Homework 2
CSE 258
Spring 2017

Xiaowen Mao
PID: A53220159
x9mao@eng.ucsd.edu

1. Randomly re-shuffle

Lambda	Train	Validate	Test
0	0.745710784314	0.748315982854	0.761175750153
0.01	0.746323529412	0.747091243111	0.761788120024
1.0	0.736519607843	0.72933251684	0.748928352725
100.0	0.656862745098	0.666258420086	0.679118187385

The source code is as showed in Figure 1.

```
print "Reading data..."
dataFile = open("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]

# Randomly re-shuffle
numpy.random.shuffle(lines)

X = [l[:-1] for l in lines]
y = [l[-1] > 5 for l in lines]
print "done"
```

```
# 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(
```

```
lambda = 0;     train=0.745710784314; validate=0.748315982854; test=0.761175750153
lambda = 0.01;  train=0.746323529412; validate=0.747091243111; test=0.761788120024
lambda = 1.0;   train=0.736519607843; validate=0.72933251684; test=0.748928352725
lambda = 100.0; train=0.656862745098; validate=0.666258420086; test=0.679118187385
```

Figure 1: Code for Task 1

2. Report Accuracy

TP = 1129

TN = 145

FP = 321

FN = 38

BER = 0.360702

The source code is as showed in Figure 2.

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

    # true positives, true negatives, false positives, false negatives
    TP = sum([(a==b and b==1) for (a,b) in zip(predictions_test,y_test)])
    TN = sum([(a==b and b==0) for (a,b) in zip(predictions_test,y_test)])
    FP = sum([(a!=b and a==1) for (a,b) in zip(predictions_test,y_test)])
    FN = sum([(a!=b and a==0) for (a,b) in zip(predictions_test,y_test)])
    # Balanced Error Rate of the classifier
    # True positive rate (TPR), True negative rate (TNR)
    TPR = TP / (TP+FN+.0)
    TNR = TN / (TN+FP+.0)

    print "TP = %d\nTN = %d\nFP = %d\nFN = %d" %(TP, TN, FP, FN)
    print "BER = %f" %(1 - (TPR+TNR)/2)

# task 2
theta = train(0.01)
performance_accuracy(theta)
```

```
TP = 1129
TN = 145
FP = 321
FN = 38
BER = 0.360702
```

Figure 2: Code for Task 2

3. Rank Predictions

Top	precision	recall
10	1.000000	0.008569
500	0.956000	0.409597
1000	0.864000	0.740360

The source code is as showed in Figure 3.

```
def rank_prediction(theta):
    scores_test = [inner(theta,x) for x in X_test]
    rank = zip(scores_test, y_test)
    rank.sort(key = lambda x:x[0], reverse = True)

    total_relavant = sum(y_test)

    for budget in [10, 500, 1000]:
        relavant = 0
        for i in range(budget):
            relavant += rank[i][1]
            precision = float(relavant)/budget
            recall = float(relavant)/total_relavant
        print "budget = %f\tprecision = %f\trecall = %f" %(budget, precision, recall)

# task 3
theta = train(0.01)
rank_prediction(theta)
```

```
budget = 10.000000      precision = 1.000000    recall = 0.008569
budget = 500.000000     precision = 0.956000    recall = 0.409597
budget = 1000.000000    precision = 0.864000    recall = 0.740360
```

Figure 3: Code for Task 3

4. Plot precision versus recall

The code and result are attached in Figure 4.

```
# task 4
import matplotlib.pyplot as plt
theta = train (0.01)
scores_test = [inner(theta,x) for x in X_test]
rank = zip(scores_test, y_test)
rank.sort(key = lambda x:x[0], reverse = True)
precision = []
recall = []

total_relavant = sum(y_test)
for budget in range(1, len(y_test)+1):
    relavant = 0
    for i in range(budget):
        relavant += rank[i][1]
    precision.append(float(relavant)/budget)
    recall.append(float(relavant)/total_relavant)

plt.plot(precision, recall)
plt.show()
```

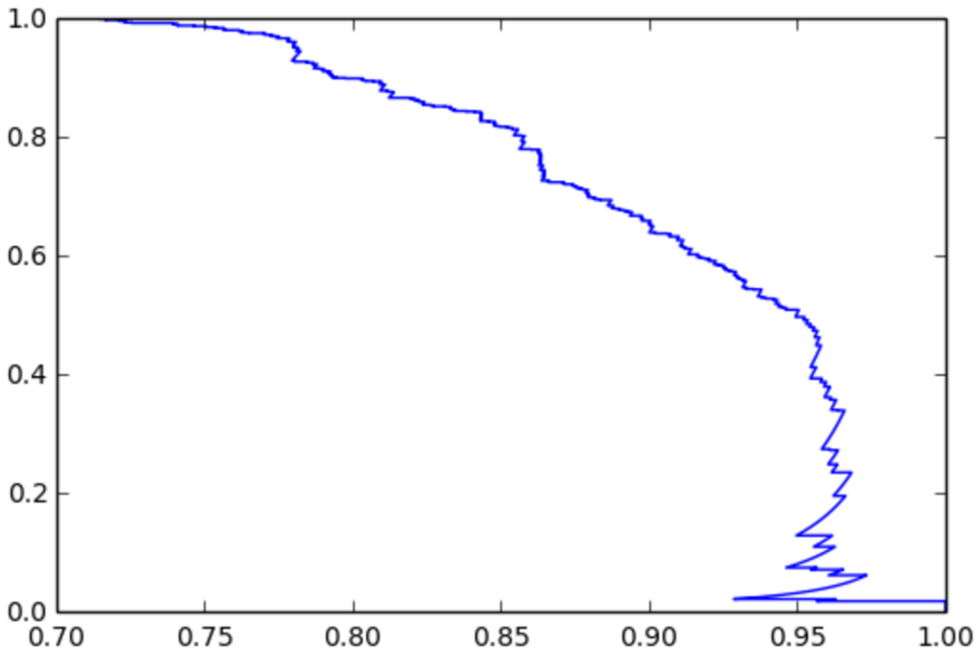

Figure 4: Code and Result for Task 4

5. Reconstruction Error

$$ReconstructionError = 3675818.617$$

The code is attached in Figure 5.

```
# task 5
X_train = numpy.matrix(X_train)
X_mean = numpy.mean(X_train, axis=0)
re = numpy.sum(numpy.square(X_train-X_mean))
print "Reconstruction error = %.3f" %re
```

```
Reconstruction error = 3675818.617
```

Figure 5: Code for Task 5

6. PCA

The code and result is attached in Figure 6.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=5)
pca.fit(X_train[:, 1:])
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]]
```

Figure 6: Code and result for task 6

7. Four PCA

$$ReconstructionError = 1345.47557416$$

The code and result is attached in Figure 7.

```
# task 7
pca = PCA(n_components=4)
pca.fit(X_train[:, 1:])
print numpy.sum(re) - len(X_train) * numpy.sum(pca.explained_variance_)
```

```
1345.47557416
```

Figure 7: Code and result for task 7

8. Linear Regressor

The code is attached in Figure 8.

```
# task 8
pca = PCA(n_components=11)
X_train_pca = pca.fit_transform(X_train[:, 1:])
X_validate_pca = pca.transform(numpy.array(X_validate)[:, 1:])
X_test_pca = pca.transform(numpy.array(X_test)[:, 1:])
y = [l[-1] for l in lines]
y_train = y[:int(len(y)/3)]
y_validate = y[int(len(y)/3):int(2*len(y)/3)]
y_test = y[int(2*len(X)/3):]

from sklearn.linear_model import LinearRegression as LR
lr = LR()
train_mse = []
test_mse = []

for i in range(1, 12):
    lr.fit(X_train_pca[:, :i], y_train)
    train_mse.append(lr.residues_ / len(y_train))
    s = lr.score(X_test_pca[:, :i], y_test)
    v = ((y_test - numpy.mean(y_test))**2).sum()
    test_mse.append((float)((1-s) * v) / len(y_test))

plt.plot(range(1, 12), train_mse, 'r', label='MSE on train')
plt.plot(range(1, 12), test_mse, 'g', label='MSE on test')
plt.legend()
```

Figure 8: Code for task 8

The result is attached in Figure 9.

```
<matplotlib.legend.Legend at 0x10d3fcad0>
```

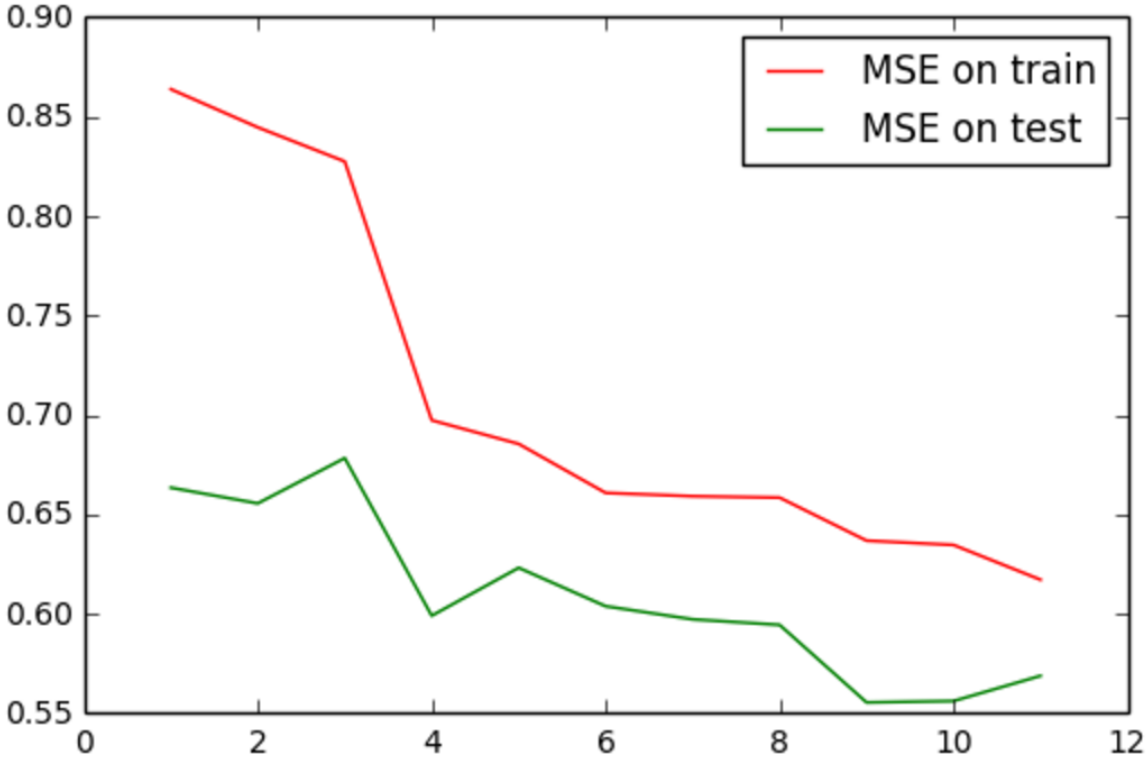

Figure 9: Result for task 8