Zhang, Jinhan PID A53211930
University of California, San Diego

January 22, 2017

HOMEWORK ONE

Problem 1. Solution

$\theta_0 = -39.1707489$, $\theta_1 = 0.0214379786$

Listing 1: Key code for Prob.1

```
def feature(datum):
    feat=[1]
    feat.append(datum['review/timeStruct']['year'])
    return feat

X = [feature(d) for d in data]
y = [d['review/overall'] for d in data]
theta,residuals,rank,s = numpy.linalg.lstsq(X,y)
print theta
```

Problem 2. Solution

Since the range of the years in dataset is all between 1999 and 2012, we can assume that all year labels are between 1999 and 2012, thus we can use 13 boolean variables to represent year variable(since if the states of all 13 years are False, it must be the remaining year, so we can use 13 extra variable to represent 14 years.) The equation for that is:

$$review/overall \simeq \theta_0 + \theta_1 \times is1999 + \theta_2 \times is2000 + \theta_3 \times is2001 + \theta_4 \times is2002 + \theta_5 \times is2003 + \theta_6 \times is2004 + \theta_7 \times is2005 + \theta_8 \times is2006 + \theta_9 \times is2007 + \theta_{10} \times is2008 + \theta_{11} \times is2009 + \theta_{12} \times is2010 + \theta_{13} \times is2011$$

The MSE for the model in Problem 1 is: 0.49004382

The MSE for our model is: 0.4891519

We see that MSE has decreased when compared to that in Problem 1.

Listing 2: Key code for Prob.2

```
MSE1=residuals/len(y)  #residuals,X and y are what we get from code in Listing 1
print MSE1
tmp=numpy.matrix(X)
minV=tmp[:,1].min()
maxV=tmp[:,1].max()

```

```
def feature2(datum):
    feat=[1]
    tmp=datum['review/timeStruct']['year']
    l=[0 for x in range(13)]
    if tmp!=maxV:
        l[tmp-minV]=1
    feat.extend(l)
    return feat

X = [feature2(d) for d in data]
y = [d['review/overall'] for d in data]

theta,residuals,rank,s = numpy.linalg.lstsq(X,y)
MSE2=residuals/len(y)
print MSE2
```

Problem 3. Solution

Fitted Coefficients:

$\theta_0 = 256.420279$
$\theta_1 = 0.135421303$
$\theta_2 = -1.72994866$
$\theta_3 = 0.102651152$
$\theta_4 = 0.109038568$
$\theta_5 = -0.276775146$
$\theta_6 = 0.00634332168$
$\theta_7 = 0.00003.85023977$
$\theta_8 = -258.652809$
$\theta_9 = 1.19540566$
$\theta_{10} = 0.833006285$
$\theta_{11} = 0.09.79304353$

MSE on the train data : 0.602307502903

MSE on the test data : 0.562457130315

The code below stores data in dictionary, so the key field is sorted alphabetically in the code, but I've mapped the coefficients to the oringal order as the assignment required manually. The same applies for all of the rest.

Listing 3: Key code for Prob.3

```
l=len(data)     #split the data into 2 groups
train=data[0:l/2]
test=data[l/2:]

keys=["alcohol","chlorides","citric acid","density","fixed acidity",
  "free sulfur dioxide","pH","residual sugar","sulphates",
  "total sulfur dioxide","volatile acidity"]
def feature(datum):
    feat=[1]
```

```
    for key in keys:
        feat.append(eval(datum[key]))
    return feat
X=[feature(d) for d in train]
y=[eval(d["quality"]) for d in train]
theta,residuals,rank,s = numpy.linalg.lstsq(X,y)
print "Theta:",theta
print "Training Set MSE:",residuals[0]/len(train)

X_t=[feature(d) for d in test]
y_t=[eval(d["quality"]) for d in test]

X_t=numpy.matrix(X_t)
theta=numpy.matrix(theta)
y_t=numpy.matrix(y_t)
diff=theta*X_t.T-y_t
mse=diff*diff.T
k = mse.tolist()[0]
print "Test Set MSE:", k[0] / len(test)
```

Problem 4. Solution

(a) MSEs (on the test set) of all 11 ablation experiments:

$$
\begin{aligned}
Removed\_Field &: MSE \\
"fixedacidity" &: 0.559113414376 \\
"volatileacidity" &: 0.596384850161 \\
"citricacid" &: 0.562221702812 \\
"residualsugar" &: 0.553625063967 \\
"chlorides" &: 0.562629266481 \\
"freesulfurdioxide" &: 0.55614081793 \\
"totalsulfurdioxide" &: 0.562429005469 \\
"density" &: 0.544726553466 \\
"pH" &: 0.559566626382 \\
"sulphates" &: 0.557346349988 \\
"alcohol" &: 0.573214743558
\end{aligned}
$$

(b) Conclusions:

From the result above, we see the MSE is highest when we remove "volatile acidity" field, so we can conclude that "volatile acidity" feature provides the most additional information for the data. And the MSE is lowest when we remove "density" field, which means "density" feature provides the least additional information.

Listing 4: Key code for Prob.4

```
l=len(data)      #split the data into 2 groups
train=data[0:l/2]
```

```
test=data[l/2:]

keys=["alcohol","chlorides","citric acid","density","fixed acidity",
  "free sulfur dioxide","pH","residual sugar","sulphates",
  "total sulfur dioxide","volatile acidity"]
def feature(datum):
    feat=[1]
    for key in keys:
        feat.append(eval(datum[key]))
    return feat
X=[feature(d) for d in train]
y=[eval(d["quality"]) for d in train]

for i in range(1,12):
    X_ab = [x[:i] + x[i + 1:] for x in X]
    theta, residuals, rank, s = numpy.linalg.lstsq(X_ab, y)
    print "Training Set MSE:",residuals[0]/len(train)
    X_t = [feature(d) for d in test]
    y_t = [eval(d["quality"]) for d in test]

    X_ab_t = [x[:i] + x[i + 1:] for x in X_t]
    X_ab_t = numpy.matrix(X_ab_t)
    theta = numpy.matrix(theta)
    y_t = numpy.matrix(y_t)
    diff = theta * X_ab_t.T - y_t
    mse = diff * diff.T
    k=mse.tolist()[0]
    print "Test Set MSE:",k[0]/len(test)
    print
```

Problem 5. Solution

When we use the default "penalty parameter" (C=1000), the train accuracy is 100% , while the test accuracy is 66.803%. This result is mainly because we have a large "penalty parameter", which means we can hardly tolerate any mis-classification in training dataset. And that leads to overfitting, so we have a relatively low accuracy in test dataset. And when I change the C value into 0.8, the train accuracy decreases to 90.527% while the test accuracy increases to 69.906%, which means the overfitting has been weakened.

Listing 5: Key code for Prob.5

```
keys=["alcohol","chlorides","citric acid","density","fixed acidity",
  "free sulfur dioxide","pH","residual sugar","sulphates",
  "total sulfur dioxide","volatile acidity"]
def feature(datum):
    feat=[]
    for key in keys:
        feat.append(eval(datum[key]))
    return feat
X=[feature(d) for d in data]
y=[eval(d["quality"])>5 for d in data]

l=len(data)      #split the data into 2 groups
X_train = X[:l/2]
```

```
y_train = y[:1/2]

X_test = X[1/2:]
y_test = y[1/2:]

clf = svm.SVC(C=1000)
clf.fit(X_train, y_train)

train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)

accuracy_train=sum([z[0]==z[1] for z in zip(train_predictions,
  y_train)])*1.0/len(train_predictions)
accuracy_test=sum([z[0]==z[1] for z in zip(test_predictions,
  y_test)])*1.0/len(test_predictions)


print "The train accuracy is: ",accuracy_train
print "The test accuracy is: ",accuracy_test
```

Problem 6. Solution

(a) Code stub for the derivative (fprime):

```
def fprime(theta, X, y, lam):
    dl = [0.0]*len(theta)
    for j in range(len(theta)):
        for i in range(len(X)):
            logit = inner(X[i], theta)
            dl[j] += X[i][j] * (1 - sigmoid(logit))
            if not y[i]:
                dl[j] -= X[i][j]
        dl[j] -= 2 * lam * theta[j]
    # Negate the return value since we're doing gradient *ascent*
    # print "dll =", dl
    return numpy.array([-x for x in dl])
```

(b) Log-likelihood and accuracy

The log-likelihood after convergence is -1388.69589879

The accuracy is 76.929%

Submitted by Zhang, Jinhan PID A53211930 on January 22, 2017.