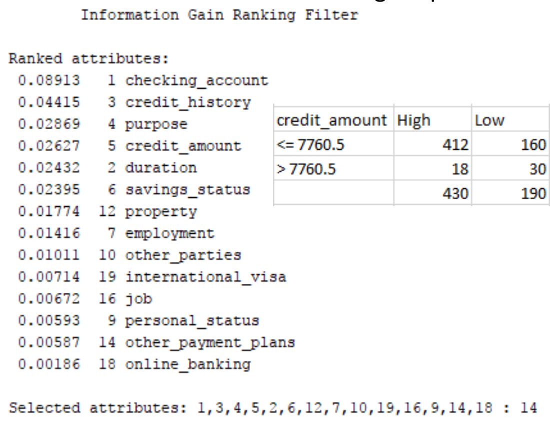
Project 1

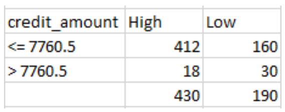
Luke Killoran – 13434418
Machine Learning (Blended Delivery) – COMP47460
Dr. Aonghus Lawlor

Q1. (a) & (b)

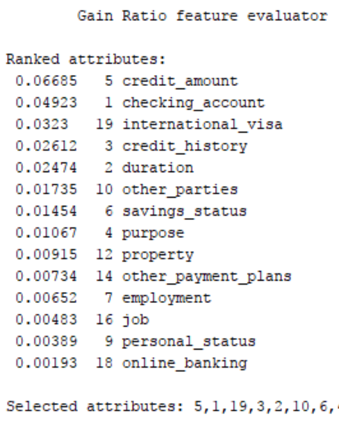
It is important to note that the dataset includes 19 attributes other than the class of interest, 'creditworthiness'. Vitally, there are no missing values among the 620 instances. Note that Weka uses stratified cross validation rather than purely random sampling. However, initially, for illustration purposes, I will display my calculations on the full training set. I chose an Information Gain and subsequently a Gain Ratio filter which resulted in the following output:



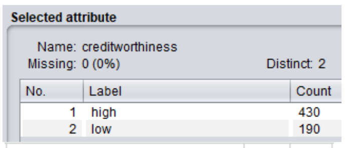
```
        Information Gain Ranking Filter

Ranked attributes:
 0.08913    1 checking_account
 0.04415    3 credit_history
 0.02869    4 purpose
 0.02627    5 credit_amount
 0.02432    2 duration
 0.02395    6 savings_status
 0.01774   12 property
 0.01416    7 employment
 0.01011   10 other_parties
 0.00714   19 international_visa
 0.00672   16 job
 0.00593    9 personal_status
 0.00587   14 other_payment_plans
 0.00186   18 online_banking

Selected attributes: 1,3,4,5,2,6,12,7,10,19,16,9,14,18 : 14
```



credit_amount	High	Low
<= 7760.5	412	160
> 7760.5	18	30
	430	190



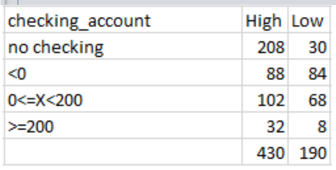
```
        Gain Ratio feature evaluator

Ranked attributes:
 0.06685    5 credit_amount
 0.04923    1 checking_account
 0.0323    19 international_visa
 0.02612    3 credit_history
 0.02474    2 duration
 0.01735   10 other_parties
 0.01454    6 savings_status
 0.01067    4 purpose
 0.00915   12 property
 0.00734   14 other_payment_plans
 0.00652    7 employment
 0.00483   16 job
 0.00389    9 personal_status
 0.00193   18 online_banking

Selected attributes: 5,1,19,3,2,10,6,4,12,14,7,16,9,18 : 14
```



Selected attribute

Name: creditworthiness
Missing: 0 (0%)   Distinct: 2

No.	Label	Count
1	high	430
2	low	190



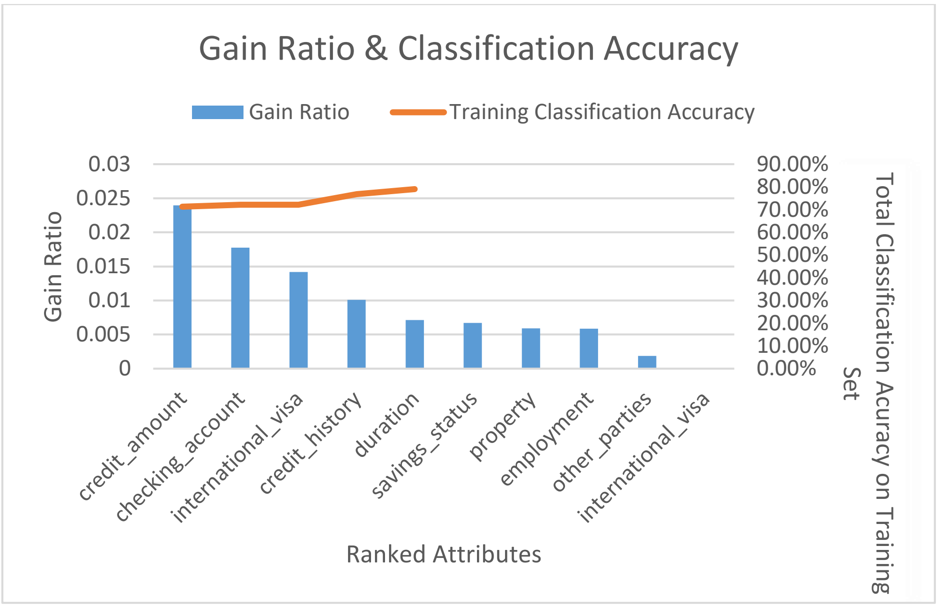
checking_account	High	Low
no checking	208	30
<0	88	84
0<=X<200	102	68
>=200	32	8
	430	190

For each variable Weka would calculate the loss in entropy (measured in bits of information) when only that attribute is discarded. For example, the entropy initially was $-\frac{430}{620}\log_2\frac{430}{620} - \frac{190}{620}\log_2\frac{190}{620} = 0.889$ bits; figures provided by the picture above. The entropy after a full split along the attribute 'checking_account' was $\left(\frac{238}{620}\right)\left(-\frac{208}{238}\log_2\frac{208}{238} - \frac{30}{238}\log_2\frac{30}{238}\right) + \left(\frac{172}{620}\right)\left(-\frac{88}{172}\log_2\frac{88}{172} - \frac{84}{172}\log_2\frac{84}{172}\right) + \left(\frac{170}{620}\right)\left(-\frac{102}{170}\log_2\frac{102}{170} - \frac{68}{170}\log_2\frac{68}{170}\right) + \left(\frac{40}{620}\right)\left(-\frac{32}{40}\log_2\frac{32}{40} - \frac{8}{40}\log_2\frac{8}{40}\right) = 0.7999$ bits. Thus, the Information Gain was $0.889 - 0.7999 = 0.089132$ bits.

The continuous variables could not be split along their values, as that would most likely lead to leaves with only 1 instance – which is extremely undesirable for a classification problem. Therefore, we sort the dataset by that attribute, and we pair each value of that attribute with the subsequent value for which a different class is observed. Then we observe the Information Gain separately for each pair split. We choose the split that maximises the Information Gain. For an example of discretization let's examine the attribute 'credit_amount' – splitting the dataset by the condition of ≤7760.5 and >7760.5 is arrived at by averaging the values 7758 and 7763, and provides an optimal Information Gain of 0.026273 bits, as the entropy after the split is $\left(\frac{572}{620}\right)\left(-\frac{412}{572}\log_2\frac{412}{572} - \frac{160}{572}\log_2\frac{160}{572}\right) + \left(\frac{48}{620}\right)\left(-\frac{18}{48}\log_2\frac{18}{48} - \frac{30}{48}\log_2\frac{30}{48}\right) = 0.862762$ bits.

However, the Information Gain filter is biased toward attributes that can be split along many values. For example if we split the dataset into one leaf nodes, the Information Gain would be maximized, but obviously this is not desirable. Thus, I adopted the Gain Ratio as my filter for feature selection. This penalises each attribute by its number of subsets. For example, the attribute 'checking_account' from before has a Gain Ratio of $\frac{0.089132}{\left(-\frac{238}{620}\log_2\frac{238}{620} - \frac{172}{620}\log_2\frac{172}{620} - \frac{170}{620}\log_2\frac{170}{620} - \frac{40}{620}\log_2\frac{40}{620}\right)} = 0.04923$. Notice the attributes that dropped down the rankings such as 'credit_history', which had 10 levels. Also, note that some stopping mechanism is required to include a certain subset of these variables. I decided to remove the variables with a Gain Ratio equal to 0, thus actually the exact same subset obtained by the Information Gain criterion. However, I could have monitored the total classification accuracy on the training set, after splitting the tree by each consecutive variable. I computed this quantity for this first 5 variables, and plotted the resulting graph. In any case, I maintained the selection method from before.

	Information Gain	Gain Ratio	Training Classification Accuracy
credit_amount	0.02627	0.06685	71.29%
checking_account	0.08913	0.04923	72.10%
international_visa	0.00714	0.0323	72.10%
credit_history	0.04415	0.02612	76.77%
duration	0.02432	0.02474	79.03%



In fact, I did not use the full training set as mentioned previously. The following was indeed the output (by 10-fold CV), however the resulting subsets chosen did not change (I selected the top 14 attributes – removing the final 5).

Information Gain:

```
average merit      average rank  attribute
 0.09  +- 0.005      1    +- 0      1 checking_account
 0.045 +- 0.004      2.1  +- 0.3    3 credit_history
 0.03  +- 0.004      3.6  +- 0.92   4 purpose
 0.028 +- 0.007      4.1  +- 1.14   2 duration
 0.024 +- 0.003      5.4  +- 0.92   6 savings_status
 0.018 +- 0.004      6.7  +- 0.9   12 property
 0.015 +- 0.002      7.3  +- 0.64   7 employment
 0.01  +- 0.002      9.1  +- 0.7   10 other_parties
 0.018 +- 0.015     10    +- 6.97   5 credit_amount
 0.007 +- 0.002     10.8  +- 1.47  16 job
 0.006 +- 0.002     10.9  +- 1.7   14 other_payment_plans
 0.007 +- 0.002     10.9  +- 1.3   19 international_visa
 0.006 +- 0.002     11.4  +- 1.2    9 personal_status
 0.002 +- 0         13.5  +- 0.5   18 online_banking
 0     +- 0         15.5  +- 0.5   17 num_dependents
 0     +- 0         16.5  +- 1.5   11 residence_since
 0     +- 0         16.8  +- 1.08  13 age
 0     +- 0         16.9  +- 1.92   8 installments
 0     +- 0         17.5  +- 1.57  15 existing_credits
```

Gain Ratio:

```
average merit      average rank  attribute
 0.049 +- 0.003      1.6  +- 0.66   1 checking_account
 0.033 +- 0.009      3.1  +- 1.04  19 international_visa
 0.03  +- 0.005      3.4  +- 0.66   2 duration
 0.026 +- 0.002      4.3  +- 1      3 credit_history
 0.018 +- 0.004      5.9  +- 0.7   10 other_parties
 0.015 +- 0.002      6.4  +- 1.02   6 savings_status
 0.011 +- 0.002      7.7  +- 0.64   4 purpose
 0.04  +- 0.034      7.9  +- 8.25   5 credit_amount
 0.009 +- 0.002      8.9  +- 0.83  12 property
 0.008 +- 0.003      9.9  +- 2.12  14 other_payment_plans
 0.007 +- 0.001     10.3  +- 0.64   7 employment
 0.005 +- 0.001     11.8  +- 0.6   16 job
 0.004 +- 0.001     12    +- 1.1    9 personal_status
 0.002 +- 0         13.4  +- 0.49  18 online_banking
 0     +- 0         15.8  +- 1.72  13 age
 0     +- 0         16.3  +- 1     15 existing_credits
 0     +- 0         16.9  +- 1.76  17 num_dependents
 0     +- 0         16.9  +- 0.7   11 residence_since
 0     +- 0         17.5  +- 1.86   8 installments
```

The wrapper method I initially selected was a forward best subset with J48 classifier. I did this to demonstrate the similarity with the filter approach before. Here the wrapper is a hill-climb algorithm selecting the best attributes to branch on at each iteration – starting from the null model and adding one attribute at each iteration. Similarly to before, the chosen decision tree algorithm; J48, selects attributes to branch on using the Information Gain at each node. The J48 algorithm does not use pessimistic pruning like its predecessor (C4.5) where it would post-prune the tree using subtree replacement. (If classifying error was not reduced statistically significantly on a special pruning fold dataset it would not produce that subtree.) Instead I availed of the operation subtree raising where the entire tree was constructed, and various nodes were investigated in a way to remove them and redistribute their instances if there presence was not statistically significant in declining classification error. Also the tree would not branch if one node would be less than 3 instances large. These actions are taken in order to reduce the size of the tree so as to improve the generalisation of the algorithm to unseen test sets. The dataset was split into 10 folds for cross validation using the wrapper. For each iteration, that training set (consisting of 9 folds) was used for 5-fold cross validation to decide on the decision tree format. The output is provided below, detailing the proportion of times an attribute was added to the decision tree. I also implemented backward selection – where for every iteration the tree was built using all the attributes and it was the task of wrapper to work backwards and remove the least significant ones. I ran the backward selection twice to create some separation between attributes.

Forward Selection

No of Folds		attribute	
90%	9	1	checking_account
70%	7	3	credit_history
70%	7	5	credit_amount
60%	6	8	installments
60%	6	11	residence_since
50%	5	19	international_visa
40%	4	10	other_parties
40%	4	17	num_dependents
30%	3	2	duration
30%	3	15	existing_credits
30%	3	18	online_banking
20%	2	13	age
20%	2	14	other_payment_plans
10%	1	6	savings_status
10%	1	12	property
0%	0	4	purpose
0%	0	7	employment
0%	0	9	personal_status
0%	0	16	job

Backward Selection

No of Folds		attribute	
100%	20	1	checking_account
75%	15	3	credit_history
70%	14	17	num_dependents
70%	14	19	international_visa
60%	12	10	other_parties
60%	12	5	credit_amount
50%	10	8	installments
50%	10	6	savings_status
50%	10	11	residence_since
45%	9	18	online_banking
40%	8	2	duration
40%	8	4	purpose
35%	7	7	employment
30%	6	13	age
25%	5	15	existing_credits
20%	4	12	property
20%	4	14	other_payment_plans
15%	3	9	personal_status
15%	3	16	job

Notice that the same variables that ranked the highest in the Information Gain filter (particularly for the standings on the full training set) also are appear in the most forward selection J48 attribute subsets. Also note that 4 attributes never appeared for forward selection unlike backward selection. This is because backward selection is less greedy, and often tests more combinations than forward selection where sometimes less useful variables may be used higher up in the decision tree. For this reason backward subset selection is preferable (especially since we have much more data points than attributes which would be an argument for forward selection). Therefore I used the backward subset selection as my method of choice, and ran the cross validation twice to create dispersion in the attributes in order to make the decision process simpler. I also ran the decision tree on the dataset to understand how many attributes were often used. I found that mainly 8 attributes were used with a variation between 6 – 10. Thus, I chose my set of attributes to be the 10 most occurring attributes throughout cross validation.

	Wrapper
1	checking_account
3	credit_history
17	num_dependents
19	international_visa
10	other_parties
5	credit_amount
8	installments
6	savings_status
11	residence_since
18	online_banking

	Filter
1	checking_account
3	credit_history
19	international_visa
10	other_parties
5	credit_amount
6	savings_status
2	duration
4	purpose
7	employment
12	property
14	other_payment_plans
9	personal_status
16	job

	Not in Either
13	age
15	existing_credits

	In Wrapper but not in Filter
17	num_dependents
8	installments
11	residence_since
18	online_banking

	In Filter but not in Wrapper
2	duration
4	purpose
7	employment
12	property
14	other_payment_plans
9	personal_status
16	job

The reasons why these resulting subsets are different is threefold. Firstly, filter doesn't account for when variables are in conjugation, whereas the wrapper does; especially with backward selection and a decision tree classifier (which by its very format must consider subsets of variables rather than variables independently for eg. when branching on an attribute, its effect is obviously taken into consideration). For example, it could well be that some of the attributes discarded from the wrapper but included in the filter could be because they correlate strongly with the variables already incorporated in the wrapper. Secondly, wrapper only gives credence to variables that do well in the specified classifier (in this case J48). Thus, only attributes that perform well in this decision tree format will be subsequently included in the wrapper subset. Thirdly, usually decision trees favour less variables, and this is even more the case with the minimum of instances per node and sub-tree raising mechanisms. This is in stark contrast to the filters, where the number of attributes included can seem arbitrary as it is in this case.

Q2. (c)
In the same way as before I chose a subset of features using a Naive Bayes classifier inside the best subset wrapper with backward selection. I executed 10-fold cross validation twice to create divisions between the crucial attributes and insignificant ones. Since most runs returned a subset with 13-14 attributes, I chose the top 14 variables ranked by proportion of times incorporated into the model.

Naïve Bayes Backward Selection			
No of Folds		Attribute	
100%	20	1	checking_account
100%	20	2	duration
100%	20	3	credit_history
95%	19	4	purpose
90%	18	6	savings_status
85%	17	19	international_visa
80%	16	5	credit_amount
80%	16	12	property
75%	15	7	employment
75%	15	9	personal_status
75%	15	18	online_banking
65%	13	8	installments
65%	13	16	job
55%	11	10	other_parties
30%	6	13	age
25%	5	11	residence_since
25%	5	15	existing_credits
25%	5	17	num_dependents
10%	2	14	other_payment_plans

Wrapper	
1	checking_account
2	duration
3	credit_history
4	purpose
6	savings_status
19	international_visa
5	credit_amount
12	property
7	employment
9	personal_status
18	online_banking
8	installments
16	job
10	other_parties

Filter	
1	checking_account
3	credit_history
19	international_visa
10	other_parties
5	credit_amount
6	savings_status
2	duration
4	purpose
7	employment
12	property
14	other_payment_plans
9	personal_status
16	job

Not in Either	
13	age
11	residence_since
15	existing_credits
17	num_dependents

In Wrapper but not in Filter	
8	installments

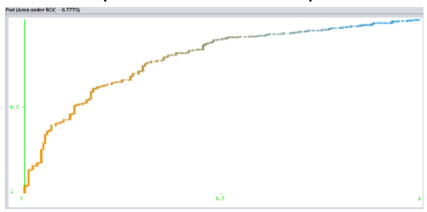
In Filter but not in Wrapper	
14	other_payment_plans

I eliminated the exiled attributes for each feature selection method, and ran all 4 with 10-fold cross validation 10 times.

	Classification %				No Classified as Low even though High			
	Filter		Wrapper		Filter		Wrapper	
Run No	NB	J48	NB	J48	NB	J48	NB	J48
1	75.4839	71.6129	75.6452	74.1935	101	111	102	114
2	75.1613	70.1613	75	76.2903	102	124	104	116
3	73.2258	71.6129	74.8387	73.0645	109	121	105	122
4	74.6774	70.9677	75.3226	73.871	103	117	102	118
5	75.6452	70.9677	74.6774	74.5161	100	128	106	117
6	75.1613	70.1613	75.3226	74.8387	101	122	104	119
7	75.3226	71.129	75.8065	75.4839	103	119	100	115
8	75.3226	71.4516	75.4839	75.4839	99	117	101	113
9	75.6452	71.2903	75.8065	73.871	99	131	101	115
10	74.5161	70.6452	75.4839	74.5161	106	119	103	117
mean	75.01614	70.99999	75.33873	74.6129	102.3	120.9	102.8	116.6
sample std dev	0.731081	0.536548	0.391058	0.947215	3.164034	5.762908	1.932184	2.633122

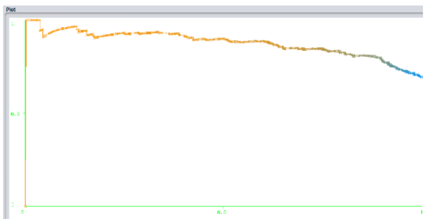
With all Variables:		
Random	Baseline	OneR
Classification %		
50	69.3548	66.9355
No Classified as Low even though High		
60	190	139

ROC Curve for NB Wrapper
(TP Rate vs FP Rate)



	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	BAR	BER	Class
J48 Wrapper	0.893	0.6	0.771	0.893	0.828	0.69	0.647	0.354	high
	0.4	0.107	0.623	0.4	0.487	0.69	0.647	0.354	low
	0.742	**0.449**	**0.726**	**0.742**	**0.723**	**0.69**	**0.647**	**0.354**	(weighted average)
NB Wrapper	0.888	0.5	0.801	0.888	0.842	0.778	0.694	0.306	high
	0.5	0.112	0.664	0.5	0.571	0.778	0.694	0.306	low
	0.769	**0.381**	**0.759**	**0.769**	**0.759**	**0.778**	**0.694**	**0.306**	(weighted average)
J48 Filter	0.849	0.584	0.767	0.849	0.806	0.653	0.633	0.368	high
	0.416	0.151	0.549	0.416	0.473	0.653	0.633	0.368	low
	0.716	**0.452**	**0.7**	**0.716**	**0.704**	**0.653**	**0.632**	**0.368**	(weighted average)
NB Filter	0.881	0.532	0.79	0.881	0.833	0.775	0.675	0.326	high
	0.468	0.119	0.636	0.468	0.539	0.775	0.675	0.326	low
	0.755	**0.405**	**0.742**	**0.755**	**0.743**	**0.775**	**0.675**	**0.325**	(weighted average)

Precision vs Recall
for NB Wrapper



Methods		T-test stats	p-value
NB Wrapper	NB Filter	1.490711414	0.08511372
J48 Wrapper	J48 Filter	8.687140691	0.00000570
NB Wrapper	J48 Wrapper	2.320422797	0.02272717
NB Filter	J48 Filter	13.10520048	0.00000018

I conducted a one-sided paired t-test (assuming paired samples since the resulting cross validation datasets were identical across all models for each run due to utilising random seeds) of the difference of the mean cross validation classification proportions. This would help in understanding whether one models' classification accuracy was statistically significantly greater than the other at the 5% level of significance. These tests were performed using the following test statistic: $t = (\bar{x}_D - 0)/\sqrt{\frac{s_D^2}{n_1}}$ (with a degrees of freedom of n - 1 = 9). I also could have conducted a Mc Nemar's test for one run of each model on a test set. This test performs well in assessing the significance of any differences between two paired proportions. As for the t-test method, the decision tree performs much better with the wrapper than without. This is expected as a decision tree can be unstable with more attributes provided. The Naive Bayes wrapper records a higher classification accuracy than its filter alter ego. This difference, however, is not statistically significant at the 5% level of significance. The Naive Bayes wrapper is consistently slightly better than the J48 wrapper. Perhaps this is because Naive Bayes assumptions are more appropriate – that all attributes are roughly independent and equally contribute to the class response. Whereas the J48 algorithm seeks greedy optimal splits in the dataset and thus, may tend to work best when not all attributes contribute equally to the class response. The filtered Naive Bayes is substantially better than the filtered edition of the J48. This again is evidence of the robustness of Naive Bayes with respect to extra irrelevant attributes, in contrast to the J48 algorithm. Note that it is pleasing to see that all algorithms improve over random choosing, choosing the most common (baseline) class ('high'), and OneR (a decision tree split along only one attribute, usually due to Information Gain or Gain Ratio). However, it is worrying that the best classification accuracy of our models only improve on the baseline by about 6 percentage points considering the increase in variance of predictions undertook by implementing an algorithm. The poor predictive power of the dataset can arguably be witnessed to a certain extent by observing the decline in classification accuracy from the baseline to the OneR algorithm. Overall, the most appropriate algorithm for this dataset must be Naive Bayes among the methods we've tried. It displays consistently better results which is evident in the t-tests, and bolsters its case when considering the extra statistics provided for one cross-validation run of the 4 models: It has the largest true positive rate or sensitivity or recall (proportion of high risk individuals classified correctly) which is of utmost important here since it would be financially disastrous for the bank to treat many high risk individuals as low risk, the smallest false positive rate (proportion of low risk individuals classified incorrectly), the greatest precision (proportion of individuals classed as high risk to be correctly classified), the highest F-measure (harmonic mean of precision and recall to provide a balanced estimate of accuracy), the largest balance accuracy rate and least balance error rate which are both useful measures in this case since the dataset is skewed toward high risk individuals, and finally the greatest area under the ROC Curve (measuring balanced performance along the true positive rate and false positive rate). The ROC Curve for the Naive Bayes wrapper stretches close to the top left corner of the graph, and clearly does better than random guessing. Therefore, in all the metrics the Naive Bayes wrapper seems to dominate, and this is possibly down to two reasons. One, none of the attributes correlate more than 25% with the response so this is concerning for a decision tree algorithm. Two, a wrapper method should always perform slightly better than a filter method given the same classification method – even though Naive Bayes is robust to extra irrelevant (though uncorrelated) variables, it would still improve using the wrapper as the wrapper considers the subsets that maximise the predictive power under the NB algorithm. However, it is true that the NB wrapper does not statistically significantly outperform the NB filter in its classification accuracy. Although I must make explicit the somewhat arbitrary nature of our filter selection for the number of attributes in the final model. For example, if we selected 10 attributes rather than 14, the classification accuracy using filtered Naive Bayes would have declined significantly (to a mean of around 72.5%). Though this is a problem for which many upgrades already exist and it should be important to be cite the dangers of overfitting when availing of a wrapper. Overall, the decided algorithm is still the Naive Bayes wrapper. There are other considerations before moving forward.

Perhaps there may be a particular cost to classifying a high risk individual as low risk. This must be considered, and a penalty could be added to give preference to predicting high risk rather than low risk. Moreover, there are many ways to improve both algorithms such as boosting for instance, for the J48 algorithm, where the tree is run on the residuals to improve its pitfalls. Also, random forests could introduce before unconsidered subsets of attributes where trees are branched on some random attribute. This increases the diversity and usually improves the performance as a result. As well as that, the dataset is quite small, so it might be possible to run a near exhaustive search to try an immense amount of combinations. If we collected more data this might substantially improve the performance of J48, as decision trees are often very dependent on the sample size. We also could have easily chosen a different filter such as the Gini Index or Correlation with the Class Response. We could have deduced a more scientific method for deciding the number of ranked attributes to incorporate into the final model too. Moreover, maybe it might be prudent to consider embedding methods which combine the advantages of both filters and wrappers. Here, a learning algorithm takes advantage of its own variable selection process and performs feature selection and classification simultaneously.

Q2. (a)

Typically the arrangement is as follows: Training Set (50% of the data), Validation Set (20% of the data), Test Set (30% of the data). That is, taking a datset where the response is known for all instances, we randomly divide it into segments in the proportions given above. However, these allocations can depend on the domain, and specifically on the size of the dataset. If the dataset is very small, the training set will inevitably need to grow to provide any semblance of association from the attributes. The training set is the subset of examples used for learning – i.e. comparing Naive Bayes, J48, Logisitic Regression and kNN algorithms. The validation set is the subset of examples used to tune the classifier. For example it could be to select the parameter values such as deciding on the optimal degree of polynomial to be used in logistic regression, or whether to prune certain subtrees of a decision tree. Then the final model is tested on the aptly named test set which is comprised of examples used only to assess the evaluation of a fully-trained classifier. This avoids a bias in evaluation of the model, where reusing examples from the validation set could lead to underestimates of the real error rate. Ideally the dataset would be split into k folds, with one fold as the test set and the other k-1 folds as the training and validation set combined. The training and validation set should be split, and the model training, tuning and evaluation process should be executed k times. This should provide more robust estimates of model performance on unseen data, and thus its generalisation capabilities. Running this process multiple times should be successful in reducing overfitting on random noise. In this case, t-tests and McNemar's test (which could investigate the proportion of wins and losses) can examine the statistical significance of one model's performance over the others.

Q2. (b)

Typically cross validation involves splitting the data into k folds randomly. However, this approach assumes that there is no relationship between the observations; that each observation is independent. This is not true of time series data, where the time dimension of observations means that we cannot randomly split them into groups due to their autocorrelation – the correlation between various lags of the data points. Instead, we must split data up and respect the temporal order in which values were observed. As well as that, the time series can often exhibit non-stationarity which can lead to widely different model outcomes in standard cross validation. We have three alternatives to cross validation in order to 'backtest' our time series model. The first is a simple train-test split (also denoted an in-sample, out-of-sample split) where the dataset maintains its order and is divided in two whereby all the training sample instances have a time element which is further in the past than all of the test sample examples. For example, given dataset with observations at time 0 through time 9: $x_0, x_1, x_2, \ldots, x_8, x_9$, the training set could be $x_0, x_1, x_2, \ldots, x_5$ and the test set $x_6, \ldots, x_9$ for a 60%-40% split. The second method is a multiple train-test that splits the dataset in many places but maintains the time aspect by constraining every training set to only include the instances that precede the test set instances in time. For example, given our dataset from before, and 4 splits, the training sets could be (x_0, x_1), (x_0, x_1, x_2, x_3), $(x_0, x_1, x_2, x_3, x_4, x_5)$, $(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ and the respective test sets could be (x_2, x_3), (x_4, x_5), (x_6, x_7), (x_8, x_9). This method reduces the chances of overfitting and fitting random noise in comparison with the previous algorithm. It also improves accuracy as it allows the model to take more recent instances into account. The third option is Walk Forward Validation. After sorting the data with respect to time, we select the minimum number of instances required to train the model. We also decide on whether we want the training set to grow as time increases (expanding window) or maintain the same size (sliding window). Then we train the model on the initial instances; the amount of which to use is the minimum of instances chosen before. Our test set is only the very next instance that is not in the training set. We evaluate the model on this instance, before we include it in our training set and test the expanded test set on the proceeding instance. If we were utilising a sliding window approach we would drop the first example of the training set each time. For example, given the minimum number of observations required is decidedly 4, and an expanded window approach, the training sets would be (x_0, x_1, x_2, x_3), $(x_0, x_1, x_2, x_3, x_4)$, $(x_0, x_1, x_2, x_3, x_4, x_5)$, ... , $(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$, and the test sets would be (x_4), (x_5), (x_6), ... , (x_9). The estimates of accuracy of the models evaluated is much more robust this way than our second option. However, it comes at a computational cost but this is not usually a problematic issue if the models are simple, and the dataset not massive. Although, careful attention needs to be paid to the window width and window type. These could be adjusted to contrive a test harness on your problem that is significantly less computationally expensive. In most cases, this would be the recommended approach in place of standard cross validation.

Q2. (c)

Naive Bayes is naive as it assumes that each of the attributes included the model are independent and contribute equally to the response class. We wish to evaluate $p(c_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|c_i)p(c_i)}{p(\boldsymbol{x})}$, the probability of the given instance taking the class i given its attribute values. Note that $\boldsymbol{x}$ is a vector of attribute values for the given instance. Thus, $\boldsymbol{x} = (x_1, x_2, \dots, x_d)$ for d attributes. Now, Naive Bayes assumes, conditioning on the given instance being a member of class i, that the probability distributions of the various attributes are independent of each other. Therefore $p(\boldsymbol{x}|c_i) = p(x_1|c_i)p(x_2|c_i) \cdots p(x_d|c_i)$ and this assumption is made for all classes of the response. This assumption leads the initial probability calculation to be easily attained up to a proportionality constant. This assumption is deemed naive as it is almost never true since, in reality, many variables interact with others and exhibit correlation. For example, if the class of interest was whether one was classed as high or low risk by a bank. Then given the individual was low risk, the probability distribution of their checking account balance would be independent of their employment status or place of residence, even though these attributes can obviously be correlated. However, this model usually performs quite well even when its assumptions are violated.

Q2. (d)

Firstly, it could simply be that the test set is fundamentally very different from the training set or that there has been errors in the sample collection. However, in any case, the training error is often not a reliable indication for test error. This is because the training set instances are subject to random noise, and fitting to this variation too closely drags the performance of the model down when it comes to evaluating further examples. Also, since the training set is a sample of the population, and as such it is quite possible that some instance combinations have not been observed yet. Note that it would be quite possible to achieve a training error of 0 using a decision tree (provided no instances have the exact same attribute values but different response classes). However, it is obvious that this tree is useless in predicting unseen data. Thus, it could be the case that the model is overfitted – either by fitting noise or by including too many parameters. The remedy for this issue is supplying more robust tuning and testing methods and preferring simpler models (in accordance with Occam's Razor) to optimise the accuracy-complexity trade-off. One option is to avail of is cross validation which splits the dataset randomly into k-folds to test the model k times on unseen datasets, as well as the training set being modified correspondingly at each iteration. This improves the estimates on accuracy and forces the model not to fit so closely to the training set as the learning examples do not remain the same. A model that performs well on cross validation should be a more robust model that generalises well to new data, given that the data was representative of the population. This test will usually provide a great indication of model test performance on unseen instances. Another option is techniques that introduce a penalty for complexity or smoothness constraint (like regularization or lasso in regression; that modifies the objective function to include a term accounting for the size of the slope coefficients so that when the equation is minimised, smaller coefficients are preferable (and in fact, for lasso, many are shrunk to 0, thus, discarded from the model)). Another thing to consider would be to split the dataset into training, test and validation sets, so that the model does not get tuned on the full training set, and thus, does not overfit as much. Another alternative is to do feature selection to reduce the complexity of the model. This could be in the form of a filter that ranks the variables by their association with the response, and discards the lesser ranked ones. This is also mean a wrapper selection method that evaluates the performance of various subsets of attributes in predicting the response through the use of a chosen classifier. Either way, the number of attributes can be substantially reduced, simultaneously forcing a simpler model that generalises better as a result. Another alternative is to conduct statistical tests that look to reduce the complexity of the model. This could be as simple as pruning on a decision tree, whether subtree raising or subtree replacement etc. where a subtree's instances could be redistributed or branched along a different attribute if the current subtree does not statistically significantly increase classification accuracy on some pruning test set. The next option could be to introduce more randomness into the model's creation, and hence consider a greater number combinations of subsets of the attributes to achieve an optimal model for the test set. For example, random forests (where randomly attributes are branched on) usually improve upon standard decision tree algorithms like J48. The reasoning is the same as why backward subset selection often provides an incline in classification accuracy over the greedy forward subset selection. There are boundless options at our disposal. improving the model could be as simple as trying out different models on test sets and using more than one way to decide on the final model.