哈尔滨工业大学（深圳）

统计机器学习
实验指导书

实验一 构建感知机模型实现鸢尾花数据的分类

目录

1. 实验目的

1. 学会理解数据并对数据进行预处理；
2. 理解感知机模型的原理并掌握其构建方法。

2. 实验内容

- **任务一：用 Python 自编程实现鸢尾花分类。要求：**

1、对鸢尾花 setosa 和 virginica 两个品种做分类；

2、使用 matplotlib 画图做分析，选取 2 个合适的特征；

3、根据伪代码编写梯度下降法程序；

4、采用函数式编写 python 代码，重要代码加上注释；

5、结果绘图（带分类线）。

- **任务二：用 Sklearn 库内的 Perceptron 分类器实现鸢尾花分类。要求：**

1、对鸢尾花 setosa 和 virginica 两个品种做分类；

2、使用 matplotlib 画图做分析，选取 2 个合适的特征；

3、调用 sklearn 库完成感知机模型的定义与训练；

4、调整感知机模型参数，使得测试结果的准确率为 100%；

5、采用函数式编写 python 代码，重要代码加上注释；

6、结果绘图（带分类线）。

- **思考题**

1、在做数据处理时，为什么要转化为 dataFrame 格式来处理，不能直接用 numpy 吗？

2、怎样挑选合适的特征来做分类，理由是什么？

3、为什么要使用随机种子做数据分割？

4、使用 sklearn 库来编码，学习率对迭代过程和最终结果有无影响？若有/无影响的话，条件是什么？

5、本次实验能对 versicolor 和 virginica 两种鸢尾花做分类吗？能的话，实现出来；不能的话，说明理由。

3. 实验环境

- Python3.7 + PyCharm / Anaconda

4. 实验步骤

下面以选取 setosa 和 versicolor 两类鸢尾花数据，选用 sepal length 和 sepal width 两个变量为样例，来做感知机模型的分类任务。

4.1 任务一（python 自编程）

4.1.1 准备数据

（1）导入必要的包

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
```

（2）加载数据集

```
iris = load_iris()
print(iris.data.shape)  # data对应了样本特征
print(iris.target.shape)  # target对应了样本的类别（目标属性）
print(iris.target)  # 显示所有样本的目标属性
print(iris.target_names)  # 显示所有样本的目标属性名称
print(iris.feature_names) # 显示样本中的4个特征名称
```

输出结果为：

```
(150, 4)
(150,)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
['setosa' 'versicolor' 'virginica']
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

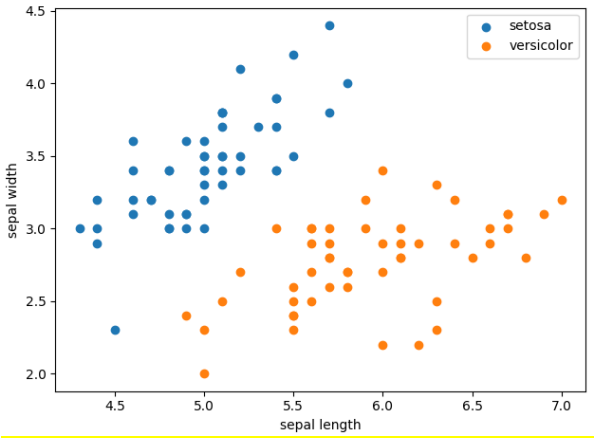
（3）将列表式的数据转化为转换为 DataFrame

```
# 将鸢尾花4个特征，以4列存入pandas的数据框架
df = pd.DataFrame(iris.data, columns=iris.feature_names)
# 在最后一列追加 加入（目标值）列数据
df['label'] = iris.target
# 显示df每一行的标签
df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']
print(df)
```

（4）原数据可视化

```
plt.scatter(df[:50]['sepal length'], df[:50]['sepal width'], label='setosa')
plt.scatter(df[50:100]['sepal length'], df[50:100]['sepal width'], label='versicolor')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend()
plt.show()
```

画图结果：



现象： sepal length 和 sepal width 两个变量沿着一条“瘦”直线排列，所以是强相关的。

结论：选取这两个变量对 setosa 和 versicolor 两类鸢尾花实现能线性分类。

（5）数据切片（选取前 100 行，为 setosa 和 versicolor 两类鸢尾花数据）

```
# 选取数据，前100行，前两个特征，最后一列的目标值
data = np.array(df.iloc[:100, [0, 1, -1]])
# 生成感知机的标签值，+1， -1，第一种 - 1，第二种 + 1
for i in range(len(data)):
    if data[i,-1] == 0:
        data[i,-1] = -1
print(data)
```

4.1.2 定义和训练模型

（1）先定义一个 Model 类

```
# 数据线性可分，二分类数据
class Model:
    def __init__(self,data):
        self.w = np.ones(len(data[0]) - 1, dtype=np.float32)
        self.b = 0
        self.l_rate = 0.1


    def sign(self, x, w, b):
        y = np.dot(x, w) + b
        return y

    # 随机梯度下降法
    def fit(self, X_train, y_train):
        """请根据右侧的伪代码，自行编写程序，计算出w和b"""
```

伪代码

输入：训练数据集$T=\{(x_1, y_1)\cdots(x_n, y_n)\}$

(a)选出初始值w_0，b_0以及学习率 η;

(b)在训练数据集中选取数据(x_1, y_1)

(c)如果$y_i(wx_i+b)\leq 0$:

$$w=w+\eta y_i x_i$$

$$b=b+\eta y_i$$

(d) 转至（b），直到训练集中没有误分类点

任务：根据伪代码，自行完成红框内的编程。

（2）后调用

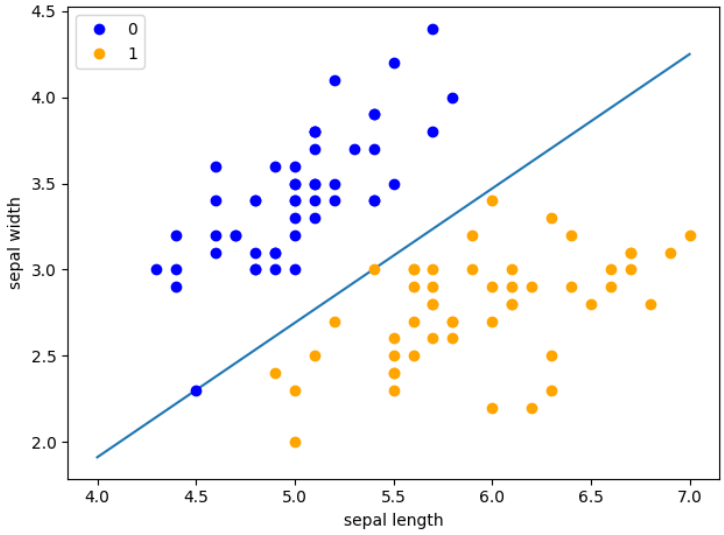
```
#构造感知机对象，对数据集进行训练，得出模型参数
perceptron = Model(data)
perceptron.fit(X, y)
```

4.1.3 绘制图像

```
x_points = np.linspace(4, 7, 10)
y_ = -(perceptron.w[0] * x_points + perceptron.b) / perceptron.w[1]
plt.plot(x_points, y_)

plt.plot(data[:50, 0], data[:50, 1], 'bo', color='blue', label='0')
plt.plot(data[50:100, 0], data[50:100, 1], 'bo', color='orange', label='1')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend()
plt.show()
```

画图结果：



注意：（4.5 2.3）、（5.4 3.1）这两个点只是看起来近似在分类线上，实际上代入前面感知机模型，分别得到的是<0 ，>0，所以是能线性分类的。

4.2 任务二（使用 Sklearn 库来编程）

4.2.1 准备数据

同 4.1.1 的前（1）-（5）一样，本小节需要分测试集和训练集。

（6）数据分割

```
# X是除最后一列外的所有列，y是最后一列
X, y = data[:, :-1], data[:, -1]
# 调用sklearn的train_test_split方法，将数据随机分为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X,   #被划分的样本特征集
                                                    y,   #被划分的样本目标集
                                                    test_size=0.3, #测试样本占比
                                                    random_state=1) #随机数种子
```

参数 test_size：样本占比，如果是整数的话就是样本的数量

random_state：是随机数的种子。其实就是该组随机数的编号，在需要重复试验的时候，保证得到一组一样的随机数。比如你每次都填 1，其他参数一样的情况下你得到的随机数组是一样的。但填 0 或不填，每次都会不一样。

4.2.2 配置模型

调用 sklearn 库函数

```
#clf = Perceptron()  #定义感知机
clf = Perceptron(fit_intercept=False, max_iter=1000, shuffle=False)
```

各个参数对应的含义如下：

序号	部分重要参数	默认值	可选值
1	fit_intercept(计算模型的截距)	True	为False时，则数据中心化处理
2	max_iter(迭代次数)	1000	如果tol不为None，则为1000
3	tol (终止条件)	None	(previous_loss -loss)<tol, 比如 tol=1e-3
4	shuffle(每次迭代后清洗训练数据)	True	False
5	eta(学习率)	1	(0,1]
6	penalty (正则化项)	None	'l2'or 'l1' or 'elasticnet'
7	alpha (正则化系数)	0.0001	

其中最重要的参数是学习率和迭代次数。

4.2.3 训练模型

```
clf.fit(X_train, y_train)    #使用训练数据进行训练
```

4.2.4 评估模型

（1）调用方法，计算模型的准确率

```
#计算模型的权重、截距、迭代次数
print("特征权重：", clf.coef_)  # 特征权重 w
print("截距（偏置）:", clf.intercept_)  # 截距 b
print("迭代次数:", clf.n_iter_)
#评价模型
print(clf.score(X_test, y_test))
```

输出结果为：

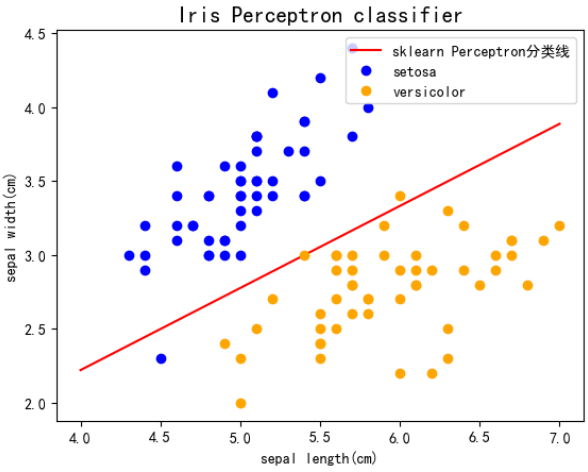
```
特征权重： [[ 31.7 -57.1]]
截距（偏置）: [0.]
迭代次数: 30
0.9666666666666667
```

（2）绘制图形，观察分类效果

```
x_ponits = np.arange(4, 8)
y_ = -(clf.coef_[0][0] * x_ponits + clf.intercept_) / clf.coef_[0][1]
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.plot(x_ponits, y_, 'r', label='sklearn Perceptron分类线')

plt.plot(data[:50, 0], data[:50, 1], 'bo', color='blue', label='setosa')
plt.plot(data[50:100, 0], data[50:100, 1], 'bo', color='orange', label='versicolor')
plt.xlabel('sepal length(cm)')
plt.ylabel('sepal width(cm)')
plt.title('Iris Perceptron classifier', fontsize=15)
plt.legend()
plt.show()
```

画图结果为：



另外，主函数代码如下：

```
if __name__ == '__main__':
    #加载数据集
    df = create_df()
    # 原数据可视化
    show_image(df)
    #数据切片
    data=create_data(df)
    #数据分割
    # X是除最后一列外的所有列，y是最后一列
    X, y = data[:, :-1], data[:, -1]
    # 调用sklearn的train_test_split方法，将数据随机分为训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X,  #被划分的样本特征集
                                                        y,  #被划分的样本目标集
                                                        test_size=0.3, #测试样本占比
                                                        random_state=1) #随机数种子
    # 定义感知机
    clf = Perceptron(fit_intercept=False, max_iter=1000, shuffle=False)
    # 使用训练数据进行训练
    clf.fit(X_train, y_train)
    #计算模型的权重、截距、迭代次数
    print("特征权重：", clf.coef_)  # 特征权重 w
    print("截距（偏置）:", clf.intercept_)  # 截距 b
    print("迭代次数:", clf.n_iter_)
    #评价模型
    print(clf.score(X_test, y_test))
    #绘制图形，观察分类结果
    show(clf, data)
```

5. 附录（Python 常用机器学习库）

5.1 Numpy

NumPy 是 Numerical Python 的简称，是高性能计算和数据分析的基础包。包括：

1. 一个强大的 N 维数组对象 ndarray；
2. 比较成熟的（广播）函数库；
3. 用于整合 C/C++和 Fortran 代码的工具包；
4. 实用的线性代数、傅里叶变换和随机数生成函数。

5.1.1 安装 Numpy 库

打开 cmd 命令行窗口执行：

```
pip install numpy
```

5.1.2 ndarray 对象

Numpy 的强大功能主要基于底层的一个 ndarray 结构，其可以生成 N 维数组对象。ndarray 内部构成：

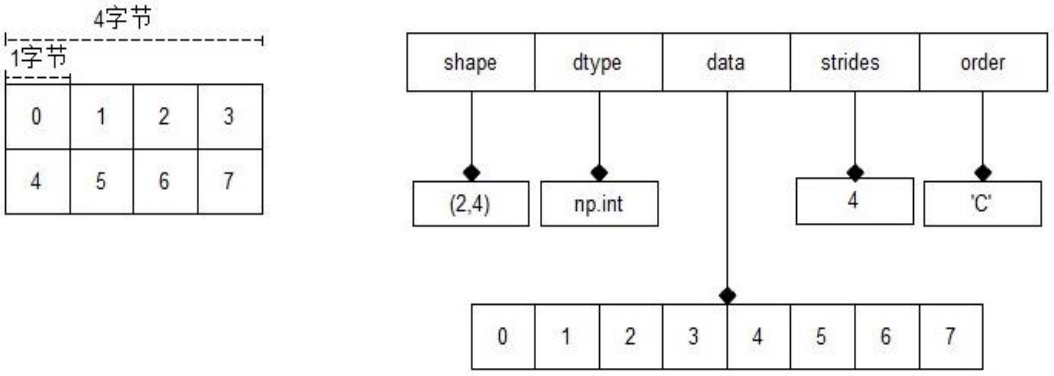
1)数组形状 shape：一个表示数组各维大小的整数元组。

2)数组数据 data：一个指向内存中数据的指针。

3)数据类型 dtype：一个描述数组的类型对象。

4)跨度 strides：一个元组，表示当前维度移动到下一个位置需要跨越的字节数。

5)数组顺序 order：访问数组元素的主顺序，如“C”为行主序，“F”为列主序等。



ndarray 的数据结构

5.1.3 创建 ndarray

array 函数：

numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

参数说明：

名称	描述
object	数组或嵌套的数列
dtype	数组元素的数据类型，可选
copy	对象是否需要复制，可选
order	创建数组的样式，C为行方向，F为列方向，A为任意方向（默认）
subok	默认返回一个与基类类型一致的数组
ndimin	指定生成数组的最小维度

【例】建立一个一维 ndarray 数组

```
import numpy as np
a = np.array([1,2,3])
print(a)
```

【例】创建二维数组。

```
import numpy as np
a = np.array([[1,2], [3,4]])
print(a)
```

【例】使用 ndmin 参数设置数组的最小维度。

```
import numpy as np
a = np.array([1,2,3,4,5], ndmin=2)
print(a)
```

【例】使用 dtype 参数设置为数组类型为复数。

```
import numpy as np
a = np.array([1,2,3], dtype = np.complex)
print(a)
```

5.1.4 Numpy 数据类型

Numpy 内置了 24 种数组标量（array scaler）类型，也支持 Python 的基本数据类型。

名称	描述
bool_	布尔型，True或False
int8	有符号字节类型，范围为 -128~127
int16	有符号16位整数，范围为 -32768~ 32767
int32	有符号32位整数，范围为 -2^{31}~ 2^{31}-1
int64	有符号64位整数，范围为 -2^{63}~2^{63}-1
uint8	无符号字节类型，范围为 0 ~ 255
uint16	无符号16位整数，范围为 0 ~ 65535
uint32	无符号32位整数，范围为 0 ~ 2^{32}-1
uint64	无符号64位整数，范围为 0 ~ 2^{64}-1
float_	64位浮点数，同float64
float16	16位浮点数
float32	32位浮点数
float64	64位（双精度）浮点数，同float_
complex_	128 位复数，同complex128
complex64	32位复数
complex128	128位复数，同complex_

数据类型对象 (dtype)

Numpy 中的 dtype(data type object)是由 nump.dtype 类产生的数据类型对象，其作用是描述数组元素对应的内存区域的使用。其内部结构包括数据类型、数据的字节数、各组成部分的顺序、各字段的名称等。

【例】使用 dtype 对象设置数据类型。

```
import numpy as np
x=np.array(5,dtype="float32")
print('x为:',x)
print('x对象的data属性:  ',x.data)
print('x对象的size属性:',x.size)
print('x对象的维数:',x.ndim)
y=np.array(x,dtype="bool_")
print('转换为bool类型的x为:',y)
z=np.array(y,dtype="float16")
print('True值转换为float16类型为:',z)
```

运行结果：

```
x为: 5.0
x对象的data属性:  <memory at 0x0000019378D2E048>
x对象的size属性: 1
x对象的维数: 0
转换为bool类型的x为: True
True值转换为float16类型为: 1.0
```

5.1.5 Numpy 数组属性

ndarray.shape 代表数组的维度，返回值为一个元组。

【例】显示数组的维度。

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print (a.shape)
```

【例】调整数组大小。

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
a.shape =  (3,2)
print (a)
```

其他几个常见的数组属性（同学们可以自行尝试下）

用法	说明
ndarray.size	数组元素个数
ndarray.ndim	数组维度
ndarray.dtype	数组元素类型

5.1.6 其他创建数组的方式

（1）**numpy.empty** 创建一个指定形状（shape）、数据类型（dtype）且未初始化的数组。格式：numpy.empty(shape, dtype = float, order = 'C')

【例】创建一个空数组

```
import numpy as np
x = np.empty([3,2], dtype = int)
print (x)
```

（2）**numpy.zeros** 创建指定大小的数组，以 0 填充。

格式：umpy.zeros(shape, dtype = float, order = 'C’)

【例】创建一个全 0 数组。

```
import numpy as np
# 默认为浮点数
x = np.zeros(5)
print(x)
# 设置类型为整数
y = np.zeros((5,), dtype = np.int)
print(y)
# 自定义类型
z = np.zeros((2,2), dtype = [('x', 'i4'), ('y', 'i4')])
print(z)
```

（3）**numpy.ones** 创建指定形状的数组，数组元素以 1 来填充。

【例】建立一个全 1 数组。

```
import numpy as np
# 默认为浮点数
x = np.ones(5)
print(x)
# 自定义类型
x = np.ones([2,2], dtype = int)
print(x)
```

5.1.7 切片、迭代和索引

（1）切片（slice） 对 ndarray 进行切片操作与一维数组相同，用索引标记切片的起始和终止位置。

【例】二维数组 ndarray 的切片

```
import numpy as np
# 创建一个4行6列的二维数组
arr = np.arange(24).reshape(4,6)
print('arr =\n',arr)
# 截取第2行到最后一行，第1列到第4列构成的ndarray
arr1 = arr[1:, :3]
print('B = \n',arr1)
```

（2）迭代（iteration）

(a) ndarray 也可以通过 for 循环来实现迭代。当维数多于一维时，迭代操作使用嵌套的 for 循环。

【例】使用嵌套 for 循环对 ndarray 数组进行迭代遍历。

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
for xline in a:
    for yitem in xline:
        print(yitem,end=' ')
```

（b）Numpy 还包含一个循环迭代器类 numpy.nditer,所生成的迭代器(Iterator)对象是一个根据位置进行遍历的对象。

【例】使用 nditer 对象对 ndarray 数组进行迭代。

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print(a)
print(np.nditer(a))
for x in np.nditer(a):
    print(x,end=' ')
```

5.1.8 Numpy 计算

【例】简单条件运算。

```
import numpy as np
stus_score = np.array([[80, 88], [82, 81], [84, 75], [86, 83], [75, 81]])
result=[stus_score> 80]
print(result)
```

【例】np.where 函数实现数据筛选。

```
import numpy as np
num = np.random.normal(0, 1, (3,4))
print(num)
num[num<0.5]=0
print(num)
print(np.where(num>0.5,1,0))
```

【例】ndarray 的统计计算。

```
stus_score = np.array([[80, 88], [82, 81], [84, 75], [86, 83], [75, 81]])
# 求每一列的最大值(0表示列)
result = np.amax(stus_score, axis=0)
print(result)
# 求每一行的最大值(1表示行)
result = np.amax(stus_score, axis=1)
print(result)
# 求每一行的最小值(1表示行)
result = np.amin(stus_score, axis=1)
print(result)
# 求每一列的平均值(0表示列)
result = np.mean(stus_score, axis=0)
print(result)
```

5.2 Pandas

5.2.1 安装 Pandas 库

打开 cmd 命令行窗口执行：

```
pip install pandas
```

5.2.2 Pandas 核心数据结构

维数	名称	描述
1	Series	带标签的一维同构数组
2	DataFrame	带标签的，大小可变的，二维异构表格

Series 是带标签的一维数组，可存储整数、浮点数、字符串、Python 对象等类型的数据。轴标签统称为索引。Series 中只允许存储相同的数据类型，这样可以更有效的使用内存，提高运算效率。调用 pd.Series 函数即可创建 Series。

DataFrame 是一个表格型的数据结构，类似于 Excel 或 sql 表，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔值等）。DataFrame 既有行索引也有列索引，

它可以被看做由 Series 组成的字典（共用同一个索引），可用多维数组字典、列表字典生成 DataFrame。

5.2.3 Series 对象

由一组数据以及一组与之相关的数据标签（即索引）组成。

（1）创建 Series 对象 pd.Series(data, index)

其中 data 表示数据值，index 是索引，缺省情况下是 0 到 N-1(N 为数据的长度)的整数型索引。访问 Series 对象成员可以用索引编号，也可以按索引名。

【例】创建一个 Series 对象。

```
import pandas as pd
s = pd.Series([1,3,5,9,6,8])
print(s)
```

【例】为一个地理位置数据创建 Series 对象。

```
import pandas as pd
#使用列表创建，索引值为默认值。
s1=pd.Series([1,1,1,1,1])
print(s1)
#使用字典创建，索引值为字典的key值
s2=pd.Series({'Longitude':39,'Latitude':116,'Temperature':23})
print('First value in s2:',s2['Longitude'])
#使用range函数生成的迭代序列设置索引值
s3=pd.Series([3.4,0.8,2.1,0.3,1.5],range(5,10))
print('First value in s3:',s3[5])
```

5.2.4 DataFrame 对象

DataFrame 是一个表格型的数据结构。列索引（columns）对应字段名，行索引（index）对应行号，值（values）是一个二维数组。每一列表示一个独立的属性，各个列的数据类型（数值、字符串、布尔值等）可以不同。

DataFrame 既有行索引也有列索引，所以 DataFrame 也可以看成是 Series 的容器。

（1）创建 DataFrame 对象 DataFrame([data, index, columns, dtype, copy])

【例】创建 DataFrame

```
import numpy as np
import pandas as pd
dates= pd.date_range('20160101',periods=6)
df=pd.DataFrame(np.random.randn(6,4), index=dates, columns=['a','b','c','d'])
print(df)
```

运行结果为：

```
                   a         b         c         d
2016-01-01  0.547597  0.053966 -1.108513  0.418538
2016-01-02 -0.555191 -1.004182  0.704419  0.343653
2016-01-03  0.455266  0.700746 -1.246836 -0.616007
2016-01-04 -0.134394 -0.097616  0.820291 -0.703414
2016-01-05  0.810092  0.291390 -0.855088 -0.851648
2016-01-06 -0.451751  1.413834  2.006766  0.325187
```

【例】创建一组没有给定行标签和列标签的数据

```
df1 = pd.DataFrame(np.arange(12).reshape((3,4)))
print(df1)
```

```
   0  1   2   3
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11
```

（2）访问 DataFrame 对象

可以通过索引对 DataFrame 进行访问，可以获取其中的一个或多个行和/或列。

先创建一个 df 对象

```
df = pd.DataFrame({'A' : 1., 'B' : pd.Timestamp('20130102'),
                   'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
                   'D' : np.array([3] * 4,dtype='int32'),
                   'E' : pd.Categorical(["test","train","test","train"]), 'F' : 'foo'})
```

- 查看数据详情：df

```
print(df)
```

```
     A          B    C  D      E    F
0  1.0 2013-01-02  1.0  3   test  foo
1  1.0 2013-01-02  1.0  3  train  foo
2  1.0 2013-01-02  1.0  3   test  foo
3  1.0 2013-01-02  1.0  3  train  foo
```

- 查看数据类型： df.dtypes

```
print(df.dtypes)
```

```
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

- 查看列的序号： df.index

```
print(df.index)
```

```
Int64Index([0, 1, 2, 3], dtype='int64')
```

- 查看每种数据的名称：df.columns

```
print(df.columns)
```

```
Index(['A', 'B', 'C', 'D', 'E', 'F'], dtype='object')
```

- 想看所有的值：df.values
- 查看数据描述：df.describe()

- 对数据的行/列进行排序：df.sort_index()

```
print(df.sort_index(axis=1, ascending=False))
```

```
     F      E  D    C          B    A
0  foo   test  3  1.0 2013-01-02  1.0
1  foo  train  3  1.0 2013-01-02  1.0
2  foo   test  3  1.0 2013-01-02  1.0
3  foo  train  3  1.0 2013-01-02  1.0
```

- 对数据的值进行排序：df.sort_values()

```
print(df.sort_values(by='B'))
```

```
     A          B    C  D      E    F
0  1.0 2013-01-02  1.0  3   test  foo
1  1.0 2013-01-02  1.0  3  train  foo
2  1.0 2013-01-02  1.0  3   test  foo
3  1.0 2013-01-02  1.0  3  train  foo
```

5.2.5 Pandas 库基本操作

先创建一个 6X4 的矩阵数据

```
import pandas as pd
dates = pd.date_range('20130101', periods=6)
df = pd.DataFrame(np.arange(24).reshape((6,4)),index=dates, columns=['A','B','C','D'])
```

（1）**选择数据**

- 简单筛选（如按行/列，index，跨越多行或多列筛选）

```
print(df['20130102':'20130104'])
```

```
             A   B   C   D
2013-01-02   4   5   6   7
2013-01-03   8   9  10  11
2013-01-04  12  13  14  15
```

- 根据标签 loc

```
print(df.loc['20130102'])
```

```
A    4
B    5
C    6
D    7
Name: 2013-01-02 00:00:00, dtype: int32
```

- 根据序列 iloc

```
print(df.iloc[[1,3,5],1:3])
```

```
             B   C
2013-01-02   5   6
2013-01-04  13  14
2013-01-06  21  22
```

- 通过判断的筛选

```
print(df[df.A>8])
```

```
             A   B   C   D
2013-01-04  12  13  14  15
2013-01-05  16  17  18  19
2013-01-06  20  21  22  23
```

（2）**处理缺省值**

- 直接去掉有 NaN 的行或列，可以使用 pd.dropna()

```
df.dropna(axis=0, how='any' )
```

	A	B	C	D
2013-01-01	0	1	2	3
2013-01-02	4	5	6	7
2013-01-03	8	9	10	11
2013-01-04	12	13	14	15
2013-01-05	16	17	18	19
2013-01-06	20	21	22	23

- 将 NaN 的值用其他值代替，可以使用 pd.fillna()
- 判断是否有缺失数据 NaN，可以使用 pd.isnull()

（3）导入导出文件

pandas 可以读取与存取的资料格式有很多种，像 csv、excel、json、html 与 pickle 等…， 详细请看官方说明文件

【例】新建一个 read.py 文件，读取 excel 数据文件

```
import pandas as pd #加载模块

#读取excel
data = pd.read_excel('student.xls')

#打印出data
print(data)
```

【例】新建一个 write.py 文件，写入 excel 数据文件

```
import pandas as pd #加载模块

df = pd.DataFrame(np.arange(24).reshape((6,4)),index=dates, columns=['A','B','C','D']

#写入excel
df.to_excel('test.xls')
```

5.3 Matplotlib

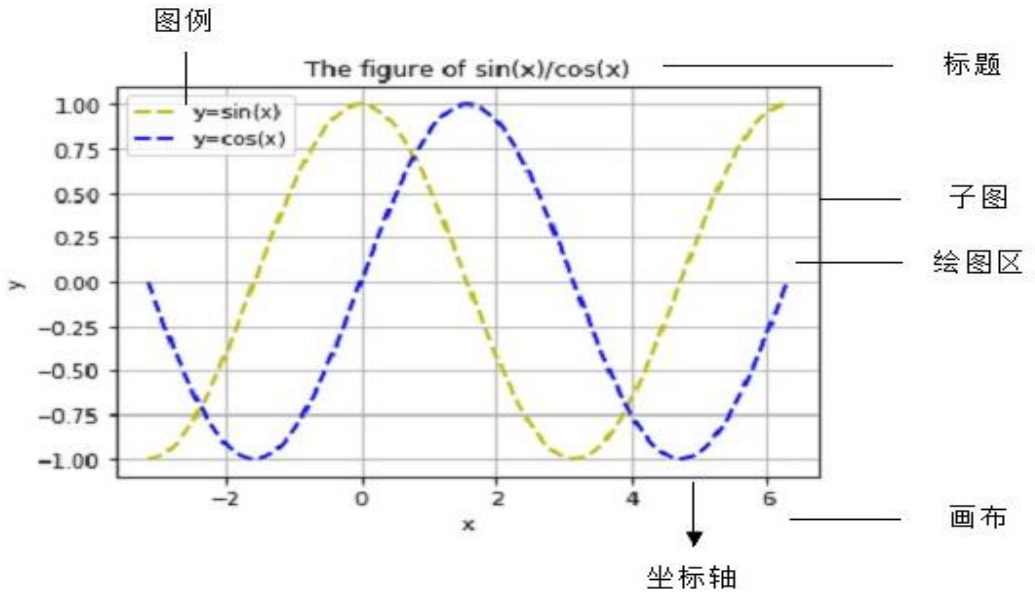
5.3.1 安装 Matplotlib 库

打开 cmd 命令行窗口执行：

```
pip install matplotlib
```

5.3.2 图表的基本结构

图表的结构一般包括：画布、图表标题、绘图区、x 轴（水平轴）和 y 轴（垂直轴）、图例等基本元素。



5.3.3 matplotlib.pyplot

Matplotlib 模块中比较常用的是 pyplot 子模块，内部包含了绘制图形所需要的功能函数。

pyplot 模块的常用函数表

函数	描述
figure	创建一个空白画布，可以指定画布的大小和像素
add_subplot	创建子图，可以指定子图的行数，列数和标号
subplots	建立一系列子图，返回fig,ax一个fig序列对象，建立一个axis序列
title	设置图表标题，可以指定标题的名称、颜色、字体等参数
xlabel	设置x轴名称，可以指定名称、颜色、字体等参数
ylabel	设置y轴名称，可以指定名称、颜色、字体等参数
xlim	指定x轴的刻度范围
ylim	指定y轴的刻度范围
legend	指定图例，及图例的大小、位置、标签
savefig	保存图形
show	显示图形

Matplotlib 的图像都位于 figure 对象中，用 plt.figure 创建一个新的画

布（空画布不能直接绘图）。如果不显式调用 figure()函数，也会默认创建一个画布供子图使用。

在画布上添加 plot 子图用 add_subplot 方法，add_subplot 函数的使用方法如下：

<子图对象>=<figure 对象>.add_subplot(nrows, ncols, index)

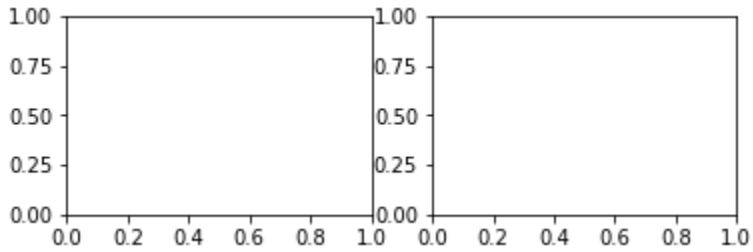
参数含义：

- nrows：子图划分成的行数
- ncols：子图划分成的列数
- index：当前子图的序号，编号从 1 开始

【例】绘制简单的 plot 图表，结果如图所示。

```
import matplotlib.pyplot as plt
fig=plt.figure()
ax1=fig.add_subplot(2,2,1)
ax2=fig.add_subplot(2,2,2)
```

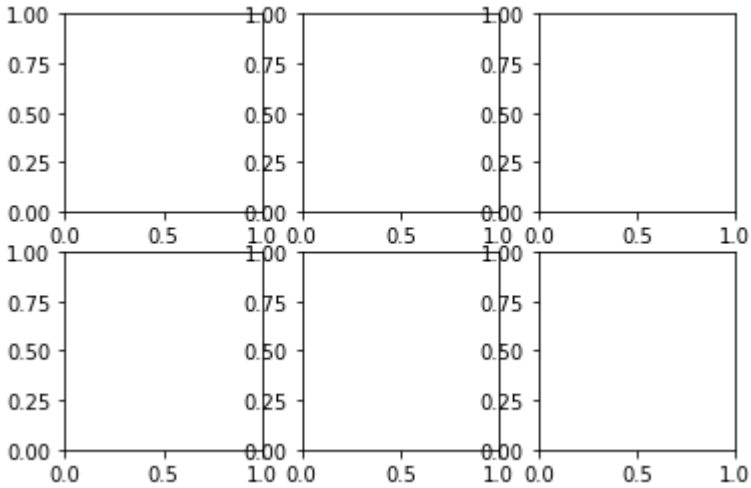
运行结果如下：



【例】六个 plot 的绘制，结果如图所示。

```
import matplotlib.pyplot as plt
fig,axes=plt.subplots(2,3)
axes
```

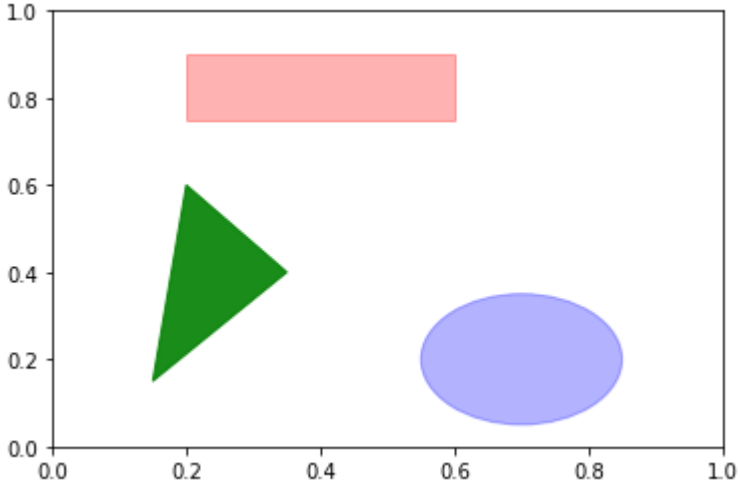
运行结果如下：



【例】在 Subplot 上绘制图形，结果如图所示。

```
import matplotlib.pyplot as plt
fig=plt.figure()
ax=fig.add_subplot(1,1,1)
rect=plt.Rectangle((0.2,0.75),0.4,0.15,color='r',alpha=0.3)
circ=plt.Circle((0.7,0.2),0.15,color='b',alpha=0.3)
pgon=plt.Polygon([[0.15,0.15],[0.35,0.4],[0.2,0.6]],color='g',alpha=0.9)
ax.add_patch(rect)
ax.add_patch(circ)
ax.add_patch(pgon)
plt.show()
```

运行结果如下：



5.3.4 plot 函数

绘制曲线可以使用 pyplot 中的 plot 函数。plot()的基本格式如下：

```
matplotlib.pyplot.plot(x,y,format_string,**kwargs)
```

参数：

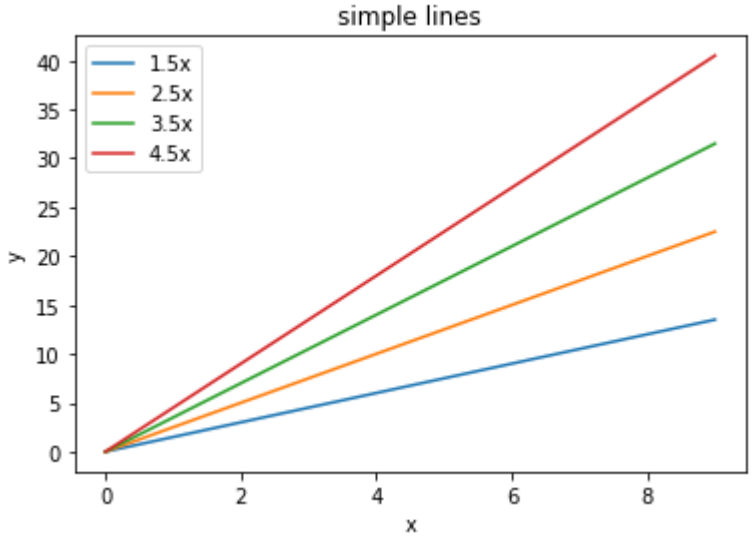
- x：x 轴数据，列表或数组，可选。
- y：y 轴数据，列表或数组。
- format_string：控制曲线的格式字符串，可选。
- **kwargs：第二组或更多组(x,y,format_string)参数。

注：当绘制多条曲线时，各条曲线的 x 不能省略。

【例】绘制简单直线，结果如图所示。

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
plt.xlabel('x')
plt.ylabel('y')
plt.plot(a,a*1.5,a,a*2.5,a,a*3.5,a,a*4.5)
plt.legend(['1.5x','2.5x','3.5x','4.5x'])
plt.title('simple lines')
plt.show()
```

运行结果如下：



5.3.5 其他类型的图表

在实际应用中，需要很多类型的图表。matplotlib.pyplot 提供了丰富的绘图函数可供选择，包括：scatter（散点图）、bar（条形图）、pie（饼图）、hist（直方图）以及的 plot（坐标图）。

(1) scatter()函数绘制散点图

matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None,

norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, *, data=None, **kwargs)

（2）hist()函数密度直方图：

matplotlib.pyplot.hist(x, bins=None, range=None, normed=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, hold=None, data=None, **kwargs)

（3）bar()绘制条形图：

matplotlib.pyplot.bar(left, height, width=0.8, bottom=None, hold=None, data=None, **kwargs)

（4）pie()绘制饼图：

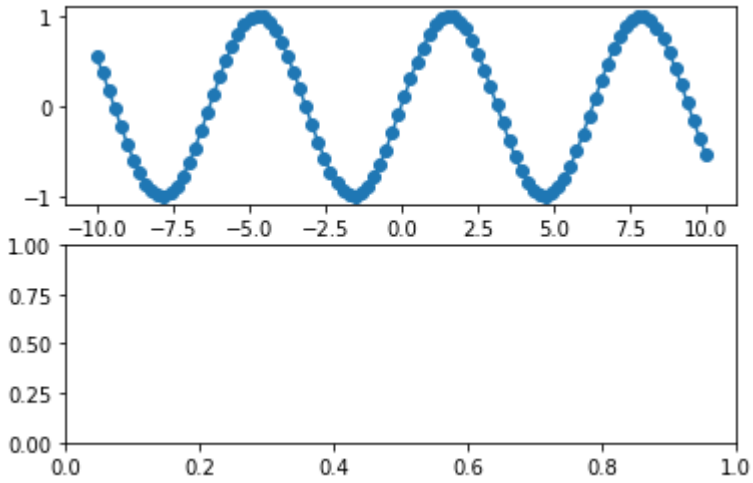
matplotlib.pyplot.pie(x, explode=None, labels=None, colors=None, autopct=None, pctdistance=0.6, shadow=False, labeldistance=1.1, startangle=None, radius=None, counterclock=True, wedgeprops=None, textprops=None, center=(0, 0), frame=False, hold=None, data=None)

【例】多个图表的绘制。

首先使用 subplots()函数确定要绘制图表的行、列数量，然后使用 subplot()方法指定当前绘图所使用的子图。

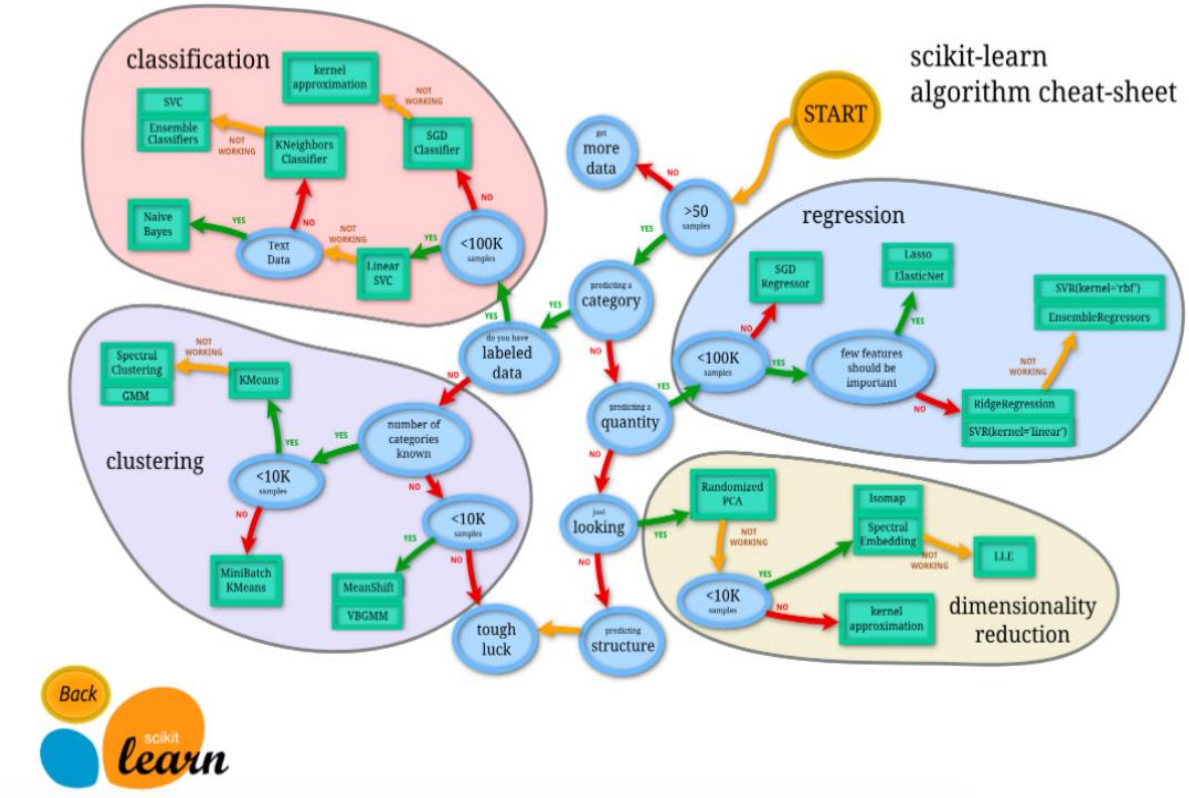
```
import numpy as np
import matplotlib.pyplot as plt
fig,axes=plt.subplots(2,1)
plt.subplot(2,1,1)
x = np.linspace(-10, 10, 100)  #列举出一百个数据点
y = np.sin(x)                  #计算出对应的y
plt.plot(x, y, marker="o")
```

运行结果如下：



5.4 Sklearn 简介

Scikit learn 的简称是 Sklearn，包含了许多最常见的机器学习算法，例如分类、回归、聚类、数据降维、数据预处理等。官方网站：http://scikit-learn.org



5.4.1 安装 Sklearn 库

打开 cmd 命令行窗口执行：

```
pip install scikit-learn
```

5.4.2 Sklearn 的一般步骤

- 获取数据，创建数据集
- 数据预处理
- 数据集拆分
- 定义模型
- 训练和预测模型

- 评估模型

（1）获取数据，创建数据集

Sklearn 提供了一个强大的数据库，包含了很多经典数据集。数据库网址为：http://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets 可以使用 Sklearn.datasets 这个数据库来获取数据。常用的数据集如下：

数据集	描述
datasets.fetch_california_housing	加载加利福尼亚住房数据集。
datasets.fetch_lfw_people	加载有标签的人脸数据集。
datasets.load_boston	加载波士顿房价数据集。
datasets.load_breast_cancer	加载乳腺癌威斯康星州数据集。
datasets.load_diabetes	加载糖尿病数据集。
datasets.load_iris	加载鸢尾花数据集。
datasets.load_wine	加载葡萄酒数据集。

使用比较著名的是鸢尾花数据集，调用如下：

```
from sklearn.datasets import load_iris
data = load_iris()
```

或者

```
from sklearn import datasets
boston = datasets. load_iris()
```

另一个经典的波士顿房价数据集，代码如下：

```
from sklearn.datasets import load_boston
boston = load_boston()
```

或者

```
from sklearn import datasets
boston = datasets.load_boston()
```

（2）数据预处理 （**注：本次实验任务暂未用到**）

Sklearn 中的 preprocessing 模块功能是数据预处理和数据标准化，能完成诸如数据标准化、正则化、二值化、编码以及数据缺失处理等。

函数名称	功能
preprocessing.Binarizer	根据阈值对数据进行二值化
preprocessing.Imputer	插值，用于填补缺失值。
preprocessing.LabelBinarizer	对标签进行二值化
preprocessing.MinMaxScaler	将数据对象中的每个数据缩放到指定范围。
preprocessing.Normalizer	将数据对象中的数据归一化为单位范数。
preprocessing.OneHotEncoder	使用one-Hot方案对整数特征编码。
preprocessing.StandardScaler	通过去除均值并缩放到单位方差来标准化。
preprocessing.normalize	将输入向量缩放为单位范数。
preprocessing.scale	沿某个轴标准化数据集。

【例】使用 Sklearn 的 preprocessing 模块对数据进行标准化处理。

```
from sklearn import preprocessing
import numpy as np
x=np.array([[3,-2,490],
            [3,0.5,520],
            [1,2,-443]])
x_scaled=preprocessing.scale(x)
print(x_scaled)
```

运行结果如下：

```
[[ 0.70710678 -1.31319831  0.67328879]
 [ 0.70710678  0.20203051  0.74039398]
 [-1.41421356  1.1111678  -1.41368277]]
```

【例】使用 preprocessing 的 MinMaxScaler 类，将数据缩放到固定区间 [0，1]。

```
from sklearn import preprocessing
import numpy as np
x=np.array([[3,-2,490],
            [3,0.5,520],
            [1,2,-443]])
min_max_scaler=preprocessing.MinMaxScaler()
x_minmax=min_max_scaler.fit_transform(x)
print(x_minmax)
```

运行结果如下：

```
[[1.         0.         0.96884735]
 [1.         0.625      1.        ]
 [0.         1.         0.        ]]
```

【例】使用 preprocessing 的 StandardScaler 标准化类。

```
from sklearn import preprocessing
import numpy as np
x=np.array([[3,-2,490],
            [3,0.5,520],
            [1,2,-443]])
scaler=preprocessing.StandardScaler().fit(x)
scaler.transform(x)
print(x)
```

运行结果如下：

```
[[ 3.00e+00 -2.00e+00  4.90e+02]
 [ 3.00e+00  5.00e-01  5.20e+02]
 [ 1.00e+00  2.00e+00 -4.43e+02]]
```

（3）数据集拆分

可以使用 Sklearn 提供的 `train_test_split` 方法，按照比例将数据集分为测试集和训练集，格式：

X_train, X_test, y_train, y_test = cross_validation.train_test_split(train_data, train_target, test_size=0.4, random_state=0)

参数解释：

- train_data：要划分的样本特征数据
- train_target：要划分的样本结果
- test_size：测试集占比，默认值为 0.3 即预留 30%测试样本。如果是整数的话就是测试集的样本数量。
- random_state：是随机数的种子。随机数种子的实质是该组随机数的编号。在需要重复试验的时候，使用同一编号能够得到同样一组随机数。比如随机数种子的值为 1、其他参数相同的情况下，每次得到的随机数是相同的。如果每次需要不一样的数据，则 random_state 设置为 None。
- 【例】将鸢尾花的数据集拆分成训练集和测试集。

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()
X_train,X_test,y_train,y_test = train_test_split(iris['data'],iris['target'],random_state=0)
print(X_train)
print(y_train)
```

运行结果如下：

```
 [6.3 2.9 5.6 1.8]
 [5.8 2.7 4.1 1. ]
 [7.7 3.8 6.7 2.2]
 [4.6 3.2 1.4 0.2]]
[1 1 2 0 2 0 0 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 0 2 1 1 1 1 2 0 0 2 1 0 0 1
 0 2 1 0 1 2 1 0 2 2 2 2 0 0 2 2 0 2 0 2 2 2 0 0 2 0 0 0 1 2 2 0 0 0 1 1 0 0
 1 0 2 1 2 1 0 2 0 2 0 0 2 0 2 1 1 1 2 2 1 1 0 1 2 2 0 1 1 1 1 0 0 0 2 1 2
 0]
```

（4）定义模型

针对不同的问题，选择合适的模型是非常重要的。如何确定学习模型，既涉及到模型的功能，还需要考虑不同数据量的情况。

```
from sklearn.linear_model import Perceptron
clf = Perceptron(fit_intercept=True, max_iter=10, shuffle=True, eta0=0.1, tol=None)  #定义感知机
```

（5）训练和预测模型

模型建立之后，需要使用数据集进行训练。通常用 SKlearn 中的 fit()函数实现；训练结束后，就可以使用模型对新的数据集进行预测，用 predict()函数实现。

```
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

（6）评估模型

sklearn.metrics 模块中提供了一些性能指标，包含评分函数、性能指标以及距离计算函数等，如下表：

常用 sklearn 分类评价指标

函数名	功能
metrics.f1_score()	计算调和均值F1指数
metrics.precision_score()	计算精确度
metrics.recall_score()	计算召回率
metrics.roc_auc_score()	根据预测分数计算接收机工作特性曲线下的计算区域（ROC/AUC）
metrics.precision_recall_fscore_support()	计算每个类的精确度，召回率，F1指数和支持
metrics.classification_report()	根据测试标签和预测标签，计算分类的精确度，召回率，F1指数和支持指标

常用 sklearn 回归评价指标

函数名	功能
metrics.mean_absolute_error()	平均绝对误差回归损失
metrics.mean_squared_error()	均方误差回归损失
metrics.r2_score()	R2回归分数函数

sklearn.model_selection 模块中提供了模型验证模型，如下表：

函数名	功能
model_selection.cross_validate()	通过交叉验证评估指标，并记录适合度/得分时间
model_selection.cross_val_score()	通过交叉验证评估分数
model_selection.learning_curve()	学习曲线
model_selection.validation_curve()	验证曲线