Mining Loans Data to Predict the Viability of Future Loans, at the Application Stage

100340229 - Nathan Page

Abstract

This report examines a loans dataset, with the objective of implementing data mining techniques to determine whether a loans provider should grant a loan to a potential customer, at the application stage, by predicting whether it will be fully paid. First, the dataset is examined, and visualisation techniques are utilised to further comprehend both the features and individual loan instances, to aid in efficient cleaning and pre-processing of the data, and effective model selection. Next, supervised learning algorithms are trained, tested, and evaluated. Finally, unsupervised k-means clustering is used on the dataset to classify the loan instances. It is found that logistic regression produces the best (supervised) results in terms of balanced accuracy (0.661) and AUROC (Area Under ROC) scores (0.724), though the results themselves are underwhelming, suggesting that data mining could be used to supplement the loan approval process, as opposed to being the basis of it. K-means clustering proves to be poor at categorising the data and only just outperforms random classification.

Key words: Data Mining, K-Means Clustering, Logistic Regression, Machine Learning

Contents

1 Introduction

The aim of the data mining procedure implemented here is to analyse loans data to predict whether a loan should be granted by a hypothetical lender, **at the application stage**. Therefore, for this implementation, many valuable features need to be discarded as they will not be known at the time of application. This methodology was chosen to mimic a real-life, industry-applicable implementation of data mining. To clarify, the goal of the model is to predict, based on certain features available at the time of the loan application, whether a future loan is likely to be paid off (in full) or not. The model will then be evaluated by comparing the predictions generated by the model, to the actual outcomes of the loans, as denoted by the 'loan_status' of the loans provided. For the supervised portion of this report, the model was provided with the labels to aid in training. For the unsupervised section, the clustering model is unaware of any true 'loan_status' categories.

2 Summary of Features

The first step of the data mining process followed here is to understand the data and the domain. The data, datatypes, and features are summarised, and visualisation techniques are implemented to help with the discovery of potentially important patterns. The specific steps taken are delineated in the subsections below.

Note: *For the sake of brevity, only one example per subsection is delineated here. For example, when outliers are explored (Section 2.6), only the plot for the 'installment' variable is shown (to demonstrate the procedure undertaken), and while all features were similarly analysed, there is not enough space to plot or describe each instance here.*

2.1 Data Download and Setup

The *'LendingClubLoans2018-2020.xlsx'* file is downloaded locally and the first sheet (containing the dataset itself) is converted into a csv file for quicker data processing. The format of certain date columns is altered within the csv file to ensure the dates can be parsed correctly. The file path is passed in to the pandas.read_csv() function[1] and saved to a dataframe object entitled *loans_data*.

2.2 Initial Analysis

The dataset has 108 columns: 107 features, one target variable ('loan_status'), and 77,159 rows (individual loans to be considered). The first task is to understand the data and the domain through comprehension of the data dictionary, in conjunction with a basic visual analysis of the data itself. It is clear from reading the data dictionary, that certain features ('int_rate', 'term') will be useful for the model. Other features are clearly irrelevant ('id') or not usable ('last_fico_range_high'). Features like 'last_fico_range_high' (removed in Section 3) would not be known at the time of application and would invalidate the model. To elaborate, if a borrower has either defaulted, or had their loan charged off, their last FICO score would suffer drastically as a result and thus, this feature highly correlates with loan status (See Figure 1) but this feature would be invalid to use as it will not be known by the lender at the time a borrower is applying for a loan.

A third category of features exists that will require further exploration and better domain understanding to better determine utility, and this exploratory analysis is continued below.

[1] pandas.read_csv: pandas.pydata.org/docs/reference/api/pandas.read_csv.html

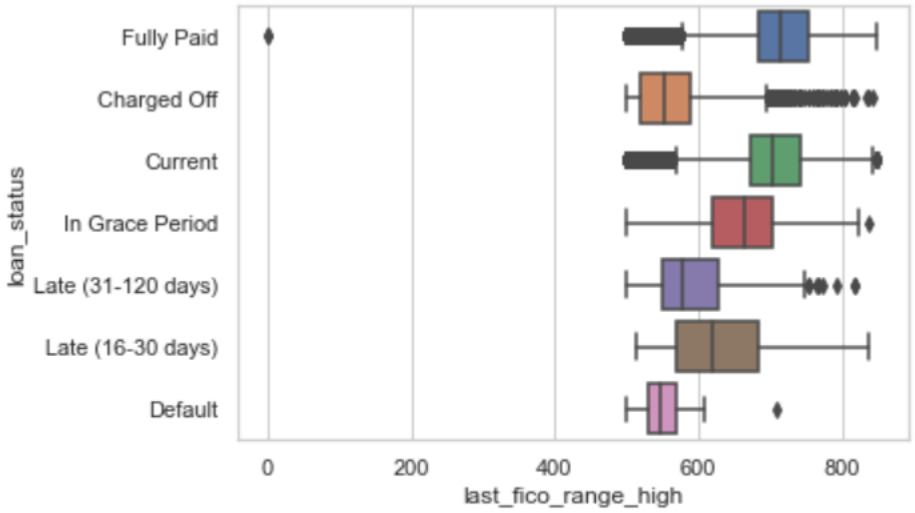


Figure 1: The 'last_fico_range_high' feature, grouped by 'loan_status'

2.3 Data Types

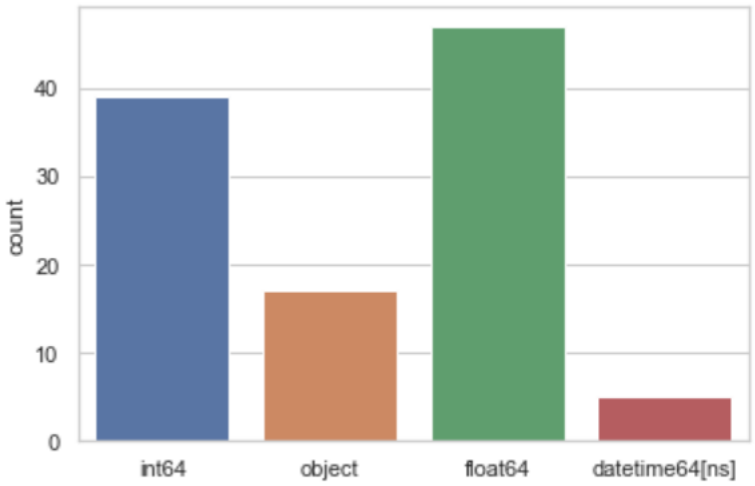


Figure 2: Initial Data Types

There are essentially three types of data to consider in our dataset (Figure 2). Numerical data (integers and floats), categorical/string data (Objects) and dates (in the form of Datetime values). These datatypes each need to be considered, cleaned, pre-processed, and potentially encoded separately, and in different ways (Section 3).

2.4 Target Variable

Figure 3 examines the target variable 'loan_status', grouped by category, and displays the number of instances in each. The dataset is unbalanced, with very few instances of 'In Grace Period', 'Late (16-30 days)', 'Late (31-120 days)', or 'Default'. This underlying class imbalance is addressed later in Section 4.4.

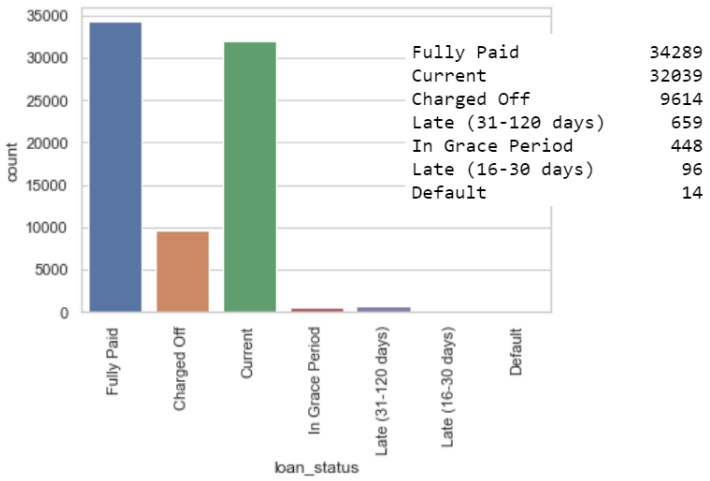


Figure 3: Target Variable Categories

2.5 Missing Data

Numerous columns contain missing values (Table 1), and these need to be accounted for. First it is essential to understand why these values are missing, and then some form of appropriate imputation may be necessary. For columns with massive amounts of data missing (over 50%), these may need to be dropped totally from the dataset.

Feature Name	Percentage of Missing Values
hardship_status	94.18%
payment_plan_start_date	94.18%
hardship_reason	94.18%
hardship_type	94.18%
deferral_term	94.18%
orig_projected_additional_accrued_interest	92.24%
hardship_last_payment_amount	92.07%
hardship_amount	92.07%
hardship_payoff_balance_amount	92.07%
verification_status_joint	87.84%
dti_joint	87.84%
revol_bal_joint	87.84%
annual_inc_joint	87.84%
mths_since_last_record	85.28%
mths_since_recent_bc_dlq	77.46%
mths_since_last_major_derog	74.24%
mths_since_recent_revol_delinq	67.12%
next_pymnt_d	56.90%
mths_since_last_delinq	51.38%

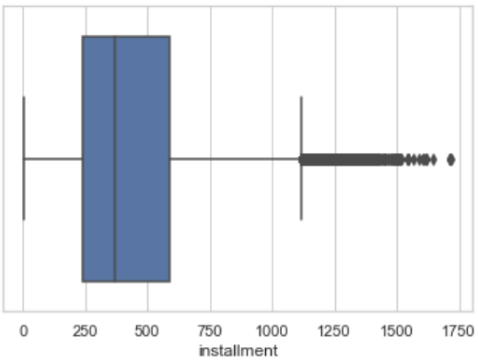
Table 1: Columns and Corresponding Missing Data Percentages (descending order)

2.6 Outliers & Range Analysis

By plotting a box-and-whisker chart[2], and analysing the range of values within a column, we can easily visualise the outliers for specific, numeric variables. We can ascertain for example, by

[2]seaborn.boxplot: seaborn.pydata.org/generated/seaborn.boxplot.html

examining the range of 'instalment' values in Figure 4, that most loan instalments are between 250 and 600, with numerous outliers to the right of the 0.75 percentile. These outliers may be errors and further analysis is required.



count	77159.000000
mean	443.024291
std	279.917146
min	7.610000
25%	239.520000
50%	368.900000
75%	591.285000
max	1717.630000

Figure 4: Outliers in the 'installment' Feature

Following deeper inspection, the larger instalment amounts appear legitimate (simply stemming from larger loans) and will be kept in the model. Similar outlier detection is done for all features; it is determined that none of the features (that are ultimately kept and input into the model) contain any outliers that require removal.

3 Data Cleaning & Pre-Processing

Before the data is input into the model, it must be cleaned and pre-processed to ensure the results produced are valid, and the process is both effective and efficient.

3.1 Feature Removal

There are several reasons features are removed: features may be invalid, useless, be highly correlated with another feature and thus redundant, or contain too many missing values, and these areas (and how each is dealt with) is delineated here.

As discussed in Section 2.2, for our implementation, we need to remove all features that would not be known at application stage. We can identify these features by looking at the data dictionary and understanding what each feature represents. As a result, the features shown in Figure 5 are removed.

```
# based on comprehension of the data, the following features won't be known at time of loan application
# thus will need to be removed to maintain validity
unwanted_columns = [
    'issue_d', 'last_pymnt_d', 'next_pymnt_d', 'recoveries', 'collection_recovery_fee',
    'total_pymnt', 'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int', 'total_rec_late_fee',
    'out_prncp', 'out_prncp_inv', 'hardship_flag', 'last_fico_range_high', 'last_fico_range_low',
    'id' # randomly assigned number -> useless
]
```

Figure 5: Unwanted Features (to be removed)

The 'id' feature is just a randomly assigned number and offers no value so is also removed.

3.1.1 Low Variance Features

For each column, we can check the number of unique values. If this number is one, it means that all rows contain the same value and thus, the column contains no useful information for

our model. The value of 'pymnt_plan' for every entry is 'n' - meaning there is no variation, and therefore no useful information, and accordingly, this column is dropped.

3.1.2 Highly Correlated Features

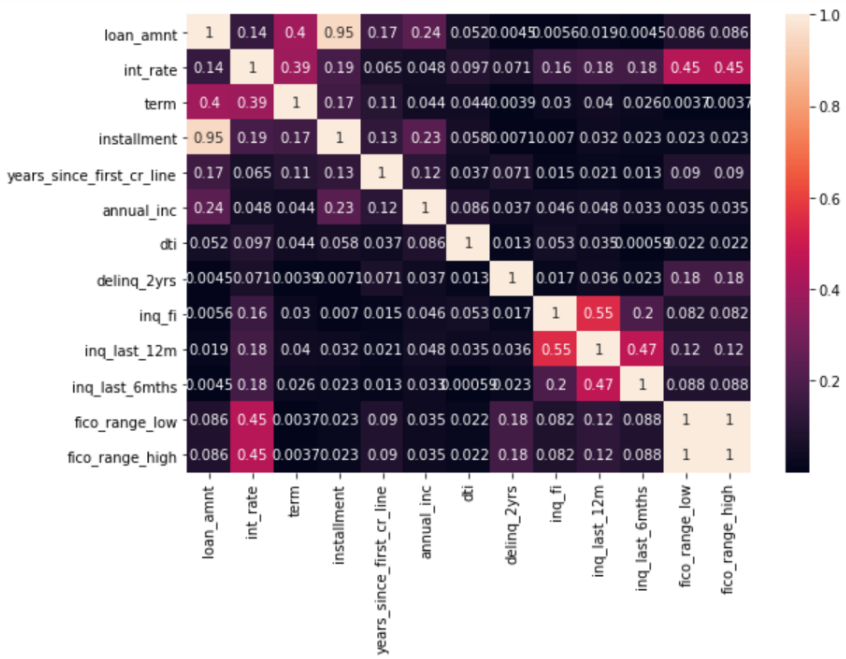


Figure 6: Correlations Heatmap for a Small Subset of Columns

If two (or more) features share very high correlation with one another, minimal information is gained from the second of these columns. We can check which columns are highly correlated and remove one of them to aid in dimensionality reduction (for improved computation) without affecting the quality of the model.

We can see for example, that 'fico_range_high' and 'fico_range_low' are exactly correlated (1.0) and there is no reason to keep both features. In this instance, the two features are combined using a lambda function to produce a 'fico_range_avg' column. Columns with high, but not exact, correlation (e.g. 'loan_amnt' and 'installment') may provide useful information in the rows where the correlation is inexact, and these columns/instances will be explored further in Section 3.3.

3.1.3 Features With a High Percentage of Missing Values

As discussed in Section 2.5, columns with more than 50% of missing values (See Table 1) are removed, resulting in 19 columns being dropped from the dataset. The remaining missing values are imputed (Section 4.2 but this is done once the data has been split into training and test sets, to prevent data leakage.

3.2 Class Groupings & Row Removal

All current loans are ongoing and thus, are not useful in determining whether a loan will ultimately be 'good' or 'bad'. Similarly, loans that are in the grace period, or one of the late categories, are also ongoing and will be removed from the data set (as we cannot yet determine whether they will end up being 'good' or 'bad').

To clarify, by 'good', we mean a loan that will be paid off in full and the lender will make the agreed upon return. A 'bad' loan is one that will not be paid in full (principal plus interest) and the lender will make less than has been agreed upon. We are then left with three categories (Figure 7) which will be encoded to two imminently: 'Charged Off' and 'Default' (bad loans), and 'Fully Paid' (good loans).

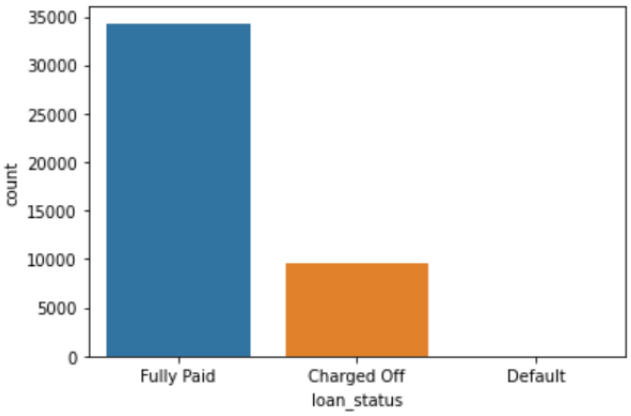


Figure 7: Remaining 'loan_status' Categories

Additionally, a subset of thirteen features are found to have the exact same number (2495) of missing values, all belonging to the same 2495 rows of data. These rows are removed as too much data is missing from each for imputation. Duplicate rows are also checked for (and would potentially be removed) but none are found.

3.3 Feature Engineering

By understanding the domain, it is possible to design new features (from the existing features) which can add value by encoding information which is better able to model the problem space. For example, the initial 'emp_title' feature is currently unusable as it contains far too many unique values to reliably encode. Instead, a new binary feature is created from this column 'job_title_provided' which details whether or not this the 'emp_title' feature is left blank, with the assumption that a significant percentage of missing values are due to unemployment, which would affect the ability of the borrower to repay the loan. Similarly, several other features are engineered (using lambda functions) which, using domain knowledge, attempt to add useful information to the model. These features, depicted below in Figure 8, will be tested for utility in 4.5.1 and following the creation of these new features, several original features are now redundant and are therefore removed.

```
# attempt to quantify how affordable the loan is
loans_data['amnt_div_by_ann_inc'] = loans_data.apply(lambda x: x.annual_inc / x.loan_amnt, axis=1)

# get one fico score (as opposed to 2 correlated columns)
loans_data['fico_range_avg'] = loans_data.apply(lambda x: (x.fico_range_high + x.fico_range_low)/2, axis=1)

# see if a job title has been given
loans_data['job_title_provided'] = loans_data['emp_title'].apply(lambda x: 1 if not pd.isnull(x) else 0)

# again, see how affordable the loan is over the longer term
loans_data['amnt_div_by_term_times_int_rate'] = loans_data.apply(lambda x: x.loan_amnt / x.term * x.int_rate, axis=1)

# get a pseudo disposable income feaute
loans_data['disposable_income'] = loans_data.apply(lambda x: x.avg_cur_bal - x.installment, axis=1)

# drop some more columns that are no longer needed
loans_data.drop(columns=['fico_range_low', 'fico_range_high', 'emp_title', 'installment'], inplace=True)
```

Figure 8: New Features

Another operation undertaken is to remove very rare instances of categorical variables and accumulate these in an 'other' column, when it makes sense to do so. For the 'purpose' column for example, there are 4 wedding and 46 renewable_energy 'purpose' instances. To reduce the number of columns needed after this column is encoded, and to prevent a scenario where the encoded 'purpose_wedding' column has no variance, we can recategorize both into the 'other' category to reduce ultimate column number by two. Furthermore, certain columns need to be transformed from object/string datatypes. Here, the 'int_rate' and 'revol_util' columns are currently objects and need to be stripped of non-numeric characters and converted to floats so that they can be understood correctly by the model. Similarly, we alter the 'earliest_cr_line' feature from a datetime to an integer, by creating a 'years_since_first_cr_line' column, which hopefully provides value to the model by establishing a pseudo credit history feature.

3.4 Encoding

The target variable 'loan_status' is currently of object datatype, with the three remaining categories 'Fully Paid', 'Default', and 'Charged Off'. To encode the target, the good loans ('Fully Paid') are set to '0' and the bad loans ('Default' and 'Charged Off') are both set to '1', reducing the number of categories from three to two, and effectively transforming the problem into a binary classification.

The remaining categorical features must also be encoded into numeric data before they can be input into the model. It is important to utilise the correct form of encoding so that data is not lost and can be interpreted by the model correctly. There are two features remaining ('application_type', and 'initial_list_status') which only have two categories. These can be encoded in a binary manner, converting one category to '0' and the other to '1', using a LabelEncoder[3]. For the remaining columns (with more than two categories and where no order exists between these categories), one-hot encoding is used, creating a series of binary columns for each category within the initial feature (which is then removed). The 'grade' feature is also retained and encoded, and because a clear order is present between the categories, ordinal encoding is used to retain this ordering, where 'A'=0, 'B'=1, 'C'=2 and so on.

4 Supervised Model Training & Evaluation

Now that the data has been cleaned and pre-processed, it can be split into training and test data. Numerous models were trained and evaluated but the most accurate, logistic regression, is discussed in greater detail here and evaluated.

4.1 Train/Test Split

First, the dataset requires splitting into separate training and test sets, so that the model can be trained and the evaluated on unseen data[4]. Accordingly, 80% is used for training and 20% for testing, with a stratified split implemented, retaining the initial class imbalance to better mirror real-life application when the model is evaluated (though the training set will be balanced in Section 4.4 to allow the model to adequately train the minority class).

4.2 Imputation

To prevent data leakage between the train and test sets, it is imperative to impute missing values separately. For the remaining columns with missing values, it is ascertained (following

[3]LabelEncoder: scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
[4]train_test_split: scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

both visual analysis and feature scoring) that median imputation is most appropriate, as this minimises the impact of outliers on the imputed value, and SimpleImputer[5] is used with the 'strategy' parameter set to 'median'. The median for each of the columns with missing values is derived from the training set and this value is used to impute missing values within the train *and* test set, ensuring that no data leakage occurs.

4.3 Scaling

Scaling is necessary for certain machine learning algorithms and standardisation in particular is used to standardise the values of features so, for example, a feature with smaller values ('grade') is not dominated by larger value features ('annual_inc').

Similar to the imputation process, the StandardScaler[6] is fit on the training data, and then used to transform both the training and test sets, once again preventing data leakage.

4.4 Resampling

To address the underlying class imbalance as discussed in Section 2.4, several resampling techniques were tested, though under-sampling proved to be the most robust (in terms of maintaining a high score for all models), and attaining the highest score for an individual model (Logistic Regression). Under-sampling can be used here[7] (removal of majority class instances, to match 1:1 with the minority class) as plenty of data exists. For problems with less available data, synthetic minority data points would need to be produced using SMOTE (Chawla et al., 2002) or a similar oversampling technique.

4.5 Model Training

Though numerous models were trained and evaluated (Random Forest Classifier[8], XGBoost Classifier[9], Linear SVC[10]), the top scoring model was the Logistic Regression Classifier. First, feature selection is used to reduce the number of columns and then hyper-parameter tuning (using cross-validation) is done to ensure the best parameters are used and the model does not overfit to the training data. Finally, a logistic regression classifier is trained, with the optimal parameters and feature numbers, and results are evaluated using the unseen test data.

4.5.1 Feature Selection

To reduce computational complexity and potentially improve the accuracy of the model, less relevant features should be removed. To determine the optimal number of features (in terms of 'auc_roc' score, which was input as the 'scoring' parameter), RFECV[11] (Recursive Feature Elimination Cross Validation) is used. It is determined that the optimal number of features, for use with the logistic regression classifier, is 57, and thus the X_train and X_test sets are transformed to reflect this reduction in columns, with just the 57 best features remaining.

[5]SimpleImputer: scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html
[6]StandardScaler: scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
[7]RandomUnderSampler: imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.ht
[8]RandomForestClassifier: scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[9]XGBoost: xgboost.readthedocs.io/en/latest/
[10]LinearSVC: scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[11]RFECV: scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html

4.5.2 Cross-Validation & Hyperparameter Tuning

Several options for several parameters are input, in conjunction with GridSearchCV[12], to evaluate different parameter combinations. RepeatedStratifiedKFold cross validation[13], is also used in this process to ensure that the the model does not overfit to the training data. The optimal parameters, in terms of cross-validation score, are found to be 'solver'='liblinear', 'penalty'='l2' and 'C'=0.1. The 'max_iter' parameter is set to 10000 to ensure the model converges and 'random_state' is set to 0 to ensure the results are repeatable. The model is fit on the training data with the optimal number of features (57) and parameters, producing the optimised (and cross-validated) logistic regression model to evaluate.

4.6 Model Evaluation & Interpretation

Several methods were used to evaluate the model. First, a confusion matrix (Figure 9) is used to visualise the classification results of individual loan instances, and whether these loan instances are classified correctly by the model, by comparing to the actual loan status on these loans. In addition, balanced accuracy is used as a metric (as opposed to accuracy score) due to the class imbalance within the test data. The optimised logistic regression classifier attains a balanced accuracy of 0.661, which seems underwhelming but by examining the confusion matrix (Figure 9), we can see it can categorise both classes correctly in the majority of instances. A closer look at the classification report in Table 2 shows particularly poor precision when classifying the 'bad' loans. If the models were applied in industry, it could be said to favour a safety first approach, preferring to classify a 'loan' as potentially bad, and recommending against borrowing in these instances, at the cost of potentially refusing numerous loans that would turn out to be paid in full. Additional research into the specific values of loans that are being refused or approved would aid in determining the viability of this strategy, in terms of long-term profitability.

Class	Precision	Recall	F1 Score
'Good' Loans	0.86	0.68	0.76
'Bad' Loans	0.37	0.65	0.47

Table 2: Classification Report

Finally, ROC-AUC score (Figure 10) is used to denote how well the classifier can distinguish

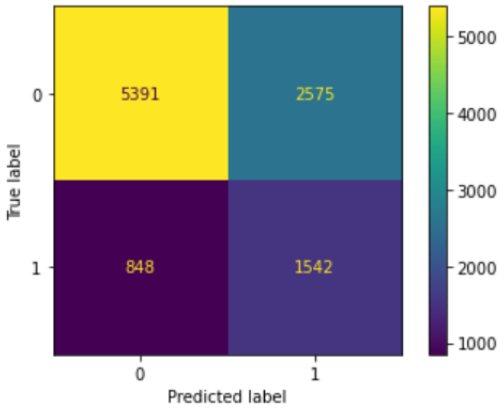


Figure 9: Logistic Regression Confusion Matrix

[12]GridSearchCV: scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[13]RepeatedStratifiedKFold: scikit-learn.org/stable/modules/generated/sklearn.model_selection.RepeatedStratifiedKFold.html

between classes, with a baseline of 0.5 used (in the case of binary classification) which would be the score achieved by random classification. A score of 0.724 shows that the model performs significantly better than the baseline for both classes.

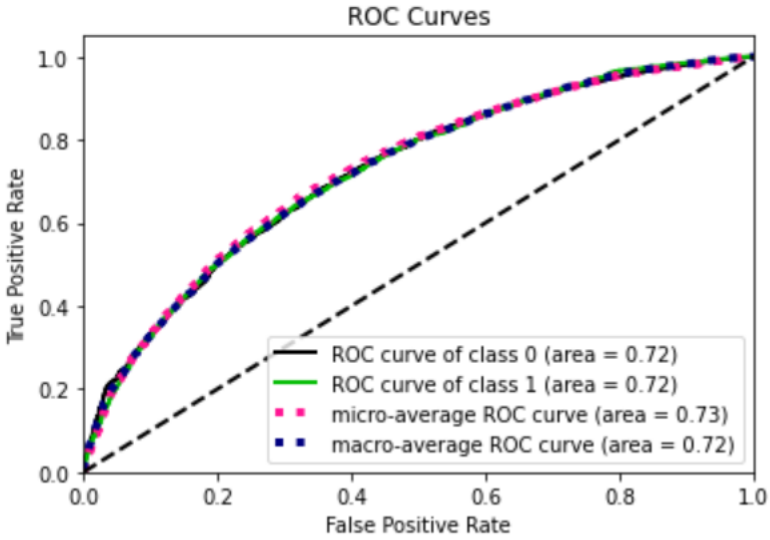


Figure 10: Logistic Regression ROC Curve

5 Unsupervised Clustering

The existing training data (from the supervised learning portion of this report) is used again here as the necessary cleaning pre-processing has already been completed and a sufficient amount of data is available. However, the 'X_train' data used is taken prior to resampling (and feature selection) so that the classes are imbalanced again to better replicate real life circumstances.

Dimensioality Reduction is used twice here, once before the model is trained to improve accuracy and again to visualise the models classifications. The algorithm used is K-means clustering[14], chosen due to its adaptability and ease of implementation.

5.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA)[15] is used first to reduce the dimensions of the data from 87 to 20, to improve clustering accuracy. It is used a second time shortly, to aid in visualising the clustering performed by the model.

5.2 Finding Optimal K using the Elbow Method

Following initial dimensionality reduction, to find the optimal number of clusters for the data and model, the Elbow Method is used (Kodinariya and Makwana, 2013) to test several k values, with the elbow of the curve determined to be the optimal k value. When run on the training data, the optimal value of k is found to be 3 (Figure 11).

[14]KMeans: scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
[15]PCA: scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

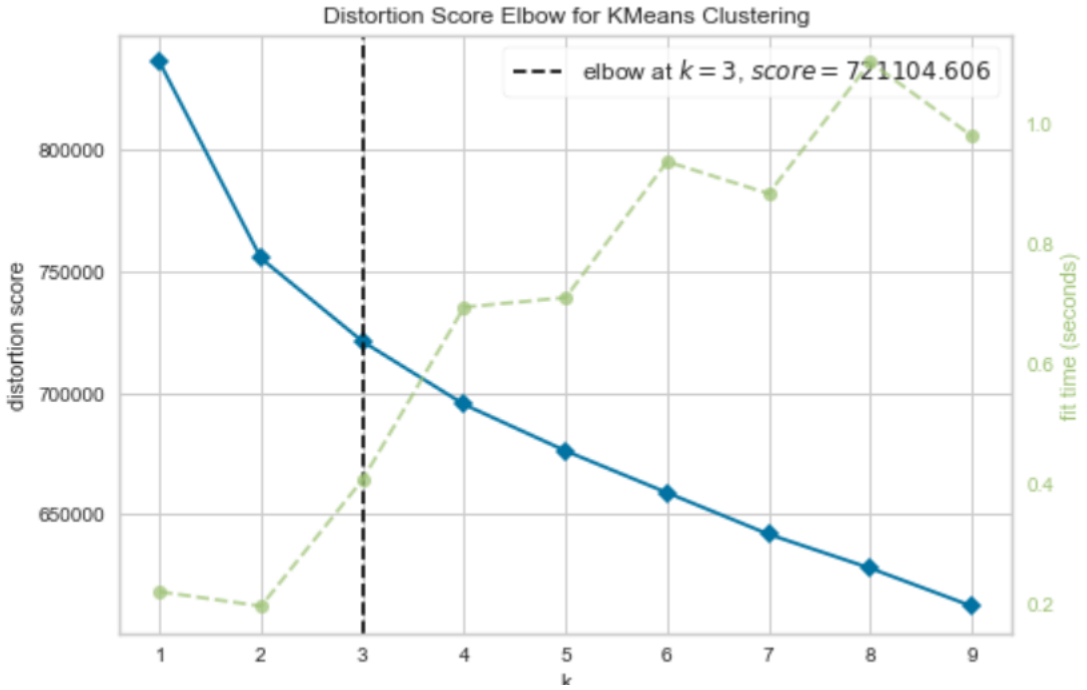


Figure 11: Visualising K-Elbow

5.3 KMeans Clustering

The KMeans model is run again, using the optimal value of k, determined above to be 3, meaning the each instance is clustered into one of three classes. Of course, the actual data contains just two classes, 'good' and 'bad' loans and the suggested third class here should be researched further.

5.4 Clustering Evaluation & Interpretation

Two metrics, in addition to the visual analysis performed above, are used to denote how successful the clustering was: Silhouette Score and Adjusted Rand Score. Silhouette score aims to quantify how distinct the clusters are (and to what extent classes exist within the data). A Silhouette Score of 0.11 is achieved, suggesting fuzzy classes with little distinction between the clusters. Adjusted Rand Score compares the classification produced by the model to the true classifications, to quantify how well the clustering performs in actuality, when compared to a baseline of random classification. An Adjusted Rand Score of 0.002 suggests that the K-means clustering model implemented here is only just able to beat random classification and offers little utility.

5.4.1 Visualisation of Actual & Predicted Clusters

By using PCA once more, and reducing the number of features used to two or three, and using 2D and 3D scatter plots, it is possible to visualise how each loan instance is classified by the clustering algorithm and compare this with the actual classifications.

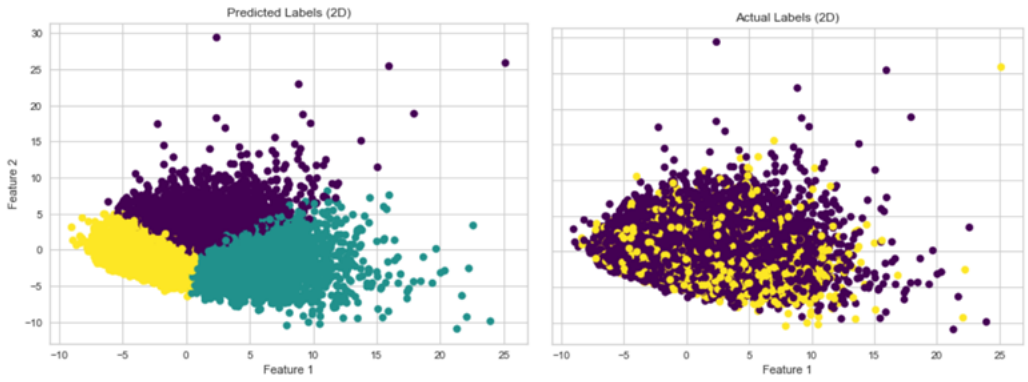


Figure 12: Two-dimensional Clustering Visualisation

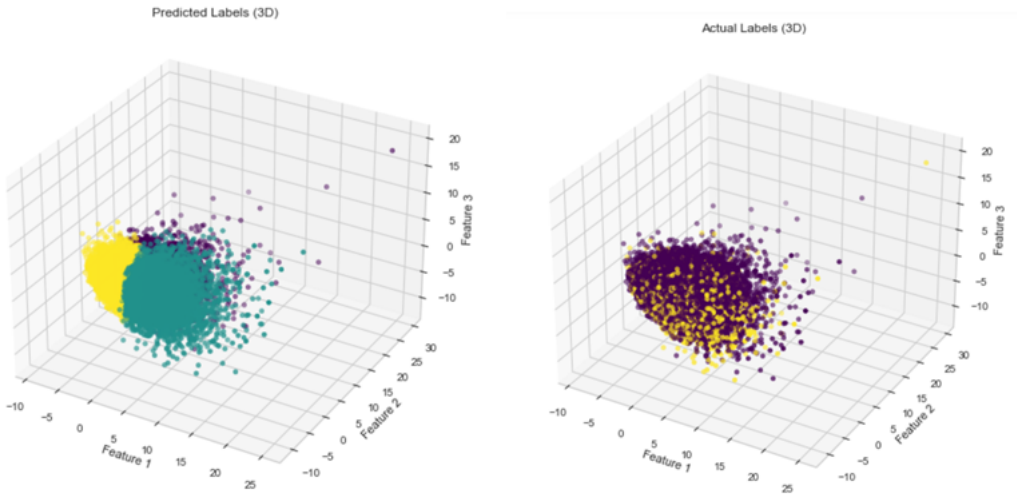


Figure 13: Three-dimensional Clustering Visualisation

By visualising the actual classifications (the graphs on the right of Figures 12 & 13), we can see no distinct clusters exists for the data (in terms of our implementation, which is predicting the future status of loans at the application stage). In both figures, the purple points denote 'good' loans, that were fully paid and the yellow points denote the 'bad' loan catogory. The blue points belong to the third category suggested by the model.

6 Conclusion

In summary, an accuracy score of 0.95 and above is very attainable if the 'future' features are left in the model. However, in terms of real-life application, this offers no utility. Instead, our method examines whether a loan should be approved at the time of application, by only considering features or variables which would be known at that point.

The best supervised model was logistic regression, achieving a balanced accuracy score of 0.661. Whether the results can be used in any meaningful way remains difficult to say. Further research into the monetary value of each loan, and how much it gains or loses the lender, could be undertaken to ascertain, to what degree, the proposed process is effective.

For the unsupervised portion, K-means clustering was used, with the number of clusters set to 3 (as determined by the elbow method), was not effective in clustering the data, resulting in a results just slightly better than random chance.

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.

Kodinariya, T. M. and Makwana, P. R. (2013). Review on determining number of cluster in k-means clustering. *International Journal*, 1(6):90–95.