# 新建dataframe

新建

import pandas as pd

a = pd.DataFrame([[1,2,3],

[4,5,6],

[7,8,9]],columns = ["feature\_1", "feature\_2", "label"])

读取

import pandas as pd

df = pd.read\_csv("datas/hour.csv", sep=",")

# Python write 问题

import time

start = time.time()

a = open('a',mode='w')

for i in range(1000000):

for j in range(10):

a.write(str(j))

a.close()

end = time.time()

end - start

import time

start = time.time()

a = open('a',mode='w')

s = ''

for i in range(1000000):

for j in range(10):

s += str(j)

a.write(s)

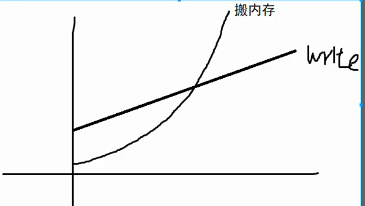
s = ''

a.close()

end = time.time()

end - start

适用场景：缓冲区较小时使用可以提高效率



# 删除dataframe列

del df["instant"]

df.drop(columns=["instant","dteday"])

# 修改dataframe列名

暴力

a.columns = ['a','b','c']

较好的方法

a.rename(columns={'A':'a', 'B':'b', 'C':'c'}, inplace = True)

# 查看dataframe字段信息

a.info()

# 修改dataframe列类型

df["instant"] = df["instant"].astype("object")

X[['Global\_active\_power',"b"]] = X[['Global\_active\_power',"b"]].astype('float64')

# 查看dataframe统计信息

a.describe()

# 获取dataframe部分列

a.iloc[:,0:3]

df.iloc[:,[-1]]

a[["feature\_1", "feature\_2"]]

# 获取dataframe列名

df.columns返回一个可迭代对象

for i in df.columns:

print(i)

# 获取dataframe的Series

一行

a.iloc[0,:]

一列

a.iloc[:,1]

a["feature\_1"]

# 合并dataframe

横向

pd.concat([a,a],axis=1)

纵向

pd.concat([a,a],axis=0)

# 获取dataframe所有列名

1. col\_names = df.columns.values.tolist()
2. col\_names = df.columns

# 获取列表里面最大值第一次出现的索引

## np.argmax(list) 只返回第一次出现的最大值的索引

# 分割训练集与测试集

from sklearn.model\_selection import train\_test\_split

X = a.iloc[:,0:-1]

Y = a["label"]

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.5)

Y\_train

# 统计Series值出现次数

a["feature\_1"].value\_counts()

# 异常数据处理

删除

a.replace('?', np.nan).dropna(how = 'any')

# pandas replace()

replace(self, to\_replace=None, value=None, inplace=False, limit=None, regex=False, method='pad', axis=None)

pandas 中 inplace 参数在很多函数中都会有，它的作用是：是否在原对象基础上进行修改

​ inplace = True：不创建新的对象，直接对原始对象进行修改；

​ inplace = False：对数据进行修改，创建并返回新的对象承载其修改结果。

默认是False，即创建新的对象进行修改，原对象不变，和深复制和浅复制有些类似。



## #Series对象值替换

s = df.iloc[2]#获取行索引为2数据

#单值替换

s.replace('?',np.nan)#用np.nan替换？

s.replace({'?':'NA'})#用NA替换？

#多值替换

s.replace(['?',r'$'],[np.nan,'NA'])#列表值替换

s.replace({'?':np.nan,'$':'NA'})#字典映射

#同缺失值填充方法类似

s.replace(['?','$'],method='pad')#向前填充

s.replace(['?','$'],method='ffill')#向前填充

s.replace(['?','$'],method='bfill')#向后填充

#limit参数控制填充次数

s.replace(['?','$'],method='bfill',limit=1)

## #DataFrame对象值替换

#单值替换

df.replace('?',np.nan)#用np.nan替换？

df.replace({'?':'NA'})#用NA替换？

#按列指定单值替换

df.replace({'EMPNO':'?'},np.nan)#用np.nan替换EMPNO列中?

df.replace({'EMPNO':'?','ENAME':'.'},np.nan)#用np.nan替换EMPNO列中?和ENAME中.

#多值替换

df.replace(['?','.','$'],[np.nan,'NA','None'])##用np.nan替换？用NA替换. 用None替换$

df.replace({'?':'NA','$':None})#用NA替换？ 用None替换$

df.replace({'?','$'},{'NA',None})#用NA替换？ 用None替换$

#正则替换

df.replace(r'\?|\.|\$',np.nan,regex=True)#用np.nan替换？或.或$原字符

df.replace([r'\?',r'\$'],np.nan,regex=True)#用np.nan替换？和$

df.replace([r'\?',r'\$'],[np.nan,'NA'],regex=True)#用np.nan替换？用NA替换$符号

df.replace(regex={r'\?':None})

#value参数显示传递

df.replace(regex=[r'\?|\.|\$'],value=np.nan)#用np.nan替换？或.或$原字符

---------------------

# 独热编码

import pandas as pd

a = pd.DataFrame([[1,2,3],

[4,5,6],

[1,8,9]],columns = ["feature\_1", "feature\_2", "label"])

from sklearn.preprocessing import OneHotEncoder

hotCoder=OneHotEncoder(sparse = False, handle\_unknown = "ignore")

hot = hotCoder.fit\_transform(a)

pd.DataFrame(hot)

b = pd.DataFrame([[1,2,3],

[4,5,6],

[10,8,9]],columns = ["feature\_1", "feature\_2", "label"])

hotCoder.transform(b)

# 多项式扩展

import pandas as pd

a = pd.DataFrame([[1,2,3],

[4,5,6],

[1,8,9]],columns = ["feature\_1", "feature\_2", "label"])

from sklearn.preprocessing import PolynomialFeatures

polyCoder = PolynomialFeatures(degree=2, include\_bias=True, interaction\_only=False)

df = polyCoder.fit\_transform(a)

pd.DataFrame(df, columns=polyCoder.get\_feature\_names())

# 标准化

import pandas as pd

a = pd.DataFrame([[1,2,3],

[4,5,6],

[7,8,9]],columns = ["feature\_1", "feature\_2", "label"])

from sklearn.preprocessing import StandardScaler

ssCoder = StandardScaler()

df = ssCoder.fit\_transform(a)

pd.DataFrame(df)

# LabelEncoder 特征数值化

from sklearn.preprocessing import LabelEncoder

import pandas as pd

a = pd.DataFrame([["b",2,3],

["a",5,6],

["a",8,9]],columns = ["feature\_1", "feature\_2", "label"])

laCoder = LabelEncoder()

b = pd.DataFrame(laCoder.fit\_transform(a["feature\_1"]))

pd.concat([a,b],axis=1)

# dataframe样本采样

df = a.sample(frac=0.66)

df = a.sample(n=3)

pd.concat([a,df])

# LinearRegression

import numpy as np

X = np.mat([[1,1],[2,1],[3,1],[4,1]])

Y = np.mat([[3.2],[4.7],[7.3],[8.5]])

from sklearn.linear\_model import LinearRegression

model = LinearRegression(fit\_intercept=False)

model.fit(X,Y)

model.coef\_

model.intercept\_

model.score(X,Y)

# Ridge

from sklearn.linear\_model import Ridge

for alpha in [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 2, 3, 5, 10]:

clf = Ridge(alpha=alpha, max\_iter=2000, solver="auto",fit\_intercept=True)

clf.fit(X\_train, Y\_train)

print("Ridge:",mse(Y\_test.values, clf.predict(X\_test)))

print(clf.n\_iter\_)

# Lasso

from sklearn.linear\_model import Lasso

for alpha in [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 2, 3]:

clf = Lasso(alpha=alpha, max\_iter=100, fit\_intercept=True)

clf.fit(X\_train, Y\_train)

print("Lasso:",mse(Y\_test.values, clf.predict(X\_test)))

print(clf.n\_iter\_)

# 模型评估

from sklearn.metrics import mean\_squared\_error

print("LinearRegression:",mean\_squared\_error(Y\_test.values, clf.predict(X\_test)))

# 混淆矩阵

pd.crosstab(Y\_test,knn.predict(X\_test),rownames=["label"],colnames=["predict"])

# 保存模型

from sklearn.externals import joblib

joblib.dump(enc,'rf.model')

enc2 = joblib.load('rf.model')

b = enc2.transform(a).toarray()

pd.DataFrame(b)

# CountVectorizer

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer, TfidfTransformer

corpus = [

'我 爱 你',

'我 恨 你'

]

y = [0,1]

vectorizer = CountVectorizer(token\_pattern="[a-zA-Z|\u4e00-\u9fa5]+")

count = vectorizer.fit\_transform(corpus)

print(vectorizer.get\_feature\_names())

print(count.toarray())

transformer = TfidfTransformer()

tfidf\_matrix = transformer.fit\_transform(count)

print(tfidf\_matrix.toarray())

tfidf\_vec = TfidfVectorizer(token\_pattern="[a-zA-Z|\u4e00-\u9fa5]+")

tfidf\_matrix = tfidf\_vec.fit\_transform(corpus)

print(tfidf\_matrix.toarray())

from sklearn.naive\_bayes import GaussianNB

model = GaussianNB()

model.fit(tfidf\_matrix.toarray(),y)

print(model.predict(tfidf\_matrix.toarray()))

corpus = [

'仇 恨',

'爱 你'

]

tfidf\_matrix = tfidf\_vec.transform(corpus)

model.predict(tfidf\_matrix.toarray())

# TfidfVectorizer

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer, TfidfTransformer

from sklearn.model\_selection import train\_test\_split

df = pd.read\_csv("datas/bayes.txt",header=None)

X = df[1]

Y = df[0]

tfCoder = TfidfVectorizer(token\_pattern="[a-zA-Z|\u4e00-\u9fa5]+")

X = tfCoder.fit\_transform(X).toarray()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0, random\_state=42)

from sklearn.naive\_bayes import GaussianNB

model = GaussianNB()

model.fit(X\_train,y\_train)

print(model.predict(X\_train))

print(y\_train.values)

# apply

from sklearn import preprocessing

import pandas as pd

enc = preprocessing.OneHotEncoder(categorical\_features=[0,1])

a = pd.DataFrame([[1,"A","a"],

[0,"B","b"],

[2,"C","c"]],columns = ["ebayno", "p\_sku", "sale"])

def f(x):

i = x.index

v = x.values\*2

print(v)

return pd.Series(v,i)

a.apply(f)

**import** pandas **as** pd  
  
a = pd.DataFrame([[1, **"A"**, **"a"**],  
 [0, **"B"**, **"bB"**],  
 [2, **"C"**, **"cC"**]])  
  
**def** f(x):  
  
 **if** x[1] **not in** x[2]:  
 **return** x  
 **else**:  
 **return** pd.Series()  
  
a = a.apply(f, axis=1)  
print(a.dropna())

# [Numpy中的矩阵合并](https://www.cnblogs.com/catmelo/p/4292960.html)

列合并/扩展：**np.column\_stack()**

行合并/扩展：**np.row\_stack()**

# Python copy

import copy  
a = [1,2,[1,2]]  
b = copy.deepcopy(a)  
a[2][0] = -1  
b

# numpy.ravel() 与numpy.flatten()

**numpy.flatten()返回一份拷贝**，对拷贝所做的修改不会影响（reflects）原始矩阵，   
**numpy.ravel()返回的是视图**（view，也颇有几分C/C++引用reference的意味），会影响（reflects）原始矩阵。

# Python pandas数据分析中常用方法

## https://blog.csdn.net/qq\_16234613/article/details/64217337

# tfidf

corpus=["hi peter",

"hi tom"]

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf2 = TfidfVectorizer(norm=None)

re = tfidf2.fit\_transform(corpus)

print(tfidf2.vocabulary\_)

print(tfidf2.get\_feature\_names())

print(re.todense())

# jupyter notebook 快捷键

#将代码块分割：点到选中的行Ctrl+Shift+-

#将代码块合并：使用Shift选中需要合并的框，Shift+m

#在代码块前增加新代码块，按a；在代码块后增加新代码块，按b；

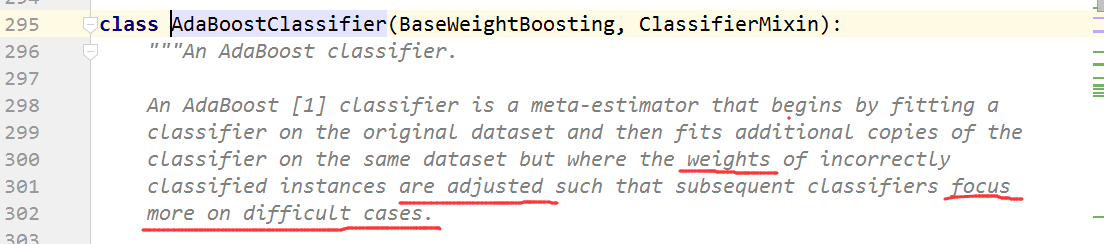
#删除代码块，按dd

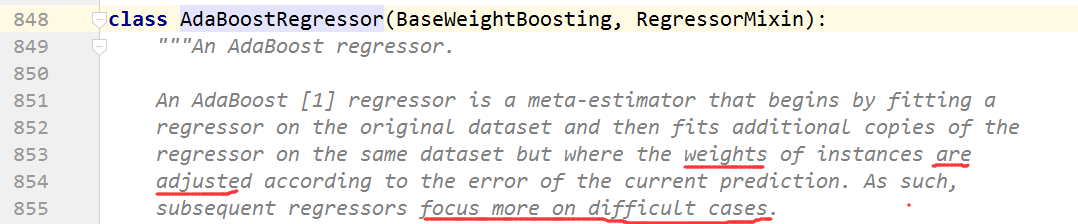
#运行当前代码块，Ctrl+Enter

#运行当前代码块并选中下一个代码块（没有就创建），Shift+Enter

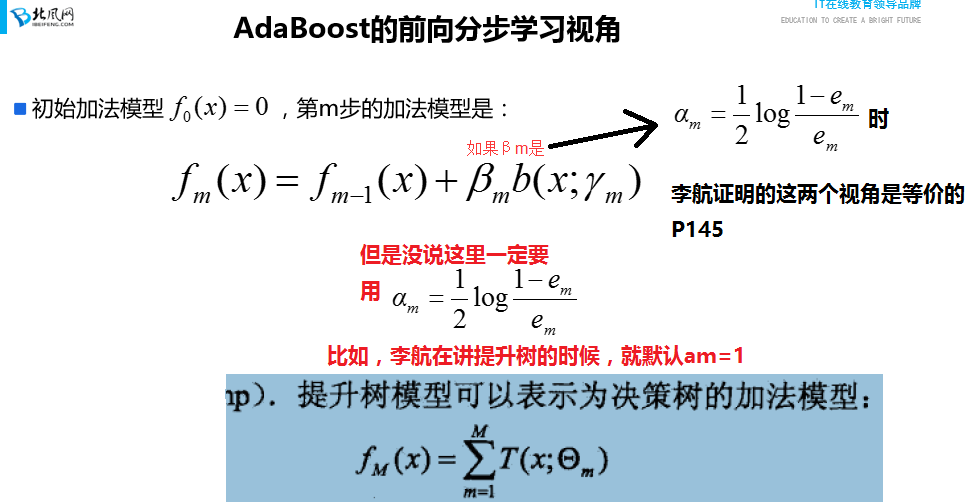
清除缓存kernel -> restart

1.首先sklearn都是从带权学习的视角出发的这是毋庸置疑的了，源码或官网上都写了



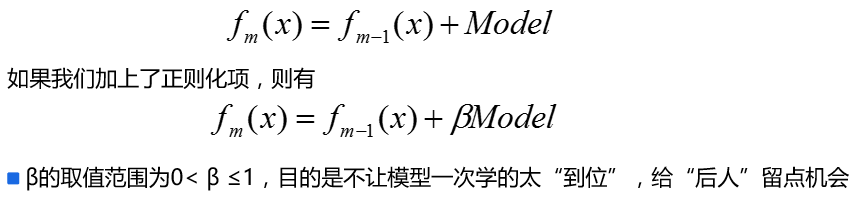


2.



3.如果从前向分布学习视角出发，

的时候，其实就不与带权学习视角下的分类问题是等价的了，这时候我自己玩自己的，比如我想加正则，都是没有关系的，就像这样



（不要一直想着这两个视角等价，其实只是在特定条件下等价）

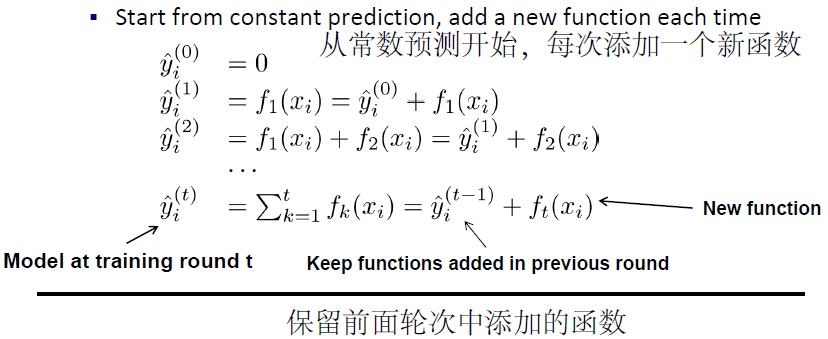
# xgboost小结

我们的目标函数是

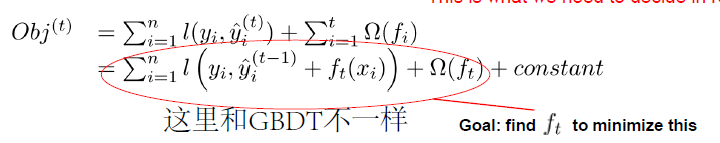


代价函数（每个损失的和）+正则化项（k棵树的复杂度的和）

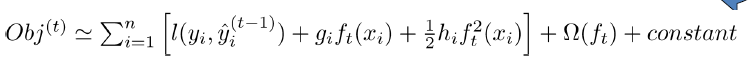
我们想让目标函数最小，但这是一个复杂的优化问题，要使用前向分布学习算法来求解，求解每一棵树不能使用SGD之类的算法，因为我们的模型是树（他不像线性模型），因此要使用加法模型



那么每一步的加法模型的求解就是最小化下图红框里的东西（代价+模型复杂度）

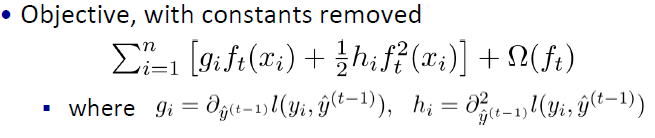


这里作者并没有直接进行最优化，而是先使用泰勒公式对目标函数变形，即：

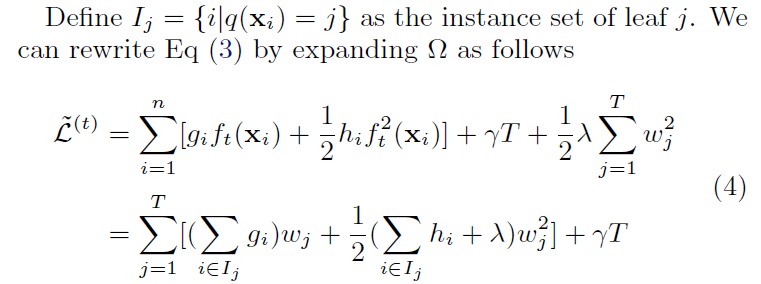


好处是使用二阶近似加快求解，另外也将损失函数抽离出来，以便日后你自定义损失函数，而内部代码无需修改，这是工程上一贯的作风（Second-order approximation can be used to quickly optimize the objective in the **general** setting）

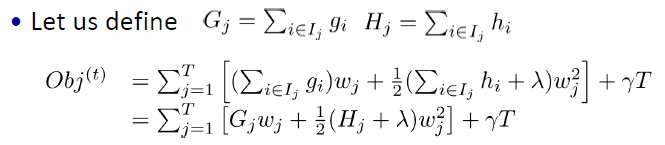
移除目标函数中的常量，变为：



作者将一棵树拆解为权值向量w和树的结构qx，并对新树ft(x)做了替代，同时将目标函数由单个样本的层面转到了叶子节点的层面，即n已经变为T了。



接着，作者这里又做了一些定义【注意：论文中的就是ppt中的】

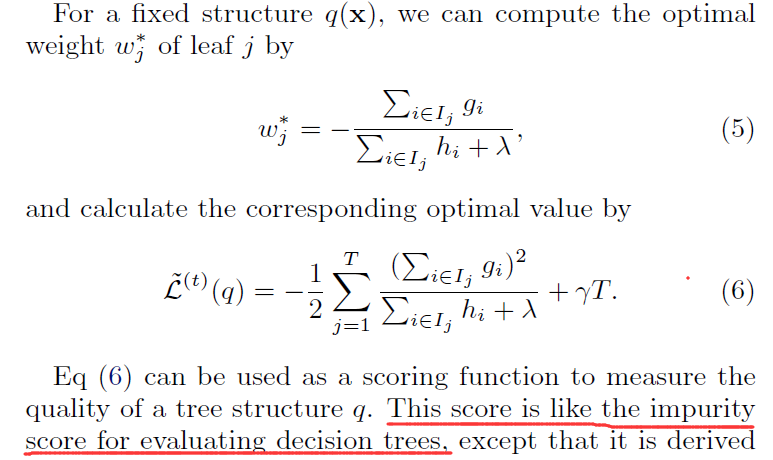


从上图可以看出，只要树的结构被确定，即知道了q(x)【给我一个样本，我可以返回他所在的节点】，那我的和就是常量了，那么Obj中就只剩一个变量w了，根据初中二次函数的知识，就能得到w\*，还能得到Obj的极小值。

那么我们问题的就归结到确定树结构q(x)的问题上了。

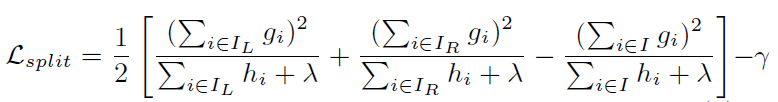
我们的思路是这样的：既然q(x)一经确定就有一个Obj，又希望Obj最小。大家有没有想到决策树构造的过程，根节点不纯度很大，随着节点的划分，都会让不纯度降低。那么在xgboost中，这个Obj就可以看做是不纯度，然后一层层地构建树（即，确定q(x)），从而让Obj变小

其实作者就是这个思路，如下图：



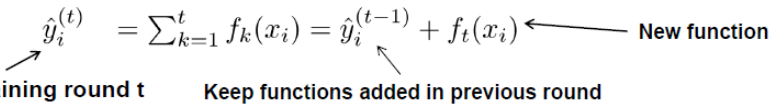
如果这个理解了，下面不就是像决策树一样使用信息增益构建树一样了吗？

贪心策略为：（最后的r表示：每次构建会多一个叶节点，这个是xgboost独有的，把模型复杂度考虑到贪心指标里了）



候选解集合我就不加赘述了

以上过程完成后，也就完成了加法模型中一棵树的构建



加入加法模型，继续加入新的树，重复以上操作，直至达到收敛条件或达到k次循环。

# 调包

## 线性模型

### LinearRegression（线性回归）

**学习地址：**https://www.cnblogs.com/pinard/p/6016029.html

class sklearn.linear\_model.LinearRegression（fit\_intercept = True，normalize = False，copy\_X = True，n\_jobs = None ）

from sklearn.linear\_model import LinearRegression

model = LinearRegression(fit\_intercept=False)

**重点参数：**

**fit\_intercept**（截距）默认为为true

**n\_jobs**(x线程数)。

**属性：**

**model.coef\_**(获取特征系数)

**model. intercept\_**(获取截距项)

**方法**：

**model. get\_params()**(获取模型参数（超参数）)

**model.score(x,y)**返回R^2

### Lasso（lasso回归，求解方法：坐标轴下降法和最小角回归法。）（Lasso ）

**Lasso详细讲解：**https://www.cnblogs.com/pinard/p/6018889.html

目标函数：

（1/（2\*n\_samples ））\*||y-Xw||^2\_2+alpha\*||w||\_1

class sklearn.linear\_model.Lasso（alpha = 1.0，fit\_intercept = True，normalize = False，precompute = False，copy\_X = True，max\_iter = 1000，tol = 0.0001，warm\_start = False，positive = False，random\_state = None，selection ='cyclic' ）

from sklearn.linear\_model import Lasso

**重点参数**：

**alpha**：默认为1超参数

**max\_iter**:最大迭代次数默认1000

**ol:**优化容差，默认0.0001.

**属性：**

**model.coef\_**(获取特征系数)

**model. intercept\_**(获取截距项)

**model. n\_iter\_**:坐标下降求解运行的迭代次数。

**方法**：

**model. get\_params()**(获取模型参数（超参数）)

**model.score(x,y)**返回R^2

### 岭回归（Ridge）

**最小化目标函数**

|| y - Xw || ^ 2 \_2 + alpha \* || w || ^ 2 \_2

class sklearn.linear\_model.Ridge（alpha = 1.0，fit\_inte

rcept = True，normalize = False，copy\_X = True，max\_iter = None，tol = 0.001，solver ='auto'，random\_state = None ）

**重点参数：**

**alpha**（超参数）

**solver**（求解器）（可选：{'auto'（根据数据类型自动选择求解器），'svd'（奇异值分解），'cholesky'（使用标准的scipy.linalg.solve函数来获得封闭形式的解决方案），'lsqr'（使用专用的正则化最小二乘例程scipy.sparse.linalg.lsqr。它是最快的并且使用迭代过程。），'sparse\_cg'（使用scipy.sparse.linalg.cg中的共轭梯度求解器。作为一种迭代算法，这个求解器比“cholesky”更适合大规模数据），'sag'（使用随机平均梯度下降），'saga'（使用其改进的，无偏见的版本SAGA）}）

**属性：**

**model.coef\_**(获取特征系数)

**model. intercept\_**(获取截距项)

**model. n\_iter\_**:坐标下降求解运行的迭代次数。

**方法**：

**model. get\_params()**(获取模型参数（超参数）)

**model.score(x,y)**返回R^2

## logistic regression（逻辑回归：分类）

**主要参数详细：**https://www.cnblogs.com/pinard/p/6035872.html

class sklearn.linear\_model.LogisticRegression(penalty=’l2’, dual=False, tol=0.0001, C=1.0, fit\_intercept=True, intercept\_scaling=1, class\_weight=None, random\_state=None, solver=’warn’, max\_iter=100, multi\_class=’warn’, verbose=0, warm\_start=False, n\_jobs=None)[[source]](https://github.com/scikit-learn/scikit-learn/blob/bac89c2/sklearn/linear_model/logistic.py#L997)

**重点参数：**

**penalty** :l1/l2('newton-cg'，'sag'和'lbfgs'，'newton-cg'求解算器只支持l2惩罚)

**class\_weight**(类别权重默认无)

**solver**（求解器）（{'newton-cg'（拟牛顿法），'lbfgs'（牛顿法），'liblinear'（最小二乘），'sag'（平均梯度下降），'saga'}，默认：'liblinear'）

**multi\_class** ： str，{'ovr'，'multinomial'，'auto'}，默认值：'ovr'

**属性：**

**model.coef\_**  #(返回模型X的系数)

**方法：**

**print(model.decision\_function(X\_test))** #样本的置信度得分是该样本与超平面的有符号距离，正的为一类，负的为一类，离零越远越可信应该。

**print(model.predict(X\_test))**  #返回每个预测样本属于的类。

**print(model.predict\_proba(X\_test))**  #每个样本属于某一类的概率，该样本属于概率最大的类

**print(model.predict\_log\_proba(X\_test))** ##每个样本属于某一类的对数概率，该样本属于对数概率接近0的类。

**print(model.densify())**  #返回超参数(数组or元祖)

**print(model.get\_params())**  #返回模型超参数（字典）

## KNN（K近邻）（分类，回归）

*cl*ass sklearn.neighbors.KNeighborsRegressor(n\_neighbors=5, weights=’uniform’, algorithm=’auto’, leaf\_size=30, p=2, metric=’minkowski’, metric\_params=None, n\_jobs=None, \*\*kwargs)

*class*sklearn.neighbors.KNeighborsClassifier（n\_neighbors = 5，weights ='uniform'，algorithm ='auto'，leaf\_size = 30，p = 2，metric ='minkowski'，metric\_params = None，n\_jobs = None，\*\* kwargs ）

**重点参数：**

**Weights** #uniform(默认等权重)，distance（权重和距离成反比）

**n\_neighbors**  #邻居个数默认为5

**algorithm**  #计算方式（auto（默认），ball\_tree（球树）,kd\_tree,brute(暴力)，推荐kd\_tree）

**leaf\_size:** 叶子数量kd\_tree默认30

**metric**  距离:默认minkowski（闵可夫斯基）,当P给定2时就是欧几里得距离。

**P：**给定距离里面的p，默认为2。

**方法：**

[**get\_params**](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier.get_params)**()**，获取模型参数

[**predict**](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier.predict)**(X)**：预测

**predict\_proba（X）** 测试数据X的返回概率估计。

## 决策树（DecisionTree）（刘建平：https://www.cnblogs.com/pinard/p/6056319.html）

### 分类决策树：

（<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier>）

class sklearn.tree.**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，min\_impurity\_split = None，class\_weight = None，presort = False ）

**重要参数：**

**Criterion**（特征选择标准）：(gini或者entropy)前者代表基尼系数就是CART树；后者是信息增益最优特征选择法（ID3，C4.5）。

**Splitter**（特征划分点选择标准）：（best"或者"random）：best适合样本数量较少，random适合样本数量较多的。

**max\_features**（划分时考虑的最大特征数）：可以使用很多种类型的值，默认是"None",意味着划分时考虑所有的特征数；如果是"log2"意味着划分时最多考虑log2N个特征；如果是"sqrt"或者"auto"意味着划分时最多考虑N−−√个特征。如果是整数，代表考虑的特征绝对数。如果是浮点数，代表考虑特征百分比，即考虑（百分比xN）取整后的特征数。其中N为样本总特征数。

一般来说，如果样本特征数不多，比如小于50，我们用默认的"None"就可以了，如果特征数非常多，我们可以灵活使用刚才描述的其他取值来控制划分时考虑的最大特征数，以控制决策树的生成时间。

**max\_depth**（决策树最大深度（常用0-100之间。）

决策树的最大深度，默认可以不输入，如果不输入的话，决策树在建立子树的时候不会限制子树的深度。一般来说，数据少或者特征少的时候可以不管这个值。如果模型样本量多，特征也多的情况下，推荐限制这个最大深度，具体的取值取决于数据的分布。常用的可以取值10-100之间。

**min\_samples\_split**（内部节点再划分所需最小样本数）：这个值限制了子树继续划分的条件，如果某节点的样本数少于min\_samples\_split，则不会继续再尝试选择最优特征来进行划分。如果样本量数量级非常大，则推荐增大这个值。我之前的一个项目例子，**有大概10万样本，建立决策树时，我选择了min\_samples\_split=10。可以作为参考。**

**min\_samples\_leaf**（叶子节点最少样本数）：这个值限制了叶子节点最少的样本数，如果某叶子节点数目小于样本数，则会和兄弟节点一起被剪枝。 默认是1,可以输入最少的样本数的整数，或者最少样本数占样本总数的百分比。如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。**之前的10万样本项目使用min\_samples\_leaf的值为5，仅供参考。**

**min\_weight\_fraction\_leaf** （叶子节点最小的样本权重和）：这个值限制了叶子节点所有样本权重和的最小值，如果小于这个值，则会和兄弟节点一起被剪枝。 默认是0，就是不考虑权重问题。一般来说，如果我们有较多样本有缺失值，或者分类树样本的分布类别偏差很大，就会引入样本权重，这时我们就要注意这个值了。

**max\_leaf\_nodes**（最大叶子节点数）：通过限制最大叶子节点数，可以防止过拟合，默认是"None”，即不限制最大的叶子节点数。如果加了限制，算法会建立在最大叶子节点数内最优的决策树。如果特征不多，可以不考虑这个值，但是如果特征分成多的话，可以加以限制，具体的值可以通过交叉验证得到。

**class\_weight**（类别权重）：指定样本各类别的的权重，主要是为了防止训练集某些类别的样本过多，导致训练的决策树过于偏向这些类别。这里可以自己指定各个样本的权重，或者用“balanced”，如果使用“balanced”，则算法会自己计算权重，样本量少的类别所对应的样本权重会高。当然，如果你的样本类别分布没有明显的偏倚，则可以不管这个参数，选择默认的"None"

**节点划分最小不纯度min\_impurity\_split**：这个值限制了决策树的增长，如果某节点的不纯度(**基尼系数，信息增益，均方差，绝对差**)小于这个阈值，则该节点不再生成子节点。即为叶子节点 。

**数据是否预排序presort**：这个值是布尔值，默认是False不排序。一般来说，如果样本量少或者限制了一个深度很小的决策树，设置为true可以让划分点选择更加快，决策树建立的更加快。如果样本量太大的话，反而没有什么好处。问题是样本量少的时候，我速度本来就不慢。所以这个值一般懒得理它就可以了。

**重要属性**

**print(model.classes\_)**  ##输出label都有哪些值。[1 2 3 4 5 6 7]

**print(model.feature\_importances\_)** ##返回每个特征重要性

**print(model.max\_features\_)**# 54 ##返回特征个数

**print(model.n\_classes\_)**  # 7 ##返回label个数

**print(model.n\_features\_)**  # 54 ##返回特征个数（目前看来是这样）

**print(model.n\_outputs\_)**  # 1

**print(model.tree\_)**  ##没看懂详情了解DecisionTreeClassifier里面查看

**重要方法：**



### 回归决策树(CART)

（<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor>）

class sklearn.tree.**DecisionTreeRegressor**（criterion ='mse'，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，min\_impurity\_split = None，presort = False ）

**重要参数：**

**Criterion**（特征选择标准）：（mse；mae）前者是均方差，后者是和均值之差的绝对值之和。推荐使用默认的"mse"。一般来说"mse"比"mae"更加精确

**Splitter**（特征划分点选择标准）：（best"或者"random）：best适合样本数量较少，random适合样本数量较多的。

**max\_features**（划分时考虑的最大特征数）：可以使用很多种类型的值，默认是"None",意味着划分时考虑所有的特征数；如果是"log2"意味着划分时最多考虑log2N个特征；如果是"sqrt"或者"auto"意味着划分时最多考虑N−−√个特征。如果是整数，代表考虑的特征绝对数。如果是浮点数，代表考虑特征百分比，即考虑（百分比xN）取整后的特征数。其中N为样本总特征数。

一般来说，如果样本特征数不多，比如小于50，我们用默认的"None"就可以了，如果特征数非常多，我们可以灵活使用刚才描述的其他取值来控制划分时考虑的最大特征数，以控制决策树的生成时间。

**max\_depth**（决策树最大深度（常用0-100之间。）

决策树的最大深度，默认可以不输入，如果不输入的话，决策树在建立子树的时候不会限制子树的深度。一般来说，数据少或者特征少的时候可以不管这个值。如果模型样本量多，特征也多的情况下，推荐限制这个最大深度，具体的取值取决于数据的分布。常用的可以取值10-100之间。

**min\_samples\_split**（内部节点再划分所需最小样本数）：这个值限制了子树继续划分的条件，如果某节点的样本数少于min\_samples\_split，则不会继续再尝试选择最优特征来进行划分。如果样本量数量级非常大，则推荐增大这个值。我之前的一个项目例子，**有大概10万样本，建立决策树时，我选择了min\_samples\_split=10。可以作为参考。**

**min\_samples\_leaf**（叶子节点最少样本数）：这个值限制了叶子节点最少的样本数，如果某叶子节点数目小于样本数，则会和兄弟节点一起被剪枝。 默认是1,可以输入最少的样本数的整数，或者最少样本数占样本总数的百分比。如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。**之前的10万样本项目使用min\_samples\_leaf的值为5，仅供参考。**

**min\_weight\_fraction\_leaf** （叶子节点最小的样本权重和）：这个值限制了叶子节点所有样本权重和的最小值，如果小于这个值，则会和兄弟节点一起被剪枝。 默认是0，就是不考虑权重问题。一般来说，如果我们有较多样本有缺失值，或者分类树样本的分布类别偏差很大，就会引入样本权重，这时我们就要注意这个值了。

**max\_leaf\_nodes**（最大叶子节点数）：通过限制最大叶子节点数，可以防止过拟合，默认是"None”，即不限制最大的叶子节点数。如果加了限制，算法会建立在最大叶子节点数内最优的决策树。如果特征不多，可以不考虑这个值，但是如果特征分成多的话，可以加以限制，具体的值可以通过交叉验证得到。

**节点划分最小不纯度min\_impurity\_split**：这个值限制了决策树的增长，如果某节点的不纯度(**基尼系数，信息增益，均方差，绝对差**)小于这个阈值，则该节点不再生成子节点。即为叶子节点 。

**数据是否预排序presort**：这个值是布尔值，默认是False不排序。一般来说，如果样本量少或者限制了一个深度很小的决策树，设置为true可以让划分点选择更加快，决策树建立的更加快。如果样本量太大的话，反而没有什么好处。问题是样本量少的时候，我速度本来就不慢。所以这个值一般懒得理它就可以了。

**重要属性：**



**重要方法：**

****

## 集成模型Bagging:RF ; Boosting:Adaboost;GBDT;XGBoost;LightGBM;CatBoost;)

### Bagging ：RF

RF(RandomForestClassifier; RandomForestRegressor)

（子模型为树模型）

**分类：**

（<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier>）

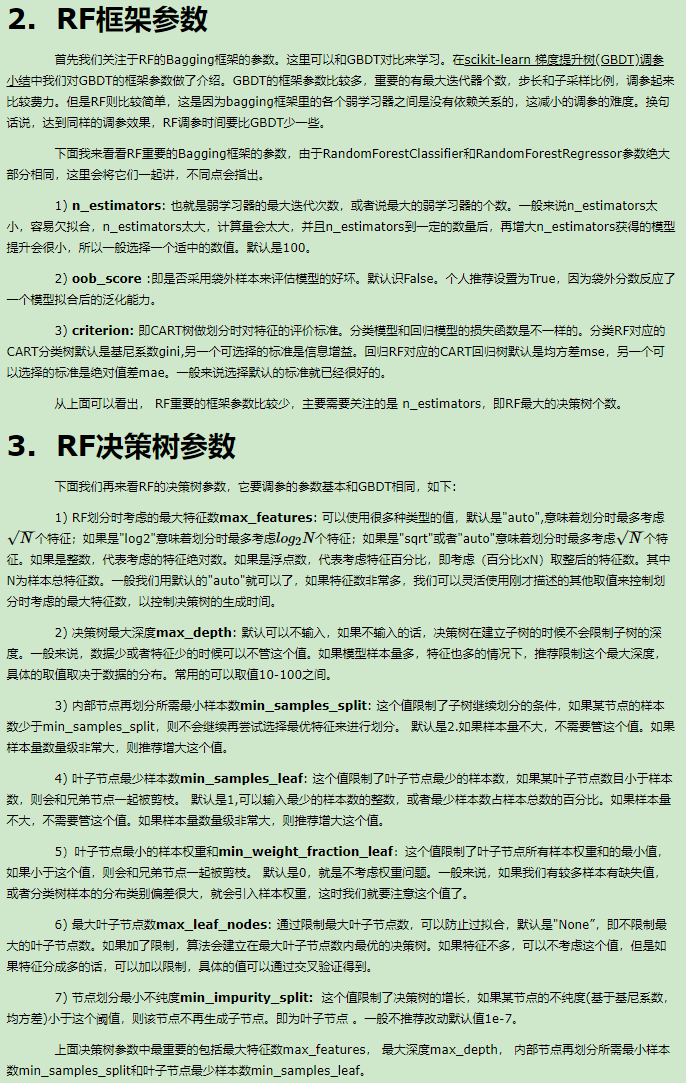
class sklearn.ensemble.RandomForestClassifier（n\_estimators ='warn'，criterion =' gini '，max\_depth = None，min\_samples\_split = 2，min\_samples\_leaf = 1，min\_weight\_fraction\_leaf = 0.0，max\_features ='auto'，max\_leaf\_nodes = None，min\_impurity\_decrease = 0.0，min\_impurity\_split = None，bootstrap =是，oob\_score = False，n\_jobs =无，random\_state =无，verbose = 0，warm\_start = False，class\_weight =无）

**重要参数：**

**n\_estimators （树的数量）:默认为10.**

**oob\_score：**即是否采用袋外样本来评估模型的好坏。默认识False。个人推荐设置为True，因为袋外分数反应了一个模型拟合后的泛化能力。

**n\_jobs ：线程数。**



**重要属性：**

estimators\_：***DecisionTreeClassifier列表***

oob\_score\_  ：使用袋外估计获得的训练数据集的分数。

feature\_importances**\_：返回特征重要性数组。**

**重要方法：**



**回归：**

class sklearn.ensemble.**RandomForestRegressor**(n\_estimators=’warn’, criterion=’mse’, 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, min\_impurity\_split=None, bootstrap=True, oob\_score=False, n\_jobs=None, random\_state=None, verbose=0, warm\_start=False)[[source]](https://github.com/scikit-learn/scikit-learn/blob/bac89c2/sklearn/ensemble/forest.py#L1038)

（<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor>）

**重要参数：参照分类**

**重要属性：参考分类**

**重要方法：参考分类**

### Boosting：AdaBoost

**分类：**

class sklearn.ensemble.**AdaBoostClassifier**（base\_estimator = None，n\_estimators = 50，learning\_rate = 1.0，algorithm ='SAMME.R'，random\_state = None ）

**（**[**https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier**](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier)**）**

**回归：**

class sklearn.ensemble.**AdaBoostRegressor**（base\_estimator = None，n\_estimators = 50，learning\_rate = 1.0，loss ='linear'，random\_state = None ）

（[**https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html#sklearn.ensemble.AdaBoostRegressor**](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html#sklearn.ensemble.AdaBoostRegressor)**）**

**重要参数：框架参数，弱学习器参数**

**框架参数：**

**AdaBoostRegressor框架参数**

　　我们首先来看看AdaBoostClassifier和AdaBoostRegressor框架参数。两者大部分框架参数相同，下面我们一起讨论这些参数，两个类如果有不同点我们会指出。

　　1）**base\_estimator：**AdaBoostClassifier和AdaBoostRegressor都有，即我们的弱分类学习器或者弱回归学习器。理论上可以选择任何一个分类或者回归学习器，不过需要支持样本权重。我们常用的一般是CART决策树或者神经网络MLP。默认是决策树，即AdaBoostClassifier默认使用CART分类树DecisionTreeClassifier，而AdaBoostRegressor默认使用CART回归树DecisionTreeRegressor。另外有一个要注意的点是，如果我们选择的AdaBoostClassifier算法是SAMME.R，则我们的弱分类学习器还需要支持概率预测，也就是在scikit-learn中弱分类学习器对应的预测方法除了predict还需要有predict\_proba。

　　2）**algorithm：**这个参数只有AdaBoostClassifier有。主要原因是scikit-learn实现了两种Adaboost分类算法，SAMME和SAMME.R。两者的主要区别是弱学习器权重的度量，SAMME使用了和我们的原理篇里二元分类Adaboost算法的扩展，即用对样本集分类效果作为弱学习器权重，而SAMME.R使用了对样本集分类的预测概率大小来作为弱学习器权重。由于SAMME.R使用了概率度量的连续值，迭代一般比SAMME快，因此AdaBoostClassifier的默认算法algorithm的值也是SAMME.R。我们一般使用默认的SAMME.R就够了，但是要注意的是使用了SAMME.R， 则弱分类学习器参数base\_estimator必须限制使用支持概率预测的分类器。SAMME算法则没有这个限制。

　　3）**loss：**这个参数只有AdaBoostRegressor有，Adaboost.R2算法需要用到。有线性‘linear’, 平方‘square’和指数 ‘exponential’三种选择, 默认是线性，一般使用线性就足够了，除非你怀疑这个参数导致拟合程度不好。这个值的意义在原理篇我们也讲到了，它对应了我们对第k个弱分类器的中第i个样本的误差的处理，即：如果是线性误差，则eki=|yi−Gk(xi)|Ek；如果是平方误差，则eki=(yi−Gk(xi))2E2k，如果是指数误差，则eki=1−exp（−yi+Gk(xi))Ek），Ek为训练集上的最大误差Ek=max|yi−Gk(xi)|i=1,2...m

　　4) **n\_estimators：** AdaBoostClassifier和AdaBoostRegressor都有，就是我们的弱学习器的最大迭代次数，或者说最大的弱学习器的个数。一般来说n\_estimators太小，容易欠拟合，n\_estimators太大，又容易过拟合，一般选择一个适中的数值。默认是50。在实际调参的过程中，我们常常将n\_estimators和下面介绍的参数learning\_rate一起考虑。

　　5) **learning\_rate:**  AdaBoostClassifier和AdaBoostRegressor都有，即每个弱学习器的权重缩减系数ν，在原理篇的正则化章节我们也讲到了，加上了正则化项，我们的强学习器的迭代公式为fk(x)=fk−1(x)+ναkGk(x)。ν的取值范围为0<ν≤1。对于同样的训练集拟合效果，较小的ν意味着我们需要更多的弱学习器的迭代次数。通常我们用步长和迭代最大次数一起来决定算法的拟合效果。所以这两个参数n\_estimators和learning\_rate要一起调参。一般来说，可以从一个小一点的ν开始调参，默认是1。

**弱学习器参数：**

**AdaBoostClassifier和AdaBoostRegressor弱学习器参数**

　　这里我们再讨论下AdaBoostClassifier和AdaBoostRegressor弱学习器参数，由于使用不同的弱学习器，则对应的弱学习器参数各不相同。这里我们仅仅讨论默认的决策树弱学习器的参数。即CART分类树DecisionTreeClassifier和CART回归树DecisionTreeRegressor。

　　DecisionTreeClassifier和DecisionTreeRegressor的参数基本类似，在scikit-learn决策树算法类库使用小结这篇文章中我们对这两个类的参数做了详细的解释。这里我们只拿出调参数时需要尤其注意的最重要几个的参数再拿出来说一遍：

　　1) **划分时考虑的最大特征数max\_features:** 可以使用很多种类型的值，默认是"None",意味着划分时考虑所有的特征数；如果是"log2"意味着划分时最多考虑log2N个特征；如果是"sqrt"或者"auto"意味着划分时最多考虑N−−√个特征。如果是整数，代表考虑的特征绝对数。如果是浮点数，代表考虑特征百分比，即考虑（百分比xN）取整后的特征数。其中N为样本总特征数。一般来说，如果样本特征数不多，比如小于50，我们用默认的"None"就可以了，如果特征数非常多，我们可以灵活使用刚才描述的其他取值来控制划分时考虑的最大特征数，以控制决策树的生成时间。

　　2) **决策树最大深max\_depth:** 默认可以不输入，如果不输入的话，决策树在建立子树的时候不会限制子树的深度。一般来说，数据少或者特征少的时候可以不管这个值。如果模型样本量多，特征也多的情况下，推荐限制这个最大深度，具体的取值取决于数据的分布。常用的可以取值10-100之间。

　　3) **内部节点再划分所需最小样本数min\_samples\_split:** 这个值限制了子树继续划分的条件，如果某节点的样本数少于min\_samples\_split，则不会继续再尝试选择最优特征来进行划分。 默认是2.如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。

　　4) **叶子节点最少样本数min\_samples\_leaf:** 这个值限制了叶子节点最少的样本数，如果某叶子节点数目小于样本数，则会和兄弟节点一起被剪枝。 默认是1,可以输入最少的样本数的整数，或者最少样本数占样本总数的百分比。如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。

　　5）**叶子节点最小的样本权重和min\_weight\_fraction\_leaf：**这个值限制了叶子节点所有样本权重和的最小值，如果小于这个值，则会和兄弟节点一起被剪枝。 默认是0，就是不考虑权重问题。一般来说，如果我们有较多样本有缺失值，或者分类树样本的分布类别偏差很大，就会引入样本权重，这时我们就要注意这个值了。

　　6) **最大叶子节点数max\_leaf\_nodes:** 通过限制最大叶子节点数，可以防止过拟合，默认是"None”，即不限制最大的叶子节点数。如果加了限制，算法会建立在最大叶子节点数内最优的决策树。如果特征不多，可以不考虑这个值，但是如果特征分成多的话，可以加以限制，具体的值可以通过交叉验证得到。

**重要属性：**

**estimators\_：弱学习器集合。**

**classes\_ ：标签数组（只有分类有）。**

**estimator\_weights\_ ：每个分类器的权重。**

**estimator\_errors\_ ：增强系统中每个估计器的分类误差。**

**feature\_importances\_：特征重要性**

**重要方法：**

**分类方法：**



**回归方法：**



### Boosting ：GBDT

**分类：**

**（**[**https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html#sklearn.ensemble.GradientBoostingClassifier**](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html#sklearn.ensemble.GradientBoostingClassifier)**）**

class sklearn.ensemble.**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，min\_impurity\_split = None，init = None，random\_state = None，max\_features = None，verbose = 0，max\_leaf\_nodes = None，warm\_start = False，presort ='auto'，validation\_fraction = 0.1，n\_iter\_no\_change =无，tol = 0.0001 ）

**回归:**

**(**[**https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor**](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor)**)**

class sklearn.ensemble.**GradientBoostingRegressor**（loss ='ls'，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，min\_impurity\_split = None，init = None，random\_state = None，max\_features = None，alpha = 0.9，verbose = 0，max\_leaf\_nodes = None，warm\_start = False，presort ='auto'，validation\_fraction = 0.1，n\_iter\_no\_change = None，tol = 0.0001 ）

**重要参数：框架参数，弱学习器参数。**

**框架参数：**

**GBDT类库boosting框架参数**

　　首先，我们来看boosting框架相关的重要参数。由于GradientBoostingClassifier和GradientBoostingRegressor的参数绝大部分相同，我们下面会一起来讲，不同点会单独指出。

**1) n\_estimators: 也就是弱学习器的最大迭代次数，或者说最大的弱学习器的个数**。一般来说n\_estimators太小，容易欠拟合，n\_estimators太大，又容易过拟合，一般选择一个适中的数值。默认是100。在实际调参的过程中，我们常常将n\_estimators和下面介绍的参数learning\_rate一起考虑。

**2) learning\_rate: 即每个弱学习器的权重缩减系数ν，也称作步长，在原理篇的正则化章节我们也讲到了，加上了正则化项，我们的强学习器的迭代公式为fk(x)=fk−1(x)+νhk(x)。**ν的取值范围为0<ν≤1。对于同样的训练集拟合效果，较小的ν意味着我们需要更多的弱学习器的迭代次数。通常我们用步长和迭代最大次数一起来决定算法的拟合效果。所以这两个参数n\_estimators和learning\_rate要一起调参。一般来说，可以从一个小一点的ν开始调参，默认是1。

**3) subsample: 即我们在原理篇的正则化章节讲到的子采样，取值为(0,1]。**注意这里的子采样和随机森林不一样，随机森林使用的是放回抽样，而这里是不放回抽样。如果取值为1，则全部样本都使用，等于没有使用子采样。如果取值小于1，则只有一部分样本会去做GBDT的决策树拟合。选择小于1的比例可以减少方差，即防止过拟合，但是会增加样本拟合的偏差，因此取值不能太低。推荐在[0.5, 0.8]之间，默认是1.0，即不使用子采样。

**4) init: 即我们的初始化的时候的弱学习器**，拟合对应原理篇里面的f0(x)，如果不输入，则用训练集样本来做样本集的初始化分类回归预测。否则用init参数提供的学习器做初始化分类回归预测。一般用在我们对数据有先验知识，或者之前做过一些拟合的时候，如果没有的话就不用管这个参数了。**有一个模型，它的输出结果会用来作为GBDT模型的起始估计，这个时候就可以用init**

**5) loss: 即我们GBDT算法中的损失函数。分类模型和回归模型的损失函数是不一样的。**

**对于分类模型，**有**对数似然损失函数"deviance"和指数损失函数"exponential"**两者输入选择。默认是对数似然损失函数"deviance"。在原理篇中对这些分类损失函数有详细的介绍。一般来说，推荐使用默认的"deviance"。它对二元分离和多元分类各自都有比较好的优化。而指数损失函数等于把我们带到了Adaboost算法。

**对于回归模型，有均方差"ls", 绝对损失"lad", Huber损失"huber"和分位数损失“quantile”**。默认是均方差"ls"。一般来说，如果数据的噪音点不多，用默认的均方差"ls"比较好。如果是噪音点较多，则推荐用抗噪音的损失函数"huber"。而如果我们需要对训练集进行分段预测的时候，则采用“quantile”。

**6) alpha：这个参数只有GradientBoostingRegressor有，当我们使用Huber损失"huber"和分位数损失“quantile”时，需要指定分位数的值。默认是0.9，如果噪音点较多，可以适当降低这个分位数的值。**

**弱学习器参数：**

**GBDT类库弱学习器参数**

　　这里我们再对GBDT的类库弱学习器的重要参数做一个总结。**由于GBDT使用了CART回归决策树**，因此它的参数基本来源于决策树类，也就是说，和DecisionTreeClassifier和DecisionTreeRegressor的参数基本类似。如果你已经很熟悉决策树算法的调参，那么这一节基本可以跳过。不熟悉的朋友可以继续看下去。

**1) 划分时考虑的最大特征数max\_features**: 可以使用很多种类型的值，默认是"None",意味着划分时考虑所有的特征数；如果是"log2"意味着划分时最多考虑log2N个特征；如果是"sqrt"或者"auto"意味着划分时最多考虑N−−√个特征。如果是整数，代表考虑的特征绝对数。如果是浮点数，代表考虑特征百分比，即考虑（百分比xN）取整后的特征数。其中N为样本总特征数。一般来说，如果样本特征数不多，比如小于50，我们用默认的"None"就可以了，如果特征数非常多，我们可以灵活使用刚才描述的其他取值来控制划分时考虑的最大特征数，以控制决策树的生成时间。

**2) 决策树最大深度max\_depth**: 默认可以不输入，如果不输入的话，决策树在建立子树的时候不会限制子树的深度。一般来说，数据少或者特征少的时候可以不管这个值。如果模型样本量多，特征也多的情况下，推荐限制这个最大深度，具体的取值取决于数据的分布。常用的可以取值10-100之间。

**3) 内部节点再划分所需最小样本数min\_samples\_split**: 这个值限制了子树继续划分的条件，如果某节点的样本数少于min\_samples\_split，则不会继续再尝试选择最优特征来进行划分。 默认是2.如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。

**4) 叶子节点最少样本数min\_samples\_leaf**: 这个值限制了叶子节点最少的样本数，如果某叶子节点数目小于样本数，则会和兄弟节点一起被剪枝。 默认是1,可以输入最少的样本数的整数，或者最少样本数占样本总数的百分比。如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。

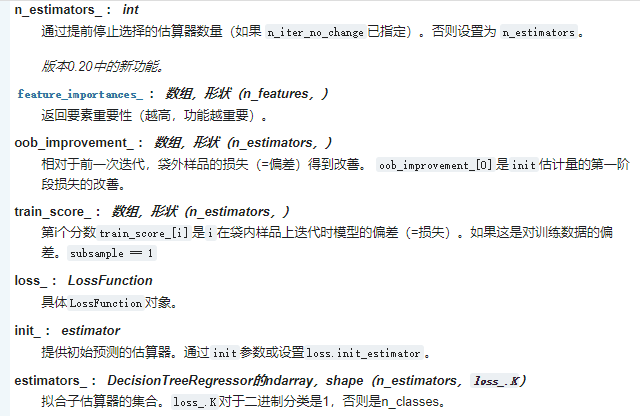
**5）叶子节点最小的样本权重和min\_weight\_fraction\_leaf**：这个值限制了叶子节点所有样本权重和的最小值，如果小于这个值，则会和兄弟节点一起被剪枝。 默认是0，就是不考虑权重问题。一般来说，如果我们有较多样本有缺失值，或者分类树样本的分布类别偏差很大，就会引入样本权重，这时我们就要注意这个值了。

**6) 最大叶子节点数max\_leaf\_nodes**: 通过限制最大叶子节点数，可以防止过拟合，默认是"None”，即不限制最大的叶子节点数。如果加了限制，算法会建立在最大叶子节点数内最优的决策树。如果特征不多，可以不考虑这个值，但是如果特征分成多的话，可以加以限制，具体的值可以通过交叉验证得到。

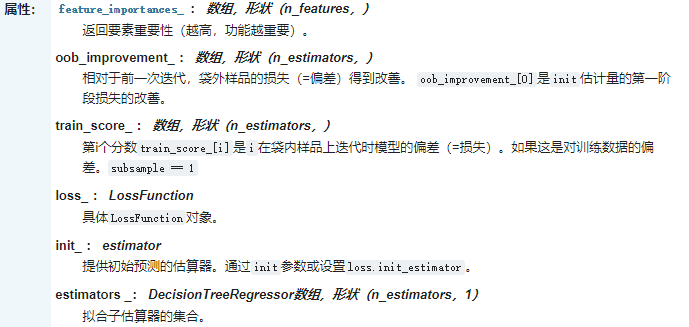
**7) 节点划分最小不纯度min\_impurity\_split**: 这个值限制了决策树的增长，如果某节点的不纯度(基于基尼系数，均方差)小于这个阈值，则该节点不再生成子节点。即为叶子节点 。一般不推荐改动默认值1e-7。

**重要属性：**

**分类：**



**回归属性：**



**分类方法：**



**回归方法：**



### Boosting ：Xgboost

**官方网址：（**[**https://xgboost.readthedocs.io/en/latest/python/python\_api.html#module-xgboost.sklearn**](https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn)**）**

**（**[**https://xgboost.readthedocs.io/en/latest/python/python\_api.html**](https://xgboost.readthedocs.io/en/latest/python/python_api.html)**）**

**（**[**https://xgboost.readthedocs.io/en/latest/parameter.html#general-parameters**](https://xgboost.readthedocs.io/en/latest/parameter.html#general-parameters)**）**

**例子：（**[**https://github.com/dmlc/xgboost/tree/master/demo/guide-python**](https://github.com/dmlc/xgboost/tree/master/demo/guide-python)**）**

**一些详解网站**

**（**[**https://www.cnblogs.com/lvpengbo/p/8822318.html**](https://www.cnblogs.com/lvpengbo/p/8822318.html)**）**

**（**[**http://www.cnblogs.com/Allen-rg/p/9266605.html**](http://www.cnblogs.com/Allen-rg/p/9266605.html)**）**

**#################################################################**

**回归，分类：Scikit-learn：**

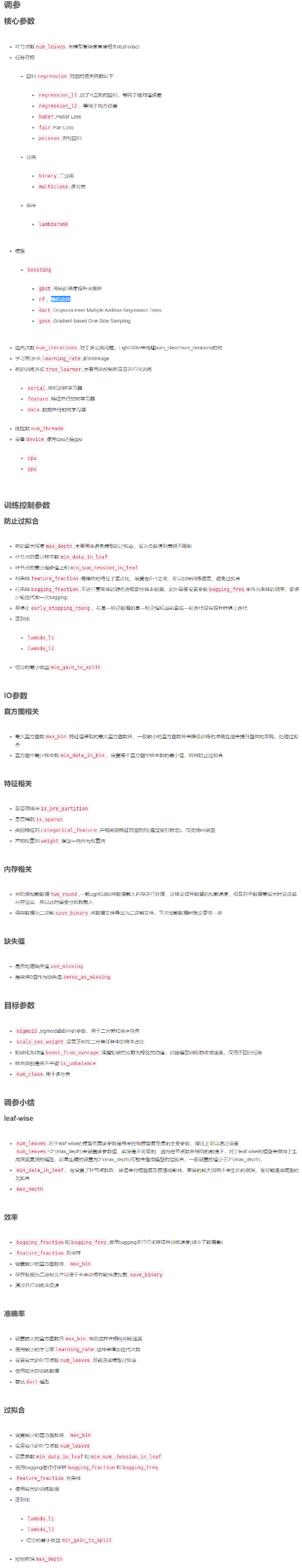
**（**[**https://xgboost.readthedocs.io/en/latest/python/python\_api.html#module-xgboost.sklearn**](https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn)**）**

### Boosting：Light GBM

**官网（**[**https://lightgbm.readthedocs.io/en/latest/**](https://lightgbm.readthedocs.io/en/latest/)**）**

**参数官网（**[**https://lightgbm.readthedocs.io/en/latest/Parameters.html#**](https://lightgbm.readthedocs.io/en/latest/Parameters.html)**）**

**Scikit-learn参数（**[**https://lightgbm.readthedocs.io/en/latest/Python-API.html#scikit-learn-api**](https://lightgbm.readthedocs.io/en/latest/Python-API.html#scikit-learn-api)**）**

****

### Boosting：Catboost

**官网（**[**https://tech.yandex.com/catboost/doc/dg/concepts/python-quickstart-docpage/**](https://tech.yandex.com/catboost/doc/dg/concepts/python-quickstart-docpage/)**）**

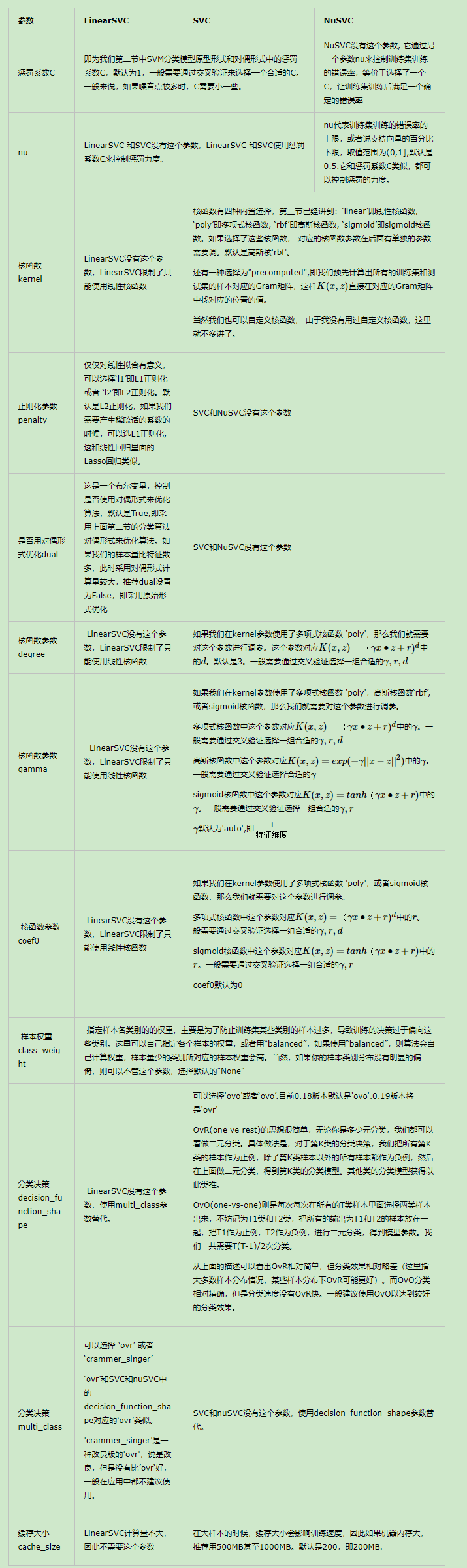
**实战（**[**https://blog.csdn.net/linxid/article/details/80723811**](https://blog.csdn.net/linxid/article/details/80723811)**）**



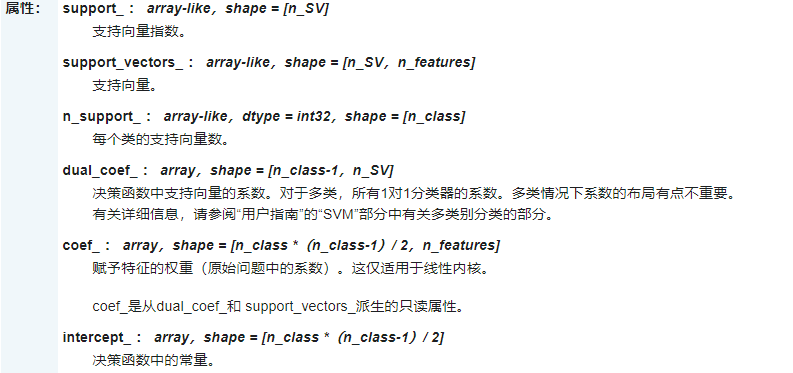
## SVM（SVC，SVR）

### 分类（SVC）

**（**[**https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)**）**

****

**重要属性：**

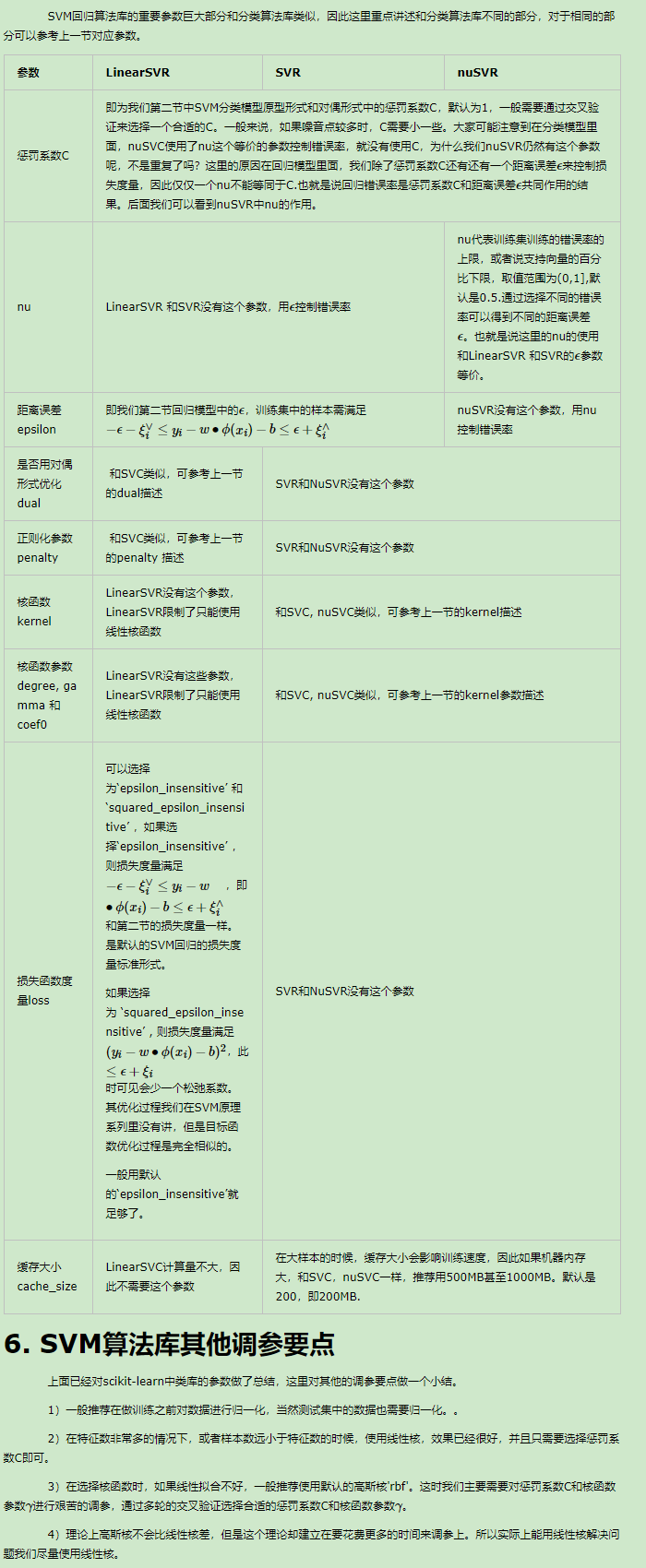


**重要方法：**



### 回归（SVR）

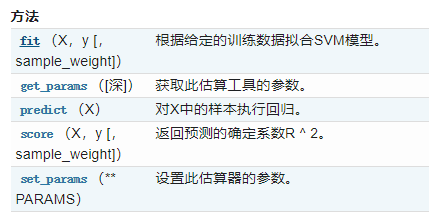
**（**[**https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR)**）**

****

**重要属性：**



**重要方法：**



## 贝叶斯（文本分类）

### 伯努利朴素贝叶斯（BNB）

class sklearn.naive\_bayes.**BernoulliNB**（alpha = 1.0，binarize = 0.0，fit\_prior = True，class\_prior = None ）

**（**[**https://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.BernoulliNB.html#sklearn.naive\_bayes.BernoulliNB**](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.BernoulliNB)**）**

**重要方法：**

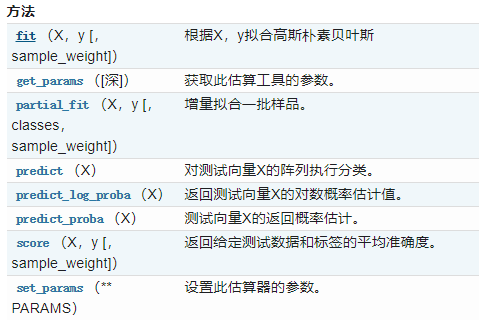


### 高斯朴素贝叶斯(GNB)

class sklearn.naive\_bayes.**GaussianNB**（priors = None，var\_smoothing = 1e-09 ）

**(**[**https://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.GaussianNB.html#sklearn.naive\_bayes.GaussianNB**](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB)**)**

**重要方法：**



### 多项式朴素贝叶斯（MNB）

class sklearn.naive\_bayes.**MultinomialNB**（alpha = 1.0，fit\_prior = True，class\_prior = None ）

**（**[**https://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.MultinomialNB.html#sklearn.naive\_bayes.MultinomialNB**](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB)**）**

**重要方法：**



## 聚类（Kmeans ，MiniBatchKMeans，密度聚类，谱聚类）

### Kmeans（均值聚类，传统Kmeans）

class sklearn.cluster.**KMeans**（n\_clusters = 8，init ='k-means ++'，n\_init = 10，max\_iter = 300，tol = 0.0001，precompute\_distances ='auto'，verbose = 0，random\_state = None，copy\_x = True，n\_jobs = None，algorithm = 'auto' ）

**（**[**https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans**](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans)**）**

**重点参数：**

KMeans类的主要参数有：

　　1) **n\_clusters**: 即我们的k值，一般需要多试一些值以获得较好的聚类效果。k值好坏的评估标准在下面会讲。

　　2）**max\_iter**： 最大的迭代次数，一般如果是凸数据集的话可以不管这个值，如果数据集不是凸的，可能很难收敛，此时可以指定最大的迭代次数让算法可以及时退出循环。

　　3）**n\_init：**用不同的初始化质心运行算法的次数。由于K-Means是结果受初始值影响的局部最优的迭代算法，因此需要多跑几次以选择一个较好的聚类效果，默认是10，一般不需要改。如果你的k值较大，则可以适当增大这个值。

　　4）**init：**即初始值选择的方式，可以为完全随机选择'random',优化过的'k-means++'或者自己指定初始化的k个质心。一般建议使用默认的'k-means++'。

　　5）**algorithm**：有“auto”, “full” or “elkan”三种选择。"full"就是我们传统的K-Means算法， “elkan”是我们原理篇讲的elkan K-Means算法。默认的"auto"则会根据数据值是否是稀疏的，来决定如何选择"full"和“elkan”。一般数据是稠密的，那么就是 “elkan”，否则就是"full"。一般来说建议直接用默认的"auto"

**重要属性：**

**cluster\_centers\_ : array, [n\_clusters, n\_features] ：**集群中心的坐标

**labels\_ :** 每个点的标签

**inertia\_ : float：**样本到其最近聚类中心的平方距离之和。

**n\_iter\_ : int：**运行的迭代次数。

**重要方法：**



### MiniBatchKMeans(MBM)

**（**class sklearn.cluster.**MiniBatchKMeans**（n\_clusters = 8，init ='k-means ++'，max\_iter = 100，batch\_size = 100，verbose = 0，compute\_labels = True，random\_state = None，tol = 0.0，max\_no\_improvement = 10，init\_size = None，n\_init = 3，reassignment\_ratio = 0.01 **）**

**（**[**https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html#sklearn.cluster.MiniBatchKMeans**](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html#sklearn.cluster.MiniBatchKMeans)**）**

**重要参数：**

　MiniBatchKMeans类的主要参数比KMeans类稍多，主要有：

　　1) **n\_clusters**: 即我们的k值，和KMeans类的n\_clusters意义一样。

　　2）**max\_iter：**最大的迭代次数， 和KMeans类的max\_iter意义一样。

　　3）**n\_init：**用不同的初始化质心运行算法的次数。这里和KMeans类意义稍有不同，KMeans类里的n\_init是用同样的训练集数据来跑不同的初始化质心从而运行算法。而MiniBatchKMeans类的n\_init则是每次用不一样的采样数据集来跑不同的初始化质心运行算法。

4）**batch\_size**：即用来跑Mini Batch KMeans算法的采样集的大小，默认是100.如果发现数据集的类别较多或者噪音点较多，需要增加这个值以达到较好的聚类效果。

　　5）**init：**即初始值选择的方式，和KMeans类的init意义一样。

　　6）**init\_size:**用来做质心初始值候选的样本个数，默认是batch\_size的3倍，一般用默认值就可以了。

　　7）**reassignment\_ratio:**某个类别质心被重新赋值的最大次数比例，这个和max\_iter一样是为了控制算法运行时间的。这个比例是占样本总数的比例，乘以样本总数就得到了每个类别质心可以重新赋值的次数。如果取值较高的话算法收敛时间可能会增加，尤其是那些暂时拥有样本数较少的质心。默认是0.01。如果数据量不是超大的话，比如1w以下，建议使用默认值。如果数据量超过1w，类别又比较多，可能需要适当减少这个比例值。具体要根据训练集来决定。

　　8）**max\_no\_improvement：**即连续多少个Mini Batch没有改善聚类效果的话，就停止算法， 和reassignment\_ratio，max\_iter一样是为了控制算法运行时间的。默认是10.一般用默认值就足够了。

**重要属性：**

**cluster\_centers\_ : array, [n\_clusters, n\_features]：**集群中心的坐标

**labels\_ :** 每个点的标签

**inertia\_ : float：**样本到其最近聚类中心的平方距离之和

**重要方法：**



### 密度聚类（DBSCAN）

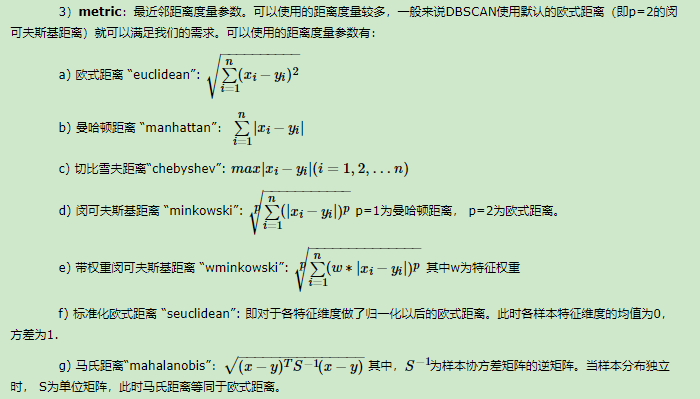
class sklearn.cluster.**DBSCAN**（eps = 0.5，min\_samples = 5，metric =' euclidean '，metric\_params = None，algorithm ='auto'，leaf\_size = 30，p = None，n\_jobs = None ）

**（**[**https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN**](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN)**）**

**重要参数：**

1**）eps**： DBSCAN算法参数，即我们的ϵϵ-邻域的距离阈值，和样本距离超过ϵϵ的样本点不在ϵϵ-邻域内。默认值是0.5.一般需要通过在多组值里面选择一个合适的阈值。eps过大，则更多的点会落在核心对象的ϵϵ-邻域，此时我们的类别数可能会减少， 本来不应该是一类的样本也会被划为一类。反之则类别数可能会增大，本来是一类的样本却被划分开。

**2）min\_samples：** DBSCAN算法参数，即样本点要成为核心对象所需要的ϵϵ-邻域的样本数阈值。默认值是5. 一般需要通过在多组值里面选择一个合适的阈值。通常和eps一起调参。在eps一定的情况下，min\_samples过大，则核心对象会过少，此时簇内部分本来是一类的样本可能会被标为噪音点，类别数也会变多。反之min\_samples过小的话，则会产生大量的核心对象，可能会导致类别数过少。



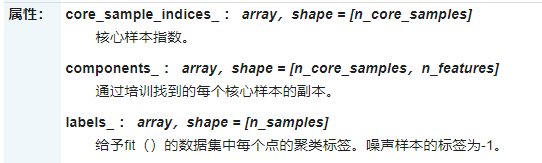
4）**algorithm**：最近邻搜索算法参数，算法一共有三种，第一种是蛮力实现，第二种是KD树实现，第三种是球树实现。这三种方法在[K近邻法(KNN)原理小结](http://www.cnblogs.com/pinard/p/6061661.html)中都有讲述，如果不熟悉可以去复习下。对于这个参数，一共有4种可选输入，‘brute’对应第一种蛮力实现，‘kd\_tree’对应第二种KD树实现，‘ball\_tree’对应第三种的球树实现， ‘auto’则会在上面三种算法中做权衡，选择一个拟合最好的最优算法。需要注意的是，如果输入样本特征是稀疏的时候，无论我们选择哪种算法，最后scikit-learn都会去用蛮力实现‘brute’。个人的经验，一般情况使用默认的 ‘auto’就够了。 如果数据量很大或者特征也很多，用"auto"建树时间可能会很长，效率不高，建议选择KD树实现‘kd\_tree’，此时如果发现‘kd\_tree’速度比较慢或者已经知道样本分布不是很均匀时，可以尝试用‘ball\_tree’。而如果输入样本是稀疏的，无论你选择哪个算法最后实际运行的都是‘brute’。

　　5）**leaf\_size**：最近邻搜索算法参数，为使用KD树或者球树时， 停止建子树的叶子节点数量的阈值。这个值越小，则生成的KD树或者球树就越大，层数越深，建树时间越长，反之，则生成的KD树或者球树会小，层数较浅，建树时间较短。默认是30. 因为这个值一般只影响算法的运行速度和使用内存大小，因此一般情况下可以不管它。

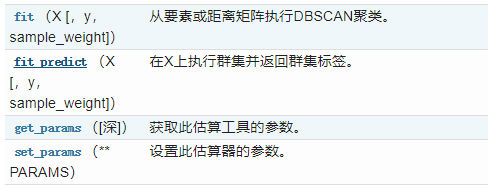
　　6） **p**: 最近邻距离度量参数。只用于闵可夫斯基距离和带权重闵可夫斯基距离中p值的选择，p=1为曼哈顿距离， p=2为欧式距离。如果使用默认的欧式距离不需要管这个参数。

　　　　以上就是DBSCAN类的主要参数介绍，其实需要调参的就是两个参数eps和min\_samples，这两个值的组合对最终的聚类效果有很大的影响。

**重要属性：**



**重要方法：**



### 谱聚类（SpectralClustering）

class sklearn.cluster.**SpectralClustering**（n\_clusters = 8，eigen\_solver = None，random\_state = None，n\_init = 10，gamma = 1.0，affinity =' rbf '，n\_neighbors = 10，eigen\_tol = 0.0，assign\_labels ='kmeans'，degree = 3，coef0 = 1，kernel\_params = None，n\_jobs = None ）

**（**[**https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html#sklearn.cluster.SpectralClustering**](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html#sklearn.cluster.SpectralClustering)**）**

**重要参数：**

1）**n\_clusters**：代表我们在对谱聚类切图时降维到的维数（原理篇第7节的k1k1），同时也是最后一步聚类算法聚类到的维数(原理篇第7节的k2k2)。也就是说scikit-learn中的谱聚类对这两个参数统一到了一起。简化了调参的参数个数。虽然这个值是可选的，但是一般还是推荐调参选择最优参数。

　　2) **affinity**: 也就是我们的相似矩阵的建立方式。可以选择的方式有三类，第一类是 'nearest\_neighbors'即K邻近法。第二类是'precomputed'即自定义相似矩阵。选择自定义相似矩阵时，需要自己调用set\_params来自己设置相似矩阵。第三类是全连接法，可以使用各种核函数来定义相似矩阵，还可以自定义核函数。最常用的是内置高斯核函数'rbf'。其他比较流行的核函数有‘linear’即线性核函数, ‘poly’即多项式核函数, ‘sigmoid’即sigmoid核函数。如果选择了这些核函数， 对应的核函数参数在后面有单独的参数需要调。自定义核函数我没有使用过，这里就不多讲了。affinity默认是高斯核'rbf'。一般来说，相似矩阵推荐使用默认的高斯核函数。

　　3) 核函数参数**gamma**: 如果我们在affinity参数使用了多项式核函数 'poly'，高斯核函数‘rbf’, 或者'sigmoid'核函数，那么我们就需要对这个参数进行调参。

**多项式核函数**中这个参数对应K(x,z)=（γx∙z+r)dK(x,z)=（γx∙z+r)d中的γγ。一般需要通过交叉验证选择一组合适的γ,r,dγ,r,d

**高斯核函数**中这个参数对应K(x,z)=exp(γ||x−z||2)K(x,z)=exp(γ||x−z||2)中的γγ。一般需要通过交叉验证选择合适的γγ

**sigmoid**核函数中这个参数对应K(x,z)=tanh（γx∙z+r)K(x,z)=tanh（γx∙z+r)中的γγ。一般需要通过交叉验证选择一组合适的γ,rγ,rγγ默认值为1.0，如果我们affinity使用'nearest\_neighbors'或者是'precomputed'，则这么参数无意义。

　　4）**核函数参数degree**：如果我们在affinity参数使用了多项式核函数 'poly'，那么我们就需要对这个参数进行调参。这个参数对应K(x,z)=（γx∙z+r)dK(x,z)=（γx∙z+r)d中的dd。默认是3。一般需要通过交叉验证选择一组合适的γ,r,dγ,r,d

　　5）**核函数参数coef0**: 如果我们在affinity参数使用了多项式核函数 'poly'，或者sigmoid核函数，那么我们就需要对这个参数进行调参。多项式核函数中这个参数对应K(x,z)=（γx∙z+r)dK(x,z)=（γx∙z+r)d中的rr。一般需要通过交叉验证选择一组合适的γ,r,dγ,r,dsigmoid核函数中这个参数对应K(x,z)=tanh（γx∙z+r)K(x,z)=tanh（γx∙z+r)中的rr。一般需要通过交叉验证选择一组合适的γ,rγ,rcoef0默认为1.

　　6）**kernel\_params**：如果affinity参数使用了自定义的核函数，则需要通过这个参数传入核函数的参数。

7 )**n\_neighbors**: 如果我们affinity参数指定为'nearest\_neighbors'即K邻近法，则我们可以通过这个参数指定KNN算法的K的个数。默认是10.我们需要根据样本的分布对这个参数进行调参。如果我们affinity不使用'nearest\_neighbors'，则无需理会这个参数。

　　8）**eigen\_solver**:1在降维计算特征值特征向量的时候，使用的工具。有 None, ‘arpack’, ‘lobpcg’, 和‘amg’4种选择。如果我们的样本数不是特别大，无需理会这个参数，使用''None暴力矩阵特征分解即可,如果样本量太大，则需要使用后面的一些矩阵工具来加速矩阵特征分解。它对算法的聚类效果无影响。

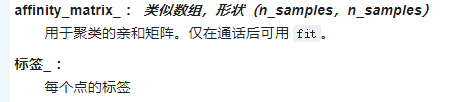
**9）eigen\_tol**：如果eigen\_solver使用了arpack’，则需要通过eigen\_tol指定矩阵分解停止条件。

**10）assign\_labels**：即最后的聚类方法的选择，有K-Means算法和 discretize算法两种算法可以选择。一般来说，默认的K-Means算法聚类效果更好。但是由于K-Means算法结果受初始值选择的影响，可能每次都不同，如果我们需要算法结果可以重现，则可以使用discretize。

**11）n\_init**：即使用K-Means时用不同的初始值组合跑K-Means聚类的次数，这个和K-Means类里面n\_init的意义完全相同，默认是10，一般使用默认值就可以。如果你的n\_clusters值较大，则可以适当增大这个值。

　　从上面的介绍可以看出，需要调参的部分除了最后的类别数**n\_clusters**，主要是相似矩阵**affinity**的选择，以及对应的相似矩阵参数。当我选定一个相似矩阵构建方法后，调参的过程就是对应的参数交叉选择的过程。对于K邻近法，需要对**n\_neighbors**进行调参，对于全连接法里面最常用的高斯核函数rbf，则需要对**gamma**进行调参。

**重要属性：**



**重要方法：**



## EM，GMM，HMM

### EM（期望最大化）

（EM算法解决这个的思路是使用启发式的迭代方法，既然我们无法直接求出模型分布参数，那么我们可以先猜想隐含数据（EM算法的E步），接着基于观察数据和猜测的隐含数据一起来极大化对数似然，求解我们的模型参数（EM算法的M步)。由于我们之前的隐藏数据是猜测的，所以此时得到的模型参数一般还不是我们想要的结果。不过没关系，我们基于当前得到的模型参数，继续猜测隐含数据（EM算法的E步），然后继续极大化对数似然，求解我们的模型参数（EM算法的M步)。以此类推，不断的迭代下去，直到模型分布参数基本无变化，算法收敛，找到合适的模型参数。

　　　　从上面的描述可以看出，EM算法是迭代求解最大值的算法，同时算法在每一次迭代时分为两步，E步和M步。一轮轮迭代更新隐含数据和模型分布参数，直到收敛，即得到我们需要的模型参数。

）

### GMM（高斯混合模型）

**from sklearn.mixture import GaussianMixture**

### HMM（隐马尔可夫）

**from hmmlearn import hmm**

　hmmlearn实现了三种HMM模型类，按照观测状态是连续状态还是离散状态，可以分为两类。GaussianHMM和GMMHMM是连续观测状态的HMM模型，而MultinomialHMM是离散观测状态的模型，也是我们在HMM原理系列篇里面使用的模型。

　　对于MultinomialHMM的模型，使用比较简单，"startprob\_"参数对应我们的隐藏状态初始分布ΠΠ, "transmat\_"对应我们的状态转移矩阵AA, "emissionprob\_"对应我们的观测状态概率矩阵BB。

　　对于连续观测状态的HMM模型，GaussianHMM类假设观测状态符合高斯分布，而GMMHMM类则假设观测状态符合混合高斯分布。一般情况下我们使用GaussianHMM即高斯分布的观测状态即可。以下对于连续观测状态的HMM模型，我们只讨论GaussianHMM类。

　　在GaussianHMM类中，"startprob\_"参数对应我们的隐藏状态初始分布ΠΠ, "transmat\_"对应我们的状态转移矩阵AA, 比较特殊的是观测状态概率的表示方法，此时由于观测状态是连续值，我们无法像MultinomialHMM一样直接给出矩阵BB。而是采用给出各个隐藏状态对应的观测状态高斯分布的概率密度函数的参数。

　　如果观测序列是一维的，则观测状态的概率密度函数是一维的普通高斯分布。如果观测序列是NN维的，则隐藏状态对应的观测状态的概率密度函数是NN维高斯分布。高斯分布的概率密度函数参数可以用μμ表示高斯分布的期望向量，ΣΣ表示高斯分布的协方差矩阵。在GaussianHMM类中，“means”用来表示各个隐藏状态对应的高斯分布期望向量μμ形成的矩阵，而“covars”用来表示各个隐藏状态对应的高斯分布协方差矩阵ΣΣ形成的三维张量。

## 主题模型：（LDA隐含狄利克雷分布（LDA））

class sklearn.decomposition.**LatentDirichletAllocation**（n\_components = 10，doc\_topic\_prior = None，topic\_word\_prior = None，learning\_method ='batch'，learning\_decay = 0.7，learning\_offset = 10.0，max\_iter = 10，batch\_size = 128，evaluate\_every = -1，total\_samples = 1000000.0，perp\_tol = 0.1，mean\_change\_tol = 0.001，max\_doc\_update\_iter = 100，n\_jobs =无，verbose = 0，random\_state =无，n\_topics =无）

**（**[**https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html#sklearn.decomposition.LatentDirichletAllocation**](https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html#sklearn.decomposition.LatentDirichletAllocation)**）**

**主要参数：**

　我们来看看LatentDirichletAllocation类的主要输入参数:

　　1) **n\_topics:**即我们的隐含主题数KK,需要调参。KK的大小取决于我们对主题划分的需求，比如我们只需要类似区分是动物，植物，还是非生物这样的粗粒度需求，那么KK值可以取的很小，个位数即可。如果我们的目标是类似区分不同的动物以及不同的植物，不同的非生物这样的细粒度需求，则KK值需要取的很大，比如上千上万。此时要求我们的训练文档数量要非常的多。

　　2) **doc\_topic\_prior:**即我们的文档主题先验Dirichlet分布θdθd的参数αα。一般如果我们没有主题分布的先验知识，可以使用默认值1/K1/K。

　　3) **topic\_word\_prior:**即我们的主题词先验Dirichlet分布βkβk的参数ηη。一般如果我们没有主题分布的先验知识，可以使用默认值1/K1/K。

　　4) **learning\_method:**即LDA的求解算法。有 ‘batch’ 和 ‘online’两种选择。 ‘batch’即我们在原理篇讲的变分推断EM算法，而"online"即在线变分推断EM算法，在"batch"的基础上引入了分步训练，将训练样本分批，逐步一批批的用样本更新主题词分布的算法。默认是"online"。选择了‘online’则我们可以在训练时使用partial\_fit函数分布训练。不过在scikit-learn 0.20版本中默认算法会改回到"batch"。建议样本量不大只是用来学习的话用"batch"比较好，这样可以少很多参数要调。而样本太多太大的话，"online"则是首先了。

　　5）**learning\_decay：**仅仅在算法使用"online"时有意义，取值最好在(0.5, 1.0]，以保证"online"算法渐进的收敛。主要控制"online"算法的学习率，默认是0.7。一般不用修改这个参数。

　　6）**learning\_offset：**仅仅在算法使用"online"时有意义，取值要大于1。用来减小前面训练样本批次对最终模型的影响。

　　7） **max\_iter** ：EM算法的最大迭代次数。

　　8）**total\_samples：**仅仅在算法使用"online"时有意义， 即分步训练时每一批文档样本的数量。在使用partial\_fit函数时需要。

　　9）**batch\_size:**仅仅在算法使用"online"时有意义， 即每次EM算法迭代时使用的文档样本的数量。

　　10）**mean\_change\_tol** :即E步更新变分参数的阈值，所有变分参数更新小于阈值则E步结束，转入M步。一般不用修改默认值。

　　　　11） **max\_doc\_update\_iter:**即E步更新变分参数的最大迭代次数，如果E步迭代次数达到阈值，则转入M步。

　　　　从上面可以看出，如果learning\_method使用"batch"算法，则需要注意的参数较少，则如果使用"online",则需要注意"learning\_decay", "learning\_offset"，“total\_samples”和“batch\_size”等参数。无论是"batch"还是"online", n\_topics(KK), doc\_topic\_prior(αα), topic\_word\_prior(ηη)都要注意。如果没有先验知识，则主要关注与主题数KK。可以说，主题数KK是LDA主题模型最重要的超参数。

**重要属性：**

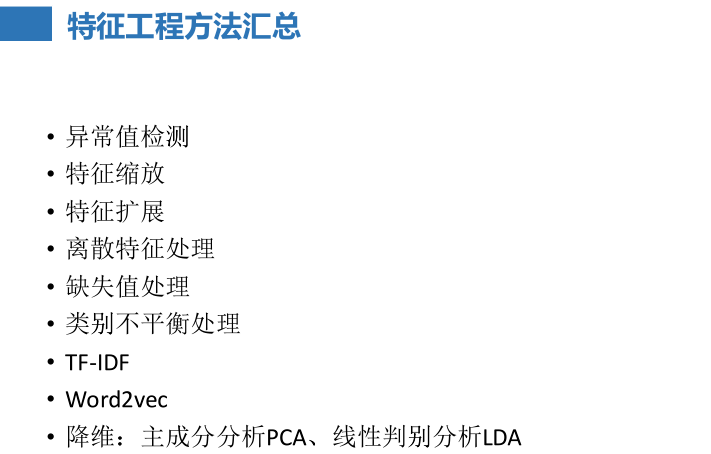
**components\_：相关的重要词。**



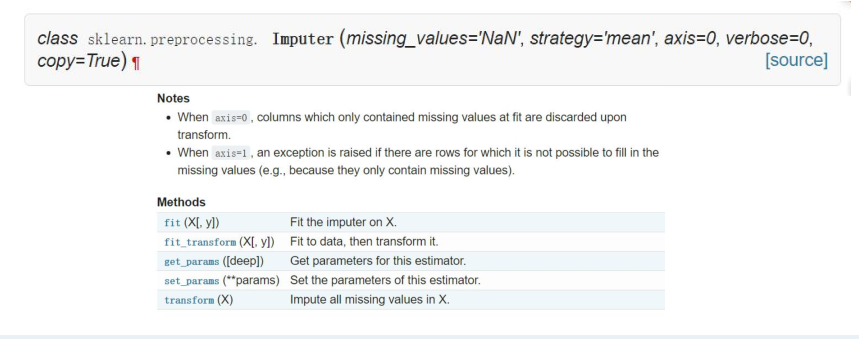
**重要方法：**

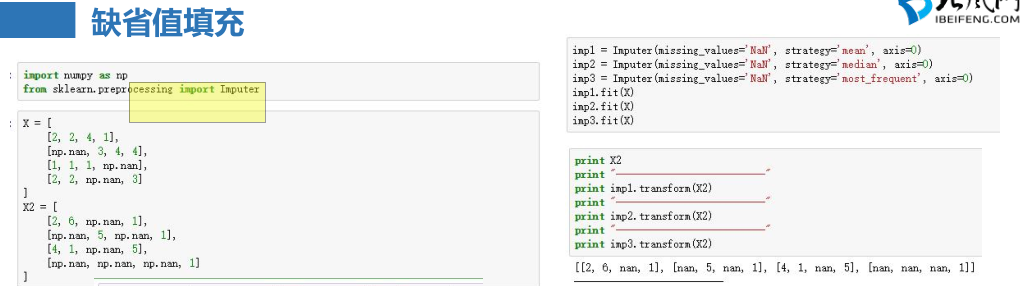


## 特征工程

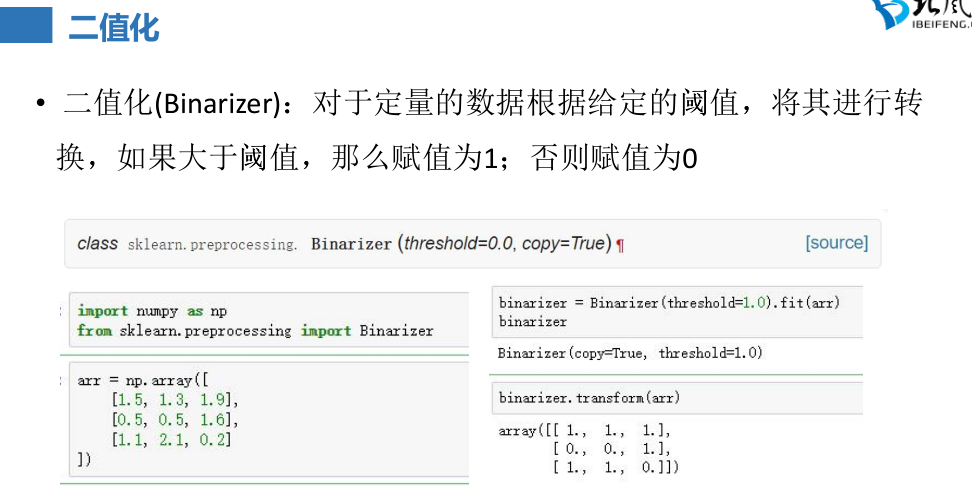


### 缺省值填充：

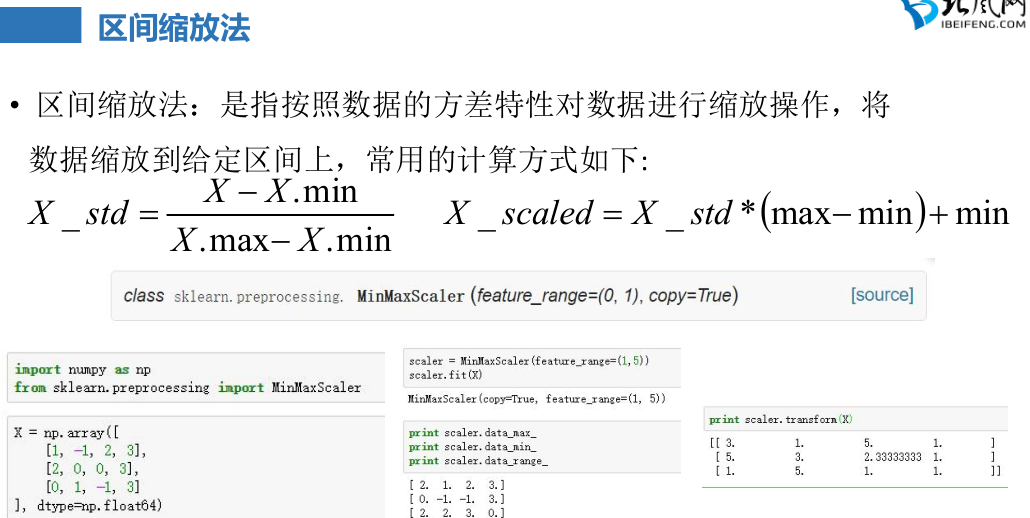




### 二值化



### 区间缩放（归一化）：

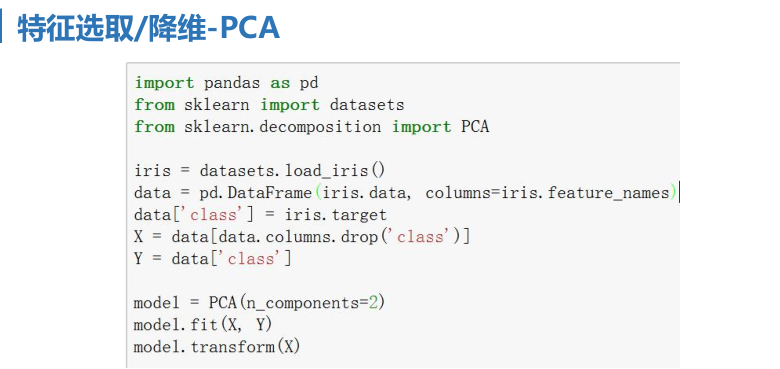


### 规范化：



### PCA（特征降维，无监督）（运用了SVD算法）

**scikit-learn的PCA算法的背后真正的实现就是用的SVD，而不是我们我们认为的暴力特征分解。**



### LDA（线性判别分析LDA）

**有监督：（投影后类内方差最小，类间方差最大）**

**在scikit-learn中， LDA类是sklearn.discriminant\_analysis.LinearDiscriminantAnalysis。那既可以用于分类又可以用于降维。当然，应用场景最多的还是降维。和PCA类似，LDA降维基本也不用调参，只需要指定降维到的维数即可。**

**我们这里对LinearDiscriminantAnalysis类的参数做一个基本的总结。**

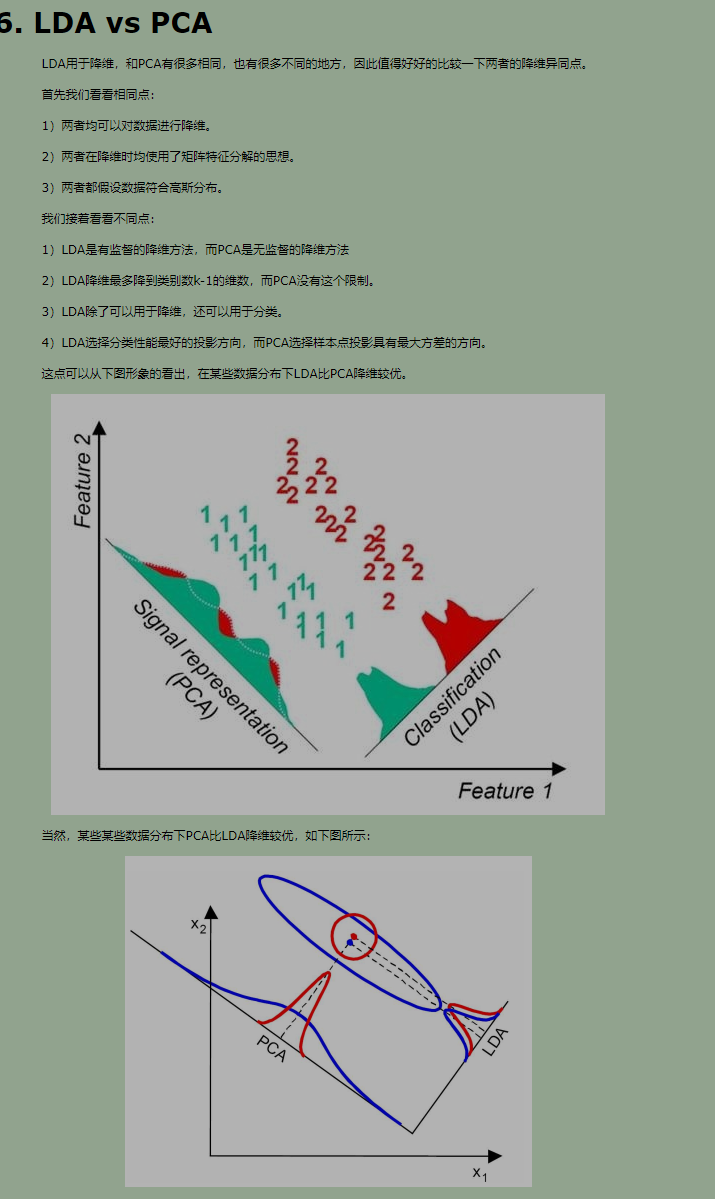
**1）solver :** 即求LDA超平面特征矩阵使用的方法。可以选择的方法有奇异值分解"svd"，最小二乘"lsqr"和特征分解"eigen"。一般来说特征数非常多的时候推荐使用svd，而特征数不多的时候推荐使用eigen。主要注意的是，如果使用svd，则不能指定正则化参数shrinkage进行正则化。默认值是svd

**2）shrinkage：**正则化参数，可以增强LDA分类的泛化能力。如果仅仅只是为了降维，则一般可以忽略这个参数。默认是None，即不进行正则化。可以选择"auto",让算法自己决定是否正则化。当然我们也可以选择不同的[0,1]之间的值进行交叉验证调参。注意shrinkage只在solver为最小二乘"lsqr"和特征分解"eigen"时有效。

**3）priors ：**类别权重，可以在做分类模型时指定不同类别的权重，进而影响分类模型建立。降维时一般不需要关注这个参数。

**4）n\_components：**即我们进行LDA降维时降到的维数。在降维时需要输入这个参数。注意只能为[1,类别数-1)范围之间的整数。如果我们不是用于降维，则这个值可以用默认的None。

　　从上面的描述可以看出，如果我们只是为了降维，则只需要输入n\_components,注意这个值必须小于“类别数-1”。PCA没有这个限制。

****