

## 机器学习应用实践（实验一）——线性回归（一）

### 一、实验目的

- 1、掌握机器学习算法的开发流程。
- 2、掌握 Scikit-Learn 官方网站的查看、学习和使用方法。
- 3、掌握 Scikit-Learn 中线性回归算法的使用，解决波士顿房价预测问题。

### 二、实验准备

- 1、配置软件环境。
  - （1）安装 Python 及开发环境。
  - （2）安装 NumPy、Pandas、Matplotlib 库。
  - （3）安装 Scikit-Learn、pytorch 库。
- 2、回顾线性回归算法的原理。
- 3、学习 Scikit-Learn 中线性回归算法涉及到的相关知识，详见“三、相关知识介绍”。

### 三、相关知识介绍

#### 1、Scikit-Learn 库介绍与内容查询

Scikit 又称 Scikit-learn 库（简称 sklearn）是一个通用型开源机器学习库，它几乎涵盖了所有机器学习算法，并且搭建了高效的数据挖掘框架。Scikit-learn 目前已成为机器学习领域最完整、最具影响力的算法库，该项目拥有较为充裕的资金支持、完整规范的运作流程、业内顶级的开发和维护团队，官网中相关内容介绍包括 sklearn 的安装和更新、算法核心原理的论文出处、算法使用方法和算法使用案例等，其详细度和完整度均在业内首屈一指，是非常优质的学习资源，如图 1 所示，官网网址：<https://scikit-learn.org/stable/index.html>。

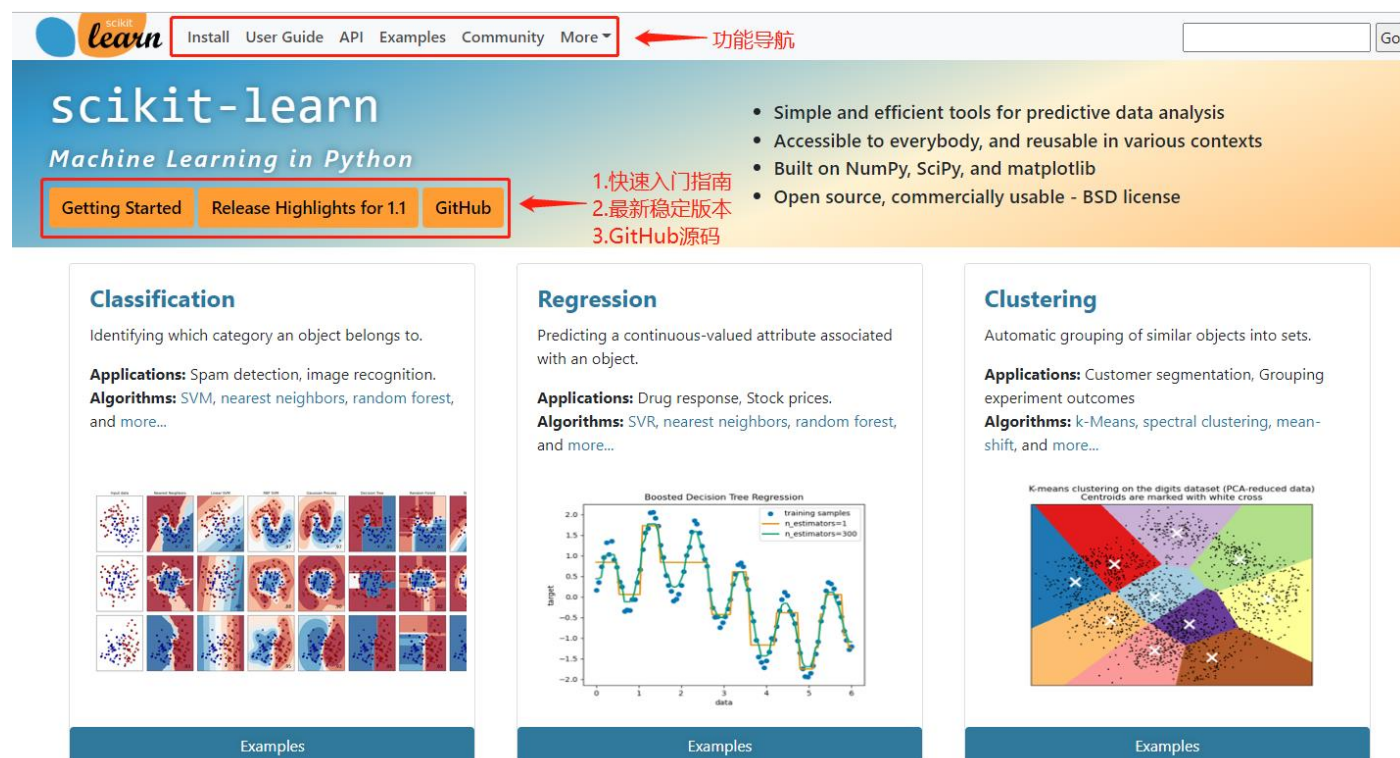


图 1 Scikit-learn 官网

**注意：**正是因为 Scikit-Learn 官网内容的完整性，国内也有许多团队试图将 Scikit-Learn 项目进行翻译、并建立相应的中文社区。但遗憾的是，由于国内的大多数团队对于开源项目的维护、管理和资金运作都缺乏必要的经验，导致诸多所谓的 Scikit-Learn 中文社区只是一堆过时的、不完整的、不准确的内容翻译拼凑而成的内容网站，所以充其量只能作为外文技术内容翻译的一个参考，而无法作为技术解释和技术学习的核心内容。因此，围绕 Scikit-Learn 的内容查阅，更推荐直接访问外文官网进行学习。

可以看到官网的宣传中主要提到 4 个特点：

- (1) 一个简单高效的数据挖掘和数据分析工具。
- (2) 对于所有人都是易用的，而且可以在各个环境中使用。
- (3) 它是基于 NumPy、Scipy 和 Matplotlib 的库。
- (4) 开源，可以商用。

Scikit 库主要分为以下 6 个板块，可以通过这六个板块进行内容查询：

- (1) 分类 (Classification)
- (2) 回归 (Regression)
- (3) 聚类 (Clustering)
- (4) 降维 (Dimensionality reduction)
- (5) 模型选择 (Model selection)
- (6) 预处理 (Preprocessing)

除了从六大板块进行内容查询之外，还可以通过 User Guide 进行查询，如图 2 所示。User Guide 包括 sklearn 所有内容的合集页面，更为细致地划分了有监督学习、无监督学习、模型选择评估、可视化、数据集下载、模型存储等内容。如果点击左上方的 Other versions，则可以下载 sklearn 所有版本的 User Guide 的 PDF 版本。

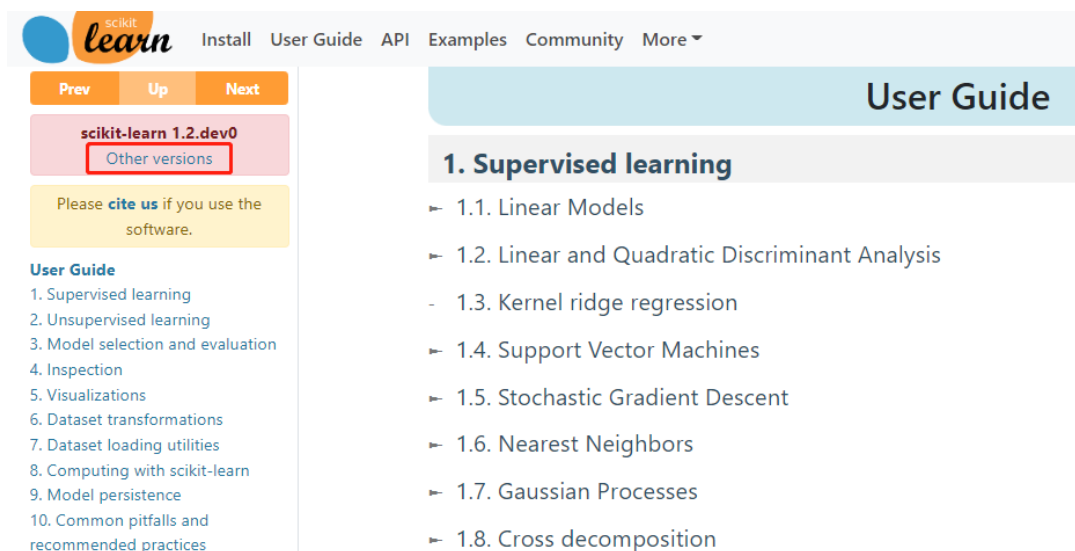


图 2 User Guide 页面

如果想根据评估器或实用函数的名字查找相关 API 说明文档，则可以点击最上方的 API 一栏进入，其中 API 查询文档根据模块首字母排序。

Examples 中按首字母顺序提供了 sklearn 中各模块功能使用的示例程序，同时提供了丰富的可视化展示方法，如图 3 所示。

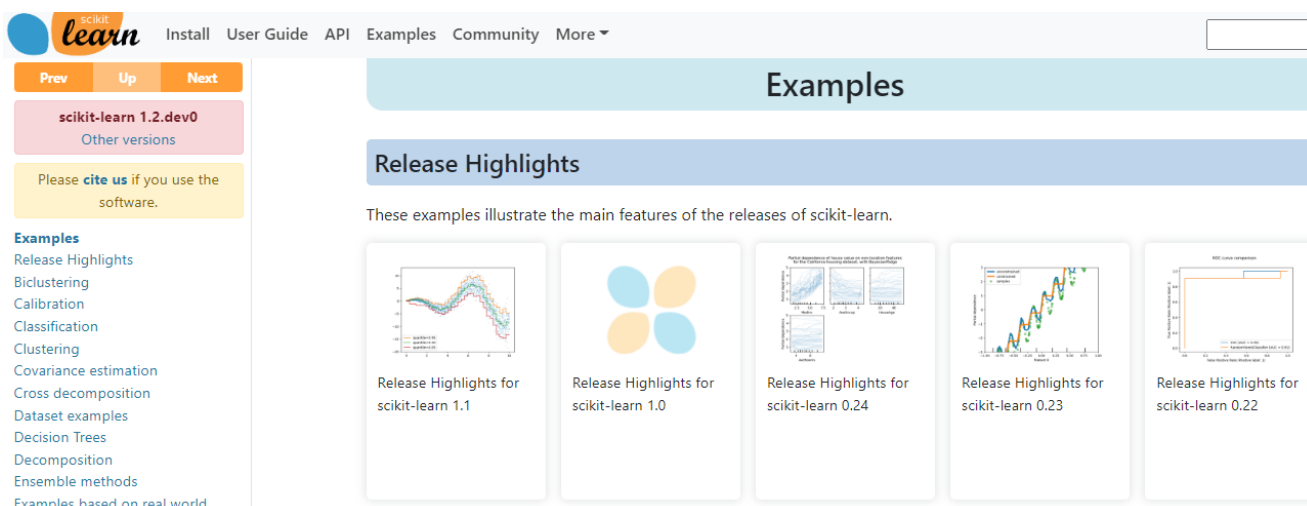


图 3 Examples 页面

其中分类和回归问题被称为有监督学习，聚类问题被称为无监督学习。机器学习的过程一般依次为预处理、降维、有监督和无监督学习、模型选择，如图 4 所示。

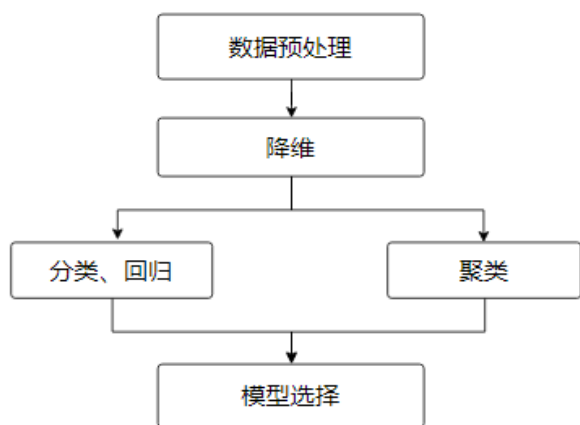


图 4 Scikit 库的机器学习过程

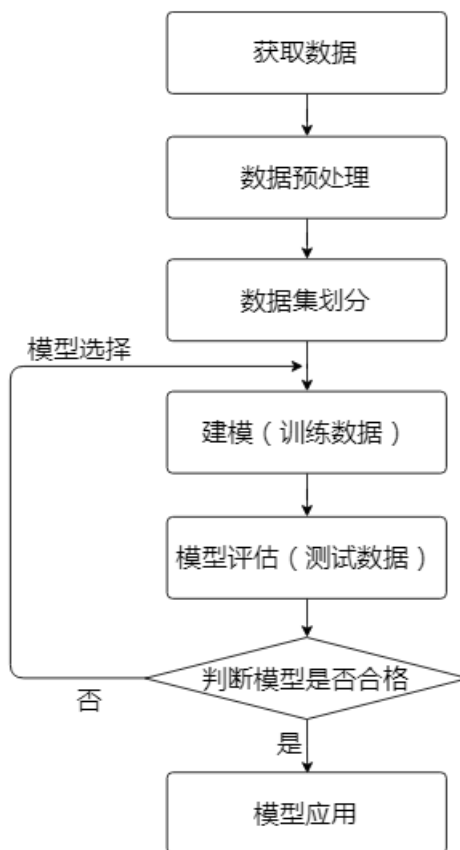


图 5 机器学习算法的开发详细流程

## 2、基于 sklearn 的机器学习算法的开发

机器学习算法的开发详细流程如图 5 所示，主要包括获取数据、数据预处理、数据集划分、建模、模型评估、模型选择和模型应用几个部分。下面针对线性回归问题按照机器学习算法的开发流程介绍 sklearn 中用到的相关 API 的使用。

### 2.1 获取数据集（sklearn 数据集 API 介绍）

sklearn 提供了非常多的内置数据集和一些创建数据集的方法。sklearn 中的数据集相关功能都在 datasets 模块下，可以通过 API 文档中的 datasets 模块进行概览，如图所示。Loaders 中提供了真实数据集，包括结构化数据集（如经典的鸢尾花数据集、波士顿房价数据集、乳腺癌数据集等），还有一些图片数据、文本数据等数据集；此外，Samples generator 中提供了许多能够创建不同数据分布的数据生成器（用 make\_\* 函数创建），可以用于创建测试评估器性能的数据。

## sklearn.datasets: Datasets

The `sklearn.datasets` module includes utilities to load datasets, including methods to load and fetch popular reference datasets. It also features some artificial data generators.

**User guide:** See the [Dataset loading utilities](#) section for further details.

### Loaders

<code>datasets.clear_data_home([data_home])</code>	Delete all the content of the data home cache.
<code>datasets.dump_svmlight_file(X, y, f, *[, ...])</code>	Dump the dataset in svmlight / libsvm file format.
<code>datasets.fetch_20newsgroups(*[, data_home, ...])</code>	Load the filenames and data from the 20 newsgroups dataset (classification).
<code>datasets.fetch_20newsgroups_vectorized(*[, ...])</code>	Load and vectorize the 20 newsgroups dataset (classification).
<code>datasets.fetch_california_housing(*[, ...])</code>	Load the California housing dataset (regression).
<code>datasets.fetch_covtype(*[, data_home, ...])</code>	Load the covtype dataset (classification).
<code>datasets.fetch_kddcup99(*[, subset, ...])</code>	Load the kddcup99 dataset (classification).

### Samples generator

<code>datasets.make_biclusters(shape, n_clusters, *)</code>	Generate a constant block diagonal structure array for biclustering.
<code>datasets.make_blobs([n_samples, n_features, ...])</code>	Generate isotropic Gaussian blobs for clustering.
<code>datasets.make_checkerboard(shape, n_clusters, *)</code>	Generate an array with block checkerboard structure for biclustering.
<code>datasets.make_circles([n_samples, shuffle, ...])</code>	Make a large circle containing a smaller circle in 2d.
<code>datasets.make_classification([n_samples, ...])</code>	Generate a random n-class classification problem.
<code>datasets.make_friedman1([n_samples, ...])</code>	Generate the "Friedman #1" regression problem.
<code>datasets.make_friedman2([n_samples, noise, ...])</code>	Generate the "Friedman #2" regression problem.
<code>datasets.make_friedman3([n_samples, noise, ...])</code>	Generate the "Friedman #3" regression problem.
<code>datasets.make_gaussian_quantiles(*[, mean, ...])</code>	Generate isotropic Gaussian and label samples by quantile.
<code>datasets.make_hastie_10_2([n_samples, ...])</code>	Generate data for binary classification used in Hastie et al. 2009, Example 10.2.
<code>datasets.make_low_rank_matrix([n_samples, ...])</code>	Generate a mostly low rank matrix with bell-shaped singular values.
<code>datasets.make_moons([n_samples, shuffle, ...])</code>	Make two interleaving half circles.

sklearn.datasets: 加载获取流行数据集。

(1) `datasets.load_*`(): 获取小规模数据集，数据包含在 datasets 里。

`sklearn.datasets.load_boston`(): 加载并返回波士顿房价数据集，样本数量 506，特征 13。

**注意:** 波士顿房价数据集在 sklearn 1.2 之后的版本中不再提供。如需使用如下代码从原始数据资源处下载：

```
import pandas as pd # doctest: +SKIP
import numpy as np

data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]
```

(2) `datasets.fetch_*`(data\_home=None): 获取大规模数据集，需要从网络上下载，函数的第一个参数是 data\_home，表示数据集下载的目录，默认是 ~/scikit\_learn\_data/。

(3) 数据集返回值介绍

load 和 fetch 返回的数据类型是 datasets.base.Bunch(字典格式)

●data: 特征数据数组，是 [n\_samples\*n\_features] 的二维 numpy.ndarray 数组。

- target**: 标签数组，是[n\_samples,]的一维 numpy.ndarray 数组。
- DESCR**: 数据描述。
- feature\_names**: 特征名。
- target\_names**: 标签名。

示例程序：

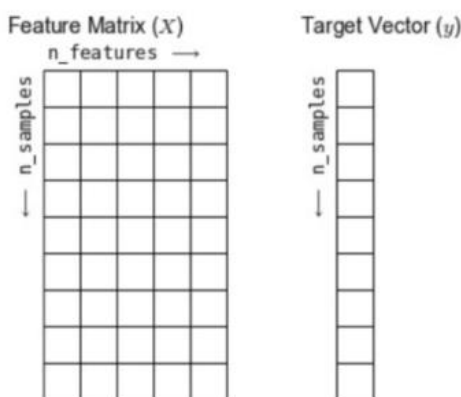
(1) 加州房价数据集：

```
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
```

(2) Ames 住房数据集：

```
from sklearn.datasets import fetch_openml
housing = fetch_openml(name="house_prices", as_frame=True)
```

在 sklearn 中，称特征矩阵为 Features Matrix( $X$ )，称标签数组为 Target Vector( $y$ )，并且以  $n\_samples$  表示数据行数、 $n\_features$  表示特征矩阵列数。



## 2.2 划分数据（分成训练集和数据集）

机器学习一般的数据集会划分为两个部分：

- 训练数据**：用于训练，构建模型。
- 测试数据**：在模型检验时使用，用于评估模型的泛化能力。

(1) 划分比例：

训练集：70%、80%、75%。

测试集：30%、20%、25%

(2) 数据集划分 API 介绍

数据集划分方法有很多，如留出法、交叉法、自助法等。这里我们先介绍 sklearn 提供的留出法，在 model\_selection 模块中提供了 train\_test\_split 方法，可以设定数据划分数据集比例，并且保持类别标签比例不变，进行数据集划分。

```
sklearn.model_selection.train_test_split(*arrays,*options)
```

**\*arrays:**

- X**: 数据集的特征值。
- y**: 数据集的标签值。

**注意**：若输入多个数组，数组的 shape[0] 要相同，否则报错。



**\*options:**

- test size: 测试集的大小，一般为 float。默认为 0.25。
- random state: 随机数种子，不同的种子会造成不同的随机采样结果。相同的种子采样结果相同。

- shuffle: 是否打乱顺序。在进行数据集划分之前是否打乱数据顺序。默认为 True。

- stratify: 分层采样。划分数据集时，是否与整个数据集中的标签比例一致。默认为 None。

**\*Returns (函数返回值):**

训练集特征值 X\_train，测试集特征值 X\_test，训练标签 y\_train，测试标签 y\_test。

示例程序：

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> list(y)
[0, 1, 2, 3, 4]
```

```
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]
```

```
>>> train_test_split(y, shuffle=False)
[[0, 1, 2], [3, 4]]
```

## 2.3 数据归一化

特征的单位或者大小相差较大，或者某特征的方差相比其他的特征要大出几个数量级，容易影响（支配）目标结果，使得一些算法无法学习到其它的特征。因此需要对数据进行归一化处理，去除数据的单位限制，将其转化为无量纲的纯数值，便于不同单位或量级的指标能够进行比较和加权。同时，归一化处理也是加速训练的方法之一，它可以使梯度下降算法更快速收敛和寻优。数据归一化的方法有很多，如最小最大标准化(Min-Max Normalization)、z-score 标准化、log 对数函数归一化、

反正切函数归一化、L2 范数归一化等等，因各自特点不同而适合于不同的机器学习算法，需要根据实际数据的特点和具体的机器学习方法进行选取。典型的数值型数据的无量纲化的方法有：最小最大标准化(Min-Max Normalization)和 z-score 标准化。

### (1) 最小最大标准化(Min-Max Normalization)

sklearn.preprocessing 提供了两种归一化算法：MinMaxScaler 和 MaxAbsScaler，分别对应两个类，前者可以将数据归一化到一个给定的范围 $[d_{\min}, d_{\max}]$ (默认为 $[0, 1]$ )，后者将数据的最大绝对值归一化到 1.0。使用 MinMaxScaler 算法时，首先要建立一个 sklearn.preprocessing.MinMaxScaler 类，语句如下：

```
>>> from sklearn.preprocessing import MinMaxScaler
>>> d_min = 2
>>> d_max = 3
>>> min_max_scaler = MinMaxScaler((dmin,dmax))
```

对应的算法效果可以看做两个步骤，首先将数值放缩到 $[0, 1]$ 。公式为：

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

该公式作用于 numpy.ndarray 数据矩阵的每一列， $x_{\max}$ 为该列的最大值， $x_{\min}$ 为该列的最小值。然后将归一化后的数据映射到给定范围 $[d_{\min}, d_{\max}]$ ，则最终结果 $x''$ 计算如下：

$$x'' = x' \times (d_{\max} - d_{\min}) + d_{\min}$$

MaxAbsScaler 算法的计算公式相对简单，可以写为：

$$x' = \frac{x}{x_{\text{MaxAbs}}}$$

其中 $x_{\text{MaxAbs}}$ 是当前列中的最大绝对值。

在建立好一个 sklearn.preprocessing.MinAbsScaler 类后，可以利用该类的内建函数 transform 或 fit\_transform 函数(对不同类型数据有更好的适应性)对具体的数据进行最小最大标准化操作，例如：

MinMaxScaler.fit\_transform(X)

- X: numpy array 格式的数据[n\_samples,n\_features]。
- 返回值：转换后的形状相同的 array。

示例程序：

1) 将数值放缩到 $[0, 1]$ ：

```
>>> X_train = np.array([[ 1., -1., 2.],
...                     [ 2., 0., 0.],
...                     [ 0., 1., -1.]])
...
>>> min_max_scaler = preprocessing.MinMaxScaler()
>>> X_train_minmax = min_max_scaler.fit_transform(X_train)
>>> X_train_minmax
array([[0.5, 0., 1.],
       [1., 0.5, 0.33333333],
       [0., 1., 0.]])
```

2) 数值放缩到 $[-1, 1]$ ：

```
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
...
>>> max_abs_scaler = preprocessing.MaxAbsScaler()
>>> X_train_maxabs = max_abs_scaler.fit_transform(X_train)
>>> X_train_maxabs
array([[ 0.5, -1. ,  1. ],
       [ 1. ,  0. ,  0. ],
       [ 0. ,  1. , -0.5]])
>>> X_test = np.array([[ -3., -1.,  4.]])
>>> X_test_maxabs = max_abs_scaler.transform(X_test)
>>> X_test_maxabs
array([[ -1.5, -1. ,  2. ]])
>>> max_abs_scaler.scale_
array([2.,  1.,  2.]])
```

(2) z-score 标准化：也叫标准差标准化，将列特征转化为标准正太分布 $\mathcal{N}(0,1)$ ，和整体样本分布相关，每个样本点都能对标准化产生影响。公式为：

$$x' = \frac{x - \mu}{\sigma}$$

公式作用于每一列， $\mu$ 为该列的平均值， $\sigma$ 为该列的标准差。

sklearn.preprocessing 提供的 z-score 标准化算法为 StandardScaler，利用以下语句实现：

sklearn.preprocessing.StandardScaler()

StandardScaler.fit\_transform(X)

●X: numpy array 格式的数据[n\_samples,n\_features]。

●返回值：转换后的形状相同的 array。

●属性：mean\_：训练集每个特征中数据的均值数组。

scaler\_：训练集每个特征中数据的标准差数组。

示例程序<sup>1</sup>：

```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> scaler
StandardScaler()

>>> scaler.mean_
array([1. ..., 0. ..., 0.33...])

>>> scaler.scale_
array([0.81..., 0.81..., 1.24...])

>>> X_scaled = scaler.transform(X_train)
>>> X_scaled
array([[ 0. ..., -1.22...,  1.33...],
       [ 1.22...,  0. ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

<sup>1</sup> 注释：示例程序中的第六行末位的.fit 函数可以令 scaler 适应于 X\_train 的数据类型，因此在数据转化时就可以直接使用 transform 函数，否则需要使用 fit\_transform 函数。



关于数据归一化，以下几点需要注意：

(1) 数据归一化不会影响数据集原始分布，也就是并不影响数据集真实规律。

(2) 如果在建模之前，对训练集特征进行了归一化处理（一般来说，无需对标签进行归一化），那么建模得到的模型参数是基于归一化数据得到的，在测试集上进行测试时，也需要对测试集进行归一化处理：使用训练集上得到的归一化参数进行训练集和测试集数据转换。以 z-score 标准化为例，测试集数据转换使用的  $\mu$  和  $\sigma$  均由训练集得到，而不是在训练集和测试集上分别计算  $\mu$  和  $\sigma$ 。

示例程序：

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train) # Don't cheat - fit only on training data
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test) # apply same transformation to test data
```

(3) 并非所有模型都受到特征数据之间差异影响，在通用的模型中，线性模型和距离类模型是两类典型的会受到特征量纲差异影响的模型，例如线性回归、KNN、K-Means 及使用 ECOC 编码的逻辑回归等，由于特征数据的大小会影响模型学习的偏重，模型会更加侧重于学习数值较大的特征，而忽视数值较小的特征中的有效信息，因此有时会出现较差的模型结果。但有些模型却不受此影响，典型的如树模型。

(4) 一旦对数据进行归一化处理，数据就失去了可解释性，也就是失去了量纲。例如对于鸢尾花数据来说，原始数据代表花瓣花萼的长宽测量结果，而如果我们对其进行归一化处理，则每条数据就无法再给予明确的现实意义，这也是在很多要求可解释性的情况下我们应该避免使用归一化方法的原因。

(5) 数据归一化能够加速梯度下降算法的收敛速度。归一化能够改变损失函数形态，使损失函数等高线图更加均匀，从而提高优化迭代的效率。

(6) sklearn 中的归一化，从功能上分为标准化 (Standardization) 和归一化 (Normalization) 两类。其中，此前所介绍的最小最大标准化和 Z-Score 标准化，都属于 Standardization 的范畴，而在 sklearn 中，Normalization 则特指针对单个样本（一行数据）利用其范数进行放缩的过程。不过二者都属于数据预处理范畴，都在 sklearn 中的 Preprocessing data 模块下。

## 2.4 线性回归模型

线性回归模型，对于  $\mathbf{x} \in \mathbb{R}^d$ ：

$$\hat{y}(\mathbf{w}, \mathbf{x}) = w_1 x_1 + w_2 x_2 + \cdots w_d x_d + w_0$$

代价函数：均方误差函数 (mean-square error, MSE)

$$J(\hat{\mathbf{w}}) = \frac{1}{N} \sum_{n=1}^N (y_n - \hat{\mathbf{w}} \hat{\mathbf{x}}_n)^2$$

其中， $N$  为数据样本总量， $\hat{\mathbf{w}}$  为方程系数所组成的向量， $\hat{\mathbf{x}}_n$  为第  $n$  个数据样本的特征向量， $\hat{\mathbf{w}} \hat{\mathbf{x}}_n$  为第  $n$  个样本的线性回归值(预测值)， $y_n$  为第  $n$  个样本的目标值(真实值)。

均方误差函数(MSE)具有如下优点：

- 等价于欧式距离的平方，易于理解；
- 为凸函数，不存在局部最小陷阱；

- 导数形式简单，有封闭解。

### (1) 线性回归的最小二乘求解

线性回归问题最基础的求解方法为最小二乘法(Ordinary Least Squares)，通过令代价函数的导数等于 0，可求得最优解（也可称封闭解或解析解）为  $\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$ ，也就是正则方程求解方法。显然，使用最小二乘法的条件是方阵  $\mathbf{X}^T \mathbf{X}$  是满秩的，行列式不为 0，即存在逆矩阵。sklearn 中提供的最小二乘法求解线性回归问题的类为：sklearn.linear\_model.LinearRegression()。

函数详情见：

[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LinearRegression.html#sklearn.linear\\_model.LinearRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression)

sklearn.linear\_model.LinearRegression 具有如下重要的超参数、属性和方法。

#### 1) 超参数（可在实例化时进行设置）

●fit\_intercept: 是否计算偏置。=True 则计算偏置，如果设置为 False，则在计算中将不使用截距（即数据应该进行中心化处理）。

●normalize: 是否对输入数据使用 L2 范式进行归一化处理。默认为 False。

●copy\_X: 建模时是否带入训练数据的副本。默认为 True。

●n\_jobs: 设置工作时并行计算的 CPU 核数。默认为 None，表示使用 1 个核。

#### 2) 重要属性

●coef\_: 线性回归问题的回归系数。

●intercept\_: 线性回归问题的偏置。当 fit\_intercept=False 时，intercept\_ 为 0。

#### 3) 重要方法

●fit(self, X, y[, sample\_weight]): 训练机器学习模型（评估器 estimator），是每个评估器都具有的方法。根据训练集数据 X 和 y，拟合线性模型，获得线性模型的最优参数。训练好的模型参数可通过属性 coef\_ 和 intercept\_ 进行查看。

●predict(self, X): 根据测试集数据 X，返回测试集 X 中样本的类别标签。

●score(self, X, y[, sample\_weight]): 根据测试集数据 X 和 y，返回预测值与真实值评价分数（评价指标根据分类、回归、聚类模型不同而不同）。

**说明：**sklearn 中对于回归问题默认使用决定系数  $R^2$  作为评价指标，而非均方误差 MSE。关于决定系数  $R^2$  的更多内容，详见本节“2.5 回归模型的常用评价指标”。

**注意：**所有接口中要求输入 X\_train 和 X\_test 的部分，输入的特征矩阵必须至少是一个二维 array，sklearn 不接受任何一维 array 作为特征矩阵被输入。如果你的数据的确只有一个特征，那必须的使用 reshape(-1,1)来给矩阵增维；如果你的数据只有一个特征和一个样本，可以使用 reshape(1,-1)给数据增维。

示例程序：

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> #  $y = 1 * x_0 + 2 * x_1 + 3$ 
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.score(X, y)
1.0
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

## (2) 线性回归的梯度下降求解

更为普遍的，线性回归问题可以使用梯度下降算法求解其数值解。sklearn 中提供了随机梯度下降优化算法 `sklearn.linear_model.SGDRegressor`。随机梯度下降算法在每次更新中根据一个样本计算损失函数的梯度，其优点是运行速度快，适用于大批量数据训练。`SGDRegressor` 可以支持不同的 loss 函数和正则化惩罚项来拟合线性回归模型。

```
sklearn.linear_model.SGDRegressor(loss="squared_loss", fit_intercept=True, learning_rate='invscaling', eta0=0.01)
```

### ●loss: 损失类型

loss='squared loss': 普通最小二乘法

fit intercept: 是否计算偏置 0

### ●learning\_rate: string, optional 学习率填充

'constant':  $\eta = \eta_0$

'optimal':  $\eta = 1.0 / (\alpha * (t + t_0))$  (默认)

'invscaling':  $\eta = \eta_0 / \text{pow}(t, \text{power\_t})$  Power\_t=0.25: 存在父类当中

对于一个常数值的学习率来说，可以使用 `learning_rate='constant'` 并使用 `eta0` 来指定学习率。

### ●SGDRegressor.coef\_: 回归系数

### ●SGDRegressor.intercept\_: 偏置

函数详情见:

[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.SGDRegressor.html?highlight=sgdregressor#sklearn.linear\\_model.SGDRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html?highlight=sgdregressor#sklearn.linear_model.SGDRegressor)

**注意:** 梯度下降算法对于特征缩放非常敏感，因此需要对数据进行归一化处理。

## (3) sklearn 中 Pipeline 机器学习流的使用

sklearn 提供了一种 Pipeline 工具能够串联多个评估器，组成一个机器学习流，从而简化模型在重复调用时的代码量。因此，数据标准化和模型训练的过程，更加通用的实现过程是采用 Pipeline 工具实现。

示例程序:

```
>>> import numpy as np
>>> from sklearn.linear_model import SGDRegressor
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.preprocessing import StandardScaler
>>> n_samples, n_features = 10, 5
>>> rng = np.random.RandomState(0)
>>> y = rng.randn(n_samples)
>>> X = rng.randn(n_samples, n_features)
>>> # Always scale the input. The most convenient way is to use a pipeline.
>>> reg = make_pipeline(StandardScaler(),
...                      SGDRegressor(max_iter=1000, tol=1e-3))
>>> reg.fit(X, y)
Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('sgdregressor', SGDRegressor())])
```

例程中使用 `make_pipeline` 创建了一个包含了缩放器(标准化)和回归器的数据挖掘流程对象 `reg`。使用 `fit` 方法训练，`pipeline` 执行的步骤是先进行标准化 `StandardScaler`，再进行 `SGDRegressor` 训练模型，`fit` 后的模型参数存在 `reg` 对象中。

`reg` 同样具有 `predict`、`score` 方法可以进行预测和性能评分。在使用 `predict` 方法时，会对新样本进行数据标准化和模型预测二个步骤，从而得出最终的预测结果。`score` 方法同理。

## 2.5 回归模型的常用评价指标

回归模型有如下一些评价指标，如表 1 所示。平均绝对误差、均方误差和中值绝对误差越接近 0，模型性能越好。可解释方差值和 R 方差越靠近 1，模型性能越好。

表 1 sklearn 中回归模型的评价指标

方法名称	最优值	sklearn 函数
平均绝对误差	0.0	<code>metrics.mean_absolute_error</code>
均方误差	0.0	<code>metrics.mean_squared_error</code>
中值绝对误差	0.0	<code>metrics.median_absolute_error</code>
可解释方差值	1.0	<code>metrics.explained_variance_score</code>
R 方值	1.0	<code>metrics.r2_score</code>

(1) 均方误差(Mean Squared Error, MSE)公式为:

$$MSE = \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2$$

其中， $y_i$  为真实值， $\hat{y}_i$  为预测值， $m$  为样本个数。

sklearn 中提供的评价指标在 `metrics` 类中，其中 MSE 方法为 `metrics.mean_squared_error`:

`sklearn.metrics.mean_squared_error(y_true,y_pred)`

- `y_true`: 真实值  $y_i$
- `y_pred`: 预测值  $\hat{y}_i$
- `return`: 浮点数结果

(2) 决定系数 (R-square 或  $R^2$ )

对于线性回归模型来说，除了 MSE（从数值层面评价模型的精确程度）以外，还可使用决定系数（R-square 或  $R^2$ ，也被称为拟合优度检验）作为其模型评估指标，它可以从统计学的角度，整体评估线性回归模型对数据集的拟合程度。决定系数的公式如下：

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

其中 SSE 是误差平方和，SSR 是回归平方和，SST 是平方和的总和， $SST = SSR + SSE$ 。

$$SSE = \sum_{i=1}^m (y_i - \hat{y}_i)^2$$

$$SSR = \sum_{i=1}^m (\bar{y} - \hat{y}_i)^2$$

$$SST = \sum_{i=1}^m (\bar{y} - y_i)^2$$

决定系数是一个[0,1]之间的值，决定系数的值越高，即越接近 1，模型拟合数据集的效果越好。

## 2.6 模型保存

当模型构建完毕之后，可以对训练好的模型进行保存，在预测时只要读取模型，直接进行预测即可，不用每次都重新训练。可以使用 Python 内置的 pickle 库中的 dump 函数进行模型保存，使用 load 函数进行模型读取。

示例程序：

```
>>> from sklearn import svm
>>> from sklearn import datasets
>>> clf = svm.SVC()
>>> X, y = datasets.load_iris(return_X_y=True)
>>> clf.fit(X, y)
SVC()

>>> import pickle
>>> s = pickle.dumps(clf)
>>> clf2 = pickle.loads(s)
>>> clf2.predict(X[0:1])
array([0])
>>> y[0]
0
```

更为普遍地，可以使用 Python 中的 joblib 库，它对 numpy 大型数组进行了特定优化，具有更快的处理速度。joblib 只能将模型保存在磁盘中，而不是字符串中，其保存的默认地址是程序文件所在的根目录。

示例程序：

```
>>> from joblib import dump, load
>>> dump(clf, 'filename.joblib')

>>> clf = load('filename.joblib')
```



## 四、实验内容

1、题目一：采用 `scikit-learn` 中的 `LinearRegression`(最小二乘)线性回归模型对波士顿房价数据集进行预测，分别使用正则方程和随机梯度下降方法建模。

具体内容：

(1) 导入数据

a) 查看数据集的描述、特征名、标签名、数据样本量等信息。

b) 获取样本的特征数据和标签数据。

(2) 划分数据（分成训练集和测试集）

(3) 数据归一化

(4) 训练模型

a) 使用 `sklearn` 中线性回归的正规方程（`LinearRegression`）优化方法建模。

b) 使用 `sklearn` 中线性回归的随机梯度下降（`SGDRegressor`）优化方法建模。

(5) 模型评估（2 个模型）

评价指标： $MSE$  和  $R^2$  值。

【讨论一】梯度下降和正规方程两种算法有何不同？

分析梯度下降和正规方程两种算法的差异（计算时间、评价指标对比）与优缺点。

提示：`python` 中计时器-`timeit.default_timer()`方法。

【讨论二】数据归一化对算法有什么影响？

对比使用数据归一化和不使用数据归一化，正规方程和梯度下降算法性能是否有差异？分析原因。

【讨论三】梯度下降算法中的学习率如何影响其工作？

尝试修改随机梯度下降算法(`SGDRegressor`)的学习率（`eta0`），观察参数对模型性能的影响（通过数据分析即可），试分析学习率与模型性能之间的关系。

【讨论四】（选做）模型的泛化能力如何？

(1) 分别计算模型在训练样本上性能和在测试样本上的性能，判断模型是过拟合还是欠拟合？

(2) 数据集划分不同对模型性能是否有影响？可尝试修改方法(`train_test_split`)中的 `test_size` 参数，观察数据集划分对模型性能的影响。

(3) 尝试使用其他线性回归模型。线性回归模型中除了 `LinearRegression`，还有 `Ridge`(岭回归)、`Lasso`、`Polynomial regression`（多项式回归）等模型，自学官网资料，使用不同模型进行建模，观察不同模型训练后的模型权重差异，试分析模型的使用场合。

## 机器学习应用实践（实验二）——线性回归（二）

### 一、实验目的

- 1、熟悉线性回归算法的原理。
- 2、熟练掌握 NumPy、Pandas、Matplotlib 库的使用。
- 3、具备使用 python 实现线性回归算法的编程能力。

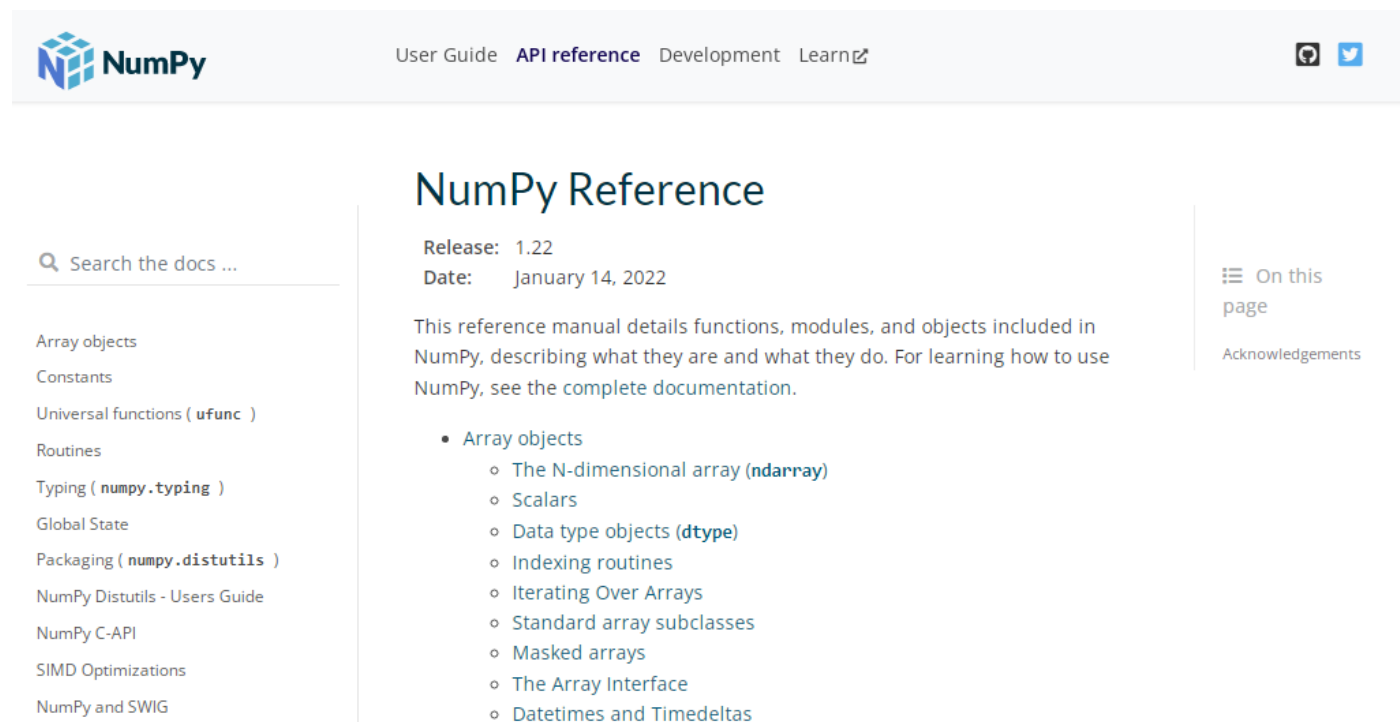
### 二、实验准备

- 1、回顾 NumPy、Pandas、matplotlib 库的基本用法。
- 2、回顾线性回归算法，最小二乘目标函数的梯度下降优化算法的实施流程。

### 三、相关知识介绍

#### 1、NumPy 库

NumPy 库是一个开源的 Python 科学计算库。可以很自然地使用数组和矩阵，也包含很多实用的数学函数。官方网站：<https://numpy.org/doc/stable/reference/index.html>。



The screenshot displays the NumPy Reference documentation page. At the top, there is a navigation bar with links for 'User Guide', 'API reference', 'Development', and 'Learn'. Below this, the 'NumPy Reference' title is prominently displayed. The page includes release information: 'Release: 1.22' and 'Date: January 14, 2022'. A brief description states: 'This reference manual details functions, modules, and objects included in NumPy, describing what they are and what they do. For learning how to use NumPy, see the complete documentation.' A sidebar on the left contains a search bar and a list of topics: 'Array objects', 'Constants', 'Universal functions (ufunc)', 'Routines', 'Typing (numpy.typing)', 'Global State', 'Packaging (numpy.distutils)', 'NumPy Distutils - Users Guide', 'NumPy C-API', 'SIMD Optimizations', and 'NumPy and SWIG'. On the right, there are links for 'On this page' and 'Acknowledgements'. The main content area lists topics under 'Array objects': 'The N-dimensional array (ndarray)', 'Scalars', 'Data type objects (dtype)', 'Indexing routines', 'Iterating Over Arrays', 'Standard array subclasses', 'Masked arrays', 'The Array Interface', and 'Datetimes and Timedeltas'.


对于数值计算任务，使用 NumPy 要比直接编写 Python 代码便捷得多，具有如下特点：

- （1）NumPy 能够直接对数组和矩阵进行操作，可以省略很多循环语句。
- （2）NumPy 众多的数学函数能简化编写代码的工作。

需掌握 NumPy 库的 ndarray 数组对象的创建、运算、切片与索引（详细资料见“NumPy 基础.PPT”），以及 NumPy 通用函数：一元通用函数、二元通用函数、数组及函数、矩阵运算函数、随机数生成函数等的使用（详细资料见“NumPy 通用函数.PPT”）。

#### 2、Pandas 库

Pandas 库是一个快速、强大、灵活、易用的开源数据分析和操作工具，官方网站：<https://pandas.pydata.org/>。



About us ▾ Getting started Documentation Community ▾ Contribute

# pandas

pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

[Install pandas now!](#)

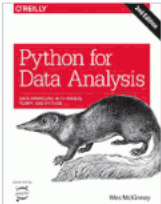
### Latest version: 1.4.2

- What's new in 1.4.2
- Release date: Apr 02, 2022
- Documentation (web)
- Documentation (pdf)
- Download source code

### Follow us

Follow @pandas\_dev

### Get the book



### Previous versions

- 1.4.1 (Feb 12, 2022)

### Getting started

- Install pandas
- Getting started







### Documentation

- User guide
- API reference
- Contributing to pandas
- Release notes

### Community

- About pandas
- Ask a question
- Ecosystem

With the support of:




具有如下特点：

- (1) Pandas 以 NumPy 为基础，能利用 NumPy 在计算方面性能高的优势。
- (2) Pandas 提供了大量处理数据的函数和方法，强大而高效。
- (3) 两种新型的结构 Series 和 DataFrame 使 Pandas 在处理表格数据非常有效。

需掌握 Pandas 库 Series 对象的创建和切片，及常用属性和函数(详细资料见“Pandas 基础.PPT”)，以及 DataFrame 对象的数据合并、转换、分组和聚合等的使用(详细资料见“Pandas 进阶.PPT”)。

### 3、matplotlib 库

matplotlib 是 Python 中最常用的可视化工具之一，功能非常强大，可以方便地绘制折线图、条形图、柱形图、散点图、盒图等 2D 图形，还可以绘制基本的 3D 图形。matplotlib 是 Python 数据可视化的基础库，在它的基础上又衍生出了多个数据可视化的工具集。基本使用教程详见资料“matplotlib 数据可视化.PPT”。官方网站：<https://matplotlib.org/stable/tutorials/introductory/usage.html>。



Plot types Examples Tutorials Reference User guide Develop Release notes

### Basic Usage

This tutorial covers some basic usage patterns and best practices to help you get started with Matplotlib.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
```

### A simple example

Matplotlib graphs your data on **Figures** (e.g., windows, Jupyter widgets, etc.), each of which can contain one or more **Axes**, an area where points can be specified in terms of x-y coordinates (or theta-r in a polar plot, x-y-z in a 3D plot, etc). The simplest way of creating a Figure with an Axes is using `pyplot.subplots`. We can then use `Axes.plot` to draw some data on the Axes:

4、线性回归问题相关原理及梯度下降算法简介（参考文献：<https://arxiv.org/pdf/1609.04747.pdf>）

表 2 线性回归问题的三种梯度下降算法

<p>线性模型：<math>\hat{y}(\mathbf{w}, \mathbf{x}) = w_1x_1 + w_2x_2 + \cdots w_dx_d + w_0</math></p> <p>即<math>f_w(\mathbf{x}) = \sum_{j=0}^M w_jx_j</math>, M 为样本的特征/属性的数量+1</p>		
<p>线性回归的代价函数：<math>J(\hat{\mathbf{w}}) = \frac{1}{2N} \sum_{i=1}^N (y_i - \hat{\mathbf{w}}\hat{\mathbf{x}}_i)^2</math></p> <p>其中，N 为训练集的样本量</p>		
梯度下降法 (Batch Gradient Descent, BGD)	随机梯度下降法 (Stochastic Gradient Descent, SGD)	小批量梯度下降法 (Mini Batch Gradient Descent, MBGD)
<p>BGD 算法思想如下：每次迭代遍历数据集时，保存每组训练数据对应的梯度。遍历结束后，计算所有数据集的梯度平均值，最后调整所有模型参数。每次的梯度为：</p> $\frac{\partial J(\hat{\mathbf{w}})}{\partial w_j} = \frac{1}{N} \sum_{i=0}^N (f_w(x^i) - y^i)x_j^i$ <p>显然，不断优化迭代之后，BGD 算法能收敛于最优解（对于非凸问题，也可能是局部最优解）。但对于大样本集收敛速度慢。</p>	<p>SGD 算法思想如下：在一次迭代中，依次遍历数据集中的每组数据，利用每组数据对应的梯度来调整模型参数。即，对于一个含有 m 组数据的数据集，在每次迭代训练中，必须调整模型参数 m 次。每次的梯度为：</p> $\frac{\partial J(\hat{\mathbf{w}})}{\partial w_j} = (f_w(x^i) - y^i)x_j^i$ <p>显然，SGD 算法相比 BGD 具有更快的收敛速度，但是每次迭代只用一组数据进行参数调整，迭代过程会非常“杂乱”。</p>	<p>MBGD 算法是 BGD 算法和 SGD 算法的折中。它的思想如下：首先将训练集随机打乱，并划分为若干均等小样本；然后每次迭代后遍历每个小样本，计算小批量样本的梯度平均值，根据计算的平均值调整模型参数。若有：N = Data Size, b = Mini Batch Size, 则每次迭代训练中，需调整模型参数(N/b)次。每次的梯度为：</p> $\frac{\partial J(\hat{\mathbf{w}})}{\partial w_j} = \frac{1}{b} \sum_{i=0}^b (f_w(x^i) - y^i)x_j^i$ <p>当小批量样本规模足够大时，小批量样本梯度向量的平均值在误差允许范围内近似等于全体训练样本梯度增量的平均值。MBGD 算法相比 BGD 算法加快了收敛速度，相比于 SGD 算法降低了样本的杂乱程度。</p>
<p>更新规则：<math>w_{j+1} := w_j - \alpha \frac{\partial J(\hat{\mathbf{w}})}{\partial w_j}</math>, 其中<math>\alpha</math>为学习率</p>		
<p>评价指标：<math>MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{\mathbf{w}}\hat{\mathbf{x}}_i)^2</math></p>		

(1) 梯度下降法 (BGD) 伪代码:

```
for i in range(nb_epochs):
    params_grad = evaluate_gradient(loss_function, data, params)
    params = params - learning_rate * params_grad
```

(2) 随机梯度下降法 (SGD) 伪代码:

```
for i in range(nb_epochs):
    np.random.shuffle(data)
    for example in data:
        params_grad = evaluate_gradient(loss_function, example, params)
        params = params - learning_rate * params_grad
```

(3) 小批量梯度下降法 (MBGD) 伪代码:

```
for i in range(nb_epochs):
    np.random.shuffle(data)
    for batch in get_batches(data, batch_size=50):
        params_grad = evaluate_gradient(loss_function, batch, params)
        params = params - learning_rate * params_grad
```

#### 四、实验内容

1、题目一：采用梯度下降法 (BGD) 优化线性回归模型，对波士顿房价进行预测。

(1) 导入数据 (从.csv 文件中导入数据代码如下)

```
#导入数据集
datafile = 'D:\\python-3.8\\Lib\\site-packages\\sklearn\\datasets\\data\\boston_house_prices.csv'
with open(datafile, encoding='utf-8') as f:
    data = np.loadtxt(f, delimiter=',', skiprows=2)
```

(2) 划分数据 (分成训练集和数据集)

(3) 数据归一化

(4) 训练模型 `model(train_x,train_y)`

(a) 初始化参数  $w$ 。

可使用 `np.concatenate` 数组拼接函数,将截距与权重参数合并在一起(也可以不拼接合并)。

(b) 求  $f(x)$ 。

(c) 求  $J(w)$ 。

参考代码:

```
m=len(X)
loss=np.sum((np.dot(X,w)-y)**2)/(2*m)
```

(d) 求梯度。

(e) 更新参数  $w$ 。

参考代码:

```
h=np.dot(X,w)
diff=h-y
for j,theta in enumerate(w):
    w[j]= theta - learning_rate /X.shape[0] * np.sum(diff * X[:,j])
```

(b-e) 的过程经过 `epochs` 次迭代。

(5)画出损失函数虽迭代次数的变化曲线。(通过损失函数变化曲线来观察梯度下降执行情况)



(6) 测试集数据进行预测，模型评估。

(7) 可视化：展示数据拟合的效果。

2、题目二（选做）：小批量梯度下降算法（MBGD）的编程实现。

## 五、实验报告要求

每个题目请按照以下说明进行书写。

(1) 开发环境：jupyter notebook 或其他开发环境，请根据自身情况如实填写。

(2) 总结算法编写流程和库函数使用经验。

(3) 程序代码、运行结果：请截图，保证图片清晰能够反映实际运行结果（如果程序较长，可分段进行程序和结果展示）。

(4) 拓展探究：对函数参数不同设置的研究、不同算法的尝试等。

(5) 调试改进：针对每个题目，有选择性的列举在实验过程中遇到的典型问题及描述、原因分析排查以及解决方式说明等。

### 注意事项：

1、实验报告严禁相互抄袭，如发现实验报告雷同，则雷同的实验报告小项评分最高为及格。

2、每次实验需提交：实验报告按“学号+姓名”形式命名。