**机器学习好伙伴之scikit-learn的使用——常用模型及其方法**

**机器学习好伙伴之scikit-learn的使用——常用模型及其方法**

[**机器学习**](https://so.csdn.net/so/search?q=%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0&spm=1001.2101.3001.7020) **好伙伴之scikit-learn的使用——常用模型及其方法**

* 1、模型的常用方法
* 2、sklearn中机器学习模型的实现
* 2.1、线性回归
* 2.2、逻辑回归
* 2.3、朴素贝叶斯
* 2.4、决策树
* 2.5、随机森林
* 2.6、SVM(支持向量机）
* 2.7、KNN（K-近邻算法）
* 2.8、adaboost

[sklearn](https://so.csdn.net/so/search?q=sklearn&spm=1001.2101.3001.7020) 中还存在许多不同的机器学习模型可以直接调用，相比于自己撰写代码，直接使用sklearn的模型可以大大提高效率。

**1、模型的常用方法**

sklearn中所有的模型都有四个固定且常用的方法，分别是model.fit、model.predict、model.get\_params、model.score。

|  |
| --- |
| Python # 用于模型训练 model.fit(X\_train, y\_train) |

|  |
| --- |
| Python # 用于模型预测 model.predict(X\_test) |

|  |
| --- |
| Python # 获得模型参数 model.get\_params() |

|  |
| --- |
| Python # 进行模型打分 model.score(X\_test, y\_test) |

**2、sklearn中机器学习模型的实现**

**2.1、** [**线性**](https://so.csdn.net/so/search?q=%E7%BA%BF%E6%80%A7&spm=1001.2101.3001.7020) **回归**

sklearn中线性回归使用最小二乘法实现，使用起来非常简单。  
线性回归是回归问题，score使用R2系数作为评价标准。  
该方法通过调用如下函数实现。

|  |
| --- |
| Python from sklearn.linear\_model import LinearRegression model = LinearRegression(fit\_intercept=True,normalize=False,copy\_X=True,n\_jobs=1) |

其主要使用的参数为：  
**1、fit\_intercept：是否计算截距。**  
**2、normalize：当其为False时，该参数将被忽略。 当其为True时，则按照一定规律归一化。**  
**3、copy\_X：是否对X数组进行复制。**  
**4、n\_jobs：指定线程数**  
应用方式如下：

|  |
| --- |
| Python # 载入数据集 from sklearn import datasets from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression import numpy as np  # 生成数据库 data\_X, data\_Y = datasets.make\_regression(n\_samples=100, n\_features=1, n\_targets=1, noise=5) # 对数据库进行划分 X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_X, data\_Y, test\_size=0.3)  # 进行线性训练 model = LinearRegression(fit\_intercept=True,normalize=False,copy\_X=True,n\_jobs=1) model.fit(X\_train,y\_train)  # 预测，比较结果 print(model.score(X\_test,y\_test)) |

实验结果为：

|  |
| --- |
| Python 0.9948848129693583 |

**2.2、** [**逻辑回归**](https://so.csdn.net/so/search?q=%E9%80%BB%E8%BE%91%E5%9B%9E%E5%BD%92&spm=1001.2101.3001.7020)

logistic回归是一种广义线性回归，可以用于计算概率。  
即线性回归用于计算回归，逻辑回归用于分类。

|  |
| --- |
| Python from sklearn.linear\_model import LogisticRegression model = LogisticRegression(penalty='l2',dual=False,C=1.0,n\_jobs=1,random\_state=20) |

其主要使用的参数为：  
**1、penalty：使用指定正则化项，可以指定为’l1’或者’l2’，L1正则化可以抵抗共线性，还会起到特征选择的作用，不重要的特征系数将会变为0；L2正则化一般不会将系数变为0，但会将不重要的特征系数变的很小，起到避免过拟合的作用。**  
**2、C：正则化强度取反，值越小正则化强度越大**  
**3、n\_jobs: 指定线程数**  
**4、random\_state：随机数生成器**  
应用方式如下：

|  |
| --- |
| Python # 载入数据集 from sklearn import datasets from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression import numpy as np  # 获取自带的数据库 data\_X, data\_Y = datasets.make\_classification(  n\_samples=300, n\_features=2,  n\_redundant=0, n\_informative=2,   random\_state=22, n\_clusters\_per\_class=2,  scale=100) # 对自带的数据库进行划分 X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_X, data\_Y, test\_size=0.3)  # 建立逻辑线性模型 model = LogisticRegression(penalty='l2',dual=False,C=1.0,n\_jobs=1,random\_state=20) # 对模型进行训练 model.fit(X\_train,y\_train) # 预测，比较结果 print(model.score(X\_test,y\_test)) |

实验结果为：

|  |
| --- |
| Python 0.9555555555555556 |

**2.3、** [**朴素贝叶斯**](https://so.csdn.net/so/search?q=%E6%9C%B4%E7%B4%A0%E8%B4%9D%E5%8F%B6%E6%96%AF&spm=1001.2101.3001.7020)

贝叶斯分类均以贝叶斯定理为基础，故统称为贝叶斯分类。  
而朴素贝叶斯分类是贝叶斯分类中最常用、简单的一种分类方法

|  |
| --- |
| Python import sklearn.naive\_bayes as bayes # 伯努利分布的朴素贝叶斯 model = bayes.BernoulliNB(alpha=1.0,binarize=0.0,fit\_prior=True,class\_prior=None)  # 高斯分布的朴素贝叶斯 model = bayes.GaussianNB() |

其主要使用的参数为：  
**1、alpha：平滑参数**  
**2、fit\_prior：是否要学习类的先验概率；false-使用统一的先验概率**  
**3、class\_prior：是否指定类的先验概率；若指定则不能根据参数调整**  
**4、binarize：二值化的阈值。**  
应用方式如下：

|  |
| --- |
| Python # 载入数据集 from sklearn import datasets from sklearn.model\_selection import train\_test\_split import sklearn.naive\_bayes as bayes import numpy as np  # 生成数据库 data\_X, data\_Y = datasets.make\_classification(  n\_samples=300, n\_features=2,  n\_redundant=0, n\_informative=2,   random\_state=22, n\_clusters\_per\_class=2,  scale=100) # 对数据库进行划分  X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_X, data\_Y, test\_size=0.3)  # 建立贝叶斯模型 # 伯努利分布的朴素贝叶斯 model = bayes.BernoulliNB(alpha=1.0,binarize=0.0,fit\_prior=True,class\_prior=None)  # 高斯分布的朴素贝叶斯 #model = bayes.GaussianNB() # 对模型进行训练 model.fit(X\_train,y\_train) # 预测，比较结果 print(model.score(X\_test,y\_test)) |

实验结果为：

|  |
| --- |
| Python 0.9333333333333333 |

**2.4、** [**决策树**](https://so.csdn.net/so/search?q=%E5%86%B3%E7%AD%96%E6%A0%91&spm=1001.2101.3001.7020)

决策树使用二叉树帮助完成分类或者回归，是一种非常实用的算法。

|  |
| --- |
| Python from sklearn.tree import DecisionTreeClassifier model = DecisionTreeClassifier(  criterion='entropy',  max\_depth=None,min\_samples\_split=2,  min\_samples\_leaf=1,max\_features=None  ) |

其主要使用的参数为：  
**1、criterion：采用gini还是entropy进行特征选择**  
**2、max\_depth：树的最大深度**  
**3、min\_samples\_split：内部节点分裂所需要的最小样本数量**  
**4、min\_samples\_leaf：叶子节点所需要的最小样本数量**  
**5、max\_features：寻找最优分割点时的最大特征数**  
应用方式如下：

|  |
| --- |
| Python # 载入数据集 # 载入数据集 from sklearn import datasets from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier from sklearn.tree import DecisionTreeRegressor import numpy as np  # 决策树用于分类 # 生成数据库 data\_X, data\_Y = datasets.make\_classification(  n\_samples=300, n\_features=2,  n\_redundant=0, n\_informative=2,   random\_state=22, n\_clusters\_per\_class=2,  scale=100) # 对数据库进行划分  X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_X, data\_Y, test\_size=0.3)  # 建立决策树模型 model = DecisionTreeClassifier(  criterion='entropy',  max\_depth=None,min\_samples\_split=2,  min\_samples\_leaf=1,max\_features=None  ) # 对模型进行训练 model.fit(X\_train,y\_train) # 预测，比较结果 print(model.score(X\_test,y\_test))    # 决策树用于回归 # 生成数据库 data\_X, data\_Y = datasets.make\_regression(n\_samples=100, n\_features=1, n\_targets=1, noise=5) # 对数据库进行划分 X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_X, data\_Y, test\_size=0.3)  # 建立决策树模型 model = DecisionTreeRegressor(  max\_depth=None,min\_samples\_split=2,  min\_samples\_leaf=1,max\_features=None  ) # 对模型进行训练 model.fit(X\_train,y\_train)  # 预测，比较结果 print(model.score(X\_test,y\_test)) |

实验结果为：

|  |
| --- |
| Python 0.9333333333333333 0.9896667209630251 |

**2.5、随机森林**

随机森林是一堆树的集合，最终结果取平均得到预测结果。

|  |
| --- |
| Python from sklearn.ensemble import RandomForestClassifier  model = RandomForestClassifier(oob\_score=True) |

其主要使用的参数为：  
**1、n\_estimators：森林中树的数量，默认是10棵，如果资源足够可以多设置一些。**  
**2、max\_features：寻找最优分隔的最大特征数，默认是"auto"。**  
**3、max\_ depth：树的最大深度。**  
**4、min\_ samples\_split：树中一个节点所需要用来分裂的最少样本数，默认是2。**  
**5、min\_ samples\_leaf：树中每个叶子节点所需要的最少的样本数。**  
应用方式如下：

|  |
| --- |
| Python # 载入数据集 from sklearn import datasets from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier  import numpy as np  # 生成数据库 data\_X, data\_Y = datasets.make\_classification(  n\_samples=300, n\_features=2,  n\_redundant=0, n\_informative=2,   random\_state=22, n\_clusters\_per\_class=2,  scale=100) # 对数据库进行划分  X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_X, data\_Y, test\_size=0.3)  # 建立随机森林模型 model = RandomForestClassifier(n\_estimators = 10,oob\_score = True)  # 对模型进行训练 model.fit(X\_train,y\_train) # 预测，比较结果 print(model.score(X\_test,y\_test)) |

实验结果为：

|  |
| --- |
| Python 0.9666666666666667 |

**2.6、** [**SVM**](https://so.csdn.net/so/search?q=SVM&spm=1001.2101.3001.7020) **(支持向量机）**

[支持向量机](https://so.csdn.net/so/search?q=%E6%94%AF%E6%8C%81%E5%90%91%E9%87%8F%E6%9C%BA&spm=1001.2101.3001.7020) 可以解决分类问题，SVM的关键在于核函数，其作用为将低位向量映射到高维空间里，使得其变得线性可分。

|  |
| --- |
| Python from sklearn.svm import SVC model = SVC(C = 1,kernel='linear') |

其主要使用的参数为：  
**1、C：误差项的惩罚系数**  
**2、kernel：核函数，默认：rbf(高斯核函数)，可选择的对象为：‘linear’,‘poly’,‘sigmoid’,‘precomputed’。**  
应用方式如下：

|  |
| --- |
| Python # 载入数据集 from sklearn import datasets from sklearn.model\_selection import train\_test\_split from sklearn.svm import SVC import numpy as np  # 生成数据库 data\_X, data\_Y = datasets.make\_classification(  n\_samples=300, n\_features=2,  n\_redundant=0, n\_informative=2,   random\_state=21, n\_clusters\_per\_class=2,  scale=100) # 对数据库进行划分  X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_X, data\_Y, test\_size=0.3)  # 建立SVM模型 model = SVC(C = 1,kernel='linear') # 对模型进行训练 model.fit(X\_train,y\_train) # 预测，比较结果 print(model.score(X\_test,y\_test)) |

实验结果为：

|  |
| --- |
| Python 0.9444444444444444 |

**2.7、KNN（K-近邻算法）**

KNN非常好用，由于其工作特点，甚至不需要训练就可以得到非常好的分类效果。

|  |
| --- |
| Python from sklearn.neighbors import KNeighborsClassifier model = KNeighborsClassifier(n\_neighbors=5,n\_jobs=1) |

其主要使用的参数为：  
**1、n\_neighbors： 使用邻居的数目**  
**2、n\_jobs：线程数**  
应用方式如下：

|  |
| --- |
| Python # 载入数据集 from sklearn import datasets from sklearn.model\_selection import train\_test\_split from sklearn.neighbors import KNeighborsClassifier import numpy as np  # 生成数据库 data\_X, data\_Y = datasets.make\_classification(  n\_samples=300, n\_features=2,  n\_redundant=0, n\_informative=2,   random\_state=21, n\_clusters\_per\_class=2,  scale=100) # 对数据库进行划分  X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_X, data\_Y, test\_size=0.3)  # 建立KNN模型 model = KNeighborsClassifier(n\_neighbors=5,n\_jobs=1) # 对模型进行训练 model.fit(X\_train,y\_train) # 预测，比较结果 print(model.score(X\_test,y\_test)) |

实验结果为：

|  |
| --- |
| Python 1.0 |

**2.8、adaboost**

adaboost是一种由弱分类器构成的强分类器，即针对同一个训练集训练不同的分类器（弱分类器），然后把这些弱分类器集合起来，构成一个更强的最终分类器（强分类器）。

|  |
| --- |
| Python from sklearn.ensemble import AdaBoostClassifier  AdaBoostClassifier(n\_estimators=100,learning\_rate=0.1) |

其主要使用的参数为：  
**1、n\_estimators： 弱分类器的数量**  
**2、learning\_rate：学习率**  
应用方式如下：

|  |
| --- |
| Python # 载入数据集 from sklearn import datasets from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import AdaBoostClassifier  import numpy as np  # 生成数据库 data\_X, data\_Y = datasets.make\_classification(  n\_samples=300, n\_features=2,  n\_redundant=0, n\_informative=2,   random\_state=21, n\_clusters\_per\_class=2,  scale=100) # 对数据库进行划分  X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_X, data\_Y, test\_size=0.3)  # 建立AdaBoost模型 model = AdaBoostClassifier(n\_estimators=100,learning\_rate=0.1) # 对模型进行训练 model.fit(X\_train,y\_train) # 预测，比较结果 print(model.score(X\_test,y\_test)) |

实验结果为：

|  |
| --- |
| Python 0.9888888888888889 |