AML HW 1

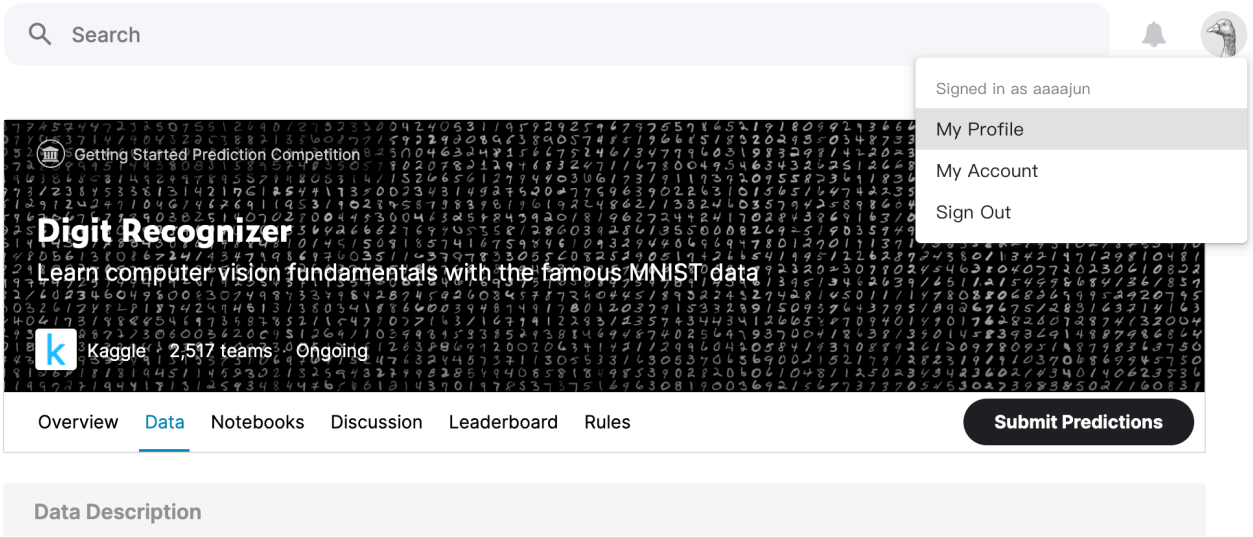
Name: Zhang Zihan NetID: zz698 Program: ORIE

Name: Scarlett Huang NetID: sh2557 Program: CM

Programming problem 1

(a)

I have joined the Digit Recognizer competition on Kaggle and my id is aaaajun.



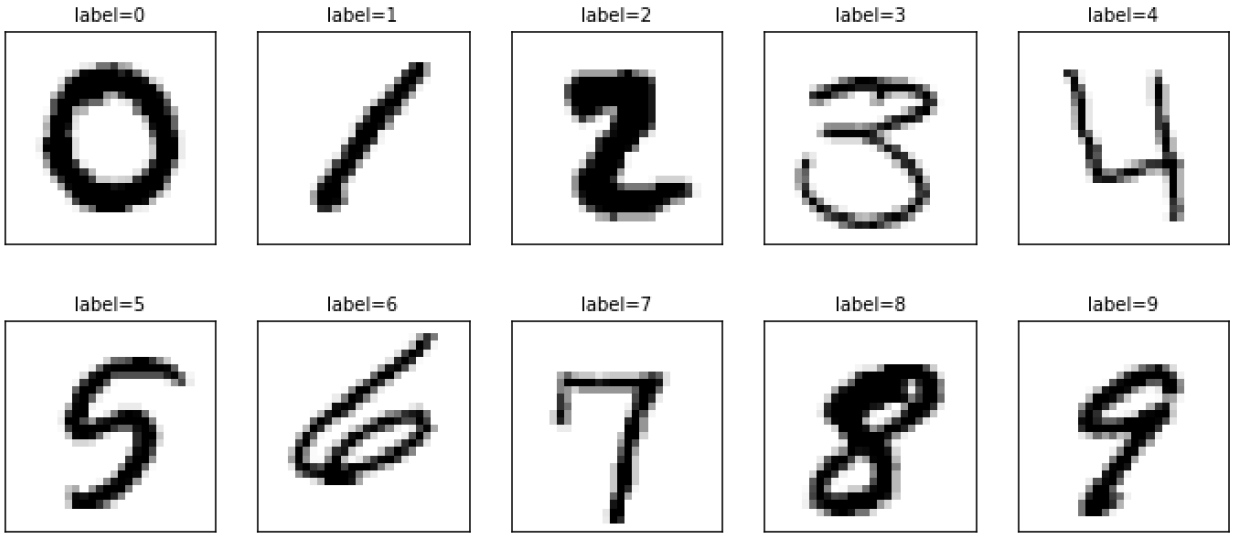
(b)

```
#Problem 1.b
x_train_image=train.drop('label',axis=1)#dataframe with index
y_train_label=train.label

X_train=x_train_image.values.reshape(42000,28,28).astype('float32')
X_test=x_test_image.values.reshape(28000,28,28).astype('float32')

x_train_data = np.array(x_train_image)#array 42000*784

images=[]
labels=[]
num=[1,0,16,7,3,8,21,6,10,11]
for i in range(10):
    images.append(X_train[num[i]])
    labels.append(y_train_label[num[i]])

def plot_images_labels(images,labels,idx,num=10):
    fig=plt.gcf()
    fig.set_size_inches(12,14)
    for i in range(0,10):
        ax=plt.subplot(5,5,i+1)
        ax.imshow(images[idx],cmap='binary')
        title='label='+str(labels[idx])
```

```
        ax.set_title(title,fontsize=10)
        ax.set_xticks([]);ax.set_yticks([])
        idx+=1
    plt.show()

plot_images_labels(images,labels,idx=0)
```

Result:

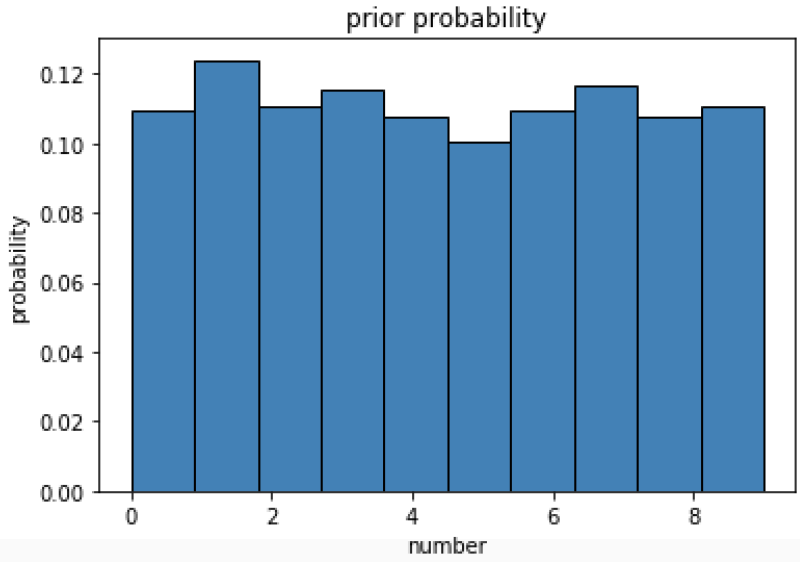
```
In [29]: runfile('/Users/zzhajun/Downloads/hw1p1', wdir='/Users/zzhajun/Downloads')
```



(c)

```
#Problem 1.c
count=np.zeros((10,), dtype=np.int)
prob=np.zeros((10,))
for i in range(len(y_train_label)):
    for j in range(10):
        if y_train_label[i]==j:
            count[j]+=1
print(count)

for i in range(10):
    prob[i]=count[i]/42000
    print(prob[i])
plt.hist(x = y_train_label, density=True, bins = 10, color = 'steelblue', edgecolor =
'black' )

plt.xlabel('number')
plt.ylabel('probability')
plt.title('prior probability')
plt.show()
```

```
In [46]: runfile('/Users/zzhajun/Downloads/hw1p1', wdir='/Users/zzhajun/Downloads')
[4132 4684 4177 4351 4072 3795 4137 4401 4063 4188]
0.09838095238095237
0.11152380952380953
0.09945238095238096
0.1035952380952381
0.09695238095238096
0.09035714285714286
0.0985
0.10478571428571429
0.09673809523809523
0.09971428571428571
```



The prior probability of the classes in the training data is not uniform across the digits.

The normalized histogram of digit counts is not even.

(d)

```
#Problem 1d
a=0
dist=[10000,10000,10000,10000,10000,10000,10000,10000,10000,10000]
match=np.zeros(10)
for i in range(10):
    for j in range(42000):
        if j!=num[i]:
            a=np.linalg.norm(images[i]-X_train[j])
            if dist[i]>a:
                dist[i]=a
                match[i]=y_train_label[j]

print(dist)
print(match)
```

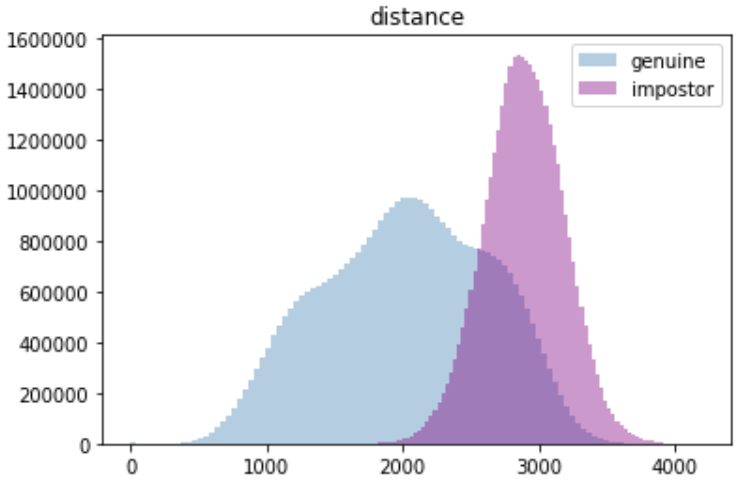
```
In [54]: runfile('/Users/zzhajun/Downloads/hw1p1', wdir='/
Users/zzhajun/Downloads')
[1046.5955, 489.67947, 1380.8772, 1832.665, 1356.881,
1066.3677, 1446.5114, 863.50104, 1593.7776, 910.5767]
[0. 1. 2. 5. 4. 5. 6. 7. 8. 9.]
```

The best macthes between my chosen sample and the rest of the training data are shown above.

Class 3 is error in this case.

(e)

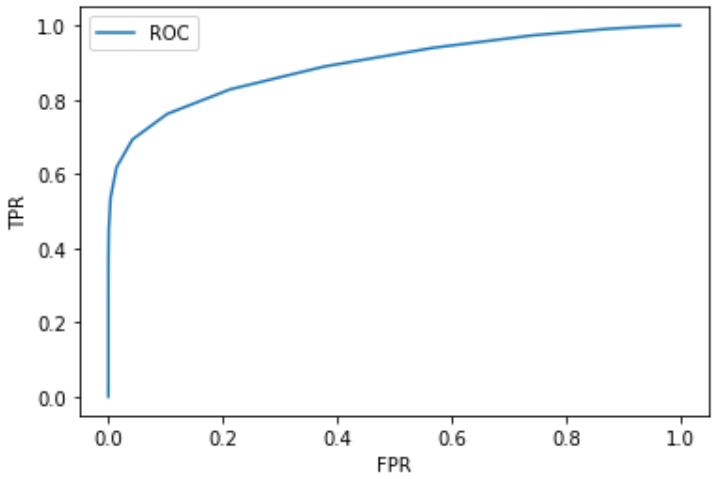
```
#Problem 1.e
num0=[]
image0=[]
total0=0
num1=[]
image1=[]
total1=0
for i in range(42000):
    if y_train_label[i]==0:
        num0.append(y_train_label[i])
        image0.append(x_train_data[i])
        total0+=1
    elif y_train_label[i]==1:
        num1.append(y_train_label[i])
        image1.append(x_train_data[i])
        total1+=1
#print(total0,total1)
dist00=cdist(image0,image0,'euclidean')
dist11=cdist(image1,image1,'euclidean')
dist01=cdist(image0,image1,'euclidean')

dist00=dist00.reshape(1,4132*4132)
dist11=dist11.reshape(1,4684*4684)
dist01=dist01.reshape(1,4132*4684)

dist00=np.array(dist00)
dist00=dist00.tolist()

dist11=np.array(dist11)
dist11=dist11.tolist()

dist01=np.array(dist01)
dist01=dist01.tolist()

dist_genu=dist00[0]+dist11[0]
dist_impo=dist01[0]+dist01[0]

sns.distplot(dist_genu, bins = 100, kde = False, hist_kws = {'color':'steelblue'}, label =
'genuine')
sns.distplot(dist_impo, bins = 100, kde = False, hist_kws = {'color':'purple'}, label =
'impostor')
plt.title('distance')
plt.legend()
plt.show()
```



The histogram of the genuine and impostor distances on the same set of axes is shown above.

(f)

```
#Problem 1.f

genu_min=min(dist_genu)
genu_max=max(dist_genu)
impo_min=min(dist_impo)
impo_max=max(dist_impo)
dist_max=max(genu_max,impo_max)
distmax=[]
distmax.append(dist_max)
print(dist_max)#max distance
inter=70
count_tp=np.zeros(60)
count_fp=np.zeros(60)

#TPR
for j in range(60):
    for i in range(39013280):
        if dist_genu[i]<distmax[0]-inter*j:
            count_tp[j]+=1
count_tp=count_tp/ 39013280

#FPr
for j in range(60):
    for i in range(38708576):
        if dist_impo[i]<distmax[0]-inter*j:
            count_fp[j]+=1
count_fp=count_fp/ 38708576

plt.plot(count_fp,count_tp,label='ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



The ROC curve from the above sets of distances is shown above.

EER= TPR=1-FPR=0.82

The error rate of a classifier that simply guesses randomly should be 0.5.

(g) see the coding file

(h)&(i)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Oct  1 21:23:35 2020

@author: zzhajun
"""

from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier as KNN

def plot_confusion_matrix(cm, title='Confusion Matrix', cmap=plt.cm.binary):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    xlocations = np.array(range(len(labels)))
    plt.xticks(xlocations, labels, rotation=90)
    plt.yticks(xlocations, labels)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

if __name__=='__main__':
    train=pd.read_csv("train.csv")
    x_train_image=train.drop('label',axis=1)#dataframe with index
    y_train_label=train.label

    X_train, X_test, y_train, y_test = train_test_split(x_train_image, y_train_label,
test_size=0.15, random_state=42)
```

```
    X_train_normalize=X_train/255
    X_test_normalize=X_test/255

    X_train_normalize=X_train_normalize.values
    X_test_normalize=X_test_normalize.values
    y_train=y_train.values
    y_test=y_test.values

    errorCount = 0.0

    mTest = 6300
    pred=[]
    true=[]
    for i in range(mTest):
        true.append(y_test[i])

    for k in range(2,10):
        neigh =KNN(n_neighbors =3, algorithm = 'auto')
        neigh.fit(X_train_normalize, y_train)
        for i in range(mTest):
            classifierResult = neigh.predict([X_test_normalize[i]])
            pred.append(classifierResult)
            #print("predict%d\tlabel%d" % (classifierResult, y_test[i]))
            if(classifierResult != y_test[i]):
                errorCount += 1.0
        acc=(1-errorCount/mTest)*100
        print(acc,'%')

    labels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

    tick_marks = np.array(range(len(labels))) + 0.5
    cm = confusion_matrix(true, pred)
    np.set_printoptions(precision=2)
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print (cm_normalized)
    plt.figure(figsize=(12, 8), dpi=120)

    ind_array = np.arange(len(labels))
    x, y = np.meshgrid(ind_array, ind_array)

    for x_val, y_val in zip(x.flatten(), y.flatten()):
        c = cm_normalized[y_val][x_val]
        if c > 0.01:
            plt.text(x_val, y_val, "%0.2f" % (c,), color='red', fontsize=7, va='center',
ha='center')
            # offset the tick
    plt.gca().set_xticks(tick_marks, minor=True)
    plt.gca().set_yticks(tick_marks, minor=True)
    plt.gca().xaxis.set_ticks_position('none')
    plt.gca().yaxis.set_ticks_position('none')
    plt.grid(True, which='minor', linestyle='-')
    plt.gcf().subplots_adjust(bottom=0.15)
    plot_confusion_matrix(cm_normalized, title='Normalized confusion matrix')
```

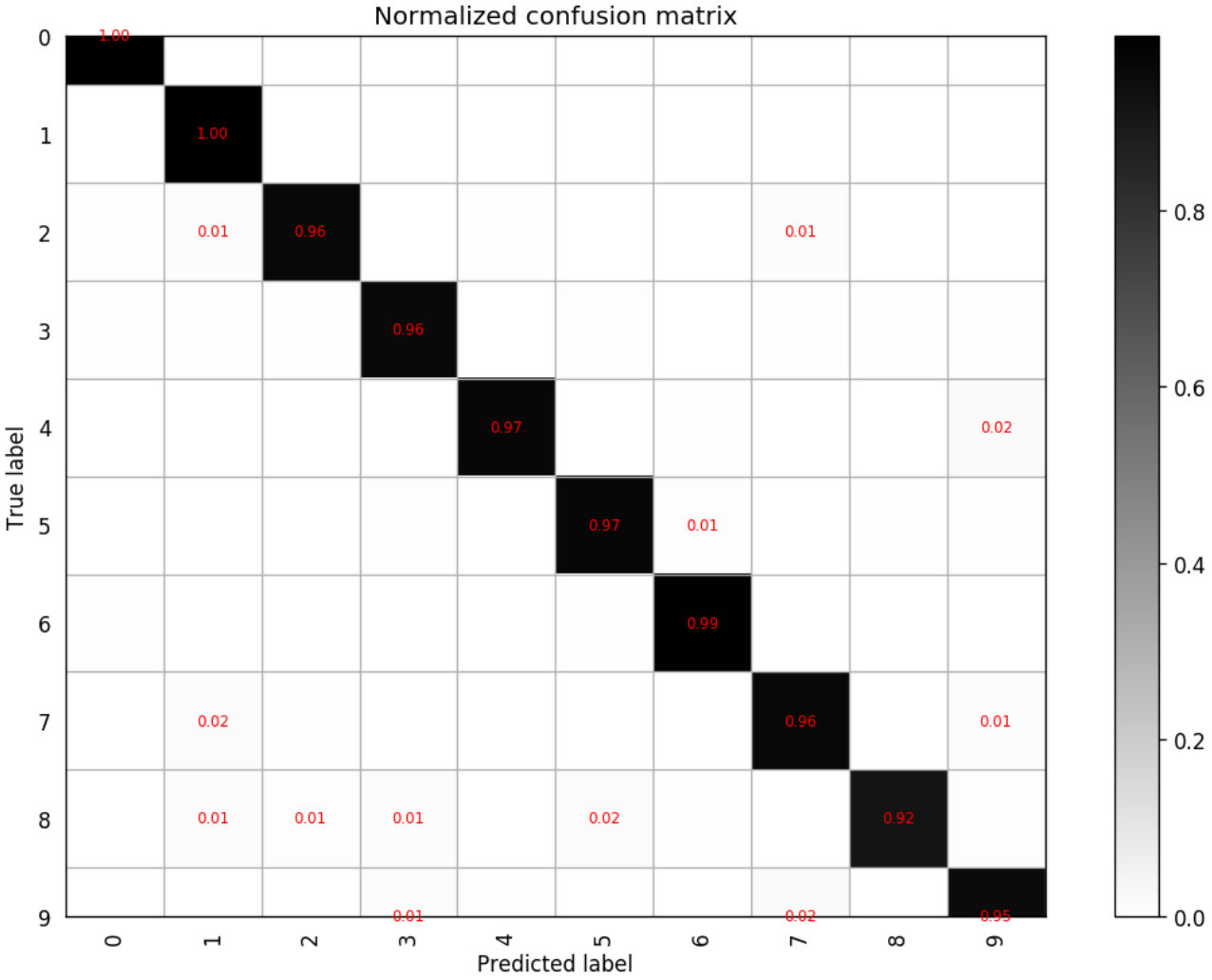
Use KNN in sklearn to calculate the optimal k

K	2	3	4	5	6	7	8	9	10
Accuracy	95%	97%	95%	93%	92%	90%	88%	87%	85%

So I choose k=3.

The accuracy and confusion matrix of sklearn is :

```
96.58730158730158 %
[[1.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   1.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.01 0.96 0.   0.   0.   0.   0.01 0.   0.  ]
 [0.   0.01 0.01 0.96 0.   0.01 0.   0.01 0.01 0.  ]
 [0.   0.   0.   0.   0.97 0.   0.   0.   0.   0.02]
 [0.   0.   0.   0.01 0.   0.97 0.01 0.   0.01 0.01]
 [0.01 0.   0.   0.   0.   0.01 0.99 0.   0.   0.  ]
 [0.   0.02 0.01 0.   0.   0.   0.   0.96 0.   0.01]
 [0.   0.01 0.01 0.01 0.   0.02 0.   0.   0.92 0.01]
 [0.   0.   0.   0.01 0.   0.   0.   0.02 0.   0.95]]
```



Digit 8 is particularly tricky to classify.

```
#!/usr/bin/env python3
```

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  2 06:57:48 2020

@author: zzhajun
"""

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  2 06:03:13 2020

@author: zzhajun
"""

import operator
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split


def classify(inX,dataset,labels,k):
    #start=time.time()
    diffMat = inX[None,:] - dataset
    #print(time.time()-start)
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances ** 0.5
    #print(time.time()-start)
    #sort distance, return index
    sortedDistIndicies = distances.argsort()
    #print(time.time()-start)
    #dictionary
    classCount = {}
    #k least distance
    for i in range(k):
        #sortedDistIndicies[0]index of min dist
        #labels[sortedDistIndicies[0]]label of min dist
        voteIlabel = labels[sortedDistIndicies[i]]
        #label as key,support key +1
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1
    #sort
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)
    #print(time.time()-start)
    return sortedClassCount[0][0]

def classify2(inX,dataset,labels,k):
    diffMat = inX[None,:] - dataset
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances ** 0.5
    sortedDistIndicies = distances.argsort()
    classCount = {}
    for i in range(k):
```

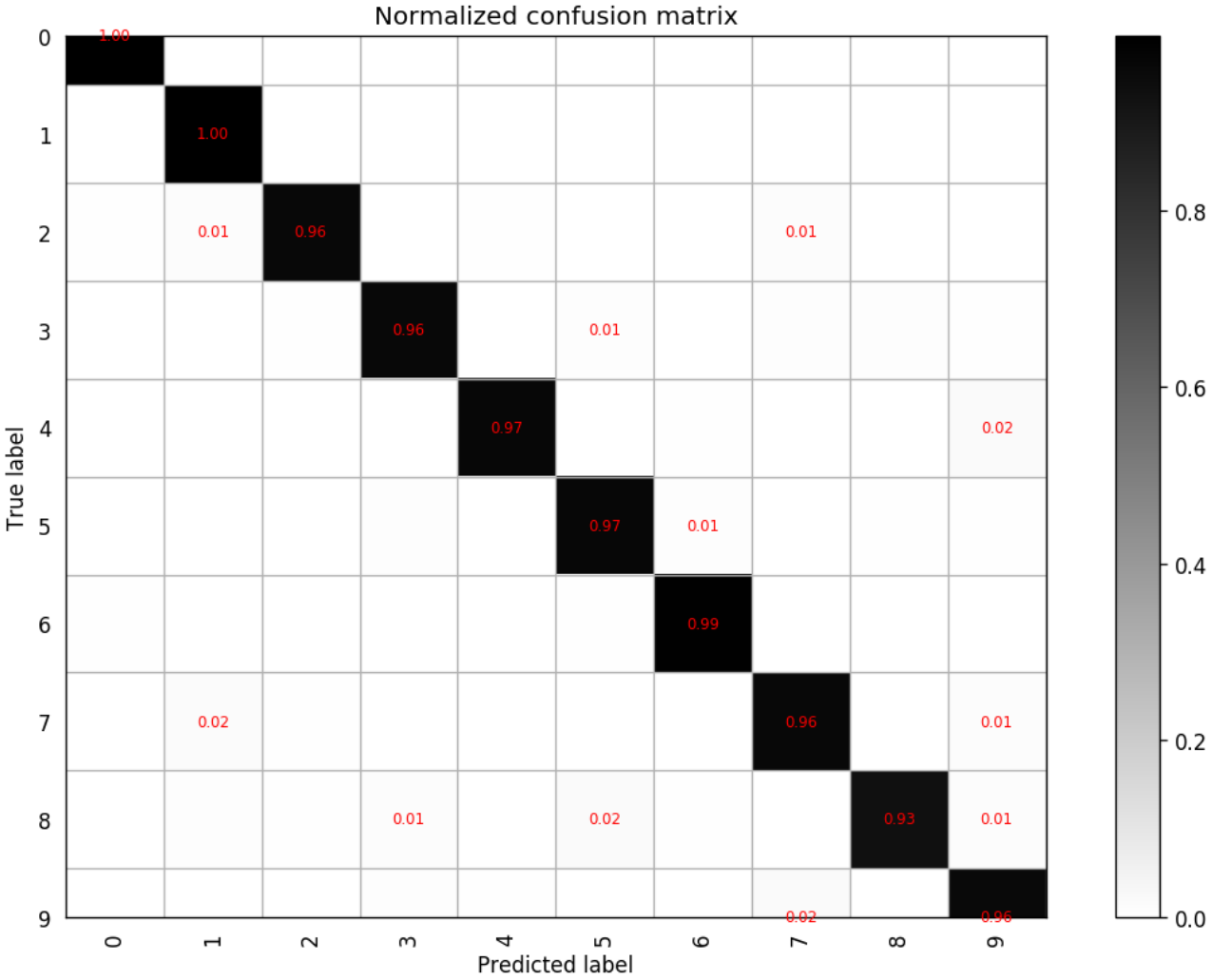
```
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]


def plot_confusion_matrix(cm, title='Confusion Matrix', cmap=plt.cm.binary):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    xlocations = np.array(range(len(labels)))
    plt.xticks(xlocations, labels, rotation=90)
    plt.yticks(xlocations, labels)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

if __name__ == '__main__':
    train=pd.read_csv("train.csv")
    x_train_image=train.drop('label',axis=1)#dataframe with index
    y_train_label=train.label

    X_train, X_test, y_train, y_test = train_test_split(x_train_image, y_train_label,
test_size=0.15, random_state=42)



    X_train_normalize=X_train/255
    X_test_normalize=X_test/255

    X_train_normalize=X_train_normalize.values
    X_test_normalize=X_test_normalize.values
    y_train=y_train.values
    y_test=y_test.values

    mTest = 6300
    mTrain=35700
    true_hold=[]
    true_train=[]
    for i in range(mTest):
        true_hold.append(y_test[i])

    for i in range(mTrain):
        true_train.append(y_train[i])
    Result_hold=[]
    Result_train=[]
    errorCount=0
    errorCount_train=0

    for i in range(mTest):
        Result_hold.append(classify( X_test_normalize[i], X_train_normalize,y_train, 3))
        if (Result_hold[i]!= y_test[i]):
            errorCount+=1
        #print( Result_hold[i],y_test[i])
    acc_hold=(1-errorCount/mTest)*100
    print(acc_hold,'%')
```

```
    for i in range(mTrain):
        Result_train.append(classify2(X_train_normalize[i], X_train_normalize,y_train, 3))
        if (Result_train[i]!= y_train[i]):
            errorCount_train+=1
        print(Result_train[i],y_train[i])
    acc_train=(1-errorCount_train/mTrain)*100
    print(acc_train,'%')


    labels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

    tick_marks = np.array(range(len(labels))) + 0.5
    #cm = confusion_matrix(true_hold, Result_hold)
    cm1 = confusion_matrix(true_train, Result_train)
    np.set_printoptions(precision=2)
    #cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    cm1_normalized = cm1.astype('float') / cm1.sum(axis=1)[:, np.newaxis]
    print (cm1_normalized)
    plt.figure(figsize=(12, 8), dpi=120)

    ind_array = np.arange(len(labels))
    x, y = np.meshgrid(ind_array, ind_array)

    for x_val, y_val in zip(x.flatten(), y.flatten()):
        c = cm_normalized[y_val][x_val]
        if c > 0.01:
            plt.text(x_val, y_val, "%0.2f" % (c,), color='red', fontsize=7, va='center',
ha='center')
            # offset the tick
    plt.gca().set_xticks(tick_marks, minor=True)
    plt.gca().set_yticks(tick_marks, minor=True)
    plt.gca().xaxis.set_ticks_position('none')
    plt.gca().yaxis.set_ticks_position('none')
    plt.grid(True, which='minor', linestyle='-')
    plt.gcf().subplots_adjust(bottom=0.15)
    plot_confusion_matrix(cm_normalized, title='Normalized confusion matrix')

    for x_val, y_val in zip(x.flatten(), y.flatten()):
        c = cm1_normalized[y_val][x_val]
        if c > 0.01:
            plt.text(x_val, y_val, "%0.2f" % (c,), color='red', fontsize=7, va='center',
ha='center')
            # offset the tick
    plt.gca().set_xticks(tick_marks, minor=True)
    plt.gca().set_yticks(tick_marks, minor=True)
    plt.gca().xaxis.set_ticks_position('none')
    plt.gca().yaxis.set_ticks_position('none')
    plt.grid(True, which='minor', linestyle='-')
    plt.gcf().subplots_adjust(bottom=0.15)
    plot_confusion_matrix(cm1_normalized, title='Normalized confusion matrix')
```

The accuracy and confusion matrix of my implementation is:

Hold:

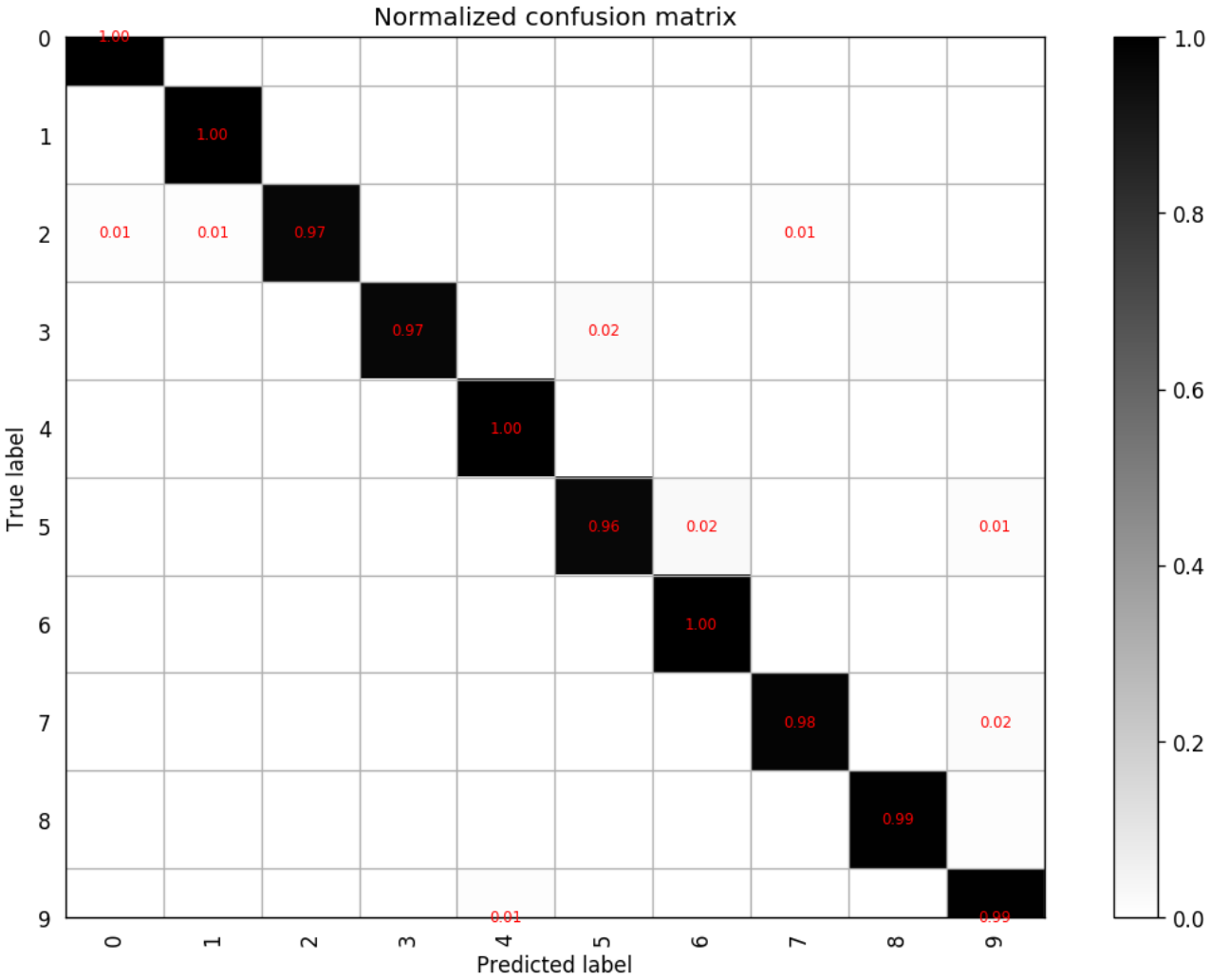
```
96.7936507936508 %
[[1.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   1.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.01 0.96 0.   0.   0.   0.   0.01 0.   0.  ]
 [0.   0.   0.   0.96 0.   0.01 0.   0.01 0.01 0.  ]
 [0.   0.   0.   0.   0.97 0.   0.   0.   0.   0.02]
 [0.   0.   0.   0.01 0.   0.97 0.01 0.   0.01 0.01]
 [0.01 0.   0.   0.   0.   0.01 0.99 0.   0.   0.  ]
 [0.   0.02 0.   0.   0.   0.   0.   0.96 0.   0.01]
 [0.   0.01 0.01 0.01 0.   0.02 0.   0.   0.93 0.01]
 [0.   0.   0.   0.01 0.   0.   0.   0.02 0.   0.96]]
```



Digit 8 is particularly tricky to classify.

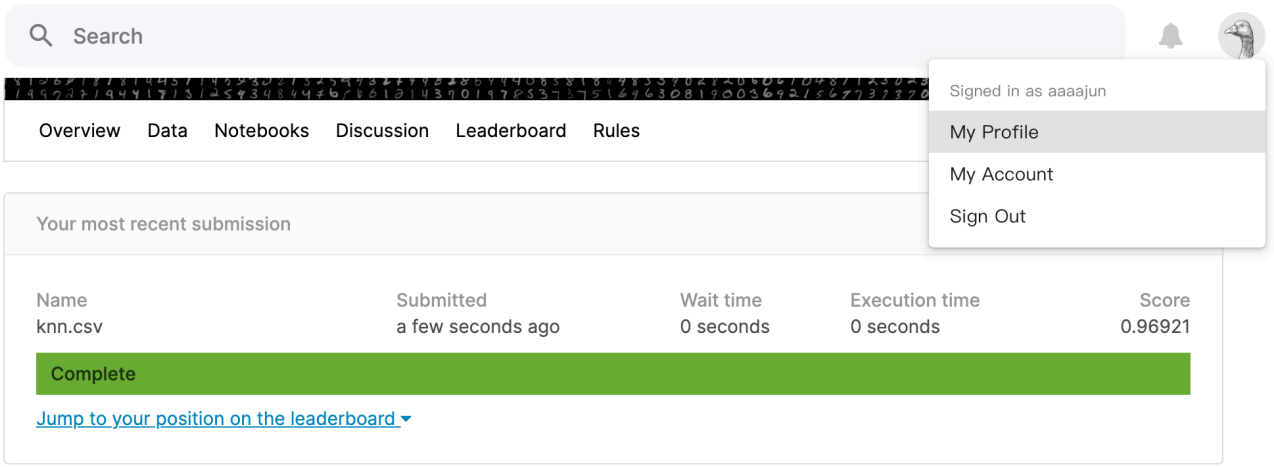
Train:

```
98.7 %
[[1.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   1.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.01 0.01 0.97 0.   0.   0.   0.   0.01 0.   0.  ]
 [0.   0.   0.   0.97 0.   0.02 0.   0.   0.01 0.  ]
 [0.   0.   0.   0.   1.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.96 0.02 0.   0.   0.01]
 [0.   0.   0.   0.   0.   0.   1.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.98 0.   0.02]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.99 0.01]
 [0.   0.   0.   0.   0.01 0.   0.   0.   0.   0.99]]
```



Digit 5 is particularly tricky to classify.

(j)



The submission page is shown above.

Programming problem 2

The description is in the coding file.

Written problem 1

$$\arg\max_{\theta} E_{\hat{p}(x,y))}[logP_\theta(y|x))]$$

$$= \arg\min_{\theta} E_{\hat{p}(x)}[KL(\hat{p}(y|x)||P_\theta(y|x)]$$

$$= \arg\min_{\theta} E_{\hat{p}(x)}[E_{\hat{p}(x,y)}[log\hat{p}(y|x) - logP_\theta(y|x)]]$$

As is the empirical data distribution (no randomness), $E_{\hat{p}(x,y)} log\hat{p}(y|x)$ is Constant

$$= C - \arg\min_{\theta} E_{\hat{p}(x)} E_{\hat{p}(x,y)}(-logP_\theta(y|x))$$

$$= C - \arg\min_{\theta} E_{\hat{p}(x,y)}(-logP_\theta(y|x))$$

$$= C + \arg\max_{\theta} E_{\hat{p}(x,y)} logP_\theta(y|x)$$

Written problem 2

(a)

suppose:

- test quality as event A
- actual quality as event B
- defective: 1
- not defective: 0
- test defective: $P(A = 1)$
- test not defective: $P(A = 0)$
- actually defective: $P(B = 1)$
- actually not defective: $P(B = 0)$

from the text we know:

- $P(A = 1|B = 1) = P(A = 0|B = 0) = 0.95$
- $P(A = 0|B = 1) = P(A = 1|B = 0) = 0.05$
- $P(B = 1) = \frac{1}{100000} = 0.00001$
- $P(B = 0) = 1 - P(B = 1) = 0.99999$

we can calculate out:

- $P(A=1)=P(A=1|B=1)P(B=1)+P(A=1|B=0)P(B=0)=0.95*0.00001+0.05*0.99999=0.050009$
- $P(A=0)=1-0.050009=0.949991$

the chances that the widge is actually defective given the test defective result:

$$P(B=1|A=1)=\frac{P(B=1)P(A=1|B=1)}{P(A=1)}=\frac{0.00001*0.95}{0.050009}=0.000189966$$

(b)

sum widgets per year = 10000000

the probability of good widgets are thrown away per year:

$$P(B=0|A=1)=1-P(B=1|A=1)=1-0.000189966=0.999810034$$

the number of good widgets are thrown away per year:

$$10000000*P(B=0|A=1)*P(A=1)$$

$$=10000000*0.999810034*0.050009=499995$$

the probability of bad widgets are still shipped to customers each year:

$$P(B=1|A=0)=\frac{P(B=1)P(A=0|B=1)}{P(A=0)}$$

the number of bad widgets are still shipped to customers each year:

$$10000000*P(B=1|A=0)*P(A=0)$$

$$=10000000*\frac{P(B=1)P(A=0|B=1)}{P(A=0)}*P(A=0)$$

$$=10000000*\frac{0.00001*0.05}{0.949991}*0.949991=5$$

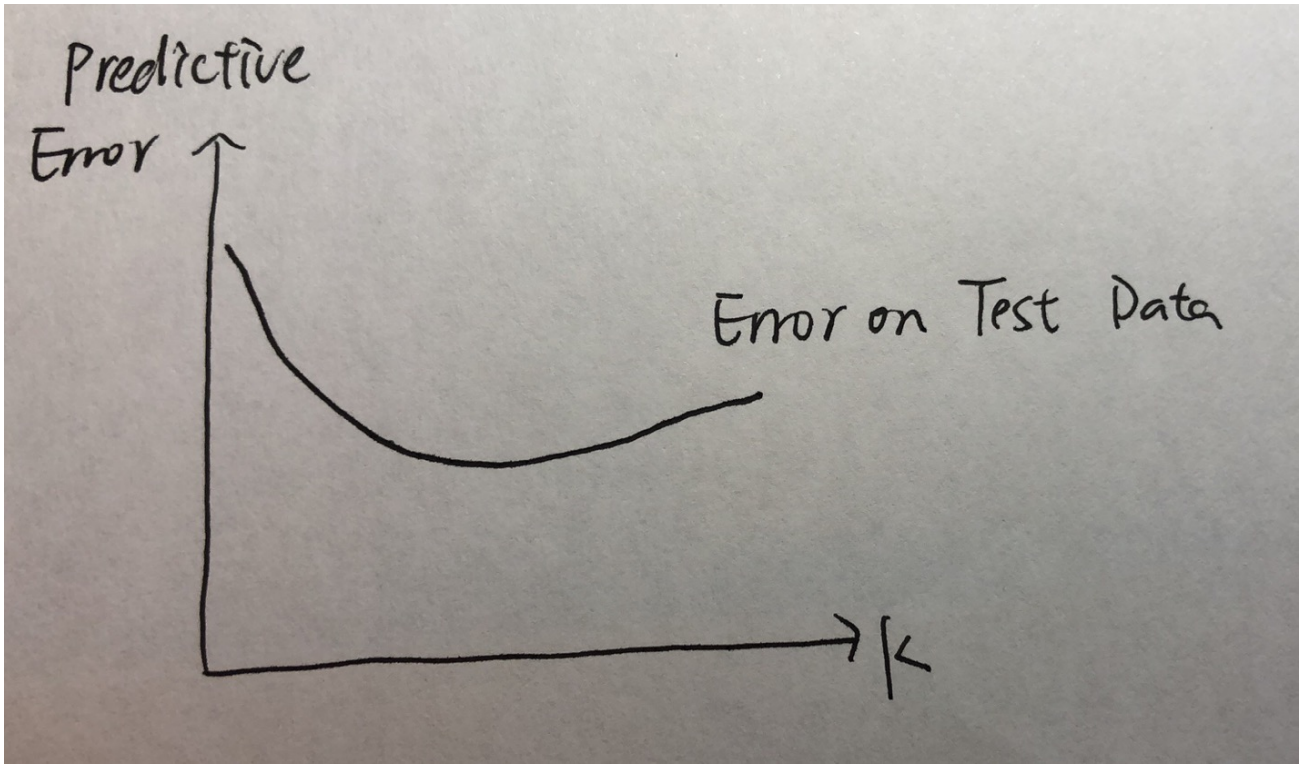
Written problem 3

(a)

The training data would be predicted with accuracy=100% since the prediction for training data point x_i includes (x_i, y_i), which would let the distance be 0. The error would be 0 when k=1. When k is large, the variance is high, the model is sensitive in the training set, which is easily affected by the proportion of the nearest k points and would be overfitting. Thus, when k increasing, the error on the training data would increased.

(b)

The average 0-1 prediction error on the held-out half would be high when k=1 since the bias is really high at that time, the model would be underfitting. The error would decrease with k increasing(decrease variance). When k>the optimal k, the error would slightly increase.



(c)

When k is large, we should consider the frequent of different label, which means we should use weighted KNN.

We can use gaussian function to optimize the weight of samples with different distances. When the distance between the training sample and the test sample increases, the weight of the distance value takes effect.

The closer neighbors are assigned more weights, while the weights of the farther neighbors are reduced accordingly, taking the weighted average.