logisticRression

October 2, 2021

[ ]:
```
import numpy
import pandas
from IPython.display import display
import math
import sqlitewrapper

# data base for caching of beta's
dbObj = sqlitewrapper.SqliteCipher(password="none")

try:
    dbObj.createTable("cache" , [["DataBase" , "TEXT"] , ["cachedBeta" ,␣
 ↪"LIST"] , ["maxIteration" , "TEXT"] , ["alpha" , "TEXT"]])
except ValueError:
    pass
```

[ ]:
```
myData = pandas.read_csv("Iris.csv")

# remove any null values
myData.dropna()

display(myData)
```

```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   …              …             …              …             …
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
```

```
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              …
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

[ ]:
```
yLabel = myData["Species"]

# select important cols only
myData = myData[["SepalLengthCm" , "SepalWidthCm" , "PetalLengthCm" ,␣
 ↪"PetalWidthCm"]]
display(myData)

display(yLabel)
```

```
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0              5.1           3.5            1.4           0.2
1              4.9           3.0            1.4           0.2
2              4.7           3.2            1.3           0.2
3              4.6           3.1            1.5           0.2
4              5.0           3.6            1.4           0.2
..             …             …              …             …
145            6.7           3.0            5.2           2.3
146            6.3           2.5            5.0           1.9
147            6.5           3.0            5.2           2.0
148            6.2           3.4            5.4           2.3
149            5.9           3.0            5.1           1.8

[150 rows x 4 columns]
```

```
0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
            …
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object
```

[ ]:
```
# normalize data using min max normalization
for columnName, columnData in myData.iteritems():
    maxI = max(columnData)
    minI = min(columnData)

    tempList = []

    for j in range(len(columnData)):
        xStar = ( ((columnData[j] - minI) / (maxI - minI)) * (1 - 0) ) + 0
        tempList.append(xStar)

    myData[columnName] = tempList

display(myData)
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	0.222222	0.625000	0.067797	0.041667
1	0.166667	0.416667	0.067797	0.041667
2	0.111111	0.500000	0.050847	0.041667
3	0.083333	0.458333	0.084746	0.041667
4	0.194444	0.666667	0.067797	0.041667
..	…	…	…	…
145	0.666667	0.416667	0.711864	0.916667
146	0.555556	0.208333	0.677966	0.750000
147	0.611111	0.416667	0.711864	0.791667
148	0.527778	0.583333	0.745763	0.916667
149	0.444444	0.416667	0.694915	0.708333

[150 rows x 4 columns]

[ ]:
```
# convert the X to matrix
# [
#     1 X11        X1k
#     1 X21        X2k
#     1 X22        X3k


#     1 Xn1        Xnk
# ]

xMat = []

for i in myData.index:
    templist = [1]

    for j in myData.iloc[i]:
        templist.append(j)
```

```
    xMat.append(templist)


xMat = numpy.array(xMat)

display(xMat)
```

```
array([[1.        , 0.22222222, 0.625     , 0.06779661, 0.04166667],
       [1.        , 0.16666667, 0.41666667, 0.06779661, 0.04166667],
       [1.        , 0.11111111, 0.5       , 0.05084746, 0.04166667],
       [1.        , 0.08333333, 0.45833333, 0.08474576, 0.04166667],
       [1.        , 0.19444444, 0.66666667, 0.06779661, 0.04166667],
       [1.        , 0.30555556, 0.79166667, 0.11864407, 0.125     ],
       [1.        , 0.08333333, 0.58333333, 0.06779661, 0.08333333],
       [1.        , 0.19444444, 0.58333333, 0.08474576, 0.04166667],
       [1.        , 0.02777778, 0.375     , 0.06779661, 0.04166667],
       [1.        , 0.16666667, 0.45833333, 0.08474576, 0.        ],
       [1.        , 0.30555556, 0.70833333, 0.08474576, 0.04166667],
       [1.        , 0.13888889, 0.58333333, 0.10169492, 0.04166667],
       [1.        , 0.13888889, 0.41666667, 0.06779661, 0.        ],
       [1.        , 0.        , 0.41666667, 0.01694915, 0.        ],
       [1.        , 0.41666667, 0.83333333, 0.03389831, 0.04166667],
       [1.        , 0.38888889, 1.        , 0.08474576, 0.125     ],
       [1.        , 0.30555556, 0.79166667, 0.05084746, 0.125     ],
       [1.        , 0.22222222, 0.625     , 0.06779661, 0.08333333],
       [1.        , 0.38888889, 0.75      , 0.11864407, 0.08333333],
       [1.        , 0.22222222, 0.75      , 0.08474576, 0.08333333],
       [1.        , 0.30555556, 0.58333333, 0.11864407, 0.04166667],
       [1.        , 0.22222222, 0.70833333, 0.08474576, 0.125     ],
       [1.        , 0.08333333, 0.66666667, 0.        , 0.04166667],
       [1.        , 0.22222222, 0.54166667, 0.11864407, 0.16666667],
       [1.        , 0.13888889, 0.58333333, 0.15254237, 0.04166667],
       [1.        , 0.19444444, 0.41666667, 0.10169492, 0.04166667],
       [1.        , 0.19444444, 0.58333333, 0.10169492, 0.125     ],
       [1.        , 0.25      , 0.625     , 0.08474576, 0.04166667],
       [1.        , 0.25      , 0.58333333, 0.06779661, 0.04166667],
       [1.        , 0.11111111, 0.5       , 0.10169492, 0.04166667],
       [1.        , 0.13888889, 0.45833333, 0.10169492, 0.04166667],
       [1.        , 0.30555556, 0.58333333, 0.08474576, 0.125     ],
       [1.        , 0.25      , 0.875     , 0.08474576, 0.        ],
       [1.        , 0.33333333, 0.91666667, 0.06779661, 0.04166667],
       [1.        , 0.16666667, 0.45833333, 0.08474576, 0.        ],
       [1.        , 0.19444444, 0.5       , 0.03389831, 0.04166667],
       [1.        , 0.33333333, 0.625     , 0.05084746, 0.04166667],
       [1.        , 0.16666667, 0.45833333, 0.08474576, 0.        ],
       [1.        , 0.02777778, 0.41666667, 0.05084746, 0.04166667],
```

```
[1.        , 0.22222222, 0.58333333, 0.08474576, 0.04166667],
[1.        , 0.19444444, 0.625     , 0.05084746, 0.08333333],
[1.        , 0.05555556, 0.125     , 0.05084746, 0.08333333],
[1.        , 0.02777778, 0.5       , 0.05084746, 0.04166667],
[1.        , 0.19444444, 0.625     , 0.10169492, 0.20833333],
[1.        , 0.22222222, 0.75      , 0.15254237, 0.125     ],
[1.        , 0.13888889, 0.41666667, 0.06779661, 0.08333333],
[1.        , 0.22222222, 0.75      , 0.10169492, 0.04166667],
[1.        , 0.08333333, 0.5       , 0.06779661, 0.04166667],
[1.        , 0.27777778, 0.70833333, 0.08474576, 0.04166667],
[1.        , 0.19444444, 0.54166667, 0.06779661, 0.04166667],
[1.        , 0.75      , 0.5       , 0.62711864, 0.54166667],
[1.        , 0.58333333, 0.5       , 0.59322034, 0.58333333],
[1.        , 0.72222222, 0.45833333, 0.66101695, 0.58333333],
[1.        , 0.33333333, 0.125     , 0.50847458, 0.5       ],
[1.        , 0.61111111, 0.33333333, 0.61016949, 0.58333333],
[1.        , 0.38888889, 0.33333333, 0.59322034, 0.5       ],
[1.        , 0.55555556, 0.54166667, 0.62711864, 0.625     ],
[1.        , 0.16666667, 0.16666667, 0.38983051, 0.375     ],
[1.        , 0.63888889, 0.375     , 0.61016949, 0.5       ],
[1.        , 0.25      , 0.29166667, 0.49152542, 0.54166667],
[1.        , 0.19444444, 0.        , 0.42372881, 0.375     ],
[1.        , 0.44444444, 0.41666667, 0.54237288, 0.58333333],
[1.        , 0.47222222, 0.08333333, 0.50847458, 0.375     ],
[1.        , 0.5       , 0.375     , 0.62711864, 0.54166667],
[1.        , 0.36111111, 0.375     , 0.44067797, 0.5       ],
[1.        , 0.66666667, 0.45833333, 0.57627119, 0.54166667],
[1.        , 0.36111111, 0.41666667, 0.59322034, 0.58333333],
[1.        , 0.41666667, 0.29166667, 0.52542373, 0.375     ],
[1.        , 0.52777778, 0.08333333, 0.59322034, 0.58333333],
[1.        , 0.36111111, 0.20833333, 0.49152542, 0.41666667],
[1.        , 0.44444444, 0.5       , 0.6440678 , 0.70833333],
[1.        , 0.5       , 0.33333333, 0.50847458, 0.5       ],
[1.        , 0.55555556, 0.20833333, 0.66101695, 0.58333333],
[1.        , 0.5       , 0.33333333, 0.62711864, 0.45833333],
[1.        , 0.58333333, 0.375     , 0.55932203, 0.5       ],
[1.        , 0.63888889, 0.41666667, 0.57627119, 0.54166667],
[1.        , 0.69444444, 0.33333333, 0.6440678 , 0.54166667],
[1.        , 0.66666667, 0.41666667, 0.6779661 , 0.66666667],
[1.        , 0.47222222, 0.375     , 0.59322034, 0.58333333],
[1.        , 0.38888889, 0.25      , 0.42372881, 0.375     ],
[1.        , 0.33333333, 0.16666667, 0.47457627, 0.41666667],
[1.        , 0.33333333, 0.16666667, 0.45762712, 0.375     ],
[1.        , 0.41666667, 0.29166667, 0.49152542, 0.45833333],
[1.        , 0.47222222, 0.29166667, 0.69491525, 0.625     ],
[1.        , 0.30555556, 0.41666667, 0.59322034, 0.58333333],
[1.        , 0.47222222, 0.58333333, 0.59322034, 0.625     ],
[1.        , 0.66666667, 0.45833333, 0.62711864, 0.58333333],
```

```
[1.        , 0.55555556, 0.125     , 0.57627119, 0.5       ],
[1.        , 0.36111111, 0.41666667, 0.52542373, 0.5       ],
[1.        , 0.33333333, 0.20833333, 0.50847458, 0.5       ],
[1.        , 0.33333333, 0.25      , 0.57627119, 0.45833333],
[1.        , 0.5       , 0.41666667, 0.61016949, 0.54166667],
[1.        , 0.41666667, 0.25      , 0.50847458, 0.45833333],
[1.        , 0.19444444, 0.125     , 0.38983051, 0.375     ],
[1.        , 0.36111111, 0.29166667, 0.54237288, 0.5       ],
[1.        , 0.38888889, 0.41666667, 0.54237288, 0.45833333],
[1.        , 0.38888889, 0.375     , 0.54237288, 0.5       ],
[1.        , 0.52777778, 0.375     , 0.55932203, 0.5       ],
[1.        , 0.22222222, 0.20833333, 0.33898305, 0.41666667],
[1.        , 0.38888889, 0.33333333, 0.52542373, 0.5       ],
[1.        , 0.55555556, 0.54166667, 0.84745763, 1.        ],
[1.        , 0.41666667, 0.29166667, 0.69491525, 0.75      ],
[1.        , 0.77777778, 0.41666667, 0.83050847, 0.83333333],
[1.        , 0.55555556, 0.375     , 0.77966102, 0.70833333],
[1.        , 0.61111111, 0.41666667, 0.81355932, 0.875     ],
[1.        , 0.91666667, 0.41666667, 0.94915254, 0.83333333],
[1.        , 0.16666667, 0.20833333, 0.59322034, 0.66666667],
[1.        , 0.83333333, 0.375     , 0.89830508, 0.70833333],
[1.        , 0.66666667, 0.20833333, 0.81355932, 0.70833333],
[1.        , 0.80555556, 0.66666667, 0.86440678, 1.        ],
[1.        , 0.61111111, 0.5       , 0.69491525, 0.79166667],
[1.        , 0.58333333, 0.29166667, 0.72881356, 0.75      ],
[1.        , 0.69444444, 0.41666667, 0.76271186, 0.83333333],
[1.        , 0.38888889, 0.20833333, 0.6779661 , 0.79166667],
[1.        , 0.41666667, 0.33333333, 0.69491525, 0.95833333],
[1.        , 0.58333333, 0.5       , 0.72881356, 0.91666667],
[1.        , 0.61111111, 0.41666667, 0.76271186, 0.70833333],
[1.        , 0.94444444, 0.75      , 0.96610169, 0.875     ],
[1.        , 0.94444444, 0.25      , 1.        , 0.91666667],
[1.        , 0.47222222, 0.08333333, 0.6779661 , 0.58333333],
[1.        , 0.72222222, 0.5       , 0.79661017, 0.91666667],
[1.        , 0.36111111, 0.33333333, 0.66101695, 0.79166667],
[1.        , 0.94444444, 0.33333333, 0.96610169, 0.79166667],
[1.        , 0.55555556, 0.29166667, 0.66101695, 0.70833333],
[1.        , 0.66666667, 0.54166667, 0.79661017, 0.83333333],
[1.        , 0.80555556, 0.5       , 0.84745763, 0.70833333],
[1.        , 0.52777778, 0.33333333, 0.6440678 , 0.70833333],
[1.        , 0.5       , 0.41666667, 0.66101695, 0.70833333],
[1.        , 0.58333333, 0.33333333, 0.77966102, 0.83333333],
[1.        , 0.80555556, 0.41666667, 0.81355932, 0.625     ],
[1.        , 0.86111111, 0.33333333, 0.86440678, 0.75      ],
[1.        , 1.        , 0.75      , 0.91525424, 0.79166667],
[1.        , 0.58333333, 0.33333333, 0.77966102, 0.875     ],
[1.        , 0.55555556, 0.33333333, 0.69491525, 0.58333333],
[1.        , 0.5       , 0.25      , 0.77966102, 0.54166667],
```

```
       [1.        , 0.94444444, 0.41666667, 0.86440678, 0.91666667],
       [1.        , 0.55555556, 0.58333333, 0.77966102, 0.95833333],
       [1.        , 0.58333333, 0.45833333, 0.76271186, 0.70833333],
       [1.        , 0.47222222, 0.41666667, 0.6440678 , 0.70833333],
       [1.        , 0.72222222, 0.45833333, 0.74576271, 0.83333333],
       [1.        , 0.66666667, 0.45833333, 0.77966102, 0.95833333],
       [1.        , 0.72222222, 0.45833333, 0.69491525, 0.91666667],
       [1.        , 0.41666667, 0.29166667, 0.69491525, 0.75      ],
       [1.        , 0.69444444, 0.5       , 0.83050847, 0.91666667],
       [1.        , 0.66666667, 0.54166667, 0.79661017, 1.        ],
       [1.        , 0.66666667, 0.41666667, 0.71186441, 0.91666667],
       [1.        , 0.55555556, 0.20833333, 0.6779661 , 0.75      ],
       [1.        , 0.61111111, 0.41666667, 0.71186441, 0.79166667],
       [1.        , 0.52777778, 0.58333333, 0.74576271, 0.91666667],
       [1.        , 0.44444444, 0.41666667, 0.69491525, 0.70833333]])
```

[ ]:
```
# number of beta's
kValue = len(xMat[0])

print(kValue)

# number of rows in data
nValue = len(xMat)

print(nValue)
```

```
5
150
```

[ ]:
```
# number of unique classes
yLabelSet = list(set(yLabel))
yLabelSet = sorted(yLabelSet)

print(yLabelSet)

lenYLabelSet = len(yLabelSet)

print(lenYLabelSet)
```

```
['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
3
```

[ ]:
```
# generate Dataset = number of unique classes = U
# dataset 1 -> y = 1 for class 1 and y = 0 for other class in data set
# dataset 2 -> y = 1 for class 2 and y = 0 for other class in data set
# dataset U -> y = 1 for class U and y = 0 for other class in data set
```

```
listOfDataSets = []

# DataFrame.copy(deep=True)

for i in range(lenYLabelSet):
    newMyData = myData.copy(deep=True)

    yCol = []

    for j in newMyData.index:
        if(yLabel.iloc[j] == yLabelSet[i]):
            yCol.append(1)
        else:
            yCol.append(0)

    newMyData["Y"] = yCol

    listOfDataSets.append(newMyData)


for i in listOfDataSets:
    display(i)
```

```
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Y
0         0.222222      0.625000       0.067797      0.041667  1
1         0.166667      0.416667       0.067797      0.041667  1
2         0.111111      0.500000       0.050847      0.041667  1
3         0.083333      0.458333       0.084746      0.041667  1
4         0.194444      0.666667       0.067797      0.041667  1
..             …             …              …             … ..
145       0.666667      0.416667       0.711864      0.916667  0
146       0.555556      0.208333       0.677966      0.750000  0
147       0.611111      0.416667       0.711864      0.791667  0
148       0.527778      0.583333       0.745763      0.916667  0
149       0.444444      0.416667       0.694915      0.708333  0

[150 rows x 5 columns]
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Y
0         0.222222      0.625000       0.067797      0.041667  0
1         0.166667      0.416667       0.067797      0.041667  0
2         0.111111      0.500000       0.050847      0.041667  0
3         0.083333      0.458333       0.084746      0.041667  0
4         0.194444      0.666667       0.067797      0.041667  0
..             …             …              …             … ..
145       0.666667      0.416667       0.711864      0.916667  0
146       0.555556      0.208333       0.677966      0.750000  0
147       0.611111      0.416667       0.711864      0.791667  0
```

```
148       0.527778      0.583333       0.745763      0.916667  0
149       0.444444      0.416667       0.694915      0.708333  0

[150 rows x 5 columns]
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Y
0         0.222222      0.625000       0.067797      0.041667  0
1         0.166667      0.416667       0.067797      0.041667  0
2         0.111111      0.500000       0.050847      0.041667  0
3         0.083333      0.458333       0.084746      0.041667  0
4         0.194444      0.666667       0.067797      0.041667  0
..             …             …              …             … ..
145       0.666667      0.416667       0.711864      0.916667  1
146       0.555556      0.208333       0.677966      0.750000  1
147       0.611111      0.416667       0.711864      0.791667  1
148       0.527778      0.583333       0.745763      0.916667  1
149       0.444444      0.416667       0.694915      0.708333  1

[150 rows x 5 columns]
```

[ ]:
```
useCaching = True
```

[ ]:
```
# %%
# BetaK's for each dataSet

# betaK's =

# beta[j] = beta[j] - alpha * slope

# slope =  ( 1/n ) * Submission 1 to n {(f[xi] - yi) * xij}

# f[xi] = 1 / 1 + e^z

# z = -1 * (beta0 + beta1*Xi1 +              + betak*Xik)


# getDataSets From cache
_ , dataSetsFromCache = dbObj.getDataFromTable("cache" , omitID=True)


betasList = []


for dataSetCount , dataSet in enumerate(listOfDataSets):

    alpha = 0.2

    # max iterations to find the beta
```

```
    maxIteration = 50000

    # accuracy of beta needed
    betaErrorTolerance = 5

    is_dataSetInCache = False
    betaCache = []

    for dataSetFromCache in dataSetsFromCache:
        if(str(dataSet) == str(dataSetFromCache[0])):
            if((dataSetFromCache[2] == str(maxIteration)) and␣
↪(dataSetFromCache[3] == str(alpha)) and (useCaching)):
                is_dataSetInCache = True
                betaCache = dataSetFromCache[1]

    if(is_dataSetInCache):
        print("\n\nusing data set from cache for class {}. On {} / {}".
↪format(yLabelSet[dataSetCount] , dataSetCount , lenYLabelSet))
        betasList.append(betaCache)
        print()
        print(betaCache)

    else:

        y = dataSet["Y"]

        print("\n\nprocessing data set for class {}. On {} / {}".
↪format(yLabelSet[dataSetCount] , dataSetCount , lenYLabelSet))


        # init bk's as zero
        # init tempk's as zero
        betaKs = [0 for _ in range(kValue)]
        tempKs = [0 for _ in range(kValue)]
        slopeKs = [0 for _ in range(kValue)]


        for iteration in range(maxIteration):

            breakLoop = False

            for j in range(kValue):
                slope = 1 / nValue
                submission = 0
                k = 1
```

```
                for i in range(nValue):
                    innerSubmission = betaKs[0]

                    for k in range(1 , kValue):
                        innerSubmission = innerSubmission + (betaKs[k] *␣
↪xMat[i][k])

                    innerSubmission = innerSubmission * -1

                    innerSubmission = 1 / (1 + math.exp(innerSubmission))

                    innerSubmission = innerSubmission - y[i]

                    innerSubmission = innerSubmission * xMat[i][j]

                    submission = submission + innerSubmission

                slope = submission * slope

                tempKs[j] = betaKs[j] - (alpha * slope)

                slopeKs[j] = slope


            betaToleranceReached = 0

            for a,b in zip(betaKs , tempKs):
                if(round(a , betaErrorTolerance) == round(b ,␣
↪betaErrorTolerance)):
                    breakLoop = True
                else:
                    breakLoop = False
                    betaToleranceReached = round(b , betaErrorTolerance) -␣
↪round(a , betaErrorTolerance)

            print("\ron {} , betaToleranced = {}".format(iteration ,␣
↪betaToleranceReached) , end="")

            if(breakLoop):
                print("break on" , iteration)
                break


            # assign bj = tempj
            for j in range(kValue):
                betaKs[j] = tempKs[j]
```

```
        print()
        print(betaKs)

        betasList.append(betaKs)

        dbObj.insertIntoTable("cache" , [str(dataSet) , betaKs ,␣
↪str(maxIteration) , str(alpha)])



print("\n\n")

for i in betasList:
    print(i)
    print()
```

```
using data set from cache for class Iris-setosa. On 0 / 3

[4.249381374876902, -4.892713066957322, 8.930445576498897, -11.945933425773562,
-11.712327739754233]


using data set from cache for class Iris-versicolor. On 1 / 3

[1.7553809754145455, -0.71854918300746, -6.775577528352232, 7.229788424682071,
-6.224092096086556]


using data set from cache for class Iris-virginica. On 2 / 3

[-22.43321822555831, -2.819676302131558, -7.540455371909699, 18.88717822479327,
21.86899598920613]



[4.249381374876902, -4.892713066957322, 8.930445576498897, -11.945933425773562,
-11.712327739754233]

[1.7553809754145455, -0.71854918300746, -6.775577528352232, 7.229788424682071,
-6.224092096086556]

[-22.43321822555831, -2.819676302131558, -7.540455371909699, 18.88717822479327,
21.86899598920613]
```

[ ]:
```
# function to predict y based on new x input
# x is the new input to predict y
# x must be a data frame type
def hypothesisFunction(beta ,  x):

    x = numpy.array(x)

    # y = 1 / 1 + e^((b0 + b1*x1 + b2*x2 +         + bk*xk)*-1)

    yPredicted = beta[0]

    for i in range(1 , len(beta)):
        yPredicted = yPredicted + ( beta[i] * x[i-1] )

    yPredicted = yPredicted * -1
    yPredicted = 1 / (1 + math.exp(yPredicted))

    return yPredicted


# function to find the max value of yPredicted and assign the label
# returns a list containing [yPredicted , label]
def maxHypothesisFunction(betaList , yLabelSet , x):

    yPredictedList = []

    for i,j in zip(yLabelSet , betaList):

        yPredicted = hypothesisFunction(j , x)

        yPredictedList.append([yPredicted , i])

    yPredictedList = sorted(yPredictedList , key=lambda x:x[0] , reverse=True)

    # returning max yPredicted
    return yPredictedList[0]
```

[ ]:
```
# function to normalize the new test data based on original data
# here original data min max are used to normalize the data
def returnNormalisedTestData(originalData , testData):

    for columnName, columnData in testData.iteritems():

        maxI = max(originalData[columnName])
        minI = min(originalData[columnName])

        xStar = ( ((columnData - minI) / (maxI - minI)) * (1 - 0) ) + 0
```

```
        testData[columnName] = xStar

    return testData
```

[ ]:
```
myData = pandas.read_csv("Iris.csv")

# remove any null values
myData.dropna()

display(myData)

confusionMatrixList = []

print()

# build confusion matrix for each class
for yLabelSetI in yLabelSet:
    print("testing for {}".format(yLabelSetI))
    TP = 0
    TN = 0
    FN = 0
    FP = 0

    # traverse data set
    for i in myData.index:
        features = myData.iloc[i][1:-1]
        features = returnNormalisedTestData(myData[["SepalLengthCm" ,␣
↪"SepalWidthCm" , "PetalLengthCm" , "PetalWidthCm"]] , features)
        predictedLabel = maxHypothesisFunction(betasList , yLabelSet , features)
        actualValue = myData.iloc[i].Species

        # was True , predicted True
        if((actualValue == yLabelSetI) and (predictedLabel[1] == yLabelSetI)):
            TP = TP + 1

        # was false , predicted True
        elif(not((actualValue == yLabelSetI)) and (predictedLabel[1] ==␣
↪yLabelSetI)):
            FP = FP + 1

        # was True , predicted False
        elif((actualValue == yLabelSetI) and (not(predictedLabel[1] ==␣
↪yLabelSetI))):
            FN = FN + 1

        # was False , predicted False
```

```
        elif((not(actualValue == yLabelSetI)) and (not(predictedLabel[1] ==␣
↪yLabelSetI))):
            TN = TN + 1

    accuracy_confusionMatrix = (TP + TN) / (TP + TN + FP + FN)
    precision_confusionMatrix = TP / (TP + FP)

    confusionMatrixList.append([yLabelSetI , [[TP , FP] , [FN , TN]] ,␣
↪accuracy_confusionMatrix , precision_confusionMatrix])

print()

for i in confusionMatrixList:
    print("Stats for {}".format(i[0]))

    print("confusion matrix  = ")

    display(i[1])

    print("accuracy = {}".format(i[2]))
    print("precision = {}".format(i[3]))

    print("\n")
```

```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   …            …             …              …             …
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              …
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
```

```
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]


testing for Iris-setosa
testing for Iris-versicolor
testing for Iris-virginica

Stats for Iris-setosa
confusion matrix  =

[[50, 0], [0, 100]]

accuracy = 1.0
precision = 1.0


Stats for Iris-versicolor
confusion matrix  =

[[47, 3], [3, 97]]

accuracy = 0.96
precision = 0.94


Stats for Iris-virginica
confusion matrix  =

[[47, 3], [3, 97]]

accuracy = 0.96
precision = 0.94
```