**K Nearest Neighbors Algorithm**

**Abstract**

The K-nn algorithm requires a set of stored records. By calculating distance between records based on distance metric, classify the unknown records based on the nearest k number of neighbors. And it is a lazy learner which does not need to generate a model to classify unknown records. Each time we only need to compute distance to other training records and use class labels of nearest neighbors to determine its label (by taking majority vote). As a result, in each classification phase, all records need to be considered.

**Algorithm**

1. Enter the number k which decides how many neighbors need to be take into account and transfer the nominal feature in to binary fashion. (eg, if only two results, then change the first one to (0,1), the second to (1,0))

2. **for** each fold **do**

3.Compute the distance between the test records and every other example.

4.Choose K closest training examplesand then use majority votes to determine the label. for unknown records.

1. **end for**
2. Compute the mean of accuracy, precision, recall and F1 value from these ten folds

**Implementation**

In my implementations, I defined two functions to help me normalize the data and calculate the accuracy and other measurements.

I defined the first function called norm\_data(matrix), which takes the original records as the only arguments. It will normalize each feature by method. Thus, all feature values would be in range (0,1). The reason why we do this is because the distance computed from one feature may dominate all other features.

Then I defined the second function called accu\_cal(truth, result), which take the ground truth of test labels and predicted labels to generate all four measurements.

During the preprocessing phase, for convenience, all nominal features are transferred to binary format so that all values from that feature has equal distance.

A linalg.norm() function was used to calculate Euclidean distance. I created four lists for accuracy and other measurements. For each fold, I add these measurements to the list. The final value for these measurements are the mean value of the results from 10 folds.

**Results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Folds | Accuracy | Precision | Recall | F1 |
| Dataset1 | 10 | 0.9033 | 0.8364 | 0.8989 | 0.8653 |
| Dataset2 | 10 | 0.6125 | 0.4560 | 0.6493 | 0.5272 |

I used k=2 for both datasets. The knn algorithm has a better result on dataset 1 because of the large feature dimensions. More features and records can increase the accuracy of knn method. However, the running time on dataset1 is approximately half a minute. The running time for knn algorithm on large dataset is a big trouble. A larger k value may cause misclassifications.

**Pros**

1. Simple mechanism.
2. Time saving compared to other complex classification methods
3. Outliers are not sensitive for Knn algorithm

**Cons**

1. Feature values needs to be normalized to avoid converging to one dominant feature
2. Large amount of data may cause high memory occupation and plenty of calculations
3. The result is sensitive to k value

**Kaggle Test**

Abstract

Our main idea is to adopt adaptive boost method to construct a strong classifier which help us classify test data. Actually, the weak classifiers we used are the combination of base classification methods. In our case, Support Vector Machine, Random Forest, Decision Tree and Logistic Regression are used in our adaboost model. Thus, the strong classifier is the combination of these four base classifiers.

Algorithm

1. Input the training dataset and number of learning rounds 4
2. Initialize the weight distribution w = 1/m (m is the number of test data)

2. **for** t = 1, … 4

3.Train the model by one of the given 4 base models h(x) which has not been used. before

4.Calculate the error

5. Determine the weight for the chosen classifier =

6. Update the weights

5. **end for**

1. Output H(x) = sign()

Analysis

When we started to tackle this Kaggle test, the adaboost model first come to our mind. In the first try, we only use knn as our weak classifier. In each boost phase, record the k value which causes the least error for each weak classifier. The learning rounds T is set to 20, which means the strong classifier is constructed by 20 weak knn classifiers with their best k value. The result of this adaboost model based on knn algorithm is 81.4%. More classifiers won’t improve the performance dramatically. Hence, we switch our mind to make combinations of different base classification methods. The final result of our current method is approximately 85%. The result is increased significantly by roughly 4 percent.

In the future, on the one hand, the algorithm can be improved by adding more base classification algorithms. On the other hand, we can construct several strong classifiers based on different base classification methods and apply majority voting to increase the accuracy of the prediction.