M1: Applied Data Science - Coursework

William Knottenbelt, wdk24

December 24, 2023

Section A

Q1

(a)

We plot density histograms of the first 20 features in Fig. 1, and observe that 14 of the features exhibit a single sharp peak at zero, suggesting that we are dealing with sparse data, where many features consist of mostly zero values. Many features, such as Fea2 and Fea13, exhibit a strong positive skew and Fea5 has a negative skew, which may indicate the presence of outliers. We also see several of the features have two peaks (Fea11, Fea14, Fea18) which could indicate the presence of distinct underlying groups in the data.

(b)

In Fig. 2, we used PCA to visualize the dataset along the two linear orthogonal axes which capture the most variance in the data [1], and observed two distinct clusters. We saw in the density plots that certain features exhibited exactly two peaks, which could indicate the presence of two underlying groups in the data that are now being observed as separate clusters in this PCA plot. We also observe a significant spread across the principal components, which is consistent with the variance observed in the feature density plots.

(c)

We applied default k-means clustering (8 clusters) to two disjoint subsets of the data and mapped the unused data onto the clusters by finding which learned centroid each unused data-point was closest to [2]. The contingency table in Fig. 3 shows high numbers in two cells, indicating two overlapping pairs of clusters between models, which suggests some natural clustering structure in the data. However, the many low-numbered cells scattered across rows 1 and 5, and column 7 reflects significant disagreement between the models, which may be a result of variability between the randomly partitioned subsets.

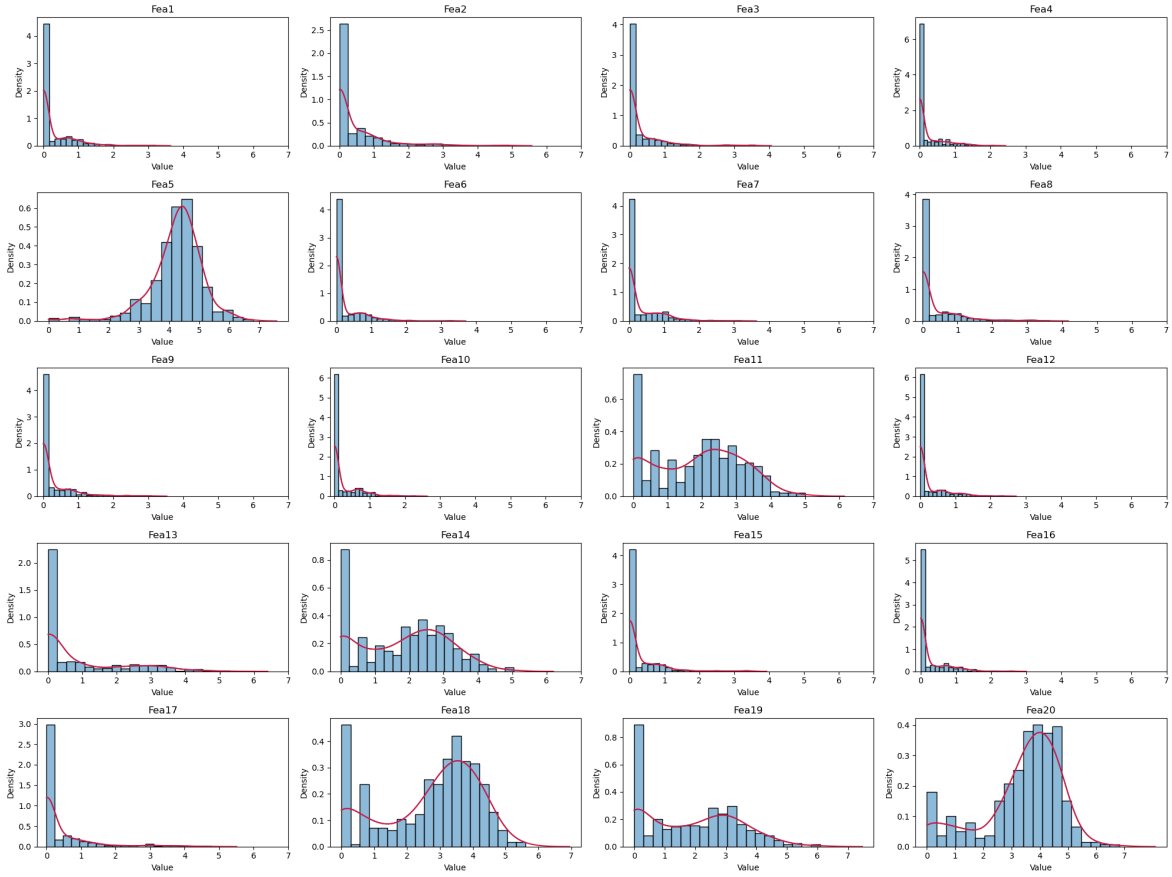

Figure 1: Density histograms for the first 20 features in dataset `A_NoiseAdded.csv`, with kernel density lines overlaid.

(d)

As discussed, the clusterings in part (c) were significantly different, showing that the clusters are highly unstable as they are sensitive to centroid initialisations and variability in the input. The cluster sizes are also very different between the models, reinforcing instability of clusterings. In model 1, there is a single dominant cluster of size 203, some intermediate-sized clusters, and three small clusters of size 1, 2 and 3. The clusters sizes of model 2 ranged from 13 to 122. This instability, and the presence of extremely small clusters, suggests that there are not 8 inherent clusters in the data. The PCA plot in Fig. 2 indicated the presence of two clusters, hence we performed k-means again with 2 clusters.

(e)

We clustered the dataset with K-Means, both before and after applying PCA, resulting in identical clusters. We identified the k-means clusters in Fig. 4, and observed that the two clusters discussed in part (b) were discovered by k-means. This demonstrates that PCA successfully preserved the dataset's natural clustered structure.

Performing K-Means before PCA preserves the data's full structure for potentially more reliable

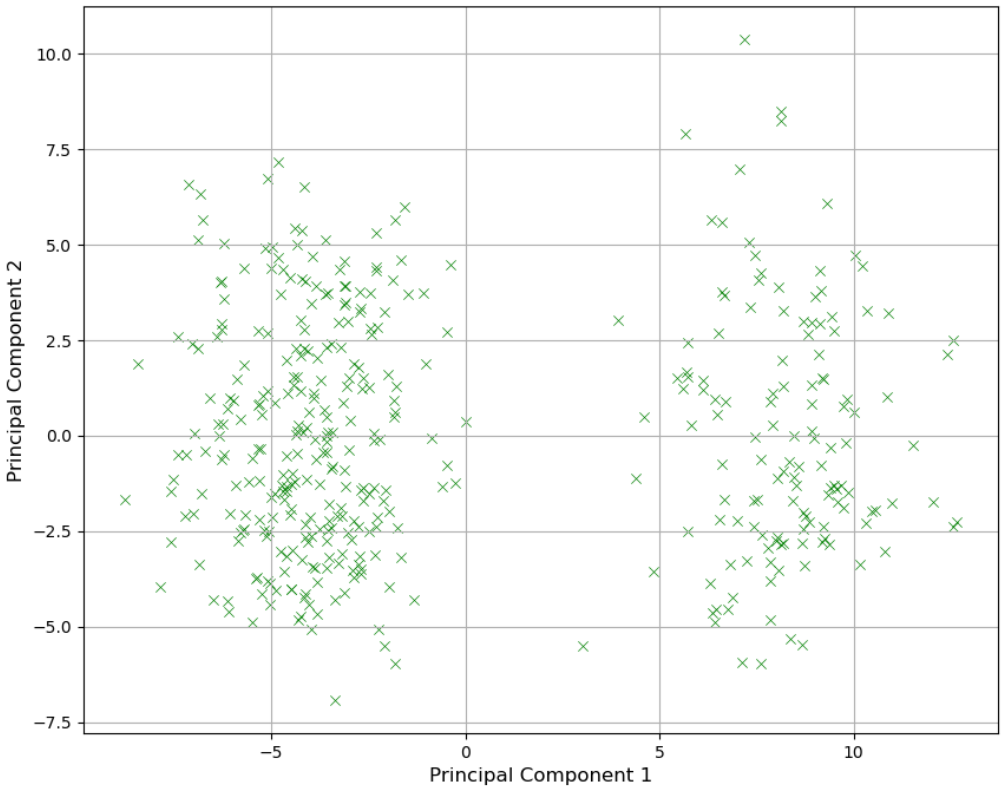

Figure 2: 2D visualisation of dataset `A_NoiseAdded.csv` obtained using principal component analysis.

clustering, but noise can impact results and PCA may not preserve relationships between clusters, making the results difficult to interpret by visualization. K-Means clustering after PCA is more computationally efficient and may reduce noise, but PCA might distort the dataset's original structure, potentially leading to biased clusters. In general, the optimal technique depends on the data: PCA before K-Means suits high-dimensional noisy data, while K-Means before PCA is better for sensitive data.

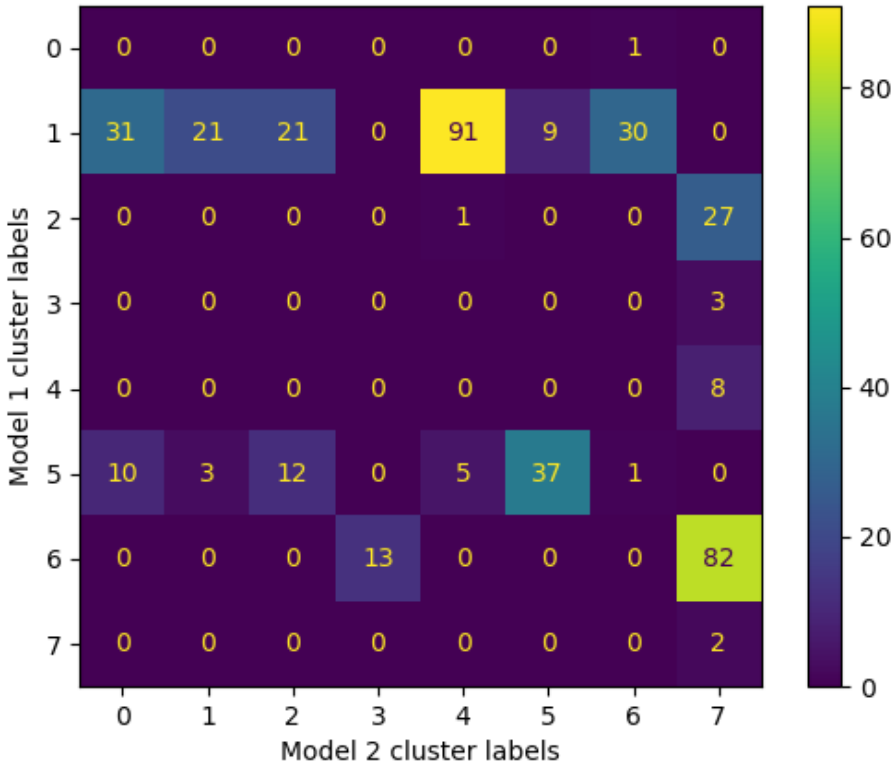

Figure 3: Contingency table of cluster predictions of dataset `A_NoiseAdded.csv`, from two k-means models trained on two disjoint equally-sized random subsets of the dataset.

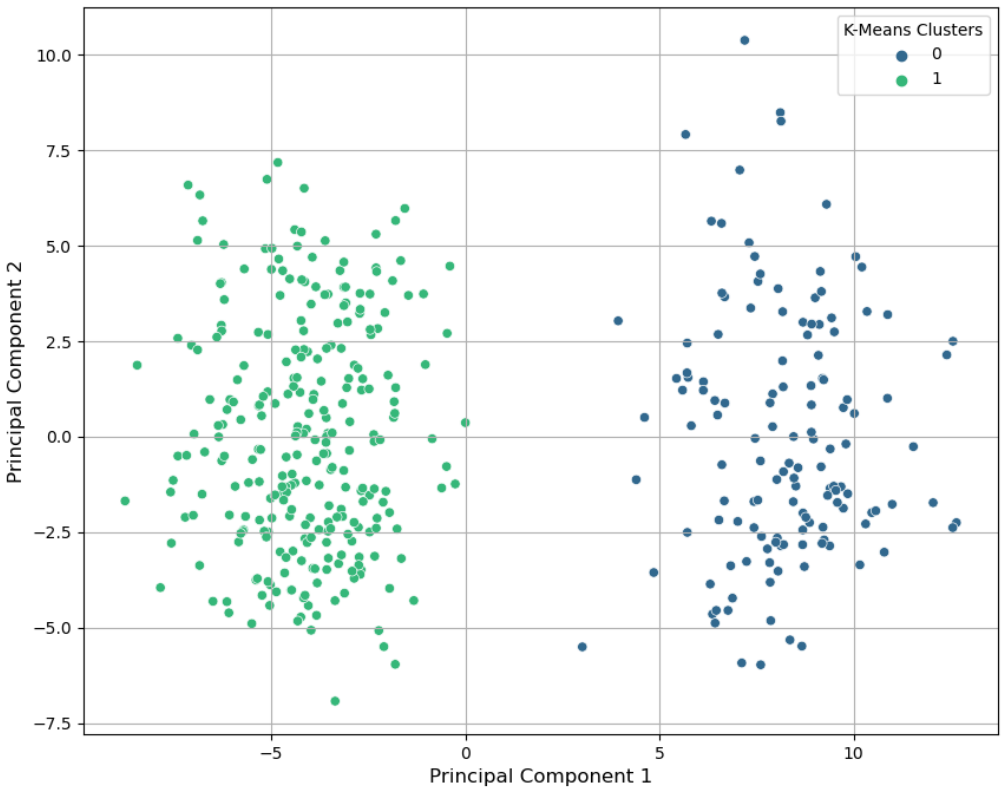

Figure 4: 2D PCA visualisation of dataset `A_NoiseAdded.csv`, with K-Means clusters identified.

Q2

(a)

As can be seen in Table 1, there are three classifications and 20 missing labels. The frequency of labels is not uniform, with classification '1.0' and '2.0' being more than twice as prevalent as classification '4.0'.

Classification	Count
1.0	179
2.0	157
4.0	72
NaN	20

Table 1: Frequency of classification labels in dataset `B_Relabelled.csv`.

(b)

We identified 20 unique duplicate pairs, half with inconsistent labels, and chose to retain one instance per pair, preserving consistent labels or marking inconsistent ones as 'NaN' for later imputation. This strategy assumes the duplicates do not carry useful information. The updated label frequencies are presented in table 2.

Classification	Count
1.0	164
2.0	149
4.0	65
NaN	30

Table 2: Frequency of classification labels in dataset `B_Relabelled.csv` after processing duplicates.

(c)

One approach to handle missing labels is to impute them using a k-Nearest Neighbors classifier [3]. This approach prevents the loss of valuable data but can introduce bias to the dataset and may be computationally expensive. Another approach is to delete the observations with missing labels. This is computationally efficient and does not risk introducing bias from incorrect imputation, but it may cause the loss of potentially important information.

Missing data is 'missing at random' if the probability of missing-ness is independent of the missing values, but depends on some of the observed data. Conversely, it is 'missing not at random' if the probability of missing-ness is dependent on the values that are missing.

(d)

We chose to predict the labels using k-nearest neighbours, since it can capture non-linear relationships using sample similarity, and the classes may not be linearly-separable. We selected hyper-parameters

using cross validation, and displayed the test set performance of the optimised classifier in Fig. 5. We see that all 14 incorrect predictions are false positives for class '1.0', indicating that the model is biased towards this class since it is the most prevalent class.

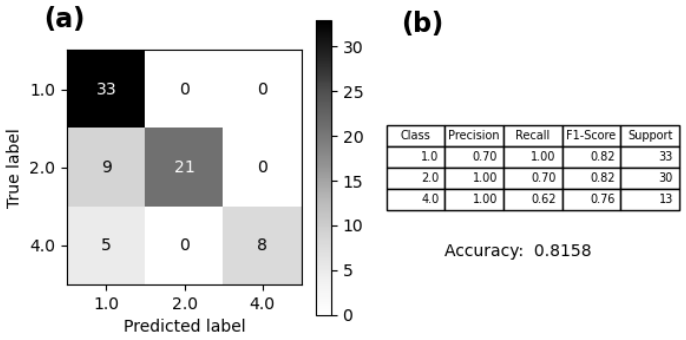

Figure 5: Test set results for k-NN class predictions of dataset `B_Relabelled.csv`. (a) Confusion Matrix. (b) Performance metrics.

We use this model to predict missing labels and display the updated frequencies in table 3. Comparing these frequencies to those in table 1 and 3, we see that only class '1.0' has increased substantially. This could be a result of the potential bias of the k-NN model towards this class, suggesting that the predicted labels may not be trustworthy.

Classification	**Count**
1.0	186
2.0	155
4.0	67

Table 3: Frequency of classification labels in dataset `B_Relabelled.csv` after processing duplicates and predicting all missing labels.

Q3

(a)

There are 11 features with 5 missing values each: Fea58, Fea142, Fea150, Fea233, Fea269, Fea299, Fea339, Fea355, Fea458, Fea466, Fea491.

(b)

One static imputation method is to replace missing values with the median of the feature, which is computationally efficient and does not depend on predictive model performance, but it does not account for relationships with other variables, reasons behind missing-ness, or variability in the data, and thus can lead to biased imputations.

Alternatively, the k-Nearest Neighbours imputation method uses the mean of 'k' closest observations [3], which leverages feature relationships for more accurate estimates when the missing values depend on

other features. However, it is computationally demanding, sensitive to outliers, and requires selection of hyper-parameters.

Multiple imputation fills missing values several times and combines the results. This allows us to quantify uncertainty of the imputations and enhances reliability by pooling outcomes from multiple methods, thus reducing the impact of individual poor predictions.

(c)

We chose random forest regression to impute the missing values, since we are dealing with a high-dimensional, sparse dataset, which is handled well by random forest since it can select only the relevant features for decision nodes and ignore irrelevant features [4]. When training the model for imputation, the other missing values are temporarily filled with the median of the feature, which was chosen over the mean for its robustness to outliers.

Fig. 6 shows that the imputed distributions are very close to the original ones. Additionally we performed a Kolmogorov-Smirnov test [5] on all imputed features, which tests the hypothesis that the feature before and after imputation is distributed by the same underlying distribution. The p-values of all tests were close to 1 (0.99 or higher), indicating that the original and imputed distributions were near identical. This shows that the imputation process successfully preserved the statistical properties of the dataset.

(d)

The standardisation approach works by calculating the Z-score [6] for all values of each feature using the formula $Z_i = \frac{x_i - \mu}{\sigma}$, where μ is the mean of the feature and σ_j is standard deviation. We can define outlier values as those with a Z-score above a certain threshold, with $Z \geq 3$ being common convention.

However, for sparse features, the non-zero values typically carry most of the meaningful information but their Z-scores are inflated by the bulk of zero values, and hence correcting those with $Z \geq 3$ could cause the loss of important information. To illustrate this point, in Fig. 7 we plotted the z-scores of all non-zero values against the sparsity of the feature the value belongs to, as a density histogram. We see that for features with sparsity greater than 0.8 (80% zero values), a large proportion of the non-zero values have Z scores larger than 3.

To address this, we decided to calculate two separate z-scores for all features with more than 20% zero values. The first Z-score, Z_a, is calculated normally and has a threshold of 7. The second Z-score, Z_b, is calculated after excluding all zero values from the feature, and has a threshold of 3. If either z-score is above its threshold, the value is defined as an outlier. This way, a value in a sparse feature is an outlier only if it is an outlier with respect to the distribution non-zero values or it is an extreme outlier with respect to the full distribution, which is important in the case of extremely sparse features, where all non-zero values would be considered outliers. For non-sparse features, we calculated the z-scores normally and used the conventional threshold of 3.

Using this method, we identified all outlier values in the dataset. There were 108 features containing only outlier values and zeros, which were immediately removed. Out of the remaining features, there were 626 outlier values, across 196 features and 313 samples.

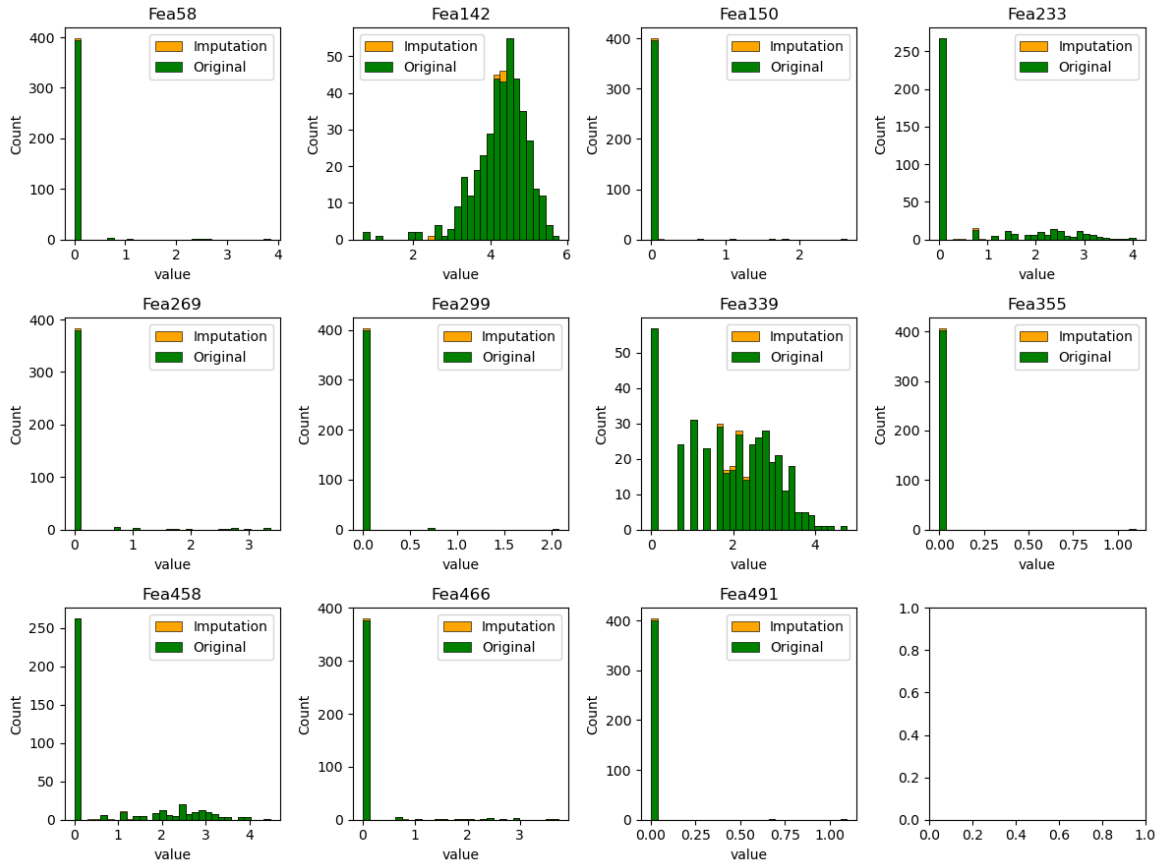

Figure 6: Histograms of the original and imputed distributions of all features with missing values in `C_MissingFeatures`.

(e)

We decided to impute these outlier values using k-nearest neighbours (k-NN), since it is more computationally efficient than random forest and there are a large number of features to impute.

We performed KS tests [5] on all imputed features, and all p-values were very high (smallest was 0.65), implying that the imputation effectively maintained the data's statistical characteristics. We plotted the original and imputed distributions of four of the features in Fig. 8. We see that the outliers in Fea336, Fea71 and Fea395, appear appropriately identified and corrected, however Fea493's imputations are in a unique position (close to 0) where no other instances appear. This demonstrates the limitations of k-NN imputation, which takes the mean of the closest samples, occasionally resulting in unique predictions when the feature consists of discrete values.

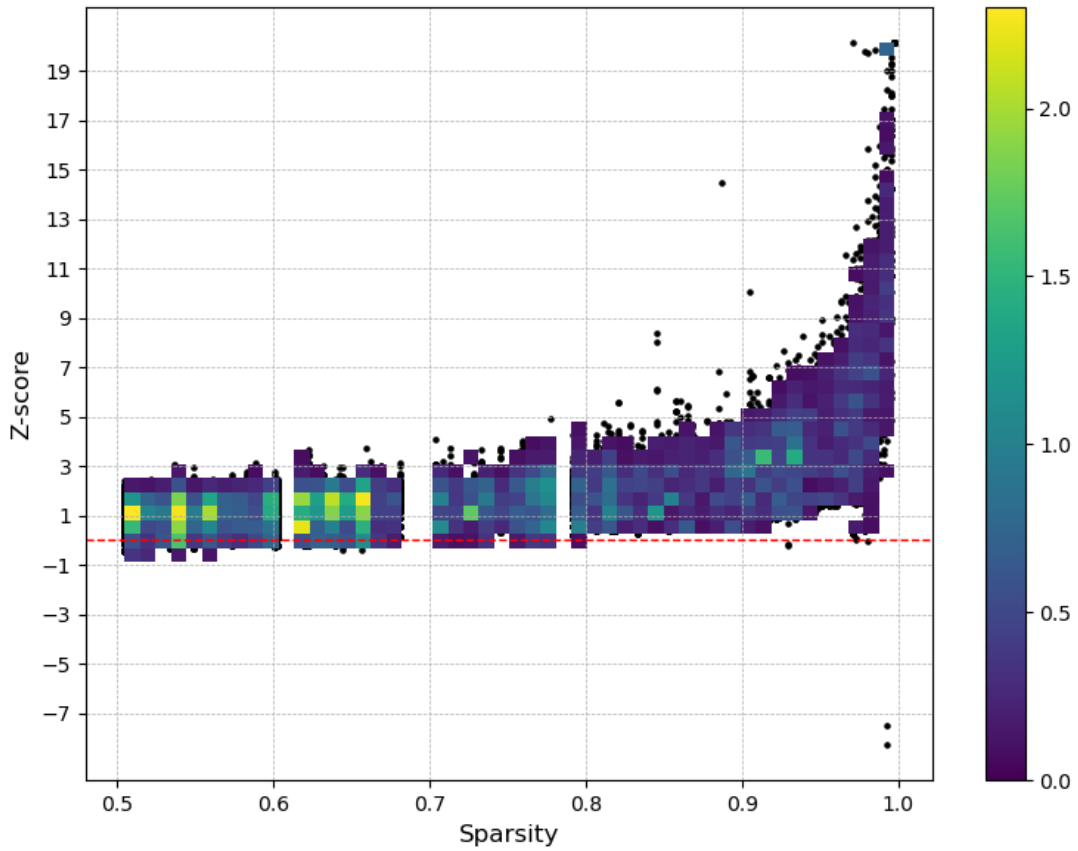

Figure 7: Z-score of all non-zero values in dataset `C_MissingFeatures` against the sparsity (proportion of zero values) of the feature the value corresponds to. Represented as a density histogram, with bins above a certain density threshold and colour indicated the density. For regions below the density threshold, the points are scattered, showing potential outliers. Red line indicates $Z = 0$.

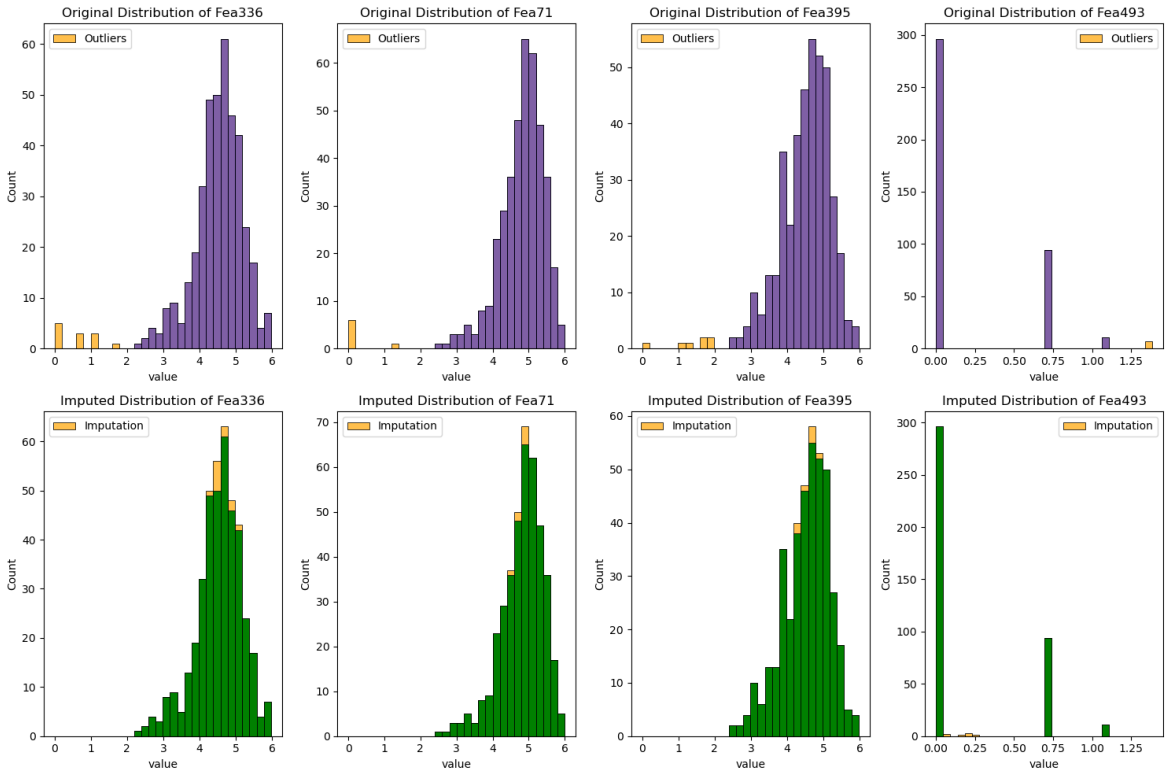

Figure 8: Distributions of four features in dataset `C_MissingFeatures`, before and after correction of outliers. The outliers and imputed values are indicated in orange.

Section B

Q4

(a)

A decision tree uses nodes to split data into homogeneous subsets using feature thresholds, leading to leaf nodes that represent the tree's predictions [7]. Bagging trains multiple decision trees on bootstrapped data samples and aggregates their predictions by majority voting, which reduces variance and prevents overfitting [8]. Random forest, an extension of bagging, uses a random subset of features at each decision node to create diverse trees, reducing overfitting and enhancing robustness [4]. A criterion to measure quality of a split is the Gini Impurity, which measures the likelihood of incorrect classification of a randomly chosen element from the set if it were randomly labelled according to the distribution of labels in the set [9]. Two hyper-parameters for random forest are the maximum depth of a tree and number of features considered at each decision node, for which a common heuristic is to use the square root of the total number of features. This heuristic reduces tree correlation while maintaining predictive power of trees.

(b)

To pre-process the data, we ensured no missing values were present, then calculated sparsity of features. It was found that many features were highly sparse, and 4.2% of features contained only zero values (100% sparsity). We removed features with sparsity $\geq 95\%$, which constituted 51% of the features. These features increase dimensionality and introduce noise, without providing useful information, which increases computational load for the classifier and hinders performance. Out of the remaining 490 features, there were no near-zero variance or highly correlated features.

Outlier values can negatively affect random forest classification by distorting decision boundaries, but correcting outliers risks losing important information. To balance these factors, we decided to identify only the most extreme outliers using the standardisation procedure described in Q3(d), with thresholds of $Z \geq 5$ for non-sparse features and $Z_a \geq 10, Z_b \geq 5$ for sparse features. We found 34 outlier values and imputed them using random forest, since it handles high-dimensional sparse data effectively.

The frequency of class labels in our dataset was not uniform, which could cause the classifier to be biased towards the more prevalent classes. To address this, we balanced the frequencies by over-sampling the minority classes, since under-sampling would cause the loss of valuable data and we had only 500 samples.

We did not scale the features as random forest is not sensitive to scale since decision nodes use thresholds to split the data.

(c)

We trained a random forest classifier with default parameters and presented its test set performance in Fig. 9. The model showed high accuracy and good performance across precision, recall, and F1-score for each class, but had slightly lower precision for class 1 due to four false positives, indicating a bias towards this class.

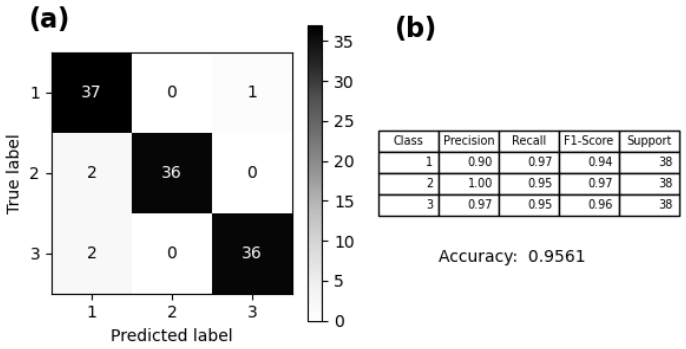

Figure 9: Test set results of default random forest classifier. (a) Confusion Matrix. (b) Performance metrics.

(d)

To optimise the number of trees, we split the training set from part (c) into a smaller training and validation set, and found that using 1000 trees optimised validation set accuracy. The test set performance of the optimised classifier is displayed in Fig. 10. We see that the performance is slightly improved across all metrics, but 3 false positives for class 1 are still present, indicating the bias remained.

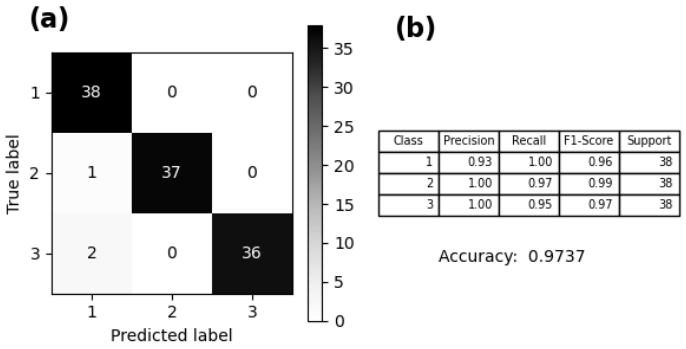

Figure 10: Test set results of random forest classifier, optimised with respect to the number of trees. (a) Confusion Matrix. (b) Performance metrics.

(e)

We calculate the feature importance as the mean of the accumulation of the impurity decrease from splitting on each feature (then normalised to sum to 1) [10]. We plot a histogram of the feature importances in Fig. 11, and observe that there is a small number of highly important features, particularly the 4 most important features: Fea64, Fea70, Fea630 and Fea162. We re-trained the classifier on these 4 features, and presented the test set results in Fig. 12. We observe very strong performance, with only 4 more incorrect predictions than the optimised classifier in Fig. 10. This demonstrates that these four features possess most of the predictive power and implies the other features are either redundant, or provide diminishing returns while increasing model complexity.

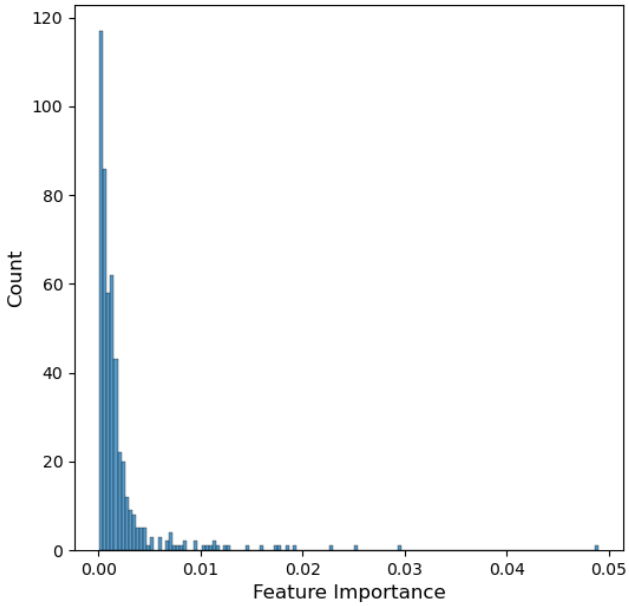

Figure 11: Histogram of importances of features in dataset `ADS_BaselineDataset.csv` for random forest classification.

(f)

For the second classifier, we chose multinomial logistic regression with Lasso regularisation, since it performs automatic feature selection by encouraging the coefficients of unimportant features to be exactly zero [11]. This is useful to prevent over-fitting in high-dimensional sparse datasets where many features are irrelevant. We used the pre-processed dataset from part (b) since all steps taken are similarly valid for logistic regression, except we normalised the features since logistic regression is sensitive to the scale of features. We chose to normalise instead of standardise since standardisation would destroy the sparseness structure of the dataset by centering the features. The model was trained and tested on the same sets as part (c), yielding 95.6% accuracy (see Fig. 13). This performance is almost as good as the random forest classifier, indicating that the classes are linearly-separable, since logistic regression is a linear model.

We calculated feature importance as the sum of absolute values of the learned coefficients for each class, since these dictate the change in log odds of classification per unit change in the feature [12], meaning higher absolute values make the model more sensitive to that feature. We find that only 111 out of 490 features have non-zero coefficients, which implies that the majority of features are entirely irrelevant for class prediction. We plot a histogram of the 111 non-zero feature importances in Fig. 14. Similarly to part (e), there were 4 particularly important features: Fea64, Fea70, Fea795, Fea947. We re-trained the model on these four features and obtained 86.8% accuracy (see Fig. 15). It is possible that the classes are less linearly-separable over just the most important features, leading to worse performance for logistic regression than random forest.

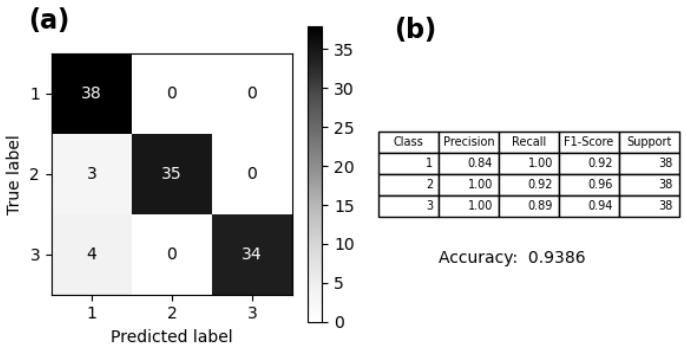

Figure 12: Test set results of random forest classifier trained on 4 most important features. (a) Confusion Matrix. (b) Performance metrics.

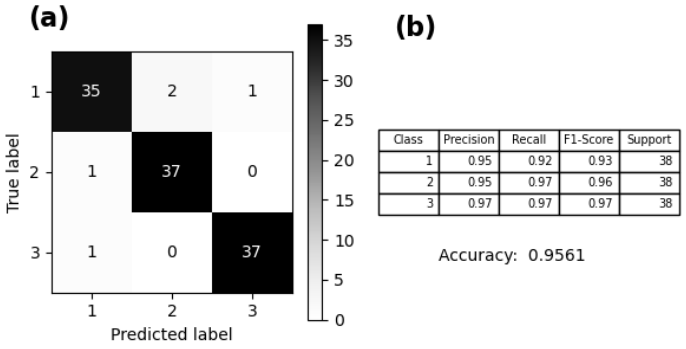

Figure 13: Test set results of logistic regression classifier. (a) Confusion Matrix. (b) Performance metrics

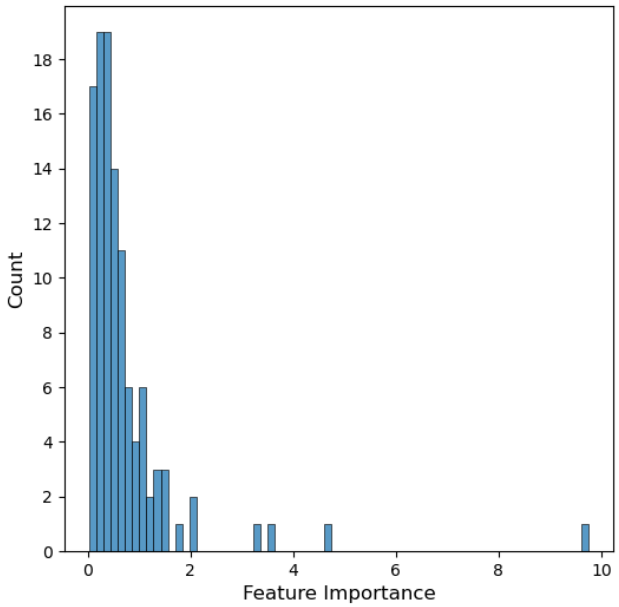

Figure 14: Histogram of non-zero importances of features in dataset `ADS_BaselineDataset.csv` for logistic regression classification.

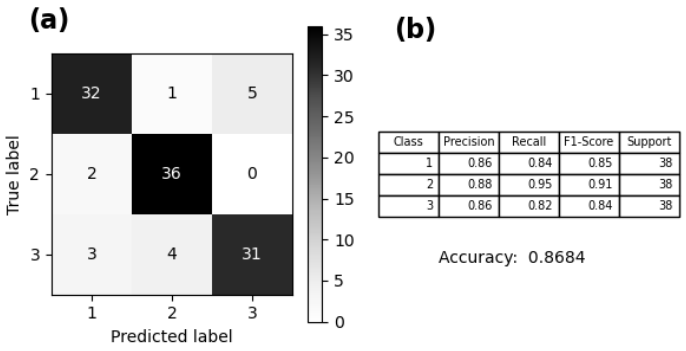

Figure 15: Test set results of logistic regression classifier trained on 4 most important features. (a) Confusion Matrix. (b) Performance metrics.

Q5

(a)

We chose to cluster the dataset using k-means and Spectral clustering. K-means clusters the dataset based on distance to centroids, while spectral clustering uses a similarity matrix to cluster the dataset in fewer dimensions [13]. k-means assumes clusters to be convex, evenly-sized and linearly-separable, while spectral clustering is able to identify complex, non-linear structures. To pre-process we removed the features with sparsity greater than 95%, corrected the most extreme outlier values as in Q4 (b) and normalised the features.

To determine the optimal number of clusters, we used the Elbow method [14] and Silhouette method [15] with k-means clusters, as plotted in Fig. 16. In the Elbow method, the mean within-cluster sum of squares (inertia) is plotted against number of clusters, then the optimal choice is that which resembles an 'elbow'. For the Silhouette method, we plot the silhouette score, which measure how similar a point is to its own cluster, against number of clusters and the optimal choice is that with the highest score. In Fig. 16(a), there is no sharp elbow except for a vague elbow at 4 clusters, however there is a clear maximum in Fig. 16(b) at 2 clusters, thus we choose 2 clusters.

The contingency table in Fig. 17 shows significant disagreement between the k-means and Spectral clusterings. We applied PCA and UMAP to visualise the data in 2D in Fig. 18, with the points coloured according to the cluster labels, and observed that the k-means clusters are best separated in the PCA plot, while the Spectral clusters are slightly better separated by UMAP. PCA is a linear technique, while UMAP aims to preserve local and global structures that can be non-linear [16], hence these results suggest that the k-means clusters are linearly-separable, while the Spectral clusters are non-linear.

(b)

We trained two classifiers to predict cluster labels obtained in part (a): logistic regression with Lasso regularisation, and random forest. For the k-means clusters, logistic regression was more accurate (Fig. 19(a)), but for the spectral clusters, random forest had better performance (Fig. 19(b)). This reinforces the idea that the k-means clusters are linearly-separable and the spectral clusters are not.

In each case, we calculated feature importance as described in Q4(e), and re-performed the clusterings using the 50 most important features. Contingency tables in Fig. 20 compare the clusterings with and without the less important features. We see the k-means models agree reasonably well, while the the Spectral models are substantially less aligned. This could be a result of random forest underestimating the importance of sparse features which are crucial to the clustered structure. This happens because dense features with higher variance tend to have better splitting power at decision nodes.

(c)

In Fig. 21 we plot a 2D visualisation of the dataset in 2D with points coloured according to (i) k-means clusters, (ii) the most discriminant feature, (iii) the second most discriminant feature. We see that the k-means clusters are separated in the top right and bottom left, and the discriminant features have drastically different values in these regions, which indicates that they are key factors characterising the cluster structure.

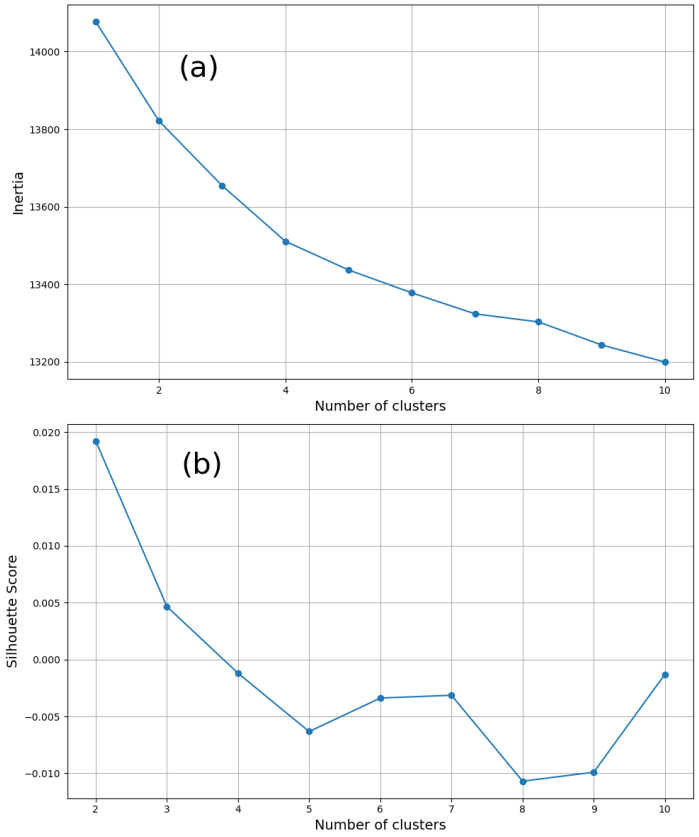

Figure 16: Plots used to select optimal number of clusters. (a) Elbow method. (b) Silhouette method.

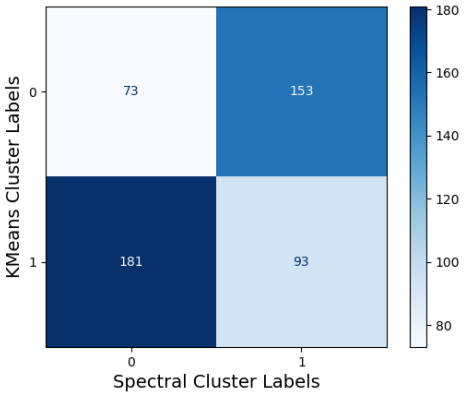

Figure 17: Contingency table comparing k-means and spectral clusterings.

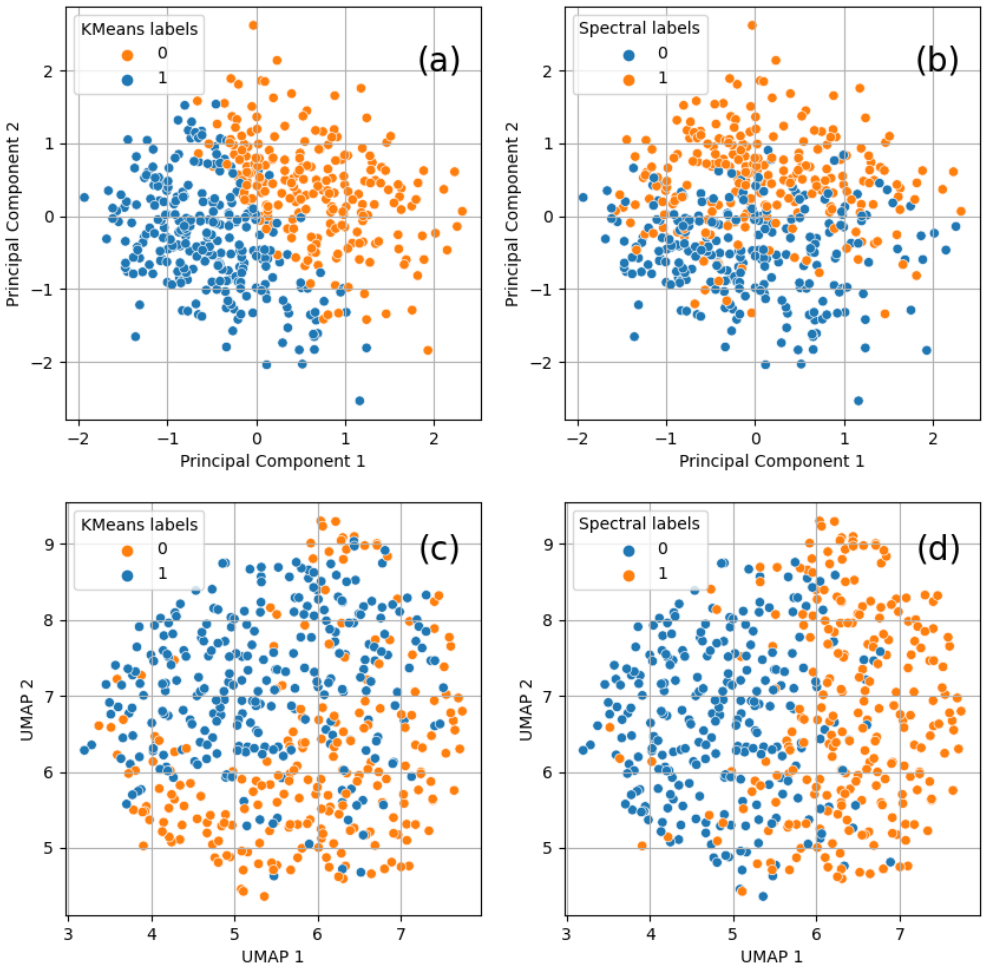

Figure 18: (a) k-means clusters indicated in PCA visualisation. (b) spectral clusters indicated in PCA visualisation. (c) k-means clusters indicated in UMAP visualisation. (d) spectral clusters indicated in UMAP visualisation.

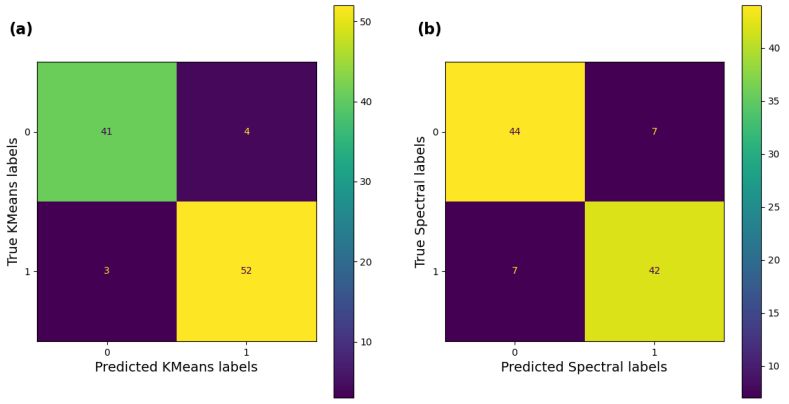

Figure 19: Confusion matrices showing cluster predictions. (a) Logistic regression predicting k-means labels. (b) Random forest predicting spectral labels.

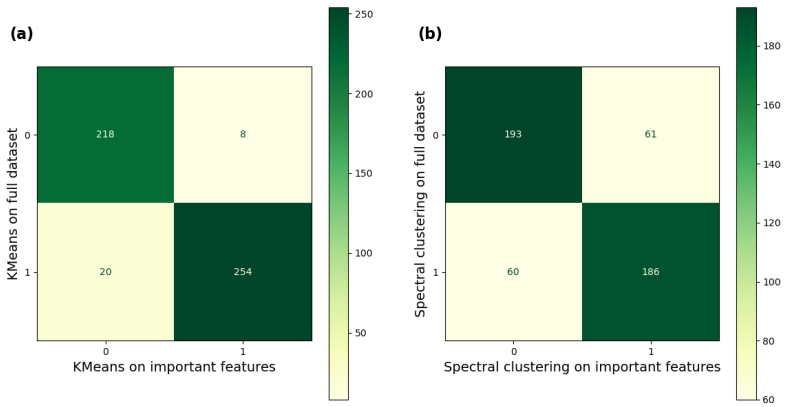

Figure 20: Contingency tables of clusterings trained on full dataset vs just the 50 most important features. (a) k-means clusters. (b) spectral clusters.

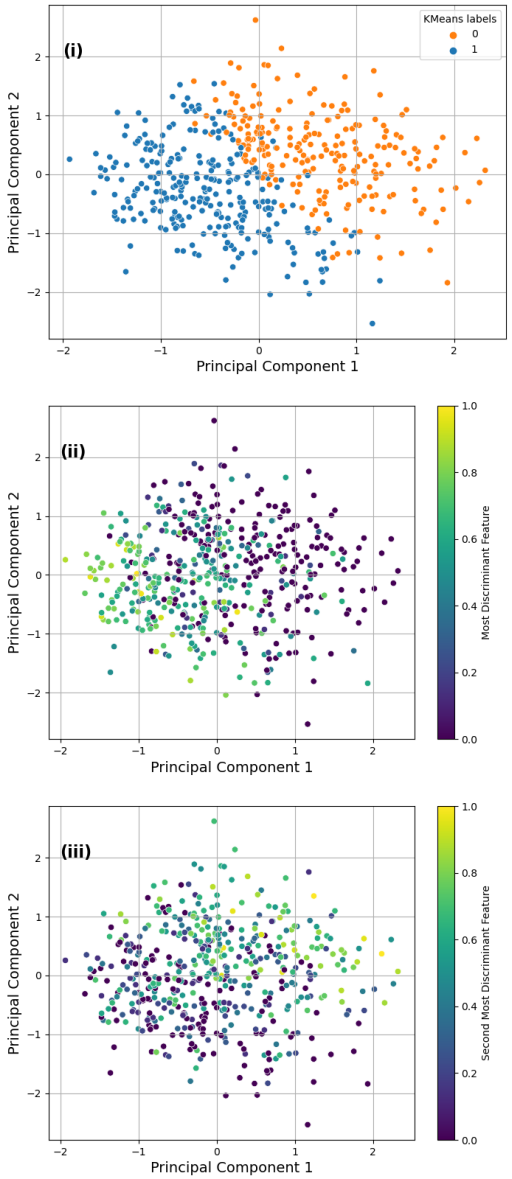

Figure 21: PCA visualisation of dataset `ADS_BaselineDataset.csv`. The points are coloured according to (i) k-means labels, (ii) most discriminant feature, (iii) second most discriminant feature.

References

[1] The Pennsylvania State University - Eberly College of Science. *11.1 - Principal Component Analysis (PCA) Procedure*. STAT 505 - Applied Multivariate Statistical Analysis. Available at: https://online.stat.psu.edu/stat505/lesson/11/11.1.

[2] The Pennsylvania State University - Eberly College of Science. *14.8 - K-Means Procedure*. STAT 505 - Applied Multivariate Statistical Analysis. Available at: https://online.stat.psu.edu/stat505/lesson/14/14.8.

[3] The Pennsylvania State University - Eberly College of Science. *K-Nearest Neighbor Classifiers*. STAT 508 - Applied Data Mining and Statistical Learning. Available at: https://online.stat.psu.edu/stat508/lesson/k.

[4] The Pennsylvania State University - Eberly College of Science. *11.11 - From Bagging to Random Forests*. STAT 508 - Applied Data Mining and Statistical Learning. Available at: https://online.stat.psu.edu/stat508/lesson/11/11.11.

[5] The Pennsylvania State University - Eberly College of Science. *Lesson 22: Kolmogorov-Smirnov Goodness-of-Fit Test*. STAT 415 - Introduction to Mathematical Statistics. Available at: https://online.stat.psu.edu/stat415/lesson/22.

[6] The Pennsylvania State University - Eberly College of Science. *2.2.8 - z-scores*. STAT 200 - Elementary Statistics. Available at: https://online.stat.psu.edu/stat200/lesson/2/2.2/2.2.8.

[7] The Pennsylvania State University - Eberly College of Science. *11.1 - Construct the Tree*. STAT 508 - Applied Data Mining and Statistical Learning. Available at: https://online.stat.psu.edu/stat508/lesson/11/11.1.

[8] The Pennsylvania State University - Eberly College of Science. *11.10 - Bagging*. STAT 508 - Applied Data Mining and Statistical Learning. Available at: https://online.stat.psu.edu/stat508/lesson/11/11.10.

[9] LearnDataSci. *Gini Impurity*. Available at: https://www.learndatasci.com/glossary/gini-impurity.

[10] scikit-learn developers. *Feature importances with a forest of trees*. In: scikit-learn 1.3.2 documentation. Available at: https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html.

[11] Robert J. Tibshirani. *Lasso and Sparsity in Statistics*. Stanford University. Available at: https://ssc.ca/sites/default/files/data/Members/public/Publications/BookFiles/Book/79-91.pdf

[12] *12.1 - Logistic Regression*. STAT 462 - Applied Regression Analysis. The Pennsylvania State University. Available at: https://online.stat.psu.edu/stat462/node/207/.

[13] William Fleshman (2019). *Spectral Clustering: Foundation and Application*. Towards Data Science. Published: February 21, 2019. Available at: https://towardsdatascience.com/spectral-clustering-aba2640c0d5b.

[14] Basil Saji (2023). *Elbow Method for Finding the Optimal Number of Clusters in K-Means*. Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/.

[15] Peter J. Rousseeuw (1987). *Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis*. Computational and Applied Mathematics 20: 53-65. Available at: `https://www.sciencedirect.com/science/article/pii/0377042787901257`.

[16] Leland McInnes, John Healy, and James Melville (2020). *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. arXiv:1802.03426. Available at: `https://arxiv.org/abs/1802.03426`.

Appendix

A README

This repository contains all code necessary to reproduce all results discussed in `report/main.pdf`.

A.0.1 Installation

Clone the repository:

```
$ git clone https://gitlab.developers.cam.ac.uk/phy/data-intensive-science-mphil/m1_assessment/wdk24.
$ cd wdk24
```

To generate the docker image and run the container, navigate to the root directory and use:

```
$ docker build -t <image_name> .
$ docker run -ti <image_name>
```

Alternatively, clone the conda environment in 'environment.yml'.

A.0.2 Usage

The data required for the project is saved in 'data/' (including processed data which is created by running the scripts).

The solutions to the questions are found by running scripts titled `solve_[question-number]_[parts].py`. For example, running `solve_3_abc.py` will yield the solutions to question 3 part a, b and c. Plots will be saved in the `plots/` directory and any additional processed data will be saved in `data/`. Other results will be printed to the terminal.

Below are the commands to generate all the results in the project. The scripts should be run in the order given below, since certain scripts depend on pre-processed data from earlier scripts (although these datasets were included in the repository just in case).

```
$ python src/solve_1_a.py
$ python src/solve_1_b.py
```

```
$ python src/solve_1_cde.py
$ python src/solve_2.py
$ python src/solve_3_abc.py  # 20s
$ python src/solve_3_de.py
$ python src/solve_4_b.py    # 1m 50s
$ python src/solve_4_c.py
$ python src/solve_4_de.py
$ python src/solve_4_f.py
$ python src/solve_5_a.py    # 2m
$ python src/solve_5_bc.py   # 15s
```

The module 'src/utils.py' contains extra functionality which is re-used in various scripts.

A.0.3 Machine Specifications and Timing

I ran all scripts on my personal laptop with the following specifications:

- Chip: Apple M1 Pro
- Total Number of Cores: 8 (6 performance and 2 efficiency)
- Memory (RAM): 16 GB
- Operating System: macOS Sonoma v14.0

Timing for scripts

- `solve_3_abc.py` took 20s
- `solve_4_b.py` took 1 minute 50 seconds
- `solve_5_a.py` took 2 minutes
- `solve_5_bc.py` took 15 seconds
- All other scripts ran in less than 10 seconds