyterLab. Also make sure port `8888` is not occupied.

Support

For any questions, feedback, or assistance, please feel free to reach out via email at sd2022@cam.ac.uk.

License

This project is licensed under the MIT License - see the LICENSE file for details.

Project Status

The project is in a state ready for submission. All essential features have been implemented, and the codebase is stable. Future updates may focus on minor improvements, bug fixes, or optimisations.

Note on the Use of auto-generation tools

GitHub Co-Pilot assisted the author in producing all function docstrings present in the project repository. No specific commands have been given, instead auto-completion suggestions have occasionally been accepted.

Authors and Acknowledgment

This project is maintained by Steven Dillmann at the University of Cambridge.

24th December 2023

Merry Christmas 🎄