Ekin Beykın No:121520052

Sınıflandırma Problemleri

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

In [14]:
```
data=pd.read_csv("C:/Users/EKİN/OneDrive/Masaüstü/smoking.csv")

data = data.replace({'gender': {'F': 1, 'M': 0}, 'tartar': {'Y':1, 'N':0},
                     'oral': {'Y':1, 'N':0}})

data.head()
```

Out[14]:

	ID	gender	age	height(cm)	weight(kg)	waist(cm)	eyesight(left)	eyesight(right)	hear
0	0	1	40	155	60	81.3	1.2	1.0	
1	1	1	40	160	60	81.0	0.8	0.6	
2	2	0	55	170	60	80.0	0.8	0.8	
3	3	0	40	165	70	88.0	1.5	1.5	
4	4	1	40	155	60	86.0	1.0	1.0	

5 rows × 27 columns

In [271…
```
#Bağımlı değişkenimizi 0 ve 1 değerleri almış olan smoking olarak seçiyoruz.
#durumunun elimizdeki cinsiyet, yaş, boy, kilo gibi birçok bağımsız değişker

#Öncelikle veride eksik gozlem var mı yok mu bunu araştıralım.
```

In [16]:
```
data.isnull().any().sum()
```

Out[16]:
```
0
```

In [17]:
```
#Veride eksik gözlem bulunmuyor.
```

In [18]:
```
#Verimiz hakkında genel bilgileri inceleyelim.
```

In [19]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2499 entries, 0 to 2498
Data columns (total 27 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2499 non-null   int64  
 1   gender               2499 non-null   int64  
 2   age                  2499 non-null   int64  
 3   height(cm)           2499 non-null   int64  
 4   weight(kg)           2499 non-null   int64  
 5   waist(cm)            2499 non-null   float64
 6   eyesight(left)       2499 non-null   float64
 7   eyesight(right)      2499 non-null   float64
 8   hearing(left)        2499 non-null   float64
 9   hearing(right)       2499 non-null   float64
 10  systolic             2499 non-null   float64
 11  relaxation           2499 non-null   float64
 12  fasting blood sugar  2499 non-null   float64
 13  Cholesterol          2499 non-null   float64
 14  triglyceride         2499 non-null   float64
 15  HDL                  2499 non-null   float64
 16  LDL                  2499 non-null   float64
 17  hemoglobin           2499 non-null   float64
 18  Urine protein        2499 non-null   float64
 19  serum creatinine     2499 non-null   float64
 20  AST                  2499 non-null   float64
 21  ALT                  2499 non-null   float64
 22  Gtp                  2499 non-null   float64
 23  oral                 2499 non-null   int64  
 24  dental caries        2499 non-null   int64  
 25  tartar               2499 non-null   int64  
 26  smoking              2499 non-null   int64  
dtypes: float64(18), int64(9)
memory usage: 527.3 KB
```

In [20]:
```
#Verimizden "ID" sütununu çıkaralım.
```

In [21]:
```
smoking = data.drop("ID",  axis = 1)
smoking.info()
```

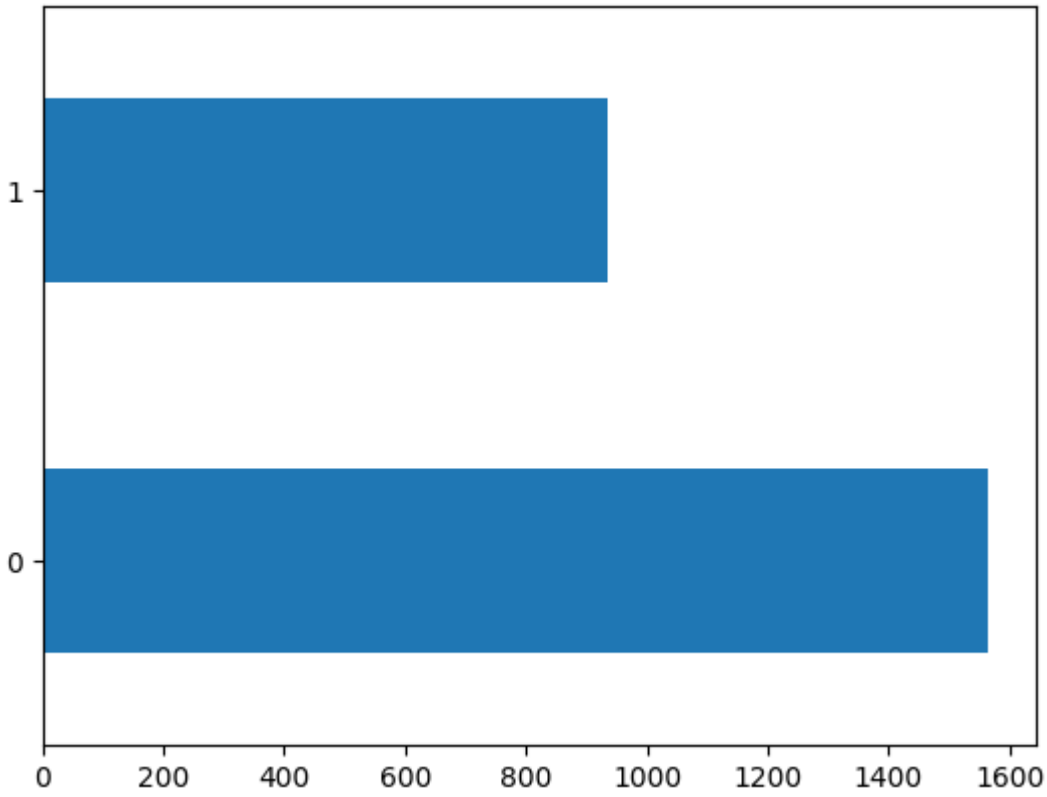
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2499 entries, 0 to 2498
Data columns (total 26 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   gender               2499 non-null   int64  
 1   age                  2499 non-null   int64  
 2   height(cm)           2499 non-null   int64  
 3   weight(kg)           2499 non-null   int64  
 4   waist(cm)            2499 non-null   float64
 5   eyesight(left)       2499 non-null   float64
 6   eyesight(right)      2499 non-null   float64
 7   hearing(left)        2499 non-null   float64
 8   hearing(right)       2499 non-null   float64
 9   systolic             2499 non-null   float64
 10  relaxation           2499 non-null   float64
 11  fasting blood sugar  2499 non-null   float64
 12  Cholesterol          2499 non-null   float64
 13  triglyceride         2499 non-null   float64
 14  HDL                  2499 non-null   float64
 15  LDL                  2499 non-null   float64
 16  hemoglobin           2499 non-null   float64
 17  Urine protein        2499 non-null   float64
 18  serum creatinine     2499 non-null   float64
 19  AST                  2499 non-null   float64
 20  ALT                  2499 non-null   float64
 21  Gtp                  2499 non-null   float64
 22  oral                 2499 non-null   int64  
 23  dental caries        2499 non-null   int64  
 24  tartar               2499 non-null   int64  
 25  smoking              2499 non-null   int64  
dtypes: float64(18), int64(8)
memory usage: 507.7 KB
```

In [23]:
```
smoking["smoking"].value_counts()
```

Out[23]:
```
0    1564
1     935
Name: smoking, dtype: int64
```

In [24]:
```
#1564 sigara içen 935 içmeyen kişi olduğunu gördük, bunu bir de grafikle gös
```

In [272…
```
smoking["smoking"].value_counts().plot.barh();
```



In [26]:
```
smoking.describe().T
```

Out[26]:

	count	mean	std	min	25%	50%	75%	max
gender	2499.0	0.363745	0.481173	0.0	0.0	0.0	1.0	1.0
age	2499.0	44.387755	11.765885	20.0	40.0	40.0	55.0	80.0
height(cm)	2499.0	164.631853	9.250566	135.0	160.0	165.0	170.0	190.0
weight(kg)	2499.0	65.758303	12.781467	35.0	55.0	65.0	75.0	120.0
waist(cm)	2499.0	81.857143	9.170951	57.2	76.0	82.0	88.0	114.0
eyesight(left)	2499.0	1.009884	0.406289	0.1	0.8	1.0	1.2	9.9
eyesight(right)	2499.0	1.002081	0.450562	0.1	0.8	1.0	1.2	9.9
hearing(left)	2499.0	1.026010	0.159198	1.0	1.0	1.0	1.0	2.0
hearing(right)	2499.0	1.028011	0.165038	1.0	1.0	1.0	1.0	2.0
systolic	2499.0	121.214486	13.523651	82.0	111.0	120.0	130.0	184.0
relaxation	2499.0	75.782313	9.610417	50.0	70.0	76.0	81.0	120.0
fasting blood sugar	2499.0	99.008403	21.622728	56.0	89.0	96.0	103.0	423.0
Cholesterol	2499.0	197.543417	35.628150	96.0	174.0	196.0	220.0	373.0
triglyceride	2499.0	127.724690	70.197305	21.0	75.0	110.0	163.5	399.0
HDL	2499.0	57.365346	14.348396	24.0	47.0	56.0	66.0	128.0
LDL	2499.0	114.793918	32.752663	9.0	94.0	114.0	136.0	272.0
hemoglobin	2499.0	14.622609	1.562761	7.1	13.6	14.8	15.8	18.8
Urine protein	2499.0	1.091236	0.414508	1.0	1.0	1.0	1.0	5.0
serum creatinine	2499.0	0.892357	0.287784	0.1	0.7	0.9	1.0	10.3
AST	2499.0	25.708283	14.647422	6.0	19.0	23.0	28.0	341.0
ALT	2499.0	26.051220	18.392172	1.0	15.0	21.0	30.0	252.0
Gtp	2499.0	39.799920	49.056804	6.0	17.0	25.0	43.5	836.0
oral	2499.0	1.000000	0.000000	1.0	1.0	1.0	1.0	1.0
dental caries	2499.0	0.210084	0.407450	0.0	0.0	0.0	0.0	1.0
tartar	2499.0	0.553822	0.497194	0.0	0.0	1.0	1.0	1.0
smoking	2499.0	0.374150	0.483999	0.0	0.0	0.0	1.0	1.0

In [28]:
```
y = smoking["smoking"]
x = smoking.drop("smoking" , axis = 1)
```

Lojistik Regresyon

In [29]:
```
import statsmodels.api as sm
import statsmodels.formula as smf
```

In [30]:
```
log_model= sm.Logit(y,x).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.471070
         Iterations 7
```

In [31]:
```
log_model.summary()
```

Out[31]:

Logit Regression Results

Dep. Variable:	smoking	**No. Observations:**	2499
Model:	Logit	**Df Residuals:**	2474
Method:	MLE	**Df Model:**	24
Date:	Tue, 02 May 2023	**Pseudo R-squ.:**	0.2875
Time:	00:53:25	**Log-Likelihood:**	-1177.2
converged:	True	**LL-Null:**	-1652.2
Covariance Type:	nonrobust	**LLR p-value:**	3.823e-185

	coef	std err	z	P>|z|	[0.025	0.975]
gender	-2.7312	0.229	-11.903	0.000	-3.181	-2.281
age	-0.0044	0.005	-0.834	0.404	-0.015	0.006
height(cm)	0.0163	0.011	1.545	0.122	-0.004	0.037
weight(kg)	-0.0115	0.010	-1.099	0.272	-0.032	0.009
waist(cm)	-0.0108	0.012	-0.876	0.381	-0.035	0.013
eyesight(left)	0.1767	0.153	1.157	0.247	-0.123	0.476
eyesight(right)	-0.2292	0.147	-1.560	0.119	-0.517	0.059
hearing(left)	-0.3082	0.425	-0.725	0.468	-1.141	0.525
hearing(right)	0.0081	0.395	0.021	0.984	-0.767	0.783
systolic	-0.0171	0.006	-2.852	0.004	-0.029	-0.005
relaxation	0.0275	0.008	3.299	0.001	0.011	0.044
fasting blood sugar	0.0007	0.002	0.288	0.773	-0.004	0.005
Cholesterol	-0.0298	0.016	-1.862	0.063	-0.061	0.002
triglyceride	0.0092	0.003	2.809	0.005	0.003	0.016
HDL	0.0195	0.016	1.195	0.232	-0.013	0.052
LDL	0.0272	0.016	1.700	0.089	-0.004	0.059
hemoglobin	0.1106	0.051	2.183	0.029	0.011	0.210
Urine protein	0.0071	0.124	0.057	0.954	-0.237	0.251
serum creatinine	-0.2490	0.213	-1.169	0.243	-0.667	0.169
AST	0.0011	0.005	0.204	0.839	-0.009	0.012
ALT	-0.0121	0.004	-2.785	0.005	-0.021	-0.004
Gtp	0.0125	0.002	6.889	0.000	0.009	0.016
oral	-1.9217	2.091	-0.919	0.358	-6.020	2.177
dental caries	0.2782	0.123	2.259	0.024	0.037	0.520
rtar	0.4130	0.105	3.948	0.000	0.208	0.618

In [32]:
```
from sklearn.linear_model import LogisticRegression
log_model = LogisticRegression(solver = "liblinear").fit(x,y)
```

In [33]:
```
log_model.intercept_
```

Out[33]:
```
array([-0.40698647])
```

In [34]:
```
log_model.coef_
```

Out[34]:
```
array([[-2.69130737e+00, -4.96513345e-03,  1.13361252e-02,
        -7.60851818e-03, -1.45355787e-02,  1.58037584e-01,
        -2.21346239e-01, -3.00936033e-01, -4.56971432e-02,
        -1.75759978e-02,  2.72088395e-02,  6.07371027e-04,
        -3.02421869e-02,  9.20739154e-03,  1.91445223e-02,
         2.74580410e-02,  1.09751438e-01, -1.50990395e-03,
        -2.22650453e-01,  1.05580439e-03, -1.22111250e-02,
         1.26940259e-02, -4.06986468e-01,  2.71404057e-01,
         4.02349730e-01]])
```

In [38]:
```
#confusion matrix ve accuracy skor değerlerine, kurduğumuz modelin ne kadar
```

In [39]:
```
fits = log_model.predict(x)
```

In [40]:
```
from sklearn.metrics import accuracy_score , confusion_matrix
```

In [42]:
```
confusion_matrix(y,fits)
```

Out[42]:
```
array([[1217,  347],
       [ 273,  662]], dtype=int64)
```

In [43]:
```
    #0 olup 0 olarak doğru sınıflandırılmış 1217 gözlem var.
    #0 olduğu halde 1 olarak sınıflandırılmış 347 gözlem var
    #1 olduğu halde 0 olarap sınıflandırılmış 273 gözlem var.
    #1 olup 1 olarak doğru sınıflandırılmış 662 tane gözlem var.
```

In [ ]:
```
#Modelin ne kadar doğru sınıflandırma yapabileceğini görmek istersek:
```

In [45]:
```
accuracy_score(y, fits)
```

Out[45]:
```
0.7519007603041217
```

In [46]:
```
#Olasılıkları inceleyelim.
```

In [149…
```
log_model.predict_proba(x)
```

Out[149]:
```
array([[0.95413205, 0.04586795],
       [0.94140845, 0.05859155],
       [0.50443569, 0.49556431],
       ...,
       [0.41472914, 0.58527086],
       [0.35146155, 0.64853845],
       [0.5024588 , 0.4975412 ]])
```

In [150…
```
#AUC değerini hesaplamak ve grafiğini görmek istersek:
```

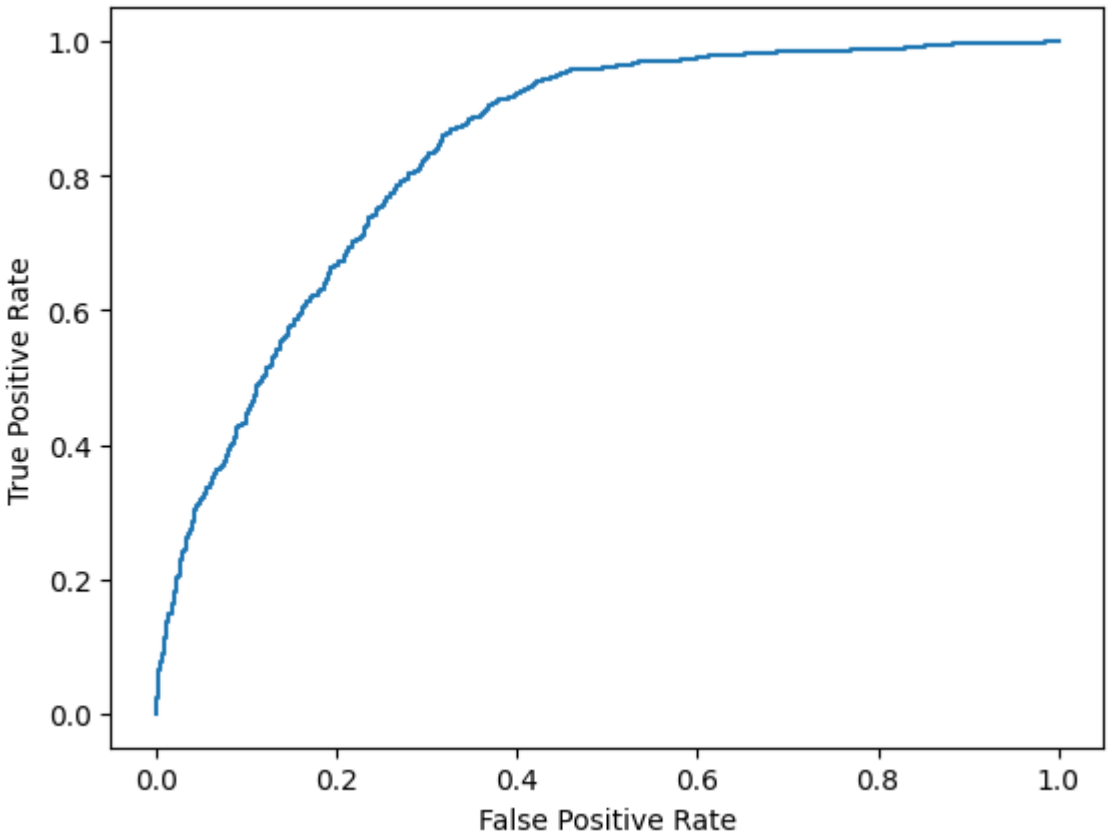
In [151…
```
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as mt
```

In [152…
```
auc_degerleri =roc_auc_score(y, fits)
```

In [153…
```
fpr , tpr, treshold_val = roc_curve(y,log_model.predict_proba(x)[:,1] )
```

In [154…
```
mt.figure()
mt.plot(fpr , tpr)
mt.xlabel("False Positive Rate")
mt.ylabel("True Positive Rate")
```

Out[154]: Text(0, 0.5, 'True Positive Rate')



In [155…
```
#%25-%75 olarak ikiye ayırdığımız model üzerinden sınıflandırma yapmayı dene
```

In [156…
```
from sklearn.model_selection import train_test_split , cross_val_score
```

In [157…
```
x_train , x_test, y_train , y_test = train_test_split(x,y, test_size=25, ran
```

In [158…
```
log_model = LogisticRegression(solver = "liblinear").fit(x_train,y_train)
```

In [159…
```
preds = log_model.predict(x_test)
accuracy_score(y_test, preds)
```

Out[159]: 0.8

In [273…
```
#Modelimiz %80 doğruluk payına ulaşmış bir test yapabiliyor.
#Lojistik Regresyondan daha iyi bir sonuç vermiş.
```

In [270…
```
#Cross validation'ı hesaplayalım.
cvLM = cross_val_score(log_model, x_test, y_test, cv=10)
```

In [162…
```
cvLM
```

Out[162]:
```
array([0.33333333, 0.66666667, 0.66666667, 1.        , 0.66666667,
       0.5       , 1.        , 0.5       , 1.        , 1.        ])
```

In [163…
```
cvLM.mean()
```

Out[163]: 0.7333333333333333

Sınıflandırma Problemi için Naive Bayes Yöntemi

In [164…
```
from sklearn.naive_bayes import GaussianNB
```

In [165…
```
naive_model = GaussianNB().fit(x_train, y_train)
preds_naive = naive_model.predict(x_test)
```

In [166…
```
#Olasılıkları görelim.
```

In [167…
```
probs = naive_model.predict_proba(x_test)
```

In [168…
```
accuracy_score(y_test, preds_naive)
```

Out[168]: 0.76

In [169…
```
# %76 doğruluk payıyla lojistik regresyon daha iyi sonuc verememiş.
```

In [170…
```
cvNB = cross_val_score(naive_model, x_test, y_test , cv =10)
```

In [171…
```
cvNB
```

Out[171]:
```
array([1.        , 0.66666667, 1.        , 1.        , 0.66666667,
       0.5       , 1.        , 1.        , 0.        , 1.        ])
```

In [172…
```
cvNB.mean()
```

Out[172]: 0.7833333333333333

KNN Yöntemi

In [173…
```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier().fit(x_train, y_train)
```

```
preds_knn = knn_model.predict(x_test)
```

In [174…
```
accuracy_score(y_test,preds_knn )
```

Out[174]: 0.8

In [175…
```
#Optimize etmediğimiz haliyle %80 doğruluk payı olan bir test elde ettik.
#Modelimizi optimize ederek tekrar deneyelim.
#Önce hiperparametrelere bakalım:
```

In [176…
```
?knn_model
```

```
Type:        KNeighborsClassifier
String form: KNeighborsClassifier()
File:        c:\users\ekİn\appdata\local\programs\python\python311\lib\site-p
ackages\sklearn\neighbors\_classification.py
Docstring:
Classifier implementing the k-nearest neighbors vote.

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

Parameters
----------
n_neighbors : int, default=5
    Number of neighbors to use by default for :meth:`kneighbors` queries.

weights : {'uniform', 'distance'}, callable or None, default='uniform'
    Weight function used in prediction.  Possible values:

    - 'uniform' : uniform weights.  All points in each neighborhood
      are weighted equally.
    - 'distance' : weight points by the inverse of their distance.
      in this case, closer neighbors of a query point will have a
      greater influence than neighbors which are further away.
    - [callable] : a user-defined function which accepts an
      array of distances, and returns an array of the same shape
      containing the weights.

algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'
    Algorithm used to compute the nearest neighbors:

    - 'ball_tree' will use :class:`BallTree`
    - 'kd_tree' will use :class:`KDTree`
    - 'brute' will use a brute-force search.
    - 'auto' will attempt to decide the most appropriate algorithm
      based on the values passed to :meth:`fit` method.

    Note: fitting on sparse input will override the setting of
    this parameter, using brute force.

leaf_size : int, default=30
    Leaf size passed to BallTree or KDTree.  This can affect the
    speed of the construction and query, as well as the memory
    required to store the tree.  The optimal value depends on the
    nature of the problem.

p : int, default=2
    Power parameter for the Minkowski metric. When p = 1, this is
    equivalent to using manhattan_distance (l1), and euclidean_distance
    (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metric : str or callable, default='minkowski'
    Metric to use for distance computation. Default is "minkowski", which
    results in the standard Euclidean distance when p = 2. See the
    documentation of `scipy.spatial.distance
    <https://docs.scipy.org/doc/scipy/reference/spatial.distance.html>`_ and
    the metrics listed in
    :class:`~sklearn.metrics.pairwise.distance_metrics` for valid metric
```

```
        values.

        If metric is "precomputed", X is assumed to be a distance matrix and
        must be square during fit. X may be a :term:`sparse graph`, in which
        case only "nonzero" elements may be considered neighbors.

        If metric is a callable function, it takes two arrays representing 1D
        vectors as inputs and must return one value indicating the distance
        between those vectors. This works for Scipy's metrics, but is less
        efficient than passing the metric name as a string.

    metric_params : dict, default=None
        Additional keyword arguments for the metric function.

    n_jobs : int, default=None
        The number of parallel jobs to run for neighbors search.
        ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
        ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
        for more details.
        Doesn't affect :meth:`fit` method.

    Attributes
    ----------
    classes_ : array of shape (n_classes,)
        Class labels known to the classifier

    effective_metric_ : str or callble
        The distance metric used. It will be same as the `metric` parameter
        or a synonym of it, e.g. 'euclidean' if the `metric` parameter set to
        'minkowski' and `p` parameter set to 2.

    effective_metric_params_ : dict
        Additional keyword arguments for the metric function. For most metrics
        will be same with `metric_params` parameter, but may also contain the
        `p` parameter value if the `effective_metric_` attribute is set to
        'minkowski'.

    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_samples_fit_ : int
        Number of samples in the fitted data.

    outputs_2d_ : bool
        False when `y`'s shape is (n_samples, ) or (n_samples, 1) during fit
        otherwise True.
```

```
--------
RadiusNeighborsClassifier: Classifier based on neighbors within a fixed radiu
s.
KNeighborsRegressor: Regression based on k-nearest neighbors.
RadiusNeighborsRegressor: Regression based on neighbors within a fixed radiu
s.
NearestNeighbors: Unsupervised learner for implementing neighbor searches.

Notes
-----
See :ref:`Nearest Neighbors <neighbors>` in the online documentation
for a discussion of the choice of ``algorithm`` and ``leaf_size``.

.. warning::

   Regarding the Nearest Neighbors algorithms, if it is found that two
   neighbors, neighbor `k+1` and `k`, have identical distances
   but different labels, the results will depend on the ordering of the
   training data.

https://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm

Examples
--------
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[0.666... 0.333...]]
```

In [177…
```
knn_params = {"n_neighbors":np.arange(1,5), "leaf_size":np.arange(1,5), "p"
```

In [178…
```
from sklearn.model_selection import GridSearchCV
```

In [179…
```
knn_mod = KNeighborsClassifier()
knn_cv = GridSearchCV(knn_mod, knn_params, cv = 10 , verbose=2).fit(x_train,
```

```
Fitting 10 folds for each of 32 candidates, totalling 320 fits
[CV] END ....................leaf_size=1, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=1; total time=
0.0s
...............leaf_size=1, n_neighbors=2, p=1; total time=
```

```
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=2; total time=
```

```
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=1, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=1; total time=
```

```
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=2; total time=
```

```
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=3, p=2; total time=
```

```
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=2, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=1; total time=
0.0s
................leaf_size=3, n_neighbors=1, p=1; total time=
```

```
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=2; total time=
0.0s
...............leaf_size=3, n_neighbors=2, p=2; total time=
```

```
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=1; total time=
```

```
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=3, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=2; total time=
0.0s
................leaf_size=4, n_neighbors=1, p=2; total time=
```

```
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=1, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=2, p=2; total time=
```

```
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=3, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=1; total time=
0.0s
...............leaf_size=4, n_neighbors=4, p=1; total time=
```

```
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=1; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=2; total time=
0.0s
[CV] END ....................leaf_size=4, n_neighbors=4, p=2; total time=
0.0s
```

In [180…

```
knn_cv.best_params_
```

Out[180]:

```
{'leaf_size': 1, 'n_neighbors': 4, 'p': 1}
```

In [181…

```
knn_model_best = KNeighborsClassifier(n_neighbors=4, leaf_size=1).fit(x_trai
preds_knn_best = knn_model.predict(x_test)
```

In [182…

```
accuracy_score(y_test, preds_knn_best)
```

Out[182]:

```
0.8
```

In [183…

```
#Optimize ettikten sonra da önceki modelimizle aynı oranı aldık.
```

Sınıflandırma için SVC-SVM

In [184…

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

In [185…

```
svc_model = SVC(kernel = "linear").fit(x_train,y_train)
svc_preds = svc_model.predict(x_test)
```

In [186…

```
accuracy_score(y_test, svc_preds)
```

Out[186]:

```
0.76
```

In [190…

```
#Doğruluk payı %76 olan bir teste ulaştık.
#Bunu optimize edelim.
```

In [187…
```
svc_params = {"C":np.arange(1,5)}
```

In [188…
```
svc_mod = SVC(kernel="linear")
svc_cv = GridSearchCV(svc_mod, svc_params, cv = 10).fit(x_train, y_train)
```

In [189…
```
svc_cv.best_params_
```

Out[189]:
```
{'C': 3}
```

In [191…
```
#Buradan c=3 değerini alıyoruz.
```

In [192…
```
svc_model_best = SVC(kernel= "linear" ,C=3).fit(x_train, y_train)
```

In [193…
```
preds_svc_best = svc_model_best.predict(x_test)
accuracy_score(y_test, preds_svc_best)
```

Out[193]:
```
0.72
```

In [274…
```
# Modeli optimize ettikten sonra %76'dan %72'e bir düşüş yaşandı.
#Bunun nedeni daha dar bir parametre kullanmamız.
```

Doğrusal Olmayan SVC

In [195…
```
svc_model = SVC(kernel = "rbf").fit(x_train,y_train)
svc_preds = svc_model.predict(x_test)
accuracy_score(y_test, svc_preds)
```

Out[195]:
```
0.72
```

In [196…
```
# %72 doğruluk payı modelimizin başlangıç değeri.
#Modelimizi optimize edelim.
```

In [197…
```
svc_params = {"C":np.arange(1,15) , "gamma": [0.0001, 0.01, 1]}
```

In [198…
```
svc_mod = SVC(kernel="rbf")
svc_cv = GridSearchCV(svc_mod, svc_params, cv = 10).fit(x_train, y_train)
```

In [199…
```
svc_cv.best_params_
```

Out[199]:
```
{'C': 8, 'gamma': 0.0001}
```

In [200…
```
svc_model_best = SVC(kernel= "rbf" ,C= 8, gamma = 0.0001).fit(x_train, y_tra
```

In [201…
```
preds_svc_best = svc_model_best.predict(x_test)
accuracy_score(y_test, preds_svc_best)
```

Out[201]:
```
0.72
```

In [202…
```
#Optimize ettikten sonra modelin doğruluk payı değişmemiştir.
```

Sınıflandırma İçin Yapay Sinir Ağları

In [203…
```
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
```

In [204…
```
scaler = StandardScaler()
scaler.fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

In [205…
```
mlp_model = MLPClassifier().fit(x_train_scaled,y_train)
preds_mlp = mlp_model.predict(x_test_scaled)
```

```
C:\Users\EKİN\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\neural_network\_multilayer_perceptron.py:679: ConvergenceWarning: Stochast
ic Optimizer: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  warnings.warn(
```

In [206…
```
accuracy_score(y_test, preds_mlp)
```

Out[206]: 0.6

In [208…
```
#Modelimiz için başlangıç doğruluk payı %60 olarak hesaplandı.
#Bu değeri arttırmak için modelimizi optimize edelim.
```

In [209…
```
?mlp_model
```

Type: MLPClassifier
String form: MLPClassifier()
File: c:\users\ekİn\appdata\local\programs\python\python311\lib\site-packages\sklearn\neural_network_multilayer_perceptron.py
Docstring:

```
Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic
gradient descent.

.. versionadded:: 0.18

Parameters
----------
hidden_layer_sizes : array-like of shape(n_layers - 2,), default=(100,)
    The ith element represents the number of neurons in the ith
    hidden layer.

activation : {'identity', 'logistic', 'tanh', 'relu'}, default='relu'
    Activation function for the hidden layer.

    - 'identity', no-op activation, useful to implement linear bottleneck,
      returns f(x) = x

    - 'logistic', the logistic sigmoid function,
      returns f(x) = 1 / (1 + exp(-x)).

    - 'tanh', the hyperbolic tan function,
      returns f(x) = tanh(x).

    - 'relu', the rectified linear unit function,
      returns f(x) = max(0, x)

solver : {'lbfgs', 'sgd', 'adam'}, default='adam'
    The solver for weight optimization.

    - 'lbfgs' is an optimizer in the family of quasi-Newton methods.

    - 'sgd' refers to stochastic gradient descent.

    - 'adam' refers to a stochastic gradient-based optimizer proposed
      by Kingma, Diederik, and Jimmy Ba

    Note: The default solver 'adam' works pretty well on relatively
    large datasets (with thousands of training samples or more) in terms of
    both training time and validation score.
    For small datasets, however, 'lbfgs' can converge faster and perform
    better.

alpha : float, default=0.0001
    Strength of the L2 regularization term. The L2 regularization term
    is divided by the sample size when added to the loss.

batch_size : int, default='auto'
    Size of minibatches for stochastic optimizers.
    If the solver is 'lbfgs', the classifier will not use minibatch.
```

```
    When set to "auto", `batch_size=min(200, n_samples)`.

learning_rate : {'constant', 'invscaling', 'adaptive'}, default='constant'
    Learning rate schedule for weight updates.

    - 'constant' is a constant learning rate given by
      'learning_rate_init'.

    - 'invscaling' gradually decreases the learning rate at each
      time step 't' using an inverse scaling exponent of 'power_t'.
      effective_learning_rate = learning_rate_init / pow(t, power_t)

    - 'adaptive' keeps the learning rate constant to
      'learning_rate_init' as long as training loss keeps decreasing.
      Each time two consecutive epochs fail to decrease training loss by at
      least tol, or fail to increase validation score by at least tol if
      'early_stopping' is on, the current learning rate is divided by 5.

    Only used when ``solver='sgd'``.

learning_rate_init : float, default=0.001
    The initial learning rate used. It controls the step-size
    in updating the weights. Only used when solver='sgd' or 'adam'.

power_t : float, default=0.5
    The exponent for inverse scaling learning rate.
    It is used in updating effective learning rate when the learning_rate
    is set to 'invscaling'. Only used when solver='sgd'.

max_iter : int, default=200
    Maximum number of iterations. The solver iterates until convergence
    (determined by 'tol') or this number of iterations. For stochastic
    solvers ('sgd', 'adam'), note that this determines the number of epochs
    (how many times each data point will be used), not the number of
    gradient steps.

shuffle : bool, default=True
    Whether to shuffle samples in each iteration. Only used when
    solver='sgd' or 'adam'.

random_state : int, RandomState instance, default=None
    Determines random number generation for weights and bias
    initialization, train-test split if early stopping is used, and batch
    sampling when solver='sgd' or 'adam'.
    Pass an int for reproducible results across multiple function calls.
    See :term:`Glossary <random_state>`.

tol : float, default=1e-4
    Tolerance for the optimization. When the loss or score is not improving
    by at least ``tol`` for ``n_iter_no_change`` consecutive iterations,
    unless ``learning_rate`` is set to 'adaptive', convergence is
    considered to be reached and training stops.

verbose : bool, default=False
    Whether to print progress messages to stdout.
```

```
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. See :term:`the Glossary <warm_start>`.

momentum : float, default=0.9
    Momentum for gradient descent update. Should be between 0 and 1. Only
    used when solver='sgd'.

nesterovs_momentum : bool, default=True
    Whether to use Nesterov's momentum. Only used when solver='sgd' and
    momentum > 0.

early_stopping : bool, default=False
    Whether to use early stopping to terminate training when validation
    score is not improving. If set to true, it will automatically set
    aside 10% of training data as validation and terminate training when
    validation score is not improving by at least tol for
    ``n_iter_no_change`` consecutive epochs. The split is stratified,
    except in a multilabel setting.
    If early stopping is False, then the training stops when the training
    loss does not improve by more than tol for n_iter_no_change consecutive
    passes over the training set.
    Only effective when solver='sgd' or 'adam'.

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 early_stopping is True.

beta_1 : float, default=0.9
    Exponential decay rate for estimates of first moment vector in adam,
    should be in [0, 1). Only used when solver='adam'.

beta_2 : float, default=0.999
    Exponential decay rate for estimates of second moment vector in adam,
    should be in [0, 1). Only used when solver='adam'.

epsilon : float, default=1e-8
    Value for numerical stability in adam. Only used when solver='adam'.

n_iter_no_change : int, default=10
    Maximum number of epochs to not meet ``tol`` improvement.
    Only effective when solver='sgd' or 'adam'.

    .. versionadded:: 0.20

max_fun : int, default=15000
    Only used when solver='lbfgs'. Maximum number of loss function calls.
    The solver iterates until convergence (determined by 'tol'), number
    of iterations reaches max_iter, or this number of loss function calls.
    Note that number of loss function calls will be greater than or equal
    to the number of iterations for the `MLPClassifier`.

    .. versionadded:: 0.22
```

```
Attributes
----------
classes_ : ndarray or list of ndarray of shape (n_classes,)
    Class labels for each output.

loss_ : float
    The current loss computed with the loss function.

best_loss_ : float or None
    The minimum loss reached by the solver throughout fitting.
    If `early_stopping=True`, this attribute is set ot `None`. Refer to
    the `best_validation_score_` fitted attribute instead.

loss_curve_ : list of shape (`n_iter_`,)
    The ith element in the list represents the loss at the ith iteration.

validation_scores_ : list of shape (`n_iter_`,) or None
    The score at each iteration on a held-out validation set. The score
    reported is the accuracy score. Only available if `early_stopping=True`,
    otherwise the attribute is set to `None`.

best_validation_score_ : float or None
    The best validation score (i.e. accuracy score) that triggered the
    early stopping. Only available if `early_stopping=True`, otherwise the
    attribute is set to `None`.

t_ : int
    The number of training samples seen by the solver during fitting.

coefs_ : list of shape (n_layers - 1,)
    The ith element in the list represents the weight matrix corresponding
    to layer i.

intercepts_ : list of shape (n_layers - 1,)
    The ith element in the list represents the bias vector corresponding to
    layer i + 1.

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_ : int
    The number of iterations the solver has run.

n_layers_ : int
    Number of layers.

n_outputs_ : int
    Number of outputs.
```

```
out_activation_ : str
    Name of the output activation function.

See Also
--------
MLPRegressor : Multi-layer Perceptron regressor.
BernoulliRBM : Bernoulli Restricted Boltzmann Machine (RBM).

Notes
-----
MLPClassifier trains iteratively since at each time step
the partial derivatives of the loss function with respect to the model
parameters are computed to update the parameters.

It can also have a regularization term added to the loss function
that shrinks model parameters to prevent overfitting.

This implementation works with data represented as dense numpy arrays or
sparse scipy arrays of floating point values.

References
----------
Hinton, Geoffrey E. "Connectionist learning procedures."
Artificial intelligence 40.1 (1989): 185-234.

Glorot, Xavier, and Yoshua Bengio.
"Understanding the difficulty of training deep feedforward neural networks."
International Conference on Artificial Intelligence and Statistics. 2010.

:arxiv:`He, Kaiming, et al (2015). "Delving deep into rectifiers:
Surpassing human-level performance on imagenet classification." <1502.01852>`

:arxiv:`Kingma, Diederik, and Jimmy Ba (2014)
"Adam: A method for stochastic optimization." <1412.6980>`

Examples
--------
>>> from sklearn.neural_network import MLPClassifier
>>> from sklearn.datasets import make_classification
>>> from sklearn.model_selection import train_test_split
>>> X, y = make_classification(n_samples=100, random_state=1)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
...                                                     random_state=1)
>>> clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
>>> clf.predict_proba(X_test[:1])
array([[0.038..., 0.961...]])
>>> clf.predict(X_test[:5, :])
array([1, 0, 1, 0, 1])
>>> clf.score(X_test, y_test)
0.8...
```

In [210…

```
mlp_params = {"alpha": [0.1, 0.01] , "hidden_layer_sizes": [(50,50,50),(100,
              "solver" : ["lbfgs" , "adam", "sgd"] , "activation" : ["relu"
```

In [211…
```
mlp = MLPClassifier()
```

In [212…
```
mlp_cv = GridSearchCV(mlp, mlp_params , cv = 10, n_jobs=-1, verbose=2).fit(x
```

```
Fitting 10 folds for each of 24 candidates, totalling 240 fits
C:\Users\EKİN\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\neural_network\_multilayer_perceptron.py:679: ConvergenceWarning: Stochast
ic Optimizer: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  warnings.warn(
```

In [213…
```
mlp_cv.best_params_
```

Out[213]:
```
{'activation': 'logistic',
 'alpha': 0.01,
 'hidden_layer_sizes': (50, 50, 50),
 'solver': 'adam'}
```

In [215…
```
mlp_best = MLPClassifier(activation= "logistic" ,alpha = 0.01, hidden_layer_
```

In [216…
```
preds_mlp_best = mlp_best.predict(x_test_scaled)
```

In [217…
```
accuracy_score(y_test, preds_mlp_best)
```

Out[217]:
```
0.72
```

In [275…
```
#Optimize ettikten sonra doğruluk payımız %72'ye yükseldi.
#Verilerimizi daha geniş bir aralıkta seçseydik daha iyi bir sonuç da alabil
#Bir de test üzerinden modele bakarsak:
```

In [219…
```
mlp_model = MLPClassifier().fit(x_test_scaled,y_test)
preds_mlp_test = mlp_model.predict(x_test_scaled)
```

```
C:\Users\EKİN\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklea
rn\neural_network\_multilayer_perceptron.py:679: ConvergenceWarning: Stochast
ic Optimizer: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  warnings.warn(
```

In [220…
```
accuracy_score(y_test, preds_mlp_test)
```

Out[220]:
```
1.0
```

In [222…
```
#Doğruluk payı %100 olan bir test elde ettik. Bu testin over-fitting olma ih
```

Sınıflandırma İçin CART

In [223…
```
from sklearn.tree import DecisionTreeClassifier
```

In [224…
```
cart_model = DecisionTreeClassifier().fit(x_train, y_train)
```

In [225…
```
preds_cart = cart_model.predict(x_test)
accuracy_score(y_test, preds_cart)
```

Out[225]: 0.56

In [227…
```
#Modelimizin doğruluk payı %56 olarak bulundu.
#Bunu yükseltmek için modelimizi optimize edelim.
```

In [228…
```
?cart_model
```

Type: DecisionTreeClassifier
String form: DecisionTreeClassifier()
File: c:\users\ekİn\appdata\local\programs\python\python311\lib\site-packages\sklearn\tree_classes.py
Docstring:

```
A decision tree classifier.

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

Parameters
----------
criterion : {"gini", "entropy", "log_loss"}, default="gini"
    The function to measure the quality of a split. Supported criteria are
    "gini" for the Gini impurity and "log_loss" and "entropy" both for the
    Shannon information gain, see :ref:`tree_mathematical_formulation`.

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
    [illegible]ht 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
          `max(1, int(max_features * n_features_in_))` features are considere
d 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`.

        .. deprecated:: 1.1
            The `"auto"` option was deprecated in 1.1 and will be removed
            in 1.3.

    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.

                    added:: 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).

                                  : 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.      ])
```

In [229…
```
cart_params = {"max_depth": list(range(1,10)) , "min_samples_split" :list(ra
```

In [230…
```
cart = DecisionTreeClassifier()
```

In [231…
```
cart_cv = GridSearchCV(cart, cart_params , cv = 10, n_jobs=-1, verbose = 2).
```

```
Fitting 10 folds for each of 72 candidates, totalling 720 fits
```

In [232…
```
cart_cv.best_params_
```

Out[232]:
```
{'max_depth': 6, 'min_samples_split': 9}
```

In [233…
```
cart_best_model = DecisionTreeClassifier(max_depth=6 , min_samples_split=9 )
```

In [234…
```
best_cart_preds = cart_best_model.predict(x_test)
accuracy_score(y_test, best_cart_preds)
```

Out[234]:
```
0.84
```

In [235…
```
#Optimizasyonlardan sonra doğruluk payı %56'dan %84'e çıktı.
#Şu ana kadar ulaştığımız en iyi sonuç.
```

Random Forrest Yöntemi

In [236…
```
from sklearn.ensemble import RandomForestClassifier
```

In [237…
```
rf_model = RandomForestClassifier().fit(x_train,y_train)
rf_preds = rf_model.predict(x_test)
accuracy_score(y_test, rf_preds)
```

Out[237]:
```
0.76
```

In [238…
```
#Modelimizin doğruluk payı %76 olarak başlıyoruz.
#Modelimizi optimize edelim.
```

In [239…
```
rf_params = {"max_depth":[2,5] , "max_features":[2,5] , "n_estimators":[10,1
```

In [240…
```
rf = RandomForestClassifier()
```

In [241…
```
rf_cv = GridSearchCV(rf, rf_params, cv = 10, n_jobs = -1, verbose=2).fit(x_t
```

```
Fitting 10 folds for each of 16 candidates, totalling 160 fits
```

In [242…
```
rf_best_model = RandomForestClassifier(max_depth=2 , max_features=8 , min_sa
```

In [243…
```
fr_best_preds = rf_best_model.predict(x_test)
```

In [244…
```
accuracy_score(y_test, fr_best_preds)
```

Out[244]: 0.72

In [245…
```
#Optimizasyonlardan sonra doğruluk payımız %72'ye düştü.
```

Gradiant Boosting Yöntemi

In [246…
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [247…
```
gb_model = GradientBoostingClassifier().fit(x_train,y_train)
```

In [248…
```
gb_preds = gb_model.predict(x_test)
```

In [249…
```
accuracy_score(y_test, gb_preds)
```

Out[249]: 0.64

In [250…
```
#Modelimizin doğruluk payı %64 olarak başlıyoruz.
#Modelimizi optimize edelim.
```

In [251…
```
gb_params = {"learning_rate": [0.001,0.01] , "n_estimators": [50,100] , "max
```

In [252…
```
gb = GradientBoostingClassifier()
```

In [253…
```
gb_cv = GridSearchCV(gb, gb_params , cv=10 , n_jobs =-1 , verbose=2).fit(x_t
```

```
Fitting 10 folds for each of 16 candidates, totalling 160 fits
```

In [254…
```
gb_cv.best_params_
```

Out[254]:
```
{'learning_rate': 0.01,
 'max_depth': 5,
 'min_samples_split': 2,
 'n_estimators': 100}
```

In [255…
```
gb_best_model = GradientBoostingClassifier(learning_rate = 0.01,n_estimators
```

In [256…
```
gb_best_preds = gb_best_model.predict(x_test)
```

In [257…
```
accuracy_score(y_test, gb_best_preds)
```

Out[257]: 0.76

In [276…
```
#Modelimizin doğruluk payı %64'ten %76'ya çıkıyor
#Ortalama bir değer fakat daha iyisini bulmuştuk.
```

Sınıflandırma İçin XGBoost Yöntemi

In [259…
```
from xgboost import XGBClassifier
```

In [260…
```
xgb_model = XGBClassifier().fit(x_train, y_train)
xgb_preds = xgb_model.predict(x_test)
accuracy_score(y_test, xgb_preds)
```

Out[260]: 0.76

In [264…
```
#Modelimizin doğruluk payı başlangıç değeri %76.
#Modelimizi optimize edelim.
```

In [265…
```
xgb_params = {"n_estimators":[10,100] , "subsample":[0.6,0.7] , "max_depth":
```

In [263…
```
xgb = XGBClassifier()
xgb_cv = GridSearchCV(xgb, xgb_params, cv = 10 , n_jobs = -1 , verbose=2).fi
```

```
Fitting 10 folds for each of 48 candidates, totalling 480 fits
[03:38:41] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscali
ng-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\learner.cc:767:
Parameters: { "min_samples_split" } are not used.
```

In [266…
```
xgb_cv.best_estimator_
```

Out[266]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_t
ypes=None,
              gamma=None, gpu_id=None, grow_policy=None, importance
_type=None,
              interaction_constraints=None, learning_rate=0.1, max_
bin=None,
```

In [267…
```
xgb_best_model =  XGBClassifier(learning_rate  =0.1 , max_depth = 6, min_sam
```

```
[03:39:02] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscali
ng-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\learner.cc:767:
Parameters: { "min_sample_split" } are not used.
```

In [268…
```
best_xgb_preds = xgb_best_model.predict(x_test)
accuracy_score(y_test, best_xgb_preds)
```

Out[268]: 0.64

In [277…
```
#Optimizasyonlardan sonra modelimizin doğruluk payı %64'e düştü.
#Bizim için yeterli bir sonuç değil.
```

Uygulamasını yaptığımız tüm yöntemler arasından modelimiz için optimize ettiğimiz "CART Yöntemi" bize %84'lük doğruluk payıyla en doğru sonucu vermiştir.

In [ ]:
```

```