Assignment 6

December 14, 2015

Abhishek Mehra and Mridul Birla

1 Implementation of KNN

KNN or K nearest neighbors is a concept which basically works after comparing any test sample with all the training data available. To find which class it belongs to, we choose the best among k selected nearest neighbors to it. The concept seems pretty easy and simple. But when we execute it, we realize there is a lot more to it.

1.1 Code Flow

- Initially the training data is read and stored in form of a list, with all feature vectors. So, the whole training data is a list of list for us with each list corresponding to each attribute of the training data.
- Then for each test data, we compute it's euclidean distance from each training data (all attributes) and select the nearest k attributes.
- Out of these selected k attributes, the most common class label is selected and assigned to the test data under consideration
- Lastly, we calculate the accuracy by checking the total number of correct observations over the total number of test cases.

The initial code runs fine but takes a lot of time for execution. The average time for execution now is 35 - 40 minutes as it compares each test data to each of the training data example. Also, below is the distribution of accuracy for different values of k (Table 1). Using weka, we checked multiple cross-validation results on the test data provided to us, and tuned the parameter k to a value with maximum accuracy and minimum variance. We select the in range of the most optimal value and achieve around 71% accuracy.

1.2 Improvements

We weren't happy with the execution time of the code, so we tried a couple of options to reduce the execution time.

1. Tried using other heuristics to calculate distance between neighbors. Distances which we tried were Euclidean, absolute Distance or Manhattan Distance and finally a dot product product value. The best results were obtained with Euclidean.
2. We then thought of reducing the number of vector features which each instance has. So, we tried converting the RGB values to single luminescence value. Thus, now the comparison only happens for 64 dimensional vector, thus increasing the speed 5 fold. Now the code execution finishes in 8-10 minutes approximately.
3. Finally, we have also normalized the data so that each value is between 0 and 1. We tried using both normalized and not normalized data and checked the accuracy. There was not much of a difference in accuracy.

Now the question is whether this is a fair trade. The accuracy reduces by around 5 percent but the time also decreases 5 fold. I think it depends on the user application. Maybe somewhere it would be helpful to have faster results and the modified algorithm would work better. But in some scenarios, the accuracy would have prime importance. Thus, we have submitted both the files here. You can run the files by implementing **knn** for basic KNN with runtime around 30 minutes. The other can be run by implementing **knn_modified** which gives a runtime of 8-10 minutes but accuracy of around 65%

Luminance/Intensity = **0.2989*red + 0.5870*green + 0.1140*blue**

k	Accuracy
11	approx 69.41
51	approx 70.88
71	approx 70.98
101	approx 70.68
151	approx 70.21

Table 1: k vs Accuracy