----------Breast cancer Wisconsin (Diagnostic)----------

Predict whether the cancer is benign or malignant

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. n the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server: ftp ftp.cs.wisc.edu cd math-prog/cpo-dataset/machine-learn/WDBC/

Also can be found on UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Attribute Information:

1. ID number
2. Diagnosis (M = malignant, B = benign) 3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter) b) texture (standard deviation of gray-scale values) c) perimeter d) area e) smoothness (local variation in radius lengths) f) compactness (perimeter^2 / area - 1.0) g) concavity (severity of concave portions of the contour) h) concave points (number of concave portions of the contour) i) symmetry j) fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

All feature values are recoded with four significant digits.

Missing attribute values: none

Class distribution: 357 benign, 212 malignant

```
import jovian
```

```
jovian.commit(project="Breast cancer Wisconsin (Diagnostic)")
```

```
[jovian] Detected Colab notebook...
[jovian] Please enter your API key ( from https://jovian.ai/ ):
API KEY: ··········
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

```
'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'
```

Downloading the Dataset

```
!pip install jovian
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
```

```
wheels/public/simple/
Requirement already satisfied: jovian in /usr/local/lib/python3.7/dist-packages
(0.2.41)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from
jovian) (2.23.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (from
jovian) (3.13)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from
jovian) (7.1.2)
Requirement already satisfied: uuid in /usr/local/lib/python3.7/dist-packages (from
jovian) (1.30)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-
packages (from requests->jovian) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests->jovian) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-
packages (from requests->jovian) (2022.5.18.1)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
(from requests->jovian) (2.10)
```

```
!pip install opendatasets
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting opendatasets
  Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from
opendatasets) (7.1.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from
opendatasets) (4.64.0)
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (from
opendatasets) (1.5.12)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages
(from kaggle->opendatasets) (1.15.0)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from
kaggle->opendatasets) (1.24.3)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages
(from kaggle->opendatasets) (6.1.2)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from
kaggle->opendatasets) (2.23.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from
kaggle->opendatasets) (2022.5.18.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-
packages (from kaggle->opendatasets) (2.8.2)
```

```
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-packages (from python-slugify->kaggle->opendatasets) (1.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle->opendatasets) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle->opendatasets) (3.0.4)
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.22
```

```
import opendatasets as od
```

```
od.download('https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data')
```

```
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: swetsheersh
Your Kaggle Key: ··········
Downloading breast-cancer-wisconsin-data.zip to ./breast-cancer-wisconsin-data
100%|██████████| 48.6k/48.6k [00:00<00:00, 12.6MB/s]
```

```
import os
```

```
os.listdir()
```

```
['.config', 'breast-cancer-wisconsin-data', 'sample_data']
```

```
os.listdir('./breast-cancer-wisconsin-data')
```

```
['data.csv']
```

```
import pandas as pd
```

```
df=pd.read_csv('./breast-cancer-wisconsin-data/data.csv')
df
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.2
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.2

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.1
...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.1
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.1
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.1
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.2
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.0

569 rows × 33 columns

problem statement

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. n the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server: ftp ftp.cs.wisc.edu cd math-prog/cpo-dataset/machine-learn/WDBC/

Also can be found on UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Attribute Information:

1. ID number 2) Diagnosis (M = malignant, B = benign) 3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter) b) texture (standard deviation of gray-scale values) c) perimeter d) area e) smoothness (local variation in radius lengths) f) compactness (perimeter^2 / area - 1.0) g) concavity (severity of concave portions of the contour) h) concave points (number of concave portions of the contour) i) symmetry j) fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

All feature values are recoded with four significant digits.

Missing attribute values: none

Class distribution: 357 benign, 212 malignant

```
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
```

```
df.nunique()
```

```
id                         569
diagnosis                    2
radius_mean                456
texture_mean               479
```

```
perimeter_mean             522
area_mean                  539
smoothness_mean            474
compactness_mean           537
concavity_mean             537
concave points_mean        542
symmetry_mean              432
fractal_dimension_mean     499
radius_se                  540
texture_se                 519
perimeter_se               533
area_se                    528
smoothness_se              547
compactness_se             541
concavity_se               533
concave points_se          507
symmetry_se                498
fractal_dimension_se       545
radius_worst               457
texture_worst              511
perimeter_worst            514
area_worst                 544
smoothness_worst           411
compactness_worst          529
concavity_worst            539
concave points_worst       492
symmetry_worst             500
fractal_dimension_worst    535
Unnamed: 32                  0
dtype: int64
```

```
df.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mea
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.00000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.10434
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.05281
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.01938
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.06492
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.09263
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.13040
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.34540

8 rows × 32 columns

```
df.corr()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compa
id	1.000000	0.074626	0.099770	0.073159	0.096893	-0.012968	
radius_mean	0.074626	1.000000	0.323782	0.997855	0.987357	0.170581	
texture_mean	0.099770	0.323782	1.000000	0.329533	0.321086	-0.023389	
perimeter_mean	0.073159	0.997855	0.329533	1.000000	0.986507	0.207278	
area_mean	0.096893	0.987357	0.321086	0.986507	1.000000	0.177028	
smoothness_mean	-0.012968	0.170581	-0.023389	0.207278	0.177028	1.000000	
compactness_mean	0.000096	0.506124	0.236702	0.556936	0.498502	0.659123	
concavity_mean	0.050080	0.676764	0.302418	0.716136	0.685983	0.521984	
concave points_mean	0.044158	0.822529	0.293464	0.850977	0.823269	0.553695	
symmetry_mean	-0.022114	0.147741	0.071401	0.183027	0.151293	0.557775	
fractal_dimension_mean	-0.052511	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	
radius_se	0.143048	0.679090	0.275869	0.691765	0.732562	0.301467	
texture_se	-0.007526	-0.097317	0.386358	-0.086761	-0.066280	0.068406	
perimeter_se	0.137331	0.674172	0.281673	0.693135	0.726628	0.296092	
area_se	0.177742	0.735864	0.259845	0.744983	0.800086	0.246552	
smoothness_se	0.096781	-0.222600	0.006614	-0.202694	-0.166777	0.332375	
compactness_se	0.033961	0.206000	0.191975	0.250744	0.212583	0.318943	
concavity_se	0.055239	0.194204	0.143293	0.228082	0.207660	0.248396	
concave points_se	0.078768	0.376169	0.163851	0.407217	0.372320	0.380676	
symmetry_se	-0.017306	-0.104321	0.009127	-0.081629	-0.072497	0.200774	
fractal_dimension_se	0.025725	-0.042641	0.054458	-0.005523	-0.019887	0.283607	
radius_worst	0.082405	0.969539	0.352573	0.969476	0.962746	0.213120	
texture_worst	0.064720	0.297008	0.912045	0.303038	0.287489	0.036072	
perimeter_worst	0.079986	0.965137	0.358040	0.970387	0.959120	0.238853	
area_worst	0.107187	0.941082	0.343546	0.941550	0.959213	0.206718	
smoothness_worst	0.010338	0.119616	0.077503	0.150549	0.123523	0.805324	
compactness_worst	-0.002968	0.413463	0.277830	0.455774	0.390410	0.472468	
concavity_worst	0.023203	0.526911	0.301025	0.563879	0.512606	0.434926	
concave points_worst	0.035174	0.744214	0.295316	0.771241	0.722017	0.503053	
symmetry_worst	-0.044224	0.163953	0.105008	0.189115	0.143570	0.394309	
fractal_dimension_worst	-0.029866	0.007066	0.119205	0.051019	0.003738	0.499316	
Unnamed: 32	NaN	NaN	NaN	NaN	NaN	NaN	

32 rows × 32 columns

```
!pip install plotly matplotlib seaborn --quiet
```

```
import plotly.express as px
import matplotlib
```

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set_style('darkgrid')
matplotlib.rcParams['font.size']=14
matplotlib.rcParams['figure.figsize']=(10,6)
matplotlib.rcParams['figure.facecolor']='#00000000'
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

```
'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'
```

```
px.histogram(df,x='diagnosis',title='diagnosis count plot')
```

```
px.scatter(df,x='radius_mean',y='texture_mean',color='diagnosis')
```

```
px.scatter(df,x='smoothness_mean',y='texture_mean',color='diagnosis')
```

```
px.scatter(df,x='smoothness_mean',y='fractal_dimension_worst',color='diagnosis')
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

```
'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'
```

Preparing the Data for Training

```
df.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
dtype: int64
```

```
df.drop('Unnamed: 32',axis=1,inplace=True)
```

```
df
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.2
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.2
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.1
...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.1
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.1
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.1
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.2
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.0

569 rows × 32 columns

```
df.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
dtype: int64
```

```
input_col=list(df.columns)[2:]
target_col='diagnosis'
```

```
print(input_col)
```

```
['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']
```

```
target_col
```

```
'diagnosis'
```

```
input_df=df[input_col].copy()
target_df=df[target_col].copy()
```

```
input_df
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800
...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000

569 rows × 30 columns

```
target_df
```

```
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
```

```
import numpy as np
```

```
input_df.select_dtypes(include=np.number).columns.tolist()
```

```
['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']
```

```
input_df.select_dtypes('object').columns.tolist()
```

```
[]
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'

Imputting missing numeric values

```
!pip install sklearn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
```

```
Requirement already satisfied: sklearn in /usr/local/lib/python3.7/dist-packages (0.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages
(from sklearn) (1.0.2)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
(from scikit-learn->sklearn) (1.1.0)
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-packages
(from scikit-learn->sklearn) (1.21.6)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn->sklearn) (3.1.0)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages
(from scikit-learn->sklearn) (1.4.1)
```

```
from sklearn.impute import SimpleImputer
```

```
imputer=SimpleImputer(strategy='mean')
```

```
imputer.fit(input_df)
```

```
SimpleImputer()
```

```
input_df[input_col]=imputer.transform(input_df)
```

```
input_df
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800
...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000

569 rows × 30 columns

```
input_df.isna().sum()
```

```
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
dtype: int64
```

```
input_df.describe()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_m
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000(
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000(
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426

8 rows × 30 columns

Encoding Categorical Data

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 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
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

```
from sklearn.preprocessing import OneHotEncoder
```

```
df=df.fillna('unknown')
```

```
encoder=OneHotEncoder(sparse=False,handle_unknown="ignore")
```

```
help(encoder)
```

```
Help on OneHotEncoder in module sklearn.preprocessing._encoders object:

class OneHotEncoder(_BaseEncoder)
 |  OneHotEncoder(*, categories='auto', drop=None, sparse=True, dtype=<class
'numpy.float64'>, handle_unknown='error')
 |
 |  Encode categorical features as a one-hot numeric array.
 |
 |  The input to this transformer should be an array-like of integers or
 |  strings, denoting the values taken on by categorical (discrete) features.
 |  The features are encoded using a one-hot (aka 'one-of-K' or 'dummy')
 |  encoding scheme. This creates a binary column for each category and
 |  returns a sparse matrix or dense array (depending on the ``sparse``
 |  parameter)
 |
 |  By default, the encoder derives the categories based on the unique values
 |  in each feature. Alternatively, you can also specify the `categories`
 |  manually.
 |
 |  This encoding is needed for feeding categorical data to many scikit-learn
 |  estimators, notably linear models and SVMs with the standard kernels.
 |
 |  Note: a one-hot encoding of y labels should use a LabelBinarizer
 |  instead.
 |
 |  Read more in the :ref:`User Guide <preprocessing_categorical_features>`.
 |
 |  Parameters
 |  ----------
 |  categories : 'auto' or a list of array-like, default='auto'
 |      Categories (unique values) per feature:
 |
 |      - 'auto' : Determine categories automatically from the training data.
 |      - list : ``categories[i]`` holds the categories expected in the ith
 |        column. The passed categories should not mix strings and numeric
 |        values within a single feature, and should be sorted in case of
```

```
|        numeric values.
|
|      The used categories can be found in the ``categories_`` attribute.
|
|      .. versionadded:: 0.20
|
|  drop : {'first', 'if_binary'} or an array-like of shape (n_features,),
default=None
|      Specifies a methodology to use to drop one of the categories per
|      feature. This is useful in situations where perfectly collinear
|      features cause problems, such as when feeding the resulting data
|      into a neural network or an unregularized regression.
|
|      However, dropping one category breaks the symmetry of the original
|      representation and can therefore induce a bias in downstream models,
|      for instance for penalized linear classification or regression models.
|
|      - None : retain all features (the default).
|      - 'first' : drop the first category in each feature. If only one
|        category is present, the feature will be dropped entirely.
|      - 'if_binary' : drop the first category in each feature with two
|        categories. Features with 1 or more than 2 categories are
|        left intact.
|      - array : ``drop[i]`` is the category in feature ``X[:, i]`` that
|        should be dropped.
|
|      .. versionadded:: 0.21
|         The parameter `drop` was added in 0.21.
|
|      .. versionchanged:: 0.23
|         The option `drop='if_binary'` was added in 0.23.
|
|  sparse : bool, default=True
|      Will return sparse matrix if set True else will return an array.
|
|  dtype : number type, default=float
|      Desired dtype of output.
|
|  handle_unknown : {'error', 'ignore'}, default='error'
|      Whether to raise an error or ignore if an unknown categorical feature
|      is present during transform (default is to raise). When this parameter
|      is set to 'ignore' and an unknown category is encountered during
|      transform, the resulting one-hot encoded columns for this feature
```

```
|      will be all zeros. In the inverse transform, an unknown category
|      will be denoted as None.
|
|  Attributes
|  ----------
|  categories_ : list of arrays
|      The categories of each feature determined during fitting
|      (in order of the features in X and corresponding with the output
|      of ``transform``). This includes the category specified in ``drop``
|      (if any).
|
|  drop_idx_ : array of shape (n_features,)
|      - ``drop_idx_[i]`` is the index in ``categories_[i]`` of the category
|        to be dropped for each feature.
|      - ``drop_idx_[i] = None`` if no category is to be dropped from the
|        feature with index ``i``, e.g. when `drop='if_binary'` and the
|        feature isn't binary.
|      - ``drop_idx_ = None`` if all the transformed features will be
|        retained.
|
|      .. versionchanged:: 0.23
|         Added the possibility to contain `None` values.
|
|  n_features_in_ : int
|      Number of features seen during :term:`fit`.
|
|      .. versionadded:: 1.0
|
|  feature_names_in_ : ndarray of shape (`n_features_in_`,)
|      Names of features seen during :term:`fit`. Defined only when `X`
|      has feature names that are all strings.
|
|      .. versionadded:: 1.0
|
|  See Also
|  --------
|  OrdinalEncoder : Performs an ordinal (integer)
|    encoding of the categorical features.
|  sklearn.feature_extraction.DictVectorizer : Performs a one-hot encoding of
|    dictionary items (also handles string-valued features).
|  sklearn.feature_extraction.FeatureHasher : Performs an approximate one-hot
|    encoding of dictionary items or strings.
|  LabelBinarizer : Binarizes labels in a one-vs-all
```

```
|    fashion.
|  MultiLabelBinarizer : Transforms between iterable of
|    iterables and a multilabel format, e.g. a (samples x classes) binary
|    matrix indicating the presence of a class label.
|
|  Examples
|  --------
|  Given a dataset with two features, we let the encoder find the unique
|  values per feature and transform the data to a binary one-hot encoding.
|
|  >>> from sklearn.preprocessing import OneHotEncoder
|
|  One can discard categories not seen during `fit`:
|
|  >>> enc = OneHotEncoder(handle_unknown='ignore')
|  >>> X = [['Male', 1], ['Female', 3], ['Female', 2]]
|  >>> enc.fit(X)
|  OneHotEncoder(handle_unknown='ignore')
|  >>> enc.categories_
|  [array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
|  >>> enc.transform([['Female', 1], ['Male', 4]]).toarray()
|  array([[1., 0., 1., 0., 0.],
|         [0., 1., 0., 0., 0.]])
|  >>> enc.inverse_transform([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0]])
|  array([['Male', 1],
|         [None, 2]], dtype=object)
|  >>> enc.get_feature_names_out(['gender', 'group'])
|  array(['gender_Female', 'gender_Male', 'group_1', 'group_2', 'group_3'], ...)
|
|  One can always drop the first column for each feature:
|
|  >>> drop_enc = OneHotEncoder(drop='first').fit(X)
|  >>> drop_enc.categories_
|  [array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
|  >>> drop_enc.transform([['Female', 1], ['Male', 2]]).toarray()
|  array([[0., 0., 0.],
|         [1., 1., 0.]])
|
|  Or drop a column for feature only having 2 categories:
|
|  >>> drop_binary_enc = OneHotEncoder(drop='if_binary').fit(X)
|  >>> drop_binary_enc.transform([['Female', 1], ['Male', 2]]).toarray()
|  array([[0., 1., 0., 0.],
```

```
 |        [1., 0., 1., 0.]])
 |
 |  Method resolution order:
 |      OneHotEncoder
 |      _BaseEncoder
 |      sklearn.base.TransformerMixin
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, categories='auto', drop=None, sparse=True, dtype=<class
'numpy.float64'>, handle_unknown='error')
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X, y=None)
 |      Fit OneHotEncoder to X.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          The data to determine the categories of each feature.
 |
 |      y : None
 |          Ignored. This parameter exists only for compatibility with
 |          :class:`~sklearn.pipeline.Pipeline`.
 |
 |      Returns
 |      -------
 |      self
 |          Fitted encoder.
 |
 |  fit_transform(self, X, y=None)
 |      Fit OneHotEncoder to X, then transform X.
 |
 |      Equivalent to fit(X).transform(X) but more convenient.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          The data to encode.
 |
 |      y : None
```

```
 |          Ignored. This parameter exists only for compatibility with
 |          :class:`~sklearn.pipeline.Pipeline`.
 |
 |      Returns
 |      -------
 |      X_out : {ndarray, sparse matrix} of shape                 (n_samples,
n_encoded_features)
 |          Transformed input. If `sparse=True`, a sparse matrix will be
 |          returned.
 |
 |  get_feature_names(self, input_features=None)
 |      DEPRECATED: get_feature_names is deprecated in 1.0 and will be removed in 1.2.
Please use get_feature_names_out instead.
 |
 |      Return feature names for output features.
 |
 |          Parameters
 |          ----------
 |          input_features : list of str of shape (n_features,)
 |              String names for input features if available. By default,
 |              "x0", "x1", ... "xn_features" is used.
 |
 |          Returns
 |          -------
 |          output_feature_names : ndarray of shape (n_output_features,)
 |              Array of feature names.
 |
 |  get_feature_names_out(self, input_features=None)
 |      Get output feature names for transformation.
 |
 |      Parameters
 |      ----------
 |      input_features : array-like of str or None, default=None
 |          Input features.
 |
 |          - If `input_features` is `None`, then `feature_names_in_` is
 |            used as feature names in. If `feature_names_in_` is not defined,
 |            then names are generated: `[x0, x1, ..., x(n_features_in_)]`.
 |          - If `input_features` is an array-like, then `input_features` must
 |            match `feature_names_in_` if `feature_names_in_` is defined.
 |
 |      Returns
 |      -------
```

```
 |      feature_names_out : ndarray of str objects
 |          Transformed feature names.
 |
 |  inverse_transform(self, X)
 |      Convert the data back to the original representation.
 |
 |      When unknown categories are encountered (all zeros in the
 |      one-hot encoding), ``None`` is used to represent this category. If the
 |      feature with the unknown category has a dropped category, the dropped
 |      category will be its inverse.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape                 (n_samples,
n_encoded_features)
 |          The transformed data.
 |
 |      Returns
 |      -------
 |      X_tr : ndarray of shape (n_samples, n_features)
 |          Inverse transformed array.
 |
 |  transform(self, X)
 |      Transform X using one-hot encoding.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          The data to encode.
 |
 |      Returns
 |      -------
 |      X_out : {ndarray, sparse matrix} of shape                 (n_samples,
n_encoded_features)
 |          Transformed input. If `sparse=True`, a sparse matrix will be
 |          returned.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.TransformerMixin:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
```

```
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|
|  __repr__(self, N_CHAR_MAX=700)
|      Return repr(self).
|
|  __setstate__(self, state)
|
|  get_params(self, deep=True)
|      Get parameters for this estimator.
|
|      Parameters
|      ----------
|      deep : bool, default=True
|          If True, will return the parameters for this estimator and
|          contained subobjects that are estimators.
|
|      Returns
|      -------
|      params : dict
|          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 :class:`~sklearn.pipeline.Pipeline`). The latter have
|      parameters of the form ``<component>__<parameter>`` so that it's
|      possible to update each component of a nested object.
|
|      Parameters
|      ----------
|      **params : dict
|          Estimator parameters.
|
|      Returns
|      -------
|      self : estimator instance
```

```
|          Estimator instance.
```

```
dia=pd.DataFrame(df['diagnosis'])
```

```
encoder.fit(dia)
```

```
OneHotEncoder(handle_unknown='ignore', sparse=False)
```

```
dia
```

	diagnosis
0	M
1	M
2	M
3	M
4	M
...	...
564	M
565	M
566	M
567	M
568	B

569 rows × 1 columns

```
df['diagnosis']
```

```
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
```

```
cat_col1=list(dia.columns)
```

```
encoded_cols=list(encoder.get_feature_names(cat_col1))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:

Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and
will be removed in 1.2. Please use get_feature_names_out instead.
```

```
encoded_cols
```

```
['diagnosis_B', 'diagnosis_M']
```

```
len(encoded_cols)
```

```
2
```

```
dia[encoded_cols]=encoder.transform(dia)
```

```
dia.columns
```

```
Index(['diagnosis', 'diagnosis_B', 'diagnosis_M'], dtype='object')
```

```
dia
```

	diagnosis	diagnosis_B	diagnosis_M
0	M	0.0	1.0
1	M	0.0	1.0
2	M	0.0	1.0
3	M	0.0	1.0
4	M	0.0	1.0
...	...	...	...
564	M	0.0	1.0
565	M	0.0	1.0
566	M	0.0	1.0
567	M	0.0	1.0
568	B	1.0	0.0

569 rows × 3 columns

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

```
'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'
```

Scaling Numerical Features

```
input_df
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800
...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000

569 rows × 30 columns

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler=MinMaxScaler()
```

```
scaler.fit(input_df)
```

```
MinMaxScaler()
```

```
input_df.describe().loc[['min','max']]
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
min	6.981	9.71	43.79	143.5	0.05263	0.01938	0.0000
max	28.110	39.28	188.50	2501.0	0.16340	0.34540	0.4268

2 rows × 30 columns

```
input_df[['radius_mean','texture_mean']].loc[1]
```

```
radius_mean     20.57
texture_mean    17.77
Name: 1, dtype: float64
```

```
input_df[list(input_df.columns)]=scaler.transform(input_df)
```

```
input_df
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	0.521037	0.022658	0.545989	0.363733	0.593753	0.792037	0.703140
1	0.643144	0.272574	0.615783	0.501591	0.289880	0.181768	0.203608
2	0.601496	0.390260	0.595743	0.449417	0.514309	0.431017	0.462512
3	0.210090	0.360839	0.233501	0.102906	0.811321	0.811361	0.565604
4	0.629893	0.156578	0.630986	0.489290	0.430351	0.347893	0.463918
...	...	...	...	...	...	...	...
564	0.690000	0.428813	0.678668	0.566490	0.526948	0.296055	0.571462
565	0.622320	0.626987	0.604036	0.474019	0.407782	0.257714	0.337395
566	0.455251	0.621238	0.445788	0.303118	0.288165	0.254340	0.216753
567	0.644564	0.663510	0.665538	0.475716	0.588336	0.790197	0.823336
568	0.036869	0.501522	0.028540	0.015907	0.000000	0.074351	0.000000

569 rows × 30 columns

```
input_df.describe().loc[['min','max']]
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0
max	1.0	1.0	1.0	1.0	1.0	1.0	1.0

2 rows × 30 columns

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'

Training ,validation and Test Sets

```
from sklearn.model_selection import train_test_split
```

```
train_df,val_df,train_target,val_target=train_test_split(input_df,target_df,test_size=0
```

```
help(train_test_split)
```

```
Help on function train_test_split in module sklearn.model_selection._split:
```

```
train_test_split(*arrays, test_size=None, train_size=None, random_state=None,
shuffle=True, stratify=None)
    Split arrays or matrices into random train and test subsets.

    Quick utility that wraps input validation and
    ``next(ShuffleSplit().split(X, y))`` and application to input data
    into a single call for splitting (and optionally subsampling) data in a
    oneliner.

    Read more in the :ref:`User Guide <cross_validation>`.

    Parameters
    ----------
    *arrays : sequence of indexables with same length / shape[0]
        Allowed inputs are lists, numpy arrays, scipy-sparse
        matrices or pandas dataframes.

    test_size : float or int, default=None
        If float, should be between 0.0 and 1.0 and represent the proportion
        of the dataset to include in the test split. If int, represents the
        absolute number of test samples. If None, the value is set to the
        complement of the train size. If ``train_size`` is also None, it will
        be set to 0.25.

    train_size : float or int, default=None
        If float, should be between 0.0 and 1.0 and represent the
        proportion of the dataset to include in the train split. If
        int, represents the absolute number of train samples. If None,
        the value is automatically set to the complement of the test size.

    random_state : int, RandomState instance or None, default=None
        Controls the shuffling applied to the data before applying the split.
        Pass an int for reproducible output across multiple function calls.
        See :term:`Glossary <random_state>`.

    shuffle : bool, default=True
        Whether or not to shuffle the data before splitting. If shuffle=False
        then stratify must be None.

    stratify : array-like, default=None
        If not None, data is split in a stratified fashion, using this as
        the class labels.
```

```
    Read more in the :ref:`User Guide <stratification>`.

Returns
-------
splitting : list, length=2 * len(arrays)
    List containing train-test split of inputs.

    .. versionadded:: 0.16
        If the input is sparse, the output will be a
        ``scipy.sparse.csr_matrix``. Else, output type is the same as the
        input type.

Examples
--------
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> list(y)
[0, 1, 2, 3, 4]

>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]

>>> train_test_split(y, shuffle=False)
```

```
[[0, 1, 2], [3, 4]]
```

```
train_df
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
68	0.096928	0.257694	0.103656	0.045387	0.487226	0.373965	0.733365
181	0.667755	0.570172	0.683505	0.495228	0.554934	0.809214	0.582709
63	0.103744	0.140345	0.106489	0.049799	0.221901	0.208975	0.140300
248	0.173648	0.524518	0.167369	0.086320	0.396678	0.162444	0.055740
60	0.150930	0.174839	0.143459	0.071432	0.548614	0.187811	0.025398
...	...	...	...	...	...	...	...
71	0.090255	0.166723	0.103656	0.042630	0.408053	0.410159	0.201640
106	0.220503	0.291512	0.216847	0.114104	0.555836	0.252500	0.165651
270	0.345923	0.240446	0.321401	0.207466	0.105263	0.022606	0.016987
435	0.331251	0.335137	0.327068	0.193425	0.481809	0.288080	0.263824
102	0.246060	0.365573	0.231014	0.133701	0.248262	0.064413	0.055834

455 rows × 30 columns

```
val_df
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
204	0.259785	0.300643	0.257757	0.143542	0.424483	0.265076	0.187559
70	0.565999	0.392289	0.551517	0.418452	0.338178	0.256181	0.253046
131	0.401297	0.330402	0.400180	0.256797	0.510698	0.315686	0.343486
431	0.256472	0.269530	0.260383	0.137561	0.476393	0.344212	0.181373
540	0.215770	0.159959	0.213254	0.110032	0.426198	0.284093	0.157849
...	...	...	...	...	...	...	...
486	0.362488	0.241461	0.348421	0.221633	0.304956	0.146003	0.121649
75	0.430167	0.336152	0.416765	0.285981	0.352532	0.198945	0.228889
249	0.214823	0.176530	0.207864	0.111474	0.439379	0.180050	0.101406
238	0.342610	0.613460	0.336950	0.203775	0.267220	0.259248	0.258435
265	0.650717	0.724045	0.635132	0.541039	0.379706	0.291148	0.320291

114 rows × 30 columns

```
train_target
```

```
68     B
181    M
63     B
248    B
```

```
60     B
      ..
71     B
106    B
270    B
435    M
102    B
Name: diagnosis, Length: 455, dtype: object
```

```
val_target
```

```
204    B
70     M
131    M
431    B
540    B
      ..
486    B
75     M
249    B
238    B
265    M
Name: diagnosis, Length: 114, dtype: object
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

```
'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'
```

Training Logistic Regression

```
from sklearn.linear_model import LogisticRegression
```

```
model= LogisticRegression(solver='liblinear')
```

```
help(LogisticRegression)
```

```
Help on class LogisticRegression in module sklearn.linear_model._logistic:

class LogisticRegression(sklearn.linear_model._base.LinearClassifierMixin,
sklearn.linear_model._base.SparseCoefMixin, sklearn.base.BaseEstimator)
 |  LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0,
fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None,
```

```
solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False,
n_jobs=None, l1_ratio=None)
 |
 |  Logistic Regression (aka logit, MaxEnt) classifier.
 |
 |  In the multiclass case, the training algorithm uses the one-vs-rest (OvR)
 |  scheme if the 'multi_class' option is set to 'ovr', and uses the
 |  cross-entropy loss if the 'multi_class' option is set to 'multinomial'.
 |  (Currently the 'multinomial' option is supported only by the 'lbfgs',
 |  'sag', 'saga' and 'newton-cg' solvers.)
 |
 |  This class implements regularized logistic regression using the
 |  'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Note
 |  that regularization is applied by default**. It can handle both dense
 |  and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit
 |  floats for optimal performance; any other input format will be converted
 |  (and copied).
 |
 |  The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization
 |  with primal formulation, or no regularization. The 'liblinear' solver
 |  supports both L1 and L2 regularization, with a dual formulation only for
 |  the L2 penalty. The Elastic-Net regularization is only supported by the
 |  'saga' solver.
 |
 |  Read more in the :ref:`User Guide <logistic_regression>`.
 |
 |  Parameters
 |  ----------
 |  penalty : {'l1', 'l2', 'elasticnet', 'none'}, default='l2'
 |      Specify the norm of the penalty:
 |
 |      - ``'none'``: no penalty is added;
 |      - ``'l2'``: add a L2 penalty term and it is the default choice;
 |      - ``'l1'``: add a L1 penalty term;
 |      - ``'elasticnet'``: both L1 and L2 penalty terms are added.
 |
 |      .. warning::
 |         Some penalties may not work with some solvers. See the parameter
 |         `solver` below, to know the compatibility between the penalty and
 |         solver.
 |
 |      .. versionadded:: 0.19
 |         l1 penalty with SAGA solver (allowing 'multinomial' + L1)
```

```
|
|  dual : bool, default=False
|      Dual or primal formulation. Dual formulation is only implemented for
|      l2 penalty with liblinear solver. Prefer dual=False when
|      n_samples > n_features.
|
|  tol : float, default=1e-4
|      Tolerance for stopping criteria.
|
|  C : float, default=1.0
|      Inverse of regularization strength; must be a positive float.
|      Like in support vector machines, smaller values specify stronger
|      regularization.
|
|  fit_intercept : bool, default=True
|      Specifies if a constant (a.k.a. bias or intercept) should be
|      added to the decision function.
|
|  intercept_scaling : float, default=1
|      Useful only when the solver 'liblinear' is used
|      and self.fit_intercept is set to True. In this case, x becomes
|      [x, self.intercept_scaling],
|      i.e. a "synthetic" feature with constant value equal to
|      intercept_scaling is appended to the instance vector.
|      The intercept becomes ``intercept_scaling * synthetic_feature_weight``.
|
|      Note! the synthetic feature weight is subject to l1/l2 regularization
|      as all other features.
|      To lessen the effect of regularization on synthetic feature weight
|      (and therefore on the intercept) intercept_scaling has to be increased.
|
|  class_weight : dict or 'balanced', default=None
|      Weights associated with classes in the form ``{class_label: weight}``.
|      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))``.
|
|      Note that these weights will be multiplied with sample_weight (passed
|      through the fit method) if sample_weight is specified.
|
|      .. versionadded:: 0.17
```

```
|         *class_weight='balanced'*
|
|  random_state : int, RandomState instance, default=None
|      Used when ``solver`` == 'sag', 'saga' or 'liblinear' to shuffle the
|      data. See :term:`Glossary <random_state>` for details.
|
|  solver : {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'},
default='lbfgs'
|
|      Algorithm to use in the optimization problem. Default is 'lbfgs'.
|      To choose a solver, you might want to consider the following aspects:
|
|          - For small datasets, 'liblinear' is a good choice, whereas 'sag'
|            and 'saga' are faster for large ones;
|          - For multiclass problems, only 'newton-cg', 'sag', 'saga' and
|            'lbfgs' handle multinomial loss;
|          - 'liblinear' is limited to one-versus-rest schemes.
|
|      .. warning::
|         The choice of the algorithm depends on the penalty chosen:
|         Supported penalties by solver:
|
|         - 'newton-cg'   -   ['l2', 'none']
|         - 'lbfgs'       -   ['l2', 'none']
|         - 'liblinear'   -   ['l1', 'l2']
|         - 'sag'         -   ['l2', 'none']
|         - 'saga'        -   ['elasticnet', 'l1', 'l2', 'none']
|
|      .. note::
|         'sag' and 'saga' fast convergence is only guaranteed on
|         features with approximately the same scale. You can
|         preprocess the data with a scaler from :mod:`sklearn.preprocessing`.
|
|      .. seealso::
|         Refer to the User Guide for more information regarding
|         :class:`LogisticRegression` and more specifically the
|         `Table <https://scikit-learn.org/dev/modules/linear_model.html#logistic-
regression>`_
|         summarazing solver/penalty supports.
|         <!--
|         # noqa: E501
|         -->
|
```

```
|      .. versionadded:: 0.17
|         Stochastic Average Gradient descent solver.
|      .. versionadded:: 0.19
|         SAGA solver.
|      .. versionchanged:: 0.22
|          The default solver changed from 'liblinear' to 'lbfgs' in 0.22.
|
|  max_iter : int, default=100
|      Maximum number of iterations taken for the solvers to converge.
|
|  multi_class : {'auto', 'ovr', 'multinomial'}, default='auto'
|      If the option chosen is 'ovr', then a binary problem is fit for each
|      label. For 'multinomial' the loss minimised is the multinomial loss fit
|      across the entire probability distribution, *even when the data is
|      binary*. 'multinomial' is unavailable when solver='liblinear'.
|      'auto' selects 'ovr' if the data is binary, or if solver='liblinear',
|      and otherwise selects 'multinomial'.
|
|      .. versionadded:: 0.18
|         Stochastic Average Gradient descent solver for 'multinomial' case.
|      .. versionchanged:: 0.22
|          Default changed from 'ovr' to 'auto' in 0.22.
|
|  verbose : int, default=0
|      For the liblinear and lbfgs solvers set verbose to any positive
|      number for verbosity.
|
|  warm_start : bool, default=False
|      When set to True, reuse the solution of the previous call to fit as
|      initialization, otherwise, just erase the previous solution.
|      Useless for liblinear solver. See :term:`the Glossary <warm_start>`.
|
|      .. versionadded:: 0.17
|         *warm_start* to support *lbfgs*, *newton-cg*, *sag*, *saga* solvers.
|
|  n_jobs : int, default=None
|      Number of CPU cores used when parallelizing over classes if
|      multi_class='ovr'". This parameter is ignored when the ``solver`` is
|      set to 'liblinear' regardless of whether 'multi_class' is specified or
|      not. ``None`` means 1 unless in a :obj:`joblib.parallel_backend`
|      context. ``-1`` means using all processors.
|      See :term:`Glossary <n_jobs>` for more details.
|
```

```
|  l1_ratio : float, default=None
|      The Elastic-Net mixing parameter, with ``0 <= l1_ratio <= 1``. Only
|      used if ``penalty='elasticnet'``. Setting ``l1_ratio=0`` is equivalent
|      to using ``penalty='l2'``, while setting ``l1_ratio=1`` is equivalent
|      to using ``penalty='l1'``. For ``0 < l1_ratio <1``, the penalty is a
|      combination of L1 and L2.
|
|  Attributes
|  ----------
|
|  classes_ : ndarray of shape (n_classes, )
|      A list of class labels known to the classifier.
|
|  coef_ : ndarray of shape (1, n_features) or (n_classes, n_features)
|      Coefficient of the features in the decision function.
|
|      `coef_` is of shape (1, n_features) when the given problem is binary.
|      In particular, when `multi_class='multinomial'`, `coef_` corresponds
|      to outcome 1 (True) and `-coef_` corresponds to outcome 0 (False).
|
|  intercept_ : ndarray of shape (1,) or (n_classes,)
|      Intercept (a.k.a. bias) added to the decision function.
|
|      If `fit_intercept` is set to False, the intercept is set to zero.
|      `intercept_` is of shape (1,) when the given problem is binary.
|      In particular, when `multi_class='multinomial'`, `intercept_`
|      corresponds to outcome 1 (True) and `-intercept_` corresponds to
|      outcome 0 (False).
|
|  n_features_in_ : int
|      Number of features seen during :term:`fit`.
|
|      .. versionadded:: 0.24
|
|  feature_names_in_ : ndarray of shape (`n_features_in_`,)
|      Names of features seen during :term:`fit`. Defined only when `X`
|      has feature names that are all strings.
|
|      .. versionadded:: 1.0
|
|  n_iter_ : ndarray of shape (n_classes,) or (1, )
|      Actual number of iterations for all classes. If binary or multinomial,
|      it returns only 1 element. For liblinear solver, only the maximum
```

```
|      number of iteration across all classes is given.
|
|      .. versionchanged:: 0.20
|
|          In SciPy <= 1.0.0 the number of lbfgs iterations may exceed
|          ``max_iter``. ``n_iter_`` will now report at most ``max_iter``.
|
|  See Also
|  --------
|  SGDClassifier : Incrementally trained logistic regression (when given
|      the parameter ``loss="log"``).
|  LogisticRegressionCV : Logistic regression with built-in cross validation.
|
|  Notes
|  -----
|  The underlying C implementation uses a random number generator to
|  select features when fitting the model. It is thus not uncommon,
|  to have slightly different results for the same input data. If
|  that happens, try with a smaller tol parameter.
|
|  Predict output may not match that of standalone liblinear in certain
|  cases. See :ref:`differences from liblinear <liblinear_differences>`
|  in the narrative documentation.
|
|  References
|  ----------
|
|  L-BFGS-B -- Software for Large-scale Bound-constrained Optimization
|      Ciyou Zhu, Richard Byrd, Jorge Nocedal and Jose Luis Morales.
|      http://users.iems.northwestern.edu/~nocedal/lbfgsb.html
|
|  LIBLINEAR -- A Library for Large Linear Classification
|      https://www.csie.ntu.edu.tw/~cjlin/liblinear/
|
|  SAG -- Mark Schmidt, Nicolas Le Roux, and Francis Bach
|      Minimizing Finite Sums with the Stochastic Average Gradient
|      https://hal.inria.fr/hal-00860051/document
|
|  SAGA -- Defazio, A., Bach F. & Lacoste-Julien S. (2014).
|      SAGA: A Fast Incremental Gradient Method With Support
|      for Non-Strongly Convex Composite Objectives
|      https://arxiv.org/abs/1407.0202
|
```

```
 |  Hsiang-Fu Yu, Fang-Lan Huang, Chih-Jen Lin (2011). Dual coordinate descent
 |      methods for logistic regression and maximum entropy models.
 |      Machine Learning 85(1-2):41-75.
 |      https://www.csie.ntu.edu.tw/~cjlin/papers/maxent_dual.pdf
 |
 |  Examples
 |  --------
 |  >>> from sklearn.datasets import load_iris
 |  >>> from sklearn.linear_model import LogisticRegression
 |  >>> X, y = load_iris(return_X_y=True)
 |  >>> clf = LogisticRegression(random_state=0).fit(X, y)
 |  >>> clf.predict(X[:2, :])
 |  array([0, 0])
 |  >>> clf.predict_proba(X[:2, :])
 |  array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
 |         [9.7...e-01, 2.8...e-02, ...e-08]])
 |  >>> clf.score(X, y)
 |  0.97...
 |
 |  Method resolution order:
 |      LogisticRegression
 |      sklearn.linear_model._base.LinearClassifierMixin
 |      sklearn.base.ClassifierMixin
 |      sklearn.linear_model._base.SparseCoefMixin
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True,
intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs',
max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None,
l1_ratio=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X, y, sample_weight=None)
 |      Fit the model according to the given training data.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          Training vector, where `n_samples` is the number of samples and
 |          `n_features` is the number of features.
```

```
|
|      y : array-like of shape (n_samples,)
|          Target vector relative to X.
|
|      sample_weight : array-like of shape (n_samples,) default=None
|          Array of weights that are assigned to individual samples.
|          If not provided, then each sample is given unit weight.
|
|          .. versionadded:: 0.17
|             *sample_weight* support to LogisticRegression.
|
|      Returns
|      -------
|      self
|          Fitted estimator.
|
|      Notes
|      -----
|      The SAGA solver supports both float64 and float32 bit arrays.
|
|  predict_log_proba(self, X)
|      Predict logarithm of probability estimates.
|
|      The returned estimates for all classes are ordered by the
|      label of classes.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
|          Vector to be scored, where `n_samples` is the number of samples and
|          `n_features` is the number of features.
|
|      Returns
|      -------
|      T : array-like of shape (n_samples, n_classes)
|          Returns the log-probability of the sample for each class in the
|          model, where classes are ordered as they are in ``self.classes_``.
|
|  predict_proba(self, X)
|      Probability estimates.
|
|      The returned estimates for all classes are ordered by the
|      label of classes.
```

```
|
|      For a multi_class problem, if multi_class is set to be "multinomial"
|      the softmax function is used to find the predicted probability of
|      each class.
|      Else use a one-vs-rest approach, i.e calculate the probability
|      of each class assuming it to be positive using the logistic function.
|      and normalize these values across all the classes.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
|          Vector to be scored, where `n_samples` is the number of samples and
|          `n_features` is the number of features.
|
|      Returns
|      -------
|      T : array-like of shape (n_samples, n_classes)
|          Returns the probability of the sample for each class in the model,
|          where classes are ordered as they are in ``self.classes_``.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.linear_model._base.LinearClassifierMixin:
|
|  decision_function(self, X)
|      Predict confidence scores for samples.
|
|      The confidence score for a sample is proportional to the signed
|      distance of that sample to the hyperplane.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The data matrix for which we want to get the confidence scores.
|
|      Returns
|      -------
|      scores : ndarray of shape (n_samples,) or (n_samples, n_classes)
|          Confidence scores per `(n_samples, n_classes)` combination. In the
|          binary case, confidence score for `self.classes_[1]` where >0 means
|          this class would be predicted.
|
|  predict(self, X)
|      Predict class labels for samples in X.
```

```
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The data matrix for which we want to get the predictions.
|
|      Returns
|      -------
|      y_pred : ndarray of shape (n_samples,)
|          Vector containing the class labels for each sample.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.ClassifierMixin:
|
|  score(self, X, y, sample_weight=None)
|      Return 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 of shape (n_samples, n_features)
|          Test samples.
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          True labels for `X`.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights.
|
|      Returns
|      -------
|      score : float
|          Mean accuracy of ``self.predict(X)`` wrt. `y`.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.ClassifierMixin:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
```

```
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.linear_model._base.SparseCoefMixin:
|
|  densify(self)
|      Convert coefficient matrix to dense array format.
|
|      Converts the ``coef_`` member (back) to a numpy.ndarray. This is the
|      default format of ``coef_`` and is required for fitting, so calling
|      this method is only required on models that have previously been
|      sparsified; otherwise, it is a no-op.
|
|      Returns
|      -------
|      self
|          Fitted estimator.
|
|  sparsify(self)
|      Convert coefficient matrix to sparse format.
|
|      Converts the ``coef_`` member to a scipy.sparse matrix, which for
|      L1-regularized models can be much more memory- and storage-efficient
|      than the usual numpy.ndarray representation.
|
|      The ``intercept_`` member is not converted.
|
|      Returns
|      -------
|      self
|          Fitted estimator.
|
|      Notes
|      -----
|      For non-sparse models, i.e. when there are not many zeros in ``coef_``,
|      this may actually *increase* memory usage, so use this method with
|      care. A rule of thumb is that the number of zero elements, which can
|      be computed with ``(coef_ == 0).sum()``, must be more than 50% for this
|      to provide significant benefits.
|
|      After calling this method, further fitting with the partial_fit
|      method (if any) will not work until you call densify.
```

```
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|
|  __repr__(self, N_CHAR_MAX=700)
|      Return repr(self).
|
|  __setstate__(self, state)
|
|  get_params(self, deep=True)
|      Get parameters for this estimator.
|
|      Parameters
|      ----------
|      deep : bool, default=True
|          If True, will return the parameters for this estimator and
|          contained subobjects that are estimators.
|
|      Returns
|      -------
|      params : dict
|          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 :class:`~sklearn.pipeline.Pipeline`). The latter have
|      parameters of the form ``<component>__<parameter>`` so that it's
|      possible to update each component of a nested object.
|
|      Parameters
|      ----------
|      **params : dict
|          Estimator parameters.
|
|      Returns
|      -------
|      self : estimator instance
|          Estimator instance.
```

```
model.fit(train_df,train_target)
```

```
LogisticRegression(solver='liblinear')
```

```
print(model.coef_.tolist())
```

```
[[0.7415044441531012, 0.9607979209796121, 0.8069565984472767, 1.0555619221319144,
-0.29662730975553686, 0.4450302270910444, 1.7367620813037687, 2.435011645196136,
-0.3042097236118526, -1.345353052119477, 1.2081497102566414, -0.37774857457177485,
0.9044849329083722, 0.855123830731265, -0.3399066010930727, -0.7630118819705981,
-0.4311950979941451, -0.20185749039030731, -0.5488094413022576, -0.7233044928441416,
1.5965063502174888, 1.5600925700098058, 1.458349048394268, 1.4391474759904601,
0.7116737291753453, 0.8221201394797621, 1.3101285998932533, 2.2603319573356355,
1.0137781105284043, 0.18291275033196286]]
```

```
train_df.columns.tolist()
```

```
['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']
```

```
print(model.intercept_)
```

```
[-5.46430889]
```

```
train_preds=model.predict(train_df)
```

```
train_preds
```

```
array(['B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'M', 'M',
       'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M',
       'B', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'M', 'B', 'M', 'B', 'B',
       'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'M', 'M', 'M', 'M', 'B', 'M', 'B', 'M', 'B',
       'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'M',
       'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'B',
       'M', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B',
       'M', 'B', 'M', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'M',
       'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B',
       'B', 'M', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'M', 'B',
       'M', 'B', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'M',
       'M', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M',
       'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'M',
       'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M',
       'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B',
       'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'M',
       'M', 'M', 'B', 'M', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B',
       'M', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'M', 'M',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M', 'M',
       'B', 'B', 'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'M', 'B',
       'M', 'M', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M', 'M', 'M',
       'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'M', 'M', 'M',
       'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M',
       'M', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'M', 'M', 'M', 'B', 'B',
       'B', 'M', 'B', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M',
       'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'M',
       'M', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'B'],
      dtype=object)
```

```
train_target
```

```
68     B
```

```
181    M
63     B
248    B
60     B
      ..
71     B
106    B
270    B
435    M
102    B
Name: diagnosis, Length: 455, dtype: object
```

```
train_probs=model.predict_proba(train_df)
train_probs
```

```
array([[7.17271308e-01, 2.82728692e-01],
       [1.19461976e-03, 9.98805380e-01],
       [9.86750971e-01, 1.32490292e-02],
       [8.99067440e-01, 1.00932560e-01],
       [9.88017585e-01, 1.19824150e-02],
       [8.54898969e-02, 9.14510103e-01],
       [8.65179729e-01, 1.34820271e-01],
       [9.33033986e-01, 6.69660142e-02],
       [8.99660625e-01, 1.00339375e-01],
       [5.41479287e-02, 9.45852071e-01],
       [7.74624156e-01, 2.25375844e-01],
       [9.22621045e-03, 9.90773790e-01],
       [1.10367451e-01, 8.89632549e-01],
       [9.60446104e-01, 3.95538963e-02],
       [9.03738141e-01, 9.62618588e-02],
       [1.64057231e-02, 9.83594277e-01],
       [4.53061262e-03, 9.95469387e-01],
       [1.36172849e-02, 9.86382715e-01],
       [8.33516331e-01, 1.66483669e-01],
       [9.47091698e-01, 5.29083018e-02],
       [7.43701538e-01, 2.56298462e-01],
       [1.16615956e-01, 8.83384044e-01],
       [6.08898399e-01, 3.91101601e-01],
       [9.44155224e-01, 5.58447762e-02],
       [6.20737651e-01, 3.79262349e-01],
       [1.66053172e-01, 8.33946828e-01],
       [9.32136544e-01, 6.78634560e-02],
       [5.68345467e-03, 9.94316545e-01],
       [9.75267806e-01, 2.47321944e-02],
       [9.60516496e-01, 3.94835038e-02],
       [2.08273210e-01, 7.91726790e-01],
       [9.11089217e-01, 8.89107830e-02],
       [2.40578549e-02, 9.75942145e-01],
       [2.54742656e-02, 9.74525734e-01],
       [4.95491731e-01, 5.04508269e-01],
```

```
[9.51882241e-01, 4.81177589e-02],
[3.98500706e-01, 6.01499294e-01],
[9.64008916e-01, 3.59910840e-02],
[9.88429613e-01, 1.15703871e-02],
[8.40270490e-01, 1.59729510e-01],
[9.39992993e-01, 6.00070073e-02],
[2.15167446e-04, 9.99784833e-01],
[6.18842031e-02, 9.38115797e-01],
[9.82327131e-01, 1.76728688e-02],
[8.43975991e-01, 1.56024009e-01],
[9.77880362e-01, 2.21196376e-02],
[9.94759597e-01, 5.24040251e-03],
[9.43096654e-01, 5.69033457e-02],
[9.81240686e-01, 1.87593139e-02],
[8.81393993e-01, 1.18606007e-01],
[5.86109885e-01, 4.13890115e-01],
[9.39294092e-01, 6.07059077e-02],
[9.35079249e-01, 6.49207507e-02],
[6.58281478e-01, 3.41718522e-01],
[9.68112801e-01, 3.18871990e-02],
[3.50760008e-01, 6.49239992e-01],
[6.23559073e-01, 3.76440927e-01],
[8.96249129e-01, 1.03750871e-01],
[8.67748874e-01, 1.32251126e-01],
[7.04824542e-01, 2.95175458e-01],
[8.99777097e-01, 1.00222903e-01],
[8.37419929e-01, 1.62580071e-01],
[4.94574771e-01, 5.05425229e-01],
[6.59419149e-01, 3.40580851e-01],
[9.32109750e-01, 6.78902504e-02],
[7.88956029e-01, 2.11043971e-01],
[7.69857026e-01, 2.30142974e-01],
[8.47690756e-01, 1.52309244e-01],
[9.67636541e-01, 3.23634594e-02],
[3.55133764e-01, 6.44866236e-01],
[1.03894643e-01, 8.96105357e-01],
[4.61549830e-01, 5.38450170e-01],
[4.92660071e-01, 5.07339929e-01],
[7.59441225e-01, 2.40558775e-01],
[1.81081233e-01, 8.18918767e-01],
[9.16199458e-01, 8.38005416e-02],
[1.91593051e-02, 9.80840695e-01],
[8.46544935e-01, 1.53455065e-01],
[9.06120410e-01, 9.38795904e-02],
[9.81941745e-01, 1.80582553e-02],
[9.51049233e-01, 4.89507674e-02],
[3.06791318e-01, 6.93208682e-01],
[6.78949787e-01, 3.21050213e-01],
[9.76008553e-01, 2.39914468e-02],
```

```
[2.38026209e-02, 9.76197379e-01],
[9.60201292e-01, 3.97987085e-02],
[8.85735573e-01, 1.14264427e-01],
[9.72872865e-01, 2.71271346e-02],
[7.59626373e-03, 9.92403736e-01],
[8.93951139e-01, 1.06048861e-01],
[1.18748674e-02, 9.88125133e-01],
[2.31737682e-02, 9.76826232e-01],
[9.55857534e-01, 4.41424661e-02],
[9.39175425e-01, 6.08245747e-02],
[7.96041900e-01, 2.03958100e-01],
[4.28024045e-01, 5.71975955e-01],
[9.62763034e-01, 3.72369663e-02],
[9.85559688e-01, 1.44403115e-02],
[9.59930285e-01, 4.00697148e-02],
[9.88900220e-01, 1.10997797e-02],
[3.35864709e-01, 6.64135291e-01],
[9.73943962e-01, 2.60560384e-02],
[9.32370939e-01, 6.76290610e-02],
[9.87889941e-01, 1.21100594e-02],
[9.54444030e-01, 4.55559704e-02],
[9.75517266e-01, 2.44827343e-02],
[2.00044434e-01, 7.99955566e-01],
[6.93652693e-01, 3.06347307e-01],
[1.29153047e-02, 9.87084695e-01],
[4.84030001e-02, 9.51597000e-01],
[9.13966969e-01, 8.60330309e-02],
[9.61806146e-01, 3.81938535e-02],
[4.29993274e-03, 9.95700067e-01],
[9.16049551e-01, 8.39504495e-02],
[9.88423211e-01, 1.15767886e-02],
[8.69306422e-01, 1.30693578e-01],
[8.93103185e-01, 1.06896815e-01],
[7.50876720e-01, 2.49123280e-01],
[9.49959541e-01, 5.00404589e-02],
[9.20586659e-01, 7.94133407e-02],
[4.01728335e-01, 5.98271665e-01],
[1.29551494e-01, 8.70448506e-01],
[1.20343070e-04, 9.99879657e-01],
[9.64730828e-01, 3.52691721e-02],
[8.29640754e-01, 1.70359246e-01],
[7.73739830e-03, 9.92262602e-01],
[8.89238363e-01, 1.10761637e-01],
[9.11404354e-01, 8.85956465e-02],
[9.93257051e-02, 9.00674295e-01],
[9.00244048e-01, 9.97559518e-02],
[3.76798410e-01, 6.23201590e-01],
[9.90053315e-01, 9.94668498e-03],
[1.84647064e-02, 9.81535294e-01],
```

```
[9.02060625e-01, 9.79393751e-02],
[6.08071175e-01, 3.91928825e-01],
[9.49369814e-01, 5.06301856e-02],
[6.17829672e-01, 3.82170328e-01],
[2.95951513e-01, 7.04048487e-01],
[8.84483400e-01, 1.15516600e-01],
[9.73932208e-01, 2.60677915e-02],
[7.70937890e-01, 2.29062110e-01],
[1.39576730e-01, 8.60423270e-01],
[9.05331357e-01, 9.46686429e-02],
[6.18619745e-02, 9.38138026e-01],
[7.40397526e-01, 2.59602474e-01],
[5.70420120e-02, 9.42957988e-01],
[9.48719016e-01, 5.12809842e-02],
[2.34555043e-01, 7.65444957e-01],
[9.56330840e-01, 4.36691595e-02],
[6.35937011e-01, 3.64062989e-01],
[8.96174162e-02, 9.10382584e-01],
[7.75815965e-01, 2.24184035e-01],
[9.94788643e-01, 5.21135703e-03],
[9.08946326e-01, 9.10536736e-02],
[9.26732332e-01, 7.32676684e-02],
[2.37729669e-01, 7.62270331e-01],
[6.86534560e-01, 3.13465440e-01],
[9.32150484e-01, 6.78495162e-02],
[8.70190884e-01, 1.29809116e-01],
[2.15317552e-02, 9.78468245e-01],
[5.50496623e-01, 4.49503377e-01],
[9.09750120e-01, 9.02498796e-02],
[3.70661224e-01, 6.29338776e-01],
[8.82323023e-01, 1.17676977e-01],
[9.64385497e-01, 3.56145033e-02],
[1.70102822e-02, 9.82989718e-01],
[9.07325076e-01, 9.26749238e-02],
[9.77489434e-01, 2.25105660e-02],
[9.12892318e-01, 8.71076820e-02],
[9.14284035e-01, 8.57159654e-02],
[9.01009640e-01, 9.89903603e-02],
[6.33574645e-01, 3.66425355e-01],
[9.81071355e-01, 1.89286450e-02],
[2.62110424e-01, 7.37889576e-01],
[9.15324655e-01, 8.46753445e-02],
[8.51656494e-01, 1.48343506e-01],
[9.01044807e-01, 9.89551927e-02],
[4.86431240e-01, 5.13568760e-01],
[9.79149627e-01, 2.08503734e-02],
[4.15387288e-02, 9.58461271e-01],
[6.08977778e-01, 3.91022222e-01],
[8.80487922e-01, 1.19512078e-01],
```

```
[9.59284153e-01, 4.07158470e-02],
[3.10239740e-01, 6.89760260e-01],
[9.55333889e-01, 4.46661105e-02],
[1.42482741e-01, 8.57517259e-01],
[1.27973856e-02, 9.87202614e-01],
[9.88467663e-01, 1.15323373e-02],
[8.50676695e-01, 1.49323305e-01],
[1.48229468e-01, 8.51770532e-01],
[9.63703518e-01, 3.62964815e-02],
[1.95302505e-01, 8.04697495e-01],
[2.19636983e-01, 7.80363017e-01],
[9.27567200e-01, 7.24328004e-02],
[9.08511296e-01, 9.14887037e-02],
[9.34989228e-01, 6.50107723e-02],
[7.73937653e-01, 2.26062347e-01],
[8.28076755e-01, 1.71923245e-01],
[9.09482059e-01, 9.05179415e-02],
[9.92654872e-01, 7.34512757e-03],
[4.12679209e-02, 9.58732079e-01],
[9.26911883e-01, 7.30881166e-02],
[4.89987266e-01, 5.10012734e-01],
[8.08205440e-01, 1.91794560e-01],
[8.17344990e-01, 1.82655010e-01],
[8.28739661e-01, 1.71260339e-01],
[1.16446518e-04, 9.99883553e-01],
[7.74091771e-01, 2.25908229e-01],
[1.14320100e-03, 9.98856799e-01],
[9.61458357e-01, 3.85416429e-02],
[8.30637510e-01, 1.69362490e-01],
[1.88555548e-02, 9.81144445e-01],
[1.34189097e-02, 9.86581090e-01],
[9.84539963e-01, 1.54600368e-02],
[9.36627786e-01, 6.33722136e-02],
[4.50749619e-02, 9.54925038e-01],
[8.97871285e-01, 1.02128715e-01],
[2.33761968e-02, 9.76623803e-01],
[1.30315861e-03, 9.98696841e-01],
[9.66447794e-01, 3.35522060e-02],
[9.43250707e-04, 9.99056749e-01],
[2.26408906e-03, 9.97735911e-01],
[9.11305615e-01, 8.86943850e-02],
[6.74289749e-01, 3.25710251e-01],
[5.36218158e-02, 9.46378184e-01],
[5.68087108e-02, 9.43191289e-01],
[6.52982522e-03, 9.93470175e-01],
[8.03466548e-01, 1.96533452e-01],
[9.70860614e-01, 2.91393863e-02],
[9.16313462e-01, 8.36865380e-02],
[8.76815842e-01, 1.23184158e-01],
```

```
[1.85449695e-01, 8.14550305e-01],
[9.26538378e-01, 7.34616225e-02],
[1.34803848e-01, 8.65196152e-01],
[7.56862078e-03, 9.92431379e-01],
[1.62650102e-02, 9.83734990e-01],
[5.56456183e-01, 4.43543817e-01],
[9.78268910e-01, 2.17310901e-02],
[8.97668135e-01, 1.02331865e-01],
[5.93882774e-01, 4.06117226e-01],
[7.76041651e-01, 2.23958349e-01],
[8.74640483e-01, 1.25359517e-01],
[8.78323950e-01, 1.21676050e-01],
[9.53664623e-01, 4.63353772e-02],
[8.58278419e-01, 1.41721581e-01],
[6.69030596e-02, 9.33096940e-01],
[2.12053165e-02, 9.78794684e-01],
[4.60330421e-01, 5.39669579e-01],
[9.74101057e-01, 2.58989432e-02],
[7.09647381e-02, 9.29035262e-01],
[9.71516886e-01, 2.84831138e-02],
[9.30579168e-01, 6.94208323e-02],
[9.96278906e-01, 3.72109405e-03],
[9.51672826e-01, 4.83271741e-02],
[9.84444091e-01, 1.55559090e-02],
[2.21262677e-02, 9.77873732e-01],
[9.59548253e-01, 4.04517466e-02],
[8.59107418e-01, 1.40892582e-01],
[5.85013629e-03, 9.94149864e-01],
[9.92913831e-02, 9.00708617e-01],
[9.32559965e-01, 6.74400352e-02],
[4.11978129e-01, 5.88021871e-01],
[8.71882502e-01, 1.28117498e-01],
[1.29406295e-02, 9.87059371e-01],
[9.07276766e-01, 9.27232338e-02],
[8.94192964e-01, 1.05807036e-01],
[9.30856066e-01, 6.91439344e-02],
[9.49720592e-01, 5.02794081e-02],
[8.71685067e-01, 1.28314933e-01],
[8.41012519e-01, 1.58987481e-01],
[1.51701335e-01, 8.48298665e-01],
[9.78128669e-01, 2.18713310e-02],
[7.21816968e-01, 2.78183032e-01],
[1.74253788e-01, 8.25746212e-01],
[7.87720828e-01, 2.12279172e-01],
[9.72618389e-01, 2.73816111e-02],
[9.65496541e-01, 3.45034587e-02],
[9.58159116e-01, 4.18408842e-02],
[8.85584782e-01, 1.14415218e-01],
[7.99607961e-01, 2.00392039e-01],
```

```
[1.29510961e-02, 9.87048904e-01],
[6.11476355e-01, 3.88523645e-01],
[7.91506553e-01, 2.08493447e-01],
[3.66990048e-01, 6.33009952e-01],
[8.23557884e-01, 1.76442116e-01],
[2.02924117e-01, 7.97075883e-01],
[1.52887269e-01, 8.47112731e-01],
[7.32892396e-02, 9.26710760e-01],
[7.88762171e-01, 2.11237829e-01],
[1.58188297e-01, 8.41811703e-01],
[7.13084248e-01, 2.86915752e-01],
[9.74250558e-01, 2.57494419e-02],
[8.55315659e-02, 9.14468434e-01],
[7.71034829e-02, 9.22896517e-01],
[2.88062043e-02, 9.71193796e-01],
[9.28435546e-01, 7.15644538e-02],
[9.48357401e-01, 5.16425991e-02],
[9.05706117e-01, 9.42938834e-02],
[9.52716212e-01, 4.72837881e-02],
[6.57784706e-01, 3.42215294e-01],
[8.28397818e-01, 1.71602182e-01],
[9.20900330e-01, 7.90996701e-02],
[3.37772732e-02, 9.66222727e-01],
[9.01583992e-01, 9.84160084e-02],
[8.60406343e-01, 1.39593657e-01],
[6.78917574e-01, 3.21082426e-01],
[9.18051105e-02, 9.08194889e-01],
[9.76120922e-01, 2.38790780e-02],
[9.49061007e-01, 5.09389930e-02],
[1.82247044e-01, 8.17752956e-01],
[5.84190636e-01, 4.15809364e-01],
[9.77310849e-01, 2.26891512e-02],
[9.59875939e-02, 9.04012406e-01],
[9.72136289e-01, 2.78637108e-02],
[5.45883207e-02, 9.45411679e-01],
[2.45835375e-02, 9.75416462e-01],
[9.83878069e-01, 1.61219310e-02],
[8.92675182e-01, 1.07324818e-01],
[2.57279866e-02, 9.74272013e-01],
[8.96404308e-01, 1.03595692e-01],
[7.49751923e-02, 9.25024808e-01],
[7.54409884e-03, 9.92455901e-01],
[9.27070548e-01, 7.29294515e-02],
[5.06776816e-02, 9.49322318e-01],
[4.41538857e-01, 5.58461143e-01],
[9.78157697e-01, 2.18423031e-02],
[8.00131462e-01, 1.99868538e-01],
[5.69071196e-01, 4.30928804e-01],
[9.63862365e-01, 3.61376346e-02],
```

```
[5.42813446e-01, 4.57186554e-01],
[9.32036523e-01, 6.79634766e-02],
[8.76986288e-01, 1.23013712e-01],
[4.92789757e-01, 5.07210243e-01],
[9.81526093e-01, 1.84739068e-02],
[6.72514275e-01, 3.27485725e-01],
[2.60132601e-01, 7.39867399e-01],
[2.83701270e-02, 9.71629873e-01],
[3.66986082e-02, 9.63301392e-01],
[8.67626675e-01, 1.32373325e-01],
[9.70825268e-01, 2.91747316e-02],
[8.01998880e-01, 1.98001120e-01],
[8.17624081e-02, 9.18237592e-01],
[1.20930153e-01, 8.79069847e-01],
[9.40954112e-01, 5.90458876e-02],
[4.23873160e-02, 9.57612684e-01],
[1.56519086e-02, 9.84348091e-01],
[8.49336827e-01, 1.50663173e-01],
[9.42294950e-01, 5.77050501e-02],
[9.42646267e-01, 5.73537334e-02],
[4.58458731e-03, 9.95415413e-01],
[9.91959384e-01, 8.04061568e-03],
[7.67378098e-02, 9.23262190e-01],
[2.43827752e-01, 7.56172248e-01],
[1.20120359e-01, 8.79879641e-01],
[7.79336580e-01, 2.20663420e-01],
[6.12789771e-01, 3.87210229e-01],
[8.49016766e-01, 1.50983234e-01],
[1.53348144e-01, 8.46651856e-01],
[9.24921735e-01, 7.50782653e-02],
[9.41850928e-01, 5.81490725e-02],
[2.36667427e-01, 7.63332573e-01],
[1.05076135e-02, 9.89492387e-01],
[1.39903617e-01, 8.60096383e-01],
[1.04462165e-01, 8.95537835e-01],
[7.18270503e-01, 2.81729497e-01],
[1.55787733e-01, 8.44212267e-01],
[9.01665336e-01, 9.83346641e-02],
[9.82804758e-01, 1.71952417e-02],
[8.49931697e-01, 1.50068303e-01],
[8.54309129e-01, 1.45690871e-01],
[8.41990331e-01, 1.58009669e-01],
[9.52425204e-01, 4.75747958e-02],
[9.72132939e-01, 2.78670608e-02],
[3.32049454e-02, 9.66795055e-01],
[6.95400098e-04, 9.99304600e-01],
[3.63275391e-01, 6.36724609e-01],
[4.97867725e-04, 9.99502132e-01],
[9.44103712e-01, 5.58962884e-02],
```

```
[8.19236411e-01, 1.80763589e-01],
[9.37521026e-01, 6.24789735e-02],
[7.02953109e-01, 2.97046891e-01],
[8.00440375e-06, 9.99991996e-01],
[7.09772341e-01, 2.90227659e-01],
[2.31611246e-01, 7.68388754e-01],
[9.69376984e-01, 3.06230164e-02],
[9.63173895e-01, 3.68261045e-02],
[9.38493925e-01, 6.15060753e-02],
[8.48462236e-01, 1.51537764e-01],
[8.35944633e-01, 1.64055367e-01],
[1.14514736e-01, 8.85485264e-01],
[1.06951003e-01, 8.93048997e-01],
[3.76888676e-01, 6.23111324e-01],
[9.72562060e-01, 2.74379403e-02],
[9.33509481e-01, 6.64905192e-02],
[2.22304194e-01, 7.77695806e-01],
[6.39813488e-01, 3.60186512e-01],
[9.58068304e-01, 4.19316963e-02],
[2.38030185e-02, 9.76196981e-01],
[2.48337094e-01, 7.51662906e-01],
[7.71023556e-02, 9.22897644e-01],
[7.65636961e-02, 9.23436304e-01],
[9.40832117e-01, 5.91678834e-02],
[9.14311459e-01, 8.56885406e-02],
[5.16377133e-01, 4.83622867e-01],
[9.60618852e-02, 9.03938115e-01],
[6.88572658e-01, 3.11427342e-01],
[9.01098915e-01, 9.89010852e-02],
[8.70571578e-01, 1.29428422e-01],
[1.24746596e-02, 9.87525340e-01],
[4.54474360e-01, 5.45525640e-01],
[1.03500505e-03, 9.98964995e-01],
[9.88389935e-01, 1.16100653e-02],
[9.61160281e-01, 3.88397189e-02],
[5.16656357e-01, 4.83343643e-01],
[9.69211963e-01, 3.07880371e-02],
[9.30550448e-01, 6.94495523e-02],
[9.79945307e-01, 2.00546930e-02],
[9.81819397e-01, 1.81806028e-02],
[2.37106231e-03, 9.97628938e-01],
[5.70705722e-01, 4.29294278e-01],
[7.93675654e-01, 2.06324346e-01],
[9.72191143e-01, 2.78088573e-02],
[8.49338598e-01, 1.50661402e-01],
[9.45127376e-01, 5.48726244e-02],
[8.42564687e-01, 1.57435313e-01],
[9.43414249e-01, 5.65857512e-02],
[9.62233258e-01, 3.77667415e-02],
```

```
       [9.44666225e-01, 5.53337754e-02],
       [8.21301499e-03, 9.91786985e-01],
       [9.85505971e-01, 1.44940292e-02],
       [9.77374679e-01, 2.26253213e-02],
       [9.45472835e-01, 5.45271653e-02],
       [8.76436784e-01, 1.23563216e-01],
       [9.83502863e-01, 1.64971367e-02],
       [9.36888234e-01, 6.31117658e-02],
       [4.88665680e-01, 5.11334320e-01],
       [5.14802232e-01, 4.85197768e-01],
       [6.54024409e-01, 3.45975591e-01],
       [1.59836993e-02, 9.84016301e-01],
       [9.64460130e-01, 3.55398701e-02],
       [9.35730553e-01, 6.42694466e-02],
       [4.73933255e-01, 5.26066745e-01],
       [4.57821305e-02, 9.54217870e-01],
       [2.41859195e-02, 9.75814080e-01],
       [8.66493320e-01, 1.33506680e-01],
       [1.99685105e-01, 8.00314895e-01],
       [2.50141421e-01, 7.49858579e-01],
       [6.38790702e-01, 3.61209298e-01],
       [1.01744366e-01, 8.98255634e-01],
       [8.96951385e-01, 1.03048615e-01],
       [9.95138748e-01, 4.86125224e-03],
       [8.02062308e-01, 1.97937692e-01],
       [9.47707054e-01, 5.22929458e-02],
       [2.76843592e-01, 7.23156408e-01],
       [8.91265810e-01, 1.08734190e-01]])
```

```
from sklearn.metrics import accuracy_score,confusion_matrix
```

```
accuracy_score(train_target,train_preds)
```

```
0.9692307692307692
```

```
accuracy_score(val_target,model.predict(val_df))
```

```
0.9736842105263158
```

```
confusion_matrix(train_preds,train_target,normalize='true')
```

```
array([[0.95945946, 0.04054054],
       [0.01257862, 0.98742138]])
```

```
model.feature_names_in_
```

```
array(['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'], dtype=object)
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

```
'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'
```

From Logastic Regression I got accuracy of 97.3% on Validation data sets.

Saving Processed Data to Disk

```
print('train_df :',train_df.shape)
print('val_df :',val_df.shape)
print('train_target :',train_target.shape)
print('val_target :',val_target.shape)
```

```
train_df : (455, 30)
val_df : (114, 30)
train_target : (455,)
val_target : (114,)
```

```
!pip install pyarrow --quiet
```

```
from pyarrow import parquet
```

```
%time
train_df.to_parquet('train_df.parquet')
val_df.to_parquet('val_df.parquet')
pd.DataFrame(train_target).to_parquet('train_target.parquet')
pd.DataFrame(val_target).to_parquet('val_target.parquet')
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 8.82 µs
```

```
os.listdir()
```

```
['.config',
 'breast-cancer-wisconsin-data',
```

```
'train_target.parquet',
'train_df.parquet',
'val_df.parquet',
'val_target.parquet',
'sample_data']
```

```
train_df=pd.read_parquet('train_df.parquet')
val_df=pd.read_parquet('val_df.parquet')
train_target=pd.read_parquet('train_target.parquet')
val_target=pd.read_parquet('val_target.parquet')
```

```
train_df
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
68	0.096928	0.257694	0.103656	0.045387	0.487226	0.373965	0.733365
181	0.667755	0.570172	0.683505	0.495228	0.554934	0.809214	0.582709
63	0.103744	0.140345	0.106489	0.049799	0.221901	0.208975	0.140300
248	0.173648	0.524518	0.167369	0.086320	0.396678	0.162444	0.055740
60	0.150930	0.174839	0.143459	0.071432	0.548614	0.187811	0.025398
...	...	...	...	...	...	...	...
71	0.090255	0.166723	0.103656	0.042630	0.408053	0.410159	0.201640
106	0.220503	0.291512	0.216847	0.114104	0.555836	0.252500	0.165651
270	0.345923	0.240446	0.321401	0.207466	0.105263	0.022606	0.016987
435	0.331251	0.335137	0.327068	0.193425	0.481809	0.288080	0.263824
102	0.246060	0.365573	0.231014	0.133701	0.248262	0.064413	0.055834

455 rows × 30 columns

```
val_df
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
204	0.259785	0.300643	0.257757	0.143542	0.424483	0.265076	0.187559
70	0.565999	0.392289	0.551517	0.418452	0.338178	0.256181	0.253046
131	0.401297	0.330402	0.400180	0.256797	0.510698	0.315686	0.343486
431	0.256472	0.269530	0.260383	0.137561	0.476393	0.344212	0.181373
540	0.215770	0.159959	0.213254	0.110032	0.426198	0.284093	0.157849
...	...	...	...	...	...	...	...
486	0.362488	0.241461	0.348421	0.221633	0.304956	0.146003	0.121649
75	0.430167	0.336152	0.416765	0.285981	0.352532	0.198945	0.228889
249	0.214823	0.176530	0.207864	0.111474	0.439379	0.180050	0.101406
238	0.342610	0.613460	0.336950	0.203775	0.267220	0.259248	0.258435
265	0.650717	0.724045	0.635132	0.541039	0.379706	0.291148	0.320291

114 rows × 30 columns

```
train_target
```

	diagnosis
68	B
181	M
63	B
248	B
60	B
...	...
71	B
106	B
270	B
435	M
102	B

455 rows × 1 columns

```
val_target
```

	diagnosis
204	B
70	M
131	M
431	B
540	B
...	...
486	B
75	M
249	B
238	B
265	M

114 rows × 1 columns

Support Vector Machine(SVM)

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
```

Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic

'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'

```
from sklearn.svm import SVC
```

```
classifier=SVC(kernel='poly',random_state=42)
```

```
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='scale', coef0=0.0, shrinking=True,
probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False,
max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None)
 |
 |  C-Support Vector Classification.
 |
 |  The implementation is based on libsvm. The fit time scales at least
 |  quadratically with the number of samples and may be impractical
 |  beyond tens of thousands of samples. For large datasets
 |  consider using :class:`~sklearn.svm.LinearSVC` or
 |  :class:`~sklearn.linear_model.SGDClassifier` instead, possibly after a
 |  :class:`~sklearn.kernel_approximation.Nystroem` transformer.
 |
 |  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, default=1.0
 |      Regularization parameter. The strength of the regularization is
 |      inversely proportional to C. Must be strictly positive. The penalty
 |      is a squared l2 penalty.
 |
 |  kernel : {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable,
```

```
default='rbf'
 |      Specifies the kernel type to be used in the algorithm.
 |      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, default=3
 |      Degree of the polynomial kernel function ('poly').
 |      Ignored by all other kernels.
 |
 |  gamma : {'scale', 'auto'} or float, default='scale'
 |      Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
 |
 |      - if ``gamma='scale'`` (default) is passed then it uses
 |        1 / (n_features * X.var()) as value of gamma,
 |      - if 'auto', uses 1 / n_features.
 |
 |      .. versionchanged:: 0.22
 |         The default value of ``gamma`` changed from 'auto' to 'scale'.
 |
 |  coef0 : float, default=0.0
 |      Independent term in kernel function.
 |      It is only significant in 'poly' and 'sigmoid'.
 |
 |  shrinking : bool, default=True
 |      Whether to use the shrinking heuristic.
 |      See the :ref:`User Guide <shrinking_svm>`.
 |
 |  probability : bool, default=False
 |      Whether to enable probability estimates. This must be enabled prior
 |      to calling `fit`, will slow down that method as it internally uses
 |      5-fold cross-validation, and `predict_proba` may be inconsistent with
 |      `predict`. Read more in the :ref:`User Guide <scores_probabilities>`.
 |
 |  tol : float, default=1e-3
 |      Tolerance for stopping criterion.
 |
 |  cache_size : float, default=200
 |      Specify the size of the kernel cache (in MB).
 |
 |  class_weight : dict or 'balanced', default=None
 |      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, 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. The parameter is
|      ignored for binary classification.
|
|      .. 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*.
|
|  break_ties : bool, default=False
|      If true, ``decision_function_shape='ovr'``, and number of classes > 2,
|      :term:`predict` will break ties according to the confidence values of
|      :term:`decision_function`; otherwise the first class among the tied
|      classes is returned. Please note that breaking ties comes at a
|      relatively high computational cost compared to a simple predict.
|
|      .. versionadded:: 0.22
|
|  random_state : int, RandomState instance or None, default=None
|      Controls the pseudo random number generation for shuffling the data for
|      probability estimates. Ignored when `probability` is False.
|      Pass an int for reproducible output across multiple function calls.
```

```
|      See :term:`Glossary <random_state>`.
|
|  Attributes
|  ----------
|  class_weight_ : ndarray of shape (n_classes,)
|      Multipliers of parameter C for each class.
|      Computed based on the ``class_weight`` parameter.
|
|  classes_ : ndarray of shape (n_classes,)
|      The classes labels.
|
|  coef_ : ndarray of shape (n_classes * (n_classes - 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_`.
|
|  dual_coef_ : ndarray of shape (n_classes -1, n_SV)
|      Dual coefficients of the support vector in the decision
|      function (see :ref:`sgd_mathematical_formulation`), multiplied by
|      their targets.
|      For multiclass, coefficient for all 1-vs-1 classifiers.
|      The layout of the coefficients in the multiclass case is somewhat
|      non-trivial. See the :ref:`multi-class section of the User Guide
|      <svm_multi_class>` for details.
|
|  fit_status_ : int
|      0 if correctly fitted, 1 otherwise (will raise warning)
|
|  intercept_ : ndarray of shape (n_classes * (n_classes - 1) / 2,)
|      Constants in decision function.
|
|  n_features_in_ : int
|      Number of features seen during :term:`fit`.
|
|      .. versionadded:: 0.24
|
|  feature_names_in_ : ndarray of shape (`n_features_in_`,)
|      Names of features seen during :term:`fit`. Defined only when `X`
|      has feature names that are all strings.
|
|      .. versionadded:: 1.0
```

```
|
|  support_ : ndarray of shape (n_SV)
|      Indices of support vectors.
|
|  support_vectors_ : ndarray of shape (n_SV, n_features)
|      Support vectors.
|
|  n_support_ : ndarray of shape (n_classes,), dtype=int32
|      Number of support vectors for each class.
|
|  probA_ : ndarray of shape (n_classes * (n_classes - 1) / 2)
|  probB_ : ndarray of shape (n_classes * (n_classes - 1) / 2)
|      If `probability=True`, it corresponds to the parameters learned in
|      Platt scaling to produce probability estimates from decision values.
|      If `probability=False`, it's 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]_. For
|      more information on the multiclass case and training procedure see
|      section 8 of [1]_.
|
|  shape_fit_ : tuple of int of shape (n_dimensions_of_X,)
|      Array dimensions of training vector ``X``.
|
|  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>`_
|
|  Examples
|  --------
```

```
 |  >>> import numpy as np
 |  >>> from sklearn.pipeline import make_pipeline
 |  >>> from sklearn.preprocessing import StandardScaler
 |  >>> X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
 |  >>> y = np.array([1, 1, 2, 2])
 |  >>> from sklearn.svm import SVC
 |  >>> clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
 |  >>> clf.fit(X, y)
 |  Pipeline(steps=[('standardscaler', StandardScaler()),
 |                  ('svc', SVC(gamma='auto'))])
 |
 |  >>> print(clf.predict([[-0.8, -1]]))
 |  [1]
 |
 |  Method resolution order:
 |      SVC
 |      sklearn.svm._base.BaseSVC
 |      sklearn.base.ClassifierMixin
 |      sklearn.svm._base.BaseLibSVM
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0,
shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None,
verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False,
random_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)
 |      Evaluate the decision function for the samples in X.
 |
 |      Parameters
 |      ----------
```

```
 |      X : array-like of shape (n_samples, n_features)
 |          The input samples.
 |
 |      Returns
 |      -------
 |      X : ndarray of 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 proportional
 |      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.
 |      If decision_function_shape='ovr', the decision function is a monotonic
 |      transformation of ovo decision function.
 |
 |  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} of shape (n_samples, n_features) or
(n_samples_test, n_samples_train)
 |          For kernel="precomputed", the expected shape of X is
 |          (n_samples_test, n_samples_train).
 |
 |      Returns
 |      -------
 |      y_pred : ndarray of shape (n_samples,)
 |          Class labels for samples in X.
 |
 |  predict_log_proba(self, X)
 |      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 of shape (n_samples, n_features) or
(n_samples_test, n_samples_train)
 |          For kernel="precomputed", the expected shape of X is
 |          (n_samples_test, n_samples_train).
 |
 |      Returns
 |      -------
 |      T : ndarray of 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 :term:`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(self, X)
 |      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 of shape (n_samples, n_features)
 |          For kernel="precomputed", the expected shape of X is
 |          (n_samples_test, n_samples_train).
 |
 |      Returns
 |      -------
 |      T : ndarray of 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 :term:`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.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.svm._base.BaseSVC:
|
|  probA_
|      Parameter learned in Platt scaling when `probability=True`.
|
|      Returns
|      -------
|      ndarray of shape  (n_classes * (n_classes - 1) / 2)
|
|  probB_
|      Parameter learned in Platt scaling when `probability=True`.
|
|      Returns
|      -------
|      ndarray of shape  (n_classes * (n_classes - 1) / 2)
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.ClassifierMixin:
|
|  score(self, X, y, sample_weight=None)
|      Return 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 of shape (n_samples, n_features)
|          Test samples.
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          True labels for `X`.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights.
```

```
 |
 |      Returns
 |      -------
 |      score : float
 |          Mean accuracy of ``self.predict(X)`` wrt. `y`.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.ClassifierMixin:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  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} of shape (n_samples, n_features)
or (n_samples, n_samples)
 |          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 of shape (n_samples,)
 |          Target values (class labels in classification, real numbers in
 |          regression).
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Per-sample weights. Rescale C per sample. Higher weights
 |          force the classifier to put more emphasis on these points.
 |
 |      Returns
 |      -------
 |      self : object
 |          Fitted estimator.
 |
```

```
|      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_
|      Weights assigned to the features when `kernel="linear"`.
|
|      Returns
|      -------
|      ndarray of shape (n_features, n_classes)
|
|  n_support_
|      Number of support vectors for each class.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|
|  __repr__(self, N_CHAR_MAX=700)
|      Return repr(self).
|
|  __setstate__(self, state)
|
|  get_params(self, deep=True)
|      Get parameters for this estimator.
|
|      Parameters
|      ----------
|      deep : bool, default=True
|          If True, will return the parameters for this estimator and
|          contained subobjects that are estimators.
|
|      Returns
|      -------
|      params : dict
```

```
 |          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 :class:`~sklearn.pipeline.Pipeline`). The latter have
 |      parameters of the form ``<component>__<parameter>`` so that it's
 |      possible to update each component of a nested object.
 |
 |      Parameters
 |      ----------
 |      **params : dict
 |          Estimator parameters.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
```

```
classifier.fit(train_df,train_target)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().


SVC(kernel='poly', random_state=42)
```

```
classifier.score(train_df,train_target)
```

```
0.989010989010989
```

```
classifier.score(val_df,val_target)
```

```
0.9824561403508771
```

```
val_pred=classifier.predict(val_df)
```

```
val_pred
```

```
array(['B', 'M', 'M', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'M', 'B',
       'M', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B',
```

```
       'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M',
       'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'M',
       'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'M', 'M', 'B', 'B',
       'B', 'M', 'M', 'B', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'M', 'M', 'B', 'M', 'M',
       'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'M'], dtype=object)
```

```
accuracy_score(val_target,val_pred)
```

```
0.9824561403508771
```

```
confusion_matrix(val_target,val_pred,normalize='true')
```

```
array([[1.        , 0.        ],
       [0.04651163, 0.95348837]])
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
```

```
tree=DecisionTreeClassifier(random_state=42)
```

```
tree.fit(train_df,train_target)
```

```
DecisionTreeClassifier(random_state=42)
```

```
train_pred=model.predict(train_df)
```

```
train_pred
```

```
array(['B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'M', 'M',
       'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M',
       'B', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'M', 'B', 'M', 'B', 'B',
       'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'M', 'M', 'M', 'M', 'B', 'M', 'B', 'M', 'B',
       'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'M',
```

```
       'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'B',
       'M', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B',
       'M', 'B', 'M', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'M',
       'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B',
       'B', 'M', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'M', 'B',
       'M', 'B', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'M',
       'M', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M',
       'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'M',
       'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M',
       'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B',
       'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'M',
       'M', 'M', 'B', 'M', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B',
       'M', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'M', 'M',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M', 'M',
       'B', 'B', 'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'M', 'B',
       'M', 'M', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M', 'M', 'M',
       'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'M', 'M', 'M',
       'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M',
       'M', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'M', 'M', 'M', 'B', 'B',
       'B', 'M', 'B', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M',
       'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'M',
       'M', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'B'],
      dtype=object)
```

```
pd.value_counts(train_preds)
```

```
B    296
M    159
dtype: int64
```

```
train_probs=tree.predict_proba(train_df)
train_probs
```

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

```
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
```

```
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
```

```
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
```

```
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
```

```
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
```

```
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
```

```
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
```

```
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
```

```
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
```

```
       [1., 0.],
       [0., 1.],
       [1., 0.]])
```

```
accuracy_score(train_target,train_pred)
```

```
0.9692307692307692
```

```
tree.score(val_df,val_target)
```

```
0.9473684210526315
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'

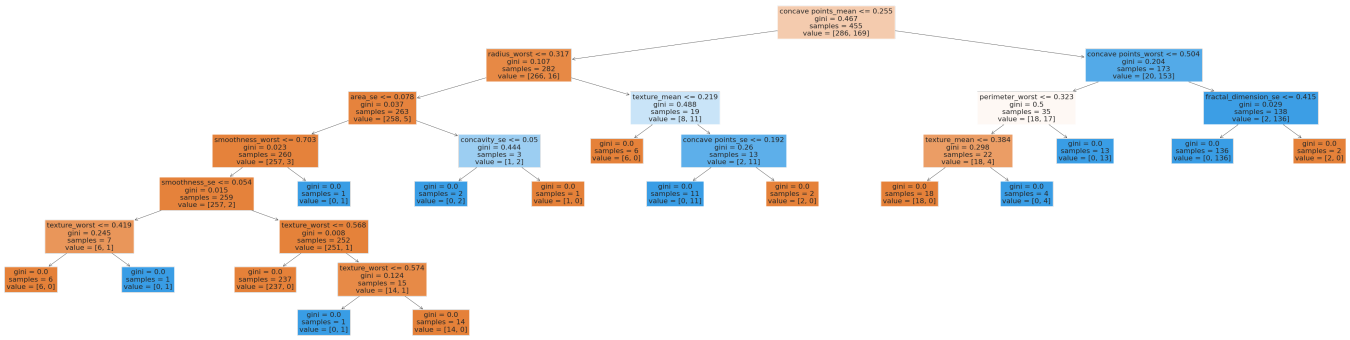
Visualization

```
from sklearn.tree import plot_tree,export_text
```

```
plt.figure(figsize=(80,20))
plot_tree(tree,feature_names=list(train_df.columns),max_depth=7,filled=True)
```

```
[Text(0.6145833333333334, 0.9375, 'concave points_mean <= 0.255\ngini = 0.467\nsamples = 455\nvalue = [286, 169]'),
 Text(0.3958333333333333, 0.8125, 'radius_worst <= 0.317\ngini = 0.107\nsamples = 282\nvalue = [266, 16]'),
 Text(0.2916666666666667, 0.6875, 'area_se <= 0.078\ngini = 0.037\nsamples = 263\nvalue = [258, 5]'),
 Text(0.20833333333333334, 0.5625, 'smoothness_worst <= 0.703\ngini = 0.023\nsamples = 260\nvalue = [257, 3]'),
 Text(0.16666666666666666, 0.4375, 'smoothness_se <= 0.054\ngini = 0.015\nsamples = 259\nvalue = [257, 2]'),
 Text(0.08333333333333333, 0.3125, 'texture_worst <= 0.419\ngini = 0.245\nsamples = 7\nvalue = [6, 1]'),
 Text(0.041666666666666664, 0.1875, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.125, 0.1875, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.25, 0.3125, 'texture_worst <= 0.568\ngini = 0.008\nsamples = 252\nvalue = [251, 1]'),
 Text(0.20833333333333334, 0.1875, 'gini = 0.0\nsamples = 237\nvalue = [237, 0]'),
 Text(0.2916666666666667, 0.1875, 'texture_worst <= 0.574\ngini = 0.124\nsamples = 15\nvalue = [14, 1]'),
 Text(0.25, 0.0625, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
```

```
 Text(0.3333333333333333, 0.0625, 'gini = 0.0\nsamples = 14\nvalue = [14, 0]'),
 Text(0.25, 0.4375, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.375, 0.5625, 'concavity_se <= 0.05\ngini = 0.444\nsamples = 3\nvalue = [1,
2]'),
 Text(0.3333333333333333, 0.4375, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.4166666666666667, 0.4375, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.5, 0.6875, 'texture_mean <= 0.219\ngini = 0.488\nsamples = 19\nvalue = [8,
11]'),
 Text(0.4583333333333333, 0.5625, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.5416666666666666, 0.5625, 'concave points_se <= 0.192\ngini = 0.26\nsamples =
13\nvalue = [2, 11]'),
 Text(0.5, 0.4375, 'gini = 0.0\nsamples = 11\nvalue = [0, 11]'),
 Text(0.5833333333333334, 0.4375, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.8333333333333334, 0.8125, 'concave points_worst <= 0.504\ngini = 0.204\nsamples
= 173\nvalue = [20, 153]'),
 Text(0.75, 0.6875, 'perimeter_worst <= 0.323\ngini = 0.5\nsamples = 35\nvalue = [18,
17]'),
 Text(0.7083333333333334, 0.5625, 'texture_mean <= 0.384\ngini = 0.298\nsamples =
22\nvalue = [18, 4]'),
 Text(0.6666666666666666, 0.4375, 'gini = 0.0\nsamples = 18\nvalue = [18, 0]'),
 Text(0.75, 0.4375, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.7916666666666666, 0.5625, 'gini = 0.0\nsamples = 13\nvalue = [0, 13]'),
 Text(0.9166666666666666, 0.6875, 'fractal_dimension_se <= 0.415\ngini = 0.029\nsamples
= 138\nvalue = [2, 136]'),
 Text(0.875, 0.5625, 'gini = 0.0\nsamples = 136\nvalue = [0, 136]'),
 Text(0.9583333333333334, 0.5625, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]')]
```




```
tree.tree_.max_depth
```

```
7
```

```
tree_text = export_text(tree, max_depth=7, feature_names=list(train_df.columns))
print(tree_text[:5000])
```

```
|--- concave points_mean <= 0.25
|   |--- radius_worst <= 0.32
|   |   |--- area_se <= 0.08
|   |   |   |--- smoothness_worst <= 0.70
|   |   |   |   |--- smoothness_se <= 0.05
|   |   |   |   |   |--- texture_worst <= 0.42
```

```
|   |   |   |   |   |   |--- class: B
|   |   |   |   |   |--- texture_worst >  0.42
|   |   |   |   |   |   |--- class: M
|   |   |   |   |--- smoothness_se >  0.05
|   |   |   |   |   |--- texture_worst <= 0.57
|   |   |   |   |   |   |--- class: B
|   |   |   |   |   |--- texture_worst >  0.57
|   |   |   |   |   |   |--- texture_worst <= 0.57
|   |   |   |   |   |   |   |--- class: M
|   |   |   |   |   |   |--- texture_worst >  0.57
|   |   |   |   |   |   |   |--- class: B
|   |   |   |--- smoothness_worst >  0.70
|   |   |   |   |--- class: M
|   |   |--- area_se >  0.08
|   |   |   |--- concavity_se <= 0.05
|   |   |   |   |--- class: M
|   |   |   |--- concavity_se >  0.05
|   |   |   |   |--- class: B
|   |--- radius_worst >  0.32
|   |   |--- texture_mean <= 0.22
|   |   |   |--- class: B
|   |   |--- texture_mean >  0.22
|   |   |   |--- concave points_se <= 0.19
|   |   |   |   |--- class: M
|   |   |   |--- concave points_se >  0.19
|   |   |   |   |--- class: B
|--- concave points_mean >  0.25
|   |--- concave points_worst <= 0.50
|   |   |--- perimeter_worst <= 0.32
|   |   |   |--- texture_mean <= 0.38
|   |   |   |   |--- class: B
|   |   |   |--- texture_mean >  0.38
|   |   |   |   |--- class: M
|   |   |--- perimeter_worst >  0.32
|   |   |   |--- class: M
|   |--- concave points_worst >  0.50
|   |   |--- fractal_dimension_se <= 0.42
|   |   |   |--- class: M
|   |   |--- fractal_dimension_se >  0.42
|   |   |   |--- class: B
```

```
list(train_df.columns)
```

```
['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']
```

Feature Importance

```
tree.feature_importances_
```

```
array([0.        , 0.05847766, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.69141955, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.01198257, 0.00123678,
       0.        , 0.00627578, 0.01593081, 0.        , 0.01855447,
       0.05229927, 0.01744516, 0.05149396, 0.        , 0.00923319,
       0.        , 0.        , 0.06565079, 0.        , 0.        ])
```

```
importance_df = pd.DataFrame({
    'feature': train_df.columns,
    'importance': tree.feature_importances_
}).sort_values('importance', ascending=False)
```

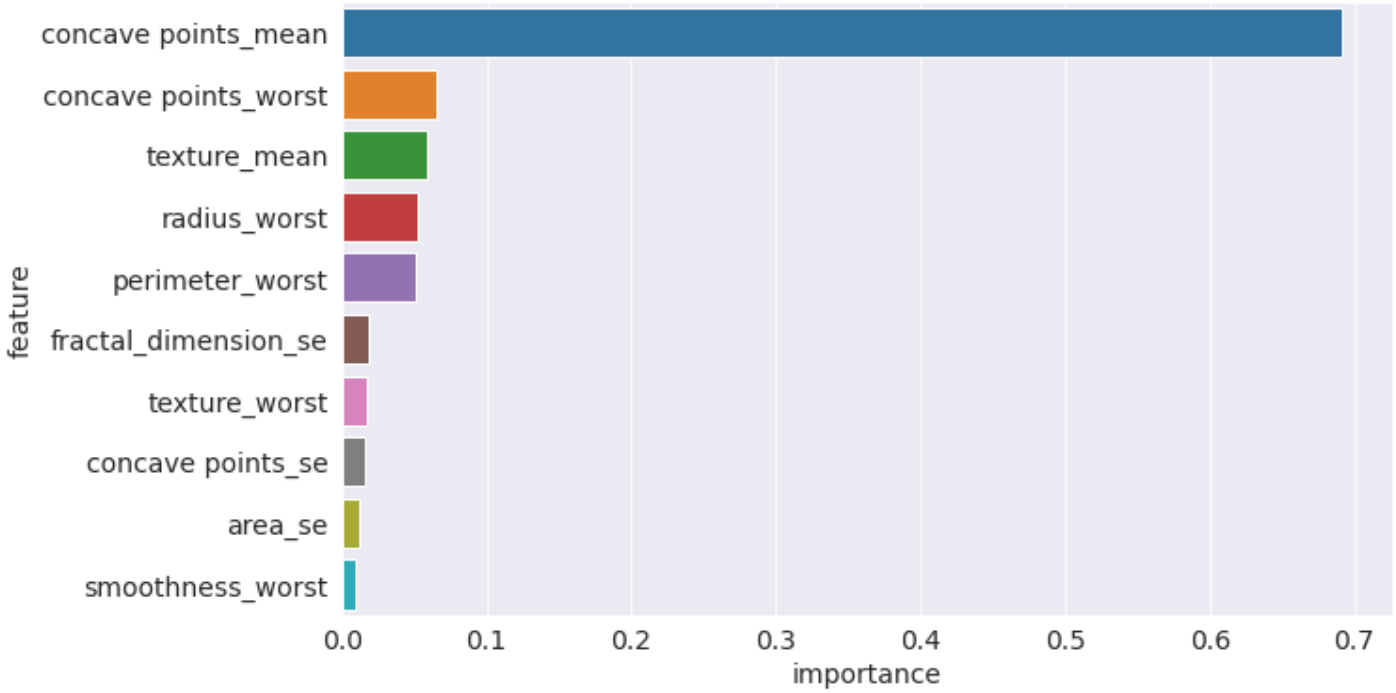
importance_df=pd.DataFrame({ 'feature' : train_df.columns, 'importance' : tree.feature_importances_
}).sort_values('importance',ascending=False)

```
importance_df
```

	feature	importance
7	concave points_mean	0.691420
27	concave points_worst	0.065651
1	texture_mean	0.058478
20	radius_worst	0.052299
22	perimeter_worst	0.051494
19	fractal_dimension_se	0.018554
21	texture_worst	0.017445
17	concave points_se	0.015931
13	area_se	0.011983
24	smoothness_worst	0.009233
16	concavity_se	0.006276
14	smoothness_se	0.001237
23	area_worst	0.000000
18	symmetry_se	0.000000
25	compactness_worst	0.000000
26	concavity_worst	0.000000
28	symmetry_worst	0.000000
0	radius_mean	0.000000
15	compactness_se	0.000000
12	perimeter_se	0.000000
11	texture_se	0.000000
10	radius_se	0.000000
9	fractal_dimension_mean	0.000000
8	symmetry_mean	0.000000
6	concavity_mean	0.000000
5	compactness_mean	0.000000
4	smoothness_mean	0.000000
3	area_mean	0.000000
2	perimeter_mean	0.000000
29	fractal_dimension_worst	0.000000

```
sns.barplot(data=importance_df.head(10),x='importance',y='feature')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f94af6b6b90>
```




```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

```
'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'
```

Hyperparameter Tuning and Overfitting

```
?DecisionTreeClassifier
```

```
tree.classes_
```

```
array(['B', 'M'], dtype=object)
```

```
model=DecisionTreeClassifier(max_depth=5,random_state=42)
```

```
model.fit(train_df,train_target)
```

```
DecisionTreeClassifier(max_depth=5, random_state=42)
```

```
model.score(train_df,train_target)
```

```
0.9956043956043956
```

```
model.score(val_df,val_target)
```

```
0.9473684210526315
```

```
def max_depth_error(md):
    model=DecisionTreeClassifier(max_depth=md,random_state=42)
    model.fit(train_df,train_target)
    train_error=1- model.score(train_df,train_target)
    val_error=1- model.score(val_df,val_target)
    return({'max_depth': md ,'training error': train_error,'val-error': val_error})
```

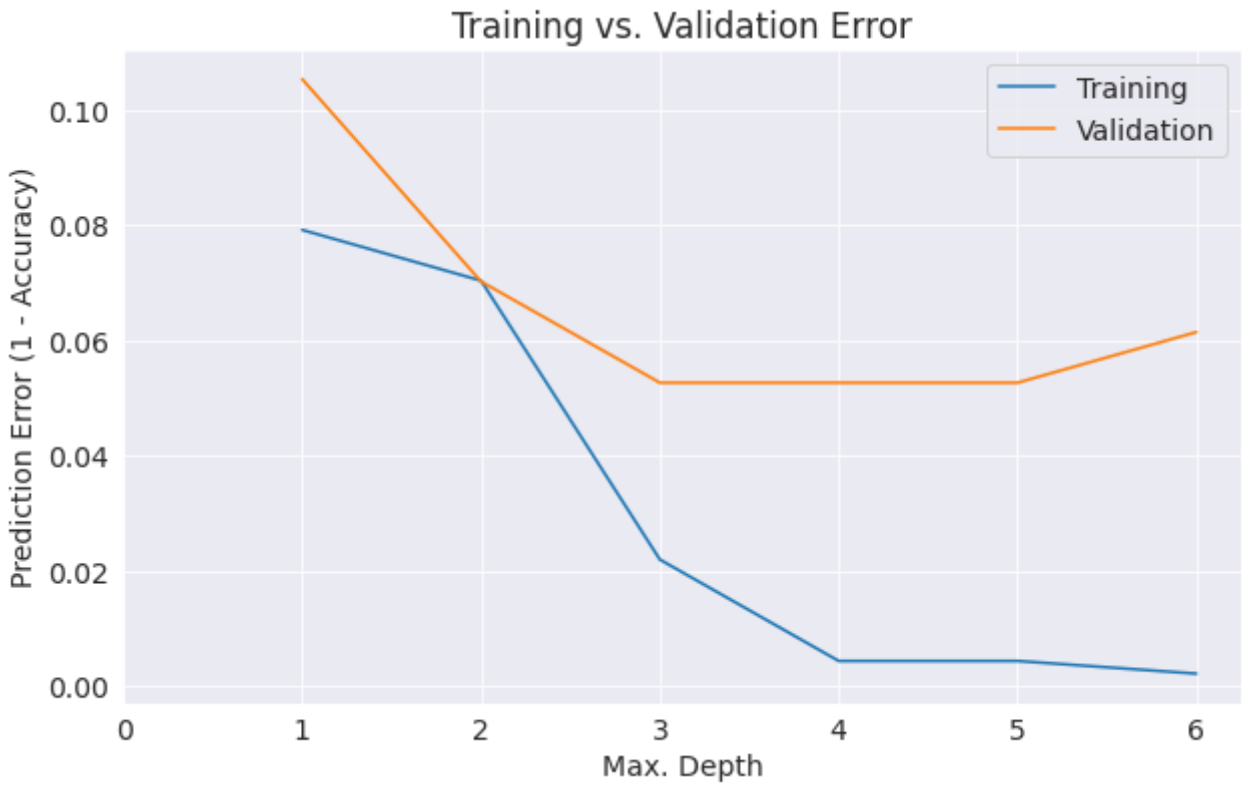
```
error_df=pd.DataFrame([max_depth_error(md) for md in range(1,7)])
```

```
error_df
```

	max_depth	training error	val-error
0	1	0.079121	0.105263
1	2	0.070330	0.070175
2	3	0.021978	0.052632
3	4	0.004396	0.052632
4	5	0.004396	0.052632
5	6	0.002198	0.061404

```
plt.figure()
plt.plot(error_df['max_depth'], error_df['training error'])
plt.plot(error_df['max_depth'], error_df['val-error'])
plt.title('Training vs. Validation Error')
plt.xticks(range(0,7, 1))
plt.xlabel('Max. Depth')
plt.ylabel('Prediction Error (1 - Accuracy)')
plt.legend(['Training', 'Validation'])
```

```
<matplotlib.legend.Legend at 0x7f94aba1cbd0>
```




```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

```
'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'
```

max_leaf_nodes

```
model=DecisionTreeClassifier(max_leaf_nodes=25,random_state=42)
```

```
model.fit(train_df,train_target)
```

```
DecisionTreeClassifier(max_leaf_nodes=25, random_state=42)
```

```
model.score(train_df,train_target)
```

```
1.0
```

```
model.score(val_df,val_target)
```

```
0.9473684210526315
```

From Decision Tree I got accuracy of 94.7% on validation data set

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'

```
help(model)
```

```
Help on DecisionTreeClassifier in module sklearn.tree._classes object:

class DecisionTreeClassifier(sklearn.base.ClassifierMixin, BaseDecisionTree)
 |  DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0,
class_weight=None, ccp_alpha=0.0)
 |
 |  A decision tree classifier.
 |
 |  Read more in the :ref:`User Guide <tree>`.
 |
 |  Parameters
 |  ----------
 |  criterion : {"gini", "entropy"}, default="gini"
 |      The function to measure the quality of a split. Supported criteria are
 |      "gini" for the Gini impurity and "entropy" for the information gain.
 |
 |  splitter : {"best", "random"}, default="best"
 |      The strategy used to choose the split at each node. Supported
 |      strategies are "best" to choose the best split and "random" to choose
 |      the best random split.
 |
 |  max_depth : int, default=None
 |      The maximum depth of the tree. If None, then nodes are expanded until
 |      all leaves are pure or until all leaves contain less than
 |      min_samples_split samples.
 |
 |  min_samples_split : int or float, default=2
 |      The minimum number of samples required to split an internal node:
 |
 |      - If int, then consider `min_samples_split` as the minimum number.
 |      - If float, then `min_samples_split` is a fraction and
 |        `ceil(min_samples_split * n_samples)` are the minimum
 |        number of samples for each split.
```

```
|
|      .. versionchanged:: 0.18
|         Added float values for fractions.
|
|  min_samples_leaf : int or float, default=1
|      The minimum number of samples required to be at a leaf node.
|      A split point at any depth will only be considered if it leaves at
|      least ``min_samples_leaf`` training samples in each of the left and
|      right branches.  This may have the effect of smoothing the model,
|      especially in regression.
|
|      - If int, then consider `min_samples_leaf` as the minimum number.
|      - If float, then `min_samples_leaf` is a fraction and
|        `ceil(min_samples_leaf * n_samples)` are the minimum
|        number of samples for each node.
|
|      .. versionchanged:: 0.18
|         Added float values for fractions.
|
|  min_weight_fraction_leaf : float, default=0.0
|      The minimum weighted fraction of the sum total of weights (of all
|      the input samples) required to be at a leaf node. Samples have
|      equal weight when sample_weight is not provided.
|
|  max_features : int, float or {"auto", "sqrt", "log2"}, default=None
|      The number of features to consider when looking for the best split:
|
|          - If int, then consider `max_features` features at each split.
|          - If float, then `max_features` is a fraction and
|            `int(max_features * n_features)` features are considered at each
|            split.
|          - If "auto", then `max_features=sqrt(n_features)`.
|          - If "sqrt", then `max_features=sqrt(n_features)`.
|          - If "log2", then `max_features=log2(n_features)`.
|          - If None, then `max_features=n_features`.
|
|      Note: the search for a split does not stop until at least one
|      valid partition of the node samples is found, even if it requires to
|      effectively inspect more than ``max_features`` features.
|
|  random_state : int, RandomState instance or None, default=None
|      Controls the randomness of the estimator. The features are always
|      randomly permuted at each split, even if ``splitter`` is set to
```

```
|      ``"best"``. When ``max_features < n_features``, the algorithm will
|      select ``max_features`` at random at each split before finding the best
|      split among them. But the best found split may vary across different
|      runs, even if ``max_features=n_features``. That is the case, if the
|      improvement of the criterion is identical for several splits and one
|      split has to be selected at random. To obtain a deterministic behaviour
|      during fitting, ``random_state`` has to be fixed to an integer.
|      See :term:`Glossary <random_state>` for details.
|
|  max_leaf_nodes : int, default=None
|      Grow a tree with ``max_leaf_nodes`` in best-first fashion.
|      Best nodes are defined as relative reduction in impurity.
|      If None then unlimited number of leaf nodes.
|
|  min_impurity_decrease : float, default=0.0
|      A node will be split if this split induces a decrease of the impurity
|      greater than or equal to this value.
|
|      The weighted impurity decrease equation is the following::
|
|          N_t / N * (impurity - N_t_R / N_t * right_impurity
|                              - N_t_L / N_t * left_impurity)
|
|      where ``N`` is the total number of samples, ``N_t`` is the number of
|      samples at the current node, ``N_t_L`` is the number of samples in the
|      left child, and ``N_t_R`` is the number of samples in the right child.
|
|      ``N``, ``N_t``, ``N_t_R`` and ``N_t_L`` all refer to the weighted sum,
|      if ``sample_weight`` is passed.
|
|      .. versionadded:: 0.19
|
|  class_weight : dict, list of dict or "balanced", default=None
|      Weights associated with classes in the form ``{class_label: weight}``.
|      If None, all classes are supposed to have weight one. For
|      multi-output problems, a list of dicts can be provided in the same
|      order as the columns of y.
|
|      Note that for multioutput (including multilabel) weights should be
|      defined for each class of every column in its own dict. For example,
|      for four-class multilabel classification weights should be
|      [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of
|      [{1:1}, {2:5}, {3:1}, {4:1}].
```

```
|
|      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))``
|
|      For multi-output, the weights of each column of y will be multiplied.
|
|      Note that these weights will be multiplied with sample_weight (passed
|      through the fit method) if sample_weight is specified.
|
|  ccp_alpha : non-negative float, default=0.0
|      Complexity parameter used for Minimal Cost-Complexity Pruning. The
|      subtree with the largest cost complexity that is smaller than
|      ``ccp_alpha`` will be chosen. By default, no pruning is performed. See
|      :ref:`minimal_cost_complexity_pruning` for details.
|
|      .. versionadded:: 0.22
|
|  Attributes
|  ----------
|  classes_ : ndarray of shape (n_classes,) or list of ndarray
|      The classes labels (single output problem),
|      or a list of arrays of class labels (multi-output problem).
|
|  feature_importances_ : ndarray of shape (n_features,)
|      The impurity-based feature importances.
|      The higher, the more important the feature.
|      The importance of a feature is computed as the (normalized)
|      total reduction of the criterion brought by that feature.  It is also
|      known as the Gini importance [4]_.
|
|      Warning: impurity-based feature importances can be misleading for
|      high cardinality features (many unique values). See
|      :func:`sklearn.inspection.permutation_importance` as an alternative.
|
|  max_features_ : int
|      The inferred value of max_features.
|
|  n_classes_ : int or list of int
|      The number of classes (for single output problems),
|      or a list containing the number of classes for each
|      output (for multi-output problems).
|
```

```
|  n_features_ : int
|      The number of features when ``fit`` is performed.
|
|      .. deprecated:: 1.0
|         `n_features_` is deprecated in 1.0 and will be removed in
|         1.2. Use `n_features_in_` instead.
|
|  n_features_in_ : int
|      Number of features seen during :term:`fit`.
|
|      .. versionadded:: 0.24
|
|  feature_names_in_ : ndarray of shape (`n_features_in_`,)
|      Names of features seen during :term:`fit`. Defined only when `X`
|      has feature names that are all strings.
|
|      .. versionadded:: 1.0
|
|  n_outputs_ : int
|      The number of outputs when ``fit`` is performed.
|
|  tree_ : Tree instance
|      The underlying Tree object. Please refer to
|      ``help(sklearn.tree._tree.Tree)`` for attributes of Tree object and
|      :ref:`sphx_glr_auto_examples_tree_plot_unveil_tree_structure.py`
|      for basic usage of these attributes.
|
|  See Also
|  --------
|  DecisionTreeRegressor : A decision tree regressor.
|
|  Notes
|  -----
|  The default values for the parameters controlling the size of the trees
|  (e.g. ``max_depth``, ``min_samples_leaf``, etc.) lead to fully grown and
|  unpruned trees which can potentially be very large on some data sets. To
|  reduce memory consumption, the complexity and size of the trees should be
|  controlled by setting those parameter values.
|
|  The :meth:`predict` method operates using the :func:`numpy.argmax`
|  function on the outputs of :meth:`predict_proba`. This means that in
|  case the highest predicted probabilities are tied, the classifier will
|  predict the tied class with the lowest index in :term:`classes_`.
```

```
 |
 |  References
 |  ----------
 |
 |  .. [1] https://en.wikipedia.org/wiki/Decision_tree_learning
 |
 |  .. [2] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification
 |         and Regression Trees", Wadsworth, Belmont, CA, 1984.
 |
 |  .. [3] T. Hastie, R. Tibshirani and J. Friedman. "Elements of Statistical
 |         Learning", Springer, 2009.
 |
 |  .. [4] L. Breiman, and A. Cutler, "Random Forests",
 |         https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
 |
 |  Examples
 |  --------
 |  >>> from sklearn.datasets import load_iris
 |  >>> from sklearn.model_selection import cross_val_score
 |  >>> from sklearn.tree import DecisionTreeClassifier
 |  >>> clf = DecisionTreeClassifier(random_state=0)
 |  >>> iris = load_iris()
 |  >>> cross_val_score(clf, iris.data, iris.target, cv=10)
 |  ...                             # doctest: +SKIP
 |  ...
 |  array([ 1.     ,  0.93...,  0.86...,  0.93...,  0.93...,
 |          0.93...,  0.93...,  1.     ,  0.93...,  1.      ])
 |
 |  Method resolution order:
 |      DecisionTreeClassifier
 |      sklearn.base.ClassifierMixin
 |      BaseDecisionTree
 |      sklearn.base.MultiOutputMixin
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, criterion='gini', splitter='best', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0,
class_weight=None, ccp_alpha=0.0)
 |      Initialize self.  See help(type(self)) for accurate signature.
```

```
|
|  fit(self, X, y, sample_weight=None, check_input=True, X_idx_sorted='deprecated')
|      Build a decision tree classifier from the training set (X, y).
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The training input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csc_matrix``.
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          The target values (class labels) as integers or strings.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights. If None, then samples are equally weighted. Splits
|          that would create child nodes with net zero or negative weight are
|          ignored while searching for a split in each node. Splits are also
|          ignored if they would result in any single class carrying a
|          negative weight in either child node.
|
|      check_input : bool, default=True
|          Allow to bypass several input checking.
|          Don't use this parameter unless you know what you do.
|
|      X_idx_sorted : deprecated, default="deprecated"
|          This parameter is deprecated and has no effect.
|          It will be removed in 1.1 (renaming of 0.26).
|
|          .. deprecated:: 0.24
|
|      Returns
|      -------
|      self : DecisionTreeClassifier
|          Fitted estimator.
|
|  predict_log_proba(self, X)
|      Predict class log-probabilities of the input samples X.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, it will be converted to
```

```
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csr_matrix``.
 |
 |      Returns
 |      -------
 |      proba : ndarray of shape (n_samples, n_classes) or list of n_outputs
such arrays if n_outputs > 1
 |          The class log-probabilities of the input samples. The order of the
 |          classes corresponds to that in the attribute :term:`classes_`.
 |
 |  predict_proba(self, X, check_input=True)
 |      Predict class probabilities of the input samples X.
 |
 |      The predicted class probability is the fraction of samples of the same
 |      class in a leaf.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csr_matrix``.
 |
 |      check_input : bool, default=True
 |          Allow to bypass several input checking.
 |          Don't use this parameter unless you know what you do.
 |
 |      Returns
 |      -------
 |      proba : ndarray of shape (n_samples, n_classes) or list of n_outputs
such arrays if n_outputs > 1
 |          The class probabilities of the input samples. The order of the
 |          classes corresponds to that in the attribute :term:`classes_`.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  n_features_
 |      DEPRECATED: The attribute `n_features_` is deprecated in 1.0 and will be
removed in 1.2. Use `n_features_in_` instead.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
```

```
|
|  __abstractmethods__ = frozenset()
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.ClassifierMixin:
|
|  score(self, X, y, sample_weight=None)
|      Return 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 of shape (n_samples, n_features)
|          Test samples.
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          True labels for `X`.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights.
|
|      Returns
|      -------
|      score : float
|          Mean accuracy of ``self.predict(X)`` wrt. `y`.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.ClassifierMixin:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from BaseDecisionTree:
|
|  apply(self, X, check_input=True)
|      Return the index of the leaf that each sample is predicted as.
```

```
|
|      .. versionadded:: 0.17
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      check_input : bool, default=True
|          Allow to bypass several input checking.
|          Don't use this parameter unless you know what you do.
|
|      Returns
|      -------
|      X_leaves : array-like of shape (n_samples,)
|          For each datapoint x in X, return the index of the leaf x
|          ends up in. Leaves are numbered within
|          ``[0; self.tree_.node_count)``, possibly with gaps in the
|          numbering.
|
|  cost_complexity_pruning_path(self, X, y, sample_weight=None)
|      Compute the pruning path during Minimal Cost-Complexity Pruning.
|
|      See :ref:`minimal_cost_complexity_pruning` for details on the pruning
|      process.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The training input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csc_matrix``.
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          The target values (class labels) as integers or strings.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights. If None, then samples are equally weighted. Splits
|          that would create child nodes with net zero or negative weight are
|          ignored while searching for a split in each node. Splits are also
|          ignored if they would result in any single class carrying a
```

```
|          negative weight in either child node.
|
|      Returns
|      -------
|      ccp_path : :class:`~sklearn.utils.Bunch`
|          Dictionary-like object, with the following attributes.
|
|          ccp_alphas : ndarray
|              Effective alphas of subtree during pruning.
|
|          impurities : ndarray
|              Sum of the impurities of the subtree leaves for the
|              corresponding alpha value in ``ccp_alphas``.
|
|  decision_path(self, X, check_input=True)
|      Return the decision path in the tree.
|
|      .. versionadded:: 0.18
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      check_input : bool, default=True
|          Allow to bypass several input checking.
|          Don't use this parameter unless you know what you do.
|
|      Returns
|      -------
|      indicator : sparse matrix of shape (n_samples, n_nodes)
|          Return a node indicator CSR matrix where non zero elements
|          indicates that the samples goes through the nodes.
|
|  get_depth(self)
|      Return the depth of the decision tree.
|
|      The depth of a tree is the maximum distance between the root
|      and any leaf.
|
|      Returns
```

```
|      -------
|      self.tree_.max_depth : int
|          The maximum depth of the tree.
|
|  get_n_leaves(self)
|      Return the number of leaves of the decision tree.
|
|      Returns
|      -------
|      self.tree_.n_leaves : int
|          Number of leaves.
|
|  predict(self, X, check_input=True)
|      Predict class or regression value for X.
|
|      For a classification model, the predicted class for each sample in X is
|      returned. For a regression model, the predicted value based on X is
|      returned.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      check_input : bool, default=True
|          Allow to bypass several input checking.
|          Don't use this parameter unless you know what you do.
|
|      Returns
|      -------
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          The predicted classes, or the predict values.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from BaseDecisionTree:
|
|  feature_importances_
|      Return the feature importances.
|
|      The importance of a feature is computed as the (normalized) total
|      reduction of the criterion brought by that feature.
```

```
|      It is also known as the Gini importance.
|
|      Warning: impurity-based feature importances can be misleading for
|      high cardinality features (many unique values). See
|      :func:`sklearn.inspection.permutation_importance` as an alternative.
|
|      Returns
|      -------
|      feature_importances_ : ndarray of shape (n_features,)
|          Normalized total reduction of criteria by feature
|          (Gini importance).
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|
|  __repr__(self, N_CHAR_MAX=700)
|      Return repr(self).
|
|  __setstate__(self, state)
|
|  get_params(self, deep=True)
|      Get parameters for this estimator.
|
|      Parameters
|      ----------
|      deep : bool, default=True
|          If True, will return the parameters for this estimator and
|          contained subobjects that are estimators.
|
|      Returns
|      -------
|      params : dict
|          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 :class:`~sklearn.pipeline.Pipeline`). The latter have
|      parameters of the form ``<component>__<parameter>`` so that it's
|      possible to update each component of a nested object.
```

```
 |
 |      Parameters
 |      ----------
 |      **params : dict
 |          Estimator parameters.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
```

Naive-bayes

```
from sklearn.naive_bayes import GaussianNB # GaussianNB # MultinomialNB --83%  #Complem
```

```
clf=GaussianNB()
```

```
help(clf)
```

```
Help on GaussianNB in module sklearn.naive_bayes object:

class GaussianNB(_BaseNB)
 |  GaussianNB(*, priors=None, var_smoothing=1e-09)
 |
 |  Gaussian Naive Bayes (GaussianNB).
 |
 |  Can perform online updates to model parameters via :meth:`partial_fit`.
 |  For details on algorithm used to update feature means and variance online,
 |  see Stanford CS tech report STAN-CS-79-773 by Chan, Golub, and LeVeque:
 |
 |      http://i.stanford.edu/pub/cstr/reports/cs/tr/79/773/CS-TR-79-773.pdf
 |
 |  Read more in the :ref:`User Guide <gaussian_naive_bayes>`.
 |
 |  Parameters
 |  ----------
 |  priors : array-like of shape (n_classes,)
 |      Prior probabilities of the classes. If specified the priors are not
 |      adjusted according to the data.
 |
 |  var_smoothing : float, default=1e-9
 |      Portion of the largest variance of all features that is added to
```

```
|      variances for calculation stability.
|
|      .. versionadded:: 0.20
|
|  Attributes
|  ----------
|  class_count_ : ndarray of shape (n_classes,)
|      number of training samples observed in each class.
|
|  class_prior_ : ndarray of shape (n_classes,)
|      probability of each class.
|
|  classes_ : ndarray of shape (n_classes,)
|      class labels known to the classifier.
|
|  epsilon_ : float
|      absolute additive value to variances.
|
|  n_features_in_ : int
|      Number of features seen during :term:`fit`.
|
|      .. versionadded:: 0.24
|
|  feature_names_in_ : ndarray of shape (`n_features_in_`,)
|      Names of features seen during :term:`fit`. Defined only when `X`
|      has feature names that are all strings.
|
|      .. versionadded:: 1.0
|
|  sigma_ : ndarray of shape (n_classes, n_features)
|      Variance of each feature per class.
|
|      .. deprecated:: 1.0
|         `sigma_` is deprecated in 1.0 and will be removed in 1.2.
|         Use `var_` instead.
|
|  var_ : ndarray of shape (n_classes, n_features)
|      Variance of each feature per class.
|
|      .. versionadded:: 1.0
|
|  theta_ : ndarray of shape (n_classes, n_features)
|      mean of each feature per class.
```

```
|
|  See Also
|  --------
|  BernoulliNB : Naive Bayes classifier for multivariate Bernoulli models.
|  CategoricalNB : Naive Bayes classifier for categorical features.
|  ComplementNB : Complement Naive Bayes classifier.
|  MultinomialNB : Naive Bayes classifier for multinomial models.
|
|  Examples
|  --------
|  >>> import numpy as np
|  >>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
|  >>> Y = np.array([1, 1, 1, 2, 2, 2])
|  >>> from sklearn.naive_bayes import GaussianNB
|  >>> clf = GaussianNB()
|  >>> clf.fit(X, Y)
|  GaussianNB()
|  >>> print(clf.predict([[-0.8, -1]]))
|  [1]
|  >>> clf_pf = GaussianNB()
|  >>> clf_pf.partial_fit(X, Y, np.unique(Y))
|  GaussianNB()
|  >>> print(clf_pf.predict([[-0.8, -1]]))
|  [1]
|
|  Method resolution order:
|      GaussianNB
|      _BaseNB
|      sklearn.base.ClassifierMixin
|      sklearn.base.BaseEstimator
|      builtins.object
|
|  Methods defined here:
|
|  __init__(self, *, priors=None, var_smoothing=1e-09)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  fit(self, X, y, sample_weight=None)
|      Fit Gaussian Naive Bayes according to X, y.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
```

```
|          Training vectors, where `n_samples` is the number of samples
|          and `n_features` is the number of features.
|
|      y : array-like of shape (n_samples,)
|          Target values.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Weights applied to individual samples (1. for unweighted).
|
|          .. versionadded:: 0.17
|             Gaussian Naive Bayes supports fitting with *sample_weight*.
|
|      Returns
|      -------
|      self : object
|          Returns the instance itself.
|
|  partial_fit(self, X, y, classes=None, sample_weight=None)
|      Incremental fit on a batch of samples.
|
|      This method is expected to be called several times consecutively
|      on different chunks of a dataset so as to implement out-of-core
|      or online learning.
|
|      This is especially useful when the whole dataset is too big to fit in
|      memory at once.
|
|      This method has some performance and numerical stability overhead,
|      hence it is better to call partial_fit on chunks of data that are
|      as large as possible (as long as fitting in the memory budget) to
|      hide the overhead.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
|          Training vectors, where `n_samples` is the number of samples and
|          `n_features` is the number of features.
|
|      y : array-like of shape (n_samples,)
|          Target values.
|
|      classes : array-like of shape (n_classes,), default=None
|          List of all the classes that can possibly appear in the y vector.
```

```
 |
 |          Must be provided at the first call to partial_fit, can be omitted
 |          in subsequent calls.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Weights applied to individual samples (1. for unweighted).
 |
 |          .. versionadded:: 0.17
 |
 |      Returns
 |      -------
 |      self : object
 |          Returns the instance itself.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  sigma_
 |      DEPRECATED: Attribute `sigma_` was deprecated in 1.0 and will be removed in1.2.
Use `var_` instead.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from _BaseNB:
 |
 |  predict(self, X)
 |      Perform classification on an array of test vectors X.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          The input samples.
 |
 |      Returns
 |      -------
 |      C : ndarray of shape (n_samples,)
 |          Predicted target values for X.
 |
 |  predict_log_proba(self, X)
```

```
|      Return log-probability estimates for the test vector X.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
|          The input samples.
|
|      Returns
|      -------
|      C : array-like of shape (n_samples, n_classes)
|          Returns the log-probability of the samples for each class in
|          the model. The columns correspond to the classes in sorted
|          order, as they appear in the attribute :term:`classes_`.
|
|  predict_proba(self, X)
|      Return probability estimates for the test vector X.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
|          The input samples.
|
|      Returns
|      -------
|      C : array-like of shape (n_samples, n_classes)
|          Returns the probability of the samples for each class in
|          the model. The columns correspond to the classes in sorted
|          order, as they appear in the attribute :term:`classes_`.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.ClassifierMixin:
|
|  score(self, X, y, sample_weight=None)
|      Return 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 of shape (n_samples, n_features)
|          Test samples.
```

```
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          True labels for `X`.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights.
|
|      Returns
|      -------
|      score : float
|          Mean accuracy of ``self.predict(X)`` wrt. `y`.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.ClassifierMixin:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|
|  __repr__(self, N_CHAR_MAX=700)
|      Return repr(self).
|
|  __setstate__(self, state)
|
|  get_params(self, deep=True)
|      Get parameters for this estimator.
|
|      Parameters
|      ----------
|      deep : bool, default=True
|          If True, will return the parameters for this estimator and
|          contained subobjects that are estimators.
|
|      Returns
|      -------
|      params : dict
```

```
 |          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 :class:`~sklearn.pipeline.Pipeline`). The latter have
 |      parameters of the form ``<component>__<parameter>`` so that it's
 |      possible to update each component of a nested object.
 |
 |      Parameters
 |      ----------
 |      **params : dict
 |          Estimator parameters.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
```

```
clf.fit(train_df,train_target)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().


GaussianNB()
```

```
clf.score(train_df,train_target)
```

```
0.9362637362637363
```

```
clf.score(val_df,val_target)
```

```
0.9649122807017544
```

```
val_pred=clf.predict(val_df)
val_pred
```

```
array(['B', 'M', 'M', 'B', 'B', 'M', 'M', 'M', 'M', 'B', 'B', 'M', 'B',
       'M', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M',
```

```
       'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'M',
       'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'M', 'M', 'B', 'B',
       'B', 'M', 'M', 'B', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'B', 'M', 'M', 'M', 'M', 'M', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'M', 'M', 'B', 'M', 'M',
       'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'M'], dtype='<U1')
```

```
result=pd.DataFrame({'predict':val_pred})  #.set_index('predict')
```

```
result
```

	predict
0	B
1	M
2	M
3	B
4	B
...	...
109	B
110	M
111	B
112	B
113	M

114 rows × 1 columns

```
result.to_csv('val_pred.csv')
```

```
os.listdir()
```

```
['.config',
 'val_pred.csv',
 'breast-cancer-wisconsin-data',
 'train_target.parquet',
 'train_df.parquet',
 'val_df.parquet',
 'val_target.parquet',
 'sample_data']
```

```
!pip install Ipython
from IPython.display import FileLink
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: Ipython in /usr/local/lib/python3.7/dist-packages
```

```
(5.5.0)
Requirement already satisfied: pexpect in /usr/local/lib/python3.7/dist-packages (from
Ipython) (4.8.0)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.7/dist-packages
(from Ipython) (0.7.5)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-packages
(from Ipython) (4.4.2)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in
/usr/local/lib/python3.7/dist-packages (from Ipython) (1.0.18)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-packages
(from Ipython) (5.1.1)
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-packages (from
Ipython) (2.6.1)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.7/dist-
packages (from Ipython) (57.4.0)
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.7/dist-
packages (from Ipython) (0.8.1)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-packages
(from prompt-toolkit<2.0.0,>=1.0.4->Ipython) (1.15.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages (from
prompt-toolkit<2.0.0,>=1.0.4->Ipython) (0.2.5)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-
packages (from pexpect->Ipython) (0.7.0)
```

```
FileLink('val_pred.csv')
```

val_pred.csv

I got 96.4% accuracy by using GaussianNB() on validation set.

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-
diagnostic
```

'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'

KNN

```
from sklearn.neighbors import KNeighborsClassifier
```

```
clf=KNeighborsClassifier(n_neighbors=4)
```

```
clf.fit(train_df,train_target)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().


KNeighborsClassifier(n_neighbors=4)
```

```
clf.score(train_df,train_target)
```

```
0.9714285714285714
```

```
clf.score(val_df,val_target)
```

```
0.9736842105263158
```

```
def test_func(md):
    clf=KNeighborsClassifier(n_neighbors=md)
    clf.fit(train_df,train_target)
    train_error=1- clf.score(train_df,train_target)
    val_error=1- clf.score(val_df,val_target)
    return({'Neighbor': md ,'training error': train_error,'val-error': val_error})
```

```
error_df=pd.DataFrame([test_func(md) for md in range(1,10)])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().

/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().
```

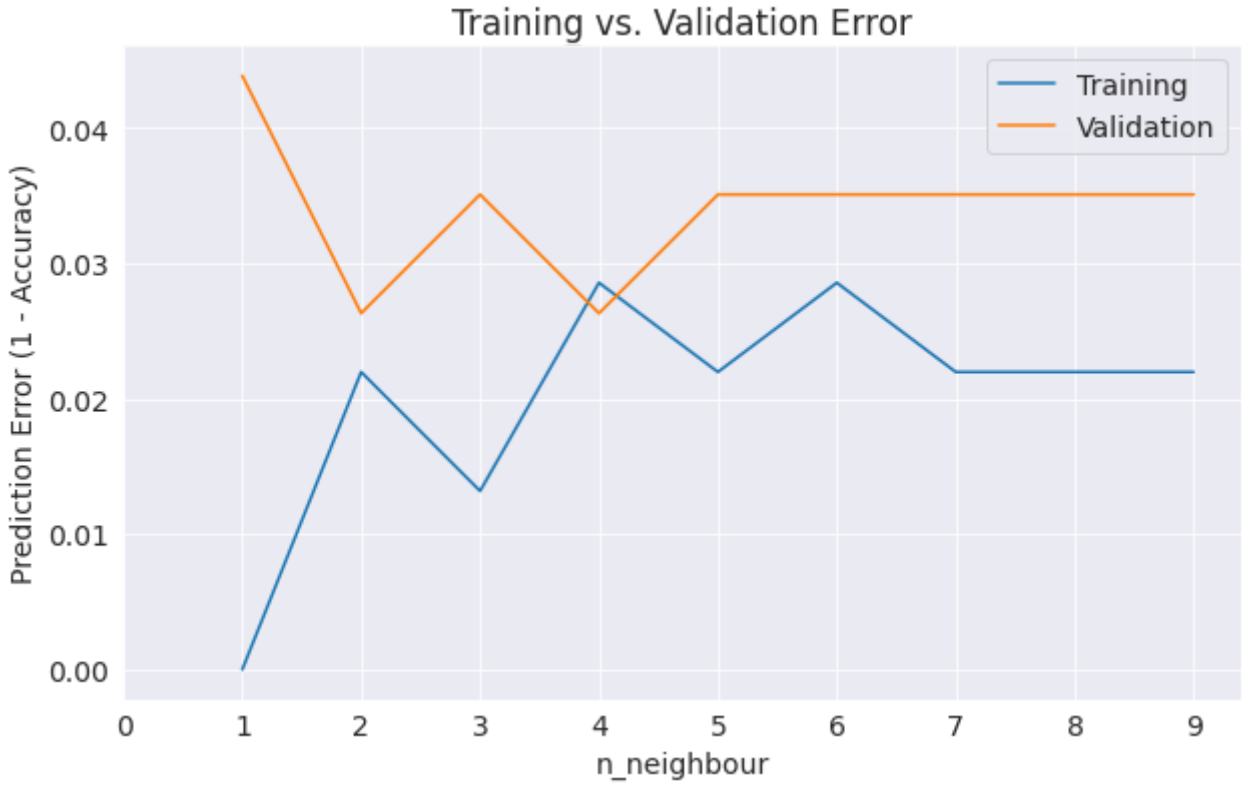
```
error_df
```

	Neighbor	training error	val-error
0	1	0.000000	0.043860
1	2	0.021978	0.026316
2	3	0.013187	0.035088

	Neighbor	training error	val-error
3	4	0.028571	0.026316
4	5	0.021978	0.035088
5	6	0.028571	0.035088
6	7	0.021978	0.035088
7	8	0.021978	0.035088
8	9	0.021978	0.035088

```
plt.figure()
plt.plot(error_df['Neighbor'], error_df['training error'])
plt.plot(error_df['Neighbor'], error_df['val-error'])
plt.title('Training vs. Validation Error')
plt.xticks(range(0,10, 1))
plt.xlabel('n_neighbour')
plt.ylabel('Prediction Error (1 - Accuracy)')
plt.legend(['Training', 'Validation'])
```

```
<matplotlib.legend.Legend at 0x7f94ab92bc50>
```



I got accuracy of 97.3% (KNN)on validation set at Neighbour count=4.

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

```
'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
```

```
model=RandomForestClassifier(n_jobs=-1,random_state=42)
model.fit(train_df,train_target)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().


RandomForestClassifier(n_jobs=-1, random_state=42)
```

```
model.score(train_df,train_target)
```

```
1.0
```

```
model.score(val_df,val_target)
```

```
0.9649122807017544
```

```
importance_df = pd.DataFrame({
    'feature': train_df.columns,
    'importance': model.feature_importances_
}).sort_values('importance', ascending=False)
```

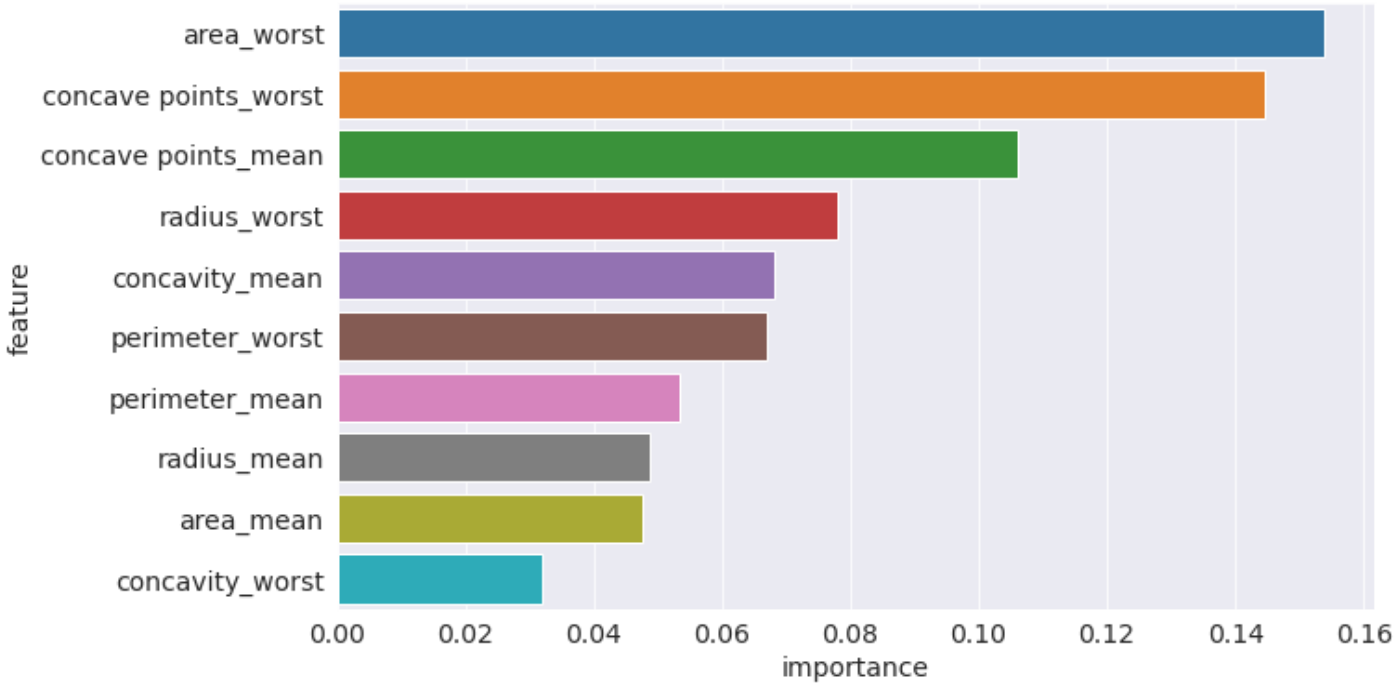
```
importance_df
```

	feature	importance
23	area_worst	0.153892
27	concave points_worst	0.144663
7	concave points_mean	0.106210
20	radius_worst	0.077987
6	concavity_mean	0.068001
22	perimeter_worst	0.067115
2	perimeter_mean	0.053270
0	radius_mean	0.048703
3	area_mean	0.047555
26	concavity_worst	0.031802
13	area_se	0.022407
21	texture_worst	0.021749
25	compactness_worst	0.020266

	feature	importance
10	radius_se	0.020139
5	compactness_mean	0.013944
1	texture_mean	0.013591
12	perimeter_se	0.011303
24	smoothness_worst	0.010644
28	symmetry_worst	0.010120
16	concavity_se	0.009386
4	smoothness_mean	0.007285
19	fractal_dimension_se	0.005321
15	compactness_se	0.005253
29	fractal_dimension_worst	0.005210
11	texture_se	0.004724
14	smoothness_se	0.004271
18	symmetry_se	0.004018
9	fractal_dimension_mean	0.003886
8	symmetry_mean	0.003770
17	concave points_se	0.003513

```
sns.barplot(data=importance_df.head(10),x='importance',y='feature')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f94ab657f90>
```




```
val_proba=model.predict_proba(val_df)
val_proba
```

```
array([[0.97, 0.03],
       [0.  , 1.  ],
```

```
[0.  , 1.  ],
[0.99, 0.01],
[1.  , 0.  ],
[0.  , 1.  ],
[0.  , 1.  ],
[0.16, 0.84],
[0.35, 0.65],
[0.94, 0.06],
[0.94, 0.06],
[0.02, 0.98],
[0.94, 0.06],
[0.12, 0.88],
[0.99, 0.01],
[0.01, 0.99],
[0.95, 0.05],
[1.  , 0.  ],
[1.  , 0.  ],
[0.  , 1.  ],
[0.85, 0.15],
[1.  , 0.  ],
[0.  , 1.  ],
[1.  , 0.  ],
[1.  , 0.  ],
[0.92, 0.08],
[1.  , 0.  ],
[0.96, 0.04],
[1.  , 0.  ],
[0.  , 1.  ],
[1.  , 0.  ],
[0.99, 0.01],
[0.77, 0.23],
[0.98, 0.02],
[1.  , 0.  ],
[1.  , 0.  ],
[0.24, 0.76],
[0.98, 0.02],
[0.  , 1.  ],
[0.9 , 0.1 ],
[1.  , 0.  ],
[0.02, 0.98],
[1.  , 0.  ],
[1.  , 0.  ],
[0.78, 0.22],
[0.99, 0.01],
[0.93, 0.07],
[0.96, 0.04],
[0.99, 0.01],
[0.97, 0.03],
[0.02, 0.98],
```

```
[0.  , 1.  ],
[0.8 , 0.2 ],
[0.87, 0.13],
[1.  , 0.  ],
[0.99, 0.01],
[1.  , 0.  ],
[0.  , 1.  ],
[0.29, 0.71],
[1.  , 0.  ],
[0.99, 0.01],
[0.  , 1.  ],
[0.  , 1.  ],
[0.92, 0.08],
[1.  , 0.  ],
[0.83, 0.17],
[0.  , 1.  ],
[0.  , 1.  ],
[1.  , 0.  ],
[1.  , 0.  ],
[0.16, 0.84],
[0.01, 0.99],
[0.99, 0.01],
[0.  , 1.  ],
[0.98, 0.02],
[0.96, 0.04],
[0.92, 0.08],
[0.71, 0.29],
[1.  , 0.  ],
[0.96, 0.04],
[0.01, 0.99],
[1.  , 0.  ],
[0.74, 0.26],
[0.01, 0.99],
[0.15, 0.85],
[0.  , 1.  ],
[0.07, 0.93],
[0.  , 1.  ],
[0.98, 0.02],
[0.99, 0.01],
[0.99, 0.01],
[0.8 , 0.2 ],
[0.77, 0.23],
[0.92, 0.08],
[1.  , 0.  ],
[1.  , 0.  ],
[0.  , 1.  ],
[0.01, 0.99],
[1.  , 0.  ],
[0.01, 0.99],
```

```
       [0.09, 0.91],
       [1.  , 0.  ],
       [0.  , 1.  ],
       [0.  , 1.  ],
       [0.98, 0.02],
       [0.98, 0.02],
       [0.99, 0.01],
       [0.  , 1.  ],
       [0.58, 0.42],
       [0.87, 0.13],
       [0.02, 0.98],
       [1.  , 0.  ],
       [0.71, 0.29],
       [0.  , 1.  ]])
```

```
model.classes_
```

```
array(['B', 'M'], dtype=object)
```

```
help(model)
```

```
Help on RandomForestClassifier in module sklearn.ensemble._forest object:

class RandomForestClassifier(ForestClassifier)
 |  RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True,
oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False,
class_weight=None, ccp_alpha=0.0, max_samples=None)
 |
 |  A random forest classifier.
 |
 |  A random forest is a meta estimator that fits a number of decision tree
 |  classifiers on various sub-samples of the dataset and uses averaging to
 |  improve the predictive accuracy and control over-fitting.
 |  The sub-sample size is controlled with the `max_samples` parameter if
 |  `bootstrap=True` (default), otherwise the whole dataset is used to build
 |  each tree.
 |
 |  Read more in the :ref:`User Guide <forest>`.
 |
 |  Parameters
 |  ----------
 |  n_estimators : int, default=100
 |      The number of trees in the forest.
 |
```

```
|      .. versionchanged:: 0.22
|         The default value of ``n_estimators`` changed from 10 to 100
|         in 0.22.
|
|  criterion : {"gini", "entropy"}, default="gini"
|      The function to measure the quality of a split. Supported criteria are
|      "gini" for the Gini impurity and "entropy" for the information gain.
|      Note: this parameter is tree-specific.
|
|  max_depth : int, default=None
|      The maximum depth of the tree. If None, then nodes are expanded until
|      all leaves are pure or until all leaves contain less than
|      min_samples_split samples.
|
|  min_samples_split : int or float, default=2
|      The minimum number of samples required to split an internal node:
|
|      - If int, then consider `min_samples_split` as the minimum number.
|      - If float, then `min_samples_split` is a fraction and
|        `ceil(min_samples_split * n_samples)` are the minimum
|        number of samples for each split.
|
|      .. versionchanged:: 0.18
|         Added float values for fractions.
|
|  min_samples_leaf : int or float, default=1
|      The minimum number of samples required to be at a leaf node.
|      A split point at any depth will only be considered if it leaves at
|      least ``min_samples_leaf`` training samples in each of the left and
|      right branches.  This may have the effect of smoothing the model,
|      especially in regression.
|
|      - If int, then consider `min_samples_leaf` as the minimum number.
|      - If float, then `min_samples_leaf` is a fraction and
|        `ceil(min_samples_leaf * n_samples)` are the minimum
|        number of samples for each node.
|
|      .. versionchanged:: 0.18
|         Added float values for fractions.
|
|  min_weight_fraction_leaf : float, default=0.0
|      The minimum weighted fraction of the sum total of weights (of all
|      the input samples) required to be at a leaf node. Samples have
```

```
|      equal weight when sample_weight is not provided.
|
|  max_features : {"auto", "sqrt", "log2"}, int or float, default="auto"
|      The number of features to consider when looking for the best split:
|
|      - If int, then consider `max_features` features at each split.
|      - If float, then `max_features` is a fraction and
|        `round(max_features * n_features)` features are considered at each
|        split.
|      - If "auto", then `max_features=sqrt(n_features)`.
|      - If "sqrt", then `max_features=sqrt(n_features)` (same as "auto").
|      - If "log2", then `max_features=log2(n_features)`.
|      - If None, then `max_features=n_features`.
|
|      Note: the search for a split does not stop until at least one
|      valid partition of the node samples is found, even if it requires to
|      effectively inspect more than ``max_features`` features.
|
|  max_leaf_nodes : int, default=None
|      Grow trees with ``max_leaf_nodes`` in best-first fashion.
|      Best nodes are defined as relative reduction in impurity.
|      If None then unlimited number of leaf nodes.
|
|  min_impurity_decrease : float, default=0.0
|      A node will be split if this split induces a decrease of the impurity
|      greater than or equal to this value.
|
|      The weighted impurity decrease equation is the following::
|
|          N_t / N * (impurity - N_t_R / N_t * right_impurity
|                              - N_t_L / N_t * left_impurity)
|
|      where ``N`` is the total number of samples, ``N_t`` is the number of
|      samples at the current node, ``N_t_L`` is the number of samples in the
|      left child, and ``N_t_R`` is the number of samples in the right child.
|
|      ``N``, ``N_t``, ``N_t_R`` and ``N_t_L`` all refer to the weighted sum,
|      if ``sample_weight`` is passed.
|
|      .. versionadded:: 0.19
|
|  bootstrap : bool, default=True
|      Whether bootstrap samples are used when building trees. If False, the
```

```
 |      whole dataset is used to build each tree.
 |
 |  oob_score : bool, default=False
 |      Whether to use out-of-bag samples to estimate the generalization score.
 |      Only available if bootstrap=True.
 |
 |  n_jobs : int, default=None
 |      The number of jobs to run in parallel. :meth:`fit`, :meth:`predict`,
 |      :meth:`decision_path` and :meth:`apply` are all parallelized over the
 |      trees. ``None`` means 1 unless in a :obj:`joblib.parallel_backend`
 |      context. ``-1`` means using all processors. See :term:`Glossary
 |      <n_jobs>` for more details.
 |
 |  random_state : int, RandomState instance or None, default=None
 |      Controls both the randomness of the bootstrapping of the samples used
 |      when building trees (if ``bootstrap=True``) and the sampling of the
 |      features to consider when looking for the best split at each node
 |      (if ``max_features < n_features``).
 |      See :term:`Glossary <random_state>` for details.
 |
 |  verbose : int, default=0
 |      Controls the verbosity when fitting and predicting.
 |
 |  warm_start : bool, default=False
 |      When set to ``True``, reuse the solution of the previous call to fit
 |      and add more estimators to the ensemble, otherwise, just fit a whole
 |      new forest. See :term:`the Glossary <warm_start>`.
 |
 |  class_weight : {"balanced", "balanced_subsample"}, dict or list of dicts,
default=None
 |      Weights associated with classes in the form ``{class_label: weight}``.
 |      If not given, all classes are supposed to have weight one. For
 |      multi-output problems, a list of dicts can be provided in the same
 |      order as the columns of y.
 |
 |      Note that for multioutput (including multilabel) weights should be
 |      defined for each class of every column in its own dict. For example,
 |      for four-class multilabel classification weights should be
 |      [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of
 |      [{1:1}, {2:5}, {3:1}, {4:1}].
 |
 |      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))``
|
|      The "balanced_subsample" mode is the same as "balanced" except that
|      weights are computed based on the bootstrap sample for every tree
|      grown.
|
|      For multi-output, the weights of each column of y will be multiplied.
|
|      Note that these weights will be multiplied with sample_weight (passed
|      through the fit method) if sample_weight is specified.
|
|  ccp_alpha : non-negative float, default=0.0
|      Complexity parameter used for Minimal Cost-Complexity Pruning. The
|      subtree with the largest cost complexity that is smaller than
|      ``ccp_alpha`` will be chosen. By default, no pruning is performed. See
|      :ref:`minimal_cost_complexity_pruning` for details.
|
|      .. versionadded:: 0.22
|
|  max_samples : int or float, default=None
|      If bootstrap is True, the number of samples to draw from X
|      to train each base estimator.
|
|      - If None (default), then draw `X.shape[0]` samples.
|      - If int, then draw `max_samples` samples.
|      - If float, then draw `max_samples * X.shape[0]` samples. Thus,
|        `max_samples` should be in the interval `(0.0, 1.0]`.
|
|      .. versionadded:: 0.22
|
|  Attributes
|  ----------
|  base_estimator_ : DecisionTreeClassifier
|      The child estimator template used to create the collection of fitted
|      sub-estimators.
|
|  estimators_ : list of DecisionTreeClassifier
|      The collection of fitted sub-estimators.
|
|  classes_ : ndarray of shape (n_classes,) or a list of such arrays
|      The classes labels (single output problem), or a list of arrays of
|      class labels (multi-output problem).
|
```

```
 |  n_classes_ : int or list
 |      The number of classes (single output problem), or a list containing the
 |      number of classes for each output (multi-output problem).
 |
 |  n_features_ : int
 |      The number of features when ``fit`` is performed.
 |
 |      .. deprecated:: 1.0
 |          Attribute `n_features_` was deprecated in version 1.0 and will be
 |          removed in 1.2. Use `n_features_in_` instead.
 |
 |  n_features_in_ : int
 |      Number of features seen during :term:`fit`.
 |
 |      .. versionadded:: 0.24
 |
 |  feature_names_in_ : ndarray of shape (`n_features_in_`,)
 |      Names of features seen during :term:`fit`. Defined only when `X`
 |      has feature names that are all strings.
 |
 |      .. versionadded:: 1.0
 |
 |  n_outputs_ : int
 |      The number of outputs when ``fit`` is performed.
 |
 |  feature_importances_ : ndarray of shape (n_features,)
 |      The impurity-based feature importances.
 |      The higher, the more important the feature.
 |      The importance of a feature is computed as the (normalized)
 |      total reduction of the criterion brought by that feature.  It is also
 |      known as the Gini importance.
 |
 |      Warning: impurity-based feature importances can be misleading for
 |      high cardinality features (many unique values). See
 |      :func:`sklearn.inspection.permutation_importance` as an alternative.
 |
 |  oob_score_ : float
 |      Score of the training dataset obtained using an out-of-bag estimate.
 |      This attribute exists only when ``oob_score`` is True.
 |
 |  oob_decision_function_ : ndarray of shape (n_samples, n_classes) or
(n_samples, n_classes, n_outputs)
 |      Decision function computed with out-of-bag estimate on the training
```

```
|      set. If n_estimators is small it might be possible that a data point
|      was never left out during the bootstrap. In this case,
|      `oob_decision_function_` might contain NaN. This attribute exists
|      only when ``oob_score`` is True.
|
|  See Also
|  --------
|  sklearn.tree.DecisionTreeClassifier : A decision tree classifier.
|  sklearn.ensemble.ExtraTreesClassifier : Ensemble of extremely randomized
|      tree classifiers.
|
|  Notes
|  -----
|  The default values for the parameters controlling the size of the trees
|  (e.g. ``max_depth``, ``min_samples_leaf``, etc.) lead to fully grown and
|  unpruned trees which can potentially be very large on some data sets. To
|  reduce memory consumption, the complexity and size of the trees should be
|  controlled by setting those parameter values.
|
|  The features are always randomly permuted at each split. Therefore,
|  the best found split may vary, even with the same training data,
|  ``max_features=n_features`` and ``bootstrap=False``, if the improvement
|  of the criterion is identical for several splits enumerated during the
|  search of the best split. To obtain a deterministic behaviour during
|  fitting, ``random_state`` has to be fixed.
|
|  References
|  ----------
|  .. [1] L. Breiman, "Random Forests", Machine Learning, 45(1), 5-32, 2001.
|
|  Examples
|  --------
|  >>> from sklearn.ensemble import RandomForestClassifier
|  >>> from sklearn.datasets import make_classification
|  >>> X, y = make_classification(n_samples=1000, n_features=4,
|  ...                            n_informative=2, n_redundant=0,
|  ...                            random_state=0, shuffle=False)
|  >>> clf = RandomForestClassifier(max_depth=2, random_state=0)
|  >>> clf.fit(X, y)
|  RandomForestClassifier(...)
|  >>> print(clf.predict([[0, 0, 0, 0]]))
|  [1]
|
```

```
 |  Method resolution order:
 |      RandomForestClassifier
 |      ForestClassifier
 |      sklearn.base.ClassifierMixin
 |      BaseForest
 |      sklearn.base.MultiOutputMixin
 |      sklearn.ensemble._base.BaseEnsemble
 |      sklearn.base.MetaEstimatorMixin
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, n_estimators=100, *, criterion='gini', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True,
oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False,
class_weight=None, ccp_alpha=0.0, max_samples=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from ForestClassifier:
 |
 |  predict(self, X)
 |      Predict class for X.
 |
 |      The predicted class of an input sample is a vote by the trees in
 |      the forest, weighted by their probability estimates. That is,
 |      the predicted class is the one with highest mean probability
 |      estimate across the trees.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, its dtype will be converted to
 |          ``dtype=np.float32``. If a sparse matrix is provided, it will be
 |          converted into a sparse ``csr_matrix``.
 |
```

```
|      Returns
|      -------
|      y : ndarray of shape (n_samples,) or (n_samples, n_outputs)
|          The predicted classes.
|
|  predict_log_proba(self, X)
|      Predict class log-probabilities for X.
|
|      The predicted class log-probabilities of an input sample is computed as
|      the log of the mean predicted class probabilities of the trees in the
|      forest.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, its dtype will be converted to
|          ``dtype=np.float32``. If a sparse matrix is provided, it will be
|          converted into a sparse ``csr_matrix``.
|
|      Returns
|      -------
|      p : ndarray of shape (n_samples, n_classes), or a list of such arrays
|          The class probabilities of the input samples. The order of the
|          classes corresponds to that in the attribute :term:`classes_`.
|
|  predict_proba(self, X)
|      Predict class probabilities for X.
|
|      The predicted class probabilities of an input sample are computed as
|      the mean predicted class probabilities of the trees in the forest.
|      The class probability of a single tree is the fraction of samples of
|      the same class in a leaf.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, its dtype will be converted to
|          ``dtype=np.float32``. If a sparse matrix is provided, it will be
|          converted into a sparse ``csr_matrix``.
|
|      Returns
|      -------
|      p : ndarray of shape (n_samples, n_classes), or a list of such arrays
```

```
|          The class probabilities of the input samples. The order of the
|          classes corresponds to that in the attribute :term:`classes_`.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.ClassifierMixin:
|
|  score(self, X, y, sample_weight=None)
|      Return 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 of shape (n_samples, n_features)
|          Test samples.
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          True labels for `X`.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights.
|
|      Returns
|      -------
|      score : float
|          Mean accuracy of ``self.predict(X)`` wrt. `y`.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.ClassifierMixin:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from BaseForest:
|
|  apply(self, X)
|      Apply trees in the forest to X, return leaf indices.
```

```
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, its dtype will be converted to
|          ``dtype=np.float32``. If a sparse matrix is provided, it will be
|          converted into a sparse ``csr_matrix``.
|
|      Returns
|      -------
|      X_leaves : ndarray of shape (n_samples, n_estimators)
|          For each datapoint x in X and for each tree in the forest,
|          return the index of the leaf x ends up in.
|
|  decision_path(self, X)
|      Return the decision path in the forest.
|
|      .. versionadded:: 0.18
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, its dtype will be converted to
|          ``dtype=np.float32``. If a sparse matrix is provided, it will be
|          converted into a sparse ``csr_matrix``.
|
|      Returns
|      -------
|      indicator : sparse matrix of shape (n_samples, n_nodes)
|          Return a node indicator matrix where non zero elements indicates
|          that the samples goes through the nodes. The matrix is of CSR
|          format.
|
|      n_nodes_ptr : ndarray of shape (n_estimators + 1,)
|          The columns from indicator[n_nodes_ptr[i]:n_nodes_ptr[i+1]]
|          gives the indicator value for the i-th estimator.
|
|  fit(self, X, y, sample_weight=None)
|      Build a forest of trees from the training set (X, y).
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
```

```
|          The training input samples. Internally, its dtype will be converted
|          to ``dtype=np.float32``. If a sparse matrix is provided, it will be
|          converted into a sparse ``csc_matrix``.
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          The target values (class labels in classification, real numbers in
|          regression).
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights. If None, then samples are equally weighted. Splits
|          that would create child nodes with net zero or negative weight are
|          ignored while searching for a split in each node. In the case of
|          classification, splits are also ignored if they would result in any
|          single class carrying a negative weight in either child node.
|
|      Returns
|      -------
|      self : object
|          Fitted estimator.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from BaseForest:
|
|  feature_importances_
|      The impurity-based feature importances.
|
|      The higher, the more important the feature.
|      The importance of a feature is computed as the (normalized)
|      total reduction of the criterion brought by that feature.  It is also
|      known as the Gini importance.
|
|      Warning: impurity-based feature importances can be misleading for
|      high cardinality features (many unique values). See
|      :func:`sklearn.inspection.permutation_importance` as an alternative.
|
|      Returns
|      -------
|      feature_importances_ : ndarray of shape (n_features,)
|          The values of this array sum to 1, unless all trees are single node
|          trees consisting of only the root node, in which case it will be an
|          array of zeros.
|
|  n_features_
```

```
 |      DEPRECATED: Attribute `n_features_` was deprecated in version 1.0 and will be
removed in 1.2. Use `n_features_in_` instead.
 |
 |      Number of features when fitting the estimator.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.ensemble._base.BaseEnsemble:
 |
 |  __getitem__(self, index)
 |      Return the index'th estimator in the ensemble.
 |
 |  __iter__(self)
 |      Return iterator over estimators in the ensemble.
 |
 |  __len__(self)
 |      Return the number of estimators in the ensemble.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from sklearn.ensemble._base.BaseEnsemble:
 |
 |  __annotations__ = {'_required_parameters': typing.List[str]}
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |
 |      Parameters
 |      ----------
 |      deep : bool, default=True
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |
 |      Returns
 |      -------
```

```
|      params : dict
|          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 :class:`~sklearn.pipeline.Pipeline`). The latter have
|      parameters of the form ``<component>__<parameter>`` so that it's
|      possible to update each component of a nested object.
|
|      Parameters
|      ----------
|      **params : dict
|          Estimator parameters.
|
|      Returns
|      -------
|      self : estimator instance
|          Estimator instance.
```

```
def test_params(**params):
    model=RandomForestClassifier(n_jobs=-1,random_state=42,**params)
    model.fit(train_df,train_target)
    return model.score(train_df,train_target),model.score(val_df,val_target)
```

```
test_params(max_depth=26)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().


(1.0, 0.9649122807017544)
```

```
test_params(max_leaf_nodes=2**26)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().


(1.0, 0.956140350877193)
```

```
test_params(max_features='log2')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().


(1.0, 0.9649122807017544)
```

```
test_params(min_samples_split=100, min_samples_leaf=60)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().


(0.9472527472527472, 0.9649122807017544)
```

```
test_params(bootstrap=False)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().


(1.0, 0.956140350877193)
```

```
test_params(class_weight='balanced')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().


(1.0, 0.9649122807017544)
```

```
test_params(n_estimators=200)
```

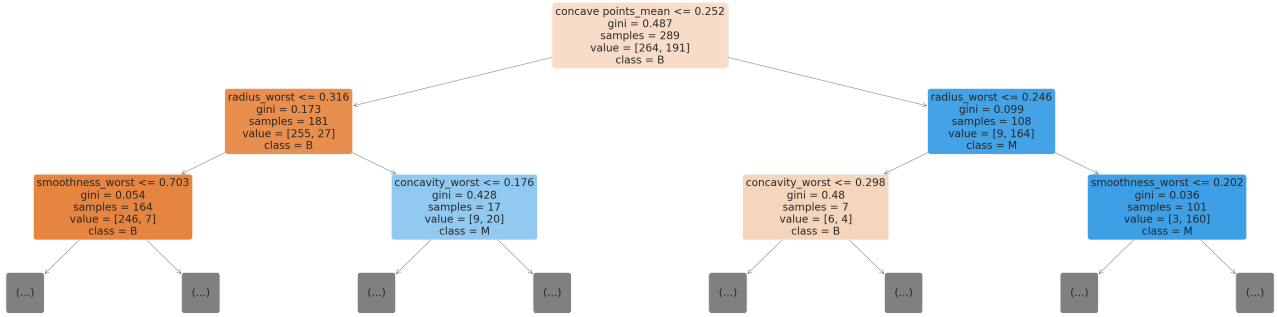
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().
```

```
(1.0, 0.9649122807017544)
```

```
plt.figure(figsize=(80,20))
plot_tree(model.estimators_[99],max_depth=2,feature_names=train_df.columns,filled=True,
```

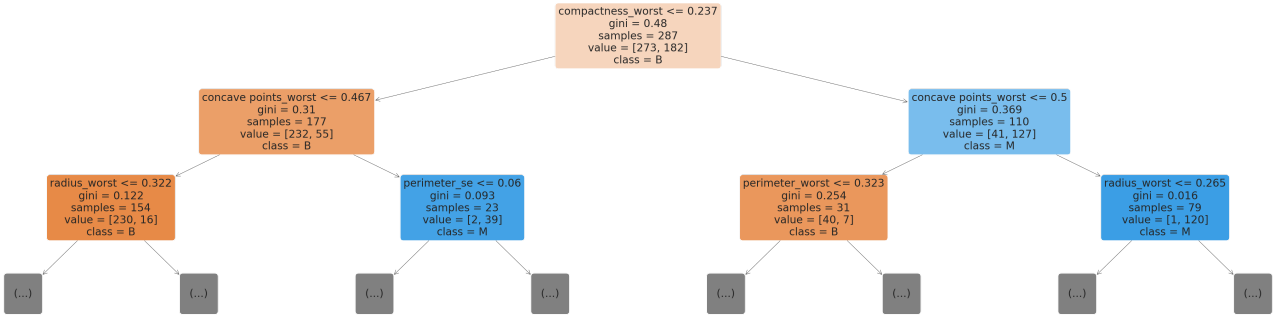
```
[Text(0.5, 0.875, 'concave points_mean <= 0.252\ngini = 0.487\nsamples = 289\nvalue =
[264, 191]\nclass = B'),
 Text(0.25, 0.625, 'radius_worst <= 0.316\ngini = 0.173\nsamples = 181\nvalue = [255,
27]\nclass = B'),
 Text(0.125, 0.375, 'smoothness_worst <= 0.703\ngini = 0.054\nsamples = 164\nvalue =
[246, 7]\nclass = B'),
 Text(0.0625, 0.125, '\n  (...)  \n'),
 Text(0.1875, 0.125, '\n  (...)  \n'),
 Text(0.375, 0.375, 'concavity_worst <= 0.176\ngini = 0.428\nsamples = 17\nvalue = [9,
20]\nclass = M'),
 Text(0.3125, 0.125, '\n  (...)  \n'),
 Text(0.4375, 0.125, '\n  (...)  \n'),
 Text(0.75, 0.625, 'radius_worst <= 0.246\ngini = 0.099\nsamples = 108\nvalue = [9,
164]\nclass = M'),
 Text(0.625, 0.375, 'concavity_worst <= 0.298\ngini = 0.48\nsamples = 7\nvalue = [6,
4]\nclass = B'),
 Text(0.5625, 0.125, '\n  (...)  \n'),
 Text(0.6875, 0.125, '\n  (...)  \n'),
 Text(0.875, 0.375, 'smoothness_worst <= 0.202\ngini = 0.036\nsamples = 101\nvalue =
[3, 160]\nclass = M'),
 Text(0.8125, 0.125, '\n  (...)  \n'),
 Text(0.9375, 0.125, '\n  (...)  \n')]
```




```
plt.figure(figsize=(80,20))
plot_tree(model.estimators_[5],max_depth=2,feature_names=train_df.columns,filled=True,r
```

```
[Text(0.5, 0.875, 'compactness_worst <= 0.237\ngini = 0.48\nsamples = 287\nvalue =
[273, 182]\nclass = B'),
 Text(0.25, 0.625, 'concave points_worst <= 0.467\ngini = 0.31\nsamples = 177\nvalue =
[232, 55]\nclass = B'),
 Text(0.125, 0.375, 'radius_worst <= 0.322\ngini = 0.122\nsamples = 154\nvalue = [230,
16]\nclass = B'),
 Text(0.0625, 0.125, '\n  (...)  \n'),
 Text(0.1875, 0.125, '\n  (...)  \n'),
 Text(0.375, 0.375, 'perimeter_se <= 0.06\ngini = 0.093\nsamples = 23\nvalue = [2,
39]\nclass = M'),
```

```
 Text(0.3125, 0.125, '\n  (...)  \n'),
 Text(0.4375, 0.125, '\n  (...)  \n'),
 Text(0.75, 0.625, 'concave points_worst <= 0.5\ngini = 0.369\nsamples = 110\nvalue =
[41, 127]\nclass = M'),
 Text(0.625, 0.375, 'perimeter_worst <= 0.323\ngini = 0.254\nsamples = 31\nvalue = [40,
7]\nclass = B'),
 Text(0.5625, 0.125, '\n  (...)  \n'),
 Text(0.6875, 0.125, '\n  (...)  \n'),
 Text(0.875, 0.375, 'radius_worst <= 0.265\ngini = 0.016\nsamples = 79\nvalue = [1,
120]\nclass = M'),
 Text(0.8125, 0.125, '\n  (...)  \n'),
 Text(0.9375, 0.125, '\n  (...)  \n')]
```



From Random Forest I got accuracy of 96% on validation data set.

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-
diagnostic
```

'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'

xgboost

```
! pip install xgboost --upgrade --quiet
from xgboost import XGBClassifier
```

```
     |████████████████████████████████| 192.9 MB 64 kB/s
```

```
model=XGBClassifier(random_state=42,n_jobs=-1,n_estimators=100,max_depth=10,learning_ra
```

```
help(model)
```

```
Help on XGBClassifier in module xgboost.sklearn object:

class XGBClassifier(XGBModel, sklearn.base.ClassifierMixin)
 |  XGBClassifier(*, objective: Union[str, Callable[[numpy.ndarray, numpy.ndarray],
Tuple[numpy.ndarray, numpy.ndarray]], NoneType] = 'binary:logistic', use_label_encoder:
```

```
bool = False, **kwargs: Any) -> None
 |
 |  Implementation of the scikit-learn API for XGBoost classification.
 |
 |
 |  Parameters
 |  ----------
 |
 |      n_estimators : int
 |          Number of boosting rounds.
 |
 |      max_depth :  Optional[int]
 |          Maximum tree depth for base learners.
 |      max_leaves :
 |          Maximum number of leaves; 0 indicates no limit.
 |      max_bin :
 |          If using histogram-based algorithm, maximum number of bins per feature
 |      grow_policy :
 |          Tree growing policy. 0: favor splitting at nodes closest to the node, i.e.
grow
 |          depth-wise. 1: favor splitting at nodes with highest loss change.
 |      learning_rate : Optional[float]
 |          Boosting learning rate (xgb's "eta")
 |      verbosity : Optional[int]
 |          The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
 |      objective : typing.Union[str, typing.Callable[[numpy.ndarray, numpy.ndarray],
typing.Tuple[numpy.ndarray, numpy.ndarray]], NoneType]
 |          Specify the learning task and the corresponding learning objective or
 |          a custom objective function to be used (see note below).
 |      booster: Optional[str]
 |          Specify which booster to use: gbtree, gblinear or dart.
 |      tree_method: Optional[str]
 |          Specify which tree method to use.  Default to auto.  If this parameter is
set to
 |          default, XGBoost will choose the most conservative option available.  It's
 |          recommended to study this option from the parameters document :doc:`tree
method
 |          </treemethod>`
 |      n_jobs : Optional[int]
 |          Number of parallel threads used to run xgboost.  When used with other
Scikit-Learn
 |          algorithms like grid search, you may choose which algorithm to parallelize
and
```

```
 |          balance the threads.  Creating thread contention will significantly slow
down both
 |          algorithms.
 |      gamma : Optional[float]
 |          (min_split_loss) Minimum loss reduction required to make a further
partition on a
 |          leaf node of the tree.
 |      min_child_weight : Optional[float]
 |          Minimum sum of instance weight(hessian) needed in a child.
 |      max_delta_step : Optional[float]
 |          Maximum delta step we allow each tree's weight estimation to be.
 |      subsample : Optional[float]
 |          Subsample ratio of the training instance.
 |      sampling_method :
 |          Sampling method. Used only by `gpu_hist` tree method.
 |            - `uniform`: select random training instances uniformly.
 |            - `gradient_based` select random training instances with higher
probability when
 |              the gradient and hessian are larger. (cf. CatBoost)
 |      colsample_bytree : Optional[float]
 |          Subsample ratio of columns when constructing each tree.
 |      colsample_bylevel : Optional[float]
 |          Subsample ratio of columns for each level.
 |      colsample_bynode : Optional[float]
 |          Subsample ratio of columns for each split.
 |      reg_alpha : Optional[float]
 |          L1 regularization term on weights (xgb's alpha).
 |      reg_lambda : Optional[float]
 |          L2 regularization term on weights (xgb's lambda).
 |      scale_pos_weight : Optional[float]
 |          Balancing of positive and negative weights.
 |      base_score : Optional[float]
 |          The initial prediction score of all instances, global bias.
 |      random_state : Optional[Union[numpy.random.RandomState, int]]
 |          Random number seed.
 |
 |          .. note::
 |
 |             Using gblinear booster with shotgun updater is nondeterministic as
 |             it uses Hogwild algorithm.
 |
 |      missing : float, default np.nan
 |          Value in the data which needs to be present as a missing value.
```

```
 |      num_parallel_tree: Optional[int]
 |          Used for boosting random forest.
 |      monotone_constraints : Optional[Union[Dict[str, int], str]]
 |          Constraint of variable monotonicity.  See :doc:`tutorial
</tutorials/monotonic>`
 |          for more information.
 |      interaction_constraints : Optional[Union[str, List[Tuple[str]]]]
 |          Constraints for interaction representing permitted interactions.  The
 |          constraints must be specified in the form of a nested list, e.g. ``[[0, 1],
[2,
 |          3, 4]]``, where each inner list is a group of indices of features that are
 |          allowed to interact with each other.  See :doc:`tutorial
 |          </tutorials/feature_interaction_constraint>` for more information
 |      importance_type: Optional[str]
 |          The feature importance type for the feature_importances\_ property:
 |
 |          * For tree model, it's either "gain", "weight", "cover", "total_gain" or
 |            "total_cover".
 |          * For linear model, only "weight" is defined and it's the normalized
coefficients
 |            without bias.
 |
 |      gpu_id : Optional[int]
 |          Device ordinal.
 |      validate_parameters : Optional[bool]
 |          Give warnings for unknown parameter.
 |      predictor : Optional[str]
 |          Force XGBoost to use specific predictor, available choices are
[cpu_predictor,
 |          gpu_predictor].
 |      enable_categorical : bool
 |
 |          .. versionadded:: 1.5.0
 |
 |          .. note:: This parameter is experimental
 |
 |          Experimental support for categorical data.  When enabled,
cudf/pandas.DataFrame
 |          should be used to specify categorical data type.  Also, JSON/UBJSON
 |          serialization format is required.
 |
 |      max_cat_to_onehot : Optional[int]
 |
```

```
 |          .. versionadded:: 1.6.0
 |
 |          .. note:: This parameter is experimental
 |
 |          A threshold for deciding whether XGBoost should use one-hot encoding based
split
 |          for categorical data.  When number of categories is lesser than the
threshold
 |          then one-hot encoding is chosen, otherwise the categories will be
partitioned
 |          into children nodes.  Only relevant for regression and binary
classification.
 |          See :doc:`Categorical Data </tutorials/categorical>` for details.
 |
 |      eval_metric : Optional[Union[str, List[str], Callable]]
 |
 |          .. versionadded:: 1.6.0
 |
 |          Metric used for monitoring the training result and early stopping.  It can
be a
 |          string or list of strings as names of predefined metric in XGBoost (See
 |          doc/parameter.rst), one of the metrics in :py:mod:`sklearn.metrics`, or any
other
 |          user defined metric that looks like `sklearn.metrics`.
 |
 |          If custom objective is also provided, then custom metric should implement
the
 |          corresponding reverse link function.
 |
 |          Unlike the `scoring` parameter commonly used in scikit-learn, when a
callable
 |          object is provided, it's assumed to be a cost function and by default
XGBoost will
 |          minimize the result during early stopping.
 |
 |          For advanced usage on Early stopping like directly choosing to maximize
instead of
 |          minimize, see :py:obj:`xgboost.callback.EarlyStopping`.
 |
 |          See :doc:`Custom Objective and Evaluation Metric
</tutorials/custom_metric_obj>`
 |          for more.
 |
```

```
 |          .. note::
 |
 |               This parameter replaces `eval_metric` in :py:meth:`fit` method.  The
old one
 |               receives un-transformed prediction regardless of whether custom
objective is
 |               being used.
 |
 |          .. code-block:: python
 |
 |              from sklearn.datasets import load_diabetes
 |              from sklearn.metrics import mean_absolute_error
 |              X, y = load_diabetes(return_X_y=True)
 |              reg = xgb.XGBRegressor(
 |                  tree_method="hist",
 |                  eval_metric=mean_absolute_error,
 |              )
 |              reg.fit(X, y, eval_set=[(X, y)])
 |
 |      early_stopping_rounds : Optional[int]
 |
 |          .. versionadded:: 1.6.0
 |
 |          Activates early stopping. Validation metric needs to improve at least once
in
 |          every **early_stopping_rounds** round(s) to continue training.  Requires at
least
 |          one item in **eval_set** in :py:meth:`fit`.
 |
 |          The method returns the model from the last iteration (not the best one).
If
 |          there's more than one item in **eval_set**, the last entry will be used for
early
 |          stopping.  If there's more than one metric in **eval_metric**, the last
metric
 |          will be used for early stopping.
 |
 |          If early stopping occurs, the model will have three additional fields:
 |          :py:attr:`best_score`, :py:attr:`best_iteration` and
 |          :py:attr:`best_ntree_limit`.
 |
 |          .. note::
 |
```

```
 |              This parameter replaces `early_stopping_rounds` in :py:meth:`fit`
method.
 |
 |      callbacks : Optional[List[TrainingCallback]]
 |          List of callback functions that are applied at end of each iteration.
 |          It is possible to use predefined callbacks by using
 |          :ref:`Callback API <callback_api>`.
 |
 |          .. note::
 |
 |             States in callback are not preserved during training, which means
callback
 |             objects can not be reused for multiple training sessions without
 |             reinitialization or deepcopy.
 |
 |          .. code-block:: python
 |
 |              for params in parameters_grid:
 |                  # be sure to (re)initialize the callbacks before each run
 |                  callbacks = [xgb.callback.LearningRateScheduler(custom_rates)]
 |                  xgboost.train(params, Xy, callbacks=callbacks)
 |
 |      kwargs : dict, optional
 |          Keyword arguments for XGBoost Booster object.  Full documentation of
parameters
 |          can be found :doc:`here </parameter>`.
 |          Attempting to set a parameter via the constructor args and \*\*kwargs
 |          dict simultaneously will result in a TypeError.
 |
 |          .. note:: \*\*kwargs unsupported by scikit-learn
 |
 |              \*\*kwargs is unsupported by scikit-learn.  We do not guarantee
 |              that parameters passed via this argument will interact properly
 |              with scikit-learn.
 |
 |          .. note::  Custom objective function
 |
 |              A custom objective function can be provided for the ``objective``
 |              parameter. In this case, it should have the signature
 |              ``objective(y_true, y_pred) -> grad, hess``:
 |
 |              y_true: array_like of shape [n_samples]
 |                  The target values
```

```
 |              y_pred: array_like of shape [n_samples]
 |                  The predicted values
 |
 |              grad: array_like of shape [n_samples]
 |                  The value of the gradient for each sample point.
 |              hess: array_like of shape [n_samples]
 |                  The value of the second derivative for each sample point
 |
 |  Method resolution order:
 |      XGBClassifier
 |      XGBModel
 |      sklearn.base.BaseEstimator
 |      sklearn.base.ClassifierMixin
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, objective: Union[str, Callable[[numpy.ndarray, numpy.ndarray],
Tuple[numpy.ndarray, numpy.ndarray]], NoneType] = 'binary:logistic', use_label_encoder:
bool = False, **kwargs: Any) -> None
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X: Any, y: Any, *, sample_weight: Union[Any, NoneType] = None,
base_margin: Union[Any, NoneType] = None, eval_set: Union[Sequence[Tuple[Any, Any]],
NoneType] = None, eval_metric: Union[str, Sequence[str], Callable[[numpy.ndarray,
xgboost.core.DMatrix], Tuple[str, float]], NoneType] = None, early_stopping_rounds:
Union[int, NoneType] = None, verbose: Union[bool, NoneType] = True, xgb_model:
Union[xgboost.core.Booster, str, xgboost.sklearn.XGBModel, NoneType] = None,
sample_weight_eval_set: Union[Sequence[Any], NoneType] = None, base_margin_eval_set:
Union[Sequence[Any], NoneType] = None, feature_weights: Union[Any, NoneType] = None,
callbacks: Union[Sequence[xgboost.callback.TrainingCallback], NoneType] = None) ->
'XGBClassifier'
 |      Fit gradient boosting classifier.
 |
 |      Note that calling ``fit()`` multiple times will cause the model object to be
 |      re-fit from scratch. To resume training from a previous checkpoint, explicitly
 |      pass ``xgb_model`` argument.
 |
 |      Parameters
 |      ----------
 |      X :
 |          Feature matrix
 |      y :
```

```
 |          Labels
 |      sample_weight :
 |          instance weights
 |      base_margin :
 |          global bias for each instance.
 |      eval_set :
 |          A list of (X, y) tuple pairs to use as validation sets, for which
 |          metrics will be computed.
 |          Validation metrics will help us track the performance of the model.
 |
 |      eval_metric : str, list of str, or callable, optional
 |          .. deprecated:: 1.6.0
 |              Use `eval_metric` in :py:meth:`__init__` or :py:meth:`set_params`
instead.
 |
 |      early_stopping_rounds : int
 |          .. deprecated:: 1.6.0
 |              Use `early_stopping_rounds` in :py:meth:`__init__` or
 |              :py:meth:`set_params` instead.
 |      verbose :
 |          If `verbose` and an evaluation set is used, writes the evaluation metric
 |          measured on the validation set to stderr.
 |      xgb_model :
 |          file name of stored XGBoost model or 'Booster' instance XGBoost model to be
 |          loaded before training (allows training continuation).
 |      sample_weight_eval_set :
 |          A list of the form [L_1, L_2, ..., L_n], where each L_i is an array like
 |          object storing instance weights for the i-th validation set.
 |      base_margin_eval_set :
 |          A list of the form [M_1, M_2, ..., M_n], where each M_i is an array like
 |          object storing base margin for the i-th validation set.
 |      feature_weights :
 |          Weight for each feature, defines the probability of each feature being
 |          selected when colsample is being used.  All values must be greater than 0,
 |          otherwise a `ValueError` is thrown.
 |
 |      callbacks :
 |          .. deprecated:: 1.6.0
 |              Use `callbacks` in :py:meth:`__init__` or :py:meth:`set_params`
instead.
 |
 |  predict(self, X: Any, output_margin: bool = False, ntree_limit: Union[int,
NoneType] = None, validate_features: bool = True, base_margin: Union[Any, NoneType] =
```

```
None, iteration_range: Union[Tuple[int, int], NoneType] = None) -> numpy.ndarray
 |      Predict with `X`.  If the model is trained with early stopping, then
`best_iteration`
 |      is used automatically.  For tree models, when data is on GPU, like cupy array
or
 |      cuDF dataframe and `predictor` is not specified, the prediction is run on GPU
 |      automatically, otherwise it will run on CPU.
 |
 |      .. note:: This function is only thread safe for `gbtree` and `dart`.
 |
 |      Parameters
 |      ----------
 |      X :
 |          Data to predict with.
 |      output_margin :
 |          Whether to output the raw untransformed margin value.
 |      ntree_limit :
 |          Deprecated, use `iteration_range` instead.
 |      validate_features :
 |          When this is True, validate that the Booster's and data's feature_names are
 |          identical.  Otherwise, it is assumed that the feature_names are the same.
 |      base_margin :
 |          Margin added to prediction.
 |      iteration_range :
 |          Specifies which layer of trees are used in prediction.  For example, if a
 |          random forest is trained with 100 rounds.  Specifying ``iteration_range=
(10,
 |          20)``, then only the forests built during [10, 20) (half open set) rounds
are
 |          used in this prediction.
 |
 |          .. versionadded:: 1.4.0
 |
 |      Returns
 |      -------
 |      prediction
 |
 |  predict_proba(self, X: Any, ntree_limit: Union[int, NoneType] = None,
validate_features: bool = True, base_margin: Union[Any, NoneType] = None,
iteration_range: Union[Tuple[int, int], NoneType] = None) -> numpy.ndarray
 |      Predict the probability of each `X` example being of a given class.
 |
 |      .. note:: This function is only thread safe for `gbtree` and `dart`.
```

```
 |
 |      Parameters
 |      ----------
 |      X : array_like
 |          Feature matrix.
 |      ntree_limit : int
 |          Deprecated, use `iteration_range` instead.
 |      validate_features : bool
 |          When this is True, validate that the Booster's and data's feature_names are
 |          identical.  Otherwise, it is assumed that the feature_names are the same.
 |      base_margin : array_like
 |          Margin added to prediction.
 |      iteration_range :
 |          Specifies which layer of trees are used in prediction.  For example, if a
 |          random forest is trained with 100 rounds.  Specifying `iteration_range=(10,
 |          20)`, then only the forests built during [10, 20) (half open set) rounds
are
 |          used in this prediction.
 |
 |      Returns
 |      -------
 |      prediction :
 |          a numpy array of shape array-like of shape (n_samples, n_classes) with the
 |          probability of each data example being of a given class.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from XGBModel:
 |
 |  __sklearn_is_fitted__(self) -> bool
 |
 |  apply(self, X: Any, ntree_limit: int = 0, iteration_range: Union[Tuple[int, int],
NoneType] = None) -> numpy.ndarray
 |      Return the predicted leaf every tree for each sample. If the model is trained
with
 |      early stopping, then `best_iteration` is used automatically.
 |
 |      Parameters
 |      ----------
 |      X : array_like, shape=[n_samples, n_features]
 |          Input features matrix.
 |
 |      iteration_range :
 |          See :py:meth:`predict`.
```

```
|
|      ntree_limit :
|          Deprecated, use ``iteration_range`` instead.
|
|      Returns
|      -------
|      X_leaves : array_like, shape=[n_samples, n_trees]
|          For each datapoint x in X and for each tree, return the index of the
|          leaf x ends up in. Leaves are numbered within
|          ``[0; 2**(self.max_depth+1))``, possibly with gaps in the numbering.
|
|  evals_result(self) -> Dict[str, Dict[str, List[float]]]
|      Return the evaluation results.
|
|      If **eval_set** is passed to the :py:meth:`fit` function, you can call
|      ``evals_result()`` to get evaluation results for all passed **eval_sets**.
When
|      **eval_metric** is also passed to the :py:meth:`fit` function, the
|      **evals_result** will contain the **eval_metrics** passed to the :py:meth:`fit`
|      function.
|
|      The returned evaluation result is a dictionary:
|
|      .. code-block:: python
|
|          {'validation_0': {'logloss': ['0.604835', '0.531479']},
|           'validation_1': {'logloss': ['0.41965', '0.17686']}}
|
|      Returns
|      -------
|      evals_result
|
|  get_booster(self) -> xgboost.core.Booster
|      Get the underlying xgboost Booster of this model.
|
|      This will raise an exception when fit was not called
|
|      Returns
|      -------
|      booster : a xgboost booster of underlying model
|
|  get_num_boosting_rounds(self) -> int
|      Gets the number of xgboost boosting rounds.
```

```
 |
 |  get_params(self, deep: bool = True) -> Dict[str, Any]
 |      Get parameters.
 |
 |  get_xgb_params(self) -> Dict[str, Any]
 |      Get xgboost specific parameters.
 |
 |  load_model(self, fname: Union[str, bytearray, os.PathLike]) -> None
 |      Load the model from a file or bytearray. Path to file can be local
 |      or as an URI.
 |
 |      The model is loaded from XGBoost format which is universal among the various
 |      XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as
 |      feature_names) will not be loaded when using binary format.  To save those
 |      attributes, use JSON/UBJ instead.  See :doc:`Model IO
</tutorials/saving_model>`
 |      for more info.
 |
 |      .. code-block:: python
 |
 |        model.load_model("model.json")
 |        # or
 |        model.load_model("model.ubj")
 |
 |      Parameters
 |      ----------
 |      fname :
 |          Input file name or memory buffer(see also save_raw)
 |
 |  save_model(self, fname: Union[str, os.PathLike]) -> None
 |      Save the model to a file.
 |
 |      The model is saved in an XGBoost internal format which is universal among the
 |      various XGBoost interfaces. Auxiliary attributes of the Python Booster object
 |      (such as feature_names) will not be saved when using binary format.  To save
 |      those attributes, use JSON/UBJ instead. See :doc:`Model IO
 |      </tutorials/saving_model>` for more info.
 |
 |      .. code-block:: python
 |
 |        model.save_model("model.json")
 |        # or
 |        model.save_model("model.ubj")
```

```
 |
 |      Parameters
 |      ----------
 |      fname : string or os.PathLike
 |          Output file name
 |
 |  set_params(self, **params: Any) -> 'XGBModel'
 |      Set the parameters of this estimator.  Modification of the sklearn method to
 |      allow unknown kwargs. This allows using the full range of xgboost
 |      parameters that are not defined as member variables in sklearn grid
 |      search.
 |
 |      Returns
 |      -------
 |      self
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from XGBModel:
 |
 |  best_iteration
 |      The best iteration obtained by early stopping.  This attribute is 0-based,
 |      for instance if the best iteration is the first round, then best_iteration is
0.
 |
 |  best_ntree_limit
 |
 |  best_score
 |      The best score obtained by early stopping.
 |
 |  coef_
 |      Coefficients property
 |
 |      .. note:: Coefficients are defined only for linear learners
 |
 |          Coefficients are only defined when the linear model is chosen as
 |          base learner (`booster=gblinear`). It is not defined for other base
 |          learner types, such as tree learners (`booster=gbtree`).
 |
 |      Returns
 |      -------
 |      coef_ : array of shape ``[n_features]`` or ``[n_classes, n_features]``
 |
 |  feature_importances_
```

```
 |      Feature importances property, return depends on `importance_type` parameter.
 |
 |      Returns
 |      -------
 |      feature_importances_ : array of shape ``[n_features]`` except for multi-class
 |      linear model, which returns an array with shape `(n_features, n_classes)`
 |
 |  feature_names_in_
 |      Names of features seen during :py:meth:`fit`.  Defined only when `X` has
feature
 |      names that are all strings.
 |
 |  intercept_
 |      Intercept (bias) property
 |
 |      .. note:: Intercept is defined only for linear learners
 |
 |          Intercept (bias) is only defined when the linear model is chosen as base
 |          learner (`booster=gblinear`). It is not defined for other base learner
types,
 |          such as tree learners (`booster=gbtree`).
 |
 |      Returns
 |      -------
 |      intercept_ : array of shape ``(1,)`` or ``[n_classes]``
 |
 |  n_features_in_
 |      Number of features seen during :py:meth:`fit`.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  ----------------------------------------------------------------------
 |  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)
|      Return 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 of shape (n_samples, n_features)
|          Test samples.
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          True labels for `X`.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights.
|
|      Returns
|      -------
|      score : float
|          Mean accuracy of ``self.predict(X)`` wrt. `y`.
```

```
train_target1=train_target
tar={'B':0 ,'M':1}
train_target1['diagnosis']=train_target1['diagnosis'].map(tar)
train_target1
```

	diagnosis
68	0
181	1
63	0
248	0
60	0

	diagnosis
...	...
71	0
106	0
270	0
435	1
102	0

455 rows × 1 columns

```
val_target1=val_target
tar={'B':0 ,'M':1}
val_target1['diagnosis']=val_target1['diagnosis'].map(tar)
val_target1
```

	diagnosis
204	0
70	1
131	1
431	0
540	0
...	...
486	0
75	1
249	0
238	0
265	1

114 rows × 1 columns

```
model.fit(train_df,train_target1)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.3, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=10, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=-1, num_parallel_tree=1, predictor='auto', random_state=42,
              reg_alpha=0, reg_lambda=1, ...)
```

```
model.score(train_df,train_target1)
```

```
1.0
```

```
model.score(val_df,val_target1)
```

```
0.956140350877193
```

```
importance_df = pd.DataFrame({
    'feature': train_df.columns,
    'importance': model.feature_importances_
}).sort_values('importance', ascending=False)
```

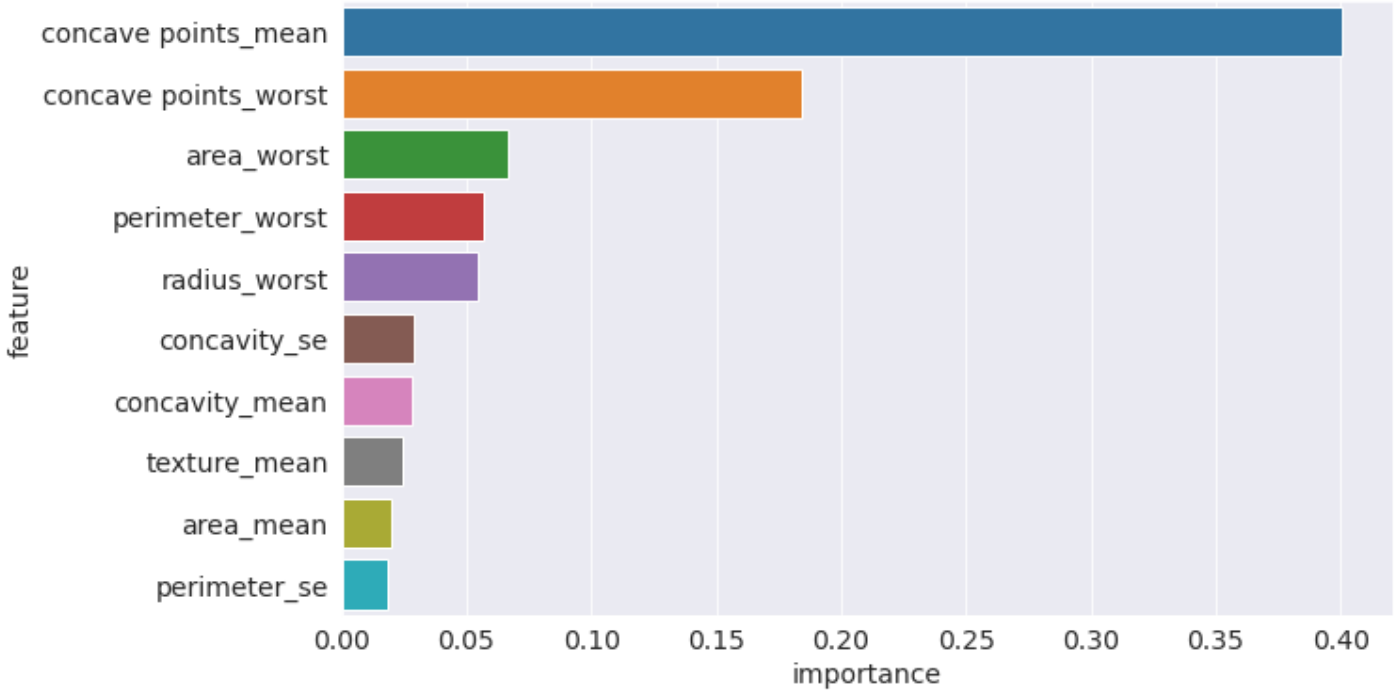
```
importance_df
```

	feature	importance
7	concave points_mean	0.400554
27	concave points_worst	0.184009
23	area_worst	0.066541
22	perimeter_worst	0.056906
20	radius_worst	0.054863
16	concavity_se	0.029083
6	concavity_mean	0.028313
1	texture_mean	0.023993
3	area_mean	0.019994
12	perimeter_se	0.018222
21	texture_worst	0.016144
26	concavity_worst	0.015475
2	perimeter_mean	0.013872
10	radius_se	0.013271
0	radius_mean	0.008075
4	smoothness_mean	0.006395
24	smoothness_worst	0.005447
5	compactness_mean	0.005364
11	texture_se	0.004580
13	area_se	0.004352
9	fractal_dimension_mean	0.004131
14	smoothness_se	0.003839
15	compactness_se	0.003570
28	symmetry_worst	0.002830
19	fractal_dimension_se	0.002348
25	compactness_worst	0.002215
8	symmetry_mean	0.002108
17	concave points_se	0.002091
18	symmetry_se	0.001414

	feature	importance
29	fractal_dimension_worst	0.000000

```
sns.barplot(data=importance_df.head(10),x='importance',y='feature')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f94aac5e990>
```



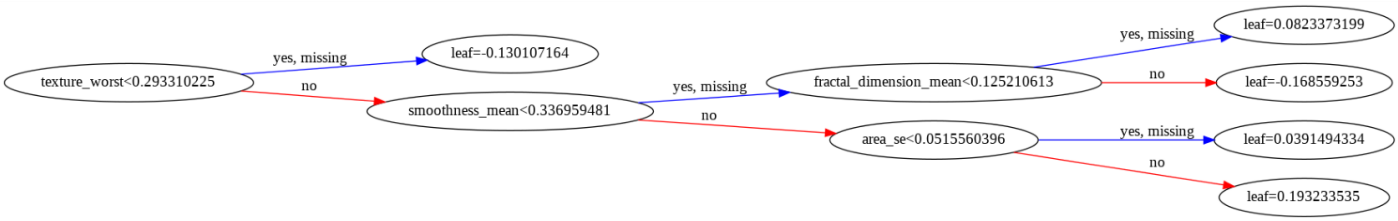

```
from xgboost import plot_tree
from matplotlib.pylab import rcParams
%matplotlib inline
rcParams['figure.figsize']=30,30
```

```
plot_tree(model,rankdir='LR')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f94aad33690>
```


```
plot_tree(model,rankdir='LR',num_trees=19)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f94b3d05750>
```




```
model=XGBClassifier(random_state=42,n_jobs=-1,n_estimators=1000,max_depth=10,learning_r
model.fit(train_df,train_target1)
model.score(train_df,train_target1),model.score(val_df,val_target1)
```

```
(1.0, 0.956140350877193)
```

From XGBOOST I got accuracy of 95% on validation data set.

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

```
'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'
```

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
model=GradientBoostingClassifier(random_state=42,n_estimators=200,max_depth=3,learning_
```

```
help(model)
```

```
Help on GradientBoostingClassifier in module sklearn.ensemble._gb object:

class GradientBoostingClassifier(sklearn.base.ClassifierMixin, BaseGradientBoosting)
 |  GradientBoostingClassifier(*, loss='deviance', learning_rate=0.1, n_estimators=100,
subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None,
random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False,
validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
 |
 |  Gradient Boosting for classification.
 |
 |  GB builds an additive model in a
 |  forward stage-wise fashion; it allows for the optimization of
 |  arbitrary differentiable loss functions. In each stage ``n_classes_``
 |  regression trees are fit on the negative gradient of the
```

```
  |  binomial or multinomial deviance loss function. Binary classification
  |  is a special case where only a single regression tree is induced.
  |
  |  Read more in the :ref:`User Guide <gradient_boosting>`.
  |
  |  Parameters
  |  ----------
  |  loss : {'deviance', 'exponential'}, default='deviance'
  |      The loss function to be optimized. 'deviance' refers to
  |      deviance (= logistic regression) for classification
  |      with probabilistic outputs. For loss 'exponential' gradient
  |      boosting recovers the AdaBoost algorithm.
  |
  |  learning_rate : float, default=0.1
  |      Learning rate shrinks the contribution of each tree by `learning_rate`.
  |      There is a trade-off between learning_rate and n_estimators.
  |
  |  n_estimators : int, default=100
  |      The number of boosting stages to perform. Gradient boosting
  |      is fairly robust to over-fitting so a large number usually
  |      results in better performance.
  |
  |  subsample : float, default=1.0
  |      The fraction of samples to be used for fitting the individual base
  |      learners. If smaller than 1.0 this results in Stochastic Gradient
  |      Boosting. `subsample` interacts with the parameter `n_estimators`.
  |      Choosing `subsample < 1.0` leads to a reduction of variance
  |      and an increase in bias.
  |
  |  criterion : {'friedman_mse', 'squared_error', 'mse', 'mae'},
default='friedman_mse'
  |      The function to measure the quality of a split. Supported criteria
  |      are 'friedman_mse' for the mean squared error with improvement
  |      score by Friedman, 'squared_error' for mean squared error, and 'mae'
  |      for the mean absolute error. The default value of 'friedman_mse' is
  |      generally the best as it can provide a better approximation in some
  |      cases.
  |
  |      .. versionadded:: 0.18
  |
  |      .. deprecated:: 0.24
  |          `criterion='mae'` is deprecated and will be removed in version
  |          1.1 (renaming of 0.26). Use `criterion='friedman_mse'` or
```

```
|          ``'squared_error'`` instead, as trees should use a squared error
|          criterion in Gradient Boosting.
|
|      .. deprecated:: 1.0
|          Criterion 'mse' was deprecated in v1.0 and will be removed in
|          version 1.2. Use `criterion='squared_error'` which is equivalent.
|
|  min_samples_split : int or float, default=2
|      The minimum number of samples required to split an internal node:
|
|      - If int, then consider `min_samples_split` as the minimum number.
|      - If float, then `min_samples_split` is a fraction and
|        `ceil(min_samples_split * n_samples)` are the minimum
|        number of samples for each split.
|
|      .. versionchanged:: 0.18
|         Added float values for fractions.
|
|  min_samples_leaf : int or float, default=1
|      The minimum number of samples required to be at a leaf node.
|      A split point at any depth will only be considered if it leaves at
|      least ``min_samples_leaf`` training samples in each of the left and
|      right branches.  This may have the effect of smoothing the model,
|      especially in regression.
|
|      - If int, then consider `min_samples_leaf` as the minimum number.
|      - If float, then `min_samples_leaf` is a fraction and
|        `ceil(min_samples_leaf * n_samples)` are the minimum
|        number of samples for each node.
|
|      .. versionchanged:: 0.18
|         Added float values for fractions.
|
|  min_weight_fraction_leaf : float, default=0.0
|      The minimum weighted fraction of the sum total of weights (of all
|      the input samples) required to be at a leaf node. Samples have
|      equal weight when sample_weight is not provided.
|
|  max_depth : int, default=3
|      The maximum depth of the individual regression estimators. The maximum
|      depth limits the number of nodes in the tree. Tune this parameter
|      for best performance; the best value depends on the interaction
|      of the input variables.
```

```
|
|  min_impurity_decrease : float, default=0.0
|      A node will be split if this split induces a decrease of the impurity
|      greater than or equal to this value.
|
|      The weighted impurity decrease equation is the following::
|
|          N_t / N * (impurity - N_t_R / N_t * right_impurity
|                              - N_t_L / N_t * left_impurity)
|
|      where ``N`` is the total number of samples, ``N_t`` is the number of
|      samples at the current node, ``N_t_L`` is the number of samples in the
|      left child, and ``N_t_R`` is the number of samples in the right child.
|
|      ``N``, ``N_t``, ``N_t_R`` and ``N_t_L`` all refer to the weighted sum,
|      if ``sample_weight`` is passed.
|
|      .. versionadded:: 0.19
|
|  init : estimator or 'zero', default=None
|      An estimator object that is used to compute the initial predictions.
|      ``init`` has to provide :meth:`fit` and :meth:`predict_proba`. If
|      'zero', the initial raw predictions are set to zero. By default, a
|      ``DummyEstimator`` predicting the classes priors is used.
|
|  random_state : int, RandomState instance or None, default=None
|      Controls the random seed given to each Tree estimator at each
|      boosting iteration.
|      In addition, it controls the random permutation of the features at
|      each split (see Notes for more details).
|      It also controls the random splitting of the training data to obtain a
|      validation set if `n_iter_no_change` is not None.
|      Pass an int for reproducible output across multiple function calls.
|      See :term:`Glossary <random_state>`.
|
|  max_features : {'auto', 'sqrt', 'log2'}, int or float, default=None
|      The number of features to consider when looking for the best split:
|
|      - If int, then consider `max_features` features at each split.
|      - If float, then `max_features` is a fraction and
|        `int(max_features * n_features)` features are considered at each
|        split.
|      - If 'auto', then `max_features=sqrt(n_features)`.
```

```
|      - If 'sqrt', then `max_features=sqrt(n_features)`.
|      - If 'log2', then `max_features=log2(n_features)`.
|      - If None, then `max_features=n_features`.
|
|      Choosing `max_features < n_features` leads to a reduction of variance
|      and an increase in bias.
|
|      Note: the search for a split does not stop until at least one
|      valid partition of the node samples is found, even if it requires to
|      effectively inspect more than ``max_features`` features.
|
|  verbose : int, default=0
|      Enable verbose output. If 1 then it prints progress and performance
|      once in a while (the more trees the lower the frequency). If greater
|      than 1 then it prints progress and performance for every tree.
|
|  max_leaf_nodes : int, default=None
|      Grow trees with ``max_leaf_nodes`` in best-first fashion.
|      Best nodes are defined as relative reduction in impurity.
|      If None then unlimited number of leaf nodes.
|
|  warm_start : bool, default=False
|      When set to ``True``, reuse the solution of the previous call to fit
|      and add more estimators to the ensemble, otherwise, just erase the
|      previous solution. See :term:`the Glossary <warm_start>`.
|
|  validation_fraction : float, default=0.1
|      The proportion of training data to set aside as validation set for
|      early stopping. Must be between 0 and 1.
|      Only used if ``n_iter_no_change`` is set to an integer.
|
|      .. versionadded:: 0.20
|
|  n_iter_no_change : int, default=None
|      ``n_iter_no_change`` is used to decide if early stopping will be used
|      to terminate training when validation score is not improving. By
|      default it is set to None to disable early stopping. If set to a
|      number, it will set aside ``validation_fraction`` size of the training
|      data as validation and terminate training when validation score is not
|      improving in all of the previous ``n_iter_no_change`` numbers of
|      iterations. The split is stratified.
|
|      .. versionadded:: 0.20
```

```
|
|  tol : float, default=1e-4
|      Tolerance for the early stopping. When the loss is not improving
|      by at least tol for ``n_iter_no_change`` iterations (if set to a
|      number), the training stops.
|
|      .. versionadded:: 0.20
|
|  ccp_alpha : non-negative float, default=0.0
|      Complexity parameter used for Minimal Cost-Complexity Pruning. The
|      subtree with the largest cost complexity that is smaller than
|      ``ccp_alpha`` will be chosen. By default, no pruning is performed. See
|      :ref:`minimal_cost_complexity_pruning` for details.
|
|      .. versionadded:: 0.22
|
|  Attributes
|  ----------
|  n_estimators_ : int
|      The number of estimators as selected by early stopping (if
|      ``n_iter_no_change`` is specified). Otherwise it is set to
|      ``n_estimators``.
|
|      .. versionadded:: 0.20
|
|  feature_importances_ : ndarray of shape (n_features,)
|      The impurity-based feature importances.
|      The higher, the more important the feature.
|      The importance of a feature is computed as the (normalized)
|      total reduction of the criterion brought by that feature.  It is also
|      known as the Gini importance.
|
|      Warning: impurity-based feature importances can be misleading for
|      high cardinality features (many unique values). See
|      :func:`sklearn.inspection.permutation_importance` as an alternative.
|
|  oob_improvement_ : ndarray of shape (n_estimators,)
|      The improvement in loss (= deviance) on the out-of-bag samples
|      relative to the previous iteration.
|      ``oob_improvement_[0]`` is the improvement in
|      loss of the first stage over the ``init`` estimator.
|      Only available if ``subsample < 1.0``
|
```

```
|  train_score_ : ndarray of shape (n_estimators,)
|      The i-th score ``train_score_[i]`` is the deviance (= loss) of the
|      model at iteration ``i`` on the in-bag sample.
|      If ``subsample == 1`` this is the deviance on the training data.
|
|  loss_ : LossFunction
|      The concrete ``LossFunction`` object.
|
|  init_ : estimator
|      The estimator that provides the initial predictions.
|      Set via the ``init`` argument or ``loss.init_estimator``.
|
|  estimators_ : ndarray of DecisionTreeRegressor of             shape (n_estimators,
``loss_.K``)
|      The collection of fitted sub-estimators. ``loss_.K`` is 1 for binary
|      classification, otherwise n_classes.
|
|  classes_ : ndarray of shape (n_classes,)
|      The classes labels.
|
|  n_features_ : int
|      The number of data features.
|
|      .. deprecated:: 1.0
|          Attribute `n_features_` was deprecated in version 1.0 and will be
|          removed in 1.2. Use `n_features_in_` instead.
|
|  n_features_in_ : int
|      Number of features seen during :term:`fit`.
|
|      .. versionadded:: 0.24
|
|  feature_names_in_ : ndarray of shape (`n_features_in_`,)
|      Names of features seen during :term:`fit`. Defined only when `X`
|      has feature names that are all strings.
|
|      .. versionadded:: 1.0
|
|  n_classes_ : int
|      The number of classes.
|
|  max_features_ : int
|      The inferred value of max_features.
```

```
|
|  See Also
|  --------
|  HistGradientBoostingClassifier : Histogram-based Gradient Boosting
|      Classification Tree.
|  sklearn.tree.DecisionTreeClassifier : A decision tree classifier.
|  RandomForestClassifier : A meta-estimator that fits a number of decision
|      tree classifiers on various sub-samples of the dataset and uses
|      averaging to improve the predictive accuracy and control over-fitting.
|  AdaBoostClassifier : A meta-estimator that begins by fitting a classifier
|      on the original dataset and then fits additional copies of the
|      classifier on the same dataset where the weights of incorrectly
|      classified instances are adjusted such that subsequent classifiers
|      focus more on difficult cases.
|
|  Notes
|  -----
|  The features are always randomly permuted at each split. Therefore,
|  the best found split may vary, even with the same training data and
|  ``max_features=n_features``, if the improvement of the criterion is
|  identical for several splits enumerated during the search of the best
|  split. To obtain a deterministic behaviour during fitting,
|  ``random_state`` has to be fixed.
|
|  References
|  ----------
|  J. Friedman, Greedy Function Approximation: A Gradient Boosting
|  Machine, The Annals of Statistics, Vol. 29, No. 5, 2001.
|
|  J. Friedman, Stochastic Gradient Boosting, 1999
|
|  T. Hastie, R. Tibshirani and J. Friedman.
|  Elements of Statistical Learning Ed. 2, Springer, 2009.
|
|  Examples
|  --------
|  The following example shows how to fit a gradient boosting classifier with
|  100 decision stumps as weak learners.
|
|  >>> from sklearn.datasets import make_hastie_10_2
|  >>> from sklearn.ensemble import GradientBoostingClassifier
|
|  >>> X, y = make_hastie_10_2(random_state=0)
```

```
 |  >>> X_train, X_test = X[:2000], X[2000:]
 |  >>> y_train, y_test = y[:2000], y[2000:]
 |
 |  >>> clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
 |  ...     max_depth=1, random_state=0).fit(X_train, y_train)
 |  >>> clf.score(X_test, y_test)
 |  0.913...
 |
 |  Method resolution order:
 |      GradientBoostingClassifier
 |      sklearn.base.ClassifierMixin
 |      BaseGradientBoosting
 |      sklearn.ensemble._base.BaseEnsemble
 |      sklearn.base.MetaEstimatorMixin
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, loss='deviance', learning_rate=0.1, n_estimators=100,
subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None,
random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False,
validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  decision_function(self, X)
 |      Compute the decision function of ``X``.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csr_matrix``.
 |
 |      Returns
 |      -------
 |      score : ndarray of shape (n_samples, n_classes) or (n_samples,)
 |          The decision function of the input samples, which corresponds to
 |          the raw values predicted from the trees of the ensemble . The
 |          order of the classes corresponds to that in the attribute
 |          :term:`classes_`. Regression and binary classification produce an
```

```
|          array of shape (n_samples,).
|
|  predict(self, X)
|      Predict class for X.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      Returns
|      -------
|      y : ndarray of shape (n_samples,)
|          The predicted values.
|
|  predict_log_proba(self, X)
|      Predict class log-probabilities for X.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      Returns
|      -------
|      p : ndarray of shape (n_samples, n_classes)
|          The class log-probabilities of the input samples. The order of the
|          classes corresponds to that in the attribute :term:`classes_`.
|
|      Raises
|      ------
|      AttributeError
|          If the ``loss`` does not support probabilities.
|
|  predict_proba(self, X)
|      Predict class probabilities for X.
|
|      Parameters
|      ----------
```

```
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      Returns
|      -------
|      p : ndarray of shape (n_samples, n_classes)
|          The class probabilities of the input samples. The order of the
|          classes corresponds to that in the attribute :term:`classes_`.
|
|      Raises
|      ------
|      AttributeError
|          If the ``loss`` does not support probabilities.
|
|  staged_decision_function(self, X)
|      Compute decision function of ``X`` for each iteration.
|
|      This method allows monitoring (i.e. determine error on testing set)
|      after each stage.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      Yields
|      ------
|      score : generator of ndarray of shape (n_samples, k)
|          The decision function of the input samples, which corresponds to
|          the raw values predicted from the trees of the ensemble . The
|          classes corresponds to that in the attribute :term:`classes_`.
|          Regression and binary classification are special cases with
|          ``k == 1``, otherwise ``k==n_classes``.
|
|  staged_predict(self, X)
|      Predict class at each stage for X.
|
|      This method allows monitoring (i.e. determine error on testing set)
|      after each stage.
```

```
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      Yields
|      -------
|      y : generator of ndarray of shape (n_samples,)
|          The predicted value of the input samples.
|
|  staged_predict_proba(self, X)
|      Predict class probabilities at each stage for X.
|
|      This method allows monitoring (i.e. determine error on testing set)
|      after each stage.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      Yields
|      ------
|      y : generator of ndarray of shape (n_samples,)
|          The predicted value of the input samples.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __abstractmethods__ = frozenset()
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.ClassifierMixin:
|
|  score(self, X, y, sample_weight=None)
|      Return 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 of shape (n_samples, n_features)
|          Test samples.
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          True labels for `X`.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights.
|
|      Returns
|      -------
|      score : float
|          Mean accuracy of ``self.predict(X)`` wrt. `y`.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.ClassifierMixin:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from BaseGradientBoosting:
|
|  apply(self, X)
|      Apply trees in the ensemble to X, return leaf indices.
|
|      .. versionadded:: 0.17
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, its dtype will be converted to
|          ``dtype=np.float32``. If a sparse matrix is provided, it will
|          be converted to a sparse ``csr_matrix``.
|
```

```
|      Returns
|      -------
|      X_leaves : array-like of shape (n_samples, n_estimators, n_classes)
|          For each datapoint x in X and for each tree in the ensemble,
|          return the index of the leaf x ends up in each estimator.
|          In the case of binary classification n_classes is 1.
|
|  fit(self, X, y, sample_weight=None, monitor=None)
|      Fit the gradient boosting model.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      y : array-like of shape (n_samples,)
|          Target values (strings or integers in classification, real numbers
|          in regression)
|          For classification, labels must correspond to classes.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights. If None, then samples are equally weighted. Splits
|          that would create child nodes with net zero or negative weight are
|          ignored while searching for a split in each node. In the case of
|          classification, splits are also ignored if they would result in any
|          single class carrying a negative weight in either child node.
|
|      monitor : callable, default=None
|          The monitor is called after each iteration with the current
|          iteration, a reference to the estimator and the local variables of
|          ``_fit_stages`` as keyword arguments ``callable(i, self,
|          locals())``. If the callable returns ``True`` the fitting procedure
|          is stopped. The monitor can be used for various things such as
|          computing held-out estimates, early stopping, model introspect, and
|          snapshoting.
|
|      Returns
|      -------
|      self : object
|          Fitted estimator.
|
```

```
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from BaseGradientBoosting:
 |
 |  feature_importances_
 |      The impurity-based feature importances.
 |
 |      The higher, the more important the feature.
 |      The importance of a feature is computed as the (normalized)
 |      total reduction of the criterion brought by that feature.  It is also
 |      known as the Gini importance.
 |
 |      Warning: impurity-based feature importances can be misleading for
 |      high cardinality features (many unique values). See
 |      :func:`sklearn.inspection.permutation_importance` as an alternative.
 |
 |      Returns
 |      -------
 |      feature_importances_ : ndarray of shape (n_features,)
 |          The values of this array sum to 1, unless all trees are single node
 |          trees consisting of only the root node, in which case it will be an
 |          array of zeros.
 |
 |  n_features_
 |      DEPRECATED: Attribute `n_features_` was deprecated in version 1.0 and will be
removed in 1.2. Use `n_features_in_` instead.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.ensemble._base.BaseEnsemble:
 |
 |  __getitem__(self, index)
 |      Return the index'th estimator in the ensemble.
 |
 |  __iter__(self)
 |      Return iterator over estimators in the ensemble.
 |
 |  __len__(self)
 |      Return the number of estimators in the ensemble.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from sklearn.ensemble._base.BaseEnsemble:
 |
 |  __annotations__ = {'_required_parameters': typing.List[str]}
 |
```

```
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|
|  __repr__(self, N_CHAR_MAX=700)
|      Return repr(self).
|
|  __setstate__(self, state)
|
|  get_params(self, deep=True)
|      Get parameters for this estimator.
|
|      Parameters
|      ----------
|      deep : bool, default=True
|          If True, will return the parameters for this estimator and
|          contained subobjects that are estimators.
|
|      Returns
|      -------
|      params : dict
|          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 :class:`~sklearn.pipeline.Pipeline`). The latter have
|      parameters of the form ``<component>__<parameter>`` so that it's
|      possible to update each component of a nested object.
|
|      Parameters
|      ----------
|      **params : dict
|          Estimator parameters.
|
|      Returns
|      -------
|      self : estimator instance
|          Estimator instance.
```

```
model.fit(train_df,train_target)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_gb.py:494:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().


GradientBoostingClassifier(n_estimators=200, random_state=42)
```

```
model.score(train_df,train_target)
```

```
1.0
```

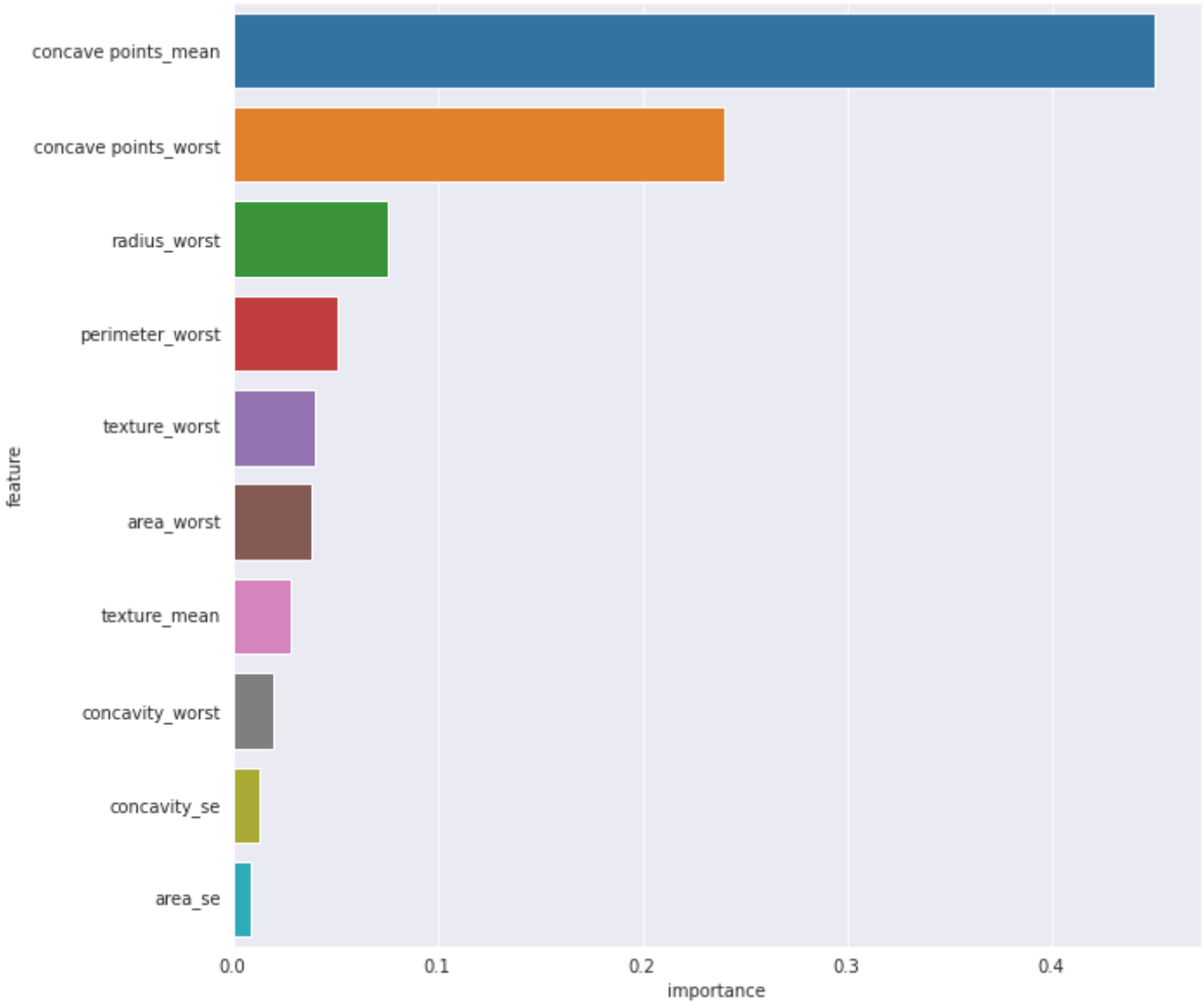
```
model.score(val_df,val_target)
```

```
0.956140350877193
```

```
importance_df = pd.DataFrame({
    'feature': train_df.columns,
    'importance': model.feature_importances_
}).sort_values('importance', ascending=False)
```

```
rcParams['figure.figsize']=10,10
sns.barplot(data=importance_df.head(10),x='importance',y='feature')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f94aaa6df10>
```




```
from sklearn.tree import plot_tree
```

From Gradient Boosting I got accuracy of 95% on validation data set.

Finally I Selected SVM

```
classifier=SVC(kernel='poly',random_state=42)
```

```
classifier.fit(train_df,train_target)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().


SVC(kernel='poly', random_state=42)
```

```
classifier.score(train_df,train_target)
```

```
0.989010989010989
```

```
classifier.score(val_df,val_target)
```

```
0.9824561403508771
```

```
val_pred=classifier.predict(val_df)
```

```
pred=pd.DataFrame(val_pred)
pred
```

	0
0	0
1	1
2	1
3	0
4	0
...	...
109	0
110	1
111	0
112	0
113	1

114 rows × 1 columns

```
tar={0: 'B' ,1: 'M'}
pred[0]=pred[0].map(tar)
pred
```

	0
0	B
1	M
2	M
3	B
4	B
...	...
109	B
110	M
111	B
112	B

	0
113	M

114 rows × 1 columns

```
pred.to_csv('val_prediction.csv')
```

```
FileLink('val_prediction.csv')
```

val_prediction.csv

At the End I got Accuracy of 98% Accuracy on both Test and Validation Data set.

Saving The Model

```
!pip install joblib
import joblib
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (1.1.0)
```

```
cancer = {
    'model': classifier ,
    'imputer': imputer,
    'scaler': scaler,
    'encoder': encoder,
    'input_cols': list(train_df.columns),
    'target_col': ['diagnosis'],
    'numeric_cols': list(train_df.columns),
    'categorical_cols': [],
    'encoded_cols': encoded_cols
}
```

```
joblib.dump(cancer,'cancer.joblib')
```

```
['cancer.joblib']
```

```
FileLink('cancer.joblib')
```

cancer.joblib

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
```

```
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

```
'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'
```

Summary and References

Finally I got Accuracy of 98% on both Test and Validation Data set.

You can Download THE Dataset from Kaggle:

https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic
```

```
'https://jovian.ai/btech60309-19/breast-cancer-wisconsin-diagnostic'
```

```
# Loading Mod
cancer1=joblib.load('cancer.joblib')
```

```

```