Austin Prince

CSE514

Homework 1

**Logistic Regression**

**Implementation**

For logistic regression I used the scikit learn package and pandas DataFrame in order to construct my model. The code I used for this implementation is as follows.

X\_test = test\_data[features]  
y\_real = test\_data['TIME']  
  
LogReg = LogisticRegression()  
LogReg.fit(X, y)  
  
log\_predict = LogReg.predict(X\_test)  
logrecacc = accuracy\_score(y\_real, log\_predict)

**Results**

*Predicted: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]*

*Accuracy Score: 0.8*

*Actual Values: [0, 1, 0, 0, 1, 0, 0, 0, 0, 0]*

**Discussion**

Logistic regression calculated our test examples with an 80 percent accuracy. This may be due to the small size of our training set. Because of this and the fact that logistic regression is an active learner, meaning that it constructs our model before applying it to our test set, it will not work well on outliers, or values in our test set that are not represented accurately by our training set. When looking through our test set it was revealed that there is at least one outlier there which will probably be misclassified by most types of active learners.

**kNN**

**Implementation**

For kNN I built my own functions that constructed my estimates for kNN. Psuedocode for kNN is below.

**def kNN**(k, test\_set, train\_set):  
 **for** datapoint **in** test\_set:  
 find train\_points **in** train\_set that are closest to datapoint  
 classify train\_points  
 clasify datapoint based on majority vote of train\_points  
 **if** train\_points **is** tied:  
 classify datapoint randomly  
 **return** predictions **for** test\_set

**Results**

*kNN n=1*

*Predicted: [0, 0, 0, 0, 1, 1, 0, 0, 0, 0]*

*Accuracy Score: 0.8*

*Actual Values: [0, 1, 0, 0, 1, 0, 0, 0, 0, 0]*

*kNN n=3*

*Predicted: [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]*

*Accuracy Score: 0.9*

*Actual Values: [0, 1, 0, 0, 1, 0, 0, 0, 0, 0]*

*kNN n=5*

*Predicted: [0, 1, 0, 0, 1, 0, 0, 0, 0, 0]*

*Accuracy Score: 1.0*

*Actual Values: [0, 1, 0, 0, 1, 0, 0, 0, 0, 0]*

**Discussion**

kNN works well as we increase k in this testing example. kNN is a lazy learner meaning that there is no model constructed before-hand on the training set that is then applied to the test set. kNN simply goes through the test set and looks at the example in the training set that match these and then makes a prediction based on this. The reason kNN may work better in this example could be partly because our train set and test set is so small in this case. kNN works best as we increase our k because if our data point is an outlier then as we increase k it is more likely that it will handle classifying outliers correctly because of the majority vote. This is due to the fact that as we increase k there are more likely to be examples within the vicinity of said outlier that will classify it correctly.

kNN works well as we increase k in general, in this case when k=5 our majority vote is completely correct. This would not necessarily always be the case, in some cases it could be that a different value of k would be better (generally not 1).

In addition to this we have a very small dataset so our optimal value of k will not be that big, because as we expand k we will start to test data points that are outside of our given classification set. We want a k that is small enough to stay within the bounds of the classification set (ie all the cluster of all examples that are classified the same) and not big enough so that we are simply taking the majority vote of the whole dataset.

**Decision Tree**

**Implementation**

**for** attribute a:  
 **for** all splits **in** attribute a:  
 find infogain of split on attribute a **with** split y1<x<y2 # for continuous variable with values from y1 to y2  
 split on attribute a at split x that gives max infogain  
 create a nodes **from** the split x **with** attribute a  
repeat until all samples split **and** classified into singular classifications # every sample in train\_set has been classified

**Results**

*Decision Tree*

*Predicted: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]*

*Accuracy Score: 0.8*

*Actual Values: [0, 1, 0, 0, 1, 0, 0, 0, 0, 0]*

*Decision Tree with pruning*

*Predicted: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]*

*Accuracy Score: 0.8*

*Actual Values: [0, 1, 0, 0, 1, 0, 0, 0, 0, 0]*

**Discussion**

For this decision tree model we constructed using the C4.5 model. This is because we want to overcome the bias the ID3 has towards attributes with a large number of values. Our samples will be biased

Using Pythons graphviz library I was able to visualize my tree both before and after pruning

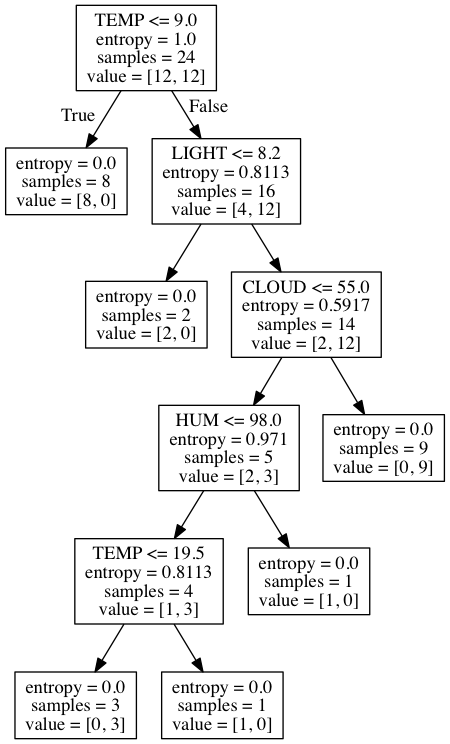
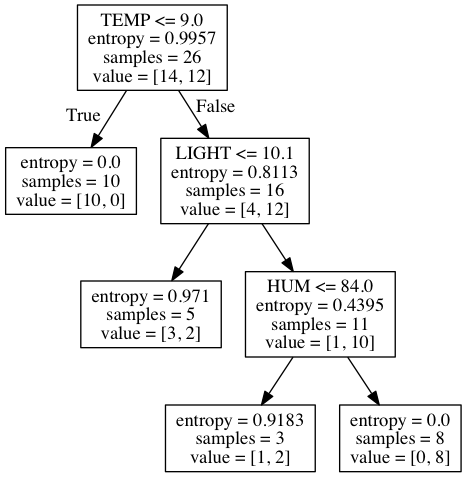


Figure : Decision Tree with Pruning

Figure : Decision Tree without Pruning

The above samples show how the decision tree with pruning will generate less splits and thus be less likely to overclassify our samples.

One possible reason that our Decision Tree did not classify our data correctly would be because of sample bias. This data is in chronological order based on the day of the year. For this reason, it may be possible that our sample is biased towards a certain season. For example specific patterns emerge in the winter season that our train\_sample is taken from and in the spring there are different weather patterns that emerge that may negatively impact the performance of our test set.

Another reason that I determined for why Decision Tree misclassified one of my examples was because temperature was determined to be the most important feature. However, in our test set one of our points is an outlier for temperature (ie low temperature but positive classification), it would make sense that this point would be misclassified.

**Random Forest**

**Implementation:** The pseudocode for Random Forest is below.

**for** i **in** range(0: num\_trees):  
 select a random sample containing .8 percent of train\_Set  
 train a decision tree based on this sample  
 append this tree to tree\_list  
**for** datapoint **in** test\_data:  
 run datapoint through tree\_list to classify  
 append classifications **return from** every tree **in** tree\_list to predict list  
 take mode of classifications to predicut datapoint

**Results**

*Predicted: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]*

*Accuracy Score: 0.8*

*Actual Values: [0, 1, 0, 0, 1, 0, 0, 0, 0, 0]*

**Discussion**

Random Forest will construct decision trees based on randomly sampling 80 percent of our train data to construct a model. It will then run our test set through all of these trees seeing how each tree classifies the test example. Then it will classify this point based on the mean of these classifications. Because one of our test points is an outlier with regards to temperature which is the feature that has the most information gain (and will most likely have the most information gain for all trees constructed), it makes sense that this example will be misclassified even using an ensemble method such as random forest.