|  |  |  |  |
| --- | --- | --- | --- |
| Xgboost.XGBClassifier | | | |
| **参数名** | **API解释** | **Sklearn默认值** | **经验值** |
| **max\_depth** | 树的深度 | 3 | 3-10 对结果影响较大 |
| **learning\_rate** | 学习率，控制每次迭代更新权重时的步长 | 0.1 | 0.01-0.2值越小，训练越慢。 |
| **n\_estimators** | 总共迭代的次数，即决策树的个数 | 100 |  |
| objective | 这个参数定义需要被最小化的损失函数。最常用的值有：  binary:logistic 二分类的逻辑回归，返回预测的概率(不是类别)。  multi:softmax 使用softmax的多分类器，返回预测的类别(不是概率)。  在这种情况下，你还需要多设一个参数：num\_class(类别数目)。  multi:softprob 和multi:softmax参数一样，但是返回的是每个数据属于各个类别的概率。 | Binary：logistic |  |
| booster | 选择每次迭代的模型，有两种选择：  gbtree：基于树的模型  gbliner：线性模型 | gbtree | gbtree |
| **gamma** | 在节点分裂时，只有分裂后损失函数的值下降了，才会分裂这个节点。Gamma指定了节点分裂所需的最小损失函数下降值。 | 0 | 0-5 |
| **min\_child\_weight** | 决定最小叶子节点样本权重和。  和GBM的 min\_child\_leaf 参数类似，但不完全一样。XGBoost的这个参数是最小样本权重的和，而GBM参数是最小样本总数。 | 0 | 1-10  对结果影响较大  值越大，越容易欠拟合；  值越小，越容易过拟合 |
| **subsample** | 这个参数控制对于每棵树，随机采样的比例。  减小这个参数的值，算法会更加保守，避免过拟合。但是，如果这个值设置得过小，它可能会导致欠拟合 | 0 | 0.5-1  防止overfitting |
| **colsample\_bytree** | 和GBM里面的max\_features参数类似。用来控制每棵随机采样的列数的占比。 | 0 | 0.5-1防止overfitting |
| colsample\_bylevel | 用来控制树的每一级的每一次分裂，对列数的采样的占比。 | 1 | 基本不调节 |
| reg\_lambda | L2正则化系数 | 1 | 大多数情况下未调参0.01,0.1,1,10,100 |
| reg\_alpha | L1正则化系数 | 1 | 大多数情况下未调参0.01,0.1,1,10,100 |
| **scale\_pos\_weight** | 正样本的权重，在二分类任务中，当正负样本比例失衡时，设置正样本的权重，模型效果更好。例如，当正负样本比例为1:10时，scale\_pos\_weight=10。 | 1 |  |
| **eval\_metric** | 对于有效数据的度量方法。  对于回归问题，默认值是rmse，对于分类问题，默认值是error。  典型值有：  rmse 均方根误差  mae 平均绝对误差  logloss 负对数似然函数值  error 二分类错误率(阈值为0.5)  merror 多分类错误率  mlogloss 多分类logloss损失函数  auc 曲线下面积 |  | 默认值取决于objective参数的取值 |
|  | http://xgboost.readthedocs.io/en/latest/python/python\_api.html#module-xgboost.training |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| svm | | | |
| 参数名 | API解释 | Sklearn默认值 | 经验值 |
| **c** | 错误项的惩罚系数。C越大，即对分错样本的惩罚程度越大，因此在训练样本中准确率越高，但是泛化能力降低，也就是对测试数据的分类准确率降低。相反，减小C的话，容许训练样本中有一些误分类错误样本，泛化能力强。 | 1.0 | 0.01,0.1,1,10,100,200 |
| **kernel** | ‘linear’:线性核函数  ‘poly’：多项式核函数  ‘rbf’：径像核函数/高斯核  ‘sigmod’:sigmod核函数  ‘precomputed’:核矩阵  precomputed表示自己提前计算好核函数矩阵，这时候算法内部就不再用核函数去计算核矩阵，而是直接用你给的核矩阵。核矩阵为如下形式：  如果如果特征数远远大于样本数的情况下,使用线性核就可以了.  如果特征数和样本数都很大,一般使用线性核  如果特征数远小于样本数,这种情况一般使用RBF. | ‘rbf’ | ‘rbf’ |
| **degree** | 这个参数只对多项式核函数有用，是指多项式核函数的阶数n  如果给的核函数参数是其他核函数，则会自动忽略该参数。 | 3 | 3,4,5,6, |
| **gamma** | 核函数系数，只对‘rbf’,‘poly’,‘sigmod’有效。  如果gamma为auto，代表其值为样本特征数的倒数，即1/n\_features. | ’auto’  1/n\_features | 0.001,0.1,1,10,100,1000  太小会导致过拟合 |
| **coef0** | 核函数中的独立项，只有对‘poly’和‘sigmod’核函数有用 | 0.0 |  |
| probability | 是否启用概率估计。 | False |  |
| shrinking | 是否采用启发式收缩方式 | True |  |
| tol | svm停止训练的误差精度 | 1e-3 |  |
| **class\_weight** | 给每个类别分别设置不同的惩罚参数C，如果没有给，则会给所有类别都给C=1，即前面参数指出的参数C.  如果给定参数‘balanced’，则使用y的值自动调整与输入数据中的类频率成反比的权重。 | 1 |  |
| **max\_iter** | 最大迭代次数，如果为-1，表示不限制 | -1 |  |
|  | http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| LR | | | |
| 参数名 | API解释 | Sklearn默认值 | 经验值 |
| **penalty** | 在调参时如果我们主要的目的只是为了解决过拟合，一般penalty选择L2正则化就够了。但是如果选择L2正则化发现还是过拟合，即预测效果差的时候，就可以考虑L1正则化。另外，如果模型的特征非常多，我们希望一些不重要的特征系数归零，从而让模型系数稀疏化的话，也可以使用L1正则化。 | False | ‘l2’ |
| dual | Dual formulation is only implemented for l2 penalty with liblinear solver | False |  |
| tol | 容许停止标准，即我们说的要迭代停止所需达到的精度要求。 | 1e-4 |  |
| **C** | Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization. | 1.0 |  |
| fit\_intercept | 指定是否应将常数（又称偏差或截距）添加到决策函数中，相当于是否加入截距项b。 | True |  |
| **class\_weight** | dict or ‘balanced’ | None |  |
| **solver** | {‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}，优化拟合参数算法选择，默认为liblinear。  liblinear：使用坐标轴下降法（西瓜书408.B5）来迭代优化损失函数。  sag：即随机平均梯度下降，类似于我们的stocGradAscent1函数，思想是常用的一阶优化方法，是求解无约束优化问题最经典，最简单的方法之一。  saga：线性收敛的随机优化算法。  newton-cg：牛顿法，sag方法使用一阶导数，而牛顿法采用了二阶泰勒展开，这样缩减了迭代轮数，但是 需要计算Hsssian矩阵的逆，所以计算复杂度较高。  lbfgs：拟牛顿法，考虑到牛顿法的Hessian矩阵求逆太过复杂，尤其在高维问题中几乎不可行，想到了用较低的代价寻找Hessian矩阵的近似逆矩阵，便有了拟牛顿法。 | ‘liblinear’ |  |
| max\_iter | 最大迭代次数 | 100 |  |
|  | http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LogisticRegression.html |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **KMEANS** | | | |
| **参数名** | API解释 | Sklearn默认值 | 经验值 |
| **n\_clusters** | 簇的个数，即你想聚成几类 | 8 |  |
| **init='k-means++'** | 初始簇中心的获取方法 | init='k-means++' |  |
| **n\_init** | 获取初始簇中心的更迭次数，为了弥补初始质心的影响，算法默认会初始10次质心 | 10 |  |
| **max\_iter** | max\_iter=300,最大迭代次数（因为kmeans算法的实现需要迭代） | 300 |  |
| **tol** | 容忍度，即kmeans运行准则收敛的条件 | tol=1e-4 |  |
| **precompute\_distances** | 是否需要提前计算距离，这个参数会在空间和时间之间做权衡，如果是True 会把整个距离矩阵都放到内存中，auto 会默认在数据样本大于featurs\*samples 的数量大于12e6 的时候False, False 时核心实现的方法是利用Cpython 来实现的 | auto |  |
| **verbose** | 冗长模式 详细意味着它将输出可用于调试和理解培训方式的消息。惯性是每个点与其最近质心的平方距离之和，即其指定的簇 | 0 |  |
| **random\_state** | 随机生成簇中心的状态条件。 | None |  |
| **copy\_x** | 对是否修改数据的一个标记，如果True，即复制了就不会修改数据。  在scikit-learn 很多接口中都会有这个参数的，就是是否对输入数据继续copy 操作，以便不修改用户的输入数据。这个要理解Python 的内存机制才会比较清楚。 | True |  |
| **algorithm** | kmeans的实现算法，有：’auto’, ‘full’, ‘elkan’, 其中 ‘full’表示用EM方式实现 | auto |  |
| **n\_jobs** | 并行设置 | 1 | 与自己电脑配置有关 |

|  |  |  |  |
| --- | --- | --- | --- |
| **mini\_batch\_kmeans** | | | |
| **参数名** | API解释 | Sklearn默认值 |
| **n\_clusters** | 簇的个数，即你想聚成几类 | 8 |
| **init** | 初始簇中心的获取方法 | init='k-means++' |
| **max\_iter** | max\_iter=300,#最大迭代次数（因为kmeans算法的实现需要迭代） | 100 |
| **batch\_size** | batch\_size的大小 | 100 |
| **verbose** | 冗长模式 详细意味着它将输出可用于调试和理解培训方式的消息。惯性是每个点与其最近质心的平方距离之和，即其指定的簇 | 0 |
| **compute\_labels** | 一旦小批量优化收敛到适合，计算完整数据集的标签分配和惯性。 | True |
| **random\_state** | 随机生成簇中心的状态条件。 | None |
| **tol** | 容忍度，即kmeans运行准则收敛的条件 | 0.0 |
| **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。 | 10 |
| **n\_init** | 获取初始簇中心的更迭次数，为了弥补初始质心的影响，算法默认会初始10次质心 | 3 |
| **init\_size** | 为加速初始化而随机采样的样本数（有时以牺牲精度为代价）： 通过在数据的随机子集上运行批处理KMeans来初始化唯一的算法。 需要大于n\_clusters | None |
| **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。 | 0.01 |

|  |  |  |  |
| --- | --- | --- | --- |
| **DBSCAN** | | | |
| 参数名 | API解释 | Sklearn默认值 | 经验值 |
| **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。 | 0.5 |  |
| **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 | 5 |  |
| **metric** | 最近邻距离度量参数。可以使用的距离度量较多，默认采用欧式距离 | 欧几里得距离 |  |
| **metric\_params** | Additional keyword arguments for the metric function | metric\_params=None |  |
| **algorithm** | optional 最近邻搜索算法参数 默认：auto。brute是蛮力实现，kd\_tree是KD树实现，ball\_tree是球树实现,auto则会在三种算法中做权衡，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’。 | auto | 默认auto即可 |
| **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。 | 30 |  |
| **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则不需要管这个参数。 | None |  |
| **n\_jobs** | 进程数 | 1 | 与自己电脑配置有关 |