MACHINE LEARNING

Submitted by,

- Name:Jagannath v v

- Reg.No.-21122024

- class:1st MSc DataScience

Lab Overview

Objectives

Questions:

1. Demonstrate the Logistic Regression for different penalties/regularisation methods - none, l1, l2 (you may use 'saga' solver as the parameter)
2. What happens when the Maximum Iterations are kept as 1, 2, 5, 10, 20, 50, 100, 500 and 1000? Is there any change in the accuracy.
3. Get the attributes: classes, *coef* and intercept_ and print the same in the above case.

Problem Definition

Apply Logistic Regression for Breast Cancer Dataset.

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
```

Performing Logistic Regression

Loading dataset

In [2]:
```
df=pd.read_csv(r'C:\Users\jagan\OneDrive\Desktop\data.csv')
df.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	co
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	co
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 33 columns

In [3]:
```
df.tail()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	co
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	

5 rows × 33 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
```

```
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [5]:
```
df.shape
```

Out[5]:
```
(569, 33)
```

In [6]:
```
### to check the missing values
df.isna().sum()
```

Out[6]:
```
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
dtype: int64
```

In [7]:
```
df.dropna(axis=1, inplace=True)
```

#

label encoding

In [8]:
```
df['diagnosis'].value_counts
```

Out[8]:
```
<bound method IndexOpsMixin.value_counts of 0      M
1      M
2      M
3      M
4      M
      ..
564    M
565    M
566    M
567    M
568    B
Name: diagnosis, Length: 569, dtype: object>
```

In [9]:
```
df.dtypes
```

Out[9]:
```
id                         int64
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
radius_se                float64
texture_se               float64
perimeter_se             float64
area_se                  float64
smoothness_se            float64
compactness_se           float64
concavity_se             float64
concave points_se        float64
symmetry_se              float64
fractal_dimension_se     float64
radius_worst             float64
texture_worst            float64
perimeter_worst          float64
area_worst               float64
smoothness_worst         float64
compactness_worst        float64
concavity_worst          float64
concave points_worst     float64
symmetry_worst           float64
fractal_dimension_worst  float64
dtype: object
```

Normalizing the labels

In [10]:
```
from sklearn.preprocessing import LabelEncoder
```

In [11]:
```
labelencoder = LabelEncoder()
df.iloc[:,1] = labelencoder.fit_transform(df.iloc[:,1].values)
```

Splitting the data and feature scaling

In [12]:
```
X = df.iloc[:,2:].values
y = df.iloc[:,1].values
```

In [13]:
```
from sklearn.model_selection import train_test_split
```

In [14]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.40)
```

In [15]:
```
#importing standard scalar
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

In [16]:
```
X_train
```

Out[16]:
```
array([[-0.77791603,  2.27198102, -0.82261281, ..., -1.67376413,
        -2.14202434, -1.39317017],
       [-0.44199969,  0.55408636, -0.4396182 , ...,  0.16612432,
         0.12354534,  0.57114016],
       [ 0.54837433,  0.80598373,  0.65008764, ...,  1.43190651,
         0.61017553,  2.47867512],
       ...,
       [-0.56072874, -0.99814606, -0.54682306, ..., -0.94605389,
         0.36847179, -0.40691781],
       [ 0.45570776,  0.02759817,  0.48192316, ...,  0.63792975,
        -0.09398802, -0.15581855],
       [-1.05012409, -1.61994677, -1.02945513, ..., -0.65136636,
         0.0671478 , -0.3296565 ]])
```

In [17]:
```
X_test
```

Out[17]:
```
array([[-0.84062568, -0.76506367, -0.86960634, ..., -0.8673476 ,
        -0.0942035 ,  0.10625965],
       [-0.12192206,  1.92987885, -0.12837143, ..., -0.26462778,
        -1.07249206, -0.12301821],
       [-1.04119413, -1.02195567, -1.06177835, ..., -1.34789862,
        -0.99110366, -0.92808438],
       ...,
       [-0.53698733, -0.02831679, -0.51029312, ..., -0.45087005,
        -1.28247413, -0.71073726],
       [-0.72362741, -0.20280948, -0.69600557, ..., -0.12667055,
```

```
         0.79455782, -0.10019417],
       [-1.32811844, -0.2343151 , -1.36578156, ..., -1.87106309,
        -1.62756095, -1.0370173 ]])
```

In [18]:
```
y_train
```

Out[18]:
```
array([0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0])
```

In [19]:
```
y_test
```

Out[19]:
```
array([0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0,
       1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0])
```

Logistic Regression Model

In [20]:
```
from sklearn.linear_model import LogisticRegression
classifer = LogisticRegression()
classifer.fit(X_train, y_train)
```

Out[20]:
```
LogisticRegression()
```

In [21]:
```
# prediction
```

In [22]:
```
predictions  = classifer.predict(X_test)
```

In [23]:
```
predictions
```

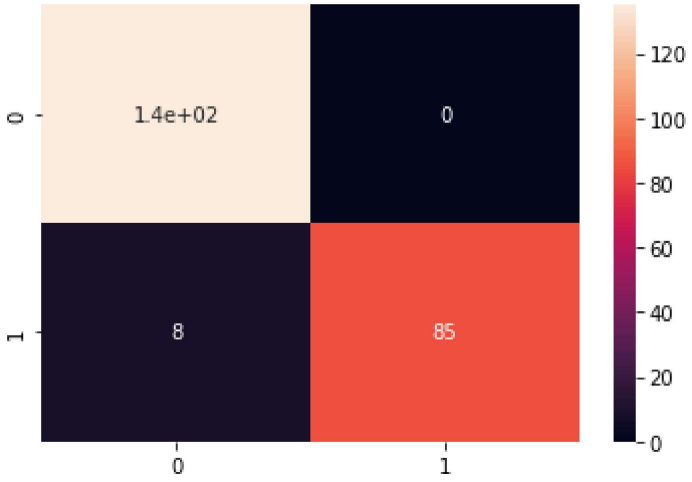
Out[23]:
```
array([0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
```

```
       0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0,
       1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0])
```

In [24]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, predictions)
print(cm)
sns.heatmap(cm, annot=True)
```

```
[[135   0]
 [  8  85]]
```

Out[24]: <AxesSubplot:>



In [25]:
```
# Accuracy
```

In [26]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,predictions))
```

```
0.9649122807017544
```

1. Demonstrate the Logistic Regression for different penalties/regularisation methods - none, l1, l2 (you may use 'saga' solver as the parameter)

In [28]:
```
x1_classifier=LogisticRegression(penalty='l2', solver='saga')
x1_classifier.fit(X_train,y_train)
predictions2  = x1_classifier.predict(X_test)
print(accuracy_score(y_test,predictions2))
```

```
0.9649122807017544
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergence
Warning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```

In [29]:
```
x1_classifier=LogisticRegression(penalty='l1', solver='saga')
x1_classifier.fit(X_train,y_train)
predictions2  = x1_classifier.predict(X_test)
print(accuracy_score(y_test,predictions2))
```

```
0.9692982456140351
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```

In [30]:
```
x1_classifier=LogisticRegression(penalty='none', solver='saga')
x1_classifier.fit(X_train,y_train)
predictions2  = x1_classifier.predict(X_test)
print(accuracy_score(y_test,predictions2))
```

```
0.9649122807017544
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```

2. What happens when the Maximum Iterations are kept as 1, 2, 5, 10, 20, 50, 100, 500 and 1000? Is there any change in the accuracy.

In [38]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

li_iter =[1,2,5,10,20,50,100,500,1000]
x_df = pd.DataFrame()
x_df['Max_iteration'] = [1,2,5,10,20,50,100,500,1000]
li2 = []
li3 = []
li4 = []
li5 = []

j = 1
for i in li_iter:
    classifer = LogisticRegression(penalty = 'l1', max_iter=i, solver = 'saga')
    classifer.fit(X_train, y_train)
    predictions  = classifer.predict(X_test)
    li2.append(accuracy_score(y_test,predictions))
    li3.append(classifer.classes_)
    li4.append(classifer.coef_)
    li5.append(classifer.intercept_)
    j += 1

x_df['Accuracy'] = li2
x_df['Classes'] = li3
x_df["Coefficient"] = li4
x_df["Intercept"] = li5

accu_df
```

```
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergence
```

```
Warning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergence
Warning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergence
Warning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergence
Warning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergence
Warning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergence
Warning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergence
Warning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergence
Warning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```

Out[38]:

	Max_iteration	Accuracy	Classes	Coefficient	Intercept
0	1	0.978070	[0, 1]	[[0.22640988246097374, 0.1930797054922324, 0.2...	[-0.19238080933748627]
1	2	0.978070	[0, 1]	[[0.29891250199731695, 0.23481326756328422, 0....	[-0.24766165273982962]
2	5	0.973684	[0, 1]	[[0.35551620312369603, 0.26804835240620944, 0....	[-0.3868290550403477]
3	10	0.969298	[0, 1]	[[0.3635969578840166, 0.32086308535804864, 0.3...	[-0.5476265045624693]
4	20	0.973684	[0, 1]	[[0.3862842192267156, 0.37555310772941053, 0.3...	[-0.6310504100822649]
5	50	0.973684	[0, 1]	[[0.34434875800227394, 0.37231648316902954, 0....	[-0.7769959307366661]
6	100	0.973684	[0, 1]	[[0.25211005706414935, 0.29944271048523974, 0....	[-0.8026969258485244]
7	500	0.964912	[0, 1]	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.379981447641...	[-0.7681990786587571]
8	1000	0.964912	[0, 1]	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.195209878073...	[-0.8004792711798449]

3. Get the attributes: classes*, coef* and intercept_ and print the same in the above case.

In [33]:

```
x1_classifier.classes_
```

Out[33]:

```
array([0, 1])
```

In [37]:

```
x1_classifier.coef_
```

Out[37]:
```
array([[ 0.32397834,  0.47490267,  0.31838556,  0.39995586,  0.13572746,
        -0.20244219,  0.73389684,  0.83025999,  0.08282472, -0.38353385,
         1.14493002, -0.13397226,  0.79948955,  0.803119  , -0.04129248,
        -0.7928175 , -0.17153889,  0.17295983, -0.45751415, -0.55126836,
         0.83769465,  1.03575343,  0.72762918,  0.80981679,  0.84653306,
         0.08998799,  0.76970489,  0.86912781,  0.59765169,  0.29508445]])
```

In [36]:
```
x1_classifier.intercept_
```

Out[36]:
```
array([-0.89014637])
```

Conclusion

In [ ]:
```
here i learned more about logistic regression
```

References

1. https://www.quora.com/What-is-regularization-in-machine-learning
2. https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a
3. https://towardsdatascience.com/regularization-an-important-concept-in-machine-learning-5891628907ea
4. https://www.geeksforgeeks.org/regularization-in-machine-learning/
5. https://analyticsindiamag.com/regularization-in-machine-learning-a-detailed-guide/
6. https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

In [ ]:
```

```