Logistic_Regression

June 19, 2021

[19]:
```
import sys
!{sys.executable} -m pip install ipynb

import warnings
warnings.filterwarnings('ignore')
```

```
Requirement already satisfied: ipynb in
c:\users\gampl\appdata\local\programs\python\python39\lib\site-packages (0.5.1)

WARNING: You are using pip version 21.1.1; however, version 21.1.2 is available.
You should consider upgrading via the
'c:\users\gampl\appdata\local\programs\python\python39\python.exe -m pip install
--upgrade pip' command.
```

[20]:
```
%%capture

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from ipynb.fs.full.CleaningData import getDataset
from ipynb.fs.full.CleaningData import getCovarianceVector
from sklearn.model_selection import train_test_split
from copy import deepcopy

pd.set_option("display.max_rows", 10000)
pd.set_option("display.max_columns", None)
```

[ ]:
```
%%time
df = getDataset(500)
```

[ ]:
```
%%time
covVec = getCovarianceVector(df)

fig = plt.figure(figsize=(16, 6)) # the figsize changes the width and height␣
↪respectively
ax = fig.add_axes([0,0,1,1])
langs = covVec.index
students = covVec
ax.bar(langs,students, align='edge', width=0.7) #width determines width of bars
```

```
plt.xticks(rotation = 90)
# plt.show()
```

[ ]:
```
# Seperating and storing y values (Status Feature)

df.head(10)

statusColumn = df['status']
del df['status']
urlColumn = df['url']
del df['url']

# Printing the feature set

print(df.columns)

# Storing the number of features

numFeatures = (len(df.columns))
# print(numFeatures)
```

[24]:
```
theta = np.random.uniform(-0.5,0.5,numFeatures) # Initialising random theta␣
↪values
origTheta = theta # Keeping a copy of original theta values

# Splitting data into test, validation and train
trainX, testX, trainY, testY = train_test_split(df, statusColumn, test_size=0.
↪15)
trainX, validationX, trainY, validationY = train_test_split(trainX, trainY,␣
↪test_size=0.2)
```

[25]:
```
%%time
def sigmoid(x,k): #Sigmoid function

    if(k==0):
        f = 1
    else:
        f = k
    return 1/(1+np.exp(-f*x))

def prediction(theta, x, k): #IPrediction function
    return sigmoid(np.matmul(np.transpose(theta),x),k)

def LogisticRegression(trainX, trainY, theta, a, regularisation): # Logistic␣
↪Regression Algorithm
    alpha = a
    newTheta = theta
```

```
    strengthOfRegularisation = regularisation
    count = 0

    while(count < 10):
        oldTheta = newTheta
        for i in trainX.index:
            for j in range(numFeatures):
                newTheta[j] = newTheta[j] + alpha*(trainY.loc[i] -␣
↪prediction(newTheta,trainX.loc[i],strengthOfRegularisation))*(trainX.
↪loc[i][j]) + alpha*strengthOfRegularisation*newTheta[j]
        count = count + 1

    return newTheta

def calculateAccuracy(x,y,theta,confusionMatrix, k): # Calculates the accuracy␣
↪and create the confusion matrix
    numberCorrect = 0
    numberIncorrect = 0

    for i in range(len(testY)):

        predicted_result = round(prediction(newTheta,testX.iloc[i], k))

        if(predicted_result == 1 and testY.iloc[i] == 0):
            confusionMatrix[0] = confusionMatrix[0] + 1
        else:
            if (predicted_result == 0 and testY.iloc[i] == 1):
                confusionMatrix[1] = confusionMatrix[1] + 1
            else:
                if(predicted_result == 1 and testY.iloc[i] == 1):
                    confusionMatrix[2] = confusionMatrix[2] + 1
                else:
                    if(predicted_result == 0 and testY.iloc[i] == 0):
                        confusionMatrix[3] = confusionMatrix[3] + 1

        if(predicted_result == testY.iloc[i]):
            numberCorrect = numberCorrect + 1
        else:
            numberIncorrect = numberIncorrect + 1

    averageCorrect = numberCorrect/len(testY)
    averageIncorrect = numberIncorrect/len(testY)

    return averageCorrect*100, averageIncorrect*100, confusionMatrix
```

```
Wall time: 0 ns
```

[26]:

```python
%%time

results = [] # Matrix to store validation results
confusionMatrix = [0,0,0,0] # Confusion Matrix
alphaArray = [0.01, 0.001, 0.0001] # Alpha Values
regularisationArray = [0, 0.001, 0.002, 0.01, 0.03] # Regularisation values
newTheta = []

theta = np.random.uniform(-0.5,0.5,numFeatures) # Initialising random theta␣
↪values
origTheta = deepcopy(theta) # Keeping a copy of original theta values

# Obtaining the optimal hyperparameters by getting the accuracy and confusion␣
↪matrix for each hyperparameter

r = 0
for i in range(3):
    for j in range(5):
        newTheta = LogisticRegression(validationX, validationY, theta,␣
↪alphaArray[i], regularisationArray[j])
        results.append(calculateAccuracy(testX,testY,newTheta,confusionMatrix,␣
↪0))
        print("Alpha = ",alphaArray[i])
        print("Regularisation = ", regularisationArray[j])
        print("==============================================")
        print("Confusion Matrix",'\n' )
        print(results[r][2][2],"    ",results[r][2][1] )
        print(results[r][2][0],"    ",results[r][2][3] )
        print("==============================================")
        print("Percentage Correct : ", results[r][0])
        print("Percentage Incorrect : ", results[r][1])
        print("==============================================",'\n')
        theta = deepcopy(origTheta)
        r = r + 1
        confusionMatrix = [0,0,0,0]
```

```
Alpha =  0.01
Regularisation =  0
==============================================
Confusion Matrix

815     78
33     797
==============================================
Percentage Correct :  93.55774811375508
Percentage Incorrect :  6.4422518862449225
==============================================
```

```
Alpha =  0.01
Regularisation =  0.001
==============================================
Confusion Matrix

739     154
57     773
==============================================
Percentage Correct :  87.7539175856065
Percentage Incorrect :  12.2460824143935
==============================================

Alpha =  0.01
Regularisation =  0.002
==============================================
Confusion Matrix

748     145
55     775
==============================================
Percentage Correct :  88.39233894370284
Percentage Incorrect :  11.607661056297156
==============================================

Alpha =  0.01
Regularisation =  0.01
==============================================
Confusion Matrix

784     109
51     779
==============================================
Percentage Correct :  90.71387115496228
Percentage Incorrect :  9.286128845037725
==============================================

Alpha =  0.01
Regularisation =  0.03
==============================================
Confusion Matrix

787     106
41     789
==============================================
Percentage Correct :  91.4683691236216
Percentage Incorrect :  8.53163087637841
==============================================
```

```
Alpha =  0.001
Regularisation =  0
===============================================
Confusion Matrix

806     87
31     799
===============================================
Percentage Correct :  93.15147997678469
Percentage Incorrect :  6.848520023215323
===============================================

Alpha =  0.001
Regularisation =  0.001
===============================================
Confusion Matrix

732     161
55     775
===============================================
Percentage Correct :  87.46372605919908
Percentage Incorrect :  12.53627394080093
===============================================

Alpha =  0.001
Regularisation =  0.002
===============================================
Confusion Matrix

733     160
54     776
===============================================
Percentage Correct :  87.57980266976205
Percentage Incorrect :  12.420197330237958
===============================================

Alpha =  0.001
Regularisation =  0.01
===============================================
Confusion Matrix

739     154
53     777
===============================================
Percentage Correct :  87.98607080673244
Percentage Incorrect :  12.013929193267558
===============================================
```

```
Alpha =  0.001
Regularisation =  0.03
==============================================
Confusion Matrix

748     145
51     779
==============================================
Percentage Correct :  88.62449216482878
Percentage Incorrect :  11.375507835171213
==============================================

Alpha =  0.0001
Regularisation =  0
==============================================
Confusion Matrix

661     232
78     752
==============================================
Percentage Correct :  82.0081253627394
Percentage Incorrect :  17.99187463726059
==============================================

Alpha =  0.0001
Regularisation =  0.001
==============================================
Confusion Matrix

669     224
67     763
==============================================
Percentage Correct :  83.11085316308764
Percentage Incorrect :  16.88914683691236
==============================================

Alpha =  0.0001
Regularisation =  0.002
==============================================
Confusion Matrix

669     224
67     763
==============================================
Percentage Correct :  83.11085316308764
Percentage Incorrect :  16.88914683691236
==============================================
```

```
Alpha =  0.0001
Regularisation =  0.01
==============================================
Confusion Matrix

670     223
67     763
==============================================
Percentage Correct :  83.16889146836913
Percentage Incorrect :  16.831108531630875
==============================================

Alpha =  0.0001
Regularisation =  0.03
==============================================
Confusion Matrix

678     215
69     761
==============================================
Percentage Correct :  83.51712130005804
Percentage Incorrect :  16.48287869994196
==============================================

Wall time: 51min 22s
```

[28]:

```
%%time
trainingResults = []
confusionMatrix = [0,0,0,0]
newTheta = []

theta = np.random.uniform(-0.5,0.5,numFeatures) # Initialising random theta␣
↪values
origTheta = deepcopy(theta) # Keeping a copy of original theta values

# Obtaining the optimal values for theta by getting the accuracy and confusion␣
↪matrix using the tuned hyperparameters

newTheta = LogisticRegression(trainX, trainY, theta, 0.01, 0)
trainingResults.append(calculateAccuracy(trainX, trainY, newTheta,␣
↪confusionMatrix, 0))
```

```
Wall time: 13min 19s
```

[30]:

```
%%time
# Obtaining the test results by using new data on the model with trained theta␣
↪values
```

```
confusionMatrix = [0,0,0,0]

testResults = []
testResults.append(calculateAccuracy(testX, testY, newTheta, confusionMatrix,␣
 ↪0))
print("Alpha = ",0.01)
print("Regularisation = ", 0)
print("==============================================")
print("Confusion Matrix",'\n' )
print(testResults[0][2][2],"   ",testResults[0][2][1] )
print(testResults[0][2][0],"   ",testResults[0][2][3] )
print("==============================================")
print("Percentage Correct : ", testResults[0][0])
print("Percentage Incorrect : ", testResults[0][1],)
print("==============================================",'\n')
```

```
Alpha =  0.01
Regularisation =  0
==============================================
Confusion Matrix

820     73
34     796
==============================================
Percentage Correct :  93.78990133488102
Percentage Incorrect :  6.210098665118979
==============================================

Wall time: 184 ms
```

```
[ ]:
```