**Scikit-Learn回归算法**

# sklearn.ensemble

## 分类

VotingClassifier

### Bagging算法

BaggingClassifier（随机样本子集）

RandomForestClassifier（随机样本子集，随机特征）

ExtraTreesClassifier（随机样本子集，随机特征，随机划分）

### Boosting算法

AdaBoostClassifier

GradientBoostingClassifier

## 回归

### Bagging算法

结合强壮复杂的模型，如完全发展的决策树。

#### BaggingRegressor

* 基本学习器；
* max\_samples
* max\_features
* bootstrap
* bootstrap\_features
* oob\_score=True

#### RandomForestRegressor

训练集的随机子集，有放回的取样，特征空间的随机子集，划分子节点时，寻找最佳划分。这种随机性使得随机森林预测的偏差增大，但是通过平均作用，方差减小了，能够补偿偏差的增加。

随机森林参数

* n\_estimators 越大越好
* max\_features
  + - max\_features=n\_features for regression
    - max\_features=sqrt(n\_features) for classification
* max\_depth=None 自由发展，得到强壮复杂的模型
* min\_samples\_split=2 fully developing the trees
* max\_leaf\_nodes
* max\_features
* min\_samples\_leaf
* bootstrap=True
* oob\_score=True

#### ExtraTreesRegressor

训练集的随机子集，有放回的取样，特征空间的随机子集，划分子节点时，每个特征的阈值随机产生，并选择一个最佳的。这导致方差减小的更多，但是偏差增加的更多。

默认情况下，bootstrap=False

### Boosting算法

结合弱学习器，如树桩

#### AdaBoostRegressor

基本思想是在不断改变的数据集上训练一序列弱学习器（仅仅比随机猜要好就行），然后把这些结果求加权和

* learning\_rate
* base\_estimator（decision stumps）
* **n\_estimators**
* **max\_depth**
* **min\_samples\_split**

#### GradientBoostingRegressor

损失函数任意可微

* **learning\_rate(与n\_estimators相互影响，learning\_rate越小，n\_estimators越大，经验表明，其越小越好，一般设置为learning\_rate<0.1)**
* **n\_estimators（通过提前停止选择）**
* **max\_depth（max\_leaf\_nodes）**
* **min\_samples\_split**
* train\_score\_
* staged\_predict
* feature\_importances\_

The train error at each iteration is stored in the train\_score\_ attribute of the gradient boosting model. The test error at each iterations can be obtained via the staged\_predict method which returns a generator that yields the predictions at each stage. Plots like these can be used to determine the optimal number of trees (i.e. n\_estimators) by early stopping. The plot on the right shows the feature importances which can be obtained via the feature\_importances\_ property.

* train\_score\_，存储每一次迭代后的训练误差（应该是均方误差）
* staged\_predict，能够计算每一次迭代后的测试误差

test\_score = np.zeros((params['n\_estimators'],), dtype=np.float64)

for i, y\_pred in enumerate(clf.staged\_predict(X\_test)):

test\_score[i] = clf.loss\_(y\_test, y\_pred)

损失函数loss:

回归：

– Least squares ('ls')其卓越的计算属性，使其成为回归的自然选择。 初始模型由目标值的平均值给出。

– Least absolute deviation ('lad') 回归的鲁棒性很好的损失函数。 初始模型由目标值的中值给出。

– Huber ('huber'):

– Quantile ('quantile'): 分位点

几个主要的例子

**Gradient Boosting regression**

每次迭代的训练得分与测试得分；

clf.train\_score\_；

clf.staged\_predict(X\_test)；

clf.loss\_(y\_test, y\_pred)

**test\_score = np.zeros((params['n\_estimators'],), dtype=np.float64)**

**for i, y\_pred in enumerate(clf.staged\_predict(X\_test)):**

**test\_score[i] = clf.loss\_(y\_test, y\_pred)**

**score = np.zeros((n\_estimators,), dtype=np.float64)**

**for i, y\_pred in enumerate(clf.staged\_decision\_function(X\_test)):**

**score[i] = clf.loss\_(y\_test, y\_pred)**

**Gradient Boosting regularization**

正则化，learning\_rate=0.01，subsample=0.5，max\_features=2

**Early stopping of Gradient Boosting**

n\_estimators，validation\_fraction=0.2，n\_iter\_no\_change=5，tol=0.01，

# 特征重要性

通常，不同的特征对预测目标值的作用不同; 在很多情况下，大多数特征实际上都是无关紧要的。在解释模型时，第一个问题通常是：什么特征最重要？它们如何有助于预测目标值？本质上，单个决策树通过选择适当的分裂点来进行特征选择。此信息可用于衡量每个特征的重要性; 其基本思想是：在树的分裂点中使用特征的次数越多，该特征就越重要。 通过简单地平均每棵树的特征重要性，可以将这种重要性概念扩展到决策树集合中去（有关详细信息，请参阅特征重要性评估）。 可以通过feature\_importances\_属性访问拟合梯度增强模型的特征重要性分数：