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：

```
    fit(X,y) : 拟合
    get_params([deep]) : 获取参数
    predict(X) : 使用MLP进行预测
    predic_log_proba(X) : 返回对数概率估计
    predic_proba(X) : 概率估计
    score(X,y[,sample_weight]) : 返回给定测试集类别预测的平均准确度
    set_params(**params) : 设置参数
```

)

注意：神经网络也可以用于数值变量预测，对应的方法为sklearn.neural_network.MLPRegressor

In [ ]:

```
from sklearn import datasets

iris = datasets.load_iris()
irisdf = pd.DataFrame(iris.data, columns = iris.feature_names)
irisdf.head()
```

In [ ]:

```
# 对自变量做标准化
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
irisZX = scaler.fit_transform(iris.data)
irisZX[:5]
```

In [ ]:

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(activation = 'logistic', hidden_layer_sizes=(5, 5),
                    solver = 'lbfgs', random_state = 1)
clf.fit(irisZX, iris.target)
```

In [ ]:

```
clf.coefs_
```

In [ ]:

```
clf.score(irisZX, iris.target)
```

In [ ]:

```
clf.predict_proba(irisZX)[:5]
```

In [ ]:

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
    irisZX, iris.target, test_size = 0.3) # 这里可以直接使用稀疏矩阵格式
x_train[0]
```

In [ ]:

```
clf = MLPClassifier(activation = 'logistic', hidden_layer_sizes=(5, 5),
                    solver = 'lbfgs', random_state = 1)
clf.fit(x_train, y_train)
```

In [ ]:

```
clf.score(x_train, y_train), clf.score(x_test, y_test)
```

7.2.2 MLP回归

MLPRegressor用于对连续因变量进行预测，模型在训练时的输出层没有使用激活函数（也可以看作是使用identity function作为激活函数），因此其损失函数就是离均差平方和。

class sklearn.neural_network.MLPRegressor(

```
    hidden_layer_sizes = (100,), activation = 'relu', solver = 'adam',
    alpha = 0.0001, batch_size = 'auto', learning_rate = 'constant'
```

)

In [ ]:

```
from sklearn import datasets

boston = datasets.load_boston()
```

In [ ]:

```
# 对自变量做标准化
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
bostonZX = scaler.fit_transform(boston.data)
bostonZX[:5]
```

In [ ]:

```
from sklearn.neural_network import MLPRegressor

clf = MLPRegressor(activation = 'logistic', hidden_layer_sizes=(5, 5),
                    solver = 'lbfgs', random_state = 1)
clf.fit(bostonZX, boston.target)
```

In [ ]:

```
clf.score(bostonZX, boston.target)
```

7.3 神经网络的超参数调整

7.4 其他神经网络算法

7.4.1 有监督学习算法网络

7.4.2 非监督学习算法网络

7.5 实战练习

请尝试使用神经网络方法对logit表单数据进行建模预测，并进行参数调优。

提示：参数调优操作请参见第8章相应内容。

对boston数据使用神经网络回归进行分析，先拆分为训练集和测试集，然后在其他参数固定不变的情况下，进行如下参数调整，观察结果变化。

将单隐含层的神经元数量设定为1~100。
将网络层数设定为1~20。
将连接函数设定为identity、logistic、tanh、relu。
将alpha设定为0.01~100。

8 支持向量机

8.1 支持向量机的基本原理

8.2 SVM分类

sklearn中的SVM分类方法：

SVC和NuSVC：是相似的方法，但参数设定不同，数学表达式也有差异。
LinearSVC：线性核函数的支持向量分类

class sklearn.svm.SVC(

```
C = 1.0 : float，错分案例的惩罚参数
    本质上是在错分样本和分界面的简单性之间进行权衡
    低的C值使分界面平滑，而高的C值则通过增加模型自由度给出更复杂的分界面

kernel = 'rbf' : 算法中使用的核函数
    'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable

degree = 3 : 多项式核函数时使用的阶次
gamma = 'auto' : 'rbf', 'poly'和'sigmoid'使用的核系数
    实际上定义了单个样本对模型的影响大小，值越小影响越大，值越大影响越小
    可以看作被模型选中作为支持向量的样本的影响半径的倒数
    'auto'时为1/n_features

coef0 = 0.0, shrinking = True

probability = False : 是否要求进行概率的估计，该选项会增加拟合时间

tol = 0.001, cache_size = 200, class_weight = None, verbose = False
max_iter = -1, decision_function_shape = 'ovr', random_state = None
```