University of Malta

ICS5110: Applied Machine Learning

Bankruptcy prediction using Logistic Regression and Bagged Decision Trees

Albert Bezzina (279484M), **Daniel Farrugia (384479M)** & **Ivan Salomone (358379M)**

January 2019

Contents

1 Introduction

Companies having liquidity problems face what is know as *financial distress*, that is the company's operating cash flow is insufficient to meet its current liabilities [1]. Financial distress forces companies to take unwanted measures, such as reducing the work force or downsizing operations, in order to alleviate the problem. Failing to emerge from financial distress leads a company to *bankruptcy*, that is a situation where a company is unable to operate any longer due to its liquidity problems.

Bankruptcy affects the social community, business partners, investors, policy makers and the economy as a whole [2]. The high costs of bankruptcy led many researchers to seek means of anticipating it. Such studies date back to the 1930's when economists used financial indicators to predict business failures. Statistical models for bankruptcy prediction were introduced in the 1960's. Later in the 1990's, artificial intelligence and machine learning models took over.

In this assignment we demonstrate the use of *Bagged Decision Trees* and *Logistic Regression* models to predict companies going bankrupt based on their financial indicators. In doing so, we also discuss the steps taken to improve the predictive performance of the two models. Moreover, in Section 2 we give some theoretical background on the techniques that were used in this study. In Section 3 we describe the methods we used to pre-process the data, build and train the predictive models and finally evaluate the results. We also describe the experiments that were conducted on the inferred models. To conclude, in Section 4 we discuss the findings of this study.

1.1 The Dataset

The dataset that is used for this assignment was created by Zikerba et al. [2] in a study aimed at demonstrating bankruptcy prediction through an ensemble of boosted trees. It consists of five years of Polish company financial indicators, together with a class indicating whether the company goes bankrupt in the sixth year. Therefore, for *Year 1* the class indicates whether the company goes bankrupt in $t + 5$ years (1=bankrupt, 0=solvent) whilst for *Year 5* the class refers to $t + 1$ year. The data for each year resides in a separate ARFF[1] file. The attributes, 64 in total, consist of accounting ratios such as *net profit / total assets* and *total liabilities / total assets*. A description of all the attributes can be found in the dataset description page[2].

Year	1	2	3	4	5
# samples	7027	10173	10503	9792	5910
% bankrupt	4	4	5	5	7
% solvent	96	96	95	95	93
% correlated	10	10	6	8	10
% uncorrelated	90	90	94	92	90
% missing	1	2	1	1	1
% outliers	1	1	1	1	1

Table 1: The percentages of bankrupt/solvent companies as an indication of imbalance, correlation between attributes, and the percentage of missing values and outliers, across the various years.

The results of a preliminary study of the data are presented in Table 1. A considerable imbalance in the class can be noted, given that the majority of companies are solvent in the sixth year. This is important to

[1] Attribute-Relation File Format
[2] http://archive.ics.uci.edu/ml/datasets/Polish+companies+bankruptcy+data

note due to its negative impact on the machine learning process. Other aspects that impair the classification ability of the models are missing values and outliers. Although the percentage of missing values and outliers is relatively small, proper handling of these is likely to give better results.

It was also found that some of the attributes are closely correlated. This is not surprising because some of the accounting ratios are closely tied. For example a correlation coefficient of 1 was found between *EBIT*[3] */ total assets*, *(gross profit + interest) / total assets* and *gross profit / total assets*. Table 1 shows the percentage of pairwise attributes that have a *correlation coefficient* $(r) \geq 0.5$. Figure 1 shows a correlation heatmap for the dataset of *Year 1*.

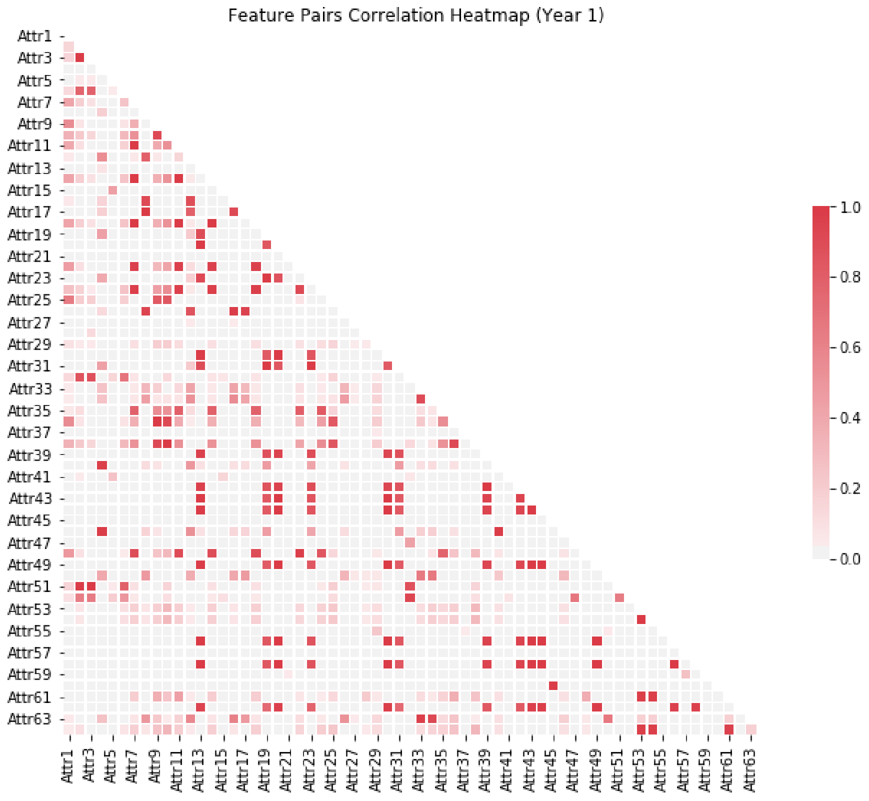


Figure 1: A correlation heatmap for Year 1.

2 Background

2.1 Logistic Regression

Logistic regression is a probabilistic discriminative model [3]. It is a binary classifier that models the probability that an instance belongs to either of two classes (i.e. 1 or 0, True or False, etc.) [4]. For each instance *i* the model will produce a probability that *i* belongs to a class c_0 (i.e. $Pr(Y_i = c_0|x_i)$). Then the model will classify *i* as c_0 if the probability is greater than a particular threshold. In many cases the threshold is set to 0.5, but this varies according to the problem at hand.

This is done by using the techniques from linear regression shown in Equation 1, where x_{1i} to x_{ni} are the values of the various attributes of instance *i*, β_0 to β_n are the corresponding weights associated to each

[3]Earnings Before Interest and Taxes

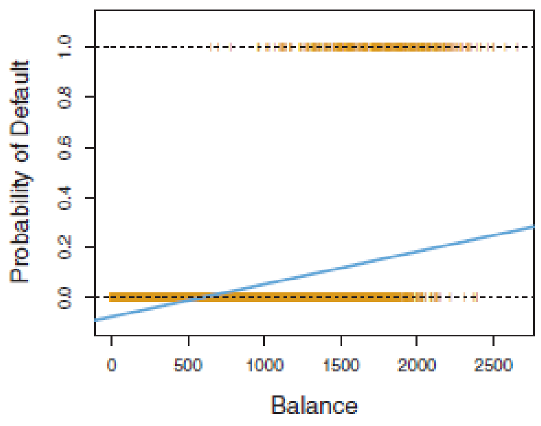
attribute [5].

$$z_{\mathrm{i}} = \beta_0 + \beta_1 x_{1\mathrm{i}} + \beta_2 x_{2\mathrm{i}} + ... + \beta_{\mathrm{n}} x_{\mathrm{ni}} \tag{1}$$

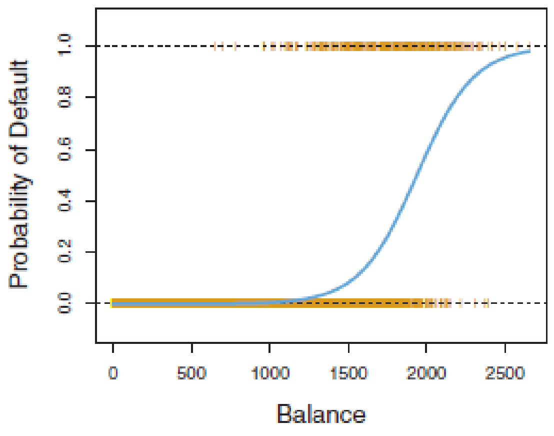
Given that this does not produce a probability between 0 and 1 as required for binary classification, the resulting z_i is plugged into the logistic (or sigmoid) function (Equation 2) to produce the desired result [5]. This is shown in Figure 2 where the plot (a) shows the estimated probability of default using the linear equation (Equation 1) while the plot (b) shows the probability transformed through the logistic function (Equation 2) [4]. Although Figure 2 shows plots for univariate data, logistic regression can also be performed on multivariate datasets.

$$Pr(c_0|x_{1\mathrm{i}}...x_{\mathrm{ni}}) = \frac{1}{1 + e^{-z_{\mathrm{i}}}} \tag{2}$$

The weight coefficients β_0 to β_{n} are estimated upon the training data and are determined by the maximum likelihood estimation [4, 6]. In other words, the vector α of parameters β_0 to β_{n} is chosen so as to maximise $\prod_{i=1}^{n} Pr(y_{\mathrm{i}}|x_{\mathrm{i}}, \alpha)$. This can be achieved using using various methods amongst which gradient descent [6].



(a) A plot using the linear function.



(b) The predicted probabilities transformed using the logistic function.

Figure 2: A plot of the *Probability of Default* against the *Balance* [4].

Although inherently a binary classifier, logistic regression can be extended to classify more than two classes [7]. The approach involves simplifying the multiclass problem into a number of binary problems, whereby for every class, a logistic regression classifier is trained to determine whether an instance is a member of the class or not. The multiclass classification is achieved by evaluating the output of the various classifiers to establish the class. This is known as the one-against-all approach.

Logistic regression is found to be less prone to overfitting than other models [6]. However, it is intolerant to noise and outliers.

2.2 Bagged Decision Trees

2.2.1 Decision Trees

A decision tree is a classifier that takes the form of nodes and branches [8]. Internal nodes represent a test on some feature, while the branches represent the outcomes of the test, leading to another internal node or

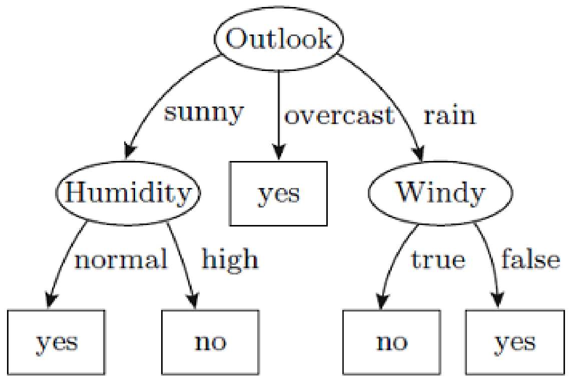


Figure 3: The resulting decision tree after training on the data in Table 2 [5].

Outlook	Temp	Humidity	Windy	Play Golf
rainy	hot	high	false	no
rainy	hot	high	true	no
overcast	hot	high	false	yes
sunny	mild	high	false	yes
sunny	cool	normal	false	yes
sunny	cool	normal	true	no
overcast	cool	normal	true	yes
rainy	mild	high	false	no
rainy	cool	normal	false	yes
sunny	mild	normal	false	yes
rainy	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
sunny	mild	high	true	no

Table 2: A sample dataset adapted from [5].

a leaf node. Each leaf node represents a class. The topmost node is called the root node. Decision trees can deal with both *discrete-valued* and *continuous-valued* attributes. Continuous values are expressed as conditions by introducing *split-points*, for example $salary < 10,000$ and $salary \geq 10,000$.

Decision tree induction algorithms include ID3, C4.5 and CART, which adopt a greedy approach (i.e. without backtracking) to the tree construction [8]. The tree is built recursively top-down, starting from the root node [5, 8]. The algorithms place at the root the attribute that best discriminates the training data. For example, let *D* be the sample dataset shown in Table 2. The algorithm evaluates the four attributes (note that *Play Golf* is the class) and it results that *outlook* best discriminates the data. So *outlook* is placed at the root. Next, the possible outcomes of *outlook* are set as branches. For each branch, the dataset is partitioned by the respective outcome. If all the examples in the partitioned dataset are of the same class then a leaf node is added and set to this class. It can be noted that for $D_{overcast}$ this holds, so a leaf node with class *yes* is added. But not for D_{sunny} so once again the attribute that best discriminates D_{sunny} is set as the node. The algorithm iterates until all leaf nodes are filled. Figure 3 shows the resulting tree.

$$Entropy(S) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{3}$$

$$Gain(S, A) = Entropy(S) - \sum_{v \in values(A)} \frac{|S_v|}{S} Entropy(S) \tag{4}$$

The function used to measure how much an attribute discriminates the data varies according to the algorithm being used [8]. ID3 uses *information gain* while C4.5 uses *gain ratio*, that is a normalised form of the *information gain*. CART on the other hand uses the *Gini index*. The information gain is obtained through Equation 4, where $Gain(S, A)$ is the information gain of an attribute A relative to a collection of examples S, $values(A)$ are all the possible values of A, S_v is the subset of S having $A = v$ and $Entropy(S)$ is obtained through Equation 3, where in turn p_i is the subset of S belonging to class i.

2.2.2 Tree Pruning

Decision tree induction is prone to *overfitting*, especially in the presence of noise or outliers [8]. Tree pruning is a method whereby part of a tree is removed to simplify or generalise the decision tree, thus addressing overfitting. Most commonly, a *post-pruning* approach is used that prunes sub-trees from the induced tree. A sub-tree at a given node is removed and replaced by a leaf node, which is labelled with the class that featured most in the pruned sub-tree. Similarly, in *pre-pruning* the tree construction process is stopped prematurely resulting in an unfinished decision tree.

2.2.3 Bagging

In bagging, *n* classifiers are trained on *n* random samples of the training data, taken with replacement [9]. This results in *n* different models. Classification of unseen instances is then done through a vote of all *n* classifiers. Ties are decided arbitrarily. Experiments in [9] show that *bagged* decision trees outperform ordinary decision trees.

2.3 Rescaling and Normalisation

Typically, features in a dataset have different scales and ranges, possibly incorporating very large numbers. These discrepancies across features tend to hinder the performance of a learning algorithm [8]. Larger numbers give the feature a greater weight, thus giving it prominence over other possibly more important features that are of a smaller dimension. This happens particularly in algorithms that use distance measurements such as *k-nearest neighbour*, *neural networks* and *clustering*. To overcome this, features are normalised or rescaled. This involves transforming the values of the various features to a smaller range, such as between *0* and *1* [8, 10, 11]. In literature, the terms normalisation and rescaling are used interchangeably.

2.3.1 Min-max normalisation

Min-max normalisation transforms the features to a range between *0* and *1* [10]. The normalised value v' is obtained through Equation 5 where v is the original unnormalised value, and A_{min} and A_{max} are respectively the minimum and maximum values of feature A.

$$v' = \frac{v - A_{\text{min}}}{A_{\text{max}} - A_{\text{min}}} \tag{5}$$

2.3.2 Z-score normalisation

Z-score normalisation (sometimes referred to as *standardisation*) transforms the data such that they have a mean of *0* and a standard deviation of *1* [10, 11]. This is obtained through Equation 6 where value v is transformed into a normalised value v'. $\bar{A}$ is the mean of feature A and σ_{A} is the standard deviation of A.

$$v' = \frac{v - \bar{A}}{\sigma_{\text{A}}} \tag{6}$$

This technique is particularly useful when the minimum and maximum of a feature are not known or when the data contain outliers [11]. In this technique a normal distribution is assumed [12].

2.3.3 Decimal scaling

Decimal scaling divides all values of a feature by a common denominator so that all new normalised values v' fall in the range $0 < v' < 1$ [8, 11]. This is achieved through Equation 7 where v is the unnormalised value and j is the smallest integer such that $max(v') < 1$.

$$v' = \frac{v}{10^j} \tag{7}$$

2.3.4 Quantile normalisation

In quantile normalisation, the distribution of each feature is harmonised [13]. To normalise a dataset X made up of n instances having p attributes, the values making up each instance are first sorted to make up X_{sort}. It should be noted that the dimensions of X_{sort} are not the same attributes of X since these are now mixed-up through sorting. X'_{sort} is then created by changing the value of each element within every dimension of X_{sort} to the mean of its respective dimension. The normalised dataset X' can then be acquired by reordering the instances of X'_{sort} to the original order of X. This normalisation method is very robust to outliers since the extreme values are replaced by the means.

2.3.5 Normalisation parameters

When normalising data, it is imperative that the normalisation parameters used to transform the data (eg. the *mean* and *standard deviation* in *z-score normalisation* or *j* in *decimal scaling*) are saved for later use [8]. When new instances need to be classified these have to be normalised using these same parameters.

2.4 Cross-validation

The error rate of a learning function should be estimated upon unseen observations since the performance of the function varies considerably upon seen and unseen instances [4]. Therefore, a subset of observations is held-out from the training set upon which the model is fit and then used to estimate the error rate. This subset is referred to as the test (or validation) set. Cross-validation (CV) extends upon this method by averaging the error rate of several hold-out runs using different data splits [14].
CV techniques can be categorised as *exhaustive data splitting* approaches and *partial data splitting approaches* [14].

2.4.1 Exhaustive data splitting CV

In exhaustive data splitting, the size n_t of the training set is decided in advance, and all possible training sets of size n_t are considered, i.e. all possible combinations of splitting the training data into a training set of size n_t are eventually used for training and evaluation [14]. *Leave-one-out* is one such method where n_t is taken as n-1. All possible training sets are used, each time holding out one observation that is then used for testing. *Leave-p-out* adopts the same approach but in this case n_t is taken as n-p.

2.4.2 Partial data splitting CV

In *k-fold* (or *v-fold*) CV, the error rate is estimated over *k* iterations of training [14]. The training data is randomly split into *k* segments, each of which is taken as a test set for one of the iterations. In some cases, the folds are stratified to contain approximately the same number of instances from each class thus keeping the dataset balanced [15]. Experiments carried out by [15] suggest the use of ten or more folds $(k \geq 10)$ since the bias is reduced considerably.

Balanced incomplete CV is another technique where test sets are systematically chosen in such a way to maintain a balance between the various data points [16]. Specifically, the test sets are chosen in such a way that every data point appears in the same number of test sets, and every pair of instances appear in the same number of test sets.

Monte-Carlo CV and *repeated learning-testing CV* are another two partial data splitting approaches that are very similar [14]. Both approaches consider several test sets of size n_v that are chosen at random. The difference between the two approaches is that in Monte-Carlo CV the test sets are generated with replacement while in repeated learning-testing CV the test sets are generated without replacement.

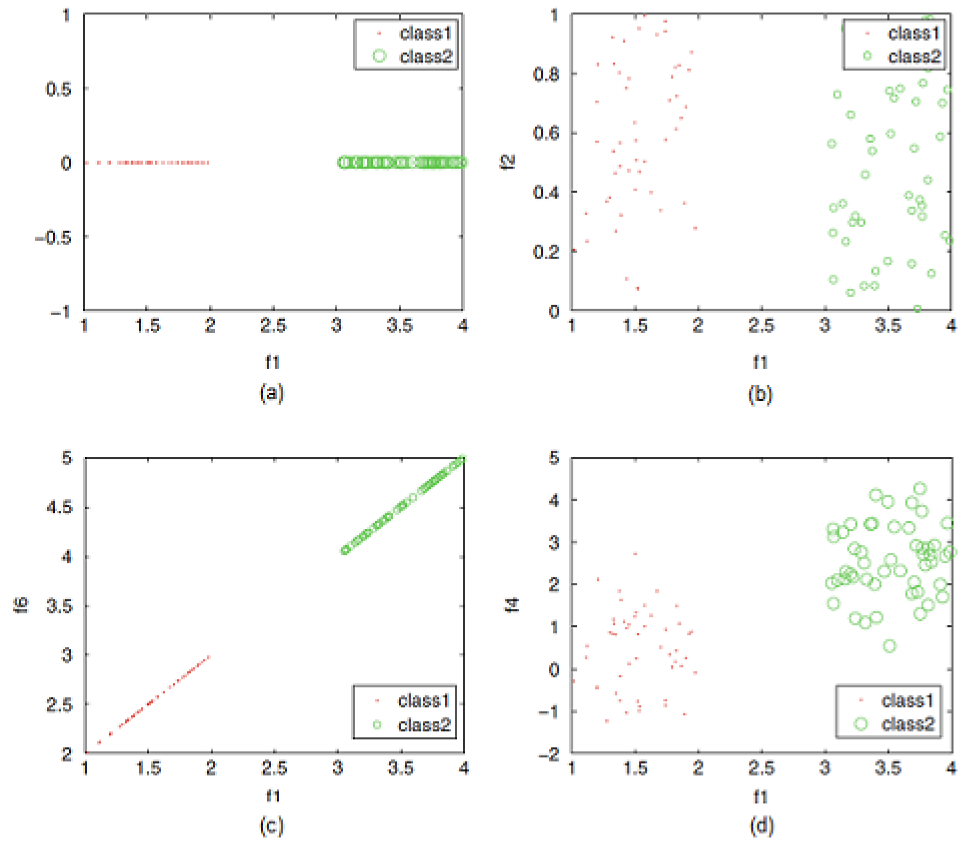


Figure 4: Example plots showing (a) a relevant feature, (b) an irrelevant feature, (c) a redundant feature and (d) a noisy feature [5].

2.5 Feature Selection and Dimensionality Reduction Methods

Having many features introduces a number of challenges to machine learning. Not all features may be relevant to the problem and these confuse the learner resulting in a decrease in the predicting capability [5]. Moreover, having many features requires more training time and more computation capacity [17]. There is also an increased risk of overfitting [18]. Therefore in the pre-processing phase we aim to reduce the number of dimensions whilst keeping the information that is important to the learning process.

2.5.1 Feature selection

To reduce the number of dimensions that are used for learning, feature selection retains a subset of the dimensions that are related to the class label whilst the remaining dimensions are discarded [intro]. A dimension is discarded if it is found to be irrelevant, redundant or noisy [5]. For example, considering feature *f1* as shown in Figure 4a, it can be seen that *f1* is a relevant feature because it can distinguish between the classes. Feature *f2* (Figure 4b) on the other hand is unable to distinguish between the two classes and is therefore irrelevant for the classification and hence discarded. As seen in Figure 4c, feature *f6* is also a relevant feature because it perfectly distinguishes between the two classes. However, f6 does not add any new information that *f1* has not already provided, so it is said to be a redundant feature. Feature *f4* (Figure 4d) is partly able to distinguish between the two classes. The overlapping instances are either caused by errors in the data (noise) or due to the nature of the attribute. This type of feature is said to be noisy and is also discarded. There are three models used for feature selection: filter, wrapper and embedded.

Filter models - Filter models rank the features by their relevance to the problem or by the degree of new information that is added [18]. *Mutual information*, *Pearson's correlation* and *maximum-relevance-minimal-redundancy* are commonly used as score functions upon which to rank the features [18]. Once the features are ranked, the top-ranking features are chosen whilst the others are discarded [5].

Wrapper models - The filter models discussed above consider features individually assuming them independent, which may or may not be true. Low ranked features may still contribute towards a prediction when considered together with other features [18]. Wrapper models address this limitation by evaluating subsets of features rather than individual features. Figure 5 shows a conceptual view of a wrapper model. The *subset generation* component takes on the original features and uses a search strategy to generate subsets of features for evaluation. The *subset evaluation* component utilises the learning algorithm upon the subset to measure and evaluate its performance. These steps are repeated until a *stopping criterion* is met. Typically, the stopping criterion is that refinements to the subset do not yield any performance improvements. Another improvement on the Filter model is that since the learning algorithm is involved in the selection of the subset, any bias introduced by the chosen algorithm is removed [5]. The drawback of this model is that, while Filter models are very efficient, wrapper models are very computationally expensive.

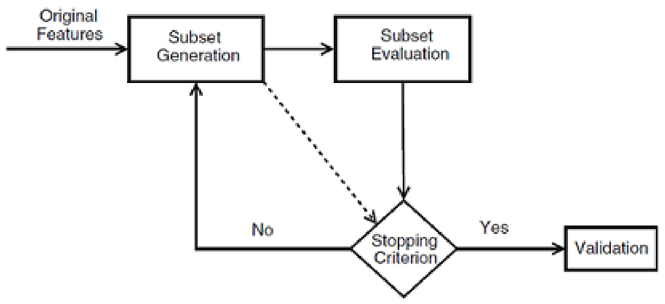


Figure 5: A conceptual view of a Wrapper Model [18].

Embedded models - Some learning algorithms have in-built mechanisms to deal with feature selection [17, 5]. This makes the selection of the features more efficient than the wrapper method and is specifically designed for the algorithm of choice. Another improvement on the other methods is that whilst the filter and wrapper methods select the features as part of the pre-processing of the data, the embedded models cater for feature selection as part of the learning process. One such example is the use of *l1-norm* regularisation in *Support Vector Machines* (SVM) [17, 18]. Here the coefficients of less important features are reduced to zeros thus eliminating them. *L2-norm* regularisation can also be used for feature selection but was found to be less effective than *l1-norm* regularisation [18]. *Decision trees* also have a feature selection strategy embedded within the algorithm [11].

2.5.2 Dimensionality reduction

In dimensionality reduction (or *feature extraction*) the number of features is reduced by defining new dimensions that are smaller in number but can represent the original dimensions through a function [11, 5]. In other words, a high-dimensional feature space is projected into a feature space having less dimensions. The challenge of dimensionality reduction is finding a feature space that preserves the information provided by the original dimensions [5].

Principal component analysis - Principal component analysis (PCA) searches for a set of linear transformations of the original dimensions that preserve the most variance and result in less dimensions [11]. The result is a number of n-dimensional orthogonal vectors. The dimension that contributes to the most variance of the original data is called the *first principal component*. Generally not all principal components are kept but only those that are accountable for approximately 95% of the variance.

Factor analysis - Factor analysis follows the same concept of PCA but instead of searching for transformations of the original dimensions, it looks for hidden factors among the dimensions [11].

Multidimensional scaling - Multidimensional scaling looks at the distance between data points in a way to preserve the relationship between the data points [11]. After calculating the distance matrix, the dimensions of the data are no longer important since the structure is rebuilt through the distance matrix.

Local Linear Embedding - Local Linear Embedding builds on the assumption that for a non-linear manifold, the surface is linear at infinitesimal distance [11]. Dimensionality reduction is done by first finding the k-NN of each data point, then calculating the linear weights of the distances between the data points and their neighbours so that it can be mapped on a lower dimensional space.

2.6 Quantitative Measurements for Model Evaluation

2.6.1 Sensitivity and Specificity

Sensitivity (or *true positive rate*) is the fraction of positive examples that are correctly labelled by a classification model [5]. Specificity (or *true negative rate*) on the other hand is the fraction of negative examples that are correctly labelled by a classification model [5]. These measures are good candidates for evaluating *class-imbalanced* datasets [8]. Sensitivity and specificity are calculated by Equation 8 and Equation 9 respectively.

$$TPR = \frac{|TP|}{|TP \cup FN|} \tag{8}$$

$$TNR = \frac{|TN|}{|TN \cup FP|} \tag{9}$$

2.6.2 Area under curve

The area under the *receiver operating characteristics* (ROC) graph, more commonly referred to as the *area under curve* (AUC), gives a measure of the overall performance of a classifier [4]. The ROC graph is a plot of the true positive rate against the false positive rate. The AUC gives a single measure that allows for the interpretation of the ROC graph [19].

The AUC statistic of a classifier is a number between 0 and 1, the closer to one the better the performance of the classifier [19, 4]. Any score less than 0.5 means the results of the classifier are worse than random guessing.

An important characteristic of the AUC statistic is that it is equivalent to the probability that the classifier ranks a randomly chosen positive example higher than a randomly chosen negative example [19]. This is equivalent to the *Wilcoxon-Mann-Whitney sum of ranks* test [5], that indicates whether two samples are drawn from a common distribution.

3 Experiments

3.1 Pre-processing

Several methods were used to pre-process the data with the aim of finding which methods give the best results. The methods were tested through a series of experiments that are discussed in Section 3.2 of this document.

Due to the considerable amount of missing values in the dataset, these were treated using four imputation methods. For ease of reference, throughout this document these shall be referred to as *mean*, *median*, *class-partitioned mean* and *class-partitioned median* imputations. The mean and median imputations impute missing values within an attribute to the mean and median respectively of that attribute. The class-partitioned mean method imputes missing values to the mean of the values within the attribute having the corresponding class. The class-partitioned median imputation does likewise but imputes the median of values for the corresponding class. One of the attributes (Attr37) was dropped because it contained a lot of missing values ($\approx$79%).

Class imbalance was tackled through the use of class weights within the models, SMOTE[4] and random under-sampling. In this case, a random search of the parameter space is used to speed up the process. Moreover, the data was normalised using min-max, l2, z-score and quantile normalisation.

The number of features were reduced through PCA, k-best features and l2 regularisation (in logistic regression). Compared to other datasets, 64 attributes are not a lot but some performance improvement may be obtained by using fewer features. Figure 6 shows the 64 dimensions as reduced through PCA.

3.2 Experiments

A number of experiments were carried out using the implemented models in order to determine the effects of the various techniques on the model performance. To do so, a baseline for both logistic regression and bagged decision trees were established, and the AUC, sensitivity and specificity were measured using 10-fold cross-validation. The AUC and 10-fold cross-validation were selected based on the arguments brought up by a similar study [2] on the evaluation of classification based on imbalanced data. We also add sensitivity and specificity as a measure of the correctness of the classification.

Throughout each experiment only one technique is varied from the baseline so that the effect of the change can be quantified. Once again the performance is measured using 10-fold cross-validation.

As a baseline model for logistic regression, a pipeline incorporating class-partitioned mean imputation and quantile normalisation is taken. Class imbalance is addressed through the use of class weights in the model, that is by giving higher weights to the minority class (bankrupt).

The baseline model for bagged decision trees incorporates class-partitioned mean imputation and caters for class imbalance through class weights.

[4]Synthetic minority over-sampling technique

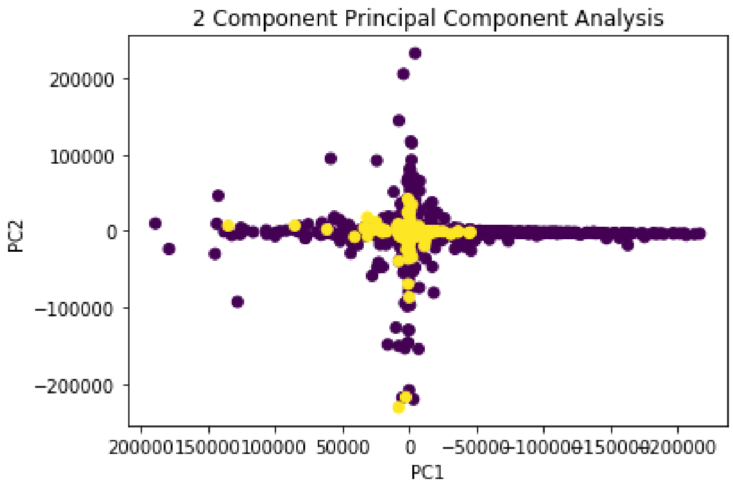


Figure 6: Dimensionality reduction of the data for Year 1 through PCA.

	Logistic Regression		Bagged Decision Trees		
Year	**Learning Rate**	**Minimum Loss**	**Max Features (%)**	**Max Samples (%)**	**# Models**
1	0.010	0.010	0.9	0.9	11
2	0.001	0.001	0.9	0.7	11
3	0.010	0.010	1.0	0.8	10
4	0.010	0.010	0.9	0.9	11
5	0.010	0.010	0.9	0.9	11

Table 3: The hyper parameters that were used throughout the experiments.

Prior to running the experiments, the optimal hyper parameters for each year of the data for both logistic regression and bagged decision trees where searched. Then these were kept constant through all runs of the models. The hyper parameters that were used throughout the experiments are presented in Table 3.

3.2.1 Experiment 1 - Measuring the performance when handling class imbalance

Setup - Two imbalance handling techniques are introduced in separate runs of pre-processing-training-testing pipelines. In the first pipeline for each classifier, the SMOTE sampling method is used while in the second pipeline random under-sampling is used. The cross-validated results are then compared against the respective baseline model of the classifier.

Expected result - Given the class imbalance in the dataset, performance improvement is expected both over- and under-sampling techniques.

Actual result - As seen in Table 4, both SMOTE and random under-sampling improve the sensitivity but decrease the specificity. This results in a slightly better AUC for SMOTE, but a decreased AUC for random under-sampling for logistic regression. The AUC for bagged decision trees improves marginally with both sampling techniques. Overall, the results show little improvement on the class weights methods embedded into the baseline models.

	Logistic Regression			**Bagged Decision Trees**		
	Baseline	**SMOTE**	**Random Under-sampling**	**Baseline**	**SMOTE**	**Random Under-sampling**
AUC	0.875	0.879	0.867	0.909	0.918	0.915
sensitivity	0.725	0.775	0.758	0.581	0.612	0.801
specificity	0.869	0.845	0.826	0.997	0.991	0.858

Table 4: The results for Experiment 1

3.2.2 Experiment 2 - Measuring the effects of different normalisation techniques on logistic regression

Setup - Three different pipelines are run, each with a different normalisation technique. The quantile normalization in the baseline model is replaced with *min-max* normalisation, *l2* normalisation and *z-score* normalisation. The cross-validated results are then compared against the baseline, which uses quantile normalisation. This experiment was only run on logistic regression since in the case of bagged decision trees normalisation does not make much sense.

Expected result - The baseline model is expected to outperform the other models because the quantile normalisation technique is more robust to outliers, which characterise the dataset being used.

Actual result - As it can be seen in Table 5, the normalisation methods under test do not improve on the baseline. Contrarily, the performance is considerably degraded. Figure 7 shows the the resulting confusion matrices of the four methods. The considerable decrease in true positives for min-max normalisation (Figure 7b) can be observed, as well as the decrease in true negatives and increase in false positives for l2 normalisation (Figure 7c). Quantile normalisation (Figure 7a) outperforms the other methods possibly due to its resistance to outliers.

	Baseline	**min-max**	**l2**	**z-score**
AUC	0.875	0.513	0.697	0.751
sensitivity	0.725	0.672	0.545	0.661
specificity	0.869	0.337	0.738	0.75

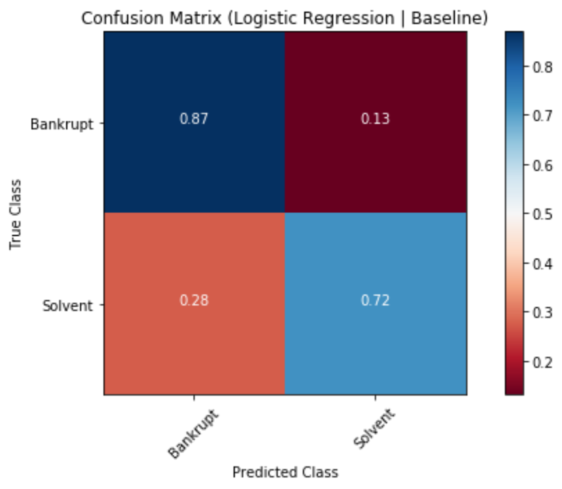
Table 5: The results for Experiment 2

3.2.3 Experiment 3 - Measuring the effects of different feature selection/dimensionality reduction techniques

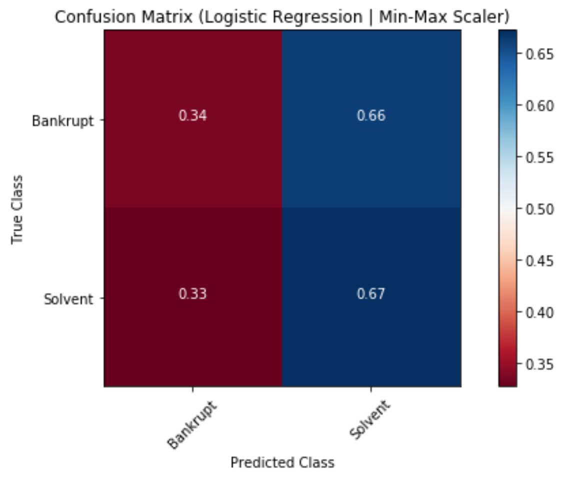
Setup - In this experiment three pipelines are run for logistic regression and two for bagged decision trees, each adding a feature selection or dimensionality reduction method to the baseline model. The methods under test for logistic regression are PCA, L2 regularisation and k-best features. For bagged decision trees only PCA and k-best features are tested since L2 regularisation does not apply for bagged decision trees. The cross-validated results are then compared to the baseline.

Expected result - Due to the correlation observed between some attributes, it is expected that feature selection and dimensionality reduction techniques should improve the performance of the models.

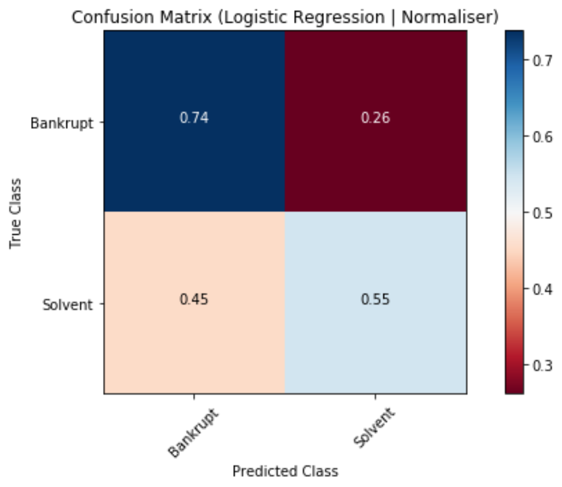
Actual result - Although PCA did slightly better than the baseline in terms of AUC for logistic regression, the expected improvement was not observed. In fact, the baseline model performed better than the models having l2 regularisation and k-best features. A slight improvement by PCA was also noted in sensitivity. For



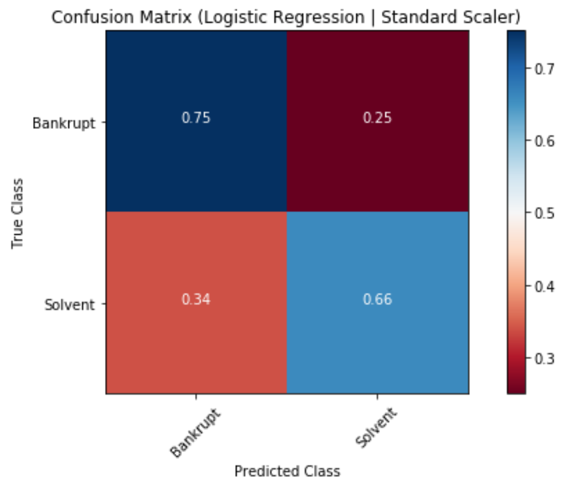
(a) Quantile normalisation



(b) Min-max normalisation



(c) L2 normalisation



(d) Z-score normalisation

Figure 7: The resulting confusion matrices of the different normalisation methods.

bagged decision trees on the other hand, PCA degraded the performance with lower results on all counts. Bagged decision trees with k-best features performed marginally worse than the baseline. This shows that bagged decision trees performs better with high-dimensional data. The results of the experiment can be seen in Table 6.

	Logistic Regression				**Bagged Decision Trees**		
	Baseline	**PCA**	**l2**	**k-best**	**Baseline**	**PCA**	**k-best**
AUC	0.875	0.877	0.874	0.865	0.909	0.875	0.902
sensitivity	0.725	0.775	0.725	0.706	0.581	0.352	0.553
specificity	0.869	0.839	0.87	0.866	0.997	0.996	0.997

Table 6: The results for Experiment 3

3.2.4 Experiment 4 - Measuring the effects of different imputation methods on logistic regression

Setup - In this experiment the class-partitioned mean imputation method used in the baseline model is replaced with three other imputation methods. The first method imputes all missing values within an attribute to the mean for the attribute, the second method imputes to the median whilst the third imputes to the of the attribute partitioned by class. The three imputation methods were run for both logistic regression and bagged decision trees. Once again the cross-validated results are then compared to the baseline.

Expected result - The baseline already incorporates a strong imputation strategy, which also takes into account the class. So the imputations using the mean and median are not expected to improve the results. The class-partitioned median may slightly improve the results, particularly due to the presence of outliers.

Actual result - As seen in Table 7, no improvement in terms of AUC was observed by either of the imputation methods under test for both logistic regression and bagged decision trees. However, the class-partitioned median imputation resulted in a better sensitivity for logistic regression. This cannot be said for bagged decision trees, however, it is noticed that specificity remained stable for all imputation methods.

	Logistic Regression				**Bagged Decision Trees**			
	Baseline	**mean**	**median**	**class-partitioned median**	**Baseline**	**mean**	**median**	**class-partitioned median**
AUC	0.875	0.867	0.815	0.833	0.909	0.858	0.837	0.904
sensitivity	0.725	0.725	0.73	0.742	0.581	0.385	0.301	0.517
specificity	0.869	0.831	0.754	0.791	0.997	0.997	0.997	0.998

Table 7: The results for Experiment 4

3.3 Models implementation

The classifiers were evaluated by selecting the model with the highest *ROC AUC* score using stratified k-fold cross-validation. The custom implementation inherits from a base class[5] provided by the reference implementation which permits the usage of a common algorithm for transforming the data (e.g. by scaling) and evaluating the best model (Algorithm 1).

All classifiers implement methods for fitting a model, predicting the class y of the test set X, and predicting the probability that X belongs to the true or negative class.

3.3.1 Logistic Regression

The fitting method initialises the coefficients β to 0 and addresses class imbalance by finding the weights γ inversely proportional to the class frequencies. The optimal coefficients are then identified using stochastic gradient descent until the maximum number of epochs is reached or the model converges (Algorithm 2).

The probability prediction method uses the coefficients learned from the fitting method to find the probability that each sample in the testing set X belongs to class 1 (bankrupt), and then computes the complement (Equation 10) to find the probability that they belong to class 0 (solvent).

$$Pr(\text{Solvent}) = 1 - Pr(\text{Bankrupt}) \tag{10}$$

[5]https://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html

```
Data: Raw dataset
Result: Performance metrics
result ← {};
for Year in Dataset do
    $\{X_{\text{train}}, y_{\text{train}}, X_{\text{test}}, y_{\text{test}}\} \leftarrow train\_test\_split(\text{Year})$;
    for Classifier in Classifiers do
        pipeline ← $make\_pipeline(\text{Transformer}, \text{Classifier})$;
        model ← $cross\_validate(\text{pipeline}, X_{\text{train}}, y_{\text{train}})$;
        $\hat{y} \leftarrow predict(\text{model}, X_{\text{test}})$;
        performance ← $evaluate(y_{\text{test}}, \hat{y})$;
        result ← result ∪ performance;
    end
end
```

Algorithm 1: Selecting and evaluating the best model.

The prediction method predicts the labels $\hat{y}$ of the corresponding testing set X by calling the probability prediction method to apply label 1 to those samples having a probability greater than 0.5 of belonging to class 1, and apply label 0 to those samples which fall below this probability threshold.

```
Data: $X_{m \times n}, y_{m \times 1}, \alpha, \epsilon$
$\beta \leftarrow [0_1, ..., 0_n]$;
$\gamma \leftarrow [1_1, ..., 1_m]$;
for $i = 0$ to $m$ do
    $\gamma[i] \leftarrow m \times (2 \times |y = y[i]|)^{-1}$;
end
for epoch in epochs do
    $\hat{y} \leftarrow \sigma(X \cdot \beta^T)$;
    loss ← $cost(y, \hat{y})$;
    if $early\_stop(loss, \epsilon)$ then
        break;
    end
    $\Delta \leftarrow \frac{1}{m} \times X^T \cdot ((\hat{y} - y) \times \gamma)$;
    $\beta \leftarrow \beta - (\alpha \cdot \Delta)$;
end
```

Algorithm 2: Logistic regression fitting method taking learning rate α, stopping criterion ϵ, $m \times n$ training matrix X and corresponding label vector y. The logistic function is represented by $\sigma()$.

3.3.2 Decision Tree

The fitting method calls the build_tree method which recursively builds a C4.5 tree (Algorithm 3). The algorithm creates a multi-way tree, finding for each node in a greedy manner the best_split that yields the largest information gain for the categorical targets. The recursion stops when either all nodes are covered or the stop_condition is hit. The stop_condition depends on the number of samples needed to make the split, minimum information gain and the maximum tree depth.

Note the custom decision tree implementation, due to all iterations needed for the greedy algorithm runs pretty slow compared to the scikit learn model which is implemented in C. Having said this, to speed-up a bit scipy library[6] is used to calculate the entropy. Note that the raw implementation of entropy is commented out.

```
Data: $X, y$
if $stop\_condition(X, y)$ then
    return $create\_leaf\_node(y)$;
end
$\mathrm{impurity, threshold, leftX, rightX, lefty, righty} \leftarrow find\_best\_split(X, y)$;
$\mathrm{left\_branch} \leftarrow build\_tree(leftX, lefty)$;
$\mathrm{right\_branch} \leftarrow build\_tree(rightX, righty)$;
return $create\_decision\_node(threshold, left\_branch, right\_branch)$;
```

Algorithm 3: Building a C4.5 Decision Tree from feature set X and label y

The prediction method does a recursive search down the tree and makes a prediction of the data sample from the outcome value of the ?nal leaf.

3.3.3 Bagging

The fitting method as shown in (Algorithm 4) trains n_e number of estimators each time using a sub-feature size n_f and sub-sample size n_s randomly picked.

```
Data: $X_{m\times n}, y_{m\times 1}, n\_e, n\_f, n\_s$
for $i = 0$ to $n\_e$ do
    $X_{n\_s\times n\_f} \leftarrow bootstrap\_feature\_sample(X_{m\times n}, \mathrm{n\_f, n\_s})$;
    $\mathrm{estimator} \leftarrow make\_estimator()$;
    $\mathrm{estimator.fit}(X_{n\_s\times n\_f}, y_{m\times 1})$;
end
```

Algorithm 4: Bagging fitting method taking number of estimators n_e, number of sub-feature size n_f, number of sub-sample size n_s, $m \times n$ training matrix X and corresponding label vector y

The predication method calls each estimator predication method based on the features that the estimator was trained on and then returns the average predictive result.

3.4 Comparison with third party libraries

The performance of the models developed in this assignment was benchmarked with their counterpart implementations in the *scikit learn*[7] libraries. The models used for comparison were the *SGDClassifier*[8] for logistic regression and the *DecisionTreeClassifier*[9] and *BaggingClassifier*[10] for bagged decision trees. The parameters for the models were kept constant.

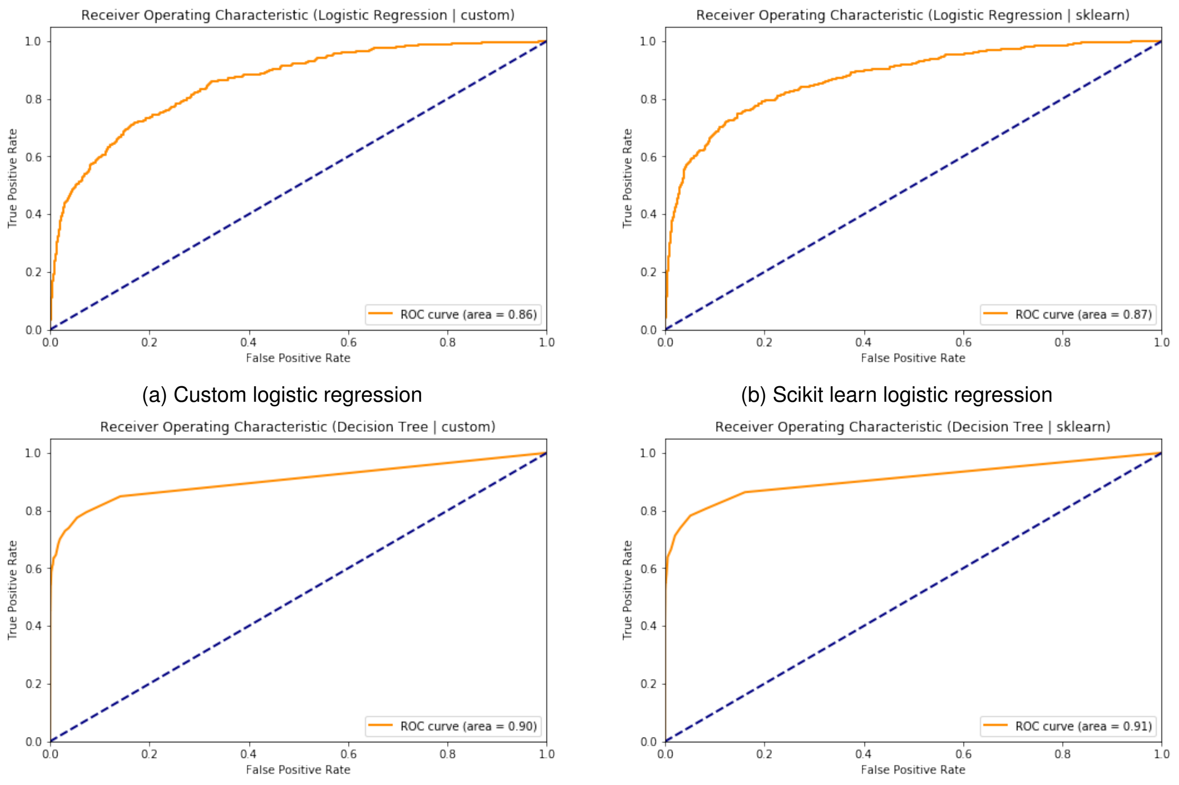
Table 8 shows the results of the comparative results of the benchmark test. It can be seen that whilst the scikit learn implementation of logistic regression outperforms our implementation, the performance of our bagged decision trees model is very close to that of scikit learn. It can also be noticed that similar to our implementations,the bagged decision trees model for scikit learn obtained much better results than the

[6]https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.entropy.html
[7]https://scikit-learn.org
[8]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
[9]https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[10]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

logistic regression model. Figure 8 shows ROC plots for the results. The improved performance of the scikit learn logistic regression model (Figure 8b) over its counterpart (Figure 8a) can be seen. The similarity between the results of the bagged decision trees models (Figure 8c and d) can also be observed. Figure 9 shows the confusion matrices for the custom models and the *scikit learn* models.

	Logistic Regression		**Bagged Decision Trees**	
	custom model	**sk-learn**	**custom model**	**sk-learn**
AUC	0.856	0.875	0.903	0.909
sensitivity	0.749	0.725	0.636	0.581
specificity	0.778	0.869	0.993	0.997

Table 8: The results of the benchmark test with *scikit learn* libraries.



(a) Custom logistic regression

(b) Scikit learn logistic regression

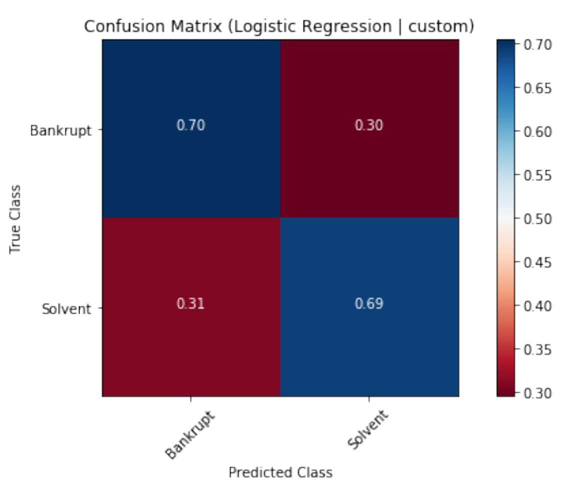
(c) Custom bagged decision trees

(d) Scikit learn bagged decision trees

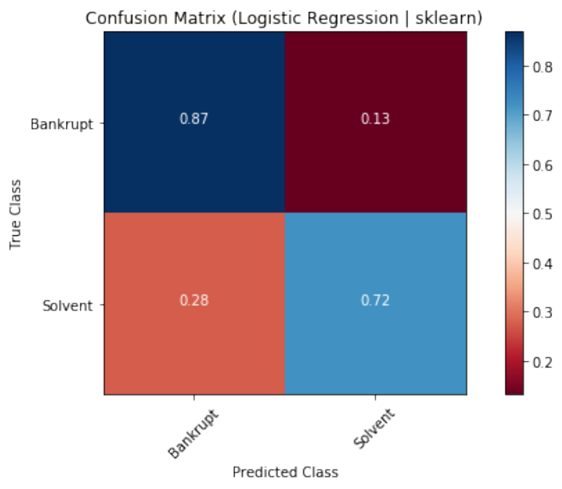
Figure 8: The resulting ROC graphs for the custom implementations and the *scikit learn* implementations of the models.

4 Conclusion

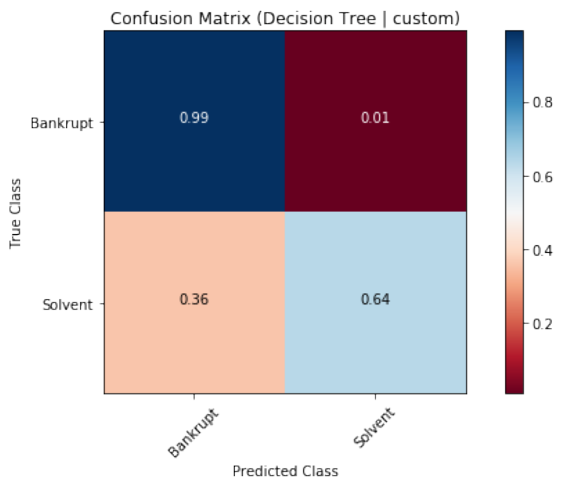
In this work we demonstrated that logistic regression and random forest are suitable machine learning techniques for classifying instances from the Polish companies dataset as bankrupt or solvent. We considered various metrics to evaluate the performance. In cases in which misclassifying bankrupt companies



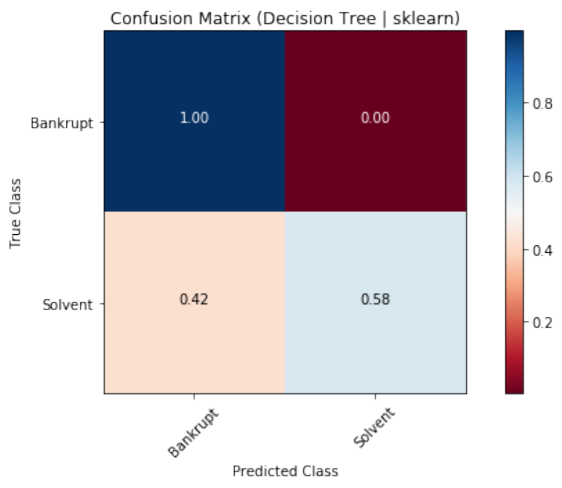
(a) Custom logistic regression



(b) Scikit learn logistic regression



(c) Custom bagged decision trees



(d) Scikit learn bagged decision trees

Figure 9: The resulting confusion matrices for the custom implementations and the *scikit learn* implementations of the models.

as solvent would incur huge losses, we recommend that unseen companies are classified with models that have a higher sensitivity are preferred over those with higher specificity to minimise the possibility of false negatives.

In our experiments we found that bagged decision trees is the best performing technique that requires little to no preprocessing such as scaling. It is also robust to outliers and does not benefit from noise reduction. We found that logistic regression is relatively straightforward to implement and that it achieves good results by training a single model rather than an ensemble of weak learners as in bagged decision trees. Although bagged decision trees were able to classify bankrupt companies much better than logistic regression, they also gave a lot of false positives making them problematic for situations where false positives are costly.

We also observed an improvement in the performance of logistic regression when using quantile normalisation since it reduces the impact of outliers. For both techniques we identified that class-partitioned mean improved the performance of the models. Moreover, we noticed that class imbalance is adequately handled by these techniques and extraneous preprocessing, such as SMOTE or random oversampling, have little to no effect.

In future work, we anticipate that further improvements can be achieved by using coordinate descent to optimise the loss function in logistic regression since the reference library achieved better results using this technique. Furthermore, we expect that the performance of bagged decision trees can be pushed further by implementing a boosting technique such as AdaBoost to include in the ensemble learners with unique classifying characteristics not found in the other learners.

References

[1] A. Tasman and E. Masdupi, "Determinant of financial distress and bankruptcy in miscellaneous industry," 2014.

[2] M. Zikeba, S. K. Tomczak, and J. M. Tomczak, "Ensemble boosted trees with synthetic features generation in application to bankruptcy prediction," *Expert Systems with Applications*, 2016.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed., ser. Information science and statistics. Springer, 2006.

[4] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[5] C. Sammut and G. I. Webb, *Encyclopedia of machine learning and data mining*. Springer, 2017.

[6] S. Dreiseitl and L. Ohno-Machado, "Logistic regression and artificial neural network classification models: a methodology review," *Journal of biomedical informatics*, vol. 35, no. 5-6, pp. 352–359, 2002.

[7] E. L. Allwein, R. E. Schapire, and Y. Singer, "Reducing multiclass to binary: A unifying approach for margin classifiers," *Journal of machine learning research*, vol. 1, no. Dec, pp. 113–141, 2000.

[8] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[9] J. R. Quinlan *et al.*, "Bagging, boosting, and c4. 5," in *AAAI/IAAI, Vol. 1*, 1996, pp. 725–730.

[10] I. B. Mohamad and D. Usman, "Standardization and its effects on k-means clustering algorithm," *Research Journal of Applied Sciences, Engineering and Technology*, vol. 6, no. 17, pp. 3299–3303, 2013.

[11] S. García, J. Luengo, and F. Herrera, *Data preprocessing in data mining*. Springer, 2015.

[12] A. Stolcke, S. Kajarekar, and L. Ferrer, "Nonparametric feature normalization for svm-based speaker verification," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*. IEEE, 2008, pp. 1577–1580.

[13] B. M. Bolstad, R. A. Irizarry, M. Åstrand, and T. P. Speed, "A comparison of normalization methods for high density oligonucleotide array data based on variance and bias," *Bioinformatics*, vol. 19, no. 2, pp. 185–193, 2003.

[14] S. Arlot, A. Celisse *et al.*, "A survey of cross-validation procedures for model selection," *Statistics surveys*, vol. 4, pp. 40–79, 2010.

[15] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," vol. 14, 03 2001.

[16] J. Shao, "Linear model selection by cross-validation," *Journal of the American statistical Association*, vol. 88, no. 422, pp. 486–494, 1993.

[17] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[18] A. M. De Silva and P. H. Leong, "Feature selection," in *Grammar-Based Feature Generation for Time-Series Prediction*. Springer, 2015, pp. 13–24.

[19] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.