Python与金融数据挖掘(15)

文欣秀

wenxinxiu@ecust.edu.cn

机器学习分类

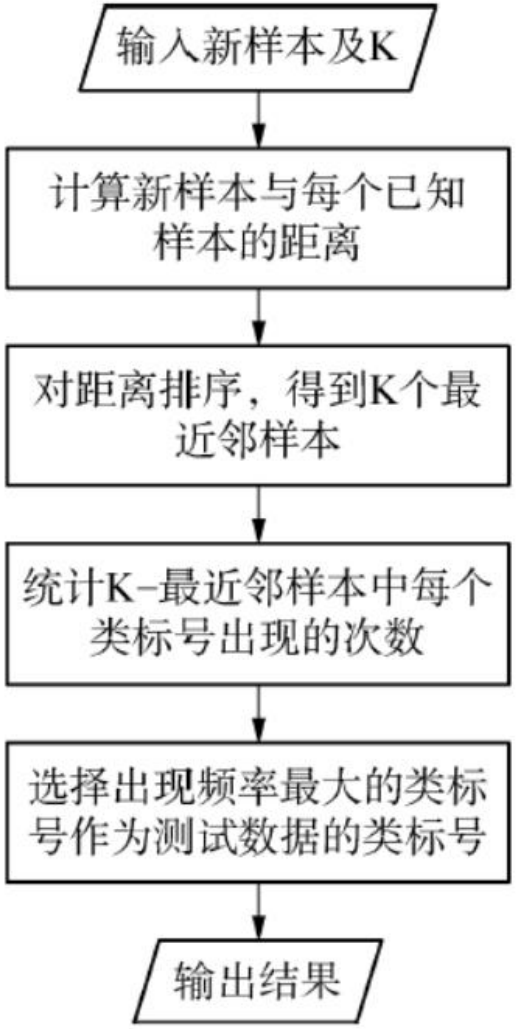
- 有监督学习（分类、回归）
- 无监督学习（聚类、降维）
- 强化学习
- 半监督学习

分类算法

- K近邻算法(KNN)
- 朴素贝叶斯算法(NB)
- 支持向量机(SVM)
- 决策树(DT)
- 逻辑回归(LR)

KNN算法流程

	A	B	C	D	E	F
1	收入	年龄	性别	历史授信额度	历史违约次数	是否违约
2	503999	46	1	0	1	1
3	452766	36	0	13583	0	1
4	100000	33	1	0	1	1
5	100000	25	0	0	1	1
6	258000	35	1	0	0	1
7	933333	31	0	28000	3	1
8	665000	40	1	5000	1	1
9	291332	38	0	0	0	1
10	259000	45	1	0	1	1
11	3076666	39	1	71000	2	1
12	695000	40	1	5000	0	1
13	600000	35	1	18000	3	1
14	440000	36	1	0	2	1
15	511999	35	0	0	2	1
16	248000	31	0	0	2	1
17	471000	40	1	0	1	1



客户违约情况预测

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
data=pd.read_csv('client.csv',encoding='gb2312')
model= KNeighborsClassifier()
X=data.iloc[:,0:5].values; y=data.iloc[:,5].values
model.fit(X, y)
income=int(input("您的收入:"));age=int(input("您的年龄:"))
sex=int(input("您的性别:"));limit=int(input("您的历史授信额度:"))
count=int(input("您的历史违约次数:"))
print('Type：',model.predict([[income,age,sex,limit,count]]))
```

```
您的收入:100000
您的年龄:22
您的性别:1
您的历史授信额度:0
您的历史违约次数:0
Type：  [1]
```

案例分析

	A	B	C	D
1	Sex	Height	Weight	Type
2	1	180	75	normal
3	1	180	85	normal
4	1	180	90	overweight
5	1	180	100	overweight
6	1	175	90	overweight
7	1	175	80	overweight
8	1	175	65	normal
9	1	175	55	underweight
10	1	170	60	normal
11	1	170	70	normal
12	1	170	80	overweight
13	1	185	90	overweight
14	1	185	75	normal
15	1	175	60	underweight
16	1	180	65	underweight
17	1	160	75	overweight
18	1	160	60	normal
19	1	170	68	normal
20	1	165	62	normal
21	1	190	75	underweight

胖瘦预测KNN算法

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
data=pd.read_csv('info.csv',encoding='gb2312')
model= KNeighborsClassifier()
X=data[['Sex','Height','Weight']].values
y= data['Type'].values
model.fit(X, y)
height=int(input("Your Height:"))
weight=int(input("Your Weight:"))
print('Type：  ',model.predict([[1,height,weight]]))
```

胖瘦预测NB算法

```
import pandas as pd
from sklearn.naive_bayes import GaussianNB
data=pd.read_csv('info.csv',encoding='gb2312')
model= GaussianNB()
X=data[['Sex','Height','Weight']].values
y= data['Type'].values
model.fit(X, y)
height=int(input("Your Height:"))
weight=int(input("Your Weight:"))
print('Type：  ',model.predict([[1,height,weight]]))
```

胖瘦预测SVC算法

```
import pandas as pd
from sklearn.svm import SVC
data=pd.read_csv('info.csv',encoding='gb2312')
model= SVC( )
X=data[['Sex','Height','Weight']].values
y= data['Type'].values
model.fit(X, y)
height=int(input("Your Height:"))
weight=int(input("Your Weight:"))
print('Type：  ',model.predict([[1,height,weight]]))
```

胖瘦预测DT算法

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
data=pd.read_csv('info.csv',encoding='gb2312')
model= DecisionTreeClassifier()
X=data[['Sex','Height','Weight']].values
y= data['Type'].values
model.fit(X, y)
height=int(input("Your Height:"))
weight=int(input("Your Weight:"))
print('Type：  ',model.predict([[1,height,weight]]))
```

胖瘦预测LR算法

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
data=pd.read_csv('info.csv',encoding='gb2312')
model= LogisticRegression(max_iter=10000)
X=data[['Sex','Height','Weight']].values
y= data['Type'].values
model.fit(X, y)
height=int(input("Your Height:"))
weight=int(input("Your Weight:"))
print('Type：  ',model.predict([[1,height,weight]]))
```

使迭代次数
不超过限制

思考题

问题： 如何比较以上算法的准确率从而选择有效的算法？

方案： 使用相同的数据，相同的方法来评估不同的算法，以便得到一个准确的结果。

	A	B	C	D	E
1	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
2	5.1	3.5	1.4	0.2	setosa
3	4.9	3	1.4	0.2	setosa
4	4.7	3.2	1.3	0.2	setosa
5	4.6	3.1	1.5	0.2	setosa
6	5	3.6	1.4	0.2	setosa
7	5.4	3.9	1.7	0.4	setosa
8	4.6	3.4	1.4	0.3	setosa
9	5	3.4	1.5	0.2	setosa
10	4.4	2.9	1.4	0.2	setosa
11	4.9	3.1	1.5	0.1	setosa

鸢尾花(iris)数据集分析

Iris 数据集： 是一个经典数据集，在统计学习和机器学习领域都经常被用作示例。数据集内包含 3 类共 150 条记录，每类各 50 个数据，每条记录都有 4 项特征：花萼长度(Sepal Length)、花萼宽度(Sepal Width)、花瓣长度(Petal Length)、花瓣宽度(Petal Width)，可以通过这4个特征预测鸢尾花卉属于iris-setosa(山鸢尾)，iris-versicolour(变色鸢尾), iris-virginica(维吉尼鸢尾)中的哪一品种。

十折交叉验证

十折交叉验证（10-fold cross-validation）：用来测试算法准确性。是常用的测试方法。**将数据集分成10份，轮流将其中9份作为训练数据，1份作为测试数据进行试验**。每次试验都会得出相应的正确率（或差错率）。10次的结果的正确率（或差错率）的平均值作为对算法精度的估计，一般还需要进行多次10折交叉验证（例如8次10折交叉验证），再求其均值，作为对算法准确性的估计。

K折交叉验证

Kfold()函数：sklearn 包中用于交叉验证的函数，一般情况将K折交叉验证用于模型调优。

函数功能：在机器学习中，将数据集data分为训练集（**training set**）**A**和测试集（**test set**）**B**，在样本量不充足的情况下，为了充分利用数据集对算法效果进行测试，将数集data随机分为**k**个包，每次将其中**1**个包作为**测试集**，剩下**k-1**个包作为**训练集**进行训练。

K折交叉验证

KFold(n_splits=10，shuffle=False，random_state=None)

n_splits : 整数，表示交叉验证的折数（即将数据集分为几份）。

shuffle : 布尔值, 表示是否要将数据打乱顺序后再进行划分，若为True时，每次划分的结果都不一样。

random_state : 默认为None。 当shuffle为True时, random_state的值影响标签的顺序。

K折交叉验证案例分析

```
from sklearn. model_selection import KFold
import numpy as np
X = ['a','b','c','d']
kf = KFold(n_splits=4)
for train, test in kf. split(X):
    print(train, test)   #打印索引
    print("-"*50)
    print(np. array(X)[train], np. array(X)[test])
    print("*"*50)
```

K折交叉验证案例分析

```
from sklearn. model_selection import KFold
import numpy as np
X = ['a','b','c','d']
kf = KFold(n_splits=4, shuffle=True)
for train, test in kf. split(X):
    print(train, test)   #打印索引
    print("-"*50)
    print(np. array(X)[train], np. array(X)[test])
    print("*"*50)
```

K折交叉验证案例分析

```
from sklearn. model_selection import KFold
import numpy as np
X = ['a','b','c','d']
kf = KFold(n_splits=4, shuffle=True, random_state=1)
for train, test in kf. split(X):
    print(train, test)   #打印索引
    print("-"*50)
    print(np. array(X)[train], np. array(X)[test])
    print("*"*50)
```

计算得分

cross_val_score()函数： 函数用于评估模型的性能

cross_val_score(estimator, X, Y, cv=kfold, scoring='accuracy')：

- **estimator：**：估计器，也就是模型
- **X, Y:** 数据值，标签值
- **cv：** 交叉验证的折数，是一个整数或者是一个交叉验证迭代器
- **scoring：** 评价指标（准确度）

算法比较（一）

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

算法比较（二）

```
# 导入数据
dataset = pd. read_csv('iris.csv')
#显示数据维度
print('数据维度: 行 %s，列 %s' % dataset. shape)
# 查看数据的前10行
print(dataset. head(10))
```

```
数据维度：行 150，列 5
```

```
数据维度：行 150，列 5
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width Species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
5           5.4          3.9           1.7          0.4  setosa
6           4.6          3.4           1.4          0.3  setosa
7           5.0          3.4           1.5          0.2  setosa
8           4.4          2.9           1.4          0.2  setosa
9           4.9          3.1           1.5          0.1  setosa
>>>
```

算法比较（三）

```
print(dataset. describe())  # 统计描述数据信息
# 分离数据集
array = dataset. values
X = array[:, 0:4]
Y = array[:, 4]
kfold = KFold(n_splits=10, shuffle=True, random_state=1)
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

算法比较（四）

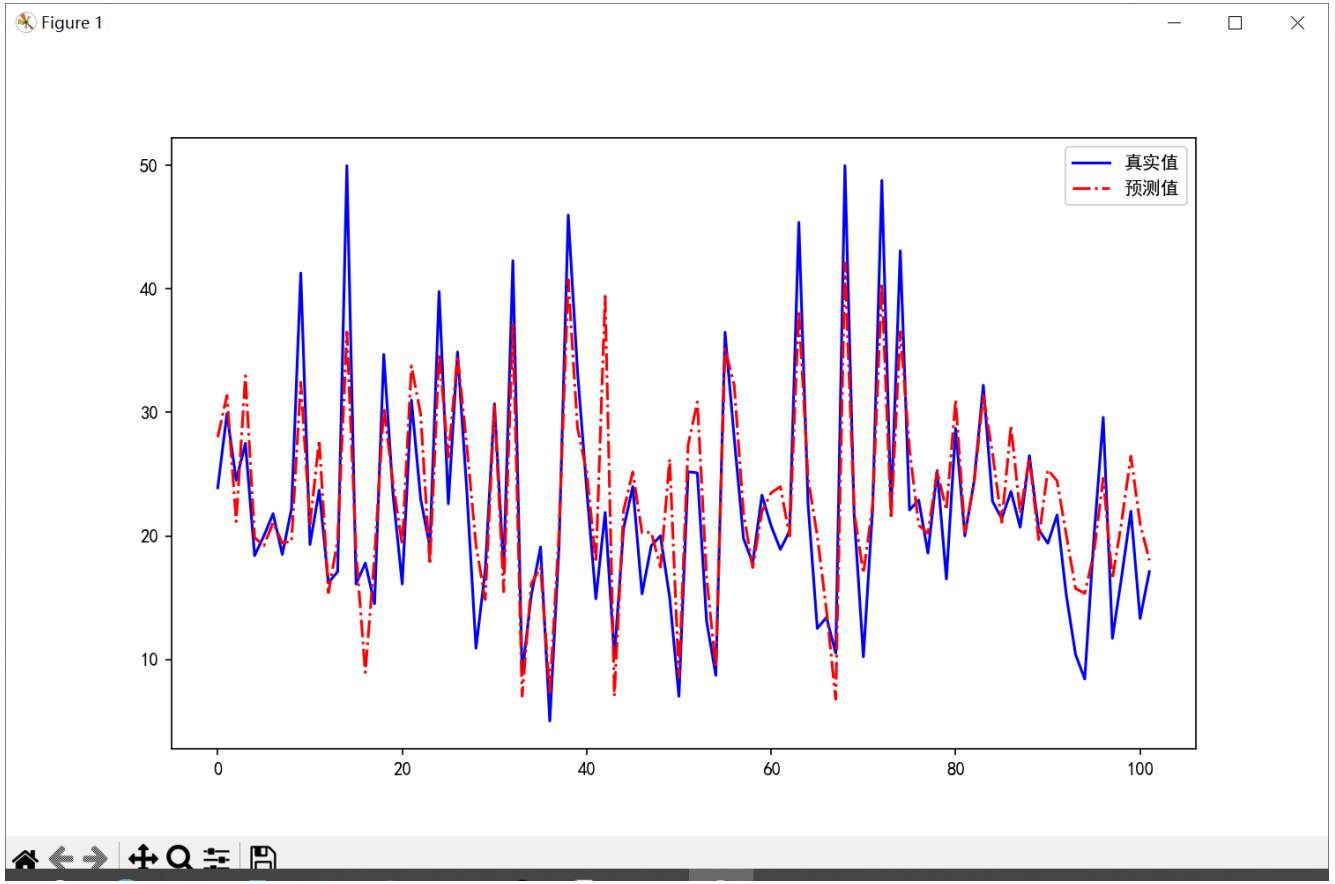
```
# 算法审查
models = {}
models['LR'] = LogisticRegression(max_iter=10000)
models['DT'] = DecisionTreeClassifier()
models['KNN'] = KNeighborsClassifier()
models['NB'] = GaussianNB()
models['SVM'] = SVC()
```

算法比较（五）

```
# 评估算法
results = []
for key in models:
    #cross_val_score:得到K折验证中每一折的得分
    cv_results = cross_val_score(models[key], X, Y, cv=kfold)
    results. append(cv_results)
    print('%s: %f (%f)' %(key, cv_results.mean(), cv_results.std()))
```

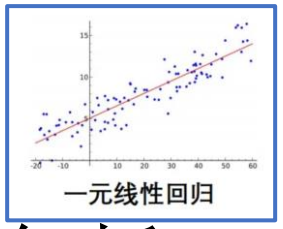
```
LR: 0.953333 (0.042687)
DT: 0.940000 (0.062893)
KNN: 0.960000 (0.044222)
NB: 0.953333 (0.052068)
SVM: 0.960000 (0.032660)
```

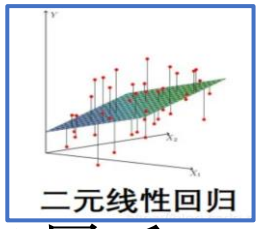
波士顿房价问题



线性回归

线性回归（Linear Regression）： 利用数理统计中回归分析来确定**两种**或**两种以上变量**间相互依赖的定量关系的一种统计分析方法，运用十分广泛。


一元线性回归


二元线性回归

一元线性回归分析： 分析中只包括一个自变量和一个因变量。

多元线性回归分析： 分析中包括两个或两个以上自变量，且因变量和自变量之间是线性关系。

波士顿房价

sklearn提供的波士顿房价数据集统计20世纪70年代中期**波士顿郊区房价**。该数据集包含**506**条记录，**13**个特征指标，第**14列**通常为目标列房价。试图能找到特征指标与房价的关系。

波士顿房价

本例首先将506组数据的数据集划分为训练集和验证集，其中**404组数据是训练样本**，剩下的**102组数据作为验证样本**。然后构建回归模型并训练模型，查看模型的13个特征的系数以及截距，获取模型的预测结果，最后绘制折线图对比预测值和真实。

Python实现线性回归步骤

- 导入对应库
- 加载数据集并划分数据集
- 在训练集上训练线性回归模型
- 使用测试集实现预测
- 绘图输出，结果可视化对比

波士顿房价回归模型

```
# （1）导入库
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import pandas as pd
```

波士顿房价回归模型

```
# （2）加载数据集
boston=pd. read_csv("boston.csv",encoding="gb2312")
x=boston. iloc[:,0:13]
y=boston. iloc[:,13]
# 分割数据为训练集和测试集
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size
=0.2,random_state=1)
print('x_train前3行数据为：', x_train[0:3])
print('y_train前3行数据为：',y_train[0:3])
```

波士顿房价回归模型

```
# （3）创建线性回归模型对象
lr=LinearRegression()
#使用训练集训练模型
lr.fit(x_train,y_train)
#显示模型
print(lr)
print("13个系数:",lr.coef_)
print("模型截距:",lr.intercept_ )
# （4）使用测试集获取预测结果
print("预测结果:",lr.predict(x_test[:5]))
```

```
LinearRegression()
```

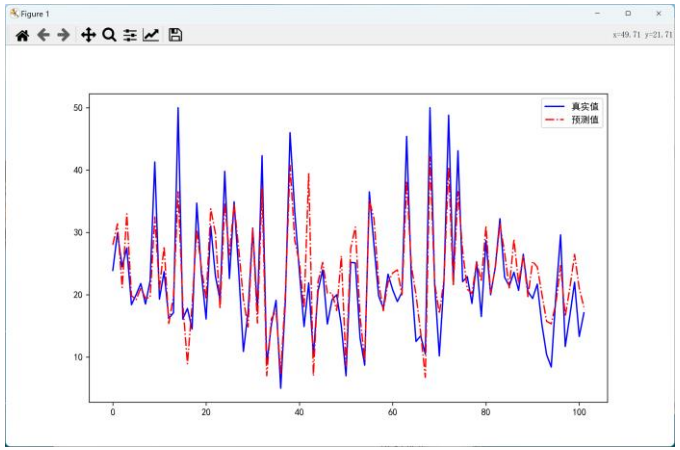
```
13个系数: [-1.01199845e-01  4.67962110e-02 -2.06902678e-02  3.58072311e+00
 -1.71288922e+01  3.92207267e+00 -5.67997339e-03 -1.54862273e+00
  2.97156958e-01 -1.00709587e-02 -7.78761318e-01  9.87125185e-03
 -5.25319199e-01]
```

```
模型截距: 32.428252866991 19
```

```
预测结果: [27.99617259 31.37458822 21.16274236 32.97684211 19.85350998]
```

波士顿房价回归模型

```
# （5）绘图对比预测值和真实值
plt. rcParams['font.sans-serif']='SimHei'
fig=plt. figure(figsize=(10,6))
y_pred=lr. predict(x_test)
plt. plot(range(y_test.shape[0]),y_test,color="blue",linestyle="-")
plt. plot(range(y_test.shape[0]),y_pred,color="red",linestyle="-.")
plt. legend(['真实值','预测值'])
plt. show()
```



数据的归一化

归一化：把每列数据都映射到**0-1**范围之内处理。scikit-learn库提供**preprocessing. MinMaxScaler**类实现了将数据缩放到一个指定的最大值和最小值（通常是1-0）之间的功能。

fit_transform()函数： MinMaxScaler类的fit_transform()函数用于把转换器实例应用到数据上，并返回转换后的数据。

波士顿房价数据归一化

对boston数据集的前5行5列数据进行归一化并打印显示。

```
import pandas as pd
from sklearn. preprocessing import MinMaxScaler
boston=pd. read_csv("boston.csv", encoding="gb2312")
X = boston. iloc[:5, :5]
minmax_scaler = MinMaxScaler()                    # 转换器实例化
boston_minmax = minmax_scaler. fit_transform(X)   # 数据归一化
print(boston_minmax)
```

波士顿房价数据归一化结果

	A	B	C	D	E
1	犯罪率	区	产业	查尔斯河	氧化氮
2	0.00632	18	2.31	0	0.538
3	0.02731	0	7.07	0	0.469
4	0.02729	0	7.07	0	0.469
5	0.03237	0	2.18	0	0.458
6	0.06905	0	2.18	0	0.458

```
[[0.          1.          0.02658487  0.          1.        ]
 [0.33460864  0.          1.          0.          0.1375    ]
 [0.33428981  0.          1.          0.          0.1375    ]
 [0.4152718   0.          0.          0.          0.        ]
 [1.          0.          0.          0.          0.        ]]
>>>
```

数据的标准化

数据标准化： scikit-learn库提供了对数据进行标准化处理的函数，包括Z-score标准化、稀疏数据标准化和带离群值的标准化。

Z-Score标准化： scikit-learn库中**preprocessing. StandardScaler**类实现了Z-Score标准化。

Z-Score公式： **z = (x-平均值) / 标准差**

波士顿房价数据标准化

对boston数据集的前5行5列数据进行标准化并打印显示。

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
boston=pd. read_csv("boston.csv", encoding="gb2312")
X = boston.iloc[:5, :5]
standerd_scaler = StandardScaler()                    # 转换器实例化
boston_standerd = standerd_scaler. fit_transform(X)   # 数据标准化
print(boston_standerd)
```

波士顿房价数据标准化结果

	A	B	C	D	E
1	犯罪率	区	产业	查尔斯河	氧化氮
2	0.00632	18	2.31	0	0.538
3	0.02731	0	7.07	0	0.469
4	0.02729	0	7.07	0	0.469
5	0.03237	0	2.18	0	0.458
6	0.06905	0	2.18	0	0.458

```
[[-1.2834352    2.         -0.77983987  0.          1.97329359]
 [-0.25317266  -0.5         1.22450018  0.         -0.31122416]
 [-0.25415433  -0.5         1.22450018  0.         -0.31122416]
 [-0.00481018  -0.5        -0.83458025  0.         -0.67542264]
 [ 1.79557237  -0.5        -0.83458025  0.         -0.67542264]]
>>>
```

数据的正则化

数据正则化： scikit-learn库提供了对数据进行正则化处理的函数，其中**preprocessing. Normalizer**类实现了将单个样本缩放到单位范数的功能。

正则化应用： 在数据集之间各个指标有共同重要比率的关系时，正则化处理有比较好的效果。

波士顿房价数据正则化

对boston数据集的前5行5列数据进行正则化并打印显示。

```
import pandas as pd
from sklearn.preprocessing import Normalizer
boston=pd. read_csv("boston.csv", encoding="gb2312")
X = boston. iloc[:5, :5]
normalizer_scaler = Normalizer()                    # 转换器实例化
boston_normalizer = normalizer_scaler. fit_transform(X) # 数据正则化
print(boston_normalizer)
```

波士顿房价数据正则化结果

	A	B	C	D	E
1	犯罪率	区	产业	查尔斯河	氧化氮
2	0.00632	18	2.31	0	0.538
3	0.02731	0	7.07	0	0.469
4	0.02729	0	7.07	0	0.469
5	0.03237	0	2.18	0	0.458
6	0.06905	0	2.18	0	0.458

```
[[3.48102083e-04 9.91429984e-01 1.27233515e-01 0.00000000e+00
  2.96327406e-02]
 [3.85430066e-03 0.00000000e+00 9.97799549e-01 0.00000000e+00
  6.61906632e-02]
 [3.85147808e-03 0.00000000e+00 9.97799560e-01 0.00000000e+00
  6.61906639e-02]
 [1.45298555e-02 0.00000000e+00 9.78532126e-01 0.00000000e+00
  2.05581520e-01]
 [3.09827227e-02 0.00000000e+00 9.78165612e-01 0.00000000e+00
  2.05504518e-01]]
>>>
```

标签二值化

标签二值化：可以把**非数字化**的数据标签转化为**数字化形式**的数据标签，例如可把“Yes”和“No”等文本标签转化为“1”和“0”的数字形式。scikit-learn库中preprocessing. **LabelBinarizer类**实现了标签二值化处理的功能，常用于文本类型的数据标签的处理。

标签二值化案例

构建数据集，进行标签二值化处理并打印显示。

```
from sklearn import preprocessing
label = ['Yes', 'No', 'Yes', 'No', 'No'] # 设置数据集
lb = preprocessing. LabelBinarizer()   # 转换器实例化
label_bin = lb. fit_transform(label)  # 标签数据二值化
print(label_bin)
```

```
[[1]
 [0]
 [1]
 [0]
 [0]]
>>>
```

机器学习分类

- 有监督学习（分类、回归）
- 无监督学习（聚类、降维）
- 强化学习
- 半监督学习

聚 类

聚类(Clustering Approach)：是按一定的距离或相似性系数将数据分成一系列相互区分的组，常用的经典聚类方法有K-means, K-medoids，isodata等。

聚类算法的应用场景：市场分析、商业经营、图像处理、决策支持、模式识别。

K-Means聚类算法

- K-Means算法属于聚类分析中划分方法里较为经典的一种，由于该算法的效率高，所以在对大规模数据进行聚类时被广泛应用。
- K-Means算法通过将样本划分k个簇类来实现数据聚类，该算法需要指定划分类的个数。

K-Means聚类算法

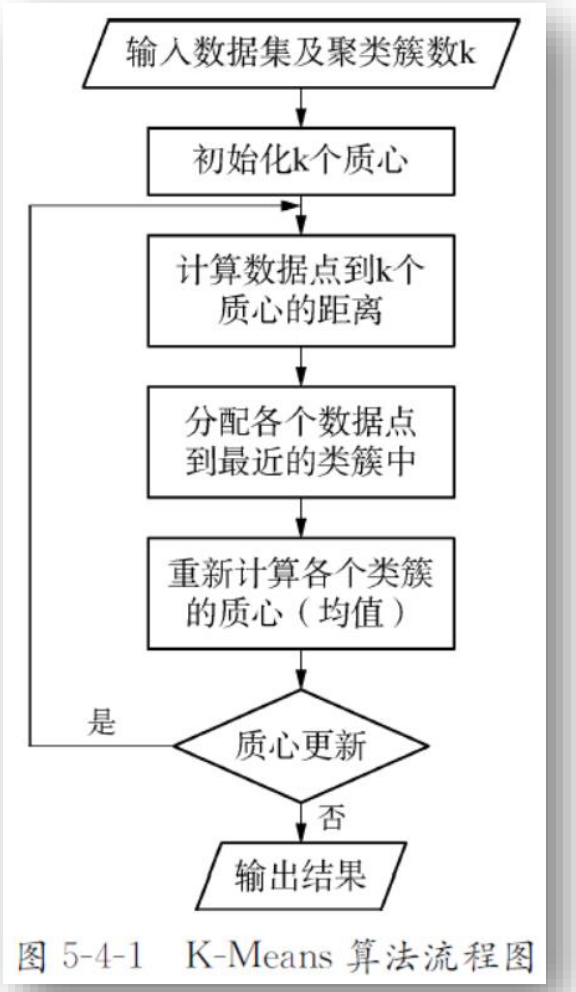


图 5-4-1　K-Means 算法流程图

K- Means算法示例

例： 对表中二维数据，使用k- means算法将其划分为2个簇，假设初始簇中心选为P7(4,5),P10(5,5)。

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
X	3	3	7	4	3	8	4	4	7	5
y	4	6	3	7	8	5	5	1	4	5

K- Means算法示例

- 根据题目,假设划分的两个簇分别为C1和C2,中心分别为(4,5)和(5,5),下面计算10个样本到这2个簇中心的距离,并将10个样本指派到与其最近的簇;

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
X	3	3	7	4	3	8	4	4	7	5
y	4	6	3	7	8	5	5	1	4	5

- 第一轮迭代结果如下:

属于簇C1的样本有:{P7,P1,P2,P4,P5,P8};

属于簇C2的样本有:{P10,P3,P6,P9};

- 重新计算新的簇中心,有:C1的中心为(3.5,5.167),C2的中心为(6.75,4.25);

K- Means算法示例

- 继续计算10个样本到新的簇的中心的距离，重新分配到新的簇中，第二轮迭代结果如下：

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
X	3	3	7	4	3	8	4	4	7	5
y	4	6	3	7	8	5	5	1	4	5

属于簇C1的样本有:{P1,P2,P4,P5,P7,P10}；

属于簇C2的样本有:{P3,P6,P8,P9}；

- 重新计算新的簇的中心，有:C1的中心为(3.67，5.83)，C2的中心为(6.5,3.25)；
- 继续计算10个样本到新的簇的中心的距离，重新分配到新的簇中，发现簇中心不再发生变化，算法终止。

K- Means聚类算法

Scikit-learn的Cluster类提供聚类分析的方法：

- 模型初始化

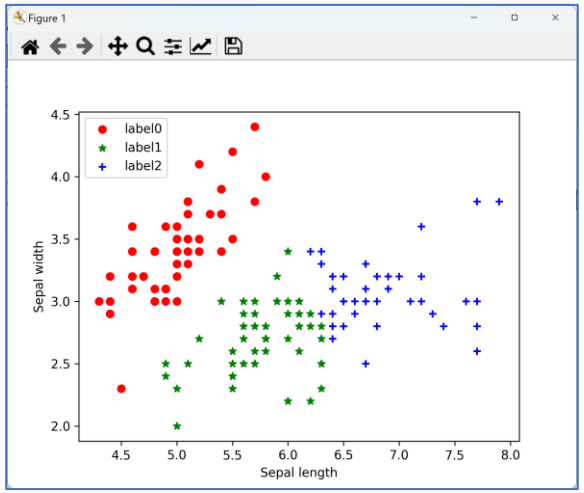
  kmeans=**Kmeans**(n_clusters)　#参数为簇的个数

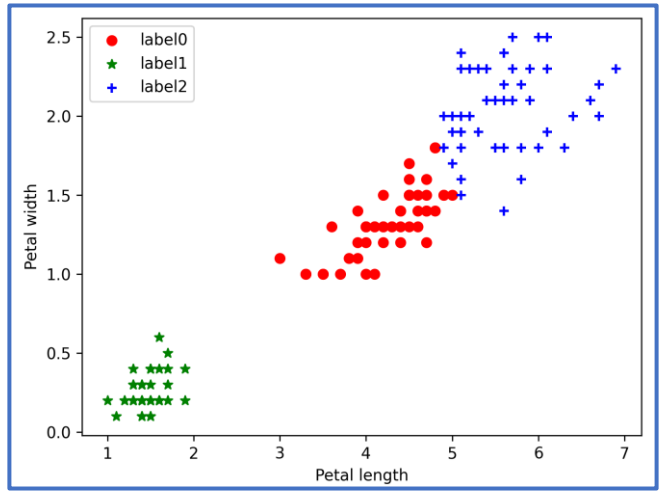
- 模型学习

  kmeans. **fit**(X)　　#参数为样本二维数组

鸢尾花聚类问题

	A	B	C	D
1	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
2	5.1	3.5	1.4	0.2
3	4.9	3	1.4	0.2
4	4.7	3.2	1.3	0.2
5	4.6	3.1	1.5	0.2
6	5	3.6	1.4	0.2
7	5.4	3.9	1.7	0.4
8	4.6	3.4	1.4	0.3
9	5	3.4	1.5	0.2
10	4.4	2.9	1.4	0.2
11	4.9	3.1	1.5	0.1
12	5.4	3.7	1.5	0.2
13	4.8	3.4	1.6	0.2





K- Means聚类算法

Scikit-learn的Cluster类提供聚类分析的方法：

- 模型初始化

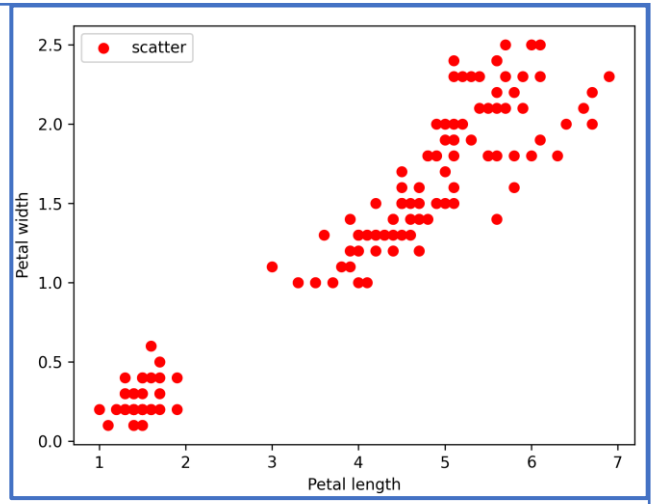
```
kmeans=Kmeans(n_clusters)    #参数为簇的个数
```

- 模型学习

```
kmeans. fit(X)                    #参数为样本二维数组
```

鸢尾花问题K- Means模型（1）

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
import pandas as pd   #导入模块
iris = pd.read_csv("iris.csv")
X = iris.loc[:,['Petal_Length', 'Petal_Width']] #读出数据
plt.scatter(X['Petal_Length'], X['Petal_Width'], c = "red", marker='o', label='scatter')
plt.xlabel('Petal length')
plt.ylabel('Petal width')
plt.legend(loc=2)
plt.show()
```

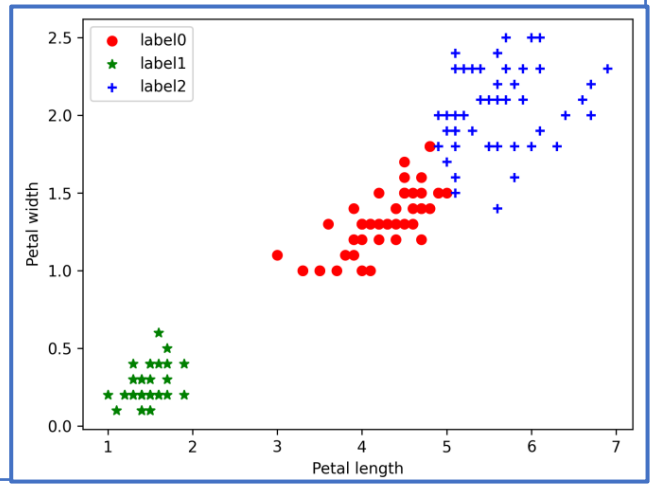


鸢尾花问题K- Means模型（2）

```
estimator = KMeans(n_clusters=3)#模型初始化
estimator.fit(X) #模型学习
label_pred = estimator.labels_ #获取聚类标签
x0 = X[label_pred == 0]
x1 = X[label_pred == 1]
x2 = X[label_pred == 2]
print(x0)
print(x1)
print(x2)
```

鸢尾花问题K- Means模型（3）

```
plt.scatter(x0['Petal_Length'], x0['Petal_Width'], c = "red", marker='o', label='label0')
plt.scatter(x1['Petal_Length'], x1['Petal_Width'], c = "green", marker='*', label='label1')
plt.scatter(x2['Petal_Length'], x2['Petal_Width'], c = "blue", marker='+', label='label2')
plt.xlabel('Petal length')
plt.ylabel('Petal width')
plt.legend(loc=2)
plt.show()
```



题型及分值

考试题型如下：

选择题：2分*20题=40分，涵盖全部教学内容

程序填空题：2分*3空*5题=30分，涵盖全部教学内容

编程题：10分*3题=30分，涵盖重点、难点教学内容

复习重点：窗体设计、matplotlib图形绘制(散点图、折线图、饼图、条形图等)、numpy数据处理、pandas数据分析、数据清洗、文献词频统计、正则与MySQL数据库、机器学习算法应用(分类、回归、聚类)

谢　谢