Alex Meza
a13208068
10/31/17

Cse 158, Homework 2:

Question 1:

Accuracy of Validation, and Testing Sets: 0.90028199436 0.577788444231

Question 2:

(no work to show)

Question 3:

False Positives: 10465 False Negatives: 106 True Positives: 5806 True Negatives: 290
Ber of classifier made from keywords in review is: 0.4954827159724935

Question 4:

Performance of Optimum Training Prediction: 0.561442457698
Performance of Optimum Validation Prediction: 0.95122097558
Performance of Optimum Testing Prediction: 0.357472850543

Question 5:

PCA components of Xtrain:
```
 [[  2.91553267e-04   3.36316078e-03  -4.92016461e-03   1.22605289e-02
    8.02994923e-01   2.01250020e-04   5.90516350e-01   7.22369875e-02
    1.75711994e-04   3.29456581e-02]
 [ -2.36616086e-03  -8.55889699e-03  -1.91717318e-02   1.45794087e-02
   -5.92410182e-01   4.74330032e-04   8.05130166e-01  -4.97366274e-03
   -1.28579556e-03   1.14021305e-02]
 [  3.99780701e-03   4.58510040e-02   1.01611015e-01   1.87044464e-03
   -6.24232301e-02  -5.23321160e-05  -3.73196976e-02   9.90698551e-01
    9.18564879e-04   2.79198868e-02]
 [ -1.52012269e-04   2.00218881e-02  -1.04163870e-02   2.44992682e-02
   -1.80912429e-02   8.60712007e-05  -2.81132009e-02  -3.02427712e-02
    2.45592725e-03   9.98424798e-01]
 [  2.55382520e-02   2.24799457e-01   9.67585118e-01   7.19960887e-03
   -2.31187652e-03   9.62965425e-03   2.29568098e-02  -1.09118707e-01
    9.38463609e-04   2.70997623e-03]
 [  3.54769091e-02   9.72027558e-01  -2.29341823e-01   1.34748385e-02
   -4.20493294e-03   8.77575267e-03   1.83813401e-03  -2.11898082e-02
    7.29985353e-03  -2.28954543e-02]
 [  5.66835271e-03  -1.55377546e-02  -3.62210660e-03   9.99363327e-01
   -5.81954419e-04   5.54584002e-03  -1.84164136e-02  -8.50413772e-04
    2.59682946e-04  -2.48036199e-02]
 [  9.96955075e-01  -4.09453361e-02  -1.74561630e-02  -6.65912506e-03
   -1.16153723e-03   6.34885323e-02   1.28440550e-03  -3.69526311e-04
   -4.48757759e-03   9.63660068e-04]
 [ -6.40484881e-02  -7.96385717e-03  -6.17769008e-03  -5.32627798e-03
    2.60276502e-04   9.97852694e-01  -7.26121605e-04   1.31450704e-03
   -7.44873354e-03   1.72584485e-04]
 [  3.70628217e-03  -7.64907081e-03   5.51204371e-04  -4.79523814e-04
   -7.71354670e-04   7.64404986e-03   1.00504146e-03  -5.89435260e-04
    9.99930789e-01  -2.29235033e-03]]
```

Questin 6:

First data point after dimentionality reduction:
```
 [ -6.73224036e-01  -1.02494007e-03  -4.42142073e-02  -4.08818841e-02
   2.22068793e-01   1.02936537e+00  -1.55353909e-02   1.95071920e+00
  -1.38498514e-01  -2.07846559e-03]
```

Question 7:

Error of using only 2 components: 8346.76973501

Question 8:
(See Plot)

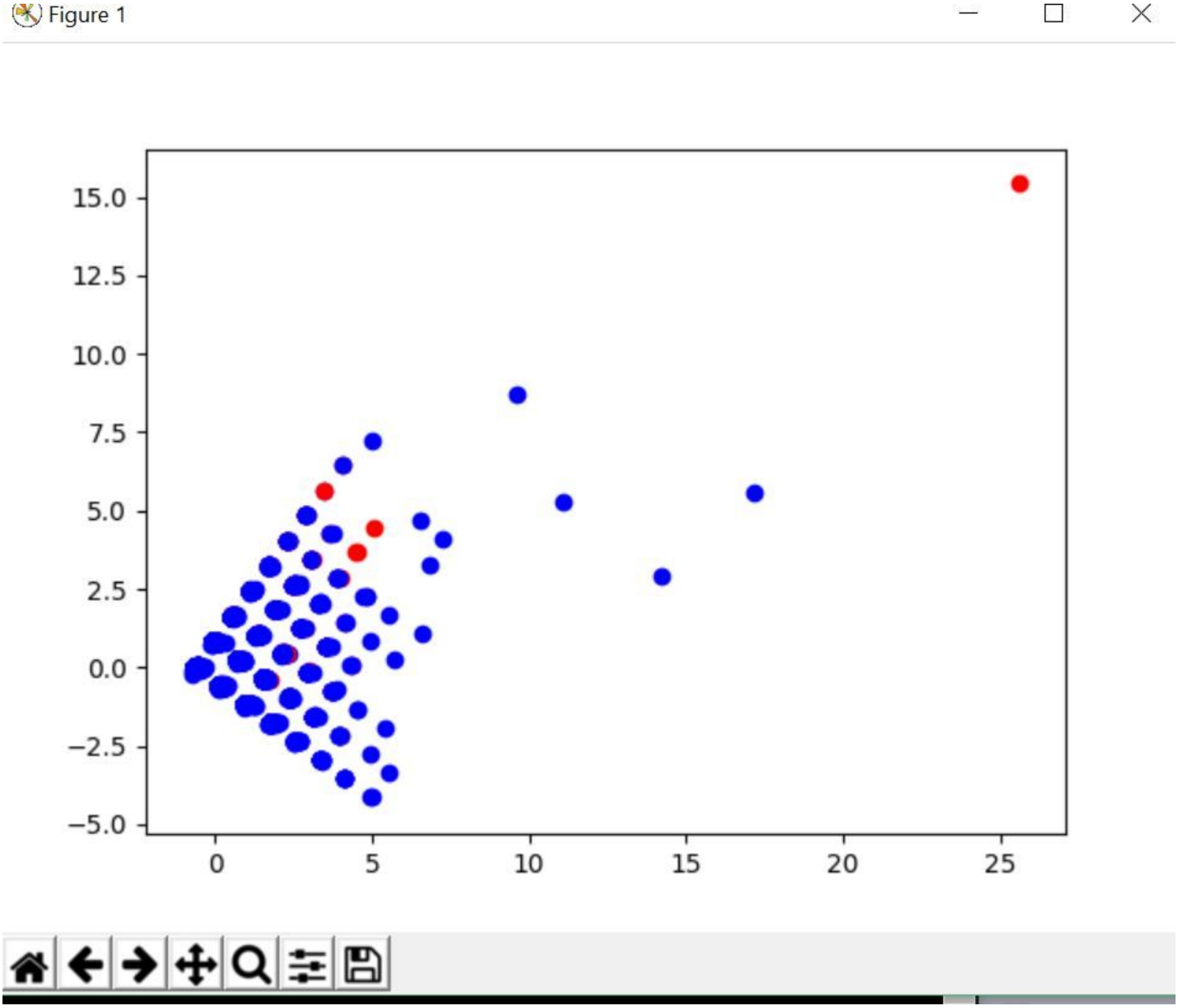
Figure 1
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0
5
10
15
20
25

```
#!/usr/bin/env python

"""
Filename: hw2.py
Author: Alex Meza
Date: 10/30/17
Description:
      ...
"""


import numpy as np
import pandas as pd
import scipy.optimize
import re
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from math import log
from math import exp

def main():

      #pre stored data base of beers
      data = pd.read_pickle("./beer.pkl")

      #for question 8
      ipa = [row['beer/style']=="American IPA" for ind,row in
data.iterrows()]
      train_ipa = ipa[:int(len(ipa)/3)]
      #create feature matrix and label for if beer has abv>=6.5%

#QUESTION 1
      print("Question 1:\n")
      #Create feature matrix and create pickle for easy access
      X = [feature(row) for index, row in data.iterrows()]
      X = np.array(X)
      pd.DataFrame(X).to_pickle("./q1Features")

      y = [row['beer/ABV'] >= 6.5 for index, row in data.iterrows()]

      #read pickle of data so dont have to recreate matrix

      X = pd.read_pickle("./q1Features")
      X = np.array(X)
      #create training, validation, and test sets
      Xtrain = X[:int(len(X)/3)]
      Xval = X[int(len(X)/3):int(2*len(X)/3)]
      Xtest = X[int(2*len(X)/3):]

      ytrain = y[:int(len(y)/3)]
      yval = y[int(len(y)/3):int(2*len(y)/3)]
      ytest = y[int(2*len(y)/3):]

      lam = 1
      theta = train(Xtrain, ytrain, lam)

      acc1 = performance(Xval.dot(theta), yval)
      acc2 = performance(Xtest.dot(theta), ytest)
      print('Accuracy of Validation, and Testing Sets:',acc1, acc2)
```

```
#QUESTION #2
      print("\nQuestion 2:\n")
      #create feature matrix of occurences of keywords in each review
      #X = [reviewFeature(row['review/text'].lower()) for ind, row in
data.iterrows()]
      #X = np.array(X)
      #original command to create pickle for easier access
      #pd.DataFrame(X).to_pickle("./reviewMatrix.pkl")

      #read dataframe from pickle(SERIOUSLY SO MUCH FASTER)
      X = pd.read_pickle("./reviewMatrix.pkl")
      X = np.array(X)

      #create testing, training, val, sets

      Xtrain = X[:int(len(X)/3)]
      Xval = X[int(len(X)/3):int(2*len(X)/3)]
      Xtest = X[int(2*len(X)/3):]

      #create classifier based on if “lactic,” “tart,” “sour,” “citric,”
“sweet,” “acid,” “hop,” “fruit,” “salt,” “spicy.” are in the review
      lam =1
      theta = train(Xtrain, ytrain, lam)
      theta = np.array(theta)
#Question 3
      print("\nQuestion 3:\n")
      #print out the balanced errorrate of the test
      ber = BER(Xtest.dot(theta), ytest)
      print("Ber of classifier made from keywords in review is: ", ber)

#Question 4
      print("\nQuestion 4:\n")
      C = [0, .01, .1, 1, 10, 100]

      theta = [train(Xtrain, ytrain, c) for c in C]
      performances = [performance(Xval.dot(t), yval) for t in theta]
      index_of_max = performances.index(max(performances))
      optimum_theta = theta[index_of_max]

      train_performance = performance(Xtrain.dot(optimum_theta), ytrain)
      val_performance = max(performances)
      test_performance = performance(Xtest.dot(optimum_theta), ytest)

      print("Performance of Optimum Training Prediction: ",
train_performance)
      print("Performance of Optimum Validation Prediction: ",
val_performance)
      print("Performance of Optimum Testing Prediction: ",
test_performance)


#Question 5
      print("\nQuestion 5:\n")
      X = X[:, 1:]
      Xtrain = X[:int(len(X)/3)]
      Xval = X[int(len(X)/3):int(2*len(X)/3)]
      Xtest = X[int(2*len(X)/3):]

      pca = PCA(n_components=10)
      pca.fit(Xtrain)
```

```
      print("PCA components of Xtrain:\n",pca.components_)

#Question 6
      print("\nQuestin 6:\n")
      X0 = pca.transform(Xtrain)
      print("First data point after dimentionality reduction:\n", X0[0])

#Question 7
      print("\nQuestion 7:\n")
      dim_reduct_error = sum(pca.explained_variance_[2:])*len(Xtrain)
      print("Error of using only 2 components: ", dim_reduct_error)


#Problem 8
      print("\n Question 8: \n")
      pca = PCA(n_components=2)

      X_reduct_train = pca.fit_transform(Xtrain).tolist()
      for (is_ipa, [x,y]) in zip(train_ipa, X_reduct_train):
            if(is_ipa):
                  plt.scatter(x,y,c='r')
            else:
                  plt.scatter(x,y,c='b')

      plt.show()
def train(X_train, y_train, lam):
      theta,_,_ = scipy.optimize.fmin_l_bfgs_b(f, [0]*len(X_train[0]),
fprime, pgtol = 10,     args = (X_train, y_train, lam))
      return theta

#performance of method
def performance(scores, y):
      predictions = [s > 0 for s in scores]
      correct = [(a==b) for (a,b) in zip(predictions,y)]
      acc = sum(correct) * 1.0 / len(correct)

      return acc
def BER(scores, y):

      predictions = [s >0  for s in scores]
      falsepos, falseneg, truepos, trueneg =0, 0,0,0
      for (a,b) in zip(predictions,y):
            if(a==b):
                  if(a==True):
                        truepos+=1
                  else:
                        trueneg+=1
            elif(a==True):
                  falsepos+=1
            else:
                  falseneg+=1
      print ("False Positives:" ,falsepos," False Negatives:", falseneg,
" True Positives: ", truepos, " True Negatives: ", trueneg)
      return
.5*((falseneg/(falseneg+truepos)+falsepos/(falsepos+trueneg)))

#get review feature vector for feature matrix
def reviewFeature(review):

      occurences =[]
```

```
      keywords = ["lactic","tart","sour","citric", 'sweet', 'acid',
'hop','fruit', 'salt', 'spicy']
      occurences.append(1)

      for word in keywords:
            count = sum(1 for _ in re.finditer(r'\b%s\b' %
re.escape(word), review))
            occurences.append(count)

      return occurences

#get feature for feature matrix
def feature(datum):
      feat = [1, datum['review/taste'], datum['review/appearance'],
datum['review/aroma'], datum['review/palate'], datum['review/overall']]
      return feat

# NEGATIVE Log-likelihood
def f(theta, X, y, lam):
      loglikelihood = 0
      for i in range(len(X)):
            logit = inner(X[i], theta)
            loglikelihood -= log(1 + exp(-logit))
            if not y[i]:
                  loglikelihood -= logit
      for k in range(len(theta)):
            loglikelihood -= lam * theta[k]*theta[k]
      # for debugging
      # print("ll =" + str(loglikelihood))
      return -float(loglikelihood)

# NEGATIVE Derivative of log-likelihood
def fprime(theta, X, y, lam):
      dl = [0]*len(theta)
      for i in range(len(X)):
            logit = inner(X[i], theta)
            for k in range(len(theta)):
                  dl[k] += X[i][k] * (1 - sigmoid(logit))
                  if not y[i]:
                        dl[k] -= X[i][k]
      for k in range(len(theta)):
            dl[k] -= lam*2*theta[k]
      return np.array([-x for x in dl])


def inner(x,y):
      return sum([x[i]*y[i] for i in range(len(x))])

def sigmoid(x):
      return 1.0 / (1 + exp(-x))

if __name__ == '__main__':
      main()
```