Applied Machine Learning HW1

September 2017

1 Programming Exercise

1. Digit Recognizer

   (a) We download the data and load the data into python as data frame.

```
import numpy
data = numpy.loadtxt('/Users/yingzhu/Desktop/train.csv', delimiter=',', skiprows=1)
```

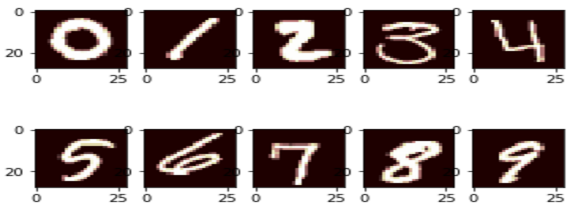
```
print(data.shape)
print(data[0].size)
```

```
(42000, 785)
785
```

   (b) The following is the display of MNIST digit

```
import matplotlib.pyplot as plt
label = data[:,0]
reshape=[]
fig = plt.figure()
for i in range(10):
    for j in range(len(data)):
        if(label[j]==i):
            tar = data[j,:][1:]
            reshape.append(tar.reshape(28,28))
            break;
```

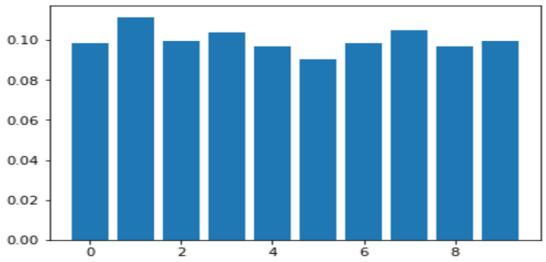
The following is the displayed MNIST digit



(c) Examine the prior probability of the classes in the training data

The prior probability is almost uniform across the digits.

```
cnt_record=[0]*10
for i in range(10):
    for j in range(len(data)):
        if(label[j]==i):
            cnt_record[i]=cnt_record[i]+1
    cnt_record[i]=cnt_record[i]/len(data)
plt.bar([i for i in range(0,10)],cnt_record)
plt.show()
```



(d) Pick one example of each digit from your training data. Then, for each sample digit, compute and show the best match

We use the first appearance of each digit sample as our target.

```
from scipy.spatial.distance import cdist

sample_digit =[]
for i in range(10):
    for j in range(len(data)):
        if(label[j]==i):
            sample_digit.append(j)
            break
```

Figure 1:

To calculate the euclidean distance between two digit, we defined a

function called distance between. The result shows that all most all digit could be correctly predicted by the nearest distance sample except digit 3. Digit 3 is mistakenly classified as 5.

```
def distances_between(i,j):
    temp=data[i,1:]-data[j,1:]
    return (temp*temp).sum()
```

```
knn1_index = []
knn1_label=[]
for i in sample_digit:
    min_index = i+1
    min = distances_between(i+1,i)
    for j in range(len(data)):
            if(distances_between(j,i)<min and j!=i):
                min_index= j
                min = distances_between(i,j)
    knn1_index.append(min_index)
    knn1_label.append(label[min_index])
```

```
print(knn1_index)
print(knn1_label)
```

```
[12950, 29704, 9536, 8981, 14787, 30073, 16240, 15275, 32586, 35742]
[0.0, 1.0, 2.0, 5.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

Figure 2:

(e) Plot histograms of the genuine and impostor distances on the same set of axes

We collect all the samples of digit 0 and digit 1 and apply the cdist function to calculate pairwise distance over the digit 0 and digit 1 sample. We plot the histograms of the genuine and imposter distances on the same graph. As shown in the following histogram, there exists small overlap between two distributions.

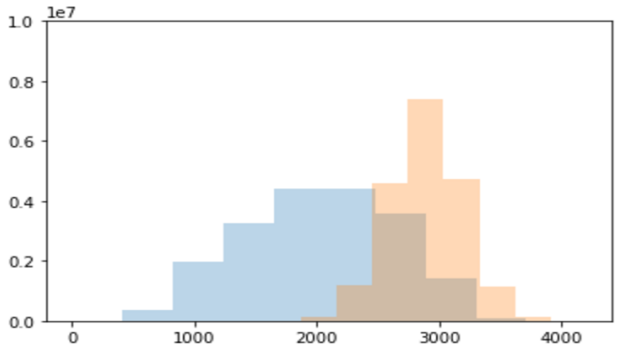


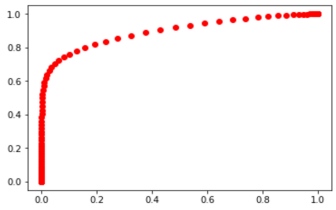
Figure 3:

(f) Generate an ROC curve from the above sets of distances.

The equal error rate is around 0.817. If we guess randomly, the error rate should be 0.5.

```
TP = []
FP = []
genuine_dist = label00_dist+label11_dist
impostor_dist = label10_dist
for cutoff in range(200,4000,40):
        tpnew = len([x for x in genuine_dist if x < cutoff])/float(len(genuine_dist))
        fpnew = len([x for x in impostor_dist if x < cutoff])/float(len(impostor_dist))
        if abs(tpnew-(-fpnew+1))<=0.01:
            print(tpnew,fpnew)
        TP.append(tpnew)
        FP.append(fpnew)
plt.plot(FP,TP,'ro')
plt.show()
```

```
0.8178076851638107 0.19198603430929623
```



(g) Implement a K-NN classifier.

```
# 1(g) K-NN classifier
def knn_classifier(x_train,y_train,x_test,k):
    dist = cdist(x_test,x_train)
    sort_index = dist.argsort()[0:len(x_test),0:k]
    y_hat=[]
    for i in range(len(x_test)):
        level_hat=[]
        for j in sort_index[i]:
            level_hat.append(y_train[j])
        ylevel_hat = max(set(level_hat), key=level_hat.count)
        y_hat.append(ylevel_hat)
    return y_hat
```

(h) Using the training data for all digits, perform 3 fold cross-validation on your K-NN classifier and report your average accuracy.
By setting k=5 and using KFold function we run 3 fold cross validation on our train set by knn classifier. The average score got by cross validation is listed below.

```
res = evaluate(data[:,1:],data[:,0])
print(res)
```

```
[0.966, 0.9645714285714285, 0.9660714285714286]
0.965547619047619
```

(i) Confusion matrix
After observing the confusion matrix, we found that digit 4,5,8 are the tricky ones. Digit 4 is easy to be classified as 9. Digit 5 is easy to mistakenly classified as 3. Also digit 8 could easily classified as 1 or 3.

```
# confudion matrix
from sklearn.metrics import confusion_matrix
X= data[:,1:]
Y= data[:,0]
x_train,x_test,y_train,y_test = cross_validation.train_test_split(X,Y)
y_pred=knn_classifier(x_train,y_train,x_test,5)
y_true = y_test
confusion_matrix(y_true, y_pred, labels=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
array([[1011,    0,    0,    0,    0,    2,    1,    0,    0,    0],
       [   0, 1160,    3,    0,    0,    0,    1,    2,    0,    1],
       [  11,   13, 1008,    2,    0,    0,    4,   26,    3,    1],
       [   2,    5,    6, 1015,    0,    9,    1,    2,    7,    2],
       [   1,    7,    0,    0,  985,    0,    2,    0,    0,   33],
       [   6,    2,    0,   22,    2,  934,   13,    1,    1,   11],
       [   6,    1,    0,    0,    0,    2, 1027,    0,    1,    0],
       [   0,   10,    4,    0,    2,    0,    0, 1083,    0,    8],
       [   3,   12,    4,   14,    2,   14,    5,    1,  956,   15],
       [   6,    0,    2,    7,   13,    3,    0,    8,    5,  968]])
```

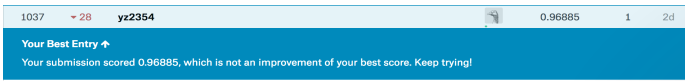
(j) Train your classifier with all of the training data, and test your classifier with the test data. Submit your results to Kaggle.

```
# train classifer with all of the training data and test classifer with test data
import numpy
test_data = numpy.loadtxt('/Users/yingzhu/Desktop/test.csv', delimiter=',', skiprows=1)
```

```
# train the classifer with all of the trainning data
import pandas as pd
y_hat_test = knn_classifier(X,Y,test_data,5)
```

```
y_hat_res=pd.DataFrame(y_hat_test)
y_hat_res.to_csv('/Users/yingzhu/Desktop/test_res.csv', index=False, header=False)
```

We run our model on the test data and submitted the result to Kaggle. As what shown in the following graph, our result's score is 0.96885. Knn model does not performed perfectly enough, in order to have a higher score, we need to explore other models.



2. (a) Load the data and use logistic regression, try to predict whether a passenger survived the disaster. You can choose the features

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import cross_validation, preprocessing
df = pd.read_csv('/Users/yingzhu/Desktop/train1.csv')
print(df.shape)
```

```
(891, 12)
```

```
# delete the features has little influence on survivial
Y=df.Survived
df=df.drop(['PassengerId','Survived','Pclass','Name','Ticket','Embarked','Cabin'],axis=1)
print(df.shape)
```

First, we load the data using pandas library. After observing the data, we firstly found that "Passenger Id", "Name","Ticket" and "Embarked" may not be relevant to the survival result.Even though both the "Cabin" and "Pclass" seems related with survival rate. The information provided by these two features could be provided by "Fare". So in our logistic regression, we only include 5 features: "Sex", "Age", "SibSp", "Parch", "Fare".

```
# change gender to int
df.Sex.replace('female',0,inplace=True)
df.Sex.replace('male',1,inplace=True)
```

```
# fill the missing number in Age with mean
for col in df:
    df[col].fillna(np.mean(df[col]),inplace=True)
```

Since the gender is presented as text in the training data, we first change the expression of gender from text to numeric. We also deal with the missing value by filling them by feature mean.

```
# scalar transfer
print(df.max() - df.min())
```

```
Sex         1.0000
Age        79.5800
SibSp       8.0000
Parch       6.0000
Fare      512.3292
dtype: float64
```

```
scaler1= preprocessing.StandardScaler().fit(df['Fare'])
df['Fare'] = scaler1.transform(df['Fare'])
scaler2= preprocessing.StandardScaler().fit(df['Parch'])
df['Parch'] = scaler2.transform(df['Parch'])
scaler3= preprocessing.StandardScaler().fit(df['SibSp'])
df['SibSp'] = scaler3.transform(df['SibSp'])
scaler4= preprocessing.StandardScaler().fit(df['Age'])
df['Age'] = scaler4.transform(df['Age'])
print(df.head(10))
```

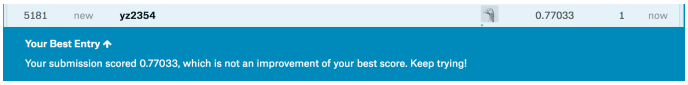
The ranges of 5 features are pretty large. So we do normalization for all the features except "Sex".

(b) After running the logistic regression using 5 folds cross validation, we get the average score as 0.787.

```
# run logistic regression
X=df[col_selected].values[:,0:]
accu=[]
for train_indices, test_indices in cross_validation.KFold(len(X),n_folds=5):
    X_train = X[train_indices]
    Y_train = Y[train_indices]
    X_test = X[test_indices]
    Y_test = Y[test_indices]
    ls = LogisticRegression()
    ls.fit(X_train,Y_train)
    accu.append(ls.score(X_test,Y_test))
print(accu)
print(np.mean(accu))
```

```
[0.7988826815642458, 0.8033707865168539, 0.7584269662921348, 0.7471910112359551, 0.8314606741573034]
0.787866423953
```

(c) Submission to Kaggle
Finally, we train our model using all the training data and run test data in it. We got 0.77 in the final Kaggle submission.



3. (a) Show $Var(X-Y) = Var(X) + Var(y) - 2Cov(X,Y)$

$$Var[X-Y] = E[(X-Y)^2] - E[X-Y]^2$$
$$= E(X^2 + Y^2 - 2XY) - (E(X) - E(Y))^2$$
$$= E(X^2) + E(Y_2) - 2E(XY) - [E(X)^2 + E(Y)^2 - 2E(X)E(Y)]$$
$$= [E(X^2) - E(X)^2] + [E(Y^2) - E(Y)^2] - 2[E(XY) - E(X)E(Y)]$$
$$= Var(X) + Var(Y) - 2Cov(X,Y)$$

(b) i. Suppose the test shows that a widget is defective. What are the chances that it's actually defective given the test result?

$$P(defective|positive) = \frac{P(defective, positive)}{P(positive)}$$
$$= \frac{P(positive|defective) * P(defective)}{P(positive|defective) * P(defective) + P(positive|notdefective) * P(notdefective)}$$
$$= \frac{0.95/100000}{0.95/100000 + 0.05 * (1 - \frac{1}{100000})} = 0.00018997$$

ii. If we throw out all widgets that are defective, how many good widgets are thrown away per year? How many bad widgets are still shipped to customers each year?

$$P(positive, notdefective) = P(positive|notdefective) * P(notdefective)$$
$$= 0.05 * (1 - \frac{1}{100000}) = 0.049995$$
$$0.049995 * 10000000 = 499995$$
$$P(defective, negative) = P(negative|defective) * P(defective)$$
$$= 0.05 * \frac{1}{100000} = 0.0000005$$
$$10000000 * 0.0000005 = 5$$

(c) i. Describe what happens to the average 0-1 prediction error on the training data when the neighbor count k varies from n to 1.

If we plot the graph of k and the prediction error, we will probably will end with a U shape curve. There should exist a threshold k* such that, choosing k* will give the lowest prediction error. For k number smaller than k*, prediction error will decrease gradually. When k=1, the nearest neighbor will be that point itself. Which means that the error rate will be zero. But for the k number ¿ k* ,the prediction error will gradually increase.When k=n, the whole data set will be the K nearest neighbor. Since

each class has exactly half of the training data, the error rate will be 0.5.

ii. We randomly choose half of the data to be removed from the training data, train on the remaining half, and test on the held-out half. Predict and explain with a sketch how the average 0-1 prediction error on the held-out validation set might change when k varies? Explain your reasoning.

The answer is very similar as what we answered in (a).When k=1, the error rate is high. The reason is it over fitting the model. Boundary for each data point are just for itself, not for the general data set.As k increase, the error rate will decrease at first. When reaching at certain point, it will increase because it starting under fitting the data points.But the different from (a), since we lost half of the data, the K* will be smaller than that in (a). When k= n, half of the data set will be the K nearest neighbor. Since we removed data randomly and each class has exactly half of the training data, the error rate will approximately equal to 0.5.

iii. We wish to choose k by cross-validation and are considering how many folds to use. Compare both the computational requirements and the validation accuracy of using different numbers of folds for kNN and recommend an appropriate number.

Let's assume we choose an appropriated number of neighbors in KNN.When using small number of fold cross-validation, the computational requirement is low and validation accuracy is low. It means that the estimated error rate will larger than true error rate.When using K=n fold cross-validation, the computational requirement is the highest. But the validation accuracy is high. In real world, the number of folds depends on how large the data sets are. When the data sets are spread out, we will try as many folds as it could be in order to collect more information. When data sets are large, 2 or 3 folds will be enough to get a high accuracy error rate.

iv. In kNN, once k is determined, all of the k-nearest neighbors are weighted equally in deciding the class label. This may be inappropriate when k is large. Suggest a modification to the algorithm that avoids this caveat.

We could use the inverse of distance of the neighbors as the weight. The neighbors with larger distance will influence less on

deciding the label and the neighbors with smaller distance will play more important roles to decide the label.

v. Give two reasons why KNN may be undesirable when the input dimension is high.

   A. When the dimension increases, the computation will be extremely expensive.kNN will be undesirable when the input dimension is high.

   B. When the dimension increases, in order to have a more precise result, we will need a larger number of example. Since the increase of dimension will decrease in a volume of space exponentially.

2 Reference

Fi, Lsi -. "K-Nearest Neighbours."