多分类因变量的情形

sklearn可以做到模型的正则拟合，但模型架构和一般介绍的形式有所差异。

In [ ]:

```
from sklearn import linear_model

reg = linear_model.LogisticRegression()

reg.fit(iris.data, iris.target)
```

In [ ]:

```
reg.intercept_, reg.coef_
```

In [ ]:

```
print(classification_report(iris.target, reg.predict(iris.data)))
```

In [ ]:

```
from sklearn import linear_model

reg = linear_model.LogisticRegression(solver = 'newton-cg',
                                      multi_class = 'multinomial')

reg.fit(iris.data, iris.target)
```

In [ ]:

```
reg.intercept_, reg.coef_
```

In [ ]:

```
print(classification_report(iris.target, reg.predict(iris.data)))
```

5.7 实战练习

尝试使用似然比检验对logit表单数据进行变量筛选，得到最终的logistic回归模型。

6 决策树模型

6.1 树模型的基本原理

6.2 各种树模型算法

6.3 树模型的sklearn实现

6.3.1 拟合决策树模型

class sklearn.tree.DecisionTreeClassifier(

```
    criterion = 'gini' : 衡量节点拆分质量的指标，{'gini', 'entropy'}

    splitter = 'best' : 节点拆分时的策略
        'best'代表最佳拆分，'random'为最佳随机拆分
    max_depth = None : 树生长的最大深度（高度）
    min_samples_split = 2 : 节点允许进一步分枝时的最低样本数
    min_samples_leaf = 1 : 叶节点的最低样本量
    min_weight_fraction_leaf = 0.0 : 有权重时叶节点的最低权重分值
    max_features = 'auto' : int/float/string/None, 搜索分支时考虑的特征数
        'auto'/'sqrt', max_features = sqrt(n_features)
        'log2', max_features = log2(n_features)
        None, max_features = n_features
    random_state = None
    max_leaf_nodes = None : 最高叶节点数量
    min_impurity_decrease = 0.0 : 分枝时需要的最低信息量下降量
    class_weight = None, presort = False

)
```

DecisionTreeClassifier类的属性：

```
    classes_ : array of shape = [n_classes] or a list of such arrays
    feature_importances_ : array of shape = [n_features], 特征重要性评价
        总和为1，也被称为gini重要性
    max_features_ : int
    n_classes_ : int or list
    n_features_ : int
    n_outputs_ : int
    tree_ : Tree object
```

注意：树模型也可以用于数值变量预测，对应的方法为sklearn.tree.DecisionTreeRegressor

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier

ct = DecisionTreeClassifier()
ct.fit(iris.data, iris.target)
```

In [ ]:

```
ct.max_features_
```

In [ ]:

```
ct.feature_importances_
```

In [ ]:

```
ct.predict(iris.data)[:10]
```

In [ ]:

```
print(classification_report(iris.target, ct.predict(iris.data)))
```

6.3.2 使用graphviz浏览树模型

sklearn.tree模块中提供了将模型导出为graphviz格式文件的功能，从而可以对模型做图形观察。

http://www.graphviz.org，下载graphviz的安装包（可选择msi格式）

安装pydot并进行所需配置，就可以在python环境中直接调用graphviz。

但这样做实际意义不大，且操作比较麻烦，不建议使用

sklearn.tree.export_graphviz(

```
decision_tree, out_file = "tree.dot"

max_depth = None, feature_names = None, class_names = None
label = 'all' : {'all', 'root', 'none'}, 是否显示杂质测量指标
filled = False : 是否对节点填色加强表示
leaves_parallel = False : 是否在树底部绘制所有叶节点
impurity = True, node_ids = False
proportion = False : 是否给出节点样本占比而不是样本量
rotate = False : 是否从左至右绘图
rounded = False : 是否绘制圆角框而不是直角长方框
special_characters = False : 是否忽略PS兼容的特殊字符
precision = 3
```

)

In [ ]:

```
from sklearn.tree import export_graphviz

export_graphviz(ct, out_file = 'tree.dot',
                feature_names = iris.feature_names,
                class_names = iris.target_names)
```

In [ ]:

```
iris.target_names
```

6.4 随机森林

class sklearn.ensemble.RandomForestClassifier/RandomForestRegressor(

```
    n_estimators = 10 : 森林中树的数量
    criterion = 'gini'/'mse' : 树生长时使用的指标
        分类树为'gini'或'entropy', 回归树为'mse'或'mae'

    max_features = 'auto' : int/float/string/None, 搜索分支时考虑的特征数
        'auto'/'sqrt', max_features = sqrt(n_features)
        'log2', max_features = log2(n_features)
        None, max_features = n_features

    max_depth = None : 树生长的最大高度
    min_samples_split = 2 : 节点允许进一步分枝时的最低样本数
    min_samples_leaf = 1 : 叶节点的最低样本量
    min_weight_fraction_leaf = 0.0 : 有权重时叶节点的最低权重分值
    max_leaf_nodes = None : 最高叶节点数量
    min_impurity_decrease = 0.0 : 分枝时需要的最低信息量下降量

    bootstrap = True : 是否使用可放回的bootstap抽样
    oob_score = False : 是否使用OOB方式计算模型准确率

    n_jobs = 1, random_state = None
    verbose = 0, warm_start = False, class_weight = None
)
```

RandomForestClassifier/RandomForestRegressor类共有的属性：

```
    estimators_ : 森林中所有基模型的列表
    feature_importances_ : array of shape = [n_features], 各特征重要性
    n_features_ : int
    n_outputs_ : int, 因变量数量
    oob_score_ : float, 使用OOB方式得到的训练集评分
```

RandomForestClassifier独有的属性：

```
    classes_ : array of shape = [n_classes] or a list of such arrays
    n_classes_ : int or list
    oob_decision_function_ : array of shape = [n_samples, n_classes]
        使用OOB方式计算出的各案例的类别预测概率
```

RandomForestRegressor类独有的属性：

```
    oob_prediction_ : array of shape = [n_samples]
```

RandomForestClassifier/RandomForestRegressor类独有的方法：

```
    apply(X) : 对X拟合模型, 并返回所属的节点索引
    decision_path(X) : 返回对应案例的决策路径
```

6.4.1 随机森林分类实例

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(500, oob_score = True)
rfc.fit(iris.data, iris.target)
```

In [ ]:

```
print(len(rfc.estimators_))
rfc.estimators_[0]
```

In [ ]:

```
rfc.estimators_[0].predict(iris.data[:5])
```

In [ ]:

```
rfc.estimators_[0].feature_importances_
```

In [ ]:

```
rfc.feature_importances_
```

In [ ]:

```
rfc.oob_score_
```

In [ ]:

```
rfc.oob_decision_function_[:5]
```

In [ ]:

```
rfc.predict_proba(iris.data[:5])
```

6.4.2 随机森林回归实例

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor(500, oob_score = True)
rfr.fit(boston.data, boston.target)
```

In [ ]:

```
rfr.feature_importances_
```

In [ ]:

```
rfr.oob_score_
```

In [ ]:

```
rfr.oob_prediction_[:5]
```

In [ ]:

```
rfr.predict(boston.data[:5])
```

6.5 AdaBoost

class sklearn.ensemble.AdaBoostClassifier(

```
    base_estimator = DecisionTreeClassifier : 使用的基估计器
    n_estimators = 50 : integer, 迭代次数
    learning_rate = 1. : float, 学习率，用于减少每个分类器的贡献
    algorithm = 'SAMME.R' : {'SAMME', 'SAMME.R'}, 具体算法
        'SAMME.R' : real boosting algorithm, 一般比SAMME更快拟合
        'SAMME' : SAMME discrete boosting algorithm
    random_state = None : int, 随机种子
```

)

class sklearn.ensemble.AdaBoostRegressor(

```
    base_estimator = DecisionTreeRegressor, n_estimators = 50
    learning_rate = 1.0, random_state = None
    loss = 'linear' {'linear', 'square', 'exponential'}
```

)

AdaBoostClassifier/AdaBoostRegressor类的属性：

```
    classes_ : array of shape = [n_classes], 类标签
    n_classes_ : int, 类别数

    estimators_ : list of classifiers
    estimator_weights_ : array of floats, 每个分类器的权重
    estimator_errors_ : array of floats, 每个分类器的误差
    feature_importances_ : array of shape = [n_features], 各属性的重要性
```

AdaBoostClassifier/AdaBoostRegressor类独有的方法：

```
    staged_predict_proba(X) : AdaBoostClassifier类的方法，分步的预测概率

    staged_predict(X) : 分步的预测值
    staged_score(X, y[, sample_weight]) : 分步的评分
```

In [ ]:

```
from sklearn.ensemble import AdaBoostClassifier

adac = AdaBoostClassifier()
adac.fit(iris.data, iris.target)
```

In [ ]:

```
adac.estimators_[0]
```

In [ ]:

```
# 列出基估计器的大小
adac.estimators_[0].tree_.node_count
```

In [ ]:

```
adac.estimator_errors_
```

In [ ]:

```
adac.feature_importances_
```

In [ ]:

```
# 注意staged系列函数生成的是generator
for item in adac.staged_predict_proba(iris.data[:5]):
    print(item)
```

In [ ]:

```
adac.predict_proba(iris.data[:5])
```

6.6 梯度提升树（GBDT）

class sklearn.ensemble.GradientBoostingClassifier/GradientBoostingRegressor(

```
    loss = 'deviance'/'ls' : 损失函数的设定
        分类: {'deviance', 'exponential'}
        回归: {'ls', 'lad', 'huber', 'quantile'}
    subsample = 1.0 : 用于训练每个基分类器的样本比例，为1.0时显然无OOB输出
        降低该比例可以减少方差，但同时增大偏差

    criterion = 'friedman_mse' : 分枝质量的评价指标
        'friedman_mse', 'mse', 'mae' , 一般认为friedman_mse的效果最好

    learning_rate = 0.1, n_estimators = 100,
    min_samples_split = 2, min_samples_leaf = 1
    min_weight_fraction_leaf = 0.0, max_depth = 3
    min_impurity_decrease = .0, min_impurity_split = None, init = None
    random_state = None, max_features = None, verbose = 0
    max_leaf_nodes = None, warm_start = False

    presort = 'auto' : Bool, 是否对数据做预排序以加速拟合
```

)

GradientBoostingClassifier/GradientBoostingRegressor类的属性:

```
feature_importances_ : array, shape = [n_features]
oob_improvement_ : array, shape = [n_estimators]
train_score_ : array, shape = [n_estimators], deviance值的迭代记录
loss_ : LossFunction
init : BaseEstimator
estimators_ : ndarray of DecisionTreeRegressor
```

GradientBoostingClassifier/GradientBoostingRegressor类独有的方法：

```
staged_predict_proba(X) : 分步的预测概率

staged_predict(X) : 分步的预测值
staged_decision_function : 分步的决策函数
```

In [ ]:

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier()
gbc.fit(iris.data, iris.target)
```

In [ ]:

```
# 注意基模型的数量 = 类别数量
gbc.estimators_[0]
```

In [ ]:

```
gbc.feature_importances_
```

In [ ]:

```
gbc.train_score_[:10]
```

In [ ]:

```
plt.plot(gbc.train_score_)
```

In [ ]:

```
gbc.predict_proba(iris.data[:5])
```

In [ ]:

```
gbc.decision_function(iris.data[:5])
```

In [ ]:

```
for item in gbc.staged_predict_proba(iris.data[:5]):
    print(item)
```

6.7 实战练习

请尝试使用分类树、随机森林、Adaboost、GBDT方法对logit表单数据进行建模分析。

请尝试使用回归树、随机森林、Adaboost、GBDT方法对CCSS数据进行建模分析。

7 神经网络

7.1 神经网络的基本原理

7.2 BP神经网络的sklearn实现

7.2.1 MLP分类

class sklearn.neural_network.MLPClassifier(

```
    hidden_layer_sizes : tuple格式，长度 = n_layers - 2
        默认(100,)，第i个元素表示第i个隐藏层的神经元个数
    activation = 'relu' : 指定连接函数
        'identity' : f(x) = x
        'logistic' : 使用sigmoid连接函数
        'tanh' : f(x) = tanh(x)
        'relu' : f(x) = max(0, x)
    solver = 'adam' : {'lbfgs', 'sgd', 'adam'}, 具体的模型拟合方法
        lbfgs : quasi-Newton方法的优化器，小数据集使用该方法更好
        sgd : 随机梯度下降
        adam : 随机梯度优化器，大样本使用该方法更好
    alpha : float，默认0.0001，正则化项参数，用于防止过拟合
    batch_size = 'auto' : int，随机优化的minibatches的大小
        当设置成'auto'，batch_size = min(200, n_samples)
    learning_rate = 'constant' : 权重更新时的学习率变化情况
        'constant': 使用'learning_rate_init'指定的恒定学习率
        'incscaling': 随着时间t使用'power_t'的逆标度指数不断降低学习率
            effective_learning_rate = learning_rate_init / pow(t, power_t)
        'adaptive': 只要训练损耗下降，就保持学习率为'learning_rate_init'
            连续两次不能降低训练损耗或验证分数停止升高至少tol时，将学习率除以5
    max_iter = 200 : int，最大迭代次数
    random_state = None : int 或RandomState，随机数生成器的状态或种子
    warm_start = False : bool, 是否使用上一次的拟合结果作为初始拟合值
```

)

MLPClassifier类的属性：

```
    classes_ : 每个输出的类标签
    loss_ : 损失函数计算出来的当前损失值
    coefs_ : 列表中的第i个元素表示i层的权重矩阵
    intercepts_ : 列表中第i个元素代表i+1层的偏差向量
    n_iter_ : 迭代次数
    n_layers_ : 层数
    n_outputs_ : 输出的个数
    out_activation_ : 输出激活函数的名称
```

MLPClassifier类的方法：