TASK 1

The objective of this task is to pre-process the dataset, for use in later tasks. This entails cleaning and transforming the data, which results in a more reliable dataset that yields better results when mined or used as input for machine learning algorithms.

Before a dataset can be pre-processed, it is important to learn what the data contains. This is done using tools in WEKA. Each instance of the dataset contains the data relating to one credit card application. The dataset contains 690 instances and 16 attributes (including class label). The class label attribute can take 2 values (+, -). This represents the credit card application being accepted or rejected.

STEP 1 – DEALING WITH MISSING VALUES

Figure C in the appendix details each attribute and the number of instances where the attribute is missing in the dataset. The total number of missing values in the dataset is 67. This means at most 67 instances have a missing value, which is 9.7% of the instances in the dataset. These instances with missing values cannot remain as they are in the dataset as they reduce the reliability of the dataset, which results in reduced performance when used for machine learning or data mining. The two options are to either remove said instances or to replace the missing values. Removing the instances is not an option as the dataset is already small and reducing its size further can lead to its own issues, such as increased possibility of overfitting during training.

Therefore, the values must be replaced. Due to the dataset having attribute names and values that are meaningless, values cannot be inferred using domain knowledge. A common methodology for replacing values is to replace with constants. Another method is to impute the values with either the mean or modal value of that attribute. Numeric attributes have their missing values imputed using the mean while nominal attributes use the modal value. Imputed values can be calculated using either only instances which are of the same class, or instances of any class. This leads to three strategies to test:

Method	How is it achieved?	Accuracy
1. Replace missing values with means and modes, imputed values calculated using instances regardless of class	WEKA's ReplaceMissingValues filter, with ignoreClass set to False	JRip: 85.07% MLP: 82.75%
2. Replace missing values with means and modes, imputed values calculated using only instances of same class	WEKA's ReplaceMissingValues filter, with ignoreClass set to True	JRip: 85.07% MLP: 82.75%
3. Replace with constants (sam, 20)	WEKA's ReplaceMissingWithUserConstant filter, with 'sam' replacing missing nominal variables and '20' replacing missing numeric values.	JRip: 86.09% MLP: 82.90%

Method 1 and 2 yielded the same result using the JRip classifier, which was confusing. The expectation was that imputing values using only instances with the same class label would introduce bias, and would increase the possibility of overfitting, leading to reduced classification accuracy. To ensure this was not an anomaly with the JRip classifier, I tested both methods using an MLP, and once again they yielded the same accuracy. Seemingly, for this dataset, the bias introduced by this is negligible. However, both methods were outperformed by method 3. Replacing missing nominal attributes with an arbitrary class and missing numeric attributes with an arbitrary number beat replacing them with the mean and mode (even if just narrowly). This was also confusing. I also tested this method using an MLP to make sure the result was repeatable across a different classifier, and it was (although just slightly). It should be noted the value of 20 was chosen as it yielded the highest accuracy. Other values yielded worse performance than methods 1 and 2. A reason why this could slightly outperform methods 1 and 2 is that some attributes have large outliers (A8, A11, A14 and A15, see appendix figure A) which could skew the mean to an area of the distribution which negatively affects the training process, leading to fewer correct classifications. Moreover, 20 is a trial-and-error discovered best option – changing the value even slightly results in less performance than methods 1 and 2. Instinctively, method 3 is appealing due to it having better classification accuracy. However, this is only a negligible improvement over methods 1 and 2, and the repeatability of the improved accuracy is a worry. Therefore, method 1 is my choice, as it has only slightly worse performance, and it has theoretical validity so less uncertainty as to its performance across a range of techniques.

Step 2 – Handling Range Differences

Range differences are typically handled using feature scaling, either normalisation or standardisation. As the absolute values of the numeric attributes vary greatly (A8 having a range of [0, 28.5] and A15 [0, 100 000]), left as they are, some attributes will dominate the resulting models whilst others will have comparatively negligible influence on the results of mining or machine learning. Normalisation transforms the values for each attribute such that the range is [0, 1]. Thus, no one attribute can dominate the model as they all have the same range. Standardisation transforms the data such that the mean is 0 and the standard deviation is 1, meaning all features have the same scale (but with no upper or lower bounds). Standardisation is best used when the data follows a Gaussian (normal) distribution (but this is by no means a requirement). Normalisation is best used when there are few or no outliers and data is uniformly distributed throughout the range. This is because if there are significant outliers and the data is not uniform, lots of the data will be squished into a very small range of values, with the rest of the distribution being sparsely populated (typically large ranges of few or no values towards the upper or lower bound of the range).

This provides 2 methods to test:

Method	How is it achieved?	Accuracy
1. Normalise data	WEKA's Normalise Filter	JRip: 86.23% MLP: 83.48% RandForest: 86.52%
2. Standardise data	WEKA's Standardise Filter	JRip: 85.22% MLP: 83.77% RandForest: 86.38%

Naïve Bayes, IBk and SMO and SGD yielded the same accuracy for each method. Getting so many results that were the same despite changing the scaling method was surprising. I checked that the relations were changing between the normalised and standardised ones, and after achieving the same results despite changing the scaling method (including on the unprocessed data) I confirmed it was working correctly by using other classifiers, and the accuracy was different. IBk and SMO are distance-based classifiers and so them yielding the same accuracy (including on the unprocessed data) was a surprise, as the rescaling should affect them. But I checked all the set-ups and tested other classifiers, and all was working correctly. Normalising the data leads to slightly better performance across a larger range of classifiers. Despite the 4 attributes that have outliers, they are evidently not significant, as if they were, method 1 would perform much worse than method 2. Standardisation may have performed slightly worse as the data does not follow a Gaussian distribution, which is preferred for using standardisation. In addition to providing slightly better performance, normalisation provides a convenient range when converting nominal values to numeric values. Thus, I choose Normalisation.

Step 3 – Text to Numeric Feature Conversion

The dataset contains both textual and numeric features. These textual features need to be converted to numeric features as some algorithms only work on datasets with numeric features. As the data is normalised, each nominal feature will be converted to have a range of [0, 1]. This range will be divided by the number of values the attribute can take, to provide an even distribution. E.g. if the nominal attribute can take four values, they will be converted to 0, 0.33, 0.66 and 1 respectively. This step is done using the RenameNominalValues filter in WEKA then updating the .arff file.

TASK 2

The objective of this task is to assess each feature's ability to predict the class variable and rank attributes by importance to identify a subset of features that are most suitable to be used for machine learning or data mining. Multiple methods will be used to assess and rank the features, and the composition of the results of these methods will be used to decide the selected features.

METHOD 1 – LINEAR REGRESSION

The result of classifying using linear regression is an equation. Each coefficient of this equation represents the feature's correlation to the class variable (and consequently its importance in being selected). Statistically the coefficient represents the change in the value of the output variable given a unit increase in the value of the attribute. The result was obtained by loading my normalised, pre-processed dataset (from task 1) into WEKA, and selecting linear regression under the classifier tab. Rankings ->

Ranking	Attribute	Coefficient Value
1	A15	1.12
2	A9	-0.59
3	A11	0.49
4	A14	-0.32
5	A8	0.23
6	A10	-0.13
7	A7	-0.12
8	A4	-0.11

All other attributes were given a coefficient of 0 (no correlation).

As the data was normalised, the coefficient value gives the feature's relative importance in predicting the class variable. Positive values indicate a positive relationship between the feature and the class variable, and a negative value a negative relationship. With the relative importance's calculated, the attributes could be ranked. A15 is the most important feature for predicting the class variable, with them having a strong correlation, with A9, A11 and A14 having moderate correlations with the class variable. A8, A10, A7, and A4 have weak correlations with the class variable, but still have some impact on its value. All other features have no impact on the value of the class variable.

METHOD 2 – INFORMATION GAIN

Information gain is a technique which calculates the entropy for each feature, the greater the entropy, the greater the information provided by knowing the value of that feature, and the more important the feature is. The results were calculated using the InfoGainAttributeEval attribute evaluator and the Ranker search method under the Select attributes tab in WEKA. Rankings ->

Ranking	Attribute	Information Gain
1	A9	0.43
2	A11	0.21
3	A10	0.15
4	A15	0.11
5	A8	0.11
6	A6	0.087
7	A3	0.041
8	A7	0.037
9	A14	0.037
10	A4	0.030
11	A5	0.030
12	A2	0.023
13	A13	0.0091
N/A	A12	0
N/A	A1	0

Consequently, A9 is the most important feature according to this method, as the most information is gained towards the prediction of the class variable when we know it's value. As the value of information gain reduces moving down the rows, so does the feature's importance. According to this method A12 and A1 provide no information toward predicting the class variable. It should be noted that lots of the attributes, whilst providing information, provide little information and so their inclusion in the final selection will be dependent upon their performance in other methods.

Ranking	Attribute	Correlation Value
1	A9	0.72

Method 3 – Correlation Evaluation

This method calculates the correlation between each feature and the output variable. The greater the correlation, the more important the feature. The results were calculated using the CorrelationAttributeEval attribute evaluator and Ranker search method, under the Select attributes tab in WEKA. Rankings ->

2	A10	0.46
3	A11	0.41
4	A8	0.32
5	A3	0.21
6	A15	0.18
7	A5	0.17
8	A4	0.17
9	A2	0.16
10	A7	0.13
11	A13	0.10
12	A14	0.10
13	A12	0.032
14	A1	0.029
15	A6	0.0040

The correlation values show that A9 is the most important feature, and A12, A1 and A6 all have negligible correlation with the class variable. It should be noted that from rank 5 through 12 there is only a 0.11 difference between the values, which is a small difference considering the largest value is 0.72 and the difference between rank 4 and 5 (only one place above this range) is 0.11 on just its own.

Discussing Obtained Results and Formulating My Own Ranking

The first step to curating my chosen subset of features is to rank them all based upon their performance across the three feature selection methods. This is done by calculating each attribute’s mean ranking position across the three methods. N.B. if an attribute received a 0 in a method, it is given rank 15 for that method.

Attribute	Method 1 Rank	Method 2 Rank	Method 3 Rank	Mean Rank
A1	15	15	14	14.67
A2	15	12	13	13.33
A3	15	7	10	10.67
A4	8	10	8	8.67
A5	15	11	7	11
A6	15	6	15	12
A7	7	8	10	8.33
A8	5	5	4	4.67
A9	2	1	1	1.33
A10	6	3	2	3.67
A11	3	2	3	2.67
A12	15	15	13	14.33
A13	15	13	11	13
A14	4	9	12	8.33
A15	1	4	6	3.67

With the mean ranks calculated, they can be put in a final ranking order according to their performance across all methods.

Ranking	Attribute	Mean Ranking
1	A9	1.33
2	A11	2.67
3	A10	3.67
4	A15	3.67
5	A8	4.67
6	A7	8.33
7	A14	8.33
8	A4	8.67
9	A3	10.67
10	A5	11
11	A6	12
12	A13	13
13	A2	13.33
14	A12	14.33
15	A1	14.67

Despite having the same mean ranking A10 was ranked above A15 and A7 was ranked above A14 as their ranks had less variance from the mean, leading to increased confidence about the generality of their importance relative to other features.

The attributes ranked 1-5 are ones that performed very well across all the methods, leading to mean rankings of less than 5 for the top 5 ranked features. The next three features (ranks 6-8) are ones that were middle-of-the-road important across the three methods, and importantly were classified by the regression model. Therefore, all features ranked 1-8 will be included in the selected features list.

All the attributes below this point in the ranking were not given a rank by the regression model, as according to the regression model they had no correlation to predicting the output variable. Therefore, ranks 9-15 can only be compared using information gain and correlation evaluation. Ranks 12-15 performed poorly across all methods so will not be included in the final selection. A6 (rank 11) performed well compared to other features in method 2, but came last in the method 3, not to mention not being included in the regression model, so shall not be included. A3 and A5 performed similarly to the attributes ranked 6-8 across methods 2 and 3, their mean rank being lower due to their lack of inclusion in the regression model. So evidently, they have some importance, and in the interest of not removing features that may depend on one another (which would lead to reduced performance) I choose to keep them in. Therefore, ranks 1-10 are to be included in my chosen set of features, and ranks 11-15 are to be removed.

TASK 3

One purpose of this task is to discover whether the performance of two classifiers is statistically significant. The other purpose is to see the effect of removing some attributes on classification. ‘A’ is classifier 1, ‘B’ is classifier 2, ‘1’ is the original dataset, and ‘2’ is the dataset with features removed. The results of A1vsB1 and A2vsB2 are used for discovering if the performance of the classifiers is statistically significant, and the results of A1vsA2 and B1vsB2 are used to assess the effect of removing features from the dataset. Classifier 1 will be Naïve Bayes, and Classifier 2 will be IBk.

T-TEST RESULTS (FOR COMPARING CLASSIFERS)

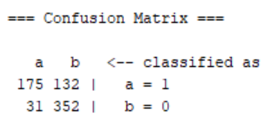
These results were obtained by running WEKA’s experimenter environment, initially loading in the original dataset and then selecting the Naïve Bayes and IBk algorithms. The experiment was then ran and paired-t-tests performed for a range of significance levels. This was then replicated but with the modified dataset loaded instead.

Confidence Level	Original Dataset (% Accuracy)		Modified Dataset (% Accuracy)	
	Naïve Bayes	IBk	Naïve Bayes	IBk
80%	76.57	81.51 v	76.74	80.59 v
85%	76.57	81.51 v	76.74	80.59 v
90%	76.57	81.51 v	76.74	80.59 v
95%	76.57	81.51 v	76.74	80.59 v
99%	76.57	81.51	76.74	80.59
100%	76.57	81.51	76.74	80.59

CLASSIFIER CONFUSION MATRICES (FOR COMPARING DATASETS)

These results were obtained by running the WEKA explorer, loading the respective dataset and then running the respective classifier on it.

Naïve Bayes on Original Dataset

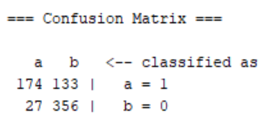


```
=== Confusion Matrix ===

   a   b   <-- classified as
 175 132 |   a = 1
  31 352 |   b = 0
```

Accuracy 76.38%

Naïve Bayes on Modified Dataset

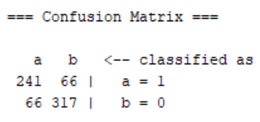


```
=== Confusion Matrix ===

   a   b   <-- classified as
 174 133 |   a = 1
  27 356 |   b = 0
```

Accuracy 76.81%

IBk on Original Dataset

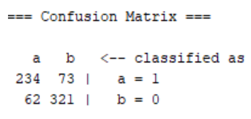


```
=== Confusion Matrix ===

   a   b   <-- classified as
 241  66 |   a = 1
  66 317 |   b = 0
```

Accuracy 80.87%

IBk on Modified Dataset



```
=== Confusion Matrix ===

   a   b   <-- classified as
 234  73 |   a = 1
  62 321 |   b = 0
```

Accuracy 80.43%

DISCUSSION OF RESULTS

1. Comparing Naïve Bayes and IBk on the Original Dataset

The results in the t-test results section above show that, on the original dataset, IBk outperforms Naïve Bayes. IBk classifies 4.94% more instances correctly than Naïve Bayes does. Moreover, the t-test results indicate that, with 95% confidence, IBk is significantly better at classifying than Naïve Bayes is on this dataset. However, when testing with 99% and 100% confidence levels, the t-test cannot say

IBk is significantly better than Naïve Bayes is. Therefore, there is a slight chance that (<5%) that IBk is not significantly better than Naïve Bayes is at classifying on this dataset.

2. Comparing Naïve Bayes and IBk on the Modified Dataset

On the modified dataset, IBk is still the better classifier, with a greater classification accuracy than that of the Naïve Bayes model. Despite the slightly smaller difference between the classification accuracies, the t-test results still show that IBk is significantly better than the Naïve Bayes, with 95% confidence, although, once again, it cannot be any more confident than that.

3. Comparing the Result of Applying Naïve Bayes to the Original and Modified Datasets

The modified dataset yields slightly higher accuracy than the original dataset when classified using Naïve Bayes, as shown in the 'Classifier Confusion Matrix' section above. However, the increased accuracy is small, with only a 0.43% difference. This shows that the removal of the chosen attributes does yield a slight performance increase when used on the Naïve Bayes Classifier, as the removed features had little influence on the ability for the classifier to predict the class variable correctly but had the possibility to confuse the classifier for certain instances, hence the slight performance increase. This can be seen in the confusion matrices, with the modified dataset having fewer False Positives and more True Negatives than the original dataset, showing removing these features allows the classifier to better judge instances where the class value is 0.

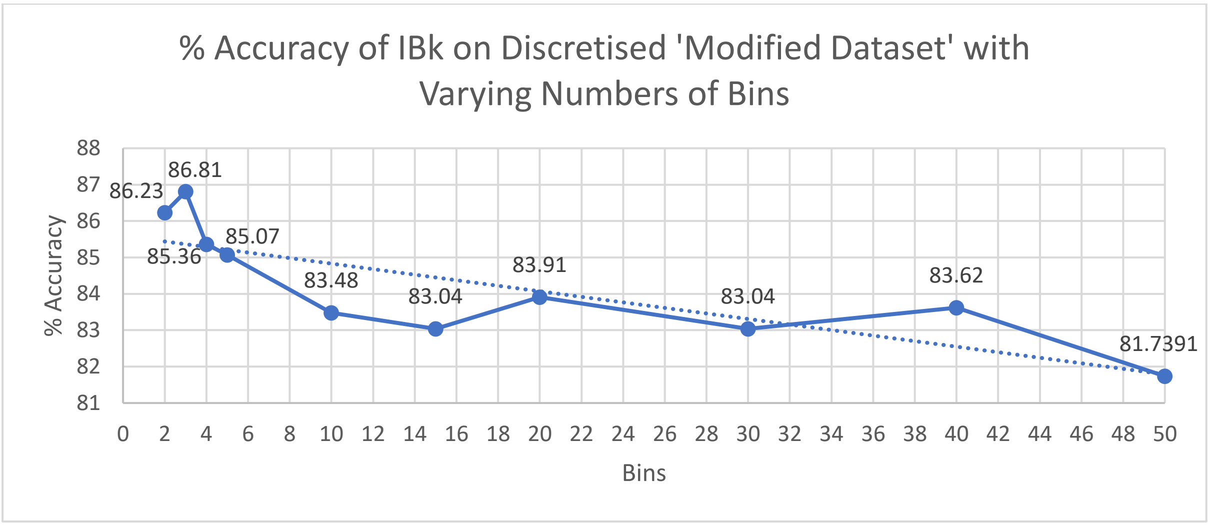
4. Comparing the Result of Applying IBk to the Original and Modified Datasets

The accuracy of classifying the modified dataset using IBk is less than when classifying the original dataset. This difference is slight, only 0.44%, but worse performance all the same. The similar performance would indicate that the removed features were relatively unimportant but looking at the confusion matrix there are fewer True Positives and more False Negatives than when the classifier was applied to the original dataset. There are increases in True Negatives and decreases in False Positives, showing removing the features caused some benefit in the classifiers ability to predict 0 for the class variable, but this came at the cost of worse classification when the class variable is 1. This could be because the removed features had some values which were important for classifying 'border' instances – ones which are near a border between different class labels in the kNN vector space, where removing those features just moves it slightly over the border and so it gets a different (wrong) class label. This difference in how the algorithms work could explain why Naïve Bayes had slightly increased performance and kNN had slightly decreased performance, but with the values remaining similar overall (0.43% increase and 0.44% decrease respectively). Another factor to consider is the small size of the dataset, where these approximate 0.4% changes in accuracy can be caused by a net change in correct classifications of 3.

TASK 4

The unsupervised discretion method Equal Width Binning will be used to create a modified version of the 'Modified Dataset' used in task 3. The result of using the best classifier from Task 3 (IBk) on this discretised dataset will be compared to its performance on the non-discretised dataset in Task 3, and the results shall be discussed, and interpretations made. WEKA's Unsupervised 'Discretise' filter was used to create a dataset of nominal attributes. The IBk classifier was then ran on this 'binned' dataset. This was done for 2, 3, 4, 5, 10, 15, 20, 30, 40, 50 bins to see how the number of bins affected the classification accuracy. The results are plotted below. It should be noted that features which were previously nominal have not had the filter applied to them, as they are still discrete (despite their type now being numeric), as the data only takes a discrete number of values in the continuous range [0, 1]. Creating arbitrarily many bins, could either merge already discrete classes into a lesser number of bins or could create lots of bins where only a proper subset of them contain data. Therefore, only features A3, A8, A11, A14 and A15 will be acted on by the discretisation filter.

RESULTS



DISCUSSION AND ANALYSIS OF RESULTS

The graph shows that the number of bins which provides the best performance is 3, with IBk correctly classifying 86.81% of instances when the modified dataset is discretised to have 3 equal width bins per feature. This outperforms IBk on the non-discretised modified dataset, which had an accuracy of 80.43% (as shown in Task 3). This is a 6.38% increase in performance, which is substantial, and this performance increase was caused just by discretisation into 3 bins. It should be noted that all amounts of bins tested produces an accuracy value greater than that of the classifier on the non-discretised modified dataset. This is enough evidence to say that in this case, binning is certainly better than not binning.

One reason for binning's increased performance over the dataset with continuous attributes could be because all the numeric variables in the modified dataset (A3, A7, A8, A11, A14, A15) have highly skewed distributions (see figure A in the appendix for the distribution of the attributes). Lots of these distributions (visually) appear exponential, where lots of the data is focussed in a small area of the distribution, with the number of instances per unit area of the distribution rapidly trailing off (exponentially) leaving large areas of the distribution very sparsely populated with data. This can be hard for machine learning models to learn, as they tend to prefer more standardised distributions. When the dataset is discretised, the distribution becomes more standard, with less skew. This can allow machine learning models to perform better and could be what leads to increased performance in this case. Another factor that could lead to increased performance on the discretised database is that outliers which cause the distribution to be highly spread have their impact reduced by being placed in bins, as opposed to having a value that is massively different from any other data in the distribution. As the data has outliers like this, this could be another reason for increased performance.

As the number of bins increases, the trend is that the accuracy of prediction decreases. This can be seen by the trendline on the above graph. This could be because, as the number of bins increases, the data's shape transforms back into the exponential shape distribution, with increased skew, and hence is classified less well.

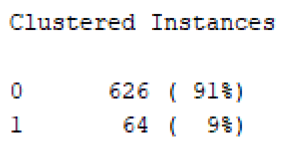
The primary merit of binning is that it can increase classification accuracy and consequently model performance, as discussed above. Moreover, numeric values take longer to process than discretised ones, so using binning can lead to faster models. Moreover, discretised data requires less storage space, a benefit when datasets can be extraordinarily large. Finally, discretised data is easier to visualise and understand than continuous data, as bar charts with a discrete number of values are easier to read and understand than a graph of a continuous distribution.

I would recommend binning when the data has non-standard distributions and outliers, as binning can reduce the effect of these on the generated model leading to greater classifier accuracy. However, I would recommend performing some tests to find the optimal number of bins, as the performance increase of binning compared to the performance of non-discretised data can vary significantly (see graph above).

TASK 5

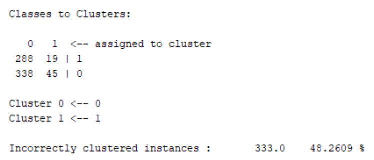
K-means clustering will be used to classify the original dataset. For this task, k will be 2, so there will be two centroids that the instances are clustered around. K = 2 as there are two values the class variable can take, + or – (called 1 or 0 in my dataset). K-means clustering is an unsupervised learning algorithm, that classifies instances based upon their similarity to other instances. The dataset is a supervised learning one, and so is labelled with class information for each instance. This information is irrelevant and will not be considered by the unsupervised k-means clustering algorithm. Consequently, the result of clustering will indicate if a property of the dataset is that similarities between feature values gives enough information for effective classification. This task was carried out using WEKA's SimpleKMeans classifier on the original dataset, with numClusters=2.

RESULTS



```
Clustered Instances

0       626 ( 91%)
1        64 (  9%)
```



```
Classes to Clusters:

   0   1  <-- assigned to cluster
 288  19 | 1
 338  45 | 0

Cluster 0 <-- 0
Cluster 1 <-- 1

Incorrectly clustered instances :       333.0    48.2609 %
```

DISCUSSION

91% of the clustered instances were classified as 0 (-). The ground truth is that only 55% of instances were labelled with -. Therefore, the clustering process is grossly over classifying +'s as –'s. k-means is a distance based algorithm, with the distance being based on the similarity between the features of the instances. What this shows is that the features are not effectively discriminating between the classes and are placing lots of instances labelled with + close to instances labelled -. This can be seen in the figure 2 in the appendix, where no discernible groupings of +'s and –'s are visible. The result of this is that the instances were only classified with an accuracy of 51.74%, which is hardly better than random guessing of classes given this is a binary classification task.

According to the confusion matrix, 338 of the instances assigned to the cluster representing '–' were labelled as '-', with there only being 45 false positives. This is less false positives than IBk in task 3 (and IBk was the best performing classifier in task 3), so k-means clustering shows some promise in this respect. However, the staggering number of false negatives renders the overall accuracy of the k-means classifier as hardly better than random guessing. 41% of instances were classified as false negatives. This shows that the features do not allow for +'s to be discerned and placed in their own area of the k-means vector space. Thus, they end up being incorrectly classified. Only 9% of instances were different enough to be classified as +, and only 2.8% of clustered instances were true positives.

Performance of clustering was worse than that of IBk, a specialised classification technique, with the accuracy of clustering being 51.74% and the accuracy of IBk on the same dataset being significantly greater, at 80.87%. The reason for this is that clustering is an unsupervised learning technique, so it does not learn from comparing its predictions with the actual classes of the instances, and so cannot improve it's performance by training, and instead finds relationships between the instances and classifies using these relationships. Supervised classification techniques, however, can learn from their mistakes through the training process, resulting in a model that is tuned to correctly classify the training data, resulting in increased accuracy when it is used on unseen test data.

Appendix

Figure A – Visualising the dataset (pre-pre-processing).

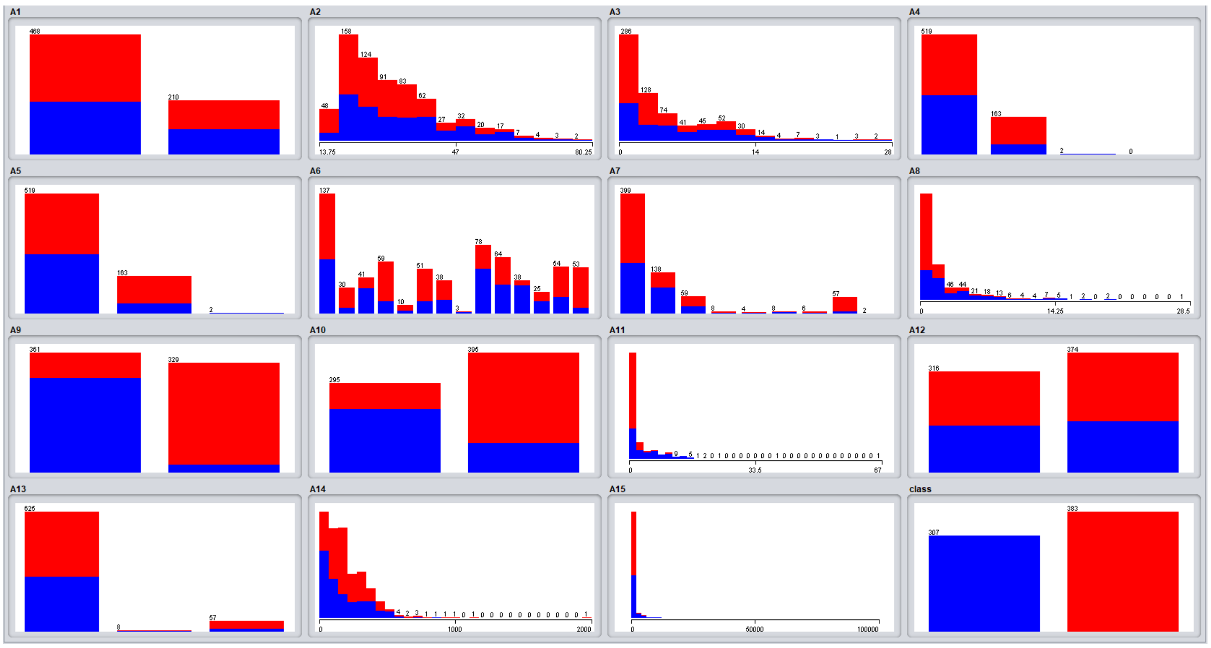


Figure B – Example result of clustering (1 is + and 0 is -).

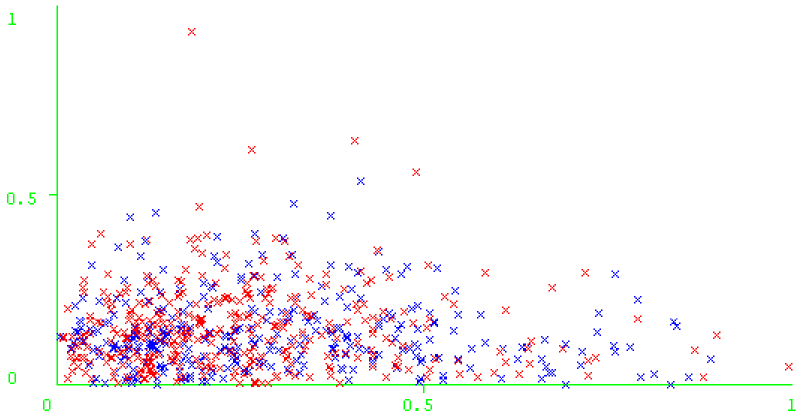


Figure C – Attribute Information Table

Attribute Name	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	class
# Missing Values	12	12	0	6	6	9	9	0	0	0	0	0	0	13	0	0
Data Type	Nom	Num	Num	Nom	Nom	Nom	Nom	Num	Nom	Nom	Num	Nom	Nom	Num	Num	Nom