Applied data science coursework

Alberto Plebani
ap2387

Number of words: 2936

Preliminary information

I run the entire code on a Ubuntu-22.04 WSL on my Huawei MateBook 16 laptop with Windows 11. To run the code I used Anaconda 23.9.0. I have allocated 8 GBs of RAM to the WSL, half of the total laptop's available RAM. The laptop has an AMD Ryzen 5 5600 Hz CPU with Radeon Graphics.

- Part 1 took 19 seconds to run with the `--features` flag, and 3 seconds without
- Part 2 took less than 1 second to run
- Part 3 took 19 seconds with the `--heatmap` flag and only 3 seconds without
- Part 4 took less than 4 seconds to run
- Part 5 took 5 seconds to run

Therefore, running all the steps require less than a couple of minutes.
ChatGPT was used to convert the README from .md to a .tex file

1 Section A

In this section I will be presenting the work done on parts 1,2 and 3.

1.1 A-1

The density plot for the first 20 features can be found in Figure 1, whereas Figure 2 displays all 20 density features overlaid in the same plot. We can see how the majority of features have a very narrow peak centred at 0, with only features 5, 11, 14, 18, 19, and 20 displaying a different behaviour.

Afterwards, I applied a 2-dimensional PCA on all 500 features, and I obtained the distribution in Figure 3. We can clearly see two different clusters based on the value of

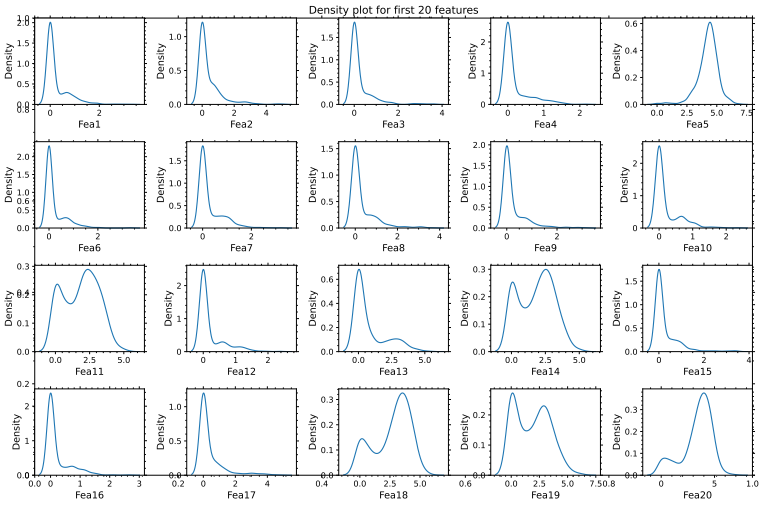


Figure 1: Density plot for all 20 features

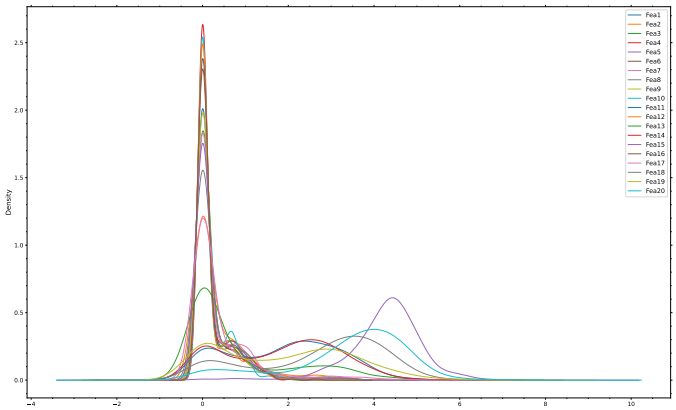


Figure 2: Density plot for all 20 features, overlaid

PC1. Therefore, we can infer that PC1 at low values represents the variables which peak at 0 and at high values represents the other variables.

Next, I split the dataset in two training sets of equal sizes, and applied k-means to to each training set, using 8 clusters. In each case, the unused data was then mapped onto the learned clusters. This is possible because the clustering assignment is based on minimising the distance between the object and the centroid. Because the centroid is the mean position of all the events belonging to the cluster, it is then straightforward to assign the unused data in the nearest centroid. In the code, this is done by fitting k-means to the subset, and then predicting to the entire dataset.

The contingency table is presented in Table 1. We can see that there are many clusters which contain only one event, and we can also note that the contingency matrix is not diagonal. A diagonal contingency matrix implies that the k-means clustering is stable, because the events in a certain cluster fitted on a test set are located in the same

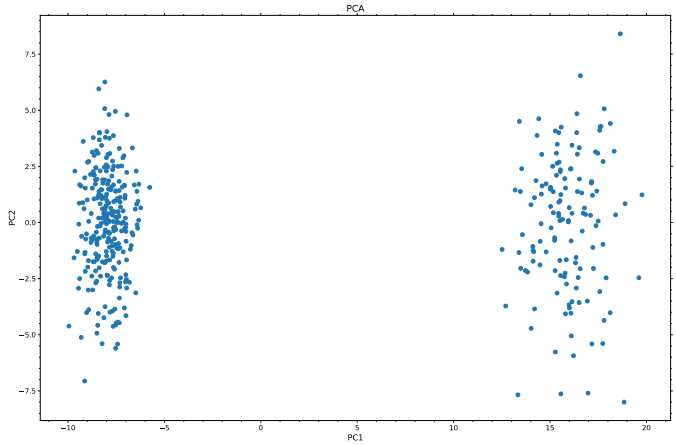


Figure 3: Visualisation of the PCA in a 2D space

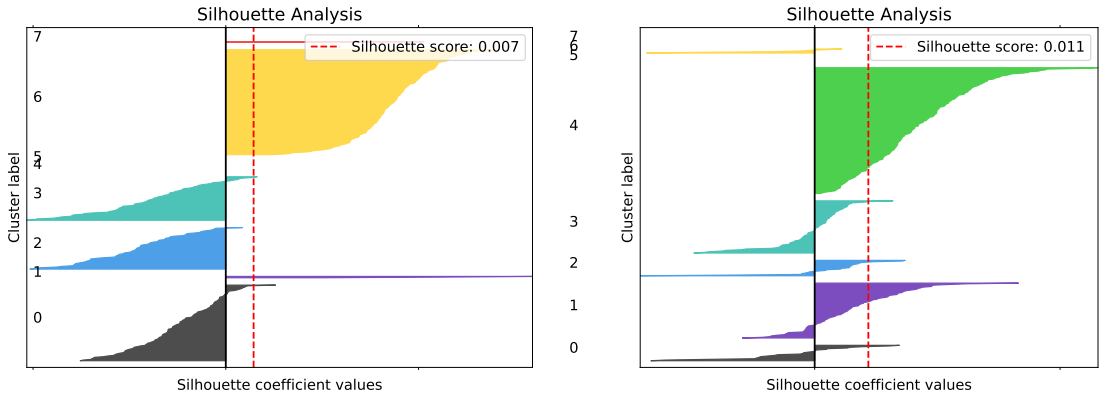
cluster also when fitting on the other set. Additionally, by looking at the silhouette plots in Figure 4, we can see how the clusters are not stable as there are many negative single silhouette values.

		k-means on test 2								
	cluster	0	1	2	3	4	5	6	7	Total
k-means on test 1	0	0	12	0	0	102	0	0	0	114
	1	0	0	0	3	0	0	0	0	3
	2	14	0	6	38	0	1	3	1	63
	3	8	0	18	36	0	0	4	0	66
	4	1	0	0	0	0	0	0	0	1
	5	1	0	0	0	0	0	0	0	1
	6	0	71	0	0	87	0	0	0	158
	7	0	0	0	2	0	0	0	0	2
	Total	24	83	24	79	189	1	7	1	408

Table 1: Contingency matrix displaying the entries in the different clusters for the two subsets

Therefore, I tested different number of clusters, from 2 to 10. I obtained the best configuration with 2 clusters, because the contingency matrix was diagonal (Table 2) and the silhouette scores were the highest Figure 5.

Finally, I identified the clusters within the PCA, as displayed in Figure 6. We can clearly see the separation between the clusters within the PCA. I also tried performing the PCA before applying k-means, and I obtained the same figure. In general, is better to do k-means before because in this way all the information contained in the dataset are used, whereas applying to the PCA means using only a smaller fraction of that. The PCA should be used after the clustering has been performed, as a way to better visualise the



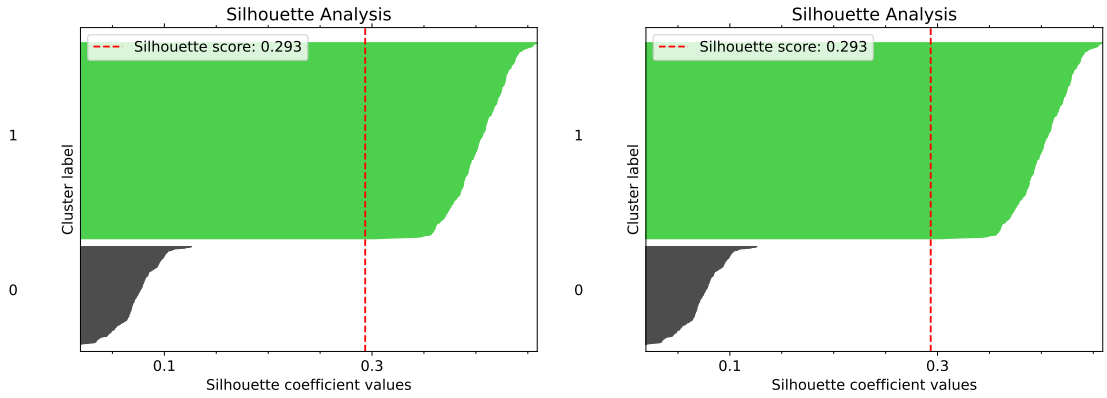
(a) Set 1 (b) Set 2

Figure 4: Silhouette scores for both sets

		set 2		
	cluster	0	1	Total
set 1	0	136	0	136
	1	0	272	272
	Total	136	272	408

Table 2: Contingency matrix displaying the entries in the different clusters for the two subsets

data.



(a) Set 1 (b) Set 2

Figure 5: Silhouette scores for both sets. We can see that they are identical.

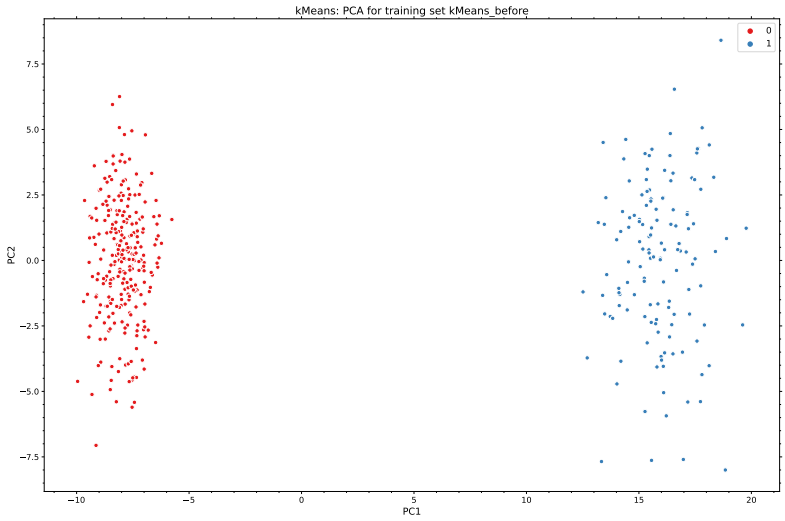


Figure 6: k-means clusters visualised within the PCA

1.2 A-2

The frequency of the labels is presented in Table 3. We can see how we have 20 unlabelled observations out of the 428 events. Among the labelled, the label 1 is the most represented, followed by label 2 and 4.

Label	Number	% on total	% on labeled data
1	179	41.8%	43.9%
2	157	36.7%	38.5%
4	72	16.8%	17.6%
Unlabelled	20	4.76%	-

Table 3: Labels frequency

Obs 1	Obs 2	Label 1	Label	Action
73	145	1	1	Removing 73
219	290	2	4	Removing both
146	408	1	2	Removing both
43	192	1	1	Removing 43
65	252	1	1	Removing 65
27	259	1	2	Removing both
383	395	2	1	Removing both
187	248	1	1	Removing 187
174	310	2	2	Removing 174
165	423	1	1	Removing 165
343	388	1	1	Removing 343
116	296	4	4	Removing 116
350	351	2	1	Removing both
119	358	1	2	Removing both
118	381	4	4	Removing 118
99	172	4	1	Removing both
29	100	2	4	Removing both
45	106	1	1	Removing 45
209	304	1	4	Removing both

Table 4: Summary of the duplicated observations and the action applied

I have then scouted the dataset for duplicated observations, by looking at rows which were identical up to the label. The duplicated observations are presented in Table 4. If also the label was equal, then I just removed one of the two, because these two observations were identical in every aspect. On the other hand, if the label is different, I removed both observations, because there was no way of knowing which one was the correct one. This method is conservative, and it implies a significant loss of data, but since the duplicated observations with wrong labels were only 10, I considered this loss negligible with respect to the total size of the dataset.

The missing data can either by at random (MAR) or not at random (MNAR). The former refers to situations where the missing data is related to a predictor variable that itself is fully recorded, whereas the latter refers to when the value of the missing data is related to the reason for it being missed. When there are observations with missing labels, two different approaches can be considered: omission and imputation. The pros for omission are that it's the simplest method, in that it consists of removing the missing labels. Furthermore, if the missing is completely at random (MCAR), omitting the data will not introduce any bias. However, if the data is not missing at ranodm (MNAR), omitting the data will cause a bias, which is why in these cases it is better to impute the data. This approach is more complicated since it requires the generation of new data, and it can introduce variance, because the imputed data may not be generated in the correct way. On the other hand, the additional gain of using imputation is that no amount of

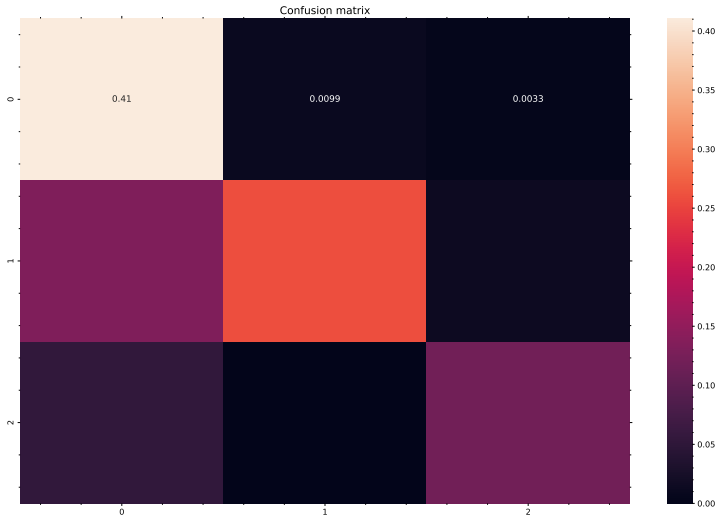


Figure 7: Visualisation of the confusion matrix

data is wasted, which is useful especially when there aren't many observations available.

I have then predicted the missing labels using k-nearest-neighbour (KNN). The confusion matrix can be found in Table 5 and can be visualized in Figure 7. This method was then used to predict the 20 missing labels, with the predictions being:

		True		
	Label	0	1	2
Pred	0	0.41	0.01	0.003
	1	0.13	0.26	0.01
	2	0.05	0	0.12

Table 5: Confusion matrix for the KNN classifier

- Label 1: 15 entries
- Label 2: 3 entries
- Label 4: 2 entries

Therefore, the final labels are displayed in Table 6, with the last column displaying the frequency before the new labels were added. We can see that the difference is minimal, because only 4.76% of the labels were missing.

1.3 A-3

There are 55 missing observations, coming from 5 samples (138, 143, 231, 263, 389) and 11 features (58, 142, 150, 233, 269, 299, 339, 355, 458, 466, 491), as displayed in Figure 8.

Label	Number	% now	% before
1	194	45.3%	43.9%
2	160	37.4%	38.5%
4	74	17.3%	17.6%

Table 6: Labels frequency

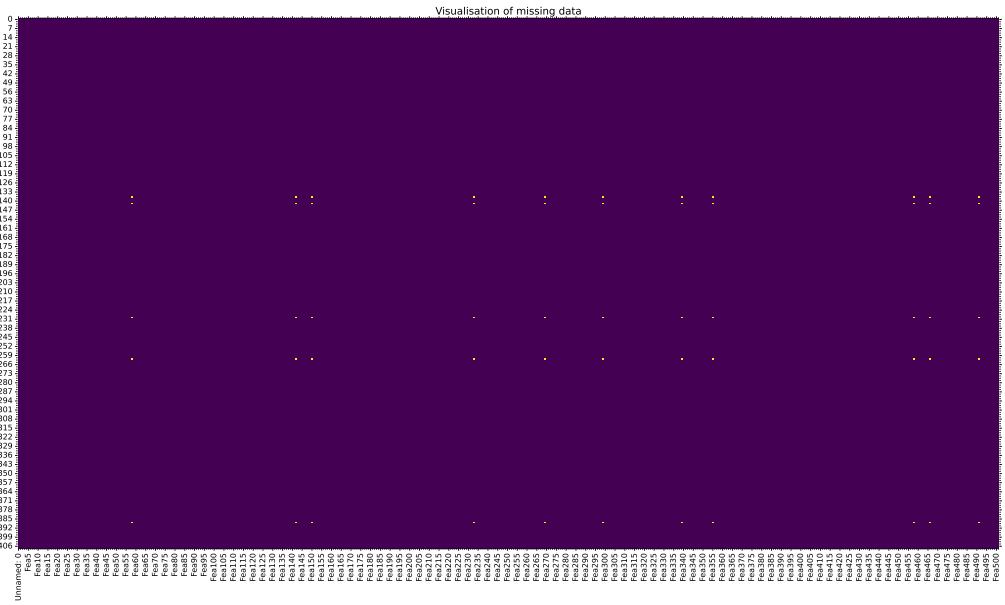


Figure 8: Visualisation of the missing data

The imputation of missing data can be done either with a static or with a model-based method. The static approach can be used for time-series data, simply by carrying forward the last observation. On the other hand, a model-based approach requires the generation of the missing data based on the available data, by predicting it with a generative model. The static approach has the disadvantage of not being able to detect variability, because we force the data to be the same as the last observation, but has the advantage of being the simplest method. The model-based approach instead allows to preserve the relationships which are present in the data, and it reduces the bias especially in situations where the missing is not at random. The con of this method is that it may introduce a variance, which can be reduced with multiple imputations, where the uncertainty associated with the imputation is taken into account.

I decided to impute the missing data using a Gaussian Mixture Model (GMM). The choice was motivated by the features density distributions, where it was clear how the majority of the features had a Gaussian-like shape, as Figure 9.

For handling outliers, two different approaches were considered. The standardisation involves scaling the data so that it has a mean of zero and a variance of 1, by doing $Z = (x - \mu)/\sigma$. If a value is greater than 3σ, then it is considered an outlier. With

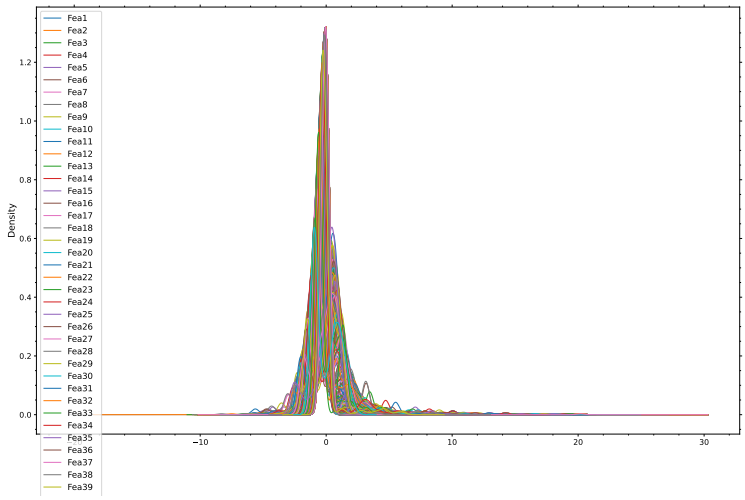


Figure 9: Density distributions of the features

this method, I found 2887 outliers. A different approach is using a model-based classifier algorithm, for instance a GMM. This binary classifier train itself at classifying separately non-outliers (label 0) and outliers (label 1), and then proceeds at substituting the outliers with the mean of the feature in which an outlier was found. Once again, the GMM was chosen because of the shape of the features. The two distributions were then compared with the pairwise distance, whose plot can be found in Figure 10. We can see how the difference between the two distributions is almost negligible, ensuring the imputation worked well.

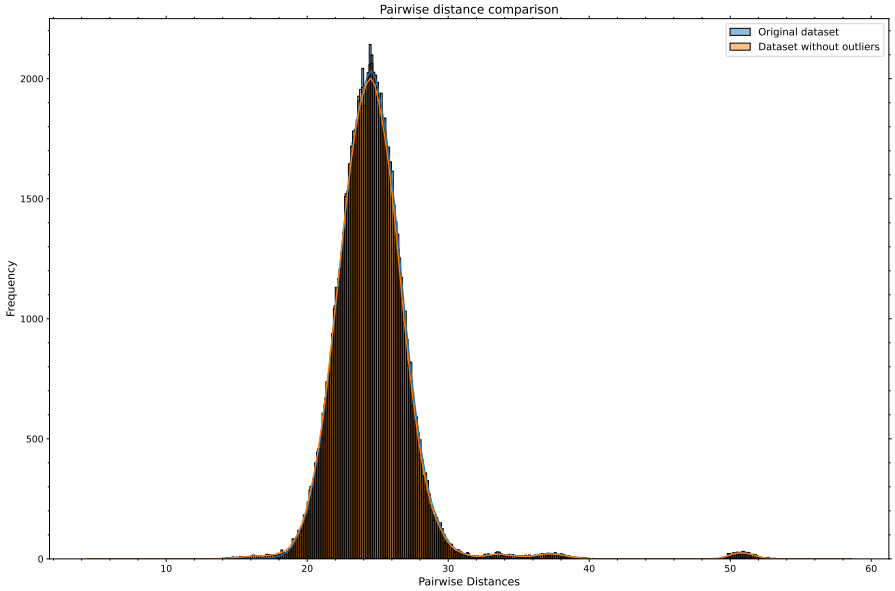


Figure 10: Comparison between original and imputed data with the pairwise distance

2 Section B

2.1 B-4

The decision tree is a supervised algorithm which represents choices and their consequences in the form a tree-like graph. Each node in the graph represents a choice and it is referred to as a leaf, whereas the set of decisions made at the node is called branch. The main limit of the trees is that they are prone to overfitting. Bagging instead is an ensemble technique in which the same model is trained multiple times on different subsets, and then the predictions are averaged, thus reducing the likelihood of overfitting. Finally, the random forest (RF) is a method that uses the same procedure as bagging, with the addition of decreasing the correlation between individual trees which reduces the variance when averaging across them. This also ensure an even lower likelihood of overfitting, and it allows for a greater generalisation of the model.

To measure the quality of the split, the Gini index can be used. The Gini index is calculated by subracting the sum of the squared probabilities of each class from one, as per Equation (1).

$$G = \sum_{i=1}^{N_C} p_i \cdot (1 - p_i) = 1 - \sum_{i=1}^{N_C} p_i^2 \tag{1}$$

When a dataset x splits in two subsets x_i, the Gini index is then the weighted average of the Gini indices of the two subsets. A lower Gini index means that the samples are predominantly from a single class, and thus the forest works on finding the classes that minimise the Gini index.

Two hyperparameters of a single tree within a RF are the maximum depth and the

maximum number of features. The former refers to the maximum depth that a tree can reach, basically limiting its complexity to avoid overfitting. By default in `scikit-learn` this hyperparameter is set to `None`, meaning there is no limit to the complexity of the trees. The latter instead specifies the maximum number of features that are considered by the tree for splitting a node. This parameter introduces randomness in the decision tree, and it can be related to a heuristic involving the number of features by setting $\text{Max}_f = \sqrt{N_f}$, as also set by the default option of the RF classifier provided by `scikit-learn`.

For the random forest classifier, the following pre-processing was applied to the data:

- Looking for empty features: features containing only 0 were discarded because they carry no information. Overall, 41 empty were found, which were subsequently removed from the dataset
- Looking for missing data: no missing observations were found
- Correlation between features: features with a correlation greater than 90% were removed. The highest correlated features are
  - Fea345 and Fea300: correlation 96% $\Rightarrow$ Fea345 removed
  - Fea388 and Fea747: correlation 93% $\Rightarrow$ Fea388 removed
  - Fea869 and Fea954: correlation 90% $\Rightarrow$ Fea869 removed
- Rescaling the data: this pre-processing method was not performed because it is not needed for RF classifier. It was however implemented in Section 2.2, where all the other pre-processings applied here were implemented as well.

The default hyperparameters of the RF were used for the training of the classifier, which include a number of trees equal to 100, alongside the aforementioned maximum depth and number of features. The accuracy of the classifier is 93%, meaning the test set classification error is 0.069.

The number of trees hyperparameter was optimised, testing values ranging from 1 to 131, in steps of 10. For each number of trees, the out-of-box score was then evaluated, with then the accuracy being $1 - \sigma_{OOB}$. The accuracy values are presented in Figure 11. As expected, the accuracy increases with the complexity of the model, eventually reaching a plateau close to 100 trees. The best configuration was obtained with 121 trees, with an accuracy comparable to the one obtained with 100 trees.

This best-model was then retrained, and the feature importance was calculated. In Figure 12, the 20 most important features are displayed. The importance of a feature is a measure of the mean decrease in Giny index when said feature is used for the splitting. The greatest the decrease, the most important the feature. For instance, we can see how of the total 951 used features, only roughly 15 have an impact greater than 1%. Therefore, I decided to re-train the RF using only those 12 features, which are 64, 70, 354, 157, 490, 673, 162, 672, 36, 630, 561 and 795. The number of features to be used was chosen after a discussion among the other CDT students, whereas chatGPT was used for understanding

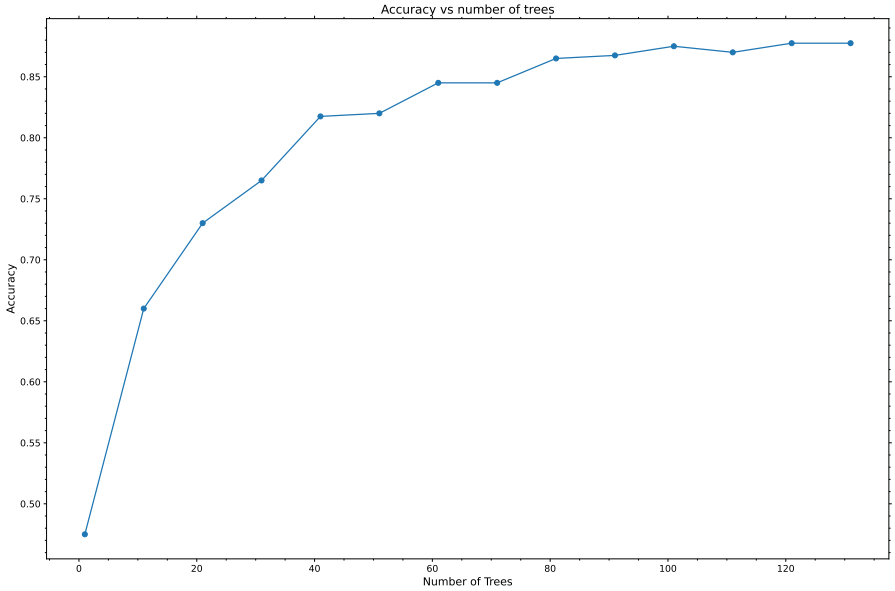


Figure 11: Accuracy vs number of trees

how to properly use the `argsort` function to sort the features by importance (line 128,129 in the code).

As expected, the accuracy decreased to 84% because only a subset of the features was used. When the number of most important features was increased, the accuracy increased, eventually saturating to almost the previous value of 93% when the 60 most important features were used. This means that the remaining features have an almost negligible contribution to the overall classifier.

The same procedure was also applied for the logistic regression classifier, with the addition of the data rescaling as part of the pre-processing. This method is a linear model, and therefore has a lower complexity with respect to the random forest. However, this method is more sensible to outliers, and it is able to capture only linear relationships, as opposed to the RF which can capture any complex non-linear relationship between the input features.

The accuracy obtained with this method was slightly better, 94%, but still comparable to the RF. I then extracted the four most important features for each of the 3 classifiers, and obtained the following:

- Classifier 1: 70, 630, 361, 886
- Classifier 2: 64, 354, 36, 222
- Classifier 3: 181, 243, 414, 170

We can see how features 70, 64, 354 and 630 are present in the 12 most important for both random forest and logistic regression.

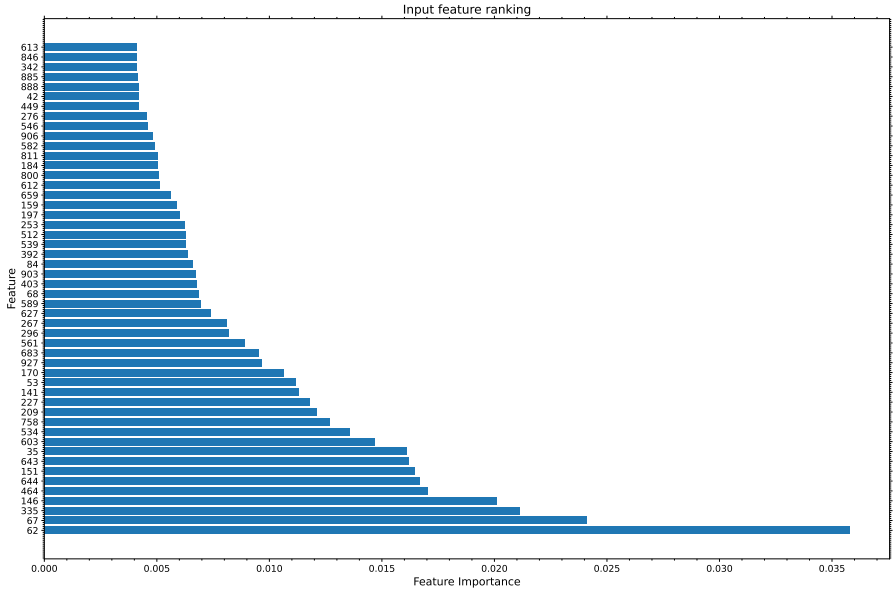


Figure 12: The importance of the 50 most important features

Retraining the logistic regression model with only the 12 best features produced a decrease in the accuracy, down to 84%, comparable with the accuracy of the RF trained using only the 12 features.

Overall, the two different classifiers returned comparable results, both when training on all the features and when using only a subset of the most important ones.

2.2 B-5

Finally, I performed unsupervised learning clustering algorithms. In order to do this, I applied the same pre-processing of Section 2.1, with the addition of rescaling the data. I used k-means and GMM. The former partitions the dataset assigning each point in a cluster, such that the within-cluster sum of squares is minimised. GMM instead represents each cluster with a Gaussian distribution, assuming the data points are a mixture of multiple Gaussian-distributed points. It provides a probabilistic assignment to the clusters, as opposed to k-means. Overall, the k-means method is more computationally efficient, but it can sometimes be unable to handle complex data distributions.

I used three clusters for each of the methods, because we know that the dataset has three labels. Furthermore, I tried also using two clusters, and I obtained a cluster containing only one entry, with the remaining 499 ending up in the other cluster.

The contingency matrix in Table 7 displays the size of the clusters. We can see how the two clustering provide clusters of similar size. However, the same events were not clustered in the same clusters. For instance, no events clustered by k-means in the smaller cluster were clustered by GMM in the smaller cluster. This means that the clustering algorithms didn't perform well, because in a perfect clustering we would have a diagonal

matrix.

		GMM			
	cluster	0	1	2	Total
k-means	0	20	44	139	203
	1	60	0	3	63
	2	128	28	78	234
	Total	208	72	220	500

Table 7: Contingency matrix displaying the entries in the different clusters for the two different methods.

I then trained a logistic regression classifier to both the k-means and the GMM outputs to predict the cluster membership.

For k-means I obtained an accuracy of 79%. I then selected the 4 most important features for each label, and I retrained using only those 12 features:

- Label 0: 270, 361, 162, 426
- 390, 47, 170, 389
- 40, 712, 32, 81

The accuracy dropped significantly to 61%. This is opposite to what was observed in Section 2.1, where using the 12 most important features caused the accuracy to drop only by 94% to 84%, meaning a relative 10% decrease, compared to a 25% here.

For GMM, I obtained an accuracy of 87%, which dropped down by only 1 point percentage (86%) when the first 12 features (181, 243, 385, 170 for the first label; 561, 972, 273, 792 for the second label; 162, 314, 609, 270 for the third label) were used. This means that the majority of the 1000 features wasn't used by the logistic regression for the classification.

In Table 8 we can see a summary of the clusterings and the trainings for k-means and for GMM. We can clearly see how for GMM the training with only 12 features is only slightly worse than the one with all the features, because there is only a small difference in the number of events classified in each label with respect to the training with all features. On the other hand, for k-means it's clear how 12 features aren't enough for the training, since the numbers change significantly from one training to the other.

Finally, in Figure 13 we can see a visualisation of the clusterings in a 2D plane, obtained after performing a PCA. We can see how the GMM identifies correctly the clusters in the 2D PCA plane, whereas for k-means the PCA is not able to describe properly the clustering, since we see many events belonging to different clusters being close to each other.

	k-means			GMM		
Label	Clustering	Pred	Pred-12 feat	Clustering	Pred	Pred-12 feat
0	203	194	211	208	206	207
1	63	45	34	72	61	59
2	234	261	255	220	233	234

Table 8: Summary of the clusterings and the trainings for k-means and GMM, both when training all features or only the 12 most important

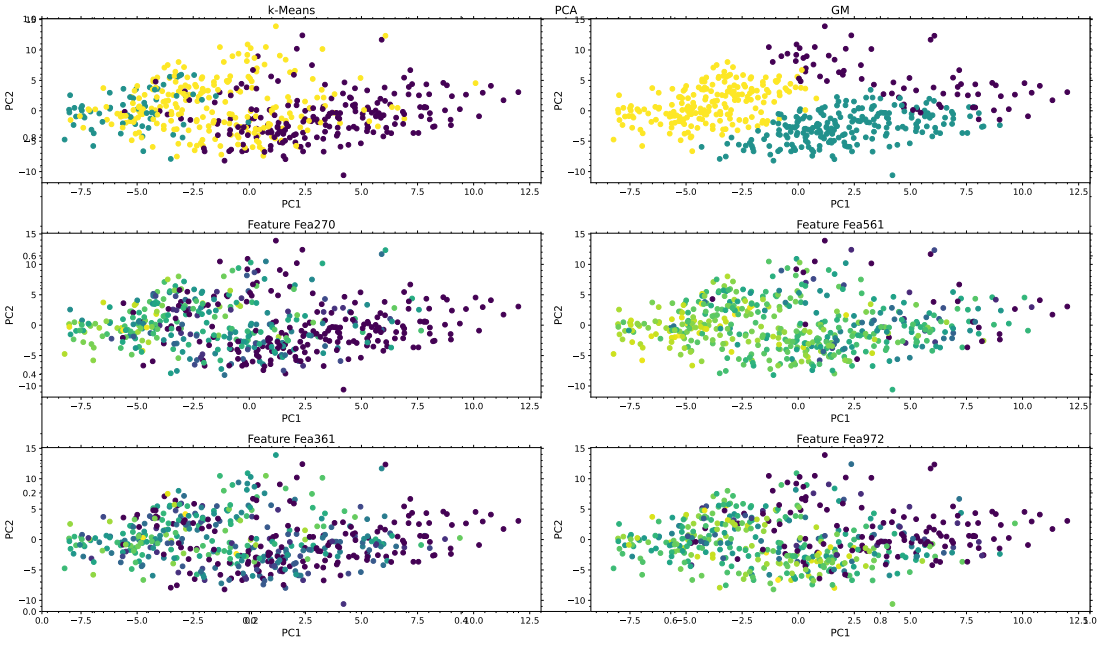


Figure 13: Visualisation of the clusters in a 2D space, with colours given by cluster membership (top), value of most important variable (medium) and value of second most important variable (bottom)

A Appendix: README

README containing instructions on how to run the code for the coursework.

The repository can be cloned with `git clone` git@gitlab.developers.cam.ac.uk:phy/data-intensive-science-mphil/m1_assessment/ap2387.git

The conda environment can be created using the `conda_env.yml`, which contains all the packages needed to run the code

```
conda env create -n mphil --file conda_env.yml
```

Report

The final report is presented in `ap2387.pdf`. The file is generated using LaTeX, but all LaTeX-related files are not being committed as per the instructions on the `.gitignore` file

Code structure

The codes to run the exercises can be found in the `src` folder, whereas the file `Helpers/HelperFunctions.py` contains the definition for the functions used to generate some plots. All plots are stored in the `plots` folder.

SECTION A

Part 1

The code for this exercise can be found in `src/solve_A_1.py`. The code can be run using `parser` options, which can be accessed with the following command

```
python src/solve_A_1.py -h
```

There are two possible flags that can be called: `--features` and `--plots`. The former is used to generate the seaborn pairplot for the first 20 features. This flag was implemented because it takes a lot of time to generate the plot. The latter is a flag that is implemented also in all the other codes, and when set it shows the plots generated while running the code instead of simply saving them in the specific `plots` folder, which for this part is `plots/Section_A_1/`

This code starts by generating density plots for the first 20 features. Then it applies PCA and it visualises the PCA, as well as the explained variance. Afterwards, the code will perform k-means on two subsets of the data, with the default number of clusters (8). After displaying the contingency matrix, the code tests values of k from 2 to 10, and for each it plots the silhouette and it displays the contingency matrix. Finally, the code will display the two clusters on the PCA, and then it will apply k-means after doing the PCA.

These two plots can be found in `plots/Section_A_1/kMeans_kMeans_before.pdf` and in `plots/Section_A_1/kMeans_PCA_before.pdf`, respectively.

The code takes 19 seconds to run without the `--features` flag and 3 seconds without.

Part 2

The code for this exercise can be found in `src/solve_A_2.py`. The `--plots` is the only parser option available.

This code starts showing the labels in the dataset, displaying also the number of missing labels.

After this, the code will look for duplicated rows, displaying which rows are equal, and in particular whether the duplicated rows have also the same label. If the labels are different, both rows are removed from the dataset, otherwise only the first one is kept.

Finally, the code will try to predict the classification labels of the missing labels, plotting the confusion matrix of the k-nearest neighbour classifier and displaying the true labels, the true labels plus the predicted missing labels and also the predicted labels for the whole dataset, to check how well the classifier performed.

The code takes 0.8 seconds to run

Part 3

The code for this exercise can be found in `src/solve_A_3.py`. The `--heatmap` is an additional flag to the `--plots` flag, which plots the `sns.heatmap` plot for the missing features.

This code looks for missing features, displaying which samples and which features have missing, alongside plotting the missing features with `sns.heatmap` if the `--feature` flag is selected, with this plot being saved in `plots/Section_A_3/missing_data.pdf`. Then the code will look for the outliers, first with the standardisation and then with the model-based GMM. With standardisation, the values for which $Z > 3$ are printed out, whereas with the latter, the predicted outliers are removed from the dataframe, and then the datasets with and without the outliers are compared with the pairwise distance comparison, whose plot is saved in `plots/Section_A_3/pairwise.pdf`.

The code takes 19 seconds to run with the `--heatmap` flag and 3 seconds without.

SECTION B

Part 4

The code for this exercise can be found in src/solve_B_4.py. The `-n, --number` option is an additional parser option to the `--plots` flag, and it determines the number of the most important features that can be included in the training in part e), with this value set to default at 12.

The code starts by pre-processing the dataset, displaying also the highest correlated features and the missing data. Then the random forest is applied, and an optimization on the number of trees is performed. The accuracy vs the number of trees is presented in a plot in `plots/Section_B_4/forest_optimisation.pdf`. Then the code takes the `N = args.number` parameter and displays the N most important features with their importance, and then re-trains the forest using only those N features.

Afterwards, the code does the same thing with Logistic regression, printing the 4 best features for each of the three classifiers.

For both the forest and regression, the accuracies for both the full training and the subset training are evaluated and printed out, as well as the classification report.

The code takes 5 seconds to run.

Part 5

The code for this exercise can be found in src/solve_B_5.py. The `--plots` is the only parser option available.

The code starts by doing the same pre-processing as in Part 4. Afterwards, k-means is applied, and a logistic regression classifier is performed on the k-means clusters. Once again, the first 12 most important features are selected, and the classifier is retrained. The same thing is done with the GMM instead of k-means, and the two models are then compared in the contingency matrix and also in a visualization with the PCA (`plots/Section_B_5/PCA.pdf`). Furthermore, accuracy and classification reports are printed for every training.

The code takes 4 seconds to run.