Logistic Regression

In [1]:
```
# import data set
import pandas as pd
```

In [2]:
```
from sklearn import datasets
```

In [4]:
```python
cancer = datasets.load_breast_cancer()
cancer
```

Out[4]:
```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic)
dataset\n--------------------------------------------\n\n**Data Set Character
istics:**\n\n    :Number of Instances: 569\n\n    :Number of Attributes: 30 n
umeric, predictive attributes and the class\n\n    :Attribute Information:\n
- radius (mean of distances from center to points on the perimeter)\n
- texture (standard deviation of gray-scale values)\n        - perimeter\n
- area\n        - smoothness (local variation in radius lengths)\n        - c
ompactness (perimeter^2 / area - 1.0)\n        - concavity (severity of conca
ve portions of the contour)\n        - concave points (number of concave port
ions of the contour)\n        - symmetry \n        - fractal dimension ("coas
tline approximation" - 1)\n\n        The mean, standard error, and "worst" or
largest (mean of the three\n        largest values) of these features were co
```

```
mputed for each image,\n        resulting in 30 features.  For instance, fiel
d 3 is Mean Radius, field\n        13 is Radius SE, field 23 is Worst Radiu
s.\n\n        - class:\n                - WDBC-Malignant\n                - W
DBC-Benign\n\n    :Summary Statistics:\n\n    ===============================
====== ====== ======\n                                           Min    Max\n
===================================== ====== ======\n    radius (mean):
6.981  28.11\n    texture (mean):                          9.71   39.28\n    per
imeter (mean):                     43.79  188.5\n    area (mean):
143.5  2501.0\n    smoothness (mean):                       0.053  0.163\n    co
mpactness (mean):                    0.019  0.345\n    concavity (mean):
0.0    0.427\n    concave points (mean):                   0.0    0.201\n    sym
metry (mean):                      0.106  0.304\n    fractal dimension (mea
n):             0.05   0.097\n    radius (standard error):              0.112
2.873\n    texture (standard error):             0.36   4.885\n    perimeter
(standard error):           0.757  21.98\n    area (standard error):
6.802  542.2\n    smoothness (standard error):           0.002  0.031\n    com
pactness (standard error):         0.002  0.135\n    concavity (standard erro
r):           0.0    0.396\n    concave points (standard error):      0.0
0.053\n    symmetry (standard error):            0.008  0.079\n    fractal di
mension (standard error):   0.001  0.03\n    radius (worst):
7.93   36.04\n    texture (worst):                         12.02  49.54\n    per
imeter (worst):                    50.41  251.2\n    area (worst):
185.2  4254.0\n    smoothness (worst):                      0.071  0.223\n    co
mpactness (worst):                   0.027  1.058\n    concavity (worst):
0.0    1.252\n    concave points (worst):                  0.0    0.291\n    sym
metry (worst):                     0.156  0.664\n    fractal dimension (wors
t):            0.055  0.208\n    ===================================== ======
======\n\n    :Missing Attribute Values: None\n\n    :Class Distribution: 212
- Malignant, 357 - Benign\n\n    :Creator:  Dr. William H. Wolberg, W. Nick S
treet, Olvi L. Mangasarian\n\n    :Donor: Nick Street\n\n    :Date: November,
1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) dataset
s.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of
a fine needle\naspirate (FNA) of a breast mass.  They describe\ncharacteristi
cs of the cell nuclei present in the image.\n\nSeparating plane described abo
ve was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Deci
sion Tree\nConstruction Via Linear Programming." Proceedings of the 4th\nMidw
est Artificial Intelligence and Cognitive Science Society,\npp. 97-101, 199
2], a classification method which uses linear\nprogramming to construct a dec
ision tree.  Relevant features\nwere selected using an exhaustive search in t
he space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual linear pro
gram used to obtain the separating plane\nin the 3-dimensional space is that
described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust Linear\nProgramm
ing Discrimination of Two Linearly Inseparable Sets",\nOptimization Methods a
nd Software 1, 1992, 23-34].\n\nThis database is also available through the U
W CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-lea
rn/WDBC/\n\n.. topic:: References\n\n   - W.N. Street, W.H. Wolberg and O.L.
Mangasarian. Nuclear feature extraction \n     for breast tumor diagnosis. IS
&T/SPIE 1993 International Symposium on \n     Electronic Imaging: Science an
d Technology, volume 1905, pages 861-870,\n     San Jose, CA, 1993.\n   - O.
L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and \n
prognosis via linear programming. Operations Research, 43(4), pages 570-577,
\n     July-August 1995.\n   - W.H. Wolberg, W.N. Street, and O.L. Mangasaria
n. Machine learning techniques\n     to diagnose breast cancer from fine-need
le aspirates. Cancer Letters 77 (1994) \n     163-171.',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'me
an area',
        'mean smoothness', 'mean compactness', 'mean concavity',
```

```
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension'], dtype='<U23'),
 'filename': 'C:\\Users\\Alekhya\\Anaconda3\\lib\\site-packages\\sklearn\\dat
asets\\data\\breast_cancer.csv'}
```

In [8]:

```python
# selecting features and target
input_data = pd.DataFrame(cancer["data"],columns=['mean radius', 'mean textu
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension'])
input_data.head()
```

Out[8]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	d
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 30 columns

In [9]:

```python
input_data.shape
```

Out[9]:

```
(569, 30)
```

In [11]:

```
output_data = pd.DataFrame(cancer["target"],columns=["target"])
output_data.head()
```

Out[11]:

	target
0	0
1	0
2	0
3	0
4	0

In [12]:

```
output_data.shape
```

Out[12]: (569, 1)

In [13]:

```
input_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
mean radius                569 non-null float64
mean texture               569 non-null float64
mean perimeter             569 non-null float64
mean area                  569 non-null float64
mean smoothness            569 non-null float64
mean compactness           569 non-null float64
mean concavity             569 non-null float64
mean concave points        569 non-null float64
mean symmetry              569 non-null float64
mean fractal dimension     569 non-null float64
radius error               569 non-null float64
texture error              569 non-null float64
perimeter error            569 non-null float64
area error                 569 non-null float64
smoothness error           569 non-null float64
compactness error          569 non-null float64
concavity error            569 non-null float64
concave points error       569 non-null float64
symmetry error             569 non-null float64
fractal dimension error    569 non-null float64
worst radius               569 non-null float64
worst texture              569 non-null float64
worst perimeter            569 non-null float64
worst area                 569 non-null float64
worst smoothness           569 non-null float64
worst compactness          569 non-null float64
worst concavity            569 non-null float64
worst concave points       569 non-null float64
worst symmetry             569 non-null float64
worst fractal dimension    569 non-null float64
dtypes: float64(30)
memory usage: 133.4 KB
```

In [14]:
```
output_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 1 columns):
target    569 non-null int32
dtypes: int32(1)
memory usage: 2.3 KB
```

In [15]:
```
# applying preprocessing technique
input_data.isnull().sum()
```

Out[15]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
dtype: int64
```

In [16]:
```
output_data.isnull().sum()
```

Out[16]:
```
target    0
dtype: int64
```

In [17]:
```
output_data["target"].value_counts()
```

Out[17]:
```
1    357
0    212
Name: target, dtype: int64
```

In [18]:
```
# seperating data for training and testing
from sklearn.model_selection import train_test_split
```

In [20]:
```
x_train,x_test,y_train,y_test = train_test_split(input_data,output_data,test
                                                 random_state=2)
```

In [22]:
```
# select the model
from sklearn.linear_model import LogisticRegression
```

In [23]:
```
log = LogisticRegression()
```

In [24]:
```
log.fit(x_train,y_train)
```

```
C:\Users\Alekhya\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
33: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\Alekhya\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: D
ataConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[24]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [26]:
```
# predict the values for testing
pred = log.predict(x_test)
pred
```

Out[26]:
```
array([1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1])
```

In [33]:
```
# evaluating the model
from sklearn.metrics import accuracy_score,classification_report,confusion_m
```

In [34]:
```
accuracy_score(y_test,pred)
```

Out[34]:
```
0.9414893617021277
```

In [35]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.92      0.92        73
           1       0.95      0.96      0.95       115

   micro avg       0.94      0.94      0.94       188
   macro avg       0.94      0.94      0.94       188
weighted avg       0.94      0.94      0.94       188
```

In [36]:
```
confusion_matrix(y_test,pred)
```

Out[36]:
```
array([[ 67,   6],
       [  5, 110]], dtype=int64)
```

Support Vector Machine

In [81]:
```
data = pd.read_csv("https://raw.githubusercontent.com/AP-State-Skill-Develop
```

In [82]:
```
data.head()
```

Out[82]:

	survived	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [83]:
```
data.shape
```

Out[83]:
```
(891, 11)
```

In [84]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
survived    891 non-null int64
pclass      891 non-null int64
name        891 non-null object
sex         891 non-null object
age         714 non-null float64
sibsp       891 non-null int64
parch       891 non-null int64
ticket      891 non-null object
fare        891 non-null float64
cabin       204 non-null object
embarked    889 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 76.6+ KB
```

In [85]:
```
891-714
```

Out[85]: 177

In [86]:
```
891-204
```

Out[86]: 687

In [87]:
```
# preprocessing
data.isnull().sum()
```

Out[87]:
```
survived      0
pclass        0
name          0
sex           0
age         177
sibsp         0
parch         0
ticket        0
fare          0
cabin       687
embarked      2
dtype: int64
```

In [88]:
```
data.isnull().sum().sum()
```

Out[88]: 866

In [89]:
```
data.drop("cabin",axis=1,inplace=True)
```

In [90]:
```
data.columns
```

Out[90]:
```
Index(['survived', 'pclass', 'name', 'sex', 'age', 'sibsp', 'parch', 'ticket',
       'fare', 'embarked'],
      dtype='object')
```

In [91]:
```
data["age"]
```

Out[91]:
```
0      22.0
1      38.0
2      26.0
3      35.0
4      35.0
5       NaN
6      54.0
7       2.0
8      27.0
9      14.0
10      4.0
11     58.0
12     20.0
13     39.0
14     14.0
15     55.0
16      2.0
17      NaN
18     31.0
19      NaN
20     35.0
21     34.0
22     15.0
23     28.0
24      8.0
25     38.0
26      NaN
27     19.0
28      NaN
29      NaN
       ...
861    21.0
862    48.0
863     NaN
864    24.0
865    42.0
866    27.0
867    31.0
868     NaN
869     4.0
870    26.0
871    47.0
872    33.0
873    47.0
874    28.0
875    15.0
876    20.0
877    19.0
878     NaN
879    56.0
880    25.0
881    33.0
882    22.0
883    28.0
```

```
884    25.0
885    39.0
886    27.0
887    19.0
888     NaN
889    26.0
890    32.0
Name: age, Length: 891, dtype: float64
```

In [92]:
```
data.age.mean()
```

Out[92]:
```
29.69911764705882
```

In [93]:
```
round(data["age"].mean())
```

Out[93]:
```
30
```

In [94]:
```
data["age"]=data["age"].fillna(round(data["age"].mean()))
```

In [95]:
```
data.isnull().sum()
```

Out[95]:
```
survived    0
pclass      0
name        0
sex         0
age         0
sibsp       0
parch       0
ticket      0
fare        0
embarked    2
dtype: int64
```

In [96]:
```
data["embarked"].dtype
```

Out[96]:
```
dtype('O')
```

In [97]:
```
data["embarked"].value_counts()
```

Out[97]:
```
S    644
C    168
Q     77
Name: embarked, dtype: int64
```

In [98]:
```
data["embarked"]=data["embarked"].fillna("S")
```

In [99]:
```
1 data.isnull().sum()
```

Out[99]:
```
survived    0
pclass      0
name        0
sex         0
age         0
sibsp       0
parch       0
ticket      0
fare        0
embarked    0
dtype: int64
```

In [100]:
```
1 data.drop("name",axis=1,inplace=True)
```

In [101]:
```
1 data.columns
```

Out[101]:
```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'ticket', 'fare',
       'embarked'],
      dtype='object')
```

In [102]:
```
1 data.shape
```

Out[102]:
```
(891, 9)
```

In [103]:
```
1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
survived    891 non-null int64
pclass      891 non-null int64
sex         891 non-null object
age         891 non-null float64
sibsp       891 non-null int64
parch       891 non-null int64
ticket      891 non-null object
fare        891 non-null float64
embarked    891 non-null object
dtypes: float64(2), int64(4), object(3)
memory usage: 62.7+ KB
```

In [104]:
```
data.head()
```

Out[104]:

	survived	pclass	sex	age	sibsp	parch	ticket	fare	embarked
0	0	3	male	22.0	1	0	A/5 21171	7.2500	S
1	1	1	female	38.0	1	0	PC 17599	71.2833	C
2	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250	S
3	1	1	female	35.0	1	0	113803	53.1000	S
4	0	3	male	35.0	0	0	373450	8.0500	S

In [105]:
```
data["ticket"].value_counts()
```

Out[105]:
```
CA. 2343           7
347082             7
1601               7
347088             6
CA 2144            6
3101295            6
382652             5
S.O.C. 14879       5
113781             4
347077             4
PC 17757           4
17421              4
349909             4
2666               4
19950              4
4133               4
W./C. 6608         4
113760             4
LINE               4
345773             3
13502              3
363291             3
C.A. 31921         3
PC 17760           3
PC 17572           3
C.A. 34651         3
35273              3
110152             3
230080             3
239853             3
                  ..
233639             1
349221             1
F.C.C. 13528       1
2687               1
F.C.C. 13531       1
SC/AH Basle 541    1
19952              1
349243             1
2672               1
368323             1
65303              1
2647               1
PC 17601           1
350034             1
250648             1
250651             1
13509              1
349223             1
349222             1
4135               1
113786             1
350026             1
228414             1
```

```
112050            1
65304             1
C.A. 24580        1
367232            1
330919            1
14973             1
7267              1
Name: ticket, Length: 681, dtype: int64
```

In [106]:

```
data.drop("ticket",axis=1,inplace=True)
```

In [107]:

```
data.columns
```

Out[107]:

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked'],
      dtype='object')
```

In [108]:

```
from sklearn.preprocessing import LabelEncoder
```

In [109]:

```
lab = LabelEncoder()
```

In [110]:

```
data["sex"] = lab.fit_transform(data["sex"])
data["sex"].head()
```

Out[110]:

```
0    1
1    0
2    0
3    0
4    1
Name: sex, dtype: int32
```

In [111]:

```
data["embarked"] = lab.fit_transform(data["embarked"])
data["embarked"].head()
```

Out[111]:

```
0    2
1    0
2    2
3    2
4    2
Name: embarked, dtype: int32
```

In [112]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
survived    891 non-null int64
pclass      891 non-null int64
sex         891 non-null int32
age         891 non-null float64
sibsp       891 non-null int64
parch       891 non-null int64
fare        891 non-null float64
embarked    891 non-null int32
dtypes: float64(2), int32(2), int64(4)
memory usage: 48.8 KB
```

In [113]:
```
data.head()
```

Out[113]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked
0	0	3	1	22.0	1	0	7.2500	2
1	1	1	0	38.0	1	0	71.2833	0
2	1	3	0	26.0	0	0	7.9250	2
3	1	1	0	35.0	1	0	53.1000	2
4	0	3	1	35.0	0	0	8.0500	2

In [114]:
```
data.columns
```

Out[114]:
```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked'],
      dtype='object')
```

In [115]:
```
# selecting features and target
input1 = data.drop("survived",axis=1)
input1.head()
```

Out[115]:

	pclass	sex	age	sibsp	parch	fare	embarked
0	3	1	22.0	1	0	7.2500	2
1	1	0	38.0	1	0	71.2833	0
2	3	0	26.0	0	0	7.9250	2
3	1	0	35.0	1	0	53.1000	2
4	3	1	35.0	0	0	8.0500	2

In [116]:
```
data.columns
```

Out[116]:
```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked'],
      dtype='object')
```

In [117]:
```
output1 = data["survived"]
output1.head()
```

Out[117]:
```
0    0
1    1
2    1
3    1
4    0
Name: survived, dtype: int64
```

In [119]:
```
# seperating data for training and testing
from sklearn.model_selection import train_test_split
```

In [132]:
```
x_train,x_test,y_train,y_test = train_test_split(input1,output1,test_size=0.
```

In [133]:

```python
# select the model
from sklearn.svm import SVC
help(SVC)
```

```
Help on class SVC in module sklearn.svm.classes:

class SVC(sklearn.svm.base.BaseSVC)
 |  SVC(C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, sh
rinking=True, probability=False, tol=0.001, cache_size=200, class_weight=Non
e, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=No
ne)
 |  
 |  C-Support Vector Classification.
 |  
 |  The implementation is based on libsvm. The fit time complexity
 |  is more than quadratic with the number of samples which makes it hard
 |  to scale to dataset with more than a couple of 10000 samples.
 |  
 |  The multiclass support is handled according to a one-vs-one scheme.
 |  
 |  For details on the precise mathematical formulation of the provided
 |  kernel functions and how `gamma`, `coef0` and `degree` affect each
 |  other, see the corresponding section in the narrative documentation:
 |  :ref:`svm_kernels`.
 |  
 |  Read more in the :ref:`User Guide <svm_classification>`.
 |  
 |  Parameters
 |  ----------
 |  C : float, optional (default=1.0)
 |      Penalty parameter C of the error term.
 |  
 |  kernel : string, optional (default='rbf')
 |      Specifies the kernel type to be used in the algorithm.
 |      It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' o
r
 |      a callable.
 |      If none is given, 'rbf' will be used. If a callable is given it is
 |      used to pre-compute the kernel matrix from data matrices; that matrix
 |      should be an array of shape ``(n_samples, n_samples)``.
 |  
 |  degree : int, optional (default=3)
 |      Degree of the polynomial kernel function ('poly').
 |      Ignored by all other kernels.
 |  
 |  gamma : float, optional (default='auto')
 |      Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
 |  
 |      Current default is 'auto' which uses 1 / n_features,
 |      if ``gamma='scale'`` is passed then it uses 1 / (n_features * X.var
())
 |      as value of gamma. The current default of gamma, 'auto', will change
 |      to 'scale' in version 0.22. 'auto_deprecated', a deprecated version o
f
 |      'auto' is used as a default indicating that no explicit value of gamm
a
```

```
 |      was passed.
 |
 |  coef0 : float, optional (default=0.0)
 |      Independent term in kernel function.
 |      It is only significant in 'poly' and 'sigmoid'.
 |
 |  shrinking : boolean, optional (default=True)
 |      Whether to use the shrinking heuristic.
 |
 |  probability : boolean, optional (default=False)
 |      Whether to enable probability estimates. This must be enabled prior
 |      to calling `fit`, and will slow down that method.
 |
 |  tol : float, optional (default=1e-3)
 |      Tolerance for stopping criterion.
 |
 |  cache_size : float, optional
 |      Specify the size of the kernel cache (in MB).
 |
 |  class_weight : {dict, 'balanced'}, optional
 |      Set the parameter C of class i to class_weight[i]*C for
 |      SVC. If not given, all classes are supposed to have
 |      weight one.
 |      The "balanced" mode uses the values of y to automatically adjust
 |      weights inversely proportional to class frequencies in the input data
 |      as ``n_samples / (n_classes * np.bincount(y))``
 |
 |  verbose : bool, default: False
 |      Enable verbose output. Note that this setting takes advantage of a
 |      per-process runtime setting in libsvm that, if enabled, may not work
 |      properly in a multithreaded context.
 |
 |  max_iter : int, optional (default=-1)
 |      Hard limit on iterations within solver, or -1 for no limit.
 |
 |  decision_function_shape : 'ovo', 'ovr', default='ovr'
 |      Whether to return a one-vs-rest ('ovr') decision function of shape
 |      (n_samples, n_classes) as all other classifiers, or the original
 |      one-vs-one ('ovo') decision function of libsvm which has shape
 |      (n_samples, n_classes * (n_classes - 1) / 2). However, one-vs-one
 |      ('ovo') is always used as multi-class strategy.
 |
 |      .. versionchanged:: 0.19
 |          decision_function_shape is 'ovr' by default.
 |
 |      .. versionadded:: 0.17
 |         *decision_function_shape='ovr'* is recommended.
 |
 |      .. versionchanged:: 0.17
 |         Deprecated *decision_function_shape='ovo' and None*.
 |
 |  random_state : int, RandomState instance or None, optional (default=None)
 |      The seed of the pseudo random number generator used when shuffling
 |      the data for probability estimates. If int, random_state is the
 |      seed used by the random number generator; If RandomState instance,
 |      random_state is the random number generator; If None, the random
 |      number generator is the RandomState instance used by `np.random`.
```

```
|
|  Attributes
|  ----------
|  support_ : array-like, shape = [n_SV]
|      Indices of support vectors.
|
|  support_vectors_ : array-like, shape = [n_SV, n_features]
|      Support vectors.
|
|  n_support_ : array-like, dtype=int32, shape = [n_class]
|      Number of support vectors for each class.
|
|  dual_coef_ : array, shape = [n_class-1, n_SV]
|      Coefficients of the support vector in the decision function.
|      For multiclass, coefficient for all 1-vs-1 classifiers.
|      The layout of the coefficients in the multiclass case is somewhat
|      non-trivial. See the section about multi-class classification in the
|      SVM section of the User Guide for details.
|
|  coef_ : array, shape = [n_class * (n_class-1) / 2, n_features]
|      Weights assigned to the features (coefficients in the primal
|      problem). This is only available in the case of a linear kernel.
|
|      `coef_` is a readonly property derived from `dual_coef_` and
|      `support_vectors_`.
|
|  intercept_ : array, shape = [n_class * (n_class-1) / 2]
|      Constants in decision function.
|
|  fit_status_ : int
|      0 if correctly fitted, 1 otherwise (will raise warning)
|
|  probA_ : array, shape = [n_class * (n_class-1) / 2]
|  probB_ : array, shape = [n_class * (n_class-1) / 2]
|      If probability=True, the parameters learned in Platt scaling to
|      produce probability estimates from decision values. If
|      probability=False, an empty array. Platt scaling uses the logistic
|      function
|      ``1 / (1 + exp(decision_value * probA_ + probB_))``
|      where ``probA_`` and ``probB_`` are learned from the dataset [2]_. Fo
r
|      more information on the multiclass case and training procedure see
|      section 8 of [1]_.
|
|  Examples
|  --------
|  >>> import numpy as np
|  >>> X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
|  >>> y = np.array([1, 1, 2, 2])
|  >>> from sklearn.svm import SVC
|  >>> clf = SVC(gamma='auto')
|  >>> clf.fit(X, y) #doctest: +NORMALIZE_WHITESPACE
|  SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
|      decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
|      max_iter=-1, probability=False, random_state=None, shrinking=True,
|      tol=0.001, verbose=False)
|  >>> print(clf.predict([[-0.8, -1]]))
```

```
 |  [1]
 |
 |  See also
 |  --------
 |  SVR
 |      Support Vector Machine for Regression implemented using libsvm.
 |
 |  LinearSVC
 |      Scalable Linear Support Vector Machine for classification
 |      implemented using liblinear. Check the See also section of
 |      LinearSVC for more comparison element.
 |
 |  References
 |  ----------
 |  .. [1] `LIBSVM: A Library for Support Vector Machines
 |      <http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf>`_
 |
 |  .. [2] `Platt, John (1999). "Probabilistic outputs for support vector
 |      machines and comparison to regularizedlikelihood methods."
 |      <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.1639>`_
 |
 |  Method resolution order:
 |      SVC
 |      sklearn.svm.base.BaseSVC
 |      abc.NewBase
 |      sklearn.svm.base.BaseLibSVM
 |      abc.NewBase
 |      sklearn.base.BaseEstimator
 |      sklearn.base.ClassifierMixin
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', co
ef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_
weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', rando
m_state=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.svm.base.BaseSVC:
 |
 |  decision_function(self, X)
 |      Evaluates the decision function for the samples in X.
 |
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_samples, n_features)
 |
 |      Returns
 |      -------
 |      X : array-like, shape (n_samples, n_classes * (n_classes-1) / 2)
```

```
 |          Returns the decision function of the sample for each class
 |          in the model.
 |          If decision_function_shape='ovr', the shape is (n_samples,
 |          n_classes).
 |
 |      Notes
 |      ------
 |      If decision_function_shape='ovo', the function values are proportiona
l
 |      to the distance of the samples X to the separating hyperplane. If the
 |      exact distances are required, divide the function values by the norm
of
 |      the weight vector (``coef_``). See also `this question
 |      <https://stats.stackexchange.com/questions/14876/
 |      interpreting-distance-from-hyperplane-in-svm>`_ for further details.
 |
 |  predict(self, X)
 |      Perform classification on samples in X.
 |
 |      For an one-class model, +1 or -1 is returned.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix}, shape (n_samples, n_features)
 |          For kernel="precomputed", the expected shape of X is
 |          [n_samples_test, n_samples_train]
 |
 |      Returns
 |      -------
 |      y_pred : array, shape (n_samples,)
 |          Class labels for samples in X.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.svm.base.BaseSVC:
 |
 |  predict_log_proba
 |      Compute log probabilities of possible outcomes for samples in X.
 |
 |      The model need to have probability information computed at training
 |      time: fit with attribute `probability` set to True.
 |
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_samples, n_features)
 |          For kernel="precomputed", the expected shape of X is
 |          [n_samples_test, n_samples_train]
 |
 |      Returns
 |      -------
 |      T : array-like, shape (n_samples, n_classes)
 |          Returns the log-probabilities of the sample for each class in
 |          the model. The columns correspond to the classes in sorted
 |          order, as they appear in the attribute `classes_`.
 |
 |      Notes
 |      -----
 |      The probability model is created using cross validation, so
```

```
 |      the results can be slightly different than those obtained by
 |      predict. Also, it will produce meaningless results on very small
 |      datasets.
 |  
 |  predict_proba
 |      Compute probabilities of possible outcomes for samples in X.
 |      
 |      The model need to have probability information computed at training
 |      time: fit with attribute `probability` set to True.
 |      
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_samples, n_features)
 |          For kernel="precomputed", the expected shape of X is
 |          [n_samples_test, n_samples_train]
 |      
 |      Returns
 |      -------
 |      T : array-like, shape (n_samples, n_classes)
 |          Returns the probability of the sample for each class in
 |          the model. The columns correspond to the classes in sorted
 |          order, as they appear in the attribute `classes_`.
 |      
 |      Notes
 |      -----
 |      The probability model is created using cross validation, so
 |      the results can be slightly different than those obtained by
 |      predict. Also, it will produce meaningless results on very small
 |      datasets.
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.svm.base.BaseLibSVM:
 |  
 |  fit(self, X, y, sample_weight=None)
 |      Fit the SVM model according to the given training data.
 |      
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix}, shape (n_samples, n_features)
 |          Training vectors, where n_samples is the number of samples
 |          and n_features is the number of features.
 |          For kernel="precomputed", the expected shape of X is
 |          (n_samples, n_samples).
 |      
 |      y : array-like, shape (n_samples,)
 |          Target values (class labels in classification, real numbers in
 |          regression)
 |      
 |      sample_weight : array-like, shape (n_samples,)
 |          Per-sample weights. Rescale C per sample. Higher weights
 |          force the classifier to put more emphasis on these points.
 |      
 |      Returns
 |      -------
 |      self : object
 |      
 |      Notes
```

```
 |      ------
 |      If X and y are not C-ordered and contiguous arrays of np.float64 and
 |      X is not a scipy.sparse.csr_matrix, X and/or y may be copied.
 |
 |      If X is a dense array, then the other methods will not support sparse
 |      matrices as input.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.svm.base.BaseLibSVM:
 |
 |  coef_
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |
 |      Parameters
 |      ----------
 |      deep : boolean, optional
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |
 |      Returns
 |      -------
 |      params : mapping of string to any
 |          Parameter names mapped to their values.
 |
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |
 |      The method works on simple estimators as well as on nested objects
 |      (such as pipelines). The latter have parameters of the form
 |      ``<component>__<parameter>`` so that it's possible to update each
 |      component of a nested object.
 |
 |      Returns
 |      -------
 |      self
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.BaseEstimator:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
```

```
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.ClassifierMixin:
 |  
 |  score(self, X, y, sample_weight=None)
 |      Returns the mean accuracy on the given test data and labels.
 |      
 |      In multi-label classification, this is the subset accuracy
 |      which is a harsh metric since you require for each sample that
 |      each label set be correctly predicted.
 |      
 |      Parameters
 |      ----------
 |      X : array-like, shape = (n_samples, n_features)
 |          Test samples.
 |      
 |      y : array-like, shape = (n_samples) or (n_samples, n_outputs)
 |          True labels for X.
 |      
 |      sample_weight : array-like, shape = [n_samples], optional
 |          Sample weights.
 |      
 |      Returns
 |      -------
 |      score : float
 |          Mean accuracy of self.predict(X) wrt. y.
```

In [134]:

```
sv = SVC(kernel="linear")
```

In [135]:

```
# train the model
sv.fit(x_train,y_train)
```

Out[135]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [136]:
```
#predicting
p = sv.predict(x_test)
p
```

Out[136]:
```
array([0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1], dtype=int64)
```

In [137]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_r
```

In [138]:
```
accuracy_score(y_test,p)
```

Out[138]:
```
0.7756410256410257
```

In [139]:
```
confusion_matrix(y_test,p)
```

Out[139]:
```
array([[159,  24],
       [ 46,  83]], dtype=int64)
```

In [140]:
```
print(classification_report(y_test,p))
```

```
              precision    recall  f1-score   support

           0       0.78      0.87      0.82       183
           1       0.78      0.64      0.70       129

   micro avg       0.78      0.78      0.78       312
   macro avg       0.78      0.76      0.76       312
weighted avg       0.78      0.78      0.77       312
```

In [ ]:
```

```

In [ ]:
```

```