Zequn Yu
PA3
CSE 151

Q1.

Passes 2, 3, 4 in perceptron:
training errors : 0.04
test errors:0.06

training errors : 0.02
test errors:0.05

training errors : 0.02
test errors:0.05

Passes 2, 3, 4 in voted perceptron:
training errors : 0.04
test errors:0.06

training errors : 0.03
test errors:0.04

training errors : 0.02
test errors:0.05

Passes 2, 3, 4 in averaged perceptron:
training errors : 0.05
test errors:0.08

training errors : 0.03
test errors:0.06

training errors : 0.03
test errors:0.05

Q2.

('The three highest:', ['file'], ['program'], ['line'])
('The three lowest:', ['he'], ['team'], ['game'])

Q3.

(a)

0.80128205 (i = 5, j = 5). Because it means the true label is 5 and the predict label is 5 as well, the rate is 80%. So the classifier 5 has the highest accuracy.

(b)

0.37142857(i = 3, j = 3). Because it means the true label is 3 and the predict label is 3 as well, but the rate is 37%. So the classifier 3 has the least accuracy.

(c)

i = 5, j = 1, 2, 3, 4, 6. It means its predict label is 5, but the error rate is 25%. So the classifier 5 most often mistakenly classifies.

* output:

```
[[0.23243243 0.27604167 0.45142857 0.25543478 0.11538462 0.34259259]
 [0.71891892 0.00520833 0.03428571 0.02173913 0.         0.        ]
 [0.01081081 0.65625    0.03428571 0.02717391 0.01282051 0.01851852]
 [0.         0.015625   0.37142857 0.         0.         0.02777778]
 [0.01621622 0.00520833 0.         0.69021739 0.         0.        ]
 [0.01621622 0.03125    0.07428571 0.00543478 0.80128205 0.12037037]
 [0.00540541 0.01041667 0.03428571 0.         0.07051282 0.49074074]]
```

```
# Author: Zequn Yu
# PID: A14712777

from __future__ import division
import numpy
from heapq import nlargest, nsmallest
from sklearn.metrics import confusion_matrix
from operator import itemgetter


# read point from .txt
train = numpy.loadtxt("pa3train.txt")
test = numpy.loadtxt("pa3test.txt")
len_feature = len(train[0])-1

# get the subset that label is 1 or 2
sub_train = []
sub_test = []
for point in train:
  if point[len_feature] == 1 or point[len_feature] == 2:
    sub  train.append(point)
for point in test:
  if point[len  feature] == 1 or point[len  feature] == 2:
    sub_test.append(point)

# function for perceptron
def perceptron(train, passes, label):
  # initialize the w
  w = numpy.zeros(len_feature)
  # as the passes passed in
  for p in range(0, passes):
    # check each point
    for feature in train:
      # if the label is what we want, set as 1
      if label == feature[len_feature]:
        new_label = 1
      # else set as -1
      else:
        new_label = -1
      #label = point[feature_len]
        # print("Now the point line is: ", point)
      # check do we need to update
      if label * numpy.dot(w, point[:len_feature]) <= 0:
        w = w + (label * point[:len_feature])
        #print("Now the w is: ", w)
  return w

# check the sign of number
def sign(number):
  if number >= 0 :
    return 1
  else :
    return -1

# function to calculate the error of perceptron
def perceptron  error(test, w, label):
  # set the wrong number
  err = 0
  # for loop to check each loop
  for feature in test:
    # check if the sign match up what we expect (1 or -1)
    sign = numpy.dot(w, feature[:len_feature])
```

```
      # when the sign is - but the label should be 1
      if sign < 0 and (feature[len_feature] == label):
        err += 1
      # when the sign is + but the label should be 2(-1)
      elif sign >= 0 and (feature[len_feature] != label):
        err += 1
    #print("Now the wrong is: ", wrong)
    #print("len(test) is: ", len(test))
    return err / len(test)

# call function to get results
print("pass 2 to pass 4 in perceptron")
for passes in range(1,5):
    w = perceptron(sub  train, passes, 1)
    # print("The final w is: ", w)
    print(perceptron_error(sub_train, w, 1))
    print(perceptron_error(sub_test, w, 1))

# function for voted perceptron
def voted  perceptron(train, passes, label):
    w = numpy.zeros(len_feature)
    cm = 1
    output = []
    # for passes time
    for p in range(0, passes):
      # for each point
      for feature in train:
        # if label match up set it as 1 (label 1)
        if label == feature[len_feature]:
          new_label = 1
        # label 2
        else:
          new_label = -1
        # check if need to update w
        if new_label * numpy.dot(w, feature[:len_feature]) <= 0:
          w = w + (label * feature[:len  feature])
          cm = 1
          # save the (w, c)
          output.append((w, cm))
        else:
          # update cm
          cm += 1
    output.append((w, cm))
    return output

# function to calculate the error
def voted  perceptron  error(test, classifiers, label):
    err = 0
    # for each point in test file
    for feature in test:
      # set test_feature and predict
      test_feature = feature[:len_feature]
      predict = 0
      # for each (w, c) calcualte the predict
      for c in classifiers:
        predict += c[1] * sign(numpy.dot(c[0], test_feature))
      # if doesn't match up then increment the wrong number
      if sign(predict) < 0 and feature[len_feature] == label:
        err += 1
      elif sign(predict) >= 0 and feature[len_feature] != label:
        err += 1
    return err / len(test)

# call function to calculate
print("pass 2 to pass 4 in voted perceptron")
for passes in range(1,5):
    classis = voted_perceptron(sub_train, passes, 1)
```

```
  print(voted_perceptron_error(sub_train, classis, 1))
  print(voted_perceptron_error(sub_test, classis, 1))

# function for averaged perceptron error
def average_perceptron_error(test, classifers, label):
  err = 0
  w = sum([classi[0] * classi[1] for classi in classifers])
  for feature in test:
    sign = numpy.dot(w, feature[:len_feature])
    if  sign < 0 and feature[len_feature] == label:
      err += 1
    elif sign >= 0 and feature[len_feature] != label:
      err += 1
  return err / len(test)

# call function to calculate
print("pass 2 to pass 4 in averaged perceptron")
for passes in range(1,5):
  classis = voted_perceptron(sub_train, passes, 1)
  print(average_perceptron_error(sub_train, classis, 1))
  print(average_perceptron_error(sub_test, classis, 1))


# wavg from averaged perceptron and three passes
# part 2
# call function to get 3 passes voted perceptron
avg_all = train_voted_perceptron(one_two_subset, 3, 1)
#print("Now the classis is: ", classis)
w = sum([avg_each[0] * avg_each[1] for avg_each in avg_all])

# use nlargest and nsmallest to get 3 highest and the three lowest
# get the index of 3 max and min value in pair
largest = nlargest(3, enumerate(w), key=lambda x: x[1])
lowest = nsmallest(3, enumerate(w), key=lambda x: x[1])
#print("largest is: ", top)
#print("lowest is: ", bot)

# set to save diction
diction = []

# read ling by line
for line in open("pa3dictionary.txt", 'r'):
  diction.append(line.strip().split('/n'))

# get the answer
print("The three highest:", diction[largest[0][0]], diction[largest[1][0]], diction[largest[2][0]])
# print(max(classis,key=itemgetter(1))[0])
# print(diction[max(classis,key=itemgetter(1))[0]])
print("The three lowest:", diction[lowest[0][0]], diction[lowest[1][0]], diction[lowest[2][0]])



# part 3: one vs all
# for each class i : 1 to 6 run 1 pass

class_one = train_perceptron(train, 1, 1)
class_two = train_perceptron(train, 1, 2)
class_three = train_perceptron(train, 1, 3)
class_four = train_perceptron(train, 1, 4)
class_five = train_perceptron(train, 1, 5)
class_six = train_perceptron(train, 1, 6)

# if ci(x) = i, then predict label i  ( exactly one i )
# if ci(x) = i ( more than one i ) or ci(x) = i ( no i , Don't know )
CX = []
CX.append(class_one)
CX.append(class_two)
CX.append(class_three)
```

```
CX.append(class_four)
CX.append(class_five)
CX.append(class  six)
# print("CX is: ", CX)

'''
For each test data, you will have {predicted label, true label}.
The predicted label can be {1, 2, 3, 4, 5, 6, Don't know} => 7 values
The true label can be {1, 2, 3, 4, 5, 6} => 6 values
'''
# function to predict the test example
def test_predict( test, CX ):
  # print("CX: ", CX )
  pred  result = []
  # for each line of feathre in testData
  for feature in test:
    #print("Now the feature is: ", feature)
    # set label as
    tmp_label = 0
    # loop to check each feature
    for i in range(0, len(CX)):
      if numpy.dot(CX[i], feature[0:-1]) > 0:
        print("legal")
        ''' if has more than one label -> Don't know '''
        if tmp_label != 0:
          print("Dont know")
          tmp_label = 0
          break
        else:
          ''' set the label '''
          print("Get label")
          tmp_label = i + 1
        ''' add in to result '''
    print("tmp_label: ", tmp_label)
    pred_result.append(tmp_label)
  return pred  result



# build confusion matrix
pred_l = test_predict( test, CX )
''' set the true label'''
print("pred_l:", pred_l)
true_l = []
for tf in test:
  true_l.append(tf[feature_len])
print("true_l: ", true  l)
my_matrix = confusion_matrix( pred_l, true_l)

print("my_martixL ")
print(my_matrix)

''' set the number of label '''
test_label = [0,0,0,0,0,0,0]
for tf in test:
  tl_idx = int(tf[feature_len])
  test  label[tl  idx] += 1
res_matrix = numpy.zeros((7,6))

''' calculate C/N for each entry '''
for r in range(0, 7):
  for c in range(1, 7):
    print("My mat: ", my_matrix[r][c] )
    print("test_m: ", test_label[c] )
    res_matrix[r][c-1] = my_matrix[r][c] / test_label[c]
```

```
print("Now con_martix is: ")
print(res_matrix)
```