In [ ]:

```
clf.best_estimator_.predict_proba(iris.data[:10])
```

8.6 实战练习

如果在模型训练完毕后，在数据中删除所有非支持向量，然后再重新建模，得到的模型结果是否会和原来完全一样？请用具体数据尝试一下，并思考原因。

请尝试对boston数据拟合SVM回归模型，并进行参数调优，找到最优模型。

提示：需要将数据拆分为训练集和验证集进行结果验证。

9 主成分分析与因子分析

9.1 主成分分析

9.1.1 主成分分析的基本原理

9.1.2 主成分分析的statsmodels实现

class statsmodels.multivariate.pca.PCA(

```
    data
    ncomp = None : 希望返回的主成分数，为None时全部返回

    standardize = True : 是否对数据做标准化，等价于用相关系数阵做PCA
    demean = True : 是否移除均数，standardize = True时该参数无效
        只使用该参数相当于基于协方差阵提取主成分
    normalize = True : 是否对提取出的主成分做标化

    gls = False : 是否使用两步GLS估计
    weights = None : 各变量在计算中的权重
    method = 'svd' : 具体使用的主成分提取方法
        'svd' 普通的SVD
        'eig' eigenvalue decomposition
        'nipals' 要提取的特征根数远少于变量数时，速度要快于SVD法

    missing = None : 对缺失值的处理方式
       {'drop-row'/'drop-col'/'drop-min' (删除行/列中较少的) /'fill-em'}
    tol = 5e-08, max_iter = 1000, tol_em = 5e-08, max_em_iter = 100
```

)

属性：

factors / scores : 提取出的主成分矩阵
loadings : 主成分的载荷矩阵,ncomp by nvar
coeff : 系数阵, nvar by ncomp
projection : 用于分析的原始数据阵
rsquare : 依次加入第i个主成分后模型的决定系数（基于变换后数据计算）
ic : Bai和Ng(2003)提出的信息准则
eigenvals : 特征根
eigenvecs : 特征向量, nvar by nvar array
weights : 各变量在计算中的权重
transformed_data : 原始数据标化并加权后的数据阵
rows / cols : PCA中使用的行/列标签列表

In [ ]:

```
plt.scatter(iris.data[:,0], iris.data[:,2])
```

In [ ]:

```
plt.scatter(iris.data[:,2], iris.data[:,3])
```

In [ ]:

```
import numpy as np
from statsmodels.multivariate.pca import PCA

pca = PCA(iris.data, ncomp = 2, normalize = False)
```

In [ ]:

```
# 注意这里输出的特征根数值实际上会*n_samples
pca.eigenvals
```

In [ ]:

```
pca.eigenvals / 150
```

In [ ]:

```
np.sqrt(pca.eigenvals/150)
```

In [ ]:

```
pca.rsquare
```

In [ ]:

```
pca.loadings
```

In [ ]:

```
Zdf = pd.DataFrame(pca.transformed_data)
Zdf['z1'] = 0.52106591 * Zdf[0] + -0.26934744 * Zdf[1] \
            + 0.5804131 * Zdf[2] + 0.56485654 * Zdf[3]
Zdf['z2'] = -0.37741762 * Zdf[0] + -0.92329566 * Zdf[1] \
            + -0.02449161 * Zdf[2] + -0.06694199 * Zdf[3]
Zdf.describe()
```

In [ ]:

```
pca.coeff
```

In [ ]:

```
# 注意此处需要使用标化后的原始变量进行计算，同时方差被放大n_samples倍
Zdf['z1'] = 10.90229629 * Zdf[0] + -5.6355742 * Zdf[1] \
            + 12.14402126 * Zdf[2] + 11.81853033 * Zdf[3]
Zdf['z2'] = -4.41924555 * Zdf[0] + -10.81102224 * Zdf[1] \
            + -0.28677632 * Zdf[2] + -0.78383484 * Zdf[3]
Zdf.describe()
```

In [ ]:

```
# 原始特征根，均数为0，方差等于奇异值
pd.DataFrame(pca.scores).describe()
```

9.1.3 主成分分析的sklearn实现

注意：sklearn中的PCA方法默认使用协方差阵，这一点和SPSS等统计软件不同。

因此必要时应当先对数据做标化

class sklearn.decomposition.PCA(

```
n_components = None : int/float/None/string，希望保留的主成分数量
    如果为None，则所有主成分均被保留，为'mle'时自动选择最佳数量

copy = True
whiten = False : 输出的主成分是否*sqrt(n_samples)/特征根，即标准化
    该转换会损失部分方差信息，但有时候会使得后续的建模效果有所改善

svd_solver = 'auto' : {'auto', 'full', 'arpack', 'randomized'}
    auto : 根据X.shape和n_components自动选择方法
    full : 完整的SVD解法，即LAPACK
    arpack : ARPACK法，要求0 < n_components <  X的列数
    randomized : Halko等提出的随机SVD法
tol = 0.0, iterated_power = 'auto', random_state = None
```

)

sklearn.decomposition.PCA类的属性：

components_ : array, 形如(n_components, n_features), 主成分系数矩阵
explained_variance_ : array, 形如(n_components,), 各主成分解释的方差量
explained_variance_ratio_ : array, 形如(n_components,), 解释方差比例
singular_values_ : array, 形如(n_components,), 各主成分对应的奇异值
mean_ : array, 形如(n_features,), 各属性的均数, 等价于X.mean(axis=1)
n_components_ : int
noise_variance_ : float, 剩余的噪声协方差

sklearn.decomposition.PCA类的方法:

fit(X[, y])
fit_transform(X[, y])
get_covariance() : 给出模型的协方差阵
get_params([deep])
get_precision() : 给出协方差矩阵的逆矩阵 (Precision Matrix)
inverse_transform(X)
score(X[, y]) : 给出样本的平均对数似然值
score_samples(X) : 给出每个样本的对数似然值
set_params(**params)
transform(X)

In [ ]:

```
from sklearn import preprocessing

X_scaled = preprocessing.scale(iris.data)
```

In [ ]:

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
# 使用标化后的数据, 因此等价于采用相关系数阵做PCA
pca.fit(X_scaled)
```

In [ ]:

```
# 给出主成分系数矩阵
pca.components_
```

In [ ]:

```
# 各主成分的方差解释量 (特征值)
pca.explained_variance_
```

In [ ]:

```
# 换算后的各主成分方差解释比例
pca.explained_variance_ratio_
```

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]
```

9.2 因子分析

9.2.1 因子分析的基本原理

9.2.2 拟合因子分析模型

注意:

sklearn中的decomposition.FactorAnalysis()功能过于简单，无使用价值。
statsmodels需要升级到0.9.0及以上版本。

class statsmodels.multivariate.factor.Factor(

endog = None : 需要分析的数据阵，为None时需要提供corr参数
n_factor = 1 : 希望提取的公因子数量
method = 'pa' : 公因子提取方法，{'pa', 'ml'}

smc = True : pa时是否使用平方多重相关
endog_names = None : str，变量名称列表

corr = None : 直接提供相关系数阵而不是原始数据
nobs = None : 使用相关系数阵时提供样本量

missing = 'drop' : 缺失值处理方式，('none', 'drop', or 'raise')

)

属性:

endog_names : Names of endogenous variables
exog_names : Names of exogenous variables

方法:

fit([maxiter, tol, start, opt_method, opt, …])
from_formula(formula, data[, subset, drop_cols]) : 基于公式建立模型
loglike(par) : Evaluate the log-likelihood function.
predict(params[, exog]) : 尚未完成开发
score(par) : 估计得分函数（对数似然值的一阶导数）

In [ ]:

```
from statsmodels.multivariate.factor import Factor

fac = Factor(iris.data, n_factor = 2, endog_names = iris.feature_names)
fac.fit()
```

In [ ]:

```
fac.endog_names
```

9.2.3 显示模型结果

class statsmodels.multivariate.factor.FactorResults(

factor : 拟合完毕的factor类

)

属性:

uniqueness : 各变量剩余的（特殊因子的）方差
communality : 公因子方差比，即1 - uniqueness
loadings : 载荷值
loadings_no_rot : 旋转前载荷值
eigenvals : 特征值
n_comp : 因子数
nbs : 案例数
fa_method : 因子提取方法
df : 模型自由度

方法:

```
factor_score_params([method]) : 因子得分系数阵
factor_scoring([endog, method, transform]) : 计算因子得分
fitted_cov() : Returns the fitted covariance matrix.
get_loadings_frame([style, sort_, …]) : 以Pandas方式给出载荷矩阵
    style = 'display' (default), 'raw' or 'strings'
load_stderr() : The standard errors of the loadings.
plot_loadings([loading_pairs, plot_prerotated]) : 载荷图
plot_scree([ncomp]) : 碎石图
rotate(method) : 进行因子旋转
summary() : 汇总输出关键结果
uniq_stderr([kurt]) : The standard errors of the uniquenesses.
```

In [ ]:

```
from statsmodels.multivariate.factor import FactorResults

fres = FactorResults(fac)
```

In [ ]:

```
# 碎石图
screeplot = fres.plot_scree()
```

In [ ]:

```
fres.communality
```

In [ ]:

```
fres.uniqueness
```

In [ ]:

```
fres.eigenvals
```

In [ ]:

```
fres.get_loadings_frame() # 'raw'
```

In [ ]:

```
fres.get_loadings_frame('raw')
```

In [ ]:

```
# 载荷图
fres.plot_loadings()
```

In [ ]:

```
fres.factor_score_params()
```

In [ ]:

```
fres.factor_scoring(iris.data)
```

In [ ]:

```
fres.summary()
```

9.2.4 因子旋转

FactorResults.rotate(

具体使用的因子旋转方法：varimax, quartimax, biquartimax, equamax, oblimin, parsimax, parsimony, biquartimin, promax

)

In [ ]:

```
fres.rotate('varimax')
```

In [ ]:

```
fres.summary()
```

In [ ]:

```
fres.get_loadings_frame()
```

In [ ]:

```
fres.plot_loadings()
```

In [ ]:

```
fres.factor_scoring(iris.data)[:5]
```

In [ ]:

```
# 使用斜交方式进行旋转
fres.rotate('oblimin')
```

In [ ]:

```
fres.summary()
```

In [ ]:

```
df = pd.DataFrame(fres.factor_scoring(iris.data))
df.plot.scatter(0, 1)
```

In [ ]:

```
from scipy import stats as ss

ss.pearsonr(df[0], df[1])
```

9.3 实战练习

尝试对ridge表单中的三个自变量提取主成分，使用主成分回归办法解决这三个自变量的共线性问题，并将结果和前述岭回归的结果相比较。

尝试对boston数据进行因子分析，各变量的具体含义参见boston.DESCR。

10 聚类分析

10.1 聚类模型概述

10.2 K-均值聚类

class sklearn.cluster.KMeans(

```
    n_clusters = 8
    init = 'k-means++' : 'k-means++'/'random'/ndarray, 初始类中心位置
        'k-means++' : 采用优化后的算法确定类中心
        'random' : 随机选取k个案例作为初始类中心
        ndarray : (n_clusters, n_features)格式提供的初始类中心位置

    n_init = 10, max_iter = 300, tol = 0.0001
    precompute_distances = 'auto' : {'auto', True, False}
        是否预先计算距离，分析速度更快，但需要更多内存
        'auto' : 如果n_samples*n_clusters > 12 million，则不事先计算距离
    verbose = 0, random_state = None, copy_x = True, n_jobs = 1
    algorithm = 'auto' : 'auto', 'full' or 'elkan', 具体使用的算法
        'full' : 经典的EM风格算法
        'elkan' : 使用三角不等式，速度更快，但不支持稀疏数据
        'auto' : 基于数据类型自动选择
```

)

KMeans类的属性：

```
    cluster_centers_ : array, [n_clusters, n_features]
    labels_ :
    inertia_ : float，各样本和其最近的类中心距离之和
```

注意：对于样本量超过1万的情形，建议使用MiniBatchKMeans，算法改进为在线增量，速度会更快