CA4010 Credit Card Approval Report

Declaration on Plagiarism
Assignment Submission Form

This form must be filled in and completed by the student(s) submitting an assignment

Name(s):	Harley Martin (17401932) Ruth Leavey (17232886)
Programme:	Computer Applications and Software Engineering
Module Code:	CA4010
Assignment Title:	CA4010 Continuous Assessment
Submission Date:	22/11/2020
Module Coordinator:	Mark Roantree

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I/We have read and understood the Assignment Regulations. I/We have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

I/We have read and understood the referencing guidelines found at http://www.dcu.ie/info/regulations/plagiarism.shtml , https://www4.dcu.ie/students/az/plagiarism and/or recommended in the assignment guidelines.

Name(s): _Harley Martin Ruth Leavey_____ Date: ___22/11/2020_____

Section 1: Introduction

Idea

Our project idea was to create a classification model for credit card applications and determine whether an applicant's application is approved or not based on applicant's personal profiles and their record of previously paid off loans.

Dataset

The dataset we used for our idea to train and test our model contained 2 CSV files, an application record and a credit record. The application record contained 438,588 tuples, where each tuple represented an applicant and the attributes that made up their personal profile (eg. applicant ID, gender, income amount, education level, marital status etc). The credit record contained tuples representing 45,985 customers and the record of whether they paid back their loans on time each month, and if not, how late they were in paying back.

Plan

1. Create status classification using credit_record

It is important to note that a class column, representing whether or not the application is approved, is not present in either application.csv or credit_record.csv. Therefore, the first step in our plan is to create a classification column for the applicants in the dataset.

This is achieved using the credit_record.csv file. The dataset is first filtered so all information available is regarding loans which are at least 20 months old. This is because determining whether someone is a good or bad customer is not possible if their loan has existed for a shorter period of time.

The attribute 'STATUS' in the table represents the number of days an applicant's loan is/was past due. It is decided that an applicant is deemed a bad customer and a risk to be granted application approval if their customer loan repayment status where loan was/is past due more than sixty days. If an applicant's customer loan repayment status where loan(s) was/is always paid within 60 days they were deemed a good customer and not a risk for application approval. The 'status' class column is created where bad customers are represented by a 1 and good customers represented by a 0. It is important to remember throughout this report that, perhaps unintuitively, 1 indicates a customer who is risky, who has a record of letting bills run overdue, who is a bad candidate for a credit card application, and 0 indicates a customer who is not risky, who has a record of paying their bills on time or only shortly overdue, who is a good candidate for a credit card application. Some of the code from https://www.kaggle.com/rikdifos/eda-vintage-analysis was used to give help produce the "good" or "bad" value to each ID.

2. Merge tables by ID

Once the class column 'status' representing customer loan repayment status is created, credit_record.csv is merged with application.csv by 'ID'. During this process, data is lost, particularly in application.csv, as only the intersecting IDs which appeared in both credit_record.csv and application.csv are used. After merging, a new dataset full_set.csv is created where each unique 'ID' has all existing attributes and values from the application.csv file as well as the 'status' attribute which represents the classification column.

3. Analyse and clean data in new merged table

The next step in our plan is to analyse the data in our new dataset full_set.csv. Here, attribute types and further attribute information are examined. After analysing the data, the next step in our plan is to apply any data cleaning which might be necessary. See *Section 2: Analyse your data* for more detail.

4. Algorithm type

Finally , the last step of our plan for this project is to decide on an algorithm to help predict the classification of whether or not an applicant is good (0) or bad (1), run the algorithm against a test set derived from our original dataset and examine the results both subjectively and objectively.

Section 2: Analyse your data

Attributes

The first form of data analysis that was carried out was the exploration of our attribute types. We found that our attributes consisted of data types object, int and float representing both nominal and numeric data. Some attributes were continuous e.g. *ID* and some were discrete *income type*.

During this analysis it was identified that both *date of birth* and *employment years* were represented in large negative integers counting backward from today, 0 where yesterday is -1. The *date of birth* column was changed to the new column *Age* where each value represents the age of the customer calculated by dividing the original value by -365. The equivalent was done for employment years to calculate the number of years each customer had been working for.

Analysis

Using the program SPSS, the data was analysed by looking at central tendency, dispersion and graphs which represented. Throughout this analysis, the following was discovered:

The overall percentage of bad customers (who were greater than 60 days overdue at any time during their loan repayment period) was 2.9 % (=444/15166).

Initially mean and standard deviation and percentages were determined to measure central tendency and dispersion for all the data attributes, both in the full dataset and in sets filtered by account repayment status (Table1). *Account repayment status* refers to the classification of whether a customer is good (0) or bad(1), where good(0) means applicant always paid loans within 60 days and bad(1) means the applicant's loan was greater than 60 days overdue.

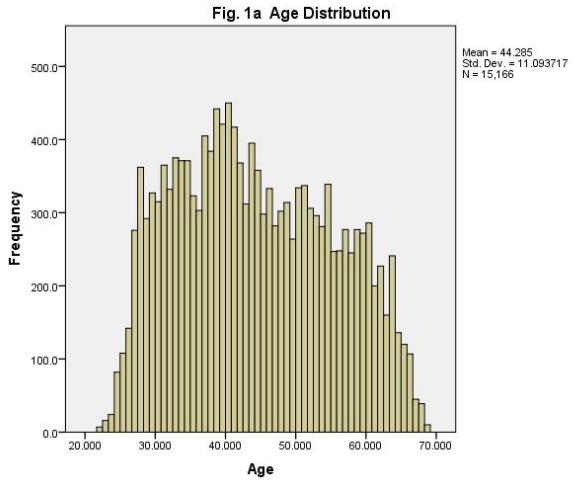
Table 1. Atrribute Characteristics of Individual Loan Customers Overall and By Account Repayment Status

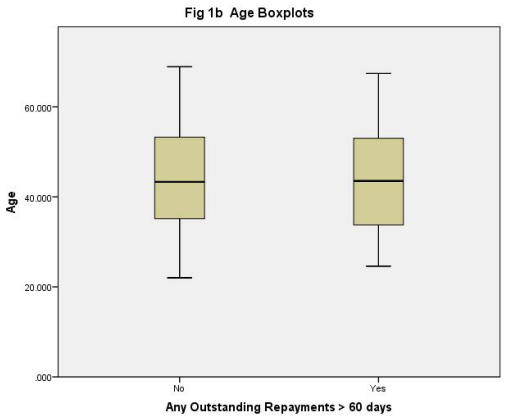
Attribute	Full Database	Customer Accounts always paid within 60 days	Customer Accounts at times > 60 days Overdue
	n=15166	n=14722	n=444
	Mean (SD) or Percent	Mean (SD) or Percent	Mean (SD) or Percent
Age	44.3 (11.1)	44.3 (11.1)	43.9 (11.5)
Female gender	67	67	63
Marital status			
Civil Marriage	7.5	8	8
Marriage	71	71	64
Separated	6	6	5
Single	12	11	16
Widow	4	4	8
No. Of Child Dependents			
0	69	69	70
1	20	20	19
2	9.2	9	9
>2	2	1	2
No. Of Siblings			
< 2	18	18	22
2	55	55	51
>2	28	28	26
Annual Earnings	189,271 (101968)	189,396 (102,229)	185,110 (92,933)
Active Years of current Employment*	7.8 (2.4)	7.8 (6.7)	6 (5.6)
Income Category			
Commercial Associate	24	24	25
Pensioner	16	16	20
State Servant	9	9	6
Student	0	0	0
Working	51	51	50
Owns a Car	40	40	37
Owns a Property	65	66	59
Has Mobile Phone	100	100	100
Has Work Phone	23	23	28
Has Home Phone	30	30	32
Uses eMail	91	91	93
Educational Level Achieved			
Academic degree	0.1	0.1	0
Higher Education	28	28	26
Incomplete Higher	4	4	6.3
Lower Secondary	1	1	2
Secondary/sec. special	67.2	67	65

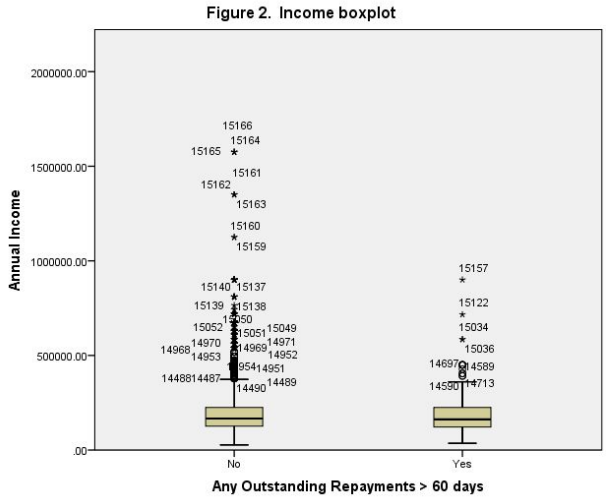
* missing data

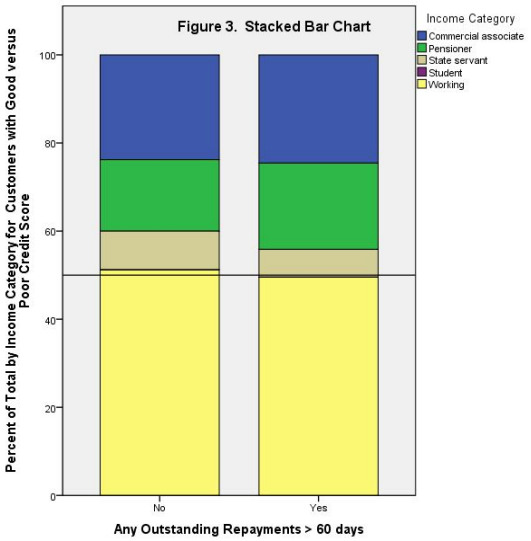
Age was found to be normally distributed on a histogram and without significant outliers on quantile plotting (example plots in figures 1a and 1b below), dispersion was similar between groups and differences were small. It was categorised into five quintile groups in preparation for the decision tree algorithm.

Annual income by contrast was significantly skewed to the right by high earners in the main dataset and in both groups (see box plot figure 2). *Annual income* also showed a large dispersion around the mean, see standard deviation (table 1). This could have been a problem if it was being used as a continuous variable in making an algorithm but for the purposes of building a decision tree algorithm it was grouped into quintile categories of earnings from lowest to highest.









Current active years of employment was the only continuous data attribute with a large amount of what appeared to be nonsensical data in the data cleaning stage affecting 2465 or 16% of all cases. The nonsensical data was coded as missing and a consideration was given to imputing this missing data to the mean value for this attribute. However, on closer analysis of our dataset, this missing data was 100% correlated with the income category 'Pensioner'. This is a common *income type* attribute (16% of all customers, table 1) that appears distributed differently by customer loan repayment status (Table 1, Table2) and was deemed likely to be important for developing a predictive algorithm. Being unable to impute a value for the *current active years of employment* missing data, we opted not to use this attribute as we risked losing a lot of informative cases.

Some small differences existed in the distribution of the other categorical attributes by *loan repayment status*(table.1). For example see *marital status* in table 1 where 64% and 71% of customers with a poor versus good repayment history were coded as *married*. In contrast *mobile phone ownership* was 100% across all groups and it will therefore not be used in future project development.

Cross Tabulation with *loan repayment status* by categorical attributes shows the relationship of different attribute categories to the percentage of all Customer Accounts based on their *repayment status*(table 2). The absolute differences in percentage distributions by attribute category in this table are small. To display this graphically we developed a number of stacked bar charts (see example Figure 3 above).

Table 2. Atrributes of Individual Loan Customers Overall Account Repayment Status

Attribute Yes	Percent of All Customer Accounts (n=15166) always paid within 60 days by Attribute Category	Percent of All Customer Accounts (n=15166) ever overdue payment > 60 days by Attribute Cateory
Female/ Male	97.2 / 96.8	2.8 / 3.2
Marital status		
Civil Marriage	96.9	3.1
Marriage	97.4	2.6
Separated	97.7	2.3
Single	96.1	3.9
Widow	94.1	5.9
No. Of Child Dependents		
0	97	3
1	97.3	2.7
2	97.1	2.9
>2	95.5	4.5
No. Of Siblings		
< 2	96.3	3.7
2	97.3	2.7
>2	97.2	2.8
Income Category		
Commercial Associate	97	3
Pensioner	96.5	3.5
State Servant	97.9	2.1
Student	100	0
Working	97.2	2.8
Owns a Car (y/n)	97.2 / 97	2.8 / 3.2
Owns a Property (y/n)	97.4 / 96.5	2.6 / 3.5
Has Work Phone (y/n)	96.5 / 97.2	3.5 / 2.8
Has Home Phone(y/n)	97 / 97.1	3 / 2.9
Uses eMail (y/n)		
Educational Level Achieved		
Academic degree	100	0
Higher Education	97.2	2.8
Incomplete Higher	95.2	4.8
Lower Secondary	94.7	5.3
Secondary/sec. special	97.2	2.8

Later attributes removal

As described later in *Section 3*, the algorithm we chose required us to calculate the information gain of each attribute. The calculations caused the removal of a couple of attributes from the dataset as their predictive power was negligible. None of the attributes had a noticeably significant predictive power. This was interesting because a simple glance at the attribute names could easily lead someone to suspect that a customer's income would probably be of significance in deciding the classification. However, when examined accurately, income is not much more influential than any other attribute. They all played a relatively equal role in deciding the class, except for those with predictive power so small that they were insignificant, such as whether or not they had an email address, or a phone.

Below is the final list of attributes used in the decision tree algorithm, along with the predictive power computed for them:

```
NAME_FAMILY_STATUS:        0.0008141807999079154      #1
AGE_QUINTILES:             0.0004854122188406862      #2
NAME_EDUCATION_TYPE:       0.0003471838255466597      #3
FLAG_OWN_REALTY:           0.00025524227472095706     #4
NAME_INCOME_TYPE:          0.00022977565392356958     #5
CNT_FAM_MEMBERS_BRACKET:   0.000208968678831023       #6
NAME_HOUSING_TYPE:         0.00018372559038076774     #7
FLAG_WORK_PHONE:           0.00014170847161787403     #8
CODE_GENDER:               0.00007739752789015508     #9#
CNT_CHILDREN_BRACKET:      0.000073917443666602726e   #10
ANNUAL_INCOME_QUINTILE:    0.00007136595742784912     #11
```

Section 3: Describe your algorithm

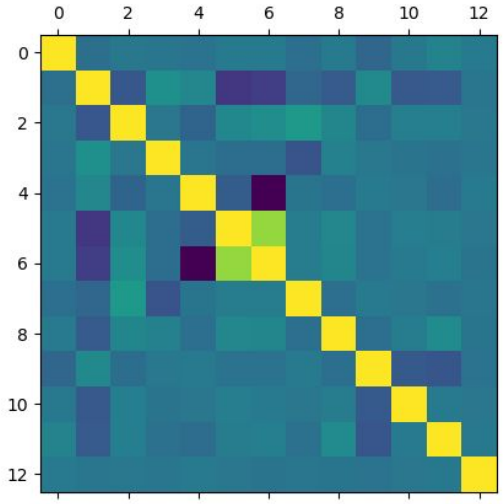
Type of algorithm

The decision tree algorithm was chosen to build the classification model. Apart from the fact that decision tree algorithms belong to unsupervised learning classification algorithms which relates to our dataset, this decision was also due to the easy-to-understand and visualisation nature of the decision tree algorithm which would hopefully allow for the best interpretation. A decision tree tries to determine which attribute has the most influence on the classification of tuples, and then uses the different values of that attribute to split the dataset into partitions as purely as possible. Pure partitions become a leaf node, and non pure partitions are split on the next most influential attribute, until every branch ends on a leaf node. The path to follow to each leaf node is a rule in the decision tree.

Information gain values calculated for each attributes

Since decision trees use information gain to choose which attribute to split on, we calculated the information gain of each attribute in our dataset. This was done to assign predictive scores to each attribute in the order that we expected to be the most influential on classification. This led us to remove some attributes, which we explained in *Section 2*. A correlation graph was used to visualize the correlation between each individual attribute and the classification attribute. This backed up our discovery that all attributes held relatively equal weight in regards to influence and that these correlations were all low.

The numbers on the x axis and the y axis of the graph represent the attributes in our tuples. The yellow squares represent a 1:1 correlation, as the diagonal line of yellow squares indicates each attribute's correlation to itself. The colours get darker as the correlation weakens. We are looking for the correlations of each attribute against the 12th attribute (the classification). As we can see along the x axis, and mirrored along the y axis, every square for the 12th attribute is a teal colour, with very little variation in shading. This supports our findings that there are no stand out attributes that contribute to the classification the most. The teal colour also indicates that the correlations that do exist are not particularly strong.



Algorithm use

We used the sklearn library in python to generate our decision tree. By default, sklearn's decision tree uses gini index for splitting which creates binary splits. Not all of our attributes were of the binary type so we configured our tree to use information gain instead.

We first randomly split our data into a training set and a test set using the train_test_split() method from sklearn's library. This split our dataset into a training set and a test set at 80%/20% ratio as we asked it to. Our instructions for this assignment were to use this ratio. The training set was divided into X_train and

Y_train. These consisted of all the tuples in our dataset but with the classification removed, and of only the classifications for those tuples in X_train, respectively. This contained 80% of the tuples in our dataset and the decision tree later used these to *learn* how to match the tuples to the corresponding classification. The testing set was divided into X_test and Y_test. These two sets contained the same respective information and the decision tree later used these to test the accuracy of its algorithm when it *guessed* the classification of the test tuples.

```
33   X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.2)
34   X_train_bal, Y_train_bal = SMOTE().fit_sample(X_train, Y_train)
```

When we ran our program and asked it to return the accuracy of the test to us, we ran into a ValueError, where our program could not return our result, as it had trouble converting strings to floats. With research we understood the sklearn decision tree only worked with numerical data so we researched further how to fix this and used skleanr's labelEncoder() method to change our attributes to numerical types.

After this, we ran our decision tree program several times. Everytime it ran, it returned a 97% accuracy rate. We realised this was not as successful as it looked at first glance because 97% of our tuples belonged to class *0* (good, or low risk candidate for credit card application) and only 3% belonged to class *1* (bad, or high risk candidate for credit card application). This meant of course that our tree had a 97% chance of guessing correctly if it guessed class *0* on any tuple. We needed to balance this out so we used the SMOTE() method (Synthetic Minority Oversampling Technique) to balance out the skew of classifications in our training set. With the SMOTE method in play, our decision tree produced an accuracy of approximately 80% after running a couple of times. This is a lower accuracy than the 97% but it is actually more *reliable* due to the balancing of the training set. The SMOTE method is explained in more detail in the results section of this report.

Our decision tree also provided us with a confusion matrix so we could gauge the number of times our tree guessed each classification and whether or not it was correct. We saw the number of times it guessed 0 and was correct, guessed 0 and was incorrect, guessed 1 and was correct, and how many times it guessed 1 and was incorrect. The results of the confusion matrix are discussed further in *Section 4* of this report.

Section 4: Present your results

1. **Present your results in an objective fashion: how accurate were you?**

Accuracy percentage

The accuracy percentage given for the decision tree algorithm was 80.94924192485168%. This number represents the computation of subset accuracy meaning how accurate the predicted test set was in relation to the actual test set.

```
(base) C:\Users\ruthl\Downloads\csv_files>python accuracy.py
Accuracy:  0.8094924192485168
```

Confusion matrix given

The confusion matrix given after running the algorithm is presented below. The confusion matrix is used to help further measure the performance of the classification algorithm so as to gain as much understanding for further interpretation.

	0	1
0	2329	614
1	57	34

From the confusion matrix that was given, probabilities were calculated to further understand the accuracy and the effectiveness of our decision tree algorithm in predicting the status of loan repayment class which tells us whether a customer is good (0) or bad (1).

The following excel table was created to further analyse our confusion matrix table and a series of probabilistic calculations were made relating to the numbers given.

	A	B	C	D
1		P0	P1	total R0, R1
2	R0	2414	514	2928
3	R1	64	42	106
4	total P0,P1	2478	556	

P0 = Predicted number of 0s
P1 = Predicted number of 1s
R0 = Real/actual number of 0s
R1 = Real/actual number of 1s
Total P0, P1 = total number for P0 (B4) and P1 (C4)
Total R0, R1 = total number for R0 (given in D2) and R1 (given in D3)

A. P(R1 | P1) = 42 / 556 = 0.07553957 = 7.6%
B. P(R0 | P0) = 2414 / 2478 = 0.97417272 = 97.4%
C. P(P1 | R1) = 42 / 106 = 0.39622642 = 39.6%
D. P(P0 | R0) = 2414 / 2928 = 0.82445355 = 82.4%
E. 1 - D = 1 - 0.82445355 = 17.6%

A. This represents the positive predictive value of the algorithm being 7.6%. Positive predictive value refers to the proportion of customers with positive risk scores (P1) who are actual/real bad customers.

B. This represents the negative predictive value of the algorithm being 97.4%. Negative predictive value refers to the proportion of customers with negative risk scores (P0) who are actual/real good customers.

C. This represents test sensitivity being 39.6%. Test sensitivity refers to the proportion of applicants who are actual/real bad customers that receive a positive risk prediction (P1).

D. In contrast, D. represents test specificity being 82.4% where test specificity refers to the proportion of applicants who are actual/real good customers that receive a negative risk prediction (P0).

E. 1 - specificity is the false positive rate. This represents the proportion of all actual/real good customers who receive a positive risk prediction (P1).

The decision tree structure
The graphical decision tree is one given a max_depth of 11 as mentioned in Section 3. The tree begins at node flag_own_realty (whether someone owns property or not) before going to the nodes age, number of children etc. until it reaches the final status classification.

2. Present your results in a subjective fashion: in your opinion why do you think you got these results?

Decision tree
We configured our decision tree to have a max depth of 11. The decision tree cannot have a depth greater than 11, as our tuples have 11 attributes, and the same attribute cannot be selected twice in the same branch. We chose this number after testing the accuracies received from different max depths. We ran our program with max depths of 2, 3, 4 5 onwards to see how our accuracies changed. The highest accuracies we received came from the instances where we ran the program with max depth of 11 so we chose this number. We could have obtained a neater, more concise tree with a smaller mx depth but we would have

to sacrifice our accuracy. Thus, we believe this to be the best choice for max depth as it balanced conciseness and accuracy as best as possible.

Regarding the resulting decision tree after implementing our algorithm, we noticed that it was a very large decision tree. In our opinion this is the result of having a large dataset, a large number of attributes and many attributes having various discrete values. However, most importantly, we believe that due to the lack of any strong predictive power associated with any attribute (represented in Section 2 and 3), it is more likely that the algorithm has to take a lot more into consideration before it can attribute a 1 or a 0 classification value.

We believe the fact that our dataset did not have any stand out influential attributes probably plays a role in the model's accuracy. As our information gain results suggested and our attribute correlation graph confirmed, all of the attributes had approximately equal effect on the classification of any given tuple. This made it difficult to choose the best splitting criterion for each internal node in the decision tree. Because of the fact that there was so little difference in choosing one attribute over another as the splitting criterion, this meant that on each run of our program, the permutation of the random selection of training and testing partitions could easily lead to different rules in our decision tree. If there were one or two attributes in our dataset that were significantly more influential on the classification than the other attributes, our accuracies could have been stronger and less varied on each different run.

Features contributing to accuracy

After we first split our dataset into training and testing partitions and ran our program to generate our decision tree, we kept getting 97% accuracy results. Momentarily, this seemed like our tree was almost perfectly classifying our test sets every time. We shortly realized that this correlated to the fact that 97% of our dataset was classified as 0 (good applicant) and 3% was classified as 1 (bad applicant).

The challenge of working with extremely imbalanced datasets is that there are too few instances of the minority class for the algorithm to create efficient rules on information gain and attribute splitting. To overcome this, we used the SMOTE() method (Synthetic Minority Oversampling Technique) to balance out the classifications in our training set. SMOTE synthesizes new instances of the minority class, based on the existing instances of that class. From the feature space, it selects a random instance of the minority class and then selects *k* of its nearest neighbours (k defaults to 5). Of these neighbours, one is randomly selected and a synthetic instance is created at a randomly selected point between these two instances in the feature space. This balancing of our classifications helped stabilize our accuracy. It reduced it from 97% to around 80% but the new result reflected a more true accuracy.

Algorithm accuracy, confusion matrix results

From our accuracy score given, confusion matrix for this test set and probabilistic calculations made from the confusion matrix, it is our opinion that the decision tree algorithm has discriminating value in predicting future customer behaviour. Although individually each attribute offered only a small predictive value, collectively the power of the decision tree modelling system with a multitude of attributes and an appropriate depth produced a very good accuracy score in this validation set of 0.8 for our algorithm.

A large group of test-set customers had a prediction score of zero and their likelihood of regular on-time payments was 97.4%. A bank would not be unduly worried about this large group of customers. A smaller group of test-set customers (18.4%) had a prediction score of 1 and their likelihood of maintaining regular on-time payments was less at 92.4%. We are aware from these percentages, that the value to the bank of the risk score in identifying the likelihood of a future good customer may exceed it's value in identifying a future bad customer. This is expected because the pre test probability of being a bad customer was very low in the first place (3%, from initial analysis).

It is also important to note that 17.6% of good customers receive a risk score of 1, falsely predicting that they will be bad customers. This 17.6% of customers are at risk of being inaccurately classified for credit card approval. Hence this produces a cost to identifying someone as a bad customer.

From all the information discussed above, the algorithm itself may vary in its application in the approach to credit card approvals/applications, outlined as our idea in the introduction section. A bank that wishes to place a large amount of importance on avoiding bad debt may wish to use this algorithm to deny credit card applications to customers with a prediction risk score 1. This may occur at a time when the bank's balance sheet includes many people with significant overdue loans or at a time when the bank's access to money is limited e.g. during the last recession. A bank might alternatively decide to set different lending limits, charges or interest rates to customers based on their prediction scores from this algorithm.

As mentioned previously, In the algorithm applied to our test-set, when it guesses 0, it is mostly correct, by a large majority, and rarely incorrect. When it guesses 1, it is mostly incorrect, by quite a significant majority. Because of this information, we are of the opinion that the majority of banks would not use this algorithm as the sole factor in deciding on approval of credit card applications. They may, however, find this algorithm beneficial for identifying customers to discuss further before credit card approval or to collect further data on.