.ipynb

October 25, 2019

[1]:
```
import numpy as np
import scipy.optimize
import random
import csv
```

[2]:
```
### Tasks - Regression

### 1. What is the distribution of ratings in the dataset? That is,
### how many 1-star, 2-star, 3-star (etc.) reviews are there?
### You may write out the values or include a simple plot (1 mark).

# Read the tsv file and extract the rating star, verified status
# and length of review as three different lists.
data = []
with open('amazon_reviews_us_Gift_Card_v1_00.tsv') as tsvfile:
  reader = csv.reader(tsvfile, delimiter='\t')
  for row in reader:
    data.append(row)

#print(len(data))
#data = np.array(data)
headline = data[0]
#print(headline)

rating = []
purStatus = []
lenOfReview = []
for i in range(len(data)):
    rating.append(data[i][7])
    purStatus.append(data[i][11])
    lenOfReview.append(len(data[i][13]))
#print(len(rating))
#print(len(purStatus))
#print(len(lenOfReview))
```

[3]:
```
# check the distribution of stars by count function
numOf1Star = rating.count('1')
```

```
numOf2Star = rating.count('2')
numOf3Star = rating.count('3')
numOf4Star = rating.count('4')
numOf5Star = rating.count('5')
```

[4]:
```
# print out the total count for star number 1,2,3 relatively
print('Total number of 1 star: ' , numOf1Star)
print('Total number of 2 star: ' , numOf2Star)
print('Total number of 3 star: ' , numOf3Star)
```

```
Total number of 1 star:  4766
Total number of 2 star:  1560
Total number of 3 star:  3147
```

[5]:
```
# trim the verified status data
verRv = purStatus[1:]
#print(verRv[0])
for i in range(len(verRv)):
    if verRv[i]  == 'Y':
       verRv[i]  = 1
    else: verRv[i]  = 0
#print(len(verRv))
#print(verRv[0:5])
```

[6]:
```
# trim the length of review data
lenOfReview = lenOfReview[1:]
```

[7]:
```
# set up our X matrix for the following regression problem
colOf1 = [1]*len(verRv)
X = np.column_stack((colOf1,verRv,lenOfReview))
X = y = np.asmatrix(X,dtype='float')
#X.shape
```

[8]:
```
# set up our y vector for the following regression problem
y = rating[1:]
y = np.asmatrix(y,dtype='float')
#y.shape
```

[9]:
```
# conduct the regression progress by the function
theta,residuals,rank,s = np.linalg.lstsq(X,y.T)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: FutureWarning:
`rcond` parameter will change to the default of machine precision times ``max(M,
N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
```

[10]:
```
### 3. Train a simple predictor to predict the star rating using two features
### Report the values of 0, 1, and 2. Briefly describe your interpretation of␣
↪these values
### Explain these in terms of the features and labels

# check the intercept and coefficients. Theta 0 is the coefficient, theta 1
# is the coefficient for verified status, theta 2 is the coefficient of review
# length.
print('Theta0 = ',theta[0])
print('Theta1 = ',theta[1])
print('Theta2 = ',theta[2])
```

```
Theta0 =  [[4.84503504]]
Theta1 =  [[0.04985806]]
Theta2 =  [[-0.00124546]]
```

[11]:
```
#Theta 0 represents the mean value of our prediction when the purchase is not␣
↪verified and the review length is 0
#Theta 1 represents the mean difference of the same length review when the␣
↪purchase is verfied or not
#Theta 2 represents the mean increment comes with one increment of review␣
↪length when the verified status is fixed

# The coefficients theta1 is positive. Therefore, if the review length is␣
↪fixed, the verified purchase will have a
# higher average rating number with 0.0498. Samely, the review length␣
↪coefficient is with a negative theta -0.001245.
# That means with a purchase in a fixed purchase status, we have an average␣
↪decrement of 0.001245 on the rating numbers.
# That also means a longer review tends to have a negative effect on our final␣
↪rating stars. So, a negative review is
# likely to be longer than a positive one.
```

[12]:
```
np.linalg.inv(X.T * X) * X.T * y.T
```

[12]:
```
matrix([[ 4.84503504e+00],
        [ 4.98580589e-02],
        [-1.24545526e-03]])
```

[13]:
```
### 4. Train another predictor that only uses one feature:
X1 = X[:,0:2]
X1.shape
```

[13]:
```
(148310, 2)
```

[14]:
```
theta1,residuals,rank,s = np.linalg.lstsq(X1,y.T)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning:
`rcond` parameter will change to the default of machine precision times ``max(M,
```

```
N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  """Entry point for launching an IPython kernel.
```

[15]:
```
### Report the values of 0 and 1
print('Theta0 = ',theta1[0])
print('Theta1 = ',theta1[1])
```

```
Theta0 =  [[4.578143]]
Theta1 =  [[0.16793392]]
```

[16]:
```
### Provide an explanation as to why these coefficients might vary so␣
↪significantly (1 mark).1

#The coefficients are different because the review length is not under our␣
↪consideration now.
#The linear regression try to solve them with a minimized MRE, without the␣
↪review length, the minimization progress
#is quite different now. We are only try to optimize the rating stars towards␣
↪the verified status. Therefore, the
#coefficients would vary through this progress.
```

[17]:
```
### 5. Split the data into two fractions  the first 90% for training, and the␣
↪remaining 10%
### testing (based on the order they appear in the file). Train the same model␣
↪as in Question
### 4 on the training set only. What is the models MSE on the training and on␣
↪the test set (1 mark)?

# Split the 90% and 10% of our data
X1train = X1[0: round(0.9*(X1.shape[0])),]
print(X1train.shape)
X1test = X1[round(0.9*(X1.shape[0])):,]
print(X1test.shape)
print(y.shape)
ytrain = y[0,0: round(0.9*(y.shape[1]))]
ytest = y[0,round(0.9*(y.shape[1])):]
print(ytrain.shape)
print(ytest.shape)
```

```
(133479, 2)
(14831, 2)
(1, 148310)
(1, 133479)
(1, 14831)
```

[18]:
```
thetaT,residualsT,rankT,sT = np.linalg.lstsq(X1train,ytrain.T)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning:
`rcond` parameter will change to the default of machine precision times ``max(M,
N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  """Entry point for launching an IPython kernel.
```

[19]:
```
print(residualsT)
```

```
[[87493.37815713]]
```

[20]:
```
# Calculate the test MSE
print(ytest.shape)
ypredict = X1test@thetaT
print(ypredict.shape)
difference = np.subtract(ypredict,ytest.T)
print(difference.shape)
```

```
(1, 14831)
(14831, 1)
(14831, 1)
```

[21]:
```
SumtestMSE = np.linalg.norm(difference)
#print(SumtestMSE)
testMSE = (SumtestMSE**2)/ypredict.shape[0]
print('test MSE',testMSE)
```

```
test MSE 0.9723851990303902
```

[22]:
```
### report the MSE for training and test sets
MSETrain = (residualsT/ytrain.shape[1])[0,0]
print('MSE for the training residual',MSETrain)
print('MSE for the testing residual',testMSE)
```

```
MSE for the training residual 0.6554842196685237
MSE for the testing residual 0.9723851990303902
```

[23]:
```
### 7. (CSE258 only) Repeat the above experiment, varying the size
### of the training and test fractions between 5% and 95% for training
### (using the complement for testing). Show how the training and test
### error vary as a function of the training set size (again using a
### simple plot or table). Does the size of the training set make a
### significant difference in testing performance? Comment on why it
```

```
### might or might not make a significant difference in this instance (2 marks).

# the fuction could change the size of training and test sets
def separateTrainNTest(X, y, percentage):
     XTrain = X[0: round(percentage*(X.shape[0])),]
     XTest = X[round(percentage*(X.shape[0])):,]
     yTrain = y[0,0: round(percentage*(y.shape[1]))]
     yTest = y[0,round(percentage*(y.shape[1])):]
     return [XTrain,XTest,yTrain,yTest]
```

[24]:
```
# conduct a new MSE calculator of regression for the training set
def MSEregressionOfTrain(XTrain,yTrain):
    thetaTrain,residualsTrain,rankTrain,sTrain = np.linalg.lstsq(XTrain,yTrain.
 ↪T)
    MSETrain = (residualsTrain/yTrain.shape[1])[0,0]
    return [thetaTrain,MSETrain]
```

[25]:
```
# conduct a new MSE calculator of regression for the test set
def MSEregressionOfTest(thetaTrain,XTest,yTest):
    ypredict = XTest@thetaTrain
    difference = np.subtract(ypredict,yTest.T)
    SumtestMSE = np.linalg.norm(difference)
    MSETest = (SumtestMSE**2)/ypredict.shape[0]
    return MSETest
```

[26]:
```
MSErecord = []
for i in range(91):
    XTrain = separateTrainNTest(X1,y,(i+5)*0.01)[0]
    XTest = separateTrainNTest(X1,y,(i+5)*0.01)[1]
    yTrain = separateTrainNTest(X1,y,(i+5)*0.01)[2]
    yTest = separateTrainNTest(X1,y,(i+5)*0.01)[3]
    MSETrain = MSEregressionOfTrain(XTrain,yTrain)[1]
    thetaTrain = MSEregressionOfTrain(XTrain,yTrain)[0]
    MSETest = MSEregressionOfTest(thetaTrain,XTest,yTest)
    MSErecord.append([MSETrain,MSETest])
MSErecord = np.asarray(MSErecord)
print(MSErecord.shape)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: FutureWarning:
`rcond` parameter will change to the default of machine precision times ``max(M,
N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  This is separate from the ipykernel package so we can avoid doing imports
until

(91, 2)
```

[27]:
```
# plot the MSE values into a plot
import matplotlib.pyplot as plt

xAxis = np.array(range(5,96))
# plot the train MSE values into a plot
plt.plot(xAxis,MSErecord[:,0],'g--')
# plot the test MSE values into a plot
plt.plot(xAxis,MSErecord[:,1], 'o--')
plt.title('Comparison of Train(green)/Test(blue) MSE based on different size')
plt.xlabel('Size of Percentage of Training Set')
plt.ylabel('MSE Value')

# We see that with the size of training set increase, the MSE goes down a␣
 ↪little bit, since more sets of
# data means the regression becomes more precise in the first place. However,␣
 ↪when the data is large enough
# the regression line would become relatively stable. So, the training MSE goes␣
 ↪up a little since more data
# comes with more possible error.
# We see that as the size of training sets increase, the MSE for test increase␣
 ↪as well. Since the increase
# of training set size means the MSE optimization more fit the training set.␣
 ↪So, the test MSE goes up.
```

[27]:
```
Text(0, 0.5, 'MSE Value')
```

[28]:
```
### Tasks  Classification (week 2):
### 8. First, lets train a predictor that estimates whether a review
### is verified using the rating and the length:
###     p(review is verified)  (0 + 1 Œ [star rating] + 2 Œ [review length])
### Train a logistic regressor to make the above prediction
### (you may use a logistic regression library with de- fault parameters,
###  e.g. linear model.LogisticRegression() from sklearn).
### Report the classification accuracy of this predictor.
### Report also the proportion of labels that are positive
### (i.e., the proportion of reviews that are verified) and
### the proportion of predictions that are positive (1 mark).

# prepare the data for logistic classification
yLogistic = X[:,1].T
#print(yLogistic)
#print(X.shape)
XLogistic = []
XLogistic = np.asarray(XLogistic)
XLogistic = np.column_stack((X[:,0],y.T, X[:,2]))
#print(XLogistic)
# prepare the training data for logistic classification
XLTrain = XLogistic[0: round(0.9*(XLogistic.shape[0])),]
```

```
XLTest = XLogistic[round(0.9*(XLogistic.shape[0])):,]
print(XLTrain.shape)
print(XLTest.shape)
yLtrain = yLogistic[0,0: round(0.9*(yLogistic.shape[1]))]
yLtest = yLogistic[0,round(0.9*(yLogistic.shape[1])):]
print(yLtrain.shape)
print(yLtest.shape)
```

```
(133479, 3)
(14831, 3)
(1, 133479)
(1, 14831)
```

[29]:
```
from sklearn.linear_model import LogisticRegression
```

[30]:
```
clf = LogisticRegression(random_state=0, solver='lbfgs',multi_class='ovr').
 ↪fit(XLTrain, yLtrain.T)
```

```
//anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py:724:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[31]:
```
clf.predict(XLTest)
print('The proportion of positive labels:',sum(clf.predict(XLTest))/14831)
proportionOfVerified = sum(yLtest.T)/14831
print('The proportion of verified review purchase:',proportionOfVerified[0,0])
clf.score(XLTest, yLtest.T)
print('The accuracy for prediction based on star rating and review length is:
 ↪',clf.score(XLTest, yLtest.T))
```

```
The proportion of positive labels: 0.9989886049490931
The proportion of verified review purchase: 0.5595711684984155
The accuracy for prediction based on star rating and review length is:
0.5597734475085968
```

[32]:
```
from datetime import datetime
date = []
for i in range(len(data)):
      date.append(data[i][-1])
date = date[1:]
#print(date)

fmt = '%Y-%m-%d'
d = datetime.strptime('2014-10-14',fmt) #start date
for i in range(len(date)):
```

```
        diff = datetime.strptime(date[i],fmt) - d
        date[i] = diff.days

# date = [datetime.strptime(x,'%Y-%m-%d') for x in date]
min(date)
```

[32]: -3652

[33]:
```
# prepare the data for logistic classification over my own chosen variables
# I choose the review date
yLogistic1 = X[:,1].T
XLogistic1 = []
XLogistic1 = np.asarray(XLogistic1)
XLogistic1 = np.column_stack((X[:,0],date))
#print(XLogistic1)

# prepare the training data for logistic classification

XLTrain1 = XLogistic1[0: round(0.9*(XLogistic1.shape[0])),]
XLTest1 = XLogistic1[round(0.9*(XLogistic1.shape[0])):,]
#print(XLTrain1.shape)
#print(XLTest1.shape)

yLtrain1 = yLogistic1[0,0: round(0.9*(yLogistic1.shape[1]))]
yLtest1 = yLogistic1[0,round(0.9*(yLogistic1.shape[1])):]
#print(yLtrain1.shape)
#print(yLtest1.shape)
```

[34]:
```
clf1 = LogisticRegression(random_state=0, solver='lbfgs',multi_class='ovr').
 ↪fit(XLTrain1, yLtrain1.T)
```

```
//anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py:724:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[35]:
```
clf1.predict(XLTest1)
sum(clf1.predict(XLTest1))
print('The accuracy for prediction based on review date is:',clf1.
 ↪score(XLTest1, yLtest1.T))
print('The proportion of positive label prediction based on review date is:
 ↪',sum(clf1.predict(XLTest1))/14831)
proportionOfVerified1 = sum(yLtest1.T)/14831
print('The proportion of verified review purchase:',proportionOfVerified1[0,0])
```

```
The accuracy for prediction based on review date is: 0.5638190277122244
The proportion of positive label prediction based on review date is:
```

```
0.9907625918683838
The proportion of verified review purchase: 0.5595711684984155
```