CAB320 - Artificial Intelligence

Assignment 2 Report

Jordan Hawkes - N7521022

Stewart Whitehead - N7561555

Preparing the data:

The goals of the project is to test how effectively several different classifiers can predict whether a tumor is benign or malignant based on multiple parameters provided in a dataset.

The first thing that was done was to create the function to prepare the dataset. All items in the second column of the dataset are stored in an initial numpy array. This array is then looped through and all items in the array that are ‘M’ are converted to a 1 and stored in an new numpy array called Y. All items in the array that are ‘B’ are converted to 0’s and stored in the Y numpy array. The complete initial file is then stored in another numpy array called X. However, the first and second column of this array are dropped. This is because the first column only contains the id number of the tumors, which have no relevance to whether the tumor is benign or malignant. The second column is dropped because this is the ‘M’ and ‘B’ values which have been stored in the Y numpy array and converted to 1’s and 0’s.

After the numpy arrays have been created the dataset is split into testing and training sets. The training set is used to create the classifier and the testing set is used to evaluate the effectiveness of the classifier.

Naive Bayes classifier:

Naive Bayes is a classifier that applies bayes theorem while assuming independence between all variables in a dataset. The existence of one variable does not add or take away from the significance of any other variable in the dataset, even if in reality there is a relationship. If the NB conditional independence assumption is true for our inputs, a Naive Bayes classifier will converge quicker than many other classifiers. This results in Naïve Bayes not requiring as large a dataset.

The Naive Bayes classifier had roughly the same success rate with a large training set as it had with a small training set. It has a success rate of approximately 95 per cent with a training set size of 455 and 92 per cent success rate with a training set size of 12. This classifier also achieved a 95 percent accuracy when evaluating the training set it had seen previously. This suggests that the Naive Bayes classifier does not need a great amount of data in order to be quite successful but is not as successful as other algorithms at evaluating data it has seen previously.

Decision tree Classifier:

The decision tree classifier looks for pure subsets. A subset is pure if a particular value of a variable always produces the same result. If the algorithm finds an instance of a pure subset, it will stop and return that result. If the algorithm does not find instances of a single variable that are pure, it will combine variables to see if a combination of parameters is pure. If the variables in a dataset are continuous, it will classify based on thresholds it sets such as a

value being greater than or less than a particular number. The larger the number of values in a dataset that create the pure subset, the greater the chances that the classifier will return the correct answer. This algorithm performs better on large datasets.

The decision tree classifier had a high success rate with a large training set, averaging a 94 percent success rate with a training set size of 455. However, it was not as successful as the Naive Bayes classifier when it was given a small training set of 12, as it had a success rate of only 76 per cent. This is because a larger dataset will reduce the likelihood of the algorithm finding a pure subset and will keep searching until it finds a pure subset that combines a larger number of variables which is therefore more likely to be correct. The mean accuracy of the algorithm when tested on validation sets was approximately 92 percent, indicating that the result returned from the test set is accurate and not overfitted

We experimented with altering the maximum depth to which the search could run. Unsurprisingly the greatest accuracy was achieved when the maximum depth was higher, however, even at a maximum depth of 2 the algorithm performed admirably with an accuracy of 85.08 percent. This is an admirable result considering the accuracy when the maximum depth is not restricted is 93.85 percent. The commented sections of the build_DT_classifier() function can be uncommented to repeat these tests.

Nearest Neighbour Classifier:

The nearest neighbour classifier performed similarly to the decision tree classifier. It had a 93 per cent success rate with a large training set of 455, but only an 80 per cent success rate with a training set of 12. This classifier did however have a 100 percent success rate when evaluating data it had seen previously. To summarize this classifier, it is very effective when it has been given a large training set or when evaluating data it has seen previously. The mean accuracy of the algorithm when tested on validation sets was approximately 93 percent, indicating that the result returned from the test set is accurate and not overfitted.

When constructing our nearest neighbour classifier, we used the ‘K Neighbours Classifier’ and set our *k=1*. We chose this classifier over ‘Nearest Neighbours’ as we later find and use the optimum value for *k* for our test cases. To find the optimum value for *k*, we placed the classifier inside of a loop incrementing the value of *k* by 2. The *k-value* is kept as an odd number to avoid instances where the nearest neighbour would achieve a tie. After iterating 1, 3, 5, …, 17, 19, the most accurate value is returned using the most effective *k-value*.

Support Vector Machine Classifier:

The support vector machine classifier performed well when classifying the testing data. It was able to achieve a 94.73 percent success rate with a large training set of 455 and 86 percent with a small training set of 12. It was 100 percent accurate when evaluating the training set, showing that this classifier can be useful when used on data it has seen previously.

The support vector machine classifier plots each data item in n-dimensional space (depending on the number of features of the data), then proceeds to find the hyperplane boundary which splits the two classes. Due to this margin placed between the two classes, the support vector machine classifier allows for better generalisations of data. The radial basis function (rbf) kernel was the kernel parameter used in our classifier initially. This kernel did not perform very well with the dataset we trained and tested it with - this is likely due to the large number of features given to it (each feature adds an additional dimension) with a small dataset. This kernel should perform well provided there was an ample amount of training data.

Other common kernels considered in the classifier were linear, poly and sigmoid. Since rbf performed less than optimally with only a 63% success rate in its predictions, we created a loop to test the other kernel types, choose the best one and report that back. While testing these other kernels, we found that sigmoid performed as well as rbf with only 64% average success rate. Linear on the other hand performed quite admirably in comparison to the others with an average success rate of almost 95%. The mean accuracy of the linear classifier when tested on validation sets was approximately 95 percent, indicating that the result returned from the test set is accurate and not overfitted. No results were achieved through poly as the kernel seemed to run endlessly. There is a possibility that editing other parameters may have allowed a poly kernel to return a result, however that was beyond the scope of this experiment.

Results

Algorithm	Training Accuracy	Testing Accuracy Train Set= 455 Test Set =114	Testing Accuracy Train set= 12 Test set=557	Cross Validation Accuracy
NB	0.947	0.9385	0.9211	
DT	1.0	0.9385	0.7580	0.9211
NN	0.9428	0.9261	0.8189	0.9298
SVM	1.0	0.9473	0.8655	0.9543