All About ML

All About ML — Part 3: Logistic Regression

Dharani J Follow
Mar 11, 2020 · 6 min read

In any data set, we can have **numerical** or/and **categorical** features. We need to be careful while dealing with the response/target variable. The modelling algorithms should be picked considering if the target variable is numerical or categorical. A basic algorithm for numerical variable is Linear Regression and for categorical variable we have Logistic Regression.

We might come across many data sets where we have to predict, for example, if a patient has a disease or not. Here we are classifying into two parts — has disease or not. This is called **classification** problem. If you might be wondering why we can not use linear regression in classification, it is not appropriate in this scenario. For example, we have a data set of some features where we have the classes for response variable as 'ClassA', 'ClassB' and 'ClassC'.

As computer does not under stand text we have to convert them into numbers and we assign Y = {1, 2, 3} for {'ClassA', 'ClassB', 'ClassC'}. When we model it using Linear Regression, to minimize the error it uses Least Squares Method. In this process, the ordering of Y will have an impact on predictions. For instance, as 3>1, ClassC is prioritized compared to ClassA. Also, the difference between ClassA, ClassB(2–1=1) and ClassB, ClassC(3–2=1) is same to its eye. But it is absolutely different in reality. There is no dependency of each class and they cannot be compared with each other. Therefore, Linear Regression is not effective for categorical target variables.

Like in Linear Regression, we have to predict Y using X and a function involving coefficients. In Logistic Regression, it calculates the probability of occurrence of the event in response variable(Y). Based on a threshold value we can classify it.

But how do we find the probability of an event? We must model p(X) using a function such that it always produces output between 0 and 1 for any value of X.

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$$

When modified this equation a little, we get

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X}$$

By adjusting coefficients β0 and β1, we get desired p(X) and thus classifying it correctly. But how do we estimate β0 and β1? Like we have Least Squares in Linear Regression, **Maximum Likelihood function** is used in Logistic Regression. In this function, we try to find β0 and β1 such that the predicted probability p(xi) of class for each observation as closely as possible to the actual class. In other words, we try to find β0 and β1 such that plugging these estimates into the model for p(X) shown above, yields a number close to one for one class, and a number close to zero for the other class. This intuition can be formalized using a mathematical equation of likelihood function:

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'}))$$

Estimates β0 and β1 are picked in a such a way that this function is maximized. Detailed mathematical equations of this function is beyond the scope.

Multiple Logistic Regression:

In many data sets we not only have one predictor/independent as shown for equations above. There will be many others and the probability function also depends on all of them. So the modified equation will be:

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}$$

Using Likelihood function we can find estimations for β0, β1, . . . , βp.

Python Tutorial for Logistic Regression:

Let us apply logistic regression on 'Breast Cancer Classification' data set. We have to classify if it is 'Malign' or 'Benign'. Let us import all the dependencies and data

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

data = pd.read_csv('data.csv')
data.head()
```

```
(569, 33)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...

5 rows × 33 columns

This data set contains 569 Rows 33 Columns

```
data.columns
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

Column names

```
#Null value check in data
data.isna().sum()
```

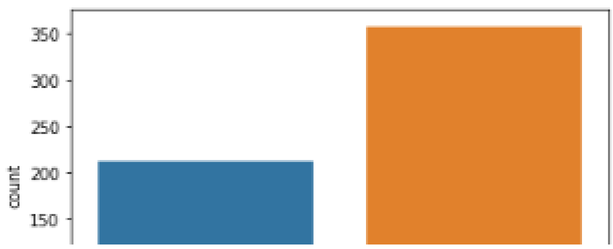
```
id                          0
diagnosis                   0
radius_mean                 0
texture_mean                0
perimeter_mean              0
area_mean                   0
smoothness_mean             0
compactness_mean            0
concavity_mean              0
concave points_mean         0
symmetry_mean               0
fractal_dimension_mean      0
radius_se                   0
texture_se                  0
perimeter_se                0
area_se                     0
smoothness_se               0
compactness_se              0
concavity_se                0
concave points_se           0
symmetry_se                 0
fractal_dimension_se        0
radius_worst                0
texture_worst               0
perimeter_worst             0
area_worst                  0
smoothness_worst            0
compactness_worst           0
concavity_worst             0
concave points_worst        0
symmetry_worst              0
fractal_dimension_worst     0
Unnamed: 32               569
```

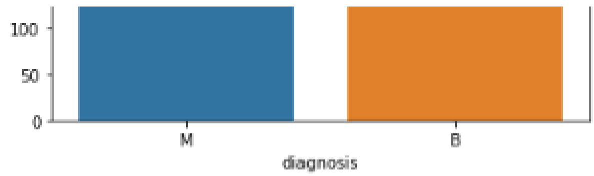
Unnamed: 32 has 569 nulls i.e., it is an empty column and should be removed and rest all are good

Let us separate the y and x in the data and remove some unwanted columns like id

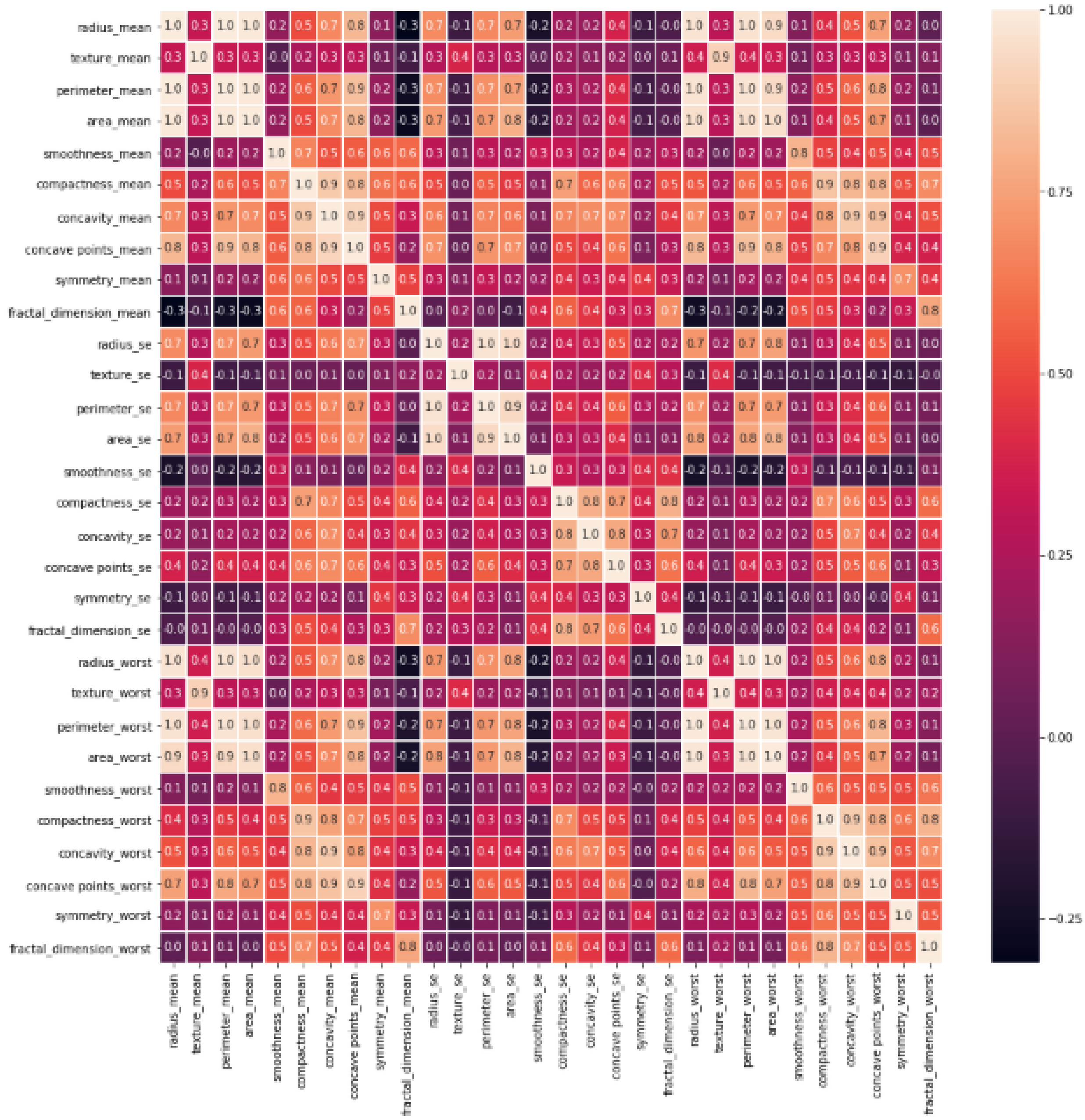
```
y = data.diagnosis #target variable
list = [‘Unnamed: 32’,’id’,’diagnosis’]
x = data.drop(list,axis = 1 ) #drop few columns
x.shape  #results in 569 rows and 30 columns

#Lets analyse the target variable

ax = sns.countplot(y,label=”Count”)
B, M = y.value_counts()
print(‘Count of Benign: ‘,B)
print(‘Count of Malignant : ‘,M)
```

```
Count of Benign:  357
Count of Malignant :  212
```





Let us observe the correlation matrix to identify co related features

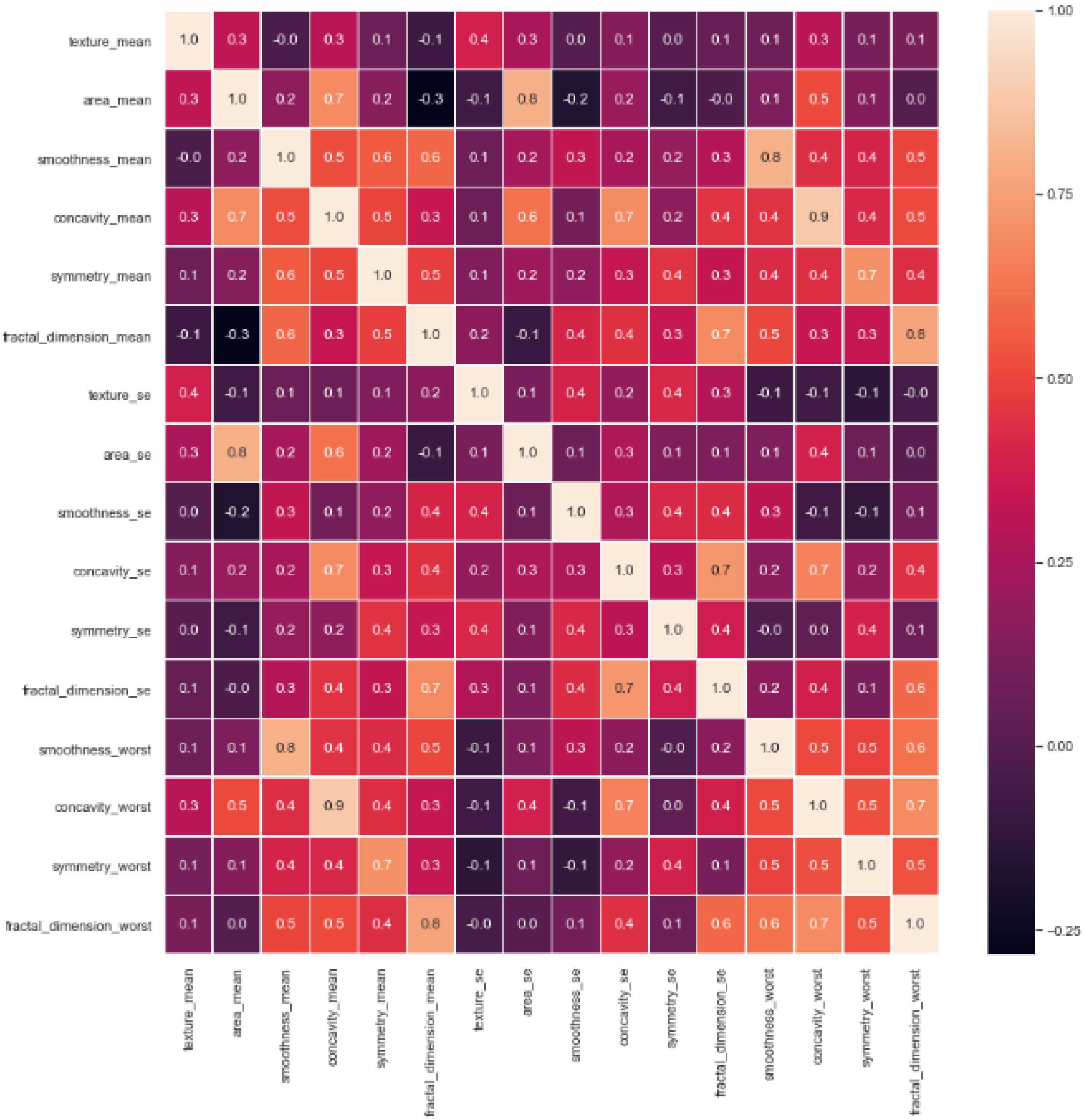



```
#From this matrix we can drop the columns with 1 as correlation
score

drop_list =
['perimeter_mean','radius_mean','compactness_mean','concave
points_mean','radius_se','perimeter_se','radius_worst','perimeter_wo
rst','compactness_worst','concave
points_worst','compactness_se','concave
points_se','texture_worst','area_worst']
```

```
x_1 = x.drop(drop_list,axis = 1 ) # do not modify x, we will use it later
```

Let us again check if there are any fields that still have correlation score of 1

```
f,ax = plt.subplots(figsize=(14, 14))
sns.heatmap(x_1.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```



There are no columns with high co relation score

The columns looks good to go ahead with modelling. First split the data for train and test with some percentage. Here I have chosen 70% train and 30% test.

```
def split_data(X,Y,size):
   x_train, x_test, y_train, y_test = train_test_split(X, Y,
test_size=size, random_state=42)
   return x_train, x_test, y_train, y_test

x_train, x_test, y_train, y_test = split_data(x_1,y,0.3)
```

Implementation of logistic regression:

```
def model_data(model,X,Y,x_test):
 model.fit(X, Y)
 x_pred = model.predict(X)      #predictions on train data
 y_pred = model.predict(x_test) #predictions on test data
 return x_pred,y_pred

logreg = LogisticRegression()
model_data(logreg,x_train,y_train,x_test)
```

Lets check the accuracy of this data and confusion matrix

```
def model_metrics(X,Y):
   confusion_matrix = metrics.confusion_matrix(Y,X)
   print(‘Accuracy: {:.2f}’.format(accuracy_score(Y,X)))
   print(‘Confusion Matrix: \n’,confusion_matrix)

print(‘Performance of logistic regression classifier on train set:’)
model_metrics(y_train,x_pred)
print(‘\n’)
print(“ – – – – – – – – – – – – – – – “)
print(‘Performance of logistic regression classifier on test set:’)
model_metrics(y_test, y_pred)
```

```
Performance of logistic regression classifier on train set:
Accuracy: 0.93
Confusion Matrix:
 [[240  19]
 [  9 130]]


------------------------------
Performance of logistic regression classifier on test set:
Accuracy: 0.96
Confusion Matrix:
 [[104   2]
 [  4  61]]
```

We see that the model has good accuracy on both train and test data. A confusion matrix shows correctness of number of observations predicted VS actual.

- In **train data set** we see that 240 and 130 are correctly predicted as Benign and Malign respectively.
- 19 are predicted as Malign when its Benign actually and 9 are predicted as Benign when it is Malign in real.
- Similarly in **test data set**, 104 and 61 are correct predictions.
- Predicted 2 observations as Malign when Actually it is Benign and predicted 4 observations as Benign when actually its Malign.

One question that might get popped — **is accuracy score well enough to understand model performance?** In our predictions given the use case in real world, it is okay if we predict Benign as Malign but if a Malign observation is predicted as Benign, that's where the problem with accuracy score comes in as it fails to capture such critical situations. There are several other Metrics used in classification to deal with this. Check them out in ***next blog.***

References: An Introduction to Statistical Learning: With Applications in R

Thank You!

Logistic Regression   Accuracy   Confusion Matrix   Classification   Machine Learning