Name: Shaunak Oke
Student No.: 18204307

Question 1)

All these results were performed with cross validation with 10 folds.

J48 classifier

J48 is a simple decision tree based classifier. Depending on entropy it splits the initial data till we obtain pure results. It involves pruning of unnecessary nodes.

results:

Correctly Classified Instances	325	90.2778 %
Incorrectly Classified Instances	35	9.7222 %
Total Number of Instances	360	

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 0.832    0.060    0.881      0.832   0.856      0.783    0.876     0.829     bad
                 0.940    0.168    0.913      0.940   0.927      0.783    0.876     0.878     good
Weighted Avg.    0.903    0.130    0.902      0.903   0.902      0.783    0.876     0.861
```

The results show that the classifier is mostly accurate. With proper selection of attributes this accuracy can be further increased. Depending on the applications' requirements the precision and recall are quite similar as f measure is close to 1.

KNN classifier results:

Knn classifier is a simple nearest neighbor classifier. It simply checks K no. of closest neighbors for some random node.

Correctly Classified Instances	323	89.7222 %
Incorrectly Classified Instances	37	10.2778 %
Total Number of Instances	360	

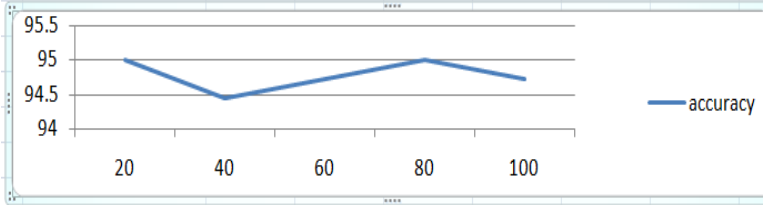
```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 0.728    0.013    0.968      0.728   0.831      0.775    0.873     0.813     bad
                 0.987    0.272    0.872      0.987   0.926      0.775    0.873     0.883     good
Weighted Avg.    0.897    0.182    0.905      0.897   0.893      0.775    0.873     0.859
```

KNN was slightly less accurate as compared to the J48 and the results are quite similar. Knn is only marginally worse than j48.

Using ensembles

J48: I got the best accuracy for ensemble size of 20 and 80 classifiers. (95%) I chose 20 size for faster results.

Bagging					
J48					
accuracy	95	94.4444	94.72222	95	94.7222
No. of members	20	40	60	80	100

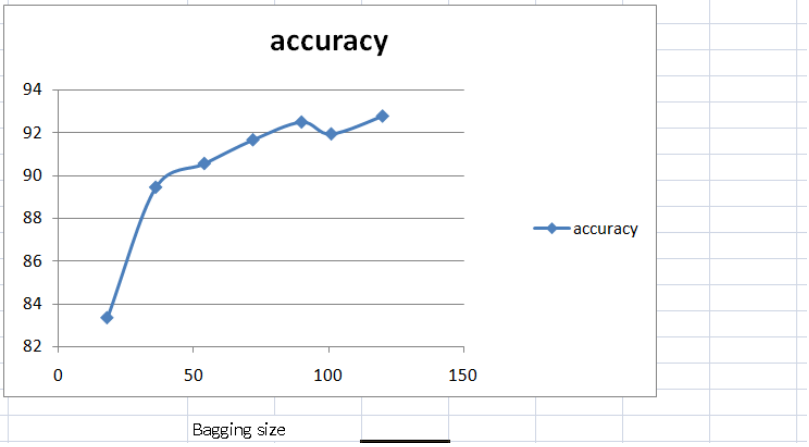


The performance stayed relatively the same for increasing size. For higher values than 100 of ensemble size the accuracy showed a slowly decreasing trend. This is because the performance

plateaus as more members don't add any benefit to the voting after a certain point.
N.B. Since Weka doesn't support exact fractions of percentages(5.5% of 360 is 20 members). I have rounded the percentage of members it actually allows like 5% yields 18 members 10% yields 36 members. And so on.

Altering the bagging size

accuracy	83.3333	89.4444	90.5555	91.6666	92.5	91.9444	92.7778
bagging size	18	36	54	72	90	101	120



As expected the accuracy rapidly increases with increase in bagging size initially. But this increase quickly plateaus as adding more samples from training set doesn't affect the classifiers a lot. As expected ensembles are good for J48 as it is sensitive to changes in data.

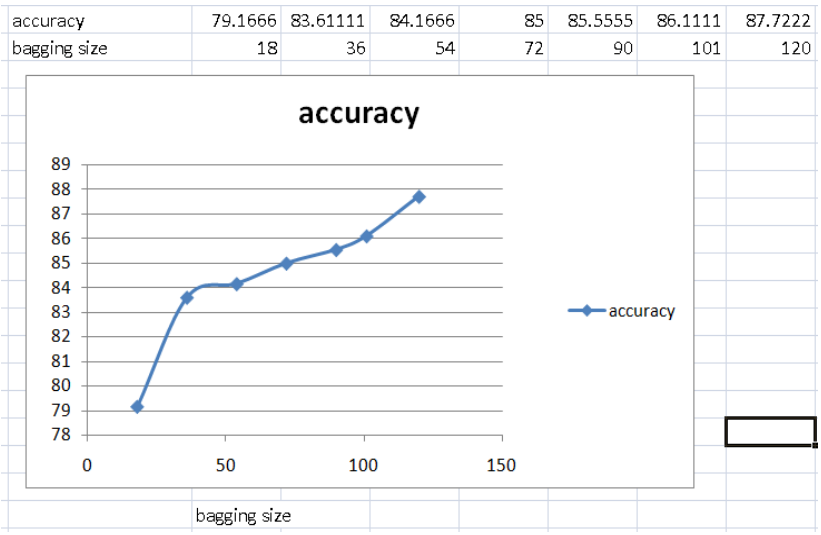
Overall there is significant improvement over the single j48 classifier of around 5%.
This is because j48 relies on entropy. Entropy will vary a lot depending on the points chosen for the dataset. Removing /adding different points can change the decision tree generated. In such cases ensembles are quite useful.

KNN:
There is minimal change in the accuracy of knn as a classifier for bagging ensemble technique.

accuracy	89.7222	89.4444	89.44444	89.7222	89.7222
No. of members	20	40	60	80	100

Selecting no. of members as 20 and varying bagging size

accuracy	79.1666	83.61111	84.1666	85	85.5555	86.1111	87.7222
bagging size	18	36	54	72	90	101	120



This shows the same trend as before. The accuracy increases just on the basis of extra new data members being available for the classifiers

Overall the accuracy of ensembles of knn barely reaches the accuracy of single classifier.(89.972). Using ensemble in this case actually increased the time taken to run the model and decreased accuracy. This is because knn is not sensitive to changes. The way it is built selecting different combination of points won't affect the class labels assigned to them as distance of nearest neighbor is used to determining the class instead of entropy/information gain. Removing/adding points is very unlikely to change the knn results. This makes using ensemble on knn worse than just a single knn classifier.

Random Subspacing

J48:

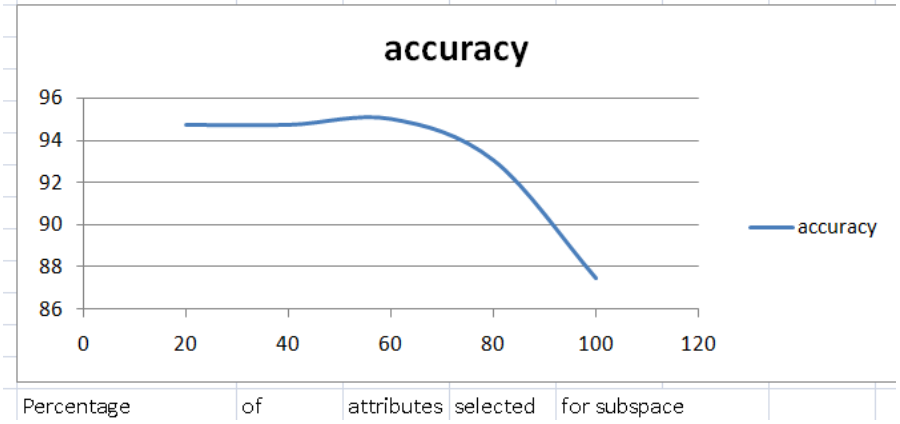
There is negligible change in accuracy when the ensemble size is changed.

accuracy	94.166	95	94.4444	95	94.7222
No. of members	20	40	60	80	100

I think this is because J48 is good at pruning bad features and hence shows good accuracy for any number of classifiers as j48 will properly prune the provided subset features. Also as the accuracy of model gets closer to 100% the improvements obtained also slow down.

I selected 40 ensemble size as it was small and among the options had the best accuracy.

accuracy	94.7222	94.7222	95	93.0556	87.5
Percentage of attribut	20	40	60	80	100
Selected					



The accuracy does not increase a lot from the default parameters. As you get closer to 100% of the subspace the accuracy starts decreasing dramatically.

I believe that this is because the size of the subspaces determine the random features selected. As you get closer to 100 the model simply starts selecting values from the entire training database randomly hence adding too much similar data for all the classifiers defeating the point of using ensembles in the first place.

This means all classifiers get more similar inputs and the diversity is reduced.

KNN:

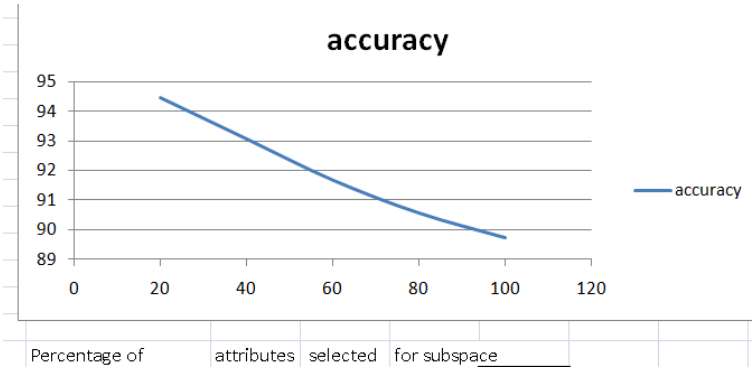
Changing the ensemble size

accuracy	92.222	93.333	93.035	95	93.0556
No. of members	20	40	60	80	100

The accuracy has significantly increased in this method as compared to increase in j48. For classifier size of 80, random subspace with knn surprisingly manages to get the same highest results as j48.

I think there is a significant change in knn because of the way subspaces are generated. It results in more diversity for the knn. There may be many attributes that have same clustered value but are of no significance. Knn may allot these attributes depending on the bad attributes instead of checking the useful features that actually determine class more accurately. Selecting features randomly will hence in a sense spread the features so that knn is likelier to assign classes more accurately.

I chose ensemble size 80 due to best accuracy to alter subspace size.



This shows similar trend as before. As the subspace % increases the diversity between different classifiers decreases. We start working on more similar data and the accuracy becomes closer to that of the single classifier.

Question 2

a)Sampling in bagging.

Bagging involves using the same classifier on the data. If the data is same as the training data we won't get any different results as classifiers are same and all will vote for the same answer regardless if its correct or not. To fix this, we generate random samples from the data, each classifier now gets different sample of data of some defined size. This increases the diversity of samples among the different classifiers. Generating samples in this way also helps automatically create suitable representatives of all the training set. This will reduce the bias of the single classifier on all the data. This makes all the classifiers choices better than a single classifier and hence bagging is successfully achieved.

b)Stochastic Gradient descent and Mini batch Gradient Descent

Stochastic gradient descent: It involves doing gradient descent for a single training example at a time. Since we move using random sample at a time, the algorithm does not move towards the local minimum steadily. Instead the cost function varies randomly although it decreases as we go on.

Mini batch Gradient Descent: it involves selecting a small batch of examples. Gradient descent is now performed batch-wise. It is a compromise between batch gradient and stochastic gradient descent methods. It further improves the performance substantially over stochastic gradient descent as we are performing on a batch of examples and this process can be parallelized. We

need to pick the batch size carefully however. If it is too large we lose the benefits of parallelization and get back to gradient descent. If it is too small however there is no benefit as we are simply doing stochastic descent and lose the benefits of faster batch processing. We can use mini batch gradient descent when there are too many features to be parallelized. In this case mini batch will significantly improve the performance over stochastic gradient descent, it will also reach the optimal value faster due to this.

c)Bias terms
Bias term is used to shift the input to output mapping of a neuron. By changing the weights we can only change the steepness of the curve. Without bias the activation function will give same output at the initial inputs of 0. This might not be desirable for our network. To resolve this we need an extra input with the same value but variable bias for the corresponding neuron. Varying this bias can effectively move the curve along the input domain and help the network adjust and set the activation curve better. Bias inputs are not affected by the previous layer results. Hence essentially you can set the initial trainable constant value of each neuron. This decides how hard it is for the neuron to send a signal. If we want to shift the curve left we use a negative bias. Doing this makes it harder to send a signal. If we want to shift the curve right we use a positive bias. This makes the activation function trigger more easily. Doing this is very hard by simply assigning weights for each node without a bias. For initial values like 0 we will never be able to change the results as all inputs are zero without having a bias.

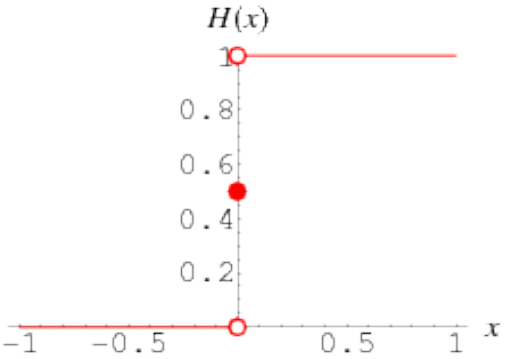
d) Hidden Layers.
Hidden layers are useful to solve complex problems. Single Perceptron has a limitation that it can only solve linearly separable problems. To solve non-linearly separable problems we need hidden layers to make more functions to decide the actual results of the network. Each hidden layer acts like a hyperplane that can further split the data and decide which portion of the space belongs to which class. Most real world problems are non-linearly separable. Linear separable problems simply test if the results are below or above some threshold. A problem can consist of identifying many different parameters. For example a image recognition system to interpret road signs will have hidden layers that all do different/simple tasks. The first hidden layer will identify the different lines/curves of the image. The second layer will try and find the relevant pixels of roughly the same contrast/hue. The third layer may try and interpret the colors in the obtained area to determine the class of the sign like (Information sign, traffic sign, construction sign), The fourth layer may be tasked with finding the curves/ lines inside the sign, then the fifth layer might be tasked to find the actual meaning of the sign. Such a complex system is impossible to program with no hidden layers because we will have to generate and devise too many input nodes manually and still setting the threshold will be a tough task for the network for so many input nodes and no layers. This highlights the importance of hidden layers.

e)Bagging vs Boosting
Bagging ensemble technique involves sampling the data randomly with replacement. Each sample is given to the different classifiers. The classifiers parallelly work on their sample and produce some classification. Each classifier is assigned the same weight which leads to simple voting on the results.

Boosting on the other hand involves random sampling only for the first classifier. The classifiers

work serially on the results of the previous classifier. The results classified wrong by a classifier are assigned a higher weightage. The next classifiers input does not involve random sampling. Instead there is more chance that the wrongly classified data occurs more frequently in the input. The different classifiers themselves are also assigned a corresponding weight. This weight is assigned based on the classification results for the respective classifiers. This ensures that the better classifications are assigned more weightage during voting. Hence the voting is done using weightage average of all the results instead of the simple average used by bagging. Boosting however is susceptible to overfitting.
Source

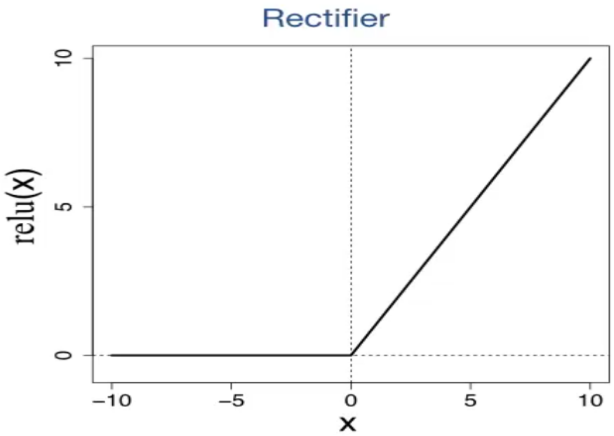
f) Activation functions
Unit Step: This is the simplest activation function. It's given by



H(x)={0 x<0
1 x>0
0.5 x=0}(graph obtained from wolfram)

This function is the simplest activation function. It is particularly useful for binary classification when we need only two results of the possible values. We can set the value for x=0 where there is discontinuity in the function.

Rectified Linear Unit:



relu(x) = max(0, x)

This is another simple activation function that works fast and is computationally less expensive. Hence it is quite popular. This is useful when we want sparser activation pattern where no neurons fire below certain threshold. It however has linear characteristics which might not be ideal depending on the problem.