HW8
Machine Leaning

21600004
Kang Seok-Un

Introduction

In this assignment, I also use the MNIST dataset, a well-known public dataset, as in the past assignments. In addition, dimension reduction is performed using PCA to reduce data from the existing 784-dimension to 2-dimension and 7-dimension. In other words, 2-dim, 7-dim, and 784-dim (raw image) are used.

At this time, KNN and Random Forest are used as algorithms used to implement Classification. In the implementation, KNN was implemented by my own, and in the case of Random Forest, Public Library was used.

Experiment

Data Preparing

KNN Model train should be performed for this assignment. However, KNN performs a distance calculation process in which time complexity is $O(n^2)$, where n is the number of the training dataset Therefore, if all of the existing 50,000 train datasets are used, it takes too much time, so a total of 10,000 train datasets, 1,000 per label, were used. The reason I picked 1,000 for each label is to prevent the bias of train data.

As shown in Figure 1, 1,000 data were randomly selected for each label. The same process was also performed on test data.

Table 1: # of each dataset samples

# of Train dataset	# of Test Dataset
10,000	1,000

```
108     train_x, train_y = train_set
109     test_x, test_y = test_set
110
111     filtered_train_x = np.empty([0, 784])
112     filtered_train_y = np.array([], dtype=int)
113
114     filtered_test_x = np.empty([0, 784])
115     filtered_test_y = np.array([], dtype=int)
116
117     for idx in range(10):
118         x_mask = np.where(train_y == idx)
119         x = train_x[x_mask]
120         y = train_y[x_mask]
121
122         rd_idx = [ idx for idx in range(0, x.shape[0])]
123         rd_idx = rd.sample(rd_idx, 1000)
124
125         x = x[rd_idx]
126         y = y[rd_idx]
127
128
129         filtered_train_x = np.append(filtered_train_x, x, axis = 0)
130         filtered_train_y = np.append(filtered_train_y, y, axis = 0)
```

Figure 1. Randomly Choosing the Data Samples

KNN

First of all, I thought it would be $O(3n^2)$ to obtain distance for each K test dataset for the same dimension of the training dataset. Therefore, we tried to recycle by storing the distance of test data and train data in "distance_Map" for specific dimensions, such as lines 66 to 80 of Figure 2. Through this, the time of the existing O(3N2) was shortened to O(N2). Next, in the case of "KNN.predict(), the distance information stored in "KNN.fit()" was used to select the most labels by selecting K data earby.

.

```
62  class MyKNN:
63      def __init__(self):
64          self.k = 1
65
66      def fit(self, train_x, train_y, test_x):
67          N_train, d_train = train_x.shape
68          N_test, d_test = test_x.shape
69
70          self.train_y = train_y
71
72          print("N_train: {0}, d_train: {1}".format(N_train, d_train))
73          print("N_test : {0}, d_test : {1}".format(N_test, d_test))
74
75          # calculate distance
76          self.distance_Map = np.zeros((N_test, N_train))
77
78          for i in range(0, N_test):
79              for j in range(0, N_train):
80                  self.distance_Map[i][j] = np.sqrt(np.sum((train_x[j] - test_x[i])**2))
81
82      def predict(self, test_y, K):
83          predicted = []
84
85          for idx in range(0, test_y.shape[0]):
86              near_distance = np.argsort(self.distance_Map[idx])
87              vote = [0 for x in range(10)]
88
89              for k in range(K):
90                  vote[self.train_y[near_distance][k]] += 1
91
92              predicted.append(np.argmax(vote))
93
94          return predicted
```

Figure 2. The Implementation of KNN

Random Forest

In the case of implementing Random Forest, the length of the code is not long because the public library was used. Looking at Figure 3, it is expected that it will be easy to understand how random forest was written.

In addition, I tried to make a difference in execution speed and performance by configuring “n_estimators” different among parameters used in the random forest model, such as 10, 100, and 500.

```
202          rf = RandomForestClassifier(n_estimators=estimator)
203          rf.fit(pca_dim_train, filtered_train_y)
204
205          y_pred_dt = rf.predict(pca_dim_test)
```

Figure 3. The Implementation of Random Forest

Result

Table 2 shows the performance results of KNN according to K and Dimension. The larger the K, the better the Accuracy is, but it is difficult to say that it has a great influence. On the other hand, the higher the Dimension, the better the accuracy is compared to K.

In the case of K, the difference in running time is not significant. However, it can be seen that the higher the Dimension, the longer the running time.

Table 2. KNN: Accuracy and running time depending on K and dimension

	K = 1		K = 5		K = 10	
	Accuracy	Time(s)	Accuracy	Time(s)	Accuracy	Time(s)
Dimension = 2	0.343	60.047	0.392	60.047	0.397	60.047
Dimension = 7	0.78	60.658	0.836	60.658	0.848	60.658
Dimension = 784	0.956	81.073	0.952	81.073	0.946	81.073

Table 3 shows the performance results of Random Forest according to "n_estimators" and dimension. In the case of "n_estimators", it is difficult to say that has a great influence on accuracy. But it has a huge influence on running time.However, like KNN, the higher the Dimension, the higher the accuracy.

Table 3: Random Forest: Accuracy and running time depending on n_estimators and dimension

	n_estimators=10		n_estimators=100		n_estimators=500	
	Accuracy	Time(s)	Accuracy	Time(s)	Accuracy	Time(s)
Dimension = 2	0.350	0.141	0.352	1.174	0.350	5.799
Dimension = 7	0.755	0.225	0.788	1.610	0.790	8.118
Dimension = 784	0.836	1.908	0.909	18.274	0.920	91.104

When comparing the accuracy and running time in Table 2 and Table 3, accuracy was generally high in KNN and Random Forest was overwhelmingly fast in running time. Therefore, when have to use two models, it seems better to use KNN if prioritize accuracy and Random Forest if prioritize speed.

Conclusion

In the case of KNN, which is a Lazy model, implementation was very simple and comfortable while performing this assignment. However, it took a very long time to evaluate the performance, so it is difficult to evaluate it as a very good model. This is because the random forest is about 40 times faster, but KNN's performance was not 40 times better.

Therefore, I think I would use random forest or use another model instead of KNN.