





3.

1. The error is 0.01625
2. tokens that are most indicative of the SPAM class ['httpaddr', 'spam', 'unsubscrib', 'ebai', 'valet']
3. For the other datasets:

*Some other datasets*

*Data size: 50; error: 0.03875*

*Data size: 100; error: 0.02625*

*Data size: 200; error: 0.02625*

*Data size: 400; error: 0.01875*

*Data size: 800; error: 0.0175*

*Data size: 1400; error: 0.01625*

The code I implemented is shown as below(language == Python):

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 12 13:56:53 2017

@author: liuchangbai

"""

*import csv*

*import os*

*import numpy as np*

*import pandas as pd*

*import nb*

*os.chdir("/Users/liuchangbai/Desktop/courses/Machine-Learning/Homework/HW2")*

*token\_list = []*

*word\_list = []*

*token\_path = os.path.expanduser('./spam\_classification/TOKENS\_LIST')*

*with open(token\_path,newline='') as token:*

*reader = csv.reader(token, delimiter=' ')*

*for row in reader:*

*token\_list.append(row)*

*for i in token\_list:*

*word\_list.append(i[1])*

*train\_path = os.path.expanduser('./spam\_classification/SPARSE.TRAIN')*

*with open(train\_path, newline='') as train:*

*reader = csv.reader(train, delimiter=' ')*

*for row in reader:*

*label.append(int(row[0]))*

*label = np.asarray(label,dtype=int)*

*count\_d\_w = np.zeros([nd,nw],dtype=int)*

*with open(train\_path, newline='') as train:*

*reader = csv.reader(train, delimiter=' ')*

*for d\_id, row in enumerate(reader):*

*current\_email = csv.reader(row[2:-1],delimiter=':')*

*for rows in current\_email:*

*w\_id = int(rows[0])*

*count = int(rows[1])*

*count\_d\_w[d\_id][w\_id-1] = count*

*df\_train = pd.DataFrame(count\_d\_w, columns = [word\_list])*

*df\_train["label"] = pd.Series(label)*

*label\_test\_buf = list()*

*test\_path = os.path.expanduser('./spam\_classification/SPARSE.TEST')*

*with open(test\_path, newline='') as test:*

*reader = csv.reader(test, delimiter=' ')*

*for row in reader:*

*label\_test\_buf.append(int(row[0]))*

*label\_test = np.asarray(label\_test\_buf,dtype=int)*

*nd\_test = len(label\_test)*

*count\_d\_w\_test = np.zeros([nd\_test,nw],dtype=int)*

*with open(test\_path, newline='') as test:*

*reader = csv.reader(test, delimiter=' ')*

*for d\_id, row in enumerate(reader):*

*current\_email = csv.reader(row[2:-1],delimiter=':')*

*for rows in current\_email:*

*w\_id = int(rows[0])*

*count = int(rows[1])*

*count\_d\_w\_test[d\_id][w\_id-1] = count*

*df\_test = pd.DataFrame(count\_d\_w\_test)*

*nb\_model = nb.train(df\_train)*

*nb\_predictions = nb.test(nb\_model, df\_test)*

*y = pd.Series(label\_test)*

*nb\_error = nb.compute\_error(y, nb\_predictions)*

*print('NB Test error: {}'.format(nb\_error))*

*words = nb.k\_most\_indicative\_words(5, nb\_model.to\_dataframe().iloc[:,:-1])*

*print('The {} most spam-worthy words are: {}'.format(len(words), words))*

4. (1)

L2-norm:

k = 1: accuracy = 94%

k = 5: accuracy = 98%

k = 9: accuracy = 96%

k = 13: accuracy = 96%

(2)

L1-norm:

k = 1: accuracy = 92%

k = 5: accuracy = 96%

k = 9: accuracy = 95%

k = 13: accuracy = 94%

It seems L2 has better accuracy than L1-norm. And when k = 5 has the best accuracy.

I think it is because this matrix is a sparse matrix and L2-norm magnifies big entries, so that it could be better than L1-norm.

*#!/usr/bin/env python3*

*# -\*- coding: utf-8 -\*-*

*"""*

*Created on Thu Oct 12 19:52:44 2017*

*@author: liuchangbai*

*"""*

*import os*

*import numpy as np*

*import scipy.io as sio*

*from sklearn import cross\_validation, neighbors*

*import operator*

*os.chdir("/Users/liuchangbai/Desktop/courses/Machine-Learning/Homework/HW2")*

*mnist\_data = sio.loadmat('mnist\_data.mat')*

*train\_array = np.array(mnist\_data['train'])*

*test\_array = np.array(mnist\_data['test'])*

*random\_numbers = np.random.choice(10000,50)*

*test\_data = test\_array[random\_numbers]*

*def KNN(feature, predict, k):*

*X = feature*

*y = predict*

*X\_train, X\_test, y\_train, y\_test = cross\_validation.train\_test\_split(X,y,test\_size = 0.2)*

*classifier = neighbors.KNeighborsClassifier()*

*classifier.fit(X\_train,y\_train)*

*accuracy = classifier.score(X\_test, y\_test)*

*return accuracy*

*def l1distance(instance1, instance2, length):*

*distance = 0*

*for x in range(1,length+1):*

*distance += abs(instance1[x]-instance2[x])*

*return distance*

*def l2distance(instance1, instance2, length):*

*distance = 0*

*for x in range(1,length+1):*

*distance+=pow((instance1[x]-instance2[x]),2)*

*return distance*

*def Neighbors(trainingSet, testInstance, k):*

*distance=[]*

*length = len(testInstance)-1*

*for x in range(len(trainingSet)):*

*dist = euclideanDistance(testInstance, trainingSet[x], length)*

*distance.append((trainingSet[x],dist))*

*distance.sort(key=operator.itemgetter(1))*

*neighbors=[]*

*for x in range(k):*

*neighbors.append(distance[x][0])*

*classVotes={}*

*for x in range(len(neighbors)):*

*response = neighbors[x][0]*

*if response in classVotes:*

*classVotes[response] += 1*

*else:*

*classVotes[response] = 1*

*sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1),reverse=True)*

*return sortedVotes[0][0]*

*def getAccuracy(testSet, predictions):*

*correct=0*

*for x in range(len(testSet)):*

*if testSet[x][0]==predictions[x]:*

*correct+=1*

*return (correct/float(len(testSet))) \*100*

*K = 5 #{1,5,9,13}*

*for x in range(len(test\_data)):*

*result = Neighbors(train\_array, test\_data[x], K)*

*predictions.append(result)*

*accuracy = getAccuracy(test\_data, predictions)*