针对boston数据，生成所有的二次、三次交互项和高阶项，然后使用特征选择功能进行筛选，随后建模，考察分析结果，并进行思考。

5 类别预测模型的训练

5.1 类别预测模型概述

5.2 logistic回归

class sklearn.linear_model.LogisticRegression(

```
    penalty = 'l2', dual = False, tol = 0.0001, C = 1.0
    fit_intercept = True, intercept_scaling = 1
    class_weight = None : dict or 'balanced', 各类的权重
         权重以{class_label: weight}形式提供，None时默认均为1
         'balanced' :  权重和频次成反比，样本量/(类别数*np.bincount(y))
    random_state = None
    solver = 'liblinear' : 具体的拟合方法
        {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}
        'liblinear’ 适用于小数据集，'sag'和'saga'针对大数据集拟合速度更快
        多分类目标变量只能使用'newton-cg', 'sag', 'saga'和'lbfgs'拟合
        'newton-cg', 'lbfgs'和'sag'只能使用L2正则化
        'liblinear'和'saga'则可以处理L1正则化

    max_iter = 100, multi_class = 'ovr', verbose = 0
    warm_start = False, n_jobs = 1
```

)

LogisticRegression类的属性：

```
    coef_ : array, shape (1, n_features) or (n_classes, n_features)
    intercept_ : array, shape (1,) or (n_classes,)
    n_iter_ : array, shape (n_classes,) or (1, )
```

LogisticRegression类的方法：

```
    decision_function(X) : Predict confidence scores for samples.
    densify() : Convert coefficient matrix to dense array format.
    fit(X, y[, sample_weight])
    get_params([deep]) : Get parameters for this estimator.
    predict(X) : Predict class labels for samples in X.
    predict_log_proba(X) : 对数概率估计
    predict_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) : Set the parameters of this estimator.
    sparsify() : Convert coefficient matrix to sparse format.
```

两分类因变量的情形

In [ ]:

```
from sklearn.preprocessing import binarize

tmpy = binarize(iris.target.reshape(-1, 1))
```

In [ ]:

```
tmpy[:10]
```

In [ ]:

```
from sklearn import linear_model

reg = linear_model.LogisticRegression()

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

In [ ]:

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

In [ ]:

```
reg.predict_proba(iris.data)[:10]
```

In [ ]:

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

In [ ]:

```
reg.score(iris.data, tmpy)
```

In [ ]:

```
from sklearn.metrics import classification_report

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

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)))
```

5.3 神经网络

5.3.1 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)
```

5.3.2 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)
```

5.4 决策树模型

5.4.1 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)))
```

In [ ]:

```
ct
```

5.4.2 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
```

5.5 随机梯度下降法用于类别预测

class sklearn.linear_model.SGDClassifier(

```
    loss = 'hinge' : str, 类别预测模型的损失函数
        'hinge', 线性SVM
        'log', logistic回归
        'huber', 比OLS对离群值更耐受
        'modified_huber', huber算法的改进
        'squared_hinge', 惩罚项为hinge的平方
        'perceptron', perceptron算法
        'epsilon_insensitive', 忽略小于epsilon的残差
        'squared_epsilon_insensitive', 忽略平方小于epsilon的残差

    penalty = 'l2' : 正则化方法, 'none', 'l2', 'l1', or 'elasticnet'

    alpha = 0.0001, l1_ratio = 0.15
    fit_intercept = True, max_iter = None
    tol = None, shuffle = True, verbose = 0
    epsilon = 0.1 : epsilon-insensitive损失函数中的参数
        用于除'squared_loss'外的另三种方法
    n_jobs=1, random_state = None
    learning_rate = 'optimal' : 学习速度的设定
        'constant': eta = eta0
        'optimal': eta = 1.0 / (alpha * (t + t0)) [default]
        'invscaling': eta = eta0 / pow(t, power_t)
    eta0 = 0.0, power_t = 0.5, class_weight = None, warm_start = False
    average = False, n_iter = None
```

)

sklearn.linear_model.SGDClassifier类的属性：

```
    coef_ : array, shape (1, n_features) if n_classes == 2
        else (n_classes, n_features)
    intercept_ : array, shape (1,) if n_classes == 2 else (n_classes,)
    n_iter_ : int
    loss_function_ : concrete LossFunction
```

sklearn.linear_model.SGDClassifier类的方法：

```
decision_function(X) : 各样本对各类别的决策函数值
densify() : 将系数矩阵转换为标准格式
sparsify() : 将系数矩阵转换为稀疏格式
fit(X, y[, coef_init, intercept_init, …])
get_params([deep])
partial_fit(X, y[, classes, sample_weight])
predict(X)
score(X, y[, sample_weight])
set_params(*args, **kwargs)
```

In [ ]:

```
from sklearn.preprocessing import scale

ZX = scale(iris.data)
```

In [ ]:

```
from sklearn.linear_model import SGDClassifier

sgdcls = SGDClassifier(max_iter = 100)
sgdcls.fit(ZX, iris.target)
```

In [ ]:

```
sgdcls.score(ZX, iris.target)
```

In [ ]:

```
sgdcls.predict(ZX)[:10]
```

In [ ]:

```
sgdcls.decision_function(ZX)[:5]
```

5.6 实战练习

尝试在使用MLP回归分析bosston数据时，将连接函数改为'identity'，并且限制神经元数量，找到神经元数量和最终模型准确率（score）之间的关系，并和上一章中的线性回归模型作比较，思考其中的原因。

使用各种不同的随机种子对iris数据拟合树模型，汇总得到的feature_*importances*以及模型评分情况，思考树模型对样本量的需求是怎样的。

6 聚类模型的训练

6.1 聚类模型概述

6.2 K-均值聚类

class sklearn.cluster.KMeans(