ASSIGNMENT 2 A53226608

VIDYA CHOCKALINGAM

Question 1

Shuffling the data before splitting into train and test.

```
In [101]:
dataFile = open("C:/Users/BHEL/Desktop/Recommendation Systems/Assignment 1/winequality-white.csv")
header = dataFile.readline()
fields = ["constant"] + header.strip().replace('"','').split(';')
featureNames = fields[:-1]
labelName = fields[-1]
lines = [[1.0] + [float(x) for x in l.split(';')] for l in dataFile]
shuffle(lines)
X = [l[:-1] for l in lines]
y = [l[-1] > 5 for l in lines]

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))
```

The accuracy for different values of lambda.

```
In [105]: ###################################################
# 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))

lambda = 0;     train=0.753676470588; validate=0.734843845683; test=0.746478873239
lambda = 0.01;  train=0.75306372549; validate=0.734231475811; test=0.746478873239
lambda = 1.0;   train=0.738970588235; validate=0.719534598898; test=0.73790569504
lambda = 100.0; train=0.667279411765; validate=0.649724433558; test=0.681567666871
```

Question 2

Code snippets and answers for q2

```python
def performance(theta):

  scores_test = [inner(theta,x) for x in X_test]
  predictions_test = [s > 0 for s in scores_test]
  tp = [(a==1 and b==1) for (a,b) in zip(predictions_test,y_test)]
  tn = [(a==0 and b==0) for (a,b) in zip(predictions_test,y_test)]
  fp = [(a==1 and b==0) for (a,b) in zip(predictions_test,y_test)]
  fn = [(a==0 and b==1) for (a,b) in zip(predictions_test,y_test)]

  #acc_test = sum(correct_test) * 1.0 / len(correct_test)
  return sum(tp),sum(tn),sum(fp),sum(fn)
```

In [19]:
```python
##################################################
# Validation pipeline                            #
##################################################
theta = train(0.01)
tp,tn,fp,fn = performance(theta)
tpr = tp/(tp+fn)
tnr = tn/(tn+fp)
ber = 1 - (1/2)*(tpr+tnr)
print("True Positives=" + str(tp) + "; True Negatives=" + str(tn) + "; False Positives=" + str(fp) + "; False Negatives=" + str(fı
    "; Balanced Error Rate=" + str(ber) )
```

```
True Positives=1129; True Negatives=145; False Positives=321; False Negatives=38; Balanced Error Rate=0.360701663412
```

Question 3

For precision and recall, the denominator and the numerator values are taken from the actual values in the dataset after arranging it in descending order wrt the predicted scores.

```python
def performance(theta):

  scores_test = [inner(theta,x) for x in X_test]
  predictions_test = [s > 0 for s in scores_test]
  correct_predictions = [a==b for (a,b) in zip(predictions_test,y_test)]
  ranking_df = pd.DataFrame(
    {
        "actual" : y_test,
        "predictions": predictions_test,
        "scores":scores_test,
        "correct_predictions": correct_predictions
    })
  ranking_sorted_df = ranking_df.sort_values(by=["scores"], ascending=[0])
  #acc_test = sum(correct_test) * 1.0 / len(correct_test)
  return ranking_sorted_df

def recall(k):
    result_top = result.head(k)
    total_relavant = sum(result["actual"])
    total_relavant_returned = sum(result_top["actual"])
    precision = total_relavant_returned/k
    recall_val = total_relavant_returned/total_relavant
    return precision, recall_val
```

In [53]:
```python
##################################################
# Validation pipeline                            #
##################################################
theta = train(0.01)
result = performance(theta)
for k in [10,500,1000]:
    precision, recall_val = recall(k)
    print("k:" + str(k) + "; precision:" + str(precision) + "; recall:" + str(recall_val) )
```

```
k:10; precision:1.0; recall:0.00856898029135
k:500; precision:0.956; recall:0.409597257926
k:1000; precision:0.864; recall:0.740359897172
```

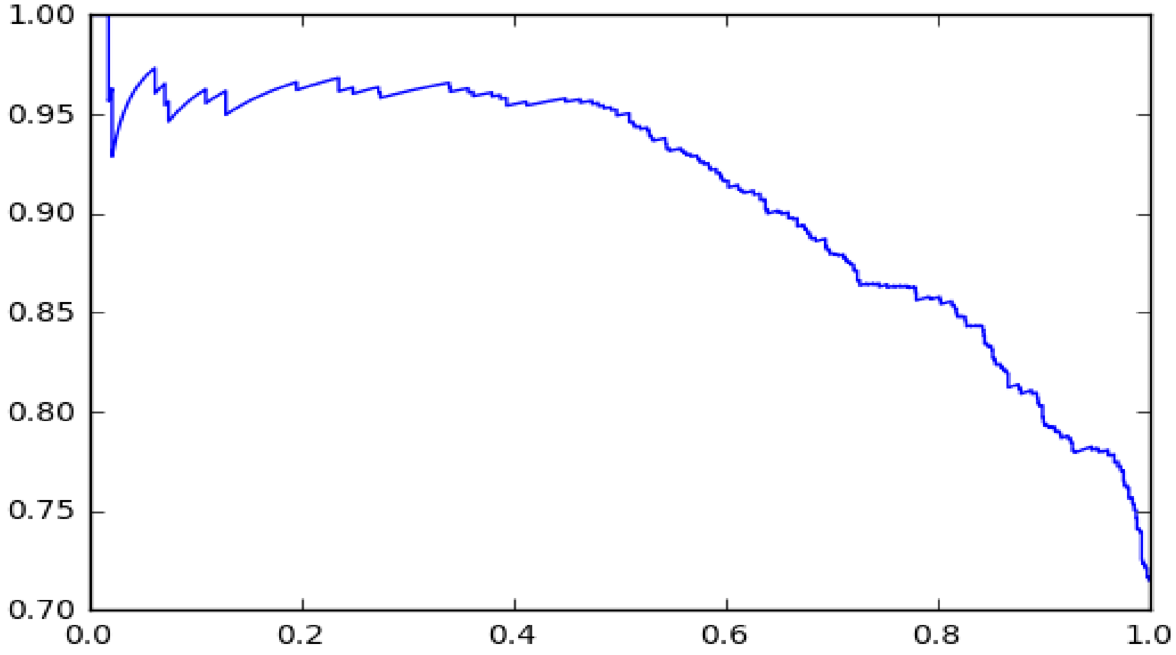
Question 4

Using the same recall function from the previous question and adding the following code,

In [10]:
```python
##################################################
# Validation pipeline                            #
##################################################
theta = train(0.01)
result = performance(theta)
x_plot = []
y_plot = []
for k in range(1,len(y_test)):
    precision, recall_val = recall(k)
    if(k == 1):
        x_plot.insert(0,recall_val)
        y_plot.insert(0,precision)
    else:
        x_plot.append(recall_val)
        y_plot.append(precision)
    #print("k:" + str(k) + "; precision:" + str(precision) + "; recall:" + str(recall_val) )
```

In [16]:
```python
x_axis = numpy.asarray(x_plot)
y_axis = numpy.asarray(y_plot)
```

In [15]:
```python
plt.plot(x_axis, y_axis)
plt.show()
```



Question 5

Subtracting each value of the every feature column from the column's corresponding mean and squaring it to get the error,

In [46]:
```
#Reconstruction error is the sum of variance for each column
Xtrain = pd.DataFrame(X_train)
error = 0
for col in Xtrain:
    error += (Xtrain[col]-numpy.mean(Xtrain[col]))**2
print(sum(error))
```

```
3675818.61688
```

Question 6

Removing the bias term from the X_train variable and using the following code,

```
In [82]: pca = PCA(n_components=11)
         pca.fit(X_train)
         print(pca.components_)
```

```
[[ 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]
 [ -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]
 [  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]
 [ -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]
 [ -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]
 [ -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]
 [ -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]
 [  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]
 [ -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]
```

The full PCA component list,

```
[[ 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]
 [ -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]
 [  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]
 [ -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]
 [ -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]
 [ -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]
 [ -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]
 [  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]
 [ -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]
 [  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]
 [  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]]
```

Question 7

Removing the bias term from the X_train list and using two different ways to check the inverse transform is correct,

```
In [18]: pca = PCA(n_components=4)
         pca.fit(X_train)
         scores = pca.transform(X_train)
         # Checking reconstruction via two methods
         ####X_re = numpy.inner(X_train-pca.mean_, numpy.inner(pca.components_.T, pca.components_.T)) + pca.mean_
         ####print(X_re[0])
         X_reconstruct = pca.inverse_transform(scores)
         ####print(X_reconstruct[0])
         X_diff = (X_train - X_reconstruct)**2
         numpy.sum(X_diff)

Out[18]: 1345.4755741010035
```

Question 8

Removing the conversion of quality variable to either 0 or 1.

Then use the following code to progressively add pca dimensions for train and test datasets.

```
In [29]: pca = PCA(n_components=11)
         pca.fit(X_train)
         print("Training Data")
         #1
         pca_dim1 = numpy.dot(X_train,pca.components_[0])
         X = numpy.vstack([numpy.ones(len(X_train)),pca_dim1]).T
         theta,residuals,rank,s = numpy.linalg.lstsq(X, y_train)
         print(residuals/len(y_train))

         #2
         pca_dim2 = numpy.dot(X_train,pca.components_[1])
         X = numpy.vstack([numpy.ones(len(X_train)),pca_dim1,pca_dim2]).T
         theta,residuals,rank,s = numpy.linalg.lstsq(X, y_train)
         print(residuals/len(y_train))

         #3
         pca_dim3 = numpy.dot(X_train,pca.components_[2])
         X = numpy.vstack([numpy.ones(len(X_train)),pca_dim1,pca_dim2,pca_dim3]).T
         theta,residuals,rank,s = numpy.linalg.lstsq(X, y_train)
         print(residuals/len(y_train))

         #4
         pca_dim4 = numpy.dot(X_train,pca.components_[3])
         X = numpy.vstack([numpy.ones(len(X_train)),pca_dim1,pca_dim2,pca_dim3,pca_dim4]).T
         theta,residuals,rank,s = numpy.linalg.lstsq(X, y_train)
         print(residuals/len(y_train))

         #5
         pca_dim5 = numpy.dot(X_train,pca.components_[4])
         X = numpy.vstack([numpy.ones(len(X_train)),pca_dim1,pca_dim2,pca_dim3,pca_dim4,pca_dim5]).T
         theta,residuals,rank,s = numpy.linalg.lstsq(X, y_train)
```

```
#6
pca_dim6 = numpy.dot(X_train,pca.components_[5])
X = numpy.vstack([numpy.ones(len(X_train)),pca_dim1,pca_dim2,pca_dim3,pca_dim4,pca_dim5,pca_dim6]).T
theta,residuals,rank,s = numpy.linalg.lstsq(X, y_train)
print(residuals/len(y_train))

#7
pca_dim7 = numpy.dot(X_train,pca.components_[6])
X = numpy.vstack([numpy.ones(len(X_train)),pca_dim1,pca_dim2,pca_dim3,pca_dim4,pca_dim5,pca_dim6,pca_dim7]).T
theta,residuals,rank,s = numpy.linalg.lstsq(X, y_train)
print(residuals/len(y_train))

#8
pca_dim8 = numpy.dot(X_train,pca.components_[7])
X = numpy.vstack([numpy.ones(len(X_train)),pca_dim1,pca_dim2,pca_dim3,pca_dim4,pca_dim5,pca_dim6,pca_dim7,pca_dim8]).T
theta,residuals,rank,s = numpy.linalg.lstsq(X, y_train)
print(residuals/len(y_train))

#9
pca_dim9 = numpy.dot(X_train,pca.components_[8])
X = numpy.vstack([numpy.ones(len(X_train)),pca_dim1,pca_dim2,pca_dim3,pca_dim4,pca_dim5,pca_dim6,pca_dim7,pca_dim8,pca_dim9]).T
theta,residuals,rank,s = numpy.linalg.lstsq(X, y_train)
print(residuals/len(y_train))

#10
pca_dim10 = numpy.dot(X_train,pca.components_[9])
X = numpy.vstack([numpy.ones(len(X_train)),pca_dim1,pca_dim2,pca_dim3,pca_dim4,pca_dim5,pca_dim6,pca_dim7,pca_dim8,pca_dim9,pca_d
theta,residuals,rank,s = numpy.linalg.lstsq(X, y_train)
print(residuals/len(y_train))
```

```
#11
pca_dim11 = numpy.dot(X_train,pca.components_[10])
X = numpy.vstack([numpy.ones(len(X_train)),pca_dim1,pca_dim2,pca_dim3,pca_dim4,pca_dim5,pca_dim6,pca_dim7,pca_dim8,pca_dim9, \
                  pca_dim10,pca_dim11]).T
theta,residuals,rank,s = numpy.linalg.lstsq(X, y_train)
print(residuals/len(y_train))

print("Test Data")
#1
pca_dim1 = numpy.dot(X_test,pca.components_[0])
X = numpy.vstack([numpy.ones(len(X_test)),pca_dim1]).T
theta,residuals,rank,s = numpy.linalg.lstsq(X, y_test)
print(residuals/len(y_test))

#2
pca_dim2 = numpy.dot(X_test,pca.components_[1])
X = numpy.vstack([numpy.ones(len(X_test)),pca_dim1,pca_dim2]).T
theta,residuals,rank,s = numpy.linalg.lstsq(X, y_test)
print(residuals/len(y_test))

#3
pca_dim3 = numpy.dot(X_test,pca.components_[2])
X = numpy.vstack([numpy.ones(len(X_test)),pca_dim1,pca_dim2,pca_dim3]).T
theta,residuals,rank,s = numpy.linalg.lstsq(X, y_test)
print(residuals/len(y_test))

#4
pca_dim4 = numpy.dot(X_test,pca.components_[3])
X = numpy.vstack([numpy.ones(len(X_test)),pca_dim1,pca_dim2,pca_dim3,pca_dim4]).T
theta,residuals,rank,s = numpy.linalg.lstsq(X, y_test)
print(residuals/len(y_test))
```

```
#11
pca_dim11 = numpy.dot(X_test,pca.components_[10])
X = numpy.vstack([numpy.ones(len(X_test)),pca_dim1,pca_dim2,pca_dim3,pca_dim4,pca_dim5,pca_dim6,pca_dim7,pca_dim8,pca_dim9, \
                 pca_dim10,pca_dim11]).T
theta,residuals,rank,s = numpy.linalg.lstsq(X, y_test)
print(residuals/len(y_test))
```

```
Training Data
[ 0.86384244]
[ 0.84466943]
[ 0.82751612]
[ 0.69738328]
[ 0.68550156]
[ 0.66085399]
[ 0.65919451]
[ 0.65848602]
[ 0.63685661]
[ 0.63468782]
[ 0.61721176]
Test Data
[ 0.65822412]
[ 0.64494421]
[ 0.64493561]
[ 0.53185889]
[ 0.52764414]
[ 0.52762365]
[ 0.51751753]
[ 0.51746658]
[ 0.47630804]
[ 0.47626403]
[ 0.47034757]
```

For both the train and the test set, the MSE progressively decreases as more number of dimensions are added but the range of the test MSE is lower than that of the train MSE.