Homework 2

Wenjun Zhang

A53218995

Problems:

1. Using the provided stub, after randomly reshuffling the data, the accuracy performance of the train/validation/test set for $\lambda \in \{0, 0.01, 1.0, 100.0\}$. are:

```
lambda = 0;     train=0.738357843137; validate=0.752602571953; test=0.758726270667
lambda = 0.01;  train=0.737745098039; validate=0.751377832211; test=0.757501530925
lambda = 1.0;   train=0.729166666667; validate=0.744641763625; test=0.740355174525
lambda = 100.0; train=0.665441176471; validate=0.679730557257; test=0.687691365585
```

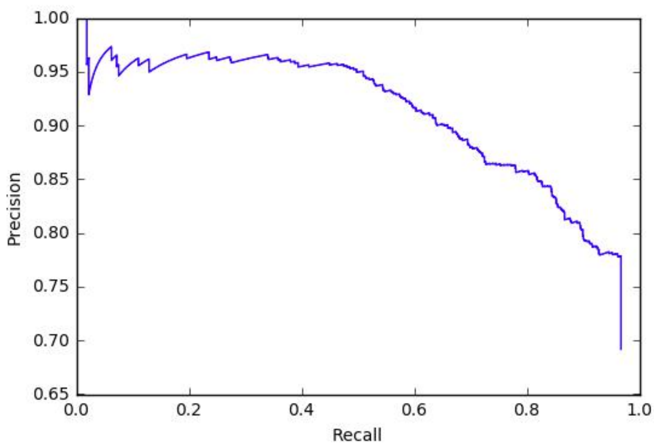
2. Performing experiments with $\lambda = 0.01$ and use only the test set from the original 1/3, 1/3, 1/3 split for this question, the number of true positives, true negatives, false positives, false negatives, and the Balanced Error Rate of the classifier are:

```
true_positive =  1129
true_negative =  145
false_positive =  321
false negative =  38
Balanced Error Rate =  0.360701663412
```

3. By sorting the predictions by confidence, the precision and the recall when returning the top 10, 500, and 1000 predictions we compute are:

```
When returning the top  10 predictions, Precision =  1.0 Recall =  0.00856898029135
When returning the top  500 predictions, Precision =  0.956 Recall =  0.409597257926
When returning the top  1000 predictions, Precision =  0.864 Recall =  0.740359897172
```

4. Plot precision versus recall as the number of results considered varies (from 1 to len(y_test)).



5. If we compress our data by replacing each of the points with their mean vector, the 'reconstruction error' defined as:

$$\sum_{x \in X} \|\bar{x} - x\|_2^2$$

The reconstruction error for the compressed data is :

```
Reconstruction Error =  [[ 3675818.61687812]]
```

In the code we set the number of dimension to be 12 since t we have added an additional feature for the bias.

6. Using the week 3 code, PCA components (i.e., the transform matrix) we find is:

```
[[  0.00000000e+00  -3.23636346e-04   1.42201752e-04   3.17030713e-04
    5.36390435e-02   9.30284526e-05   2.54030965e-01   9.65655009e-01
    3.19990241e-05  -2.95831396e-04   3.84043646e-04  -1.00526693e-02]
 [  0.00000000e+00   7.57985623e-03   1.66366340e-03  -1.04742899e-03
   -5.21677266e-02  -4.49425600e-05  -9.65020304e-01   2.56793964e-01
   -7.90089050e-06  -5.24900596e-04   1.09699394e-03   2.89827657e-03]
 [  0.00000000e+00  -1.82124420e-02  -2.54680710e-03  -3.31838657e-03
   -9.93221259e-01   1.51888372e-04   6.42297821e-02   3.91682592e-02
   -4.30929482e-04   6.93199060e-03   2.85216045e-03   8.62920933e-02]
 [  0.00000000e+00  -1.56811999e-01  -3.28220652e-03  -1.66866136e-02
   -8.28549640e-02   6.91822288e-03  -1.13029682e-03  -5.39110108e-03
    9.49080503e-04  -2.68027305e-03  -1.30498102e-03  -9.83955205e-01]
 [  0.00000000e+00   9.81360642e-01  -1.45890108e-02   5.92643662e-02
   -3.17546064e-02   5.07483182e-04   8.43759364e-03  -1.77578042e-03
    6.03725221e-04  -9.05011239e-02  -9.35630845e-03  -1.54417839e-01]
 [  0.00000000e+00  -7.76578401e-02   2.37665885e-01  -2.23406619e-02
   -5.04113878e-03   1.43564098e-02   2.14210997e-04   2.22913844e-04
   -3.36617054e-03  -8.77254205e-01  -4.08570175e-01   1.54145486e-02]
 [  0.00000000e+00  -7.36289612e-02  -2.61563804e-01   9.43067566e-01
   -2.14514264e-03   1.19104298e-02  -1.68808905e-03   1.42294158e-04
   -1.17203197e-04  -1.45895558e-01   1.23868963e-01  -2.88797236e-03]
 [  0.00000000e+00   1.37617196e-02  -2.11129619e-01   1.16514121e-01
   -5.30670319e-04  -1.05181628e-02  -1.36446528e-03   8.21179429e-04
   -3.09221855e-04   3.58358431e-01  -9.01728510e-01  -3.27758247e-03]
 [  0.00000000e+00   1.74575775e-02   9.10890084e-01   3.04081497e-01
   -2.89763923e-03   2.34615054e-02   1.17406025e-03  -3.85957239e-04
    1.23176271e-03   2.68927937e-01  -6.70756658e-02  -1.12101920e-02]
 [  0.00000000e+00   2.31513441e-03  -2.38717789e-02  -1.67445603e-02
    8.92206499e-04   9.99462734e-01  -9.81109101e-05  -3.32812875e-05
    4.14235255e-03   1.18483756e-02  -3.51543098e-03   6.92344110e-03]
 [ -0.00000000e+00   7.48312160e-04   3.08204153e-04   2.55232500e-04
    3.49846801e-04   4.12943179e-03  -6.96565372e-06   4.16951216e-06
   -9.99984215e-01   3.17948604e-03   1.53436134e-03  -1.10029138e-03]]
```

Because we have added an additional feature for the bias in the front, and its variance is 0, there is a 0 in each row of the transfer matrix.

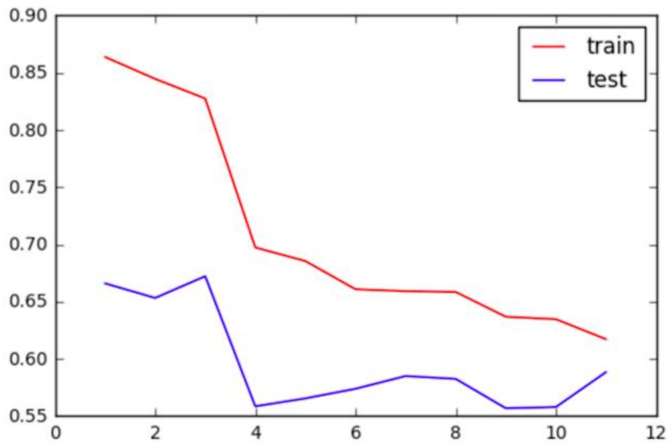
7. If we compress the data using just four PCA dimensions, the reconstruction error is :

```
When 4 dimensional PCA, Reconstruction Error =  [[ 1345.4755741]]
```

8. Train a simple linear regressor (no regularization, using the training set) to predict the quality score, using increasingly many PCA dimensions as below:

$$
\begin{aligned}
\text{model 1:} \quad \text{quality} &= \theta_0 + \theta_1 \times (\text{first pca dimension}) \\
\text{model 2:} \quad \text{quality} &= \theta_0 + \theta_1 \times (\text{first pca dimension}) + \theta_2 \times (\text{second pca dimension}) \\
\textit{etc.} &
\end{aligned}
$$

The plot of how MSE changes (on the train and test sets) as more and more dimensions are used is:



Appendix (some important parts):

Q1:

```
# randomly shuffle the data
random.shuffle(lines)
```

```
# 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 =", loglikelihood
  return -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 numpy.array([-x for x in dl])
```

```
X_train = X[:int(len(X)/3)]
y_train = y[:int(len(y)/3)]
X_validate = X[int(len(X)/3):int(2*len(X)/3)]
y_validate = y[int(len(y)/3):int(2*len(y)/3)]
X_test = X[int(2*len(X)/3):]
y_test = y[int(2*len(X)/3):]
```

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

```
# Predict
def performance(theta):
  scores_train = [inner(theta,x) for x in X_train]
  scores_validate = [inner(theta,x) for x in X_validate]
  scores_test = [inner(theta,x) for x in X_test]

  predictions_train = [s > 0 for s in scores_train]
  predictions_validate = [s > 0 for s in scores_validate]
  predictions_test = [s > 0 for s in scores_test]

  correct_train = [(a==b) for (a,b) in zip(predictions_train,y_train)]
  correct_validate = [(a==b) for (a,b) in zip(predictions_validate,y_validate)]
  correct_test = [(a==b) for (a,b) in zip(predictions_test,y_test)]

  acc_train = sum(correct_train) * 1.0 / len(correct_train)
  acc_validate = sum(correct_validate) * 1.0 / len(correct_validate)
  acc_test = sum(correct_test) * 1.0 / len(correct_test)
  return acc_train, acc_validate, acc_test
```

```
# Validation pipeline
for lam in [0, 0.01, 1.0, 100.0]:
  theta = train(lam)
  acc_train, acc_validate, acc_test = performance(theta)
  print("lambda = " + str(lam) + ";\ttrain=" + str(acc_train) + "; validate=" +
str(acc_validate) + "; test=" + str(acc_test))
```

Q2:

```
# Predict
def otherperformance(theta):
  scores_test = [inner(theta,x) for x in X_test]
  predictions_test = [s > 0 for s in scores_test]

  label_positive = sum(y_test)
  label_negative = len(y_test) - label_positive
  true_positive = sum([(a==1 and b==1) for (a,b) in
zip(predictions_test,y_test)])
  true_negative = sum([(a==0 and b==0) for (a,b) in
zip(predictions_test,y_test)])
  false_positive = sum([(a==1 and b==0) for (a,b) in
zip(predictions_test,y_test)])
  false_negative = sum([(a==0 and b==1) for (a,b) in
zip(predictions_test,y_test)])
  BER =
(false_negative*1.0/label_positive+false_positive*1.0/label_negative)/2
  return true_positive, true_negative, false_positive, false_negative, BER
```

```
# Other performances
theta2 = train(0.01)
true_positive, true_negative, false_positive, false_negative, BER =
otherperformance(theta2)
print "true_positive = ", true_positive
print "true_negative = ", true_negative
```

```
print "false_positive = ", false_positive
print "false negative = ", false_negative
print "Balanced Error Rate = ", BER
```

Q3:

```
scores_test = [inner(theta2,x) for x in X_test]
predictions_test = [s > 0 for s in scores_test]
relevant_documents = sum(y_test)
confidence = [[scores_test[i],y_test[i]] for i in range(len(y_test))]
```

```
confidence.sort(key=lambda a:a[0], reverse = True)
```

```
def percentages(number):
  relevant_retrieved=0
  for l in range(number):
    if confidence[l][0]*confidence[l][1] > 0:
      relevant_retrieved+=1
  precision = relevant_retrieved * 1.0 / number
  recall = relevant_retrieved *1.0 / relevant_documents
  return precision, recall
```

```
for number in [10, 500, 1000]:
    precision, recall = percentages(number)
    print "When returning the top ", number, "predictions, Precision = ", precision,
"Recall = ", recall
```

Q4:

```
recall_set=[]
precision_set=[]
for number in range(1,len(y_test)):
    precision, recall = percentages(number)
    recall_set.append(recall)
    precision_set.append(precision)
```

```
plt.plot(recall_set,precision_set)

plt.show()

plt.xlabel(u'Recall')

plt.ylabel(u'Precision')
```

Q5:

```
X_mean=[]

for i in range(12):

    X_temp =0

    for j in range(len(X_train)):

        X_temp+= X_train[j][i]

    X_mean_temp = X_temp / len(X_train)

    X_mean.append(X_mean_temp)

X_error=[[0] * 12 for row in range(len(X_train))]

for i in range(12):

    for j in range(len(X_train)):

        X_error[j][i] = X_train[j][i] - X_mean[i]

X_error = numpy.matrix(X_error)

reconstruction_error = 0

for j in range(len(X_train)):

    reconstruction_error+= X_error[j]*X_error[j].T

print "Reconstruction Error = ",reconstruction_error
```

Q6:

```
pca = PCA(n_components=11)

pca.fit(X_train)

print pca.components_
```

Q7:

```
reconstruction_error = 0

phi = numpy.matrix(pca.components_)

for j in range(4,11):

    reconstruction_error+= phi[j]*X_error.T*X_error*phi[j].T

print "When 4 dimensional PCA, Reconstruction Error = ",reconstruction_error
```

Q8:

```
def feature(X,number):
    pca = PCA(n_components = number)
    pca.fit(X)
    X_trans = pca.transform(X)
    feat = [[1] for row in range(len(X_trans))]
    X_trans = numpy.column_stack((feat, X_trans))
    return X_trans
```

```
def msecomp(theta, X, y):
  theta = numpy.matrix(theta).T
  X = numpy.matrix(X)
  y = numpy.matrix(y).T
  diff = X*theta - y
  diffSq = diff.T*diff
  mse = diffSq / len(y)
  return mse.tolist()
```

```
x = []
mse1 = []
mse2 = []
for number in range(1,12):
    x.append(number)
    X1 = feature(X_train, number)
    X2 = feature(X_test, number)
    theta,residuals,rank,s = numpy.linalg.lstsq(X1, y_train)
    mse1.append(msecomp(theta, X1, y_train)[0][0])
    mse2.append(msecomp(theta, X2, y_test)[0][0])
plt.plot(x, mse1, color='red', label = 'train')
plt.plot(x, mse2, color='blue', label = 'test')
plt.legend()
plt.show()
```