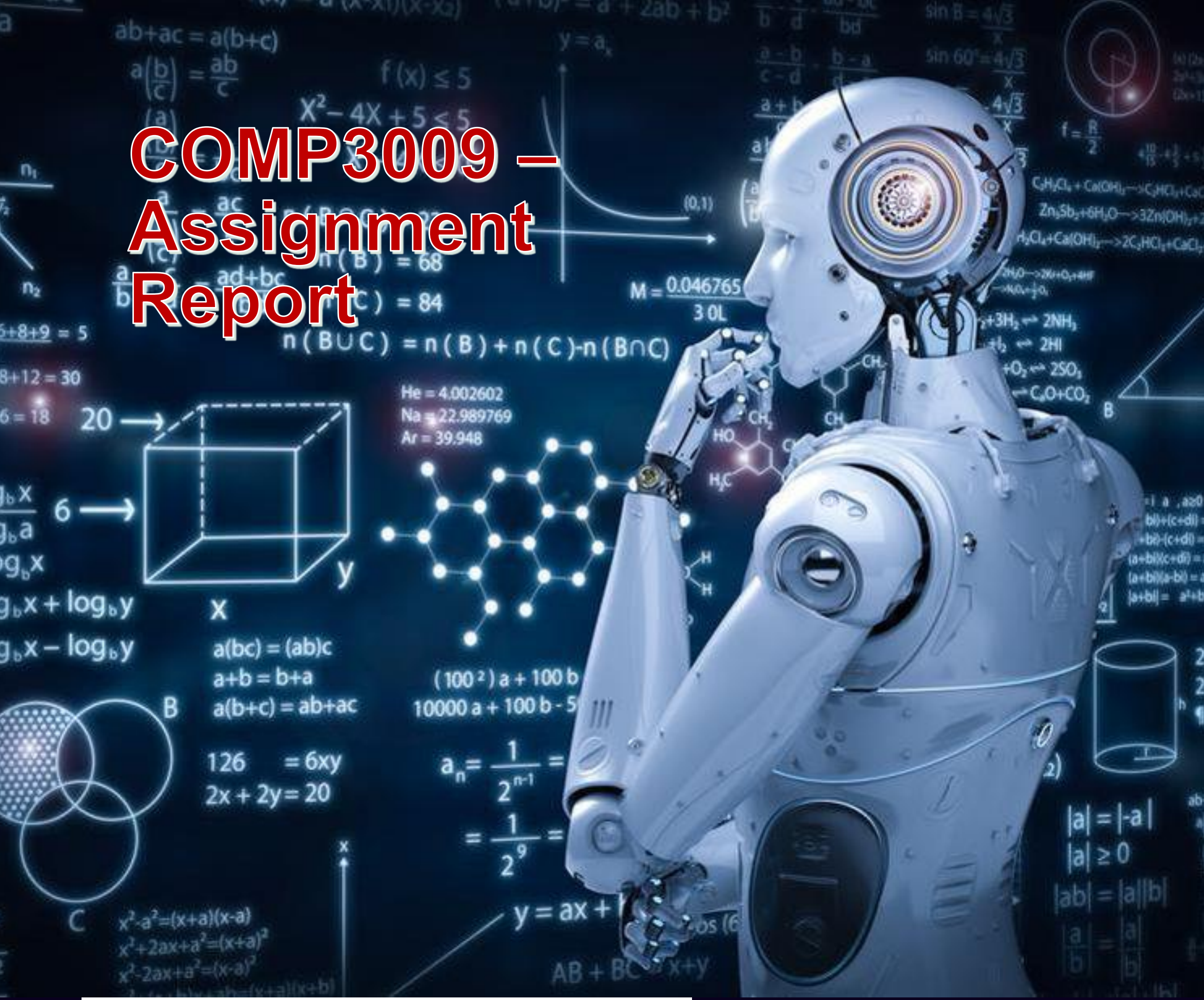
COMP3009 – Assignment Report



Author: Max Collins

Student ID: 1912 6681

Cover image sourced from (phonlamaiphoto)

Contents

Abstract

The objective of the project was to develop a model capable of predicting binary class labels of a mixed-attribute dataset with greater than 75.0% accuracy. Python 3.7.9 (Python Core Team, 2020) and the scikit-learn library (Pedregosa, et al., Scikit-learn: Machine Learning in Python, 2011) were utilised to process and model the data. The dataset was provided with the assignment specification and no collection or domain information was available. The modelling process began with an initial exploratory analysis, during which it was determined that attributes had missing entries, numeric attributes were potentially categorical and there was a major class imbalance with 72.0% of the dataset belonging to the '0' class.

Feature engineering was conducted in multiple stages, with the first consisting of removing columns with only 1 unique value and setting numeric features to categorical if they contained less than 6 unique values. The 'ID' column was removed as it functioned only as an index and columns with more than 90% missing entries were removed. A QR decomposition was used to detect and remove linearly dependent columns from the dataset, following which, duplicate rows were removed. Missing entries were imputed utilising a substitution approach for categorical attributes and a mean approach for numeric. A hot encoding was performed on the categorical attributes and the resulting dataset was split 85% training and 15% testing.

During the preprocessing phase, Synthetic Minority Oversampling Technique (SMOTE) was applied to the training dataset in order to address the class imbalance. The training dataset was then standardized and principal component analysis (PCA) was applied to diagonalize the covariance/correlation matrix. A k-means clustering approach was considered during preprocessing; however, the approach was removed as silhouette scores utilising the Manhattan distance indicated poor clustering for k in the domain [2,14]. Finally, the preprocessing workflow was packaged into a sklearn 'pipeline', for use during model development and k-fold validation.

Naïve Bayes, k-nearest neighbors, logistic regression, decision tree, neural network, random forest and AdaBoost models were developed on the training data. Each model used a grid search to optimise hyper-parameters about the sklearn defaults and was validated on the macro averaged recall value. The models achieved macro averaged recall values of 0.53, 0.66, 0.76, 0.65, 0.77, 0.71 and 0.77 on the test data respectively. A meta voting classifier was then developed using the AdaBoost, neural network and logistic regression models, and achieved a macro average recall score of 0.77 on the test data. Finally, k-fold validation was used to estimate the range of accuracies and recall values for each model; with the AdaBoost and voting classifier being selected for the final predictions.

This project highlighted the importance class balancing techniques and correct performance metrics to ensure that developed models did not favour a dominant class. Maintaining the integrity of the test data was emphasized as training data "bleed in" greatly inhibited model accuracy estimation. The project also demonstrated the importance of feature engineering, as models were significantly improved by sanitizing the dataset.

Introduction

Data classification is the mathematical technique of making decisions based on contextual data and past outcomes. Classification algorithms are widely used throughout many industries; they allow online retail stores to detect fraudulent orders, dating sites to predict the compatibility of couples and doctors to predict whether a given patient has a certain disease (Adhikari, DeNero, Wagner, & Milner, 2019). However, despite their apparent "intelligence", classification algorithms require well formatted and sanitized datasets before they can begin to "capture" any patterns that may be present. As such, it is of interest to study the process of developing and applying classification algorithms to real-world data.

The objective of the project was to develop a model capable of predicting binary class labels of a mixed-attribute dataset with greater than 75.0% accuracy. Python 3.7.9 (Python Core Team, 2020) and the scikit-learn library (Pedregosa, et al., Scikit-learn: Machine Learning in Python, 2011) were utilised to process and model the data. The dataset was provided with the assignment specification and no collection or domain information was available. The analysis was conducted in 2 phases: preparation and classification. During the preparation phase the dataset was sanitized and feature engineering was conducted to improve the overall quality of the data. The classification phase involved fitting a variety of classifiers, tuning hyperparameters and cross-validating models using k-fold validation.

The report outlines the methodology used during data preprocessing and details the data pipelines developed for classification. Model accuracy on the test dataset and the outcomes of k-fold validation are presented alongside estimates of future model performance. Finally, the classifications of the unknown data points are recorded for the AdaBoost and voting classifier.

Methodology

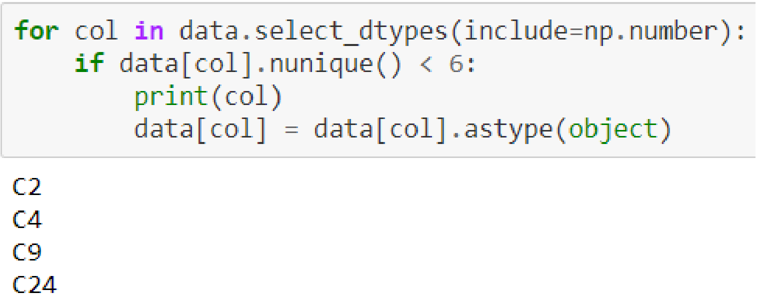
Python 3.7.9 (Python Core Team, 2020) and scikit-learn (Pedregosa, et al., Scikit-learn: Machine Learning in Python, 2011) were utilised to process and model the data. The data was stored in a 'DataFrame' object using the pandas library (The pandas development team, 2020), and all subsequent analysis was performed through the pandas application programming interface (API).

Data Preparation

The dataset was initially stored in a CSV file, which was loaded into Python via the pandas 'read_csv()' method and analysis was conducted on the resulting 'DataFrame' object. The dataset consisted of 34 columns with 1,000 labelled instances and 100 unlabeled instances to classify.

Data Types

Data types describe the set of values that a given data instance can take, examples include numeric data such as rainfall per month and categorical data such as a group of people's favourite candy. Specifying the correct data type for each attribute is important to prevent models misinterpreting input values (Donges, 2018). The pandas library automatically assigns datatypes based on whether a column contains characters or only numeric values. A key concern with the default data type allocation was the possibility of integer numeric values representing categories. Any model build on the dataset would by default interpret these values numerically, and as such potentially assign a numeric relationship between the categories. This would decrease classification accuracy as the meaningless numeric relationship between categories would introduce noise into the model. Given that no domain information was provided, the assignment of numeric attributes to categorical was based on the number of distinct values recorded. If an attribute had less than 6 distinct values, it was assumed to be categorical and was set to the 'object' data type; the numeric attributes 'C2', 'C4', 'C9' and 'C24' were reassigned to categorical in this manner.



```
for col in data.select_dtypes(include=np.number):
    if data[col].nunique() < 6:
        print(col)
        data[col] = data[col].astype(object)
```

```
C2
C4
C9
C24
```

Table 1 Output of Python script assigning columns with less than 6 distinct values to the object data type.

The table below contains the initial data type assigned by the pandas library and the data type post the reassignment process. Entries labelled ‘NA’ denote attributes that were removed before data type reassignment, see the *Irrelevant Attributes* and *Missing Entries* sections for further information.

Initial Data Type		
Attribute	Initial Data Type	Data Type Post Reassignment
ID	Numeric	NA
Class	Numeric	Numeric
C1	Numeric	Numeric
C2	Numeric	String
C3	String	String
C4	Numeric	String
C5	String	String
C6	Numeric	Numeric
C7	String	String
C8	Numeric	Numeric
C9	Numeric	String
C10	String	String
C11	String	String
C12	String	String
C13	Numeric	Numeric
C14	String	String
C15	Numeric	NA
C16	String	String
C17	String	NA
C18	String	String
C19	String	String
C20	Numeric	Numeric
C21	String	NA
C22	Numeric	NA
C23	String	String
C24	Numeric	String
C25	String	String
C26	Numeric	Numeric
C27	String	String
C28	Numeric	NA
C29	String	NA
C30	Numeric	Numeric
C31	String	String

C32	String	String

Irrelevant Attributes

Irrelevant attributes are attributes which provide no relevant information to the classification problem and can reduce the accuracy of models by increasing the “noise” in the data (Gupta, 2019). These redundant attributes can be in the form of attributes that provide inputs uncorrelated with the classification outcome or attributes which are linearly dependent on others. Upon inspection it was determined that the ‘ID’ column functioned as an index for each data instance, and as such it was removed from the dataset. A Python script was then used to detect and remove columns with only one unique value; these were the attributes: ‘C15’, ‘C17’, ‘C21’ and ‘C22’.

Linear dependence occurs when one column is a linear combination of others in the form:

$$column(i) \; = \sum_{\substack{j \in columns \\ j \neq i}} \propto_j \cdot column(j)\,, \propto_j \in \mathcal{R}$$

Linear dependence was analysed after certain numeric attributes had been assigned to categorical as per the methodology outlined in the *Data Types* section. To determine which numeric columns were linearly dependent the numpy library (Harris, et al., 2020) was utilised to apply a QR decomposition to the data. A QR decomposition factors a given matrix into the form:

$$A = QR$$

where Q is an orthogonal matrix and R is upper triangular (Weisstein, 2020). The QR decomposition can be used to determine linear dependence by considering the diagonal entries of the upper triangular matrix R. If the value on the diagonal is 0 the column can be considered dependent on the previous columns. Below is the R matrix generated from the QR decomposition of the data matrix:

C1	C6	C8	C13	C20	C26	C30
-742.374569	-7.42E+02	-118672.371	-136398.276	-948.255813	-1.09E+06	-1.19E+05
0	-1.61E-13	-585.313103	2222.630197	4.748799	1.98E+04	-5.85E+02
0	0.00E+00	-68689.1425	-2677.27633	-56.297503	-2.45E+04	-6.87E+04
0	0.00E+00	0	-87425.983	-535.643648	-6.66E+05	-5.78E-13
0	0.00E+00	0	0	-420.421022	-1.15E+05	-2.24E-13
0	0.00E+00	0	0	0	-9.67E+05	3.02E-13
0	0.00E+00	0	0	0	0.00E+00	-1.73E-11

Table 2 Upper triangular R matrix resulting from application of the QR decomposition to the numeric data.

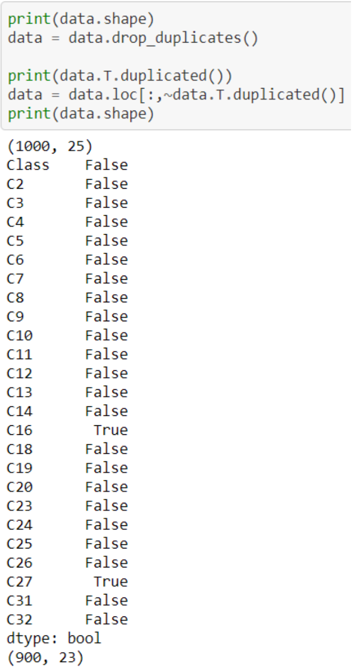
From the QR decomposition we can observe that ‘C6’ and ‘C30’ are linearly dependent. The attributes ‘C1’ and ‘C6’ are linear multiples of one another; however, given that ‘C1’ has missing entries and ‘C6’ doesn’t ‘C1’ and ‘C30’ will be removed.

Irrelevant Attribute Summary	
Attribute	Reasons for Removal
ID	Indexed each instance.
C1	Linearly dependent attribute.
C15	Only contained the numeric value 1.

C17	Only contained the string value ‘F’.
C21	Only contained the string value ‘T’.
C22	Only contained the numeric value 0.
C30	Linearly dependent attribute.

Duplicates

Duplicate data points occur when a data instance or attribute column is repeated in the dataset. This can introduce bias in models constructed on the dataset as each repeated instance increases the weight of the data point and repeated attributes unnecessarily increase the dimensionality of the data. A predictive model is then incentivized to accurately predict and model the weighted points and attributes over all others, which can lead to lower predictive power on the rest of the dataset. The pandas library ‘drop_duplicates()’ method was used to detect and remove 100 duplicate instances within the dataset. To determine if duplicate attributes were present, the categorical and numerical columns were considered separately. A QR decomposition was used on the numerical columns to select and remove linearly dependent rows from the dataset. The columns ‘C6’ and ‘C30’ were removed in this manner; for more information on the usage of the QR decomposition see the *Irrelevant Attributes* section of this report. To detect and remove duplicate categorical columns the pandas ‘duplicated()’ method was applied to the transpose of the dataset. The columns ‘C16’ and ‘C27’ were determined to be duplicates of columns ‘C5’ and ‘C18’ respectively. As such, columns ‘C16’ and ‘C27’ were removed.



```
print(data.shape)
data = data.drop_duplicates()

print(data.T.duplicated())
data = data.loc[:,~data.T.duplicated()]
print(data.shape)
```

```
(1000, 25)
Class    False
C2       False
C3       False
C4       False
C5       False
C6       False
C7       False
C8       False
C9       False
C10      False
C11      False
C12      False
C13      False
C14      False
C16       True
C18      False
C19      False
C20      False
C23      False
C24      False
C25      False
C26      False
C27       True
C31      False
C32      False
dtype: bool
(900, 23)
```

Figure 1 Outout of Python script removing duplicate columns and rows. Note that the data shape goes from (1000,25) to (900,23) indicating the removal of 100 duplicate rows and 2 duplicate categorical columns.

Duplicate Attribute Summary	
Attribute	Reasons for Removal
C1	Linearly dependent attribute.
C16	Duplicate of column C5.
C27	Duplicate of column C18.
C30	Linearly dependent attribute.

Missing Entries

Most machine learning and statistical models are incapable of directly dealing with missing entries within the input matrix (Pereira, 2020). As such it was important to detect and manage the missing entries present within the dataset; the pandas library was used to accomplish this task. The table below displays the attributes with missing entries and what percentage was missing:

Attribute	Percentage Missing
C1	0.5%
C2	0.4%
C14	0.5%
C19	0.4%
C28	99.5%
C29	99.6%

During the analysis the attributes ‘C28’ and ‘C29’ were removed before data type reassignment as they consisted of more than 99% missing values. In this case there is not enough data to accurately impute the missing entries and modelling the unknown entries would provide attributes with less than 0.6% variation in values, thus offering little predictive power. The attribute ‘C1’ was removed during the irrelevant feature analysis as it was determined to be a duplicate of ‘C6’ when missing values were ignored. The attributes ‘C2’, ‘C14’ and ‘C19’ were all categorical, and the missing values were modelled as an ‘Unknown’ category. The ‘Unknown’ category allows the classification models to handle missing entries whilst not making assumptions on their true values.

Final Attribute Selection

After removing irrelevant attributes and those with too greater number of missing entries the following table denotes the final selection of attributes:

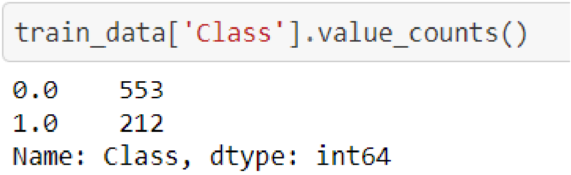
Attribute	Data Type
C2	String
C3	String
C4	String
C5	String
C6	Numeric
C7	String
C8	Numeric
C9	String
C10	String
C11	String
C12	String
C13	Numeric
C14	String
C18	String
C19	String
C20	Numeric
C23	String
C24	String
C25	String
C26	Numeric
C31	String
C32	String

Data Imbalance and Instances

A severe data imbalance introduces major biases within a classification model, which in turn greatly inhibits predictive accuracy (Jayawardena, 2020). To train a classification algorithm an error function is iteratively minimised based on the training data provided. When the training data is primarily from a single class the classification model is incentivized to predict the dominant class over the minority. The statistical likelihood that a given sample comes from either class is not equal, thus the classification model will "learn" to reflect this disparity in likelihood.

The dataset contained a major class imbalance with 72.0% of the dataset belonging to the '0' class before duplicate removal and 72.2% of the dataset belonging to the '0' class post duplicate removal. This would cause all classification algorithms developed on the data to favour the dominant class. The model bias would introduce error into the final prediction as the prediction samples were equally split between the 2 classes. In order to reduce the classification bias, the class imbalance was removed using the synthetic minority over-sampling technique (SMOTE).

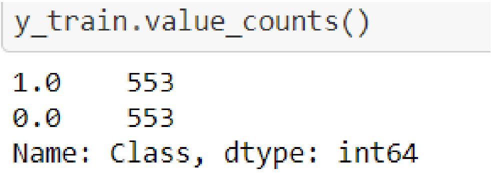
SMOTE is a method of increasing the quantity of the minority class by synthetically injecting new minority samples created from the existing minority instances (Brownlee, SMOTE for Imbalanced Classification with Python, 2020). The SMOTE process works by considering a minority class's k-nearest minority neighbours and selecting a random point between the neighbours to be considered a new minority instance. This process is repeated until enough synthetic minority samples are created to remove the class imbalance in the original dataset. SMOTE is implemented in Python via the imbalanced-learn (imblearn) library (Lemaitre, Nogueira, & Aridas, 2017). The imblearn library utilises an API consistent with scikit-learn which allows the SMOTE technique to be integrated into existing classification pipelines.



```
train_data['Class'].value_counts()
```

```
0.0    553
1.0    212
Name: Class, dtype: int64
```

Figure 2 Value counts of each class within the training data prior to SMOTE. Note the clear class imbalance of 553 instances of the '0' class to 212 instances of the '1' class.



```
y_train.value_counts()
```

```
1.0    553
0.0    553
Name: Class, dtype: int64
```

Figure 3 Value counts of each class within the training data post SMOTE. Note that the classes are now balanced with 553 instances of the '0' class to 553 instances of the '1' class.

Scaling and Standardisation

Standardisation is a process applied to numeric data which centers the mean at 0 and reduces the variance to 1 unit (Jaadi, 2019). The process is applied before principal component analysis (PCA), clustering and any predictive model that is sensitive to the units that the features are modelled in. If all features are measured in the same units then comparisons of distance between features has an underlying meaning. However, if features do not have the same units then comparisons of different features will be misleading and reduce the predictive power of distance-based models such as the k nearest neighbours classifier.

Given that no domain knowledge was provided on the dataset it is unknown whether the numeric features were in the same units. Thus, the distribution of each feature was examined by utilising the pandas library; it was then determined that the numeric features were not already standardised. As the appropriateness of PCA, distance-based classification models and clustering were going to be further explored, all numeric features were standardised via the 'StandardScaler()' class available in the scikit-learn library.

Mean and Standard Deviation Prior to Standardisation					
	C6	C8	C13	C20	C26
Mean	21.334091	3411.954062	5057.852562	34.643256	41929.94363
Standard Deviation	11.712624	2963.947035	918.918508	10.773129	31307.54864

Mean and Standard Deviation Post Standardisation					
	C6	C8	C13	C20	C26
Mean	0.0	0.0	0.0	0.0	0.0
Standard Deviation	1.0	1.0	1.0	1.0	1.0

Feature Engineering and Attribute Selection

Feature engineering is the process by which raw features are transformed to better represent the underlying data structure in order to improve predictive accuracy (Brownlee, Discover Feature Engineering, How to Engineer Features and How to Get Good at It, 2014). The feature engineering methods applied to the dataset were one hot encoding and principal component analysis (PCA). Clustering was considered but was rejected due to the poor clustering solutions achieved.

Many statistical and machine learning models require a purely numeric input matrix, thus categorical variables must be transformed prior to modelling. One-hot encoding is a method of transforming categorical features into a set of numeric features that represent the same attributes. One-hot encoding works by creating a new feature for each unique value within a preexisting categorical feature. Values within the new features are binary and denote whether a given instance was the corresponding value in the original categorical feature. One-hot encoding is implemented via the scikit-learn library (Pedregosa, et al., Scikit-learn: Machine Learning in Python, 2011), and was applied to the dataset prior to further feature engineering.

Principal component analysis (PCA) is applied to numeric data points to project them onto the axis of greatest variability (Brems, 2017). The new axes are referred to as the principal components and they describe how greatly each data instance varies in the new direction. More formally PCA seeks to create a transformation matrix P that rotates the data matrix X such that the covariance or correlation matrix of the rotated data is diagonalised. The principal components can be subsequently analysed to determine their influence on the overall sample variance. Components with little bearing on the total sample variance can then be removed from the model to reduce the dimensionality of the data. Before PCA can be applied to a dataset it is important to standardise the variables so that all variance is not loaded onto a single principal component. PCA is implemented in Python via the scikit-learn library (Pedregosa, et al., Scikit-learn: Machine Learning in Python, 2011). PCA was applied the numeric attributes 'C6', 'C8', 'C13','C20' and 'C26' after they had been standardised. The resulting components explained 33%, 21%, 20%, 18% and 7% of the total sample variance.

Consequently, no components were removed from the model as they all described a large quantity of data variability.

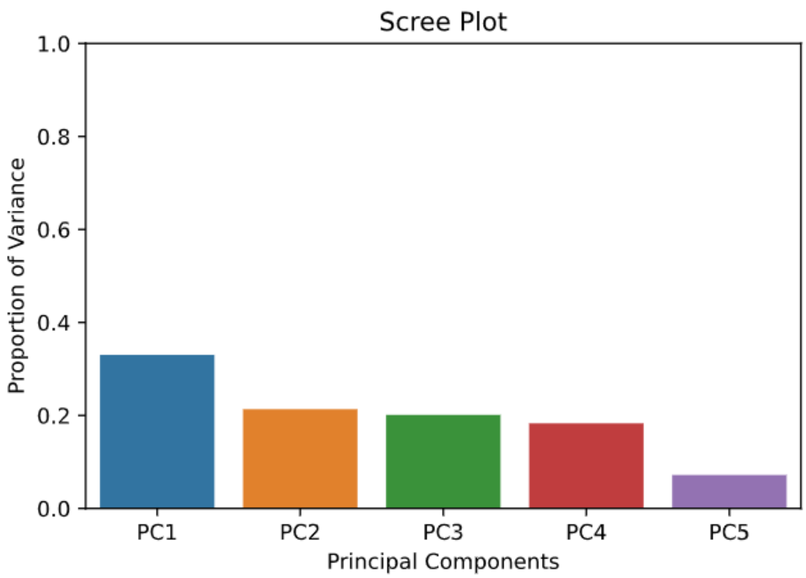


Figure 4 Scree plot displaying the proportion of total sample variance described by the 5 principal components.

	C6	C8	C13	C20	C26
C6	1.000905	0.634972	-0.012737	0.015106	-0.050197
C8	0.634972	1.000905	-0.019942	0.101628	-0.039198
C13	-0.012737	-0.019942	1.000905	-0.01324	0.066617
C20	0.015106	0.101628	-0.01324	1.000905	0.032131
C26	-0.050197	-0.039198	0.066617	0.032131	1.000905

Figure 5 Covariance matrix prior to principal component analysis. Note that it is not diagonal.

	C6	C8	C13	C20	C26
C6	1.65	0.0	0.0	0.0	0.0
C8	0.0	1.07	0.0	0.0	0.0
C13	0.0	0.0	1.01	0.0	0.0
C20	0.0	0.0	0.0	0.916	0.0
C26	0.0	0.0	0.0	0.0	0.360

Figure 6 Covariance matrix post principal component analysis. Note that it is now diagonal.

Clustering is a method of grouping data instances based on their perceived similarity to one another (Brownlee, 10 Clustering Algorithms With Python, 2020). The applicability of clustering the dataset and appending each instance's cluster as a new feature was considered during the feature engineering phase of the project. The clustering method selected was the k-means algorithm utilising the Manhattan distance and the metric for measuring the quality of clustering was the silhouette score. The k-means algorithm attempts to create k clusters which partition the sample space with the least amount of within cluster variance (Brownlee, 10 Clustering Algorithms With Python, 2020). The silhouette score is the mean of the ratio of the average intra-cluster distance and the average distance between a point and the next closest cluster (Pedregosa, et al., sklearn.metrics.silhouette_score, 2020). The silhouette score can take values in the range of [-1,1] with 1 indicating an optimal clustering solution, 0 indicating no clustering and -1 indicating poor clustering. During analysis the data was clustered using k-means with the number of clusters varying over the domain [2,14]. The silhouette score was calculated for each k value and the optimal clustering solution selected. However, the largest silhouette score was 0.109 for 2 clusters. Given the low silhouette score the clustering solution was deemed to provide little benefit to the model and was subsequently removed.

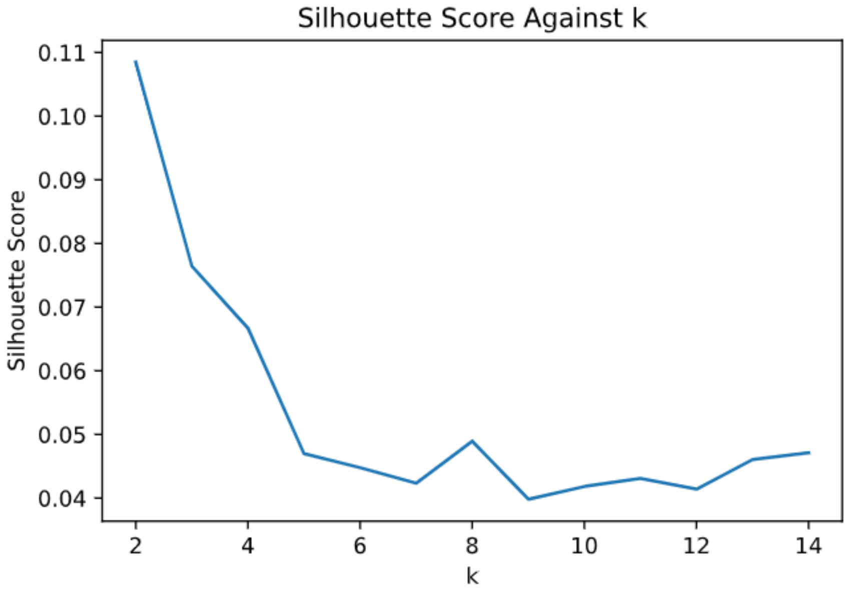


Figure 7 Silhouette score for varying number of clusters (k).

Training and Test Sets

The data was split into a training and testing set prior to model development. The training set consisted of 85% of the dataset, whilst the testing set contained the remining 15%. The training set was further split during 10-fold validation in order to estimate the accuracy of the developed models.

Data Classification

Pipeline Development

The scikit-learn library allows for the development of pipelines, which are data structures that sequentially apply transformations to input data before finally developing a predictive model. Data pipelines allow for complex workflows to be containerised, enabling hyperparameter tuning and k-fold cross-validation to be performed on the entire workflow as opposed to the final predictor. The benefit of modelling the entire workflow is that there is no data leakage between training and validation data sets, which in turn improves the estimated prediction accuracy and quality of hyperparameter tuning.

Each predictive model developed for the dataset utilised the same preprocessing pipeline with the final predictor differing between models. During the preprocessing pipeline the class imbalance was adjusted using SMOTE, the data was then standardised and principal component analysis was applied to the numeric columns. Finally, a predictive model was built on the output data.

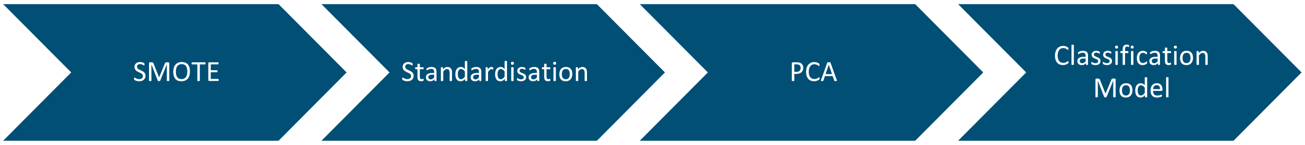


Figure 8 Data pipeline developed for each classification model.

Performance Metrics and Validation

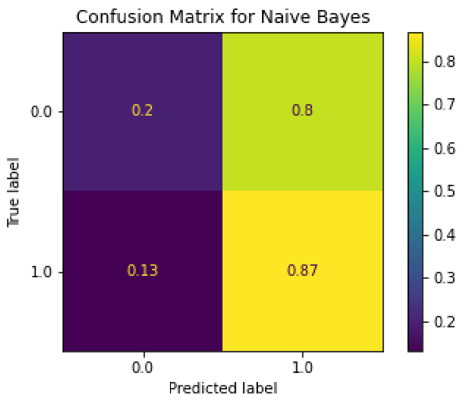
Given that the raw dataset had a considerable class imbalance the macro averaged recall score was selected as the performance metric by which each classifier would be measured. The macro average recall score is the average recall score of the predictions of each class and can be considered as the estimated accuracy of a prediction on a balanced test set. The macro average recall score takes values in the domain [0,1] with 1 denoting perfect recall and 0 denoting the worst possible recall. The recall is the proportion of samples of each class that were correctly classified. The macro averaged recall score is not a default metric in the scikit-learn library, so a custom metric was developed to implement it. The custom metric inherited from the scikit-learn library and maintained the same API so that it could be utilised in grid searches and cross-validation methods.

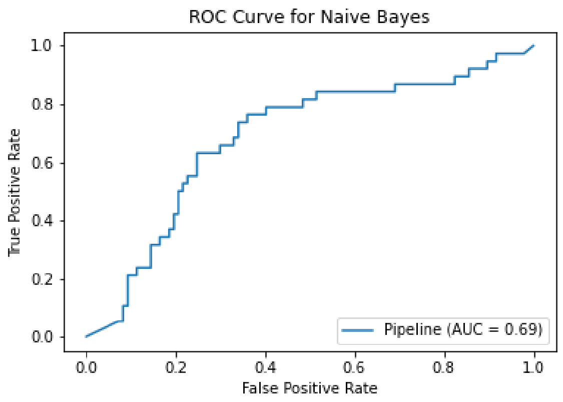
During development the training dataset was sequentially split using 10-fold validation and the model hyper-parameters were optimised in a region about their default values using a grid search. The grid search used 10-fold validation to repeatedly build and test the model using varying hyperparameters. The set of hyperparameters that achieved the greatest performance metric score are then selected and the resulting model is built on the entire training dataset.

Naïve Bayes

The Naïve Bayes classification model is a supervised learning algorithm that makes the naïve assumption that the features are independent of one another (Pedregosa, et al., Naive Bayes, 2020). The naive Bayes model did not require hyperparameter tuning and it achieved a macro average recall score of 0.53 and an accuracy of 39% on the test data.

Naïve Bayes Test Data Results				
	Precision	Recall	F1-Score	Support
0.0	0.79	0.20	0.31	97
1.0	0.30	0.87	0.44	38
Accuracy			0.39	135
Macro Average	0.54	0.53	0.38	135





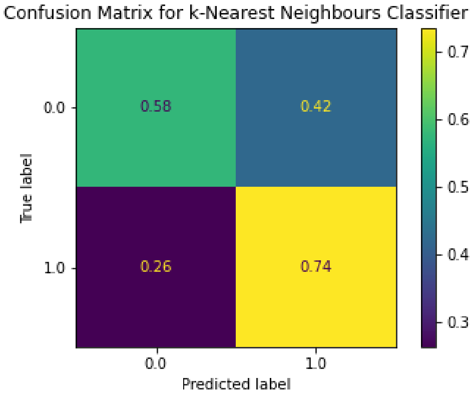
The confusion matrix for the Naïve Bayes model indicates that 20% of class 0 and 87% of class 1 were predicted as class 0 and 1 respectively. We also observe that 13% of class 1 was predicted as class 0 and 80% of class 0 was predicted as class 1.

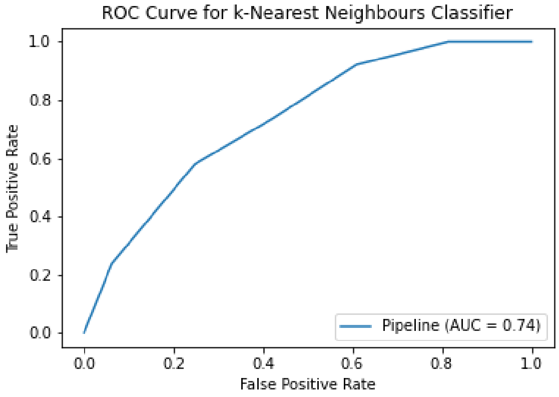
k-Nearest Neighbours

The k-nearest neighbours classification model is a supervised learning algorithm that classifies a sample based on the classification of its closest k samples (Pedregosa, et al., sklearn.neighbors.KNeighborsClassifier, 2020). The k-nearest neighbours model achieved a macro average recall score of 0.66 and an accuracy of 62% on the test data. The following hyperparameters were tuned:

- leaf_size: Denotes the size of the leaf used in the BallTree or KDTree algorithms and affects how quickly a query takes to complete, along with the memory required to store the tree. The leaf_size was optimised on the domain [25, 34] with 25 being selected.
- n_neighbors: Denotes the number of neighbours to consider when classifying a sample. The n_neighbors value was optimised on the domain [5, 9] with 6 being selected.

k-Nearest Neighbours Test Data Results				
	Precision	Recall	F1-Score	Support
0.0	0.85	0.58	0.69	97
1.0	0.41	0.74	0.52	38
Accuracy			0.62	135
Macro Average	0.63	0.66	0.61	135





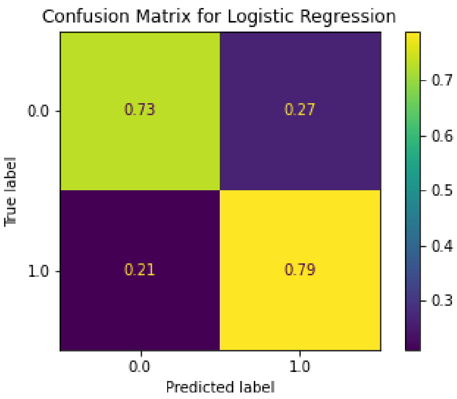
The confusion matrix for the k-nearest neighbours model indicates that 58% of class 0 and 74% of class 1 were predicted as class 0 and 1 respectively. We also observe that 26% of class 1 was predicted as class 0 and 42% of class 0 was predicted as class 1.

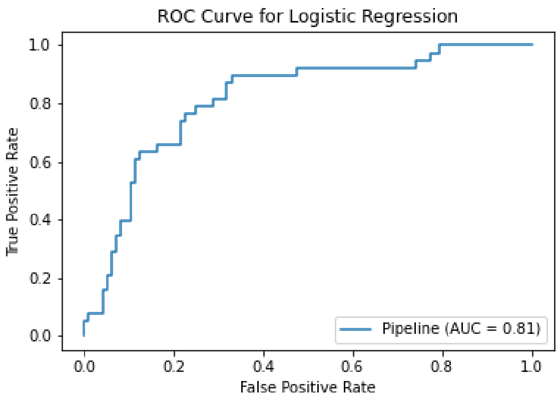
Logistic Regression

The logistic regression classification model is a supervised learning algorithm that uses a logistic function to divide the sample space and make binary classifications (Pedregosa, et al., sklearn.linear_model.LogisticRegression, 2020). The logistic regression model achieved a macro average recall score of 0.76 and an accuracy of 75% on the test data. The following hyperparameters were tuned:

- C: The inverse of the regularization strength which denotes how strongly the model attempts to prevent overfitting. The C value was optimised on the domain [0, 1] with 0.1 being selected.

Logistic Regression Test Data Results				
	Precision	Recall	F1-Score	Support
0.0	0.90	0.73	0.81	97
1.0	0.54	0.79	0.64	38
Accuracy			0.75	135
Macro Average	0.72	0.76	0.72	135





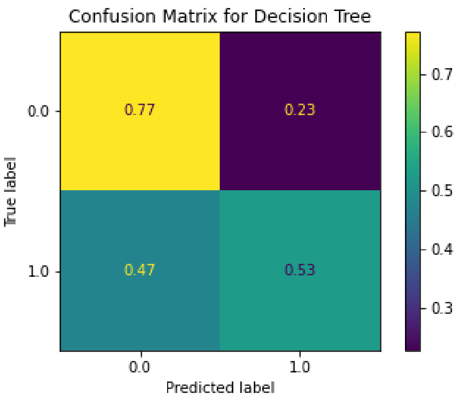
The confusion matrix for the logistic regression model indicates that 73% of class 0 and 79% of class 1 were predicted as class 0 and 1 respectively. We also observe that 21% of class 1 was predicted as class 0 and 27% of class 0 was predicted as class 1.

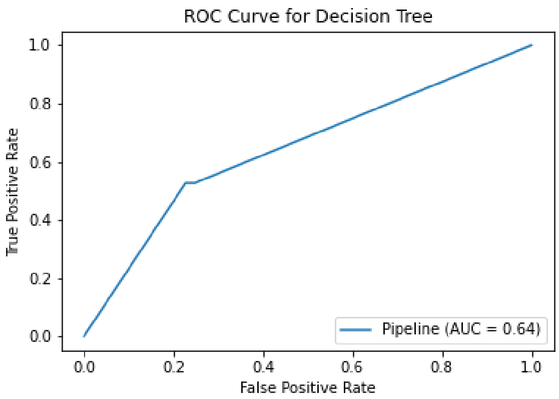
Decision Tree

The decision tree classification model is a supervised learning algorithm that uses a series of decision functions to divide the sample space and make classifications (Pedregosa, et al., sklearn.tree.DecisionTreeClassifier, 2020). The decision tree model achieved a macro average recall score of 0.65 and an accuracy of 70% on the test data. The following hyperparameters were tuned:

- criterion: The criterion function measures the quality of the split. The criterion value was optimised over the functions [‘gini’, ‘entropy’] with ‘entropy’ being selected.
- min_samples_split: Denotes the minimum number of samples required to split an internal node. The min_samples_split value was optimised over the domain [2, 4] with 3 being selected.

Decision Tree Test Data Results				
	Precision	Recall	F1-Score	Support
0.0	0.81	0.77	0.79	97
1.0	0.48	0.53	0.50	38
Accuracy			0.70	135
Macro Average	0.64	0.65	0.64	135





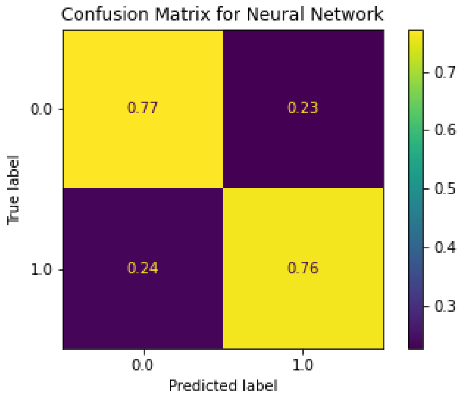
The confusion matrix for the decision tree model indicates that 77% of class 0 and 53% of class 1 were predicted as class 0 and 1 respectively. We also observe that 47% of class 1 was predicted as class 0 and 23% of class 0 was predicted as class 1.

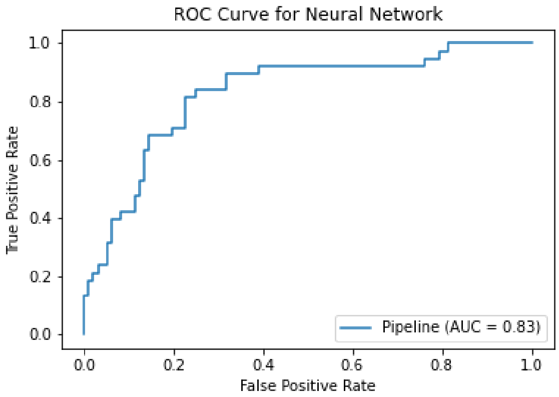
Neural Network

The neural network classification model is a supervised learning algorithm that uses a series of edges and nodes to construct a network capable of making classifications (Pedregosa, et al., Neural network models (supervised), 2020). The neural network model achieved a macro average recall score of 0.77 and an accuracy of 77% on the test data. The following hyperparameters were tuned:

- solver: Denotes the solution method for edge weight optimisation. The solver value was optimised over the functions [‘lbfgs’, ‘sgd’, ‘adam’] with ‘sg’ being selected.
- hidden_layer_sizes: Denotes the number of hidden layers and the size of the layers used in the neural network. The hidden_layer_sizes value was optimised over the networks [(90,), (100,), (110,), (90,90), (100,100), (110,110), (90,90,90), (100,100,100), (110,110,110)] with (100, 100, 100) being selected.

Neural Network Test Data Results				
	Precision	Recall	F1-Score	Support
0.0	0.89	0.77	0.83	97
1.0	0.57	0.76	0.65	38
Accuracy			0.77	135
Macro Average	0.73	0.77	0.74	135





The confusion matrix for the neural network model indicates that 77% of class 0 and 76% of class 1 were predicted as class 0 and 1 respectively. We also observe that 24% of class 1 was predicted as class 0 and 23% of class 0 was predicted as class 1.

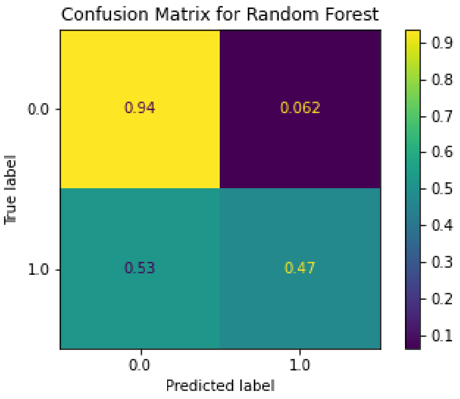
Random Forest

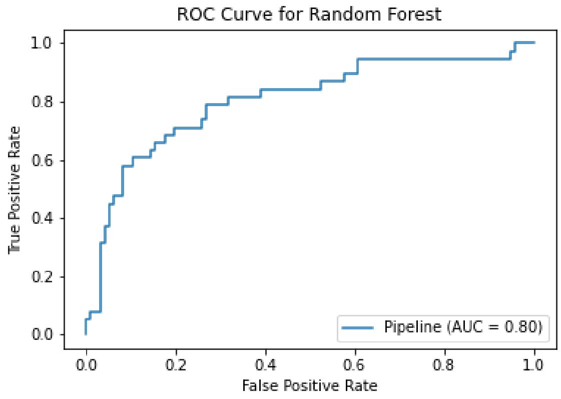
The random forest classification model is a supervised ensemble learning algorithm that creates a “forest” of decision trees that vote when making classifications. The random forest model achieved a macro average recall score of 0.71 and an accuracy of 81% on the test data. The following hyperparameters were tuned:

- criterion: The criterion function measures the quality of the split. The criterion value was optimised over the functions [‘gini’, ‘entropy’] with ‘entropy’ being selected.
- min_samples_split: Denotes the minimum number of samples required to split an internal node. The min_samples_split value was optimised over the domain [2, 4] with 4 being selected.
- n_estimators: Denotes the number of decision trees to “grow” in the model. The n_estimators value was optimised over the domain [70, 130] with 70 being selected.

Random Forest Test Data Results				
	Precision	Recall	F1-Score	Support
0.0	0.82	0.94	0.88	97
1.0	0.75	0.47	0.58	38

Accuracy			0.81	135
Macro Average	0.78	0.71	0.73	135





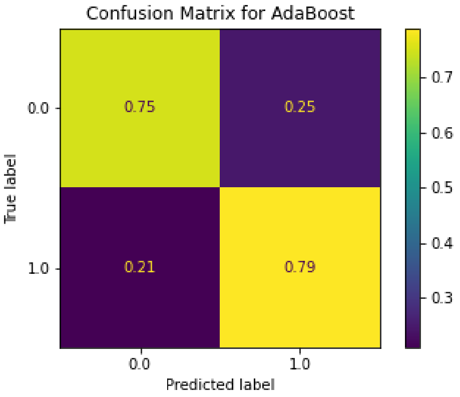
The confusion matrix for the random forest model indicates that 94% of class 0 and 47% of class 1 were predicted as class 0 and 1 respectively. We also observe that 53% of class 1 was predicted as class 0 and 6% of class 0 was predicted as class 1.

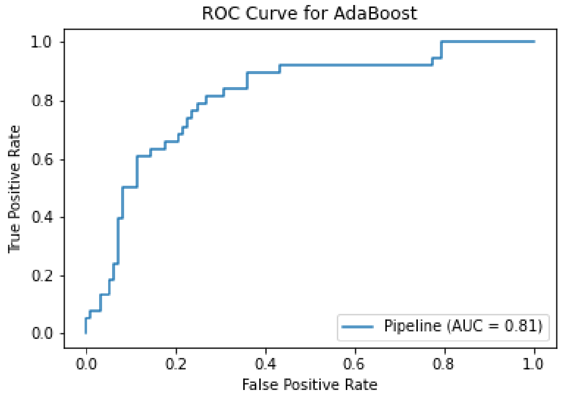
AdaBoost

The AdaBoost classification model is a supervised ensemble learning algorithm that trains multiple models where data points are weighted when they are incorrectly classified by previous models (Pedregosa, et al., sklearn.ensemble.AdaBoostClassifier, 2020). The logistic regression AdaBoost model was selected and it achieved a macro average recall score of 0.77 and an accuracy of 76% on the test data. The following hyperparameters were tuned:

- n_estimators: Denotes the number of logistic regression models to train. The n_estimators value was optimised over the domain [40, 60] with 60 being selected.

AdaBoost Test Data Results				
	Precision	Recall	F1-Score	Support
0.0	0.90	0.75	0.82	97
1.0	0.56	0.79	0.65	38
Accuracy			0.76	135
Macro Average	0.73	0.77	0.74	135



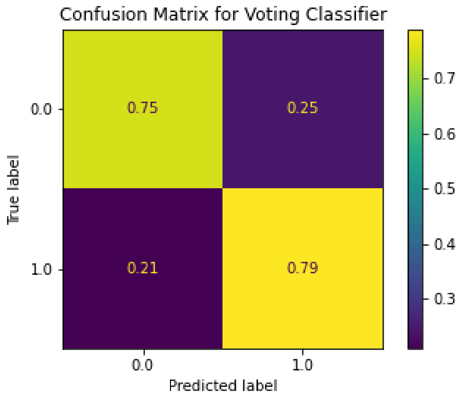


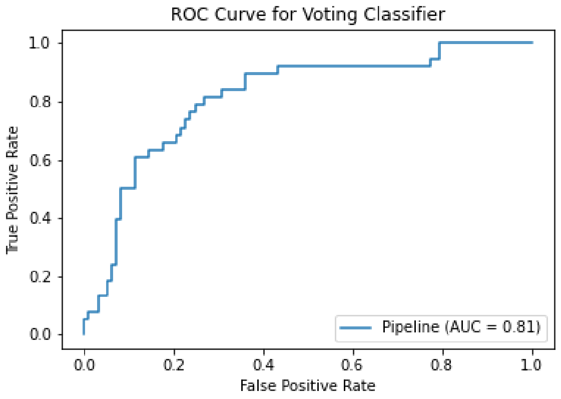
The confusion matrix for the AdaBoost model indicates that 75% of class 0 and 79% of class 1 were predicted as class 0 and 1 respectively. We also observe that 21% of class 1 was predicted as class 0 and 25% of class 0 was predicted as class 1.

Voting Classifier

The voting classifier is a supervised ensemble learning algorithm that trains multiple models which vote on the outcome of a classification problem. The logistic regression, AdaBoost and neural network models were selected for use within the voting classifier, as they had the largest macro average recall scores. The voting classifier achieved a macro average recall score of 0.77 and an accuracy of 76% on the test data. The hyperparameters of the constituent models were selected based on their previous tuning.

Voting Classifier Test Data Results				
	Precision	Recall	F1-Score	Support
0.0	0.90	0.75	0.82	97
1.0	0.56	0.79	0.65	38
Accuracy			0.76	135
Macro Average	0.73	0.77	0.74	135





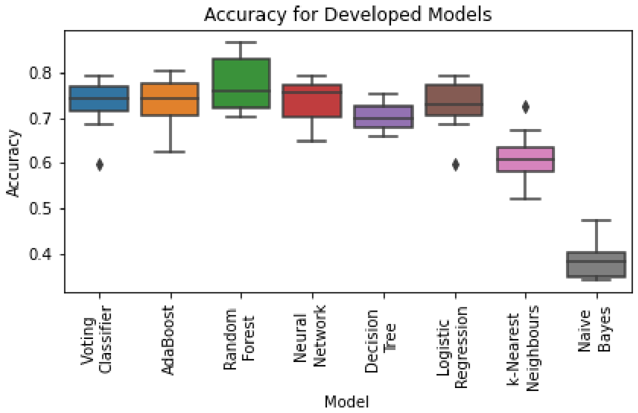
The confusion matrix for the Voting Classifier model indicates that 75% of class 0 and 79% of class 1 were predicted as class 0 and 1 respectively. We also observe that 21% of class 1 was predicted as class 0 and 25% of class 0 was predicted as class 1.

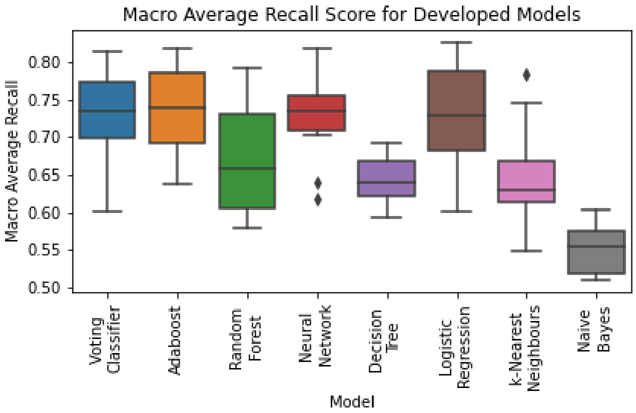
Classifier Comparison

The accuracy and macro average recall scores for each classifier over 10-fold validation is presented below:

Mean Accuracy Over 10-Fold Cross-Validation								
Trial	**Voting Classifier**	**AdaBoost**	**Random Forest**	**Neural Network**	**Decision Tree**	**Logistic Regression**	**k-Nearest Neighbours**	**Naive Bayes**
1	0.779221	0.779221	0.753247	0.792208	0.753247	0.792208	0.727273	0.38961
2	0.714286	0.714286	0.753247	0.701299	0.675325	0.714286	0.636364	0.402597
3	0.792208	0.779221	0.844156	0.766234	0.727273	0.779221	0.597403	0.376623
4	0.597403	0.623377	0.766234	0.649351	0.727273	0.597403	0.519481	0.402597
5	0.714286	0.701299	0.701299	0.662338	0.688312	0.701299	0.519481	0.376623
6	0.776316	0.802632	0.868421	0.776316	0.684211	0.776316	0.631579	0.342105
7	0.684211	0.684211	0.789474	0.710526	0.657895	0.684211	0.618421	0.473684
8	0.75	0.75	0.868421	0.789474	0.710526	0.763158	0.671053	0.342105
9	0.736842	0.736842	0.710526	0.75	0.723684	0.723684	0.592105	0.434211
10	0.75	0.763158	0.710526	0.763158	0.671053	0.736842	0.578947	0.342105
Mean	0.729477	0.733425	0.776555	0.73609	0.70188	0.726863	0.609211	0.388226

Mean Macro Average Recall Score Over 10-Fold Cross-Validation								
Trial	**Voting Classifier**	**AdaBoost**	**Random Forest**	**Neural Network**	**Decision Tree**	**Logistic Regression**	**k-Nearest Neighbours**	**Naive Bayes**
1	0.764973	0.764973	0.588475	0.755898	0.641561	0.791289	0.783575	0.577132
2	0.739564	0.739564	0.676951	0.730944	0.625227	0.739564	0.670145	0.603448
3	0.814412	0.792673	0.78905	0.783414	0.693237	0.805153	0.625604	0.543076
4	0.602166	0.638418	0.635122	0.616761	0.667608	0.602166	0.570621	0.571563
5	0.689858	0.680425	0.57783	0.640723	0.613994	0.669025	0.548349	0.512972
6	0.77682	0.794061	0.741379	0.738506	0.639847	0.77682	0.662835	0.530651
7	0.672941	0.672941	0.700392	0.702745	0.592157	0.672941	0.634118	0.587451
8	0.793989	0.819126	0.79235	0.818579	0.668852	0.827322	0.744809	0.565027
9	0.727806	0.727806	0.640499	0.731842	0.684153	0.717168	0.623991	0.509538
10	0.729798	0.739057	0.594276	0.752525	0.62037	0.707071	0.609428	0.510101
Mean	0.731233	0.736904	0.673633	0.727194	0.644701	0.730852	0.647348	0.551096





The accuracy scores appear to have a lower spread than the macro average recall scores. However, given that the data is imbalanced accuracy does not provide a representative metric for model performance. As such, the recall scores, while having a larger spread are more representative of the true performance on a balanced test dataset. It is evident from the cross-validation data that the voting classifier and AdaBoost meta models consistently achieved the highest macro average recall scores with average recall values of 0.731 and 0.736 respectively. The neural network and logistic regression models had average recall values of 0.727 and 0.731 respectively. However, the logistic regression model had a larger spread than the voting classifier and the neural network was consistently outperformed by the top three models. The Naïve Bayes model was consistently the worst performer with a maximum recall score of 0.603 and a mean score of 0.551.

The estimated model accuracies are based on the mean of the macro average recall score observed during 10-fold cross validation.

Estimated Model Accuracies								
Model	**Voting Classifier**	**AdaBoost**	**Random Forest**	**Neural Network**	**Decision Tree**	**Logistic Regression**	**k-Nearest Neighbours**	**Naïve Bayes**
Estimated Accuracy	73.1%	73.7%	67.4%	72.7%	64.5%	73.1%	64.7%	55.1%

Prediction

The unknown values will be predicted using the AdaBoost and voting classifiers as they had the largest estimated accuracies of 73.7% and 73.1% respectively.

ID	AdaBoost Prediction	Voting Classifier Prediction
1001	0	0
1002	1	1
1003	1	1
1004	0	0
1005	0	0
1006	0	0
1007	1	1
1008	1	1
1009	1	1
1010	1	1
1011	0	0
1012	0	0
1013	1	1
1014	0	0
1015	0	0
1016	1	1
1017	1	1
1018	1	1
1019	1	1
1020	1	1
1021	1	1
1022	1	1
1023	0	0
1024	0	0
1025	1	1
1026	1	1
1027	0	0
1028	0	0
1029	1	1
1030	1	1
1031	1	1
1032	0	0
1033	0	0
1034	1	1
1035	1	1
1036	1	1
1037	1	1
1038	0	0
1039	1	1
1040	1	1
1041	0	0
1042	1	1
1043	1	1
1044	0	0
1045	1	0
1046	1	1
1047	1	1
1048	0	0
1049	0	0
1050	0	0
1051	1	1
1052	0	0
1053	0	0
1054	0	0
1055	0	0
1056	1	1
1057	0	0

1058	1	1
1059	1	1
1060	1	1
1061	1	1
1062	0	0
1063	0	0
1064	0	0
1065	0	0
1066	1	1
1067	1	1
1068	1	1
1069	0	0
1070	0	0
1071	0	0
1072	1	1
1073	1	1
1074	0	0
1075	1	1
1076	0	1
1077	0	0
1078	1	1
1079	0	0
1080	0	0
1081	0	0
1082	0	0
1083	0	0
1084	0	0
1085	1	1
1086	1	1
1087	0	0
1088	0	0
1089	1	1
1090	1	1
1091	1	1
1092	1	1
1093	0	0
1094	0	0
1095	0	0
1096	0	0
1097	1	1
1098	1	1
1099	0	0
1100	0	0

Conclusion

The construction of the classification models was undertaken in two phases: data preparation and data classification. During the data preparation phase irrelevant features were removed, missing values were imputed, numeric features were standardised, principal component analysis was applied, and categorical features were hot encoded. During the classification phase, data pipelines were developed to containerise the workflow and prevent data leakage contaminating the grid search and cross-validation processes. A series of classification models were then developed on the data and their hyperparameters were tuned to maximise the macro average recall score. The AdaBoost and voting classifiers achieved the greatest recall scores of 73.7% and 73.1% respectively. As such, they were selected to classify the 100 unknown samples. Unfortunately, neither the AdaBoost nor Voting classifiers achieved an estimated accuracy of above 75%, which is below the desired accuracy level in the assignment specification. Given further feature engineering and the development of larger meta models it may have been possible to achieve the desired accuracy.

References

Adhikari, A., DeNero, J., Wagner, D., & Milner, H. (2019). *Computational and Inferential Thinking.* GitBook.

Brems, M. (2017, April 18). *A One-Stop Shop for Principal Component Analysis*. Retrieved from Towards Data Science: https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

Brownlee, J. (2014, September 26). *Discover Feature Engineering, How to Engineer Features and How to Get Good at It*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/

Brownlee, J. (2020, April 6). *10 Clustering Algorithms With Python*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/clustering-algorithms-with-python/

Brownlee, J. (2020, January 17). *SMOTE for Imbalanced Classification with Python*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Donges, N. (2018, March 18). *Data Types in Statistics*. Retrieved from Towards Data Science: https://towardsdatascience.com/data-types-in-statistics-347e152e8bee#:~:text=Datatypes%20are%20an%20important%20concept,result%20in%20a%20wrong%20analysis.

Gupta, T. (2019, August 20). *Data Preprocessing in Data Mining & Machine Learning*. Retrieved October 15, 2020, from Towards Data Science: https://towardsdatascience.com/data-preprocessing-in-data-mining-machine-learning-79a9662e2eb

Harris, C. R., Millman, K. J., Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., . . . A. (2020). Array programming with NumPy. *Nature*, 357-362. doi:https://doi.org/10.1038/s41586-020-2649-2

Jaadi, Z. (2019, September 4). *When and Why to Standardize Your Data?* Retrieved from Built In: https://builtin.com/data-science/when-and-why-standardize-your-data

Jayawardena, N. (2020, June 22). *How to Deal with Imbalanced Data*. Retrieved from Towards Data Science: https://towardsdatascience.com/how-to-deal-with-imbalanced-data-34ab7db9b100#:~:text=A%20dataset%20with%20imbalanced%20classes,as%20a%20common%20interview%20question.&text=The%20most%20common%20areas%20where,fraud%20detection%20and%20medical%20diagnosis.

Lemaitre, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *Journal of Machine Learning Research*, 1-5. Retrieved from http://jmlr.org/papers/v18/16-365

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 2825-2830. Retrieved from https://scikit-learn.org/stable/about.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2020). *Naive Bayes*. Retrieved from scikit learn: https://scikit-learn.org/stable/modules/naive_bayes.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2020). *Neural network models (supervised)*. Retrieved from scikit learn: https://scikit-learn.org/stable/modules/neural_networks_supervised.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2020). *sklearn.ensemble.AdaBoostClassifier*. Retrieved from scikit learn: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2020). *sklearn.linear_model.LogisticRegression*. Retrieved from scikit learn: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2020). *sklearn.metrics.silhouette_score*. Retrieved from scikit learn: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2020). *sklearn.neighbors.KNeighborsClassifier*. Retrieved from scikit learn: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2020). *sklearn.tree.DecisionTreeClassifier*. Retrieved from scikit learn: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Pereira, T. (2020, January 9). *The Problem of Missing Data*. Retrieved from Towards Data Science: https://towardsdatascience.com/the-problem-of-missing-data-9e16e37ef9fc

phonlamaiphoto. (n.d.). robot with education hud. *robot with education hud.* Retrieved from https://stock.adobe.com/au/199085055?as_campaign=TinEye&as_content=tineye_match&epi1=199085055&tduid=f29d2133b65c3a60116de4d9b2130177&as_channel=affiliate&as_campclass=redirect&as_source=arvato

Python Core Team. (2020). Python: A dynamic, open source programming. Python Software Foundation. Retrieved from https://www.python.org/

The pandas development team. (2020, Febuary). reback2020pandas. Zenodo. Retrieved from https://doi.org/10.5281/zenodo.3509134

Weisstein, E. W. (2020). *QR Decomposition*. Retrieved from MathWorld--A Wolfram Web Resource: https://mathworld.wolfram.com/QRDecomposition.html