In [ ]:

```
Zdf = pd.DataFrame(X_scaled)
Zdf['z1'] = 0.52237162 * Zdf[0] - 0.26335492 * Zdf[1] \
            + 0.58125401 * Zdf[2] + 0.56561105 * Zdf[3]
Zdf['z2'] = 0.37231836 * Zdf[0] + 0.92555649 * Zdf[1] \
            + 0.02109478 * Zdf[2] + 0.06541577 * Zdf[3]
Zdf.describe()
```

In [ ]:

```
Zdf.head()
```

In [ ]:

```
# 各主成分相加时应当按照携带信息量的大小进行加权
Zdf['tot'] = Zdf.z1 * 1.711828 + Zdf.z2 * 0.963018
Zdf.head(10)
```

In [ ]:

```
# 计算出主成分用于后续分析
pca.transform(X_scaled)[:5]
```

3.5 实战练习

使用GenericUnivariateSelect()类，针对boston数据实现SelectKBest、SelectPercentile、SelectFpr、SelectFdr和SelectFwe类的功能。

使用PCA方法对boston数据的自变量进行降维。

4 回归类模型的训练

4.1 线性回归模型

4.1.1 模型概述

4.1.2 线性回归模型的sklean实现

class sklearn.linear_model.LinearRegression(

```
fit_intercept = True : 模型是否包括常数项
    使用该选项就不需要在数据框中设定cons
normalize = False : 是否对数据做正则化，具体为(x - mean)/L2-norm
copy_X = True : 是否复制X矩阵
n_jobs = 1 : 使用的例程数，为-1时使用全部CPU，大样本多因变量时有加速效果
```

)

注意：

函数中的normalize参数并非进行标准正态变换。
    sklearn.preprocessing.StandardScaler可用于满足标准正态变换的需求。
数据中不能存在缺失值，否则报错。

LinearRegression类的属性：

coef_ : array，多因变量时为二维数组
intercept_ : 常数项

LinearRegression类的方法：

fit(X, y[, sample_weight]) : 拟合模型
get_params([deep]) : 获取模型的具体参数设定
predict(X) : 返回具体预测值
score(X, y[, sample_weight]) : 返回模型决定系数
set_params(**params) : 重新设定模型参数

注意：方法中没有返回系数检验结果（P值）的功能

In [ ]:

```
from sklearn import linear_model

reg = linear_model.LinearRegression()

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

In [ ]:

```
print(reg.coef_, reg.intercept_)
```

In [ ]:

```
# 返回模型拟合后的决定系数
reg.score(boston.data, boston.target)
```

In [ ]:

```
# 利用该模型进行预测
reg.predict(boston.data[:10])
```

4.2 多项式回归

class sklearn.preprocessing.PolynomialFeatures(

degree = 2 : integer，模型所考虑的高次项次方数
interaction_only = False : boolean，是否只生成交互项，忽略变量的高次方项
include_bias = True : boolean，模型中是否纳入常数项

)

PolynomialFeatures类的属性:

```
    powers_ : array, shape (n_output_features, n_input_features)
        powers_[i, j]代表第j个输入属性在第i个输出属性中的次方数
    n_input_features_ : int，输入特征的总数
    n_output_features_ : int，输出特征的总数
```

In [ ]:

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(2)

IX = poly.fit_transform(boston.data)
print(len(IX[1]))
IX[:1]
```

In [ ]:

```
from sklearn import linear_model

reg = linear_model.LinearRegression()

reg.fit(IX, boston.target)
```

In [ ]:

```
# sklearn中无法输出参数的检验结果，因此不能筛选变量，只能评价模型的整体效果
reg.score(IX, boston.target)
```

In [ ]:

```
# 进一步增加高次项
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(3)

IX = poly.fit_transform(boston.data)
len(IX[1])
```

In [ ]:

```
from sklearn import linear_model

reg = linear_model.LinearRegression()

reg.fit(IX, boston.target)
```

In [ ]:

```
reg.score(IX, boston.target)
```

4.3 岭回归、Lasso和弹性网络

4.3.1 损失函数与正则化

4.3.2 岭回归

共线性实例

案例：现测得22例胎儿的身长、头围、体重和胎儿受精周龄，具体数据见文件dmdata.xlsx的ridge表单，研究者希望能建立由前三个外形指标推测胎儿周龄的回归方程。

In [ ]:

```
dfridge = pd.read_excel("dmdata.xlsx", sheet_name = "ridge")
dfridge.head()
```

In [ ]:

```
reg = linear_model.LinearRegression()

reg.fit(dfridge.iloc[:, list(range(3))], dfridge.y)
```

In [ ]:

```
reg.score(dfridge.iloc[:, list(range(3))], dfridge.y)
```

In [ ]:

```
reg.coef_
```

拟合岭回归模型

class sklearn.linear_model.Ridge(

```
alpha = 1.0 : 模型惩罚项的系数，正数，越大惩罚力度越强
fit_intercept = True, normalize = False, copy_X = True

max_iter = None : 容许的最大迭代次数
tol = 0.001 : 收敛标准
solver='auto' : 收敛方法
    {'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag','saga'}
random_state = None : 伪随机种子的数值
```

)

注意：岭回归也可以用于分类模型，对应的方法为sklearn.linear_model.RidgeClassifier

In [ ]:

```
from sklearn import linear_model

ridge = linear_model.Ridge(alpha = 0)
ridge.fit(dfridge.iloc[:,[0,1,2]], dfridge.y)
ridge.coef_
```

In [ ]:

```
ridge.score(dfridge.iloc[:,[0,1,2]], dfridge.y)
```

使用线图协助选取最佳alpha值

In [ ]:

```
# 对自变量做标准化，以便于图形观察
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
ridgeZX = scaler.fit_transform(dfridge.iloc[:,[1,2,3]])
```

In [ ]:

```
# 设定用于筛选的一系列alpha值
n_alphas = 200
alphas = np.logspace(-10, 1, n_alphas)
alphas[:10]
```

In [ ]:

```
# 存储所有的模型结果
coefs = []
rsqs = []
for a in alphas:
    ridge = linear_model.Ridge(alpha = a)
    ridge.fit(ridgeZX, dfridge.y)
    coefs.append(ridge.coef_)
    rsqs.append(ridge.score(ridgeZX, dfridge.y))
coefs[:10]
```

In [ ]:

```
plt.plot(alphas, coefs)
```

In [ ]:

```
plt.plot(alphas, rsqs)
```

In [ ]:

```
# 放大关键区间进行观察
plt.plot(alphas, coefs)
plt.gca().set_xlim(0, 1)
ax2 = plt.gca().twinx()
ax2.plot(alphas, rsqs, 'r--', linewidth = 3)
```

In [ ]:

```
# 获取指定模型的结果
ridge = linear_model.Ridge(alpha = 0.5)
ridge.fit(dfridge.iloc[:,[1,2,3]], dfridge.y)
print(ridge.intercept_, ridge.coef_)
```

4.3.3 LASSO回归

class sklearn.linear_model.Lasso(

    alpha = 1.0, fit_intercept = True,
    normalize = False, copy_X = True

    precompute = False : 是否使用预计算的Gram矩阵加速拟合
    max_iter = 1000, tol = 0.0001
    warm_start = False : 是否使用上一次的模型拟合结果作为本次初始值
    positive = False : 系数值是否必须非负
    random_state = None
    selection = 'cyclic' : 随机更新系数还是按顺序更新，设为'random'可加速拟合

)

In [ ]:

```
# 存储所有的模型结果
coefs = []
rsqs = []
for a in alphas:
    lasso = linear_model.Lasso(alpha = a, max_iter = 10000)
    lasso.fit(ridgeZX, dfridge.y)
    coefs.append(lasso.coef_)
    rsqs.append(lasso.score(ridgeZX, dfridge.y))
coefs[:10]
```

In [ ]:

```
# 放大关键区间进行观察
plt.plot(alphas, coefs)
# plt.gca().set_xlim(0, 0.5)
ax2 = plt.gca().twinx()
ax2.plot(alphas, rsqs, 'r--', linewidth = 3)
```

In [ ]:

```
# 获取指定模型的结果
lasso = linear_model.Lasso(alpha = 1)
lasso.fit(dfridge.iloc[:,[0,1,2,3]], dfridge.y)
print(lasso.intercept_, lasso.coef_)
```

4.3.4 弹性网络

class sklearn.linear_model.ElasticNet(

    alpha = 1.0
    l1_ratio = 0.5 : L1和L2模型的混合比例
        l1_ratio = 0, 拟合Lasso回归
        l1_ratio = 1, 拟合岭回归
        0 < l1_ratio < 1, 两种模型的混合

    fit_intercept = True, normalize = False, precompute = False
    max_iter = 1000, copy_X = True, tol = 0.0001, warm_start = False
    positive = False, random_state = None, selection = 'cyclic'

)

In [ ]:

```
ratios = np.linspace(0, 1, 50)

coefs = []
rsqs = []
for r in ratios:
    enet = linear_model.ElasticNet(alpha = 0.5, l1_ratio = r)
    enet.fit(ridgeZX, dfridge.y)
    coefs.append(enet.coef_)
    rsqs.append(enet.score(ridgeZX, dfridge.y))
coefs[:10]
```

In [ ]:

```
# 放大关键区间进行观察
plt.plot(ratios, coefs)
# plt.gca().set_xlim(0, 0.5)
ax2 = plt.gca().twinx()
ax2.plot(ratios, rsqs, 'r--', linewidth = 3)
```

4.4 最小角回归

class sklearn.linear_model.Lars(

```
    fit_intercept = True, verbose = False, normalize = True
    precompute = 'auto'
    n_nonzero_coefs = 500 : 纳入模型的最大自变量数，np.inf代表无限制
    eps = 2.220446049250313e-16 : Cholesky diagonal factors的计算精度
    copy_X = True
    fit_path = True : 是否将系数路径存储在coef_path_属性中
        超大数据集可关闭该选项以加速分析
    positive = False : 是否限制系数必须非负
```

)

Lars类的属性：

```
    alphas_ : array, 形如(n_alphas+1,), 非零系数在迭代中的最大协方差绝对值
    active_ : list, length = n_alphas, 变量被纳入模型的先后顺序（索引值）
    coef_path_ : array, shape (n_features, n_alphas + 1)
        各变量在迭代中的系数改变情况，该结果可被用于模型调优
    coef_ : array, 形如(n_features,) or (n_targets, n_features), 系数值
    intercept_ : float | array, shape (n_targets,)
    n_iter_ : array-like or int, 模型迭代次数
```

In [ ]:

```
lars = linear_model.Lars(n_nonzero_coefs = 10)
lars.fit(boston.data, boston.target)
```

In [ ]:

```
lars.coef_
```

In [ ]:

```
lars.active_
```

In [ ]:

```
lars.score(boston.data, boston.target)
```

4.5 海量数据的模型拟合

4.5.1 随机梯度下降法

4.5.2 sklearn实现

class sklearn.linear_model.SGDRegressor(

```
loss = 'squared_loss' : str, 回归模型的损失函数
    {'squared_loss', 标准的OLS
    'huber', 比OLS对离群值更耐受
    '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
random_state = None
learning_rate = 'invscaling' : 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 = 0.01 : 初始学习率
power_t = 0.25, warm_start = False, average = False, n_iter = None
```

)

sklearn.linear_model.SGDRegressor类的属性:

```
coef_ : array, shape (n_features,)
intercept_ : array, shape (1,)
average_coef_ : array, shape (n_features,)
average_intercept_ : array, shape (1,)
n_iter_ : int
```

sklearn.linear_model.SGDRegressor类的方法:

```
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[, sample_weight])
predict(X)
score(X, y[, sample_weight])
set_params(*args, **kwargs)
```

In [ ]:

```
# 对数据做标准化
from sklearn.preprocessing import scale

ZX = scale(boston.data)
Zy = scale(boston.target)
```

In [ ]:

```
from sklearn.linear_model import SGDRegressor

sgdreg = SGDRegressor(max_iter = 100)
sgdreg.fit(ZX, Zy) # 这里也可以使用原始y值，模型仍然可以正常拟合
```

In [ ]:

```
sgdreg.coef_, sgdreg.intercept_
```

In [ ]:

```
sgdreg.score(ZX, Zy)
```

In [ ]:

```
# 试试使用非标化数据进行拟合
from sklearn.linear_model import SGDRegressor

sgdreg = SGDRegressor(max_iter = 1000)
sgdreg.fit(boston.data, boston.target)
```

In [ ]:

```
sgdreg.coef_, sgdreg.intercept_
```

In [ ]:

```
sgdreg.score(boston.data, boston.target)
```

4.6 实战练习

针对boston数据，尝试将LARS用于多项式回归的变量删减，看看是否能否奏效，并思考原因。

尝试使用提取主成分进行回归吧的方式对岭回归数据进行分析，比较两种处理方式的优劣。

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