EECS 545 - Homework 2

Ding Ding

February 8, 2016

1. Linear Regression

(a) i

```python
import numpy as np
from matplotlib import pyplot as plt
#1
#(a)(i)
#input the data
data=np.loadtxt("train_graphs_f16_autopilot_cruise.csv",delimiter=",", skiprows=1,usecols
    =(1,2,3,4,5,6,7))
data2=np.loadtxt("test_graphs_f16_autopilot_cruise.csv",delimiter=",", skiprows=1,usecols
    =(1,2,3,4,5,6,7))
X=data[:,:-1] #X is a feature set
Y=data[:,-1] #Y is an array with target value
X2=data2[:,:-1]#X2 is a feature set of test value
Y2=data2[:,-1] #Y2 is an array with test target value
Intercept=np.ones((Y.size,1))
X1=np.concatenate((X,Intercept),axis=1)
Intercept2=np.ones((Y2.size,1))
X22=np.concatenate((X2,Intercept2),axis=1)
M=np.array([1,2,3,4,5,6])
#get the RMSE of training data
#get the RMSE of test data
RMSE=np.zeros(6)
RMSEE=np.zeros(6)
A1=X1
B1=np.dot(np.linalg.pinv(A1),Y)
w=B1
A2=X22
s1=np.power(np.dot(A1,w)-Y,2)
s2=np.power(np.dot(A2,w)-Y2,2)
RMSE[0]=np.sqrt(np.sum(s1)/Y.size)
RMSEE[0]=np.sqrt(np.sum(s2)/Y2.size)
for i in range(0,5):
    A1=(np.concatenate((A1.T,np.power(X,i+2).T))).T
    A2=(np.concatenate((A2.T,np.power(X2,i+2).T))).T
    B1=np.dot(np.linalg.pinv(A1),Y)
    w=B1
    s1=np.power(np.dot(A1,w)-Y,2)
    s2=np.power(np.dot(A2,w)-Y2,2)
    RMSE[i+1]=np.sqrt(np.sum(s1)/Y.size)
    RMSEE[i+1]=np.sqrt(np.sum(s2)/Y2.size)
#plot it
plt.plot(M,RMSE,label='RMSE of training data')
plt.plot(M,RMSEE,label='RMSE of test data')
plt.legend()
plt.savefig('plot1')
plt.show()
```

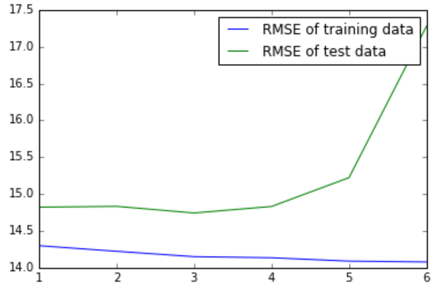

Figure 1: plot 1

ii

```
#(a)(ii)
lnlambda=np.arange(-40,21)
#get the RMSE of training data
#get the RMSE of test data
RMSE=np.zeros(lnlambda.size)
RMSE2=np.zeros(lnlambda.size)
D=np.identity(37)
for j in range(0,lnlambda.size):
    C=np.dot(A1.T,A1)+np.exp(lnlambda[j])*D
    C=np.dot(np.linalg.inv(C),A1.T)
    w=np.dot(C,Y)
    s1=np.power(np.dot(A1,w)-Y,2)
    s2=np.power(np.dot(A2,w)-Y2,2)
    RMSE[j]=np.sqrt((np.sum(s1)+(np.exp(lnlambda[j]))*np.power(np.linalg.norm(w),2))/Y.size)
    RMSE2[j]=np.sqrt((np.sum(s2)+(np.exp(lnlambda[j]))*np.power(np.linalg.norm(w),2))/Y2.size)
#plot it
plt.plot(lnlambda,RMSE,label='RMSE of training data')
plt.plot(lnlambda,RMSE2,label='RMSE of test data')
plt.legend()
plt.savefig('plot2')
plt.show()
```

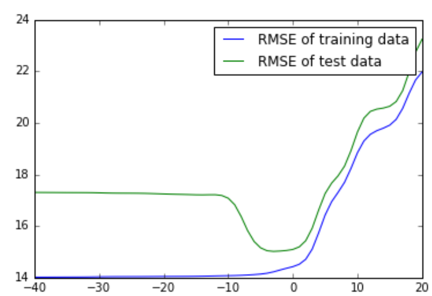

Figure 2: plot 2

(b)

```
#get tao
tao=np.logspace(-2,1,num=10,base=2)
#impute data
datal=np.loadtxt("test_locreg_f16_autopilot_cruise.csv",delimiter=",", skiprows=1,usecols
     =(1,2,3,4,5,6,7))
X1=data[:,:-1]
X22=datal[:,:-1]
Y2=datal[:,-1]
#get r(x)
def r(x,taoi):
    a=np.eye(Y.size)
    for i in range(0,Y.size):
        a[i,i]=np.exp(-(np.linalg.norm(x-X1[i,:]))**2/(2*(np.power(taoi,2))))
    return a
def w(feature,weight,target):
    return np.dot(np.dot(np.linalg.pinv(np.dot(np.sqrt(weight),feature)),np.sqrt(weight)),target)
RMSEl=np.zeros(tao.size)
for k in range(0,tao.size):
    for j in range(0,Y2.size):
        RMSEl[k]=RMSEl[k]+np.power((np.dot(w(X1,r(X22[j,:],tao[k]),Y).T,X22[j,:])-Y2[j]),2)
    RMSEl[k]=np.sqrt(RMSEl[k]/Y2.size)
#plot it
plt.plot(tao,RMSEl,"-g",label="RMSE of local reg")
plt.legend()
plt.savefig('plot3')
plt.show()
```

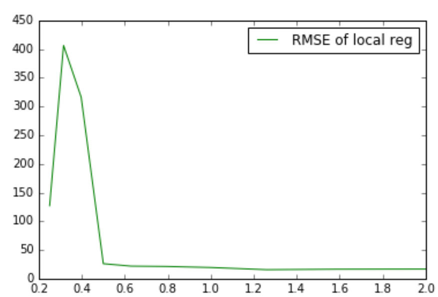

Figure 3: plot 3.1

When λ is small, it is unstable. Do the normalization:

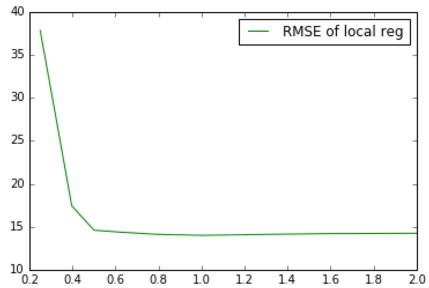

Figure 4: plot 3.2

2. Open Kaggle challenge

   After plot the relationship between each features and target value,delete two features that has no obvious trend for the value of target value.

```
#2)
#input the data
data=np.loadtxt("steel_composition_train.csv",delimiter=",", skiprows=1,usecols=(1,2,3,4,5,8,9))
data2=np.loadtxt("steel_composition_test.csv",delimiter=",", skiprows=1,usecols=(1,2,3,4,5,8))
X=data[:,:-1] #X is a feature set
Y=data[:,-1] #Y is an array with target value
X2=data2#X2 is a feature set of test value
Intercept=np.ones((Y.size,1))
X1=np.concatenate((X,Intercept),axis=1)
Intercept2=np.ones((X2[:,0].size,1))
X22=np.concatenate((X2,Intercept2),axis=1)
A1=X1
B1=np.dot(np.linalg.pinv(A1),Y)
w=B1
A2=X22
s1=np.power(np.dot(A1,w)-Y,2)
RMSE=np.sqrt(np.sum(s1)/Y.size)
for i in range(0,3):
    A1=(np.concatenate((A1.T,np.power(X,i+2).T))).T
    A2=(np.concatenate((A2.T,np.power(X2,i+2).T))).T
    B1=np.dot(np.linalg.pinv(A1),Y)
    w=B1
Y2=np.dot(A2,w)
Y2=np.array(Y2)
print Y2.shape
import csv
fl = open('steel_result.csv', 'w')
writer = csv.writer(fl,lineterminator='\n')
for values in Y2:
    writer.writerow([values])
fl.close()
```

3. Weighted Linear Regression

(a)

$$\begin{aligned} E_D W &= \frac{1}{2}\sum_{i=1}^{N} r_i(w^T x_i - t_i)^2 \\ &= \frac{1}{2}\sum_{i=1}^{N} r_i(\sum_{j=1}^{M} w_i x_{ij} - t_i)^2 \end{aligned}$$

where M is the number of features.

Set

$X = (x_{ij})_{N*M}$ A feature matrix

$w = (w_i)_{M*1}$ A vector of coefficients

$t = (t_i)_{N*1}$ A vector of the target value

$R = diag(\frac{1}{2}r_i)_{N*N}$ A diag matrix with weights in the diag

Then, we have:

$$\sum_{j=1}^{M} w_i x_{ij} = (Xw)_{N*1}$$

$$\begin{aligned} \frac{1}{2}\sum_{i=1}^{N}((\sum_{j=1}^{M} w_i x_{ij} - t_i) r_i(\sum_{j=1}^{M} w_i x_{ij} - t_i)) &= \sum_{i=1}^{N} r_i(Xw - t)^2 \\ &= (Xw - t)^T R(Xw - t) \end{aligned}$$

(b) $E_D W = (Xw - t)^T R(Xw - t)$

To minimize the error, we want it derivative equal to 0.

We have its derivative equal to:

$$\begin{aligned} \nabla_w E_D(w) &= 2X^T R(Xw - t) = 0 \\ X^T RXw &= X^T Rt \\ w^* &= (X^T RX)^{-1} X^T Rt \end{aligned}$$

(c)

$$\begin{aligned} p(t|x, w) &= \prod_{n=1}^{N} N(w^T Xn, \sigma_i^2) \\ &= \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma_i^2} exp(-\frac{(ti - w^T x_i)^2}{2\sigma_i^2}) \end{aligned}$$

where $w^T = (w_i)_{1*M}, x_i = (x_{ij})_{1*M}$ for each i

Then, we have:

$$
\begin{aligned}
logp(t|x,w) &= \sum_{i=1}^{N} log\frac{1}{\sqrt{2\pi}\sigma_i} - \frac{(ti - w^T x_i)^2}{2\sigma_i^2} \\
&= -log\sqrt{2\pi} - \sum_{i=1}^{N}(log\sigma_i + \frac{(ti - w^T x_i)^2}{2\sigma_i^2}) \\
\nabla_w logp(t|x,w) &= \sum_{i=1}^{N} \frac{(ti - w^T x_i)}{\sigma_i^2}(x_i)^T = 0 \\
w^T \sum_{i=1}^{N} \frac{x_i x_i^T}{\sigma_i^2} &= \frac{t_i x_i^T}{\sigma_i^2}
\end{aligned}
$$

follow the set from question (a),

set $R = diag(\frac{1}{2\sigma_i^2})$

We can get

$$w(X^T RX) = X^T Rt$$

Finally, we get

$$w^* = (X^T RX)^{-1} X^T Rt$$

with $r_i = \frac{1}{\sigma_i^2}$

4. Naive Bayes Classifier

(a) i The pre-processing is a process that first compute the median of the data and than map it due from median. Since nominal variables can not compute median, so this type of variable are not suitable for the preprocessing described.

To see the spambase data, we can see there are all ratio variables. Since there is no nominal data in all features and only the target value is nominal, all features are suitable for the pre-processing.

ii
```
data=np.loadtxt("spambase.train",delimiter=",")
data2=np.loadtxt("spambase.test",delimiter=",")
X=np.concatenate((data,data2))
X.shape
for i in range(0,X[0,:].size-1):
    m=np.median(X[:,i])
    for j in range(0,data[:,0].size):
        if data[j,i]<m:
```

```
            data[j,i]=1
        else:
            data[j,i]=2
    for j in range(0,data2[:,0].size):
        if data2[j,i]<m:
            data2[j,i]=1
        else:
            data2[j,i]=2
#step1 pie
pspam=(np.float16(np.count_nonzero(data[:,-1])+1))/np.float16((data[:,-1].size+2))
pham=1-pspam
print pham
#step2 sita
Qspam=np.zeros((data[0,:].size-1,1))#conditional on spam
for i in range(0,data[0,:].size-1):
    count=0
    for j in range(0,data[:,0].size):
        if data[j,-1]==1 and data[j,i]==1:
            count=count+1
        else:
            count=count
    Qspam[i,0]=np.float16((count+1))/np.float16((np.count_nonzero(data[:,-1])+2))
Qham=np.zeros((data[0,:].size-1,1))#conditional on ham
for i in range(0,data[0,:].size-1):
    count=0
    for j in range(0,data[:,0].size):
        if data[j,-1]==0 and data[j,i]==1:
            count=count+1
        else:
            count=count
    Qham[i,0]=np.float16((count+1))/np.float16((data[:,-1].size-np.count_nonzero(data[:,-1])+2))
#step3 use test value
Y=np.zeros((data2[:,0].size,1))
for i in range(0,data2[:,0].size):
    ppspam=1
    ppham=1
    for j in range(0,data2[0,:].size-1):
        if data2[i,j]==1:
            ppspam=ppspam*Qspam[j,0]
            ppham=ppham*Qham[j,0]
        else:
            ppspam=ppspam*(1-Qspam[j,0])
            ppham=ppham*(1-Qham[j,0])
    ppspam=ppspam*pspam
    ppham=ppham*pham
    if ppspam>ppham:
        Y[i,0]=1
    else:
        Y[i,0]=0
#step4 get test error
con=0
for i in range(0,data2[:,0].size):
    if Y[i]==data2[i,-1]:
        con=con+1
```

```
    else:
        con=con
percent=np.float16(con)/np.float(data2[:,0].size)
percent=1-percent
print percent
```

We get the test error of Naive Bayes classifier:

0.250288350634 If always predict the same class from the training data, we have:

```
Z=np.zeros((data2[:,0].size,1))
con=0
for i in range(0,data2[:,0].size):
    if Z[i]==data2[i,-1]:
        con=con+1
    else:
        con=con
percent=np.float16(con)/np.float(data2[:,0].size)
percent=1-percent
print percent
```

We get the test error:

0.385620915033

(b)

```
import cPickle
import os
with open('trainFeatures.pkl', 'rb') as f:
    trainFeatures = cPickle.load(f)
with open('testFeatures.pkl', 'rb') as f:
    testFeatures = cPickle.load(f)
import numpy as np

data2=testFeatures.toarray()
data=trainFeatures.toarray()
datat=open('spam_filter_train.txt','r')
Y=np.zeros((data[:,0].size,1))
i=0
for line in datat:
    if line[0:4]=='spam':
        Y[i]=1
    else:
        Y[i]=0
    i=i+1
print data2.shape
print data.shape
print Y.shape
X=np.concatenate((data,data2))
X.shape
for i in range(0,X[0,:].size):
    m=np.median(X[:,i])
    for j in range(0,data[:,0].size):
        if data[j,i]<m:
            data[j,i]=1
        else:
            data[j,i]=2
```

```
    for j in range(0,data2[:,0].size):
        if data2[j,i]<m:
            data2[j,i]=1
        else:
            data2[j,i]=2
data11=np.zeros((np.sum(Y),data[0,:].size))#spam matrix
data12=np.zeros((Y.size-np.sum(Y),data[0,:].size))#ham matrix
a=0
b=0
for j in range(0,data[:,0].size):
    if Y[j]==1:
        data11[a,:]=data[j,:]
        a=a+1
    else:
        data12[b,:]=data[j,:]
        b=b+1
Qspam=np.zeros((47922,1))#conditional on spam
Qham=np.zeros((47922,1))#conditional on ham
def q(a,b):
    return (2*a.size-np.sum(b)+1)/(a.size+2)
for k in range(data[0,:].size-1,-1,-1):
    Qspam[k,0]=q(data11[:,0],data11[:,k])
    Qham[k,0]=q(data12[:,0],data12[:,k])

pspam=(np.sum(Y)+1)/(Y.size+2)
pham=1-pspam
Z=np.zeros((data2[:,0].size,1))
for i in range(0,data2[:,0].size):
    ppspam=1
    ppham=1
    for k in range(0,data[0,:].size):
        if data2[i,k]==1:
            ppspam=ppspam*Qspam[k,0]
            ppham=ppham*Qham[k,0]
        else:
            ppspam=ppspam*(1-Qspam[k,0])
            ppham=ppham*(1-Qham[k,0])
    ppspam=ppspam*pspam
    ppham=ppham*pham
    if ppspam>ppham:
        Z[i,0]=1
    else:
        Z[i,0]=0
Z1=np.zeros(data2[:,0].size)
for i in range(0,data2[:,0].size):
    Z1[i]=Z[i,0]
import csv
fl = open('naive_result.csv', 'w')
writer = csv.writer(fl,lineterminator='\n')
for values in Z1:
    writer.writerow([values])
fl.close()
```

5. Softmax Regression

(a)

$$p(t|w) = \prod_{n=1}^{N}\prod_{k=0}^{K-1} p(C_k|\phi(x_n))^{1_{t_n=k}}$$

Then We have:

$$\begin{aligned}
E(w) &= -lnp(t|w) \\
&= -\sum_{n=1}^{N}\sum_{k=1}^{K-1} 1_{t_n=k} ln \frac{exp(w^T_{k)\phi(x_n)}}{\sum_{k=1}^{K-1} exp(w_k^T\phi(x_n))}
\end{aligned}$$

$$\begin{aligned}
&\nabla_w ln(t|w) \\
&= \frac{\partial E(w)}{\partial w_j} \\
&= \frac{\partial - \sum_{n=1}^{N}\sum_{k=1}^{K-1} 1_{t_n=k} ln \frac{exp(w_k^T)\phi(x_n)}{\sum_{k=1}^{K-1} exp(w^T_{k)\phi(x_n)}}}{\partial w_j} \\
&= -\sum_{n=1}^{N}[\phi(x_n)(1_{(t_n=j)} - \frac{exp(w_j^T\phi(x_n))}{\sum_{k=1}^{K-1} exp(w_k^T\phi(x_n))})]
\end{aligned}$$

(b)

$$\begin{aligned}
E^\lambda(w) &= E(w) + \frac{\lambda}{2}\sum_{k=1}^{K-1} w_k^T w_k \\
\nabla_w E^\lambda(w) &= -\sum_{n=1}^{N}[\phi(x_n)(1_{(t_n=j)} - \frac{exp(w_j^T\phi(x_n))}{\sum_{k=1}^{K-1} exp(w_k^T\phi(x_n))})] + \lambda w_j
\end{aligned}$$