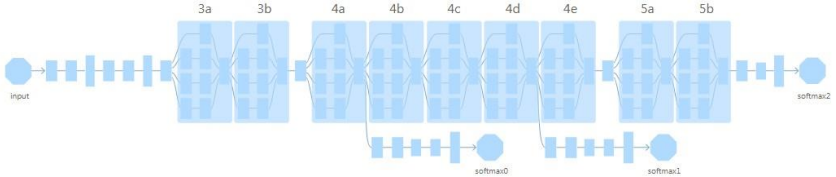
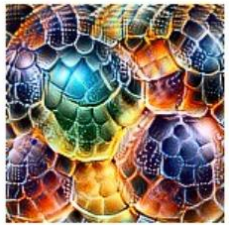
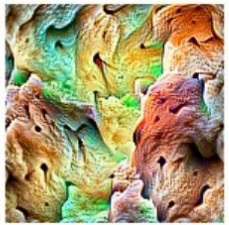
Google TensorFlow™

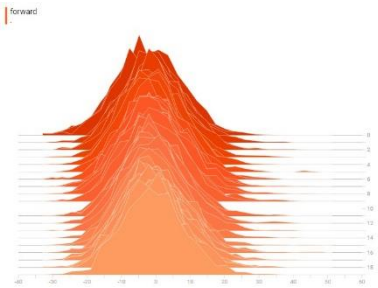
浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

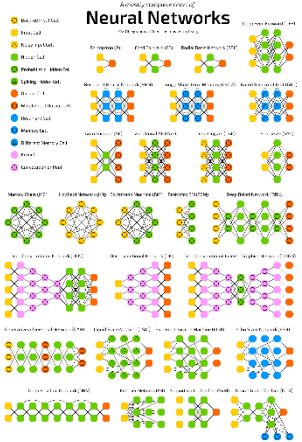


深度学习应用开发
基于TensorFlow的实践

吴明晖 李卓蓉 金苍宏
浙江大学城市学院
计算机与计算科学学院
Dept. of Computer Science
Zhejiang University City College



波士顿房价预测

多元线性回归问题TensorFlow2.0实践

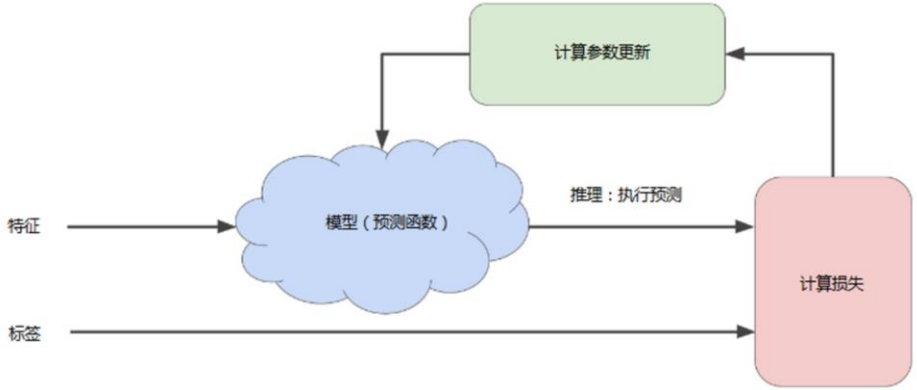
波士顿房价预测

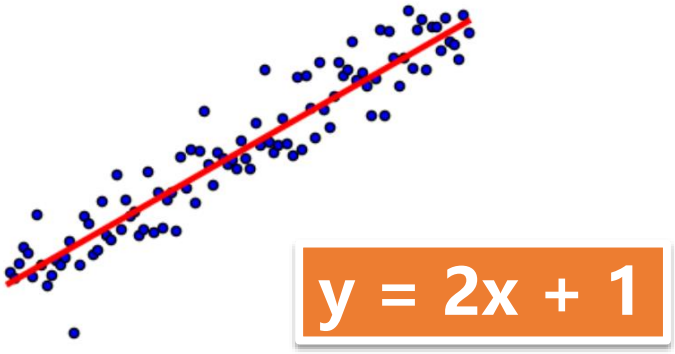
波士顿房价数据集包括**506**个样本，每个样本包括**12个特征变量**和该地区的**平均房价**

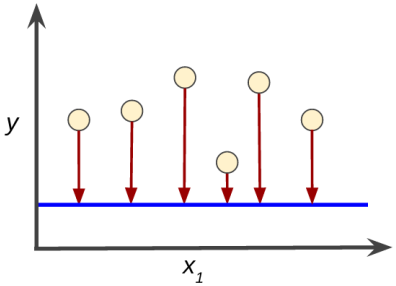
房价（单价）显然和多个特征变量相关，不是单变量线性回归（**一元线性回归**）问题

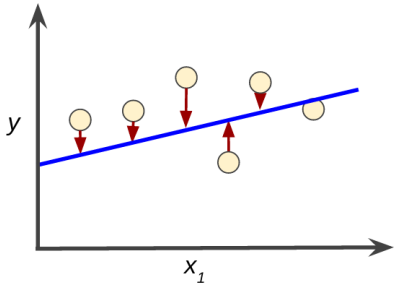
选择多个特征变量来建立线性方程，这就是多变量线性回归（**多元线性回归**）问题

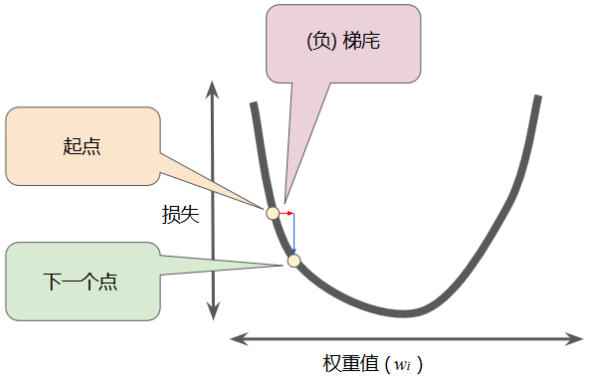
前情回顾：一元线性回归


计算参数更新
特征
模型（预测函数）
推理：执行预测
计算损失
标签


y = 2x + 1


y
x_1


y
x_1


(负) 梯度
起点
损失
下一个点
权重值（w_i）

前情回顾：机器学习的步骤

使用Tensorflow进行算法设计与训练的核心步骤

- （1）准备数据
- （2）构建模型
- （3）训练模型
- （4）进行预测

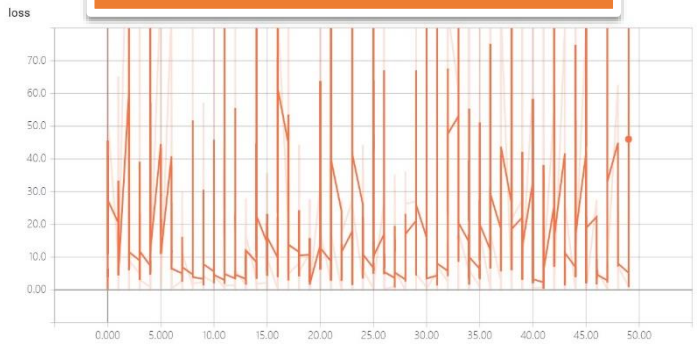
上述步骤是我们使用Tensorflow进行算法设计与训练的核心步骤，贯穿于后面介绍的具体实战中。本章用一个简单的例子来讲解这几个步骤。

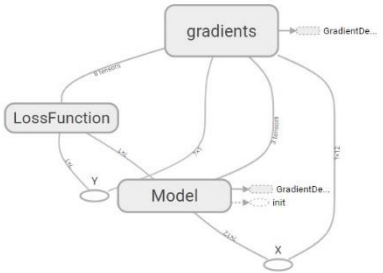
房价预测问题：多元线性回归及TensorFlow编程进阶

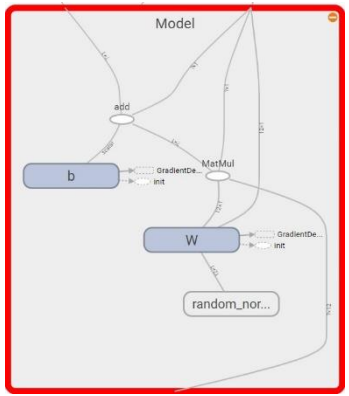
	CRIM	ZN	INDUS	CHAS	NOX	RM	\
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	
75%	3.677082	12.500000	18.100000	0.000000	0.624000	6.623500	
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	

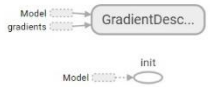
	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	\
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	
mean	68.574901	3.795043	9.549407	408.237154	18.455534	12.653063	
std	28.148861	2.105710	8.707259	168.537116	2.164946	7.141062	
min	2.900000	1.129600	1.000000	187.000000	12.600000	1.730000	
25%	45.025000	2.100175	4.000000	279.000000	17.400000	6.950000	
50%	77.5					11.360000	
75%	94.0					16.955000	
max	100.0					37.970000	

Boston房价预测










```
epoch= 1 loss= 44.3441744805 b= 3.60815 w= [[-0.60289431]
 [ 1.38135946]
 [-0.78309298]
 [ 0.49668884]
 [ 2.51221323]
 [ 7.16771173]
 [-0.05658912]
 [ 0.80084318]
 [ 0.38297549]
 [ 0.33776706]
 [ 2.31645942]
 [-4.39285994]]
epoch= 2 loss= 32.0449512219 b= 3.99625 w= [[ -1.15317142]
 [  1.96271288]
 [ -1.51236773]
 [  0.85116291]
 [  2.88758063]
 [ 10.60607815]
 [ -0.82200134]
 [  0.35972106]
 [  0.63090378]
 [ -0.2541407 ]
 [  1.15306044]
 [ -8.10907555]]
epoch= 3 loss= 27
```

```
x_test = x_test.reshape(1,12)
predict = sess.run(pred, feed_dict={x: x_test})
print("预测值：%f" % predict)

target = y_data[n]
print("标签值：%f" % target)
```

预测值：23.972641
标签值：24.500000

数据读取

数据集解读

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	MEDV
2	0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	4.98	24
3	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6
4	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7
5	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4
6	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	36.2
7	0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	5.21	28.7

CRIM: 城镇人均犯罪率

ZN：住宅用地超过 25000 sq.ft. 的比例

INDUS: 城镇非零售商用土地的比例

CHAS: 边界是河流为1，否则0

NOX: 一氧化氮浓度

RM: 住宅平均房间数

AGE: 1940年之前建成的自用房屋比例

DIS：到波士顿5个中心区域的加权距离

RAD: 辐射性公路的靠近指数

TAX: 每10000美元的全值财产税率

PTRATIO: 城镇师生比例

LSTAT: 人口中地位低下者的比例

MEDV: 自住房的平均房价，单位：千美元

读取数据

```
import tensorflow as tf    # 导入Tensorflow
import numpy as np      # 导入numpy
import matplotlib.pyplot as plt # 导入matplotlib

# 在Jupyter中，使用matplotlib显示图像需要设置为 inline 模式，否则不会在网页里显示图像
%matplotlib inline

import pandas as pd    # 导入pandas
from sklearn.utils import shuffle    # 导入sklearn的shuffle
from sklearn.preprocessing import scale    # 导入sklearn的scale

print("Tensorflow版本是：",tf.__version__) #显示当前TensorFlow版本
```

```
Tensorflow版本是： 2.0.0
```

读取数据

通过Pandas导入数据

```
df = pd.read_csv("data/boston.csv", header=0)
```

数据探索

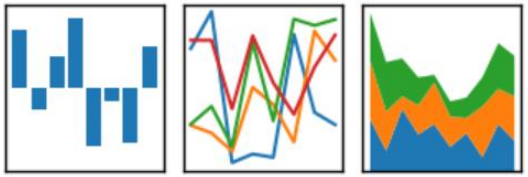
```
print (df.describe())
```

```
              CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677082   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO       LSTAT  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534   12.653063
std     28.148861    2.105710    8.707259  168.537116    2.164946    7.141062
min      2.900000    1.129600    1.000000  187.000000   12.600000    1.730000
25%     45.025000    2.100175    4.000000  279.000000   17.400000    6.950000
50%     77.500000    3.207450    5.000000  330.000000   19.050000   11.360000
75%     94.075000    5.188425   24.000000  666.000000   20.200000   16.955000
max    100.000000   12.126500   24.000000  711.000000   22.000000   37.970000

             MEDV
count  506.000000
mean    22.532806
std      9.197104
min      5.000000
25%     17.025000
50%     21.200000
75%     25.000000
max     50.000000
```

通过pandas读取数据文件，列出统计概述

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

想快速读取常规大小的数据文件时，通过创建读缓存区和其他的机制可能会造成额外的开销。此时建议采用Pandas库来处理。

Pandas官网（http://pandas.pydata.org)这样介绍Pandas：

“Pandas是一款开源的、基于BSD协议的Python库，能够提供高性能、易用的数据结构和数据分析工具。”他具有以下特点：

- 能够从CSV文件、文本文件、MS Excel、SQL数据库，甚至是用于科学用途的HDF5格式
- CSV文件加载能够自动识别列头，支持列的直接寻址
- 数据结构自动转换为Numpy的多维数组

Pandas读取数据

```
df.head(3)  # 显示前3条数据
```

]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7

```
df.tail(3)  # 显示后3条数据
```

]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	MEDV
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	7.88	11.9

数据集的划分

为什么要做数据集划分？

构建和训练机器学习模型是希望对新的数据做出良好预测

如何去保证训练的实效，可以应对以前未见过的数据呢？

全部带标签的数据参与模型训练，真的好吗？

数据集划分

构建和训练机器学习模型是希望对新的数据做出良好预测

如何去保证训练的实效，可以应对以前未见过的数据呢？

一种方法是将数据集分成两个子集：

训练集 – 用于训练模型的子集

测试集 – 用于测试模型的子集

通常，在测试集上表现是否良好是衡量能否在新数据上表现良好的有用指标，前提是：

测试集足够大

不会反复使用相同的测试集来作假

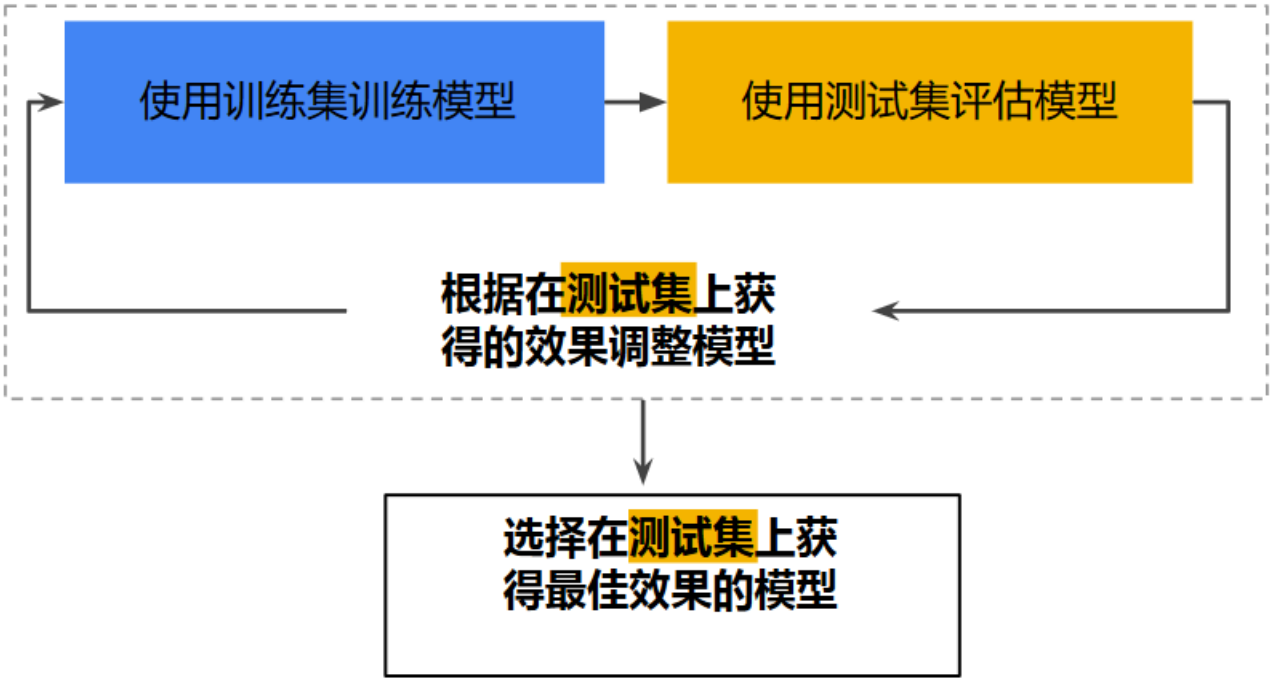
拆分数据

将单个数据集拆分为一个**训练集**和一个**测试集**

确保测试集满足以下两个条件：

- **规模足够大**，可产生具有统计意义的结果
- **能代表整个数据集**，测试集的特征应该与训练集的特征相同

工作流程


使用训练集训练模型
使用测试集评估模型
根据在测试集上获得的效果调整模型
选择在测试集上获得最佳效果的模型

有没有什么问题？

思考：

使用测试集和训练集来推动模型开发**迭代**的流程。

在每次迭代时，都会对训练数据进行训练并评估测试数据，并以基于测试数据的评估结果为指导来选择和更改各种**模型超参数**，例如学习速率和特征。这种方法是否存在问题？

多次重复执行该流程可能导致模型不知不觉地拟合了特定测试集的特性

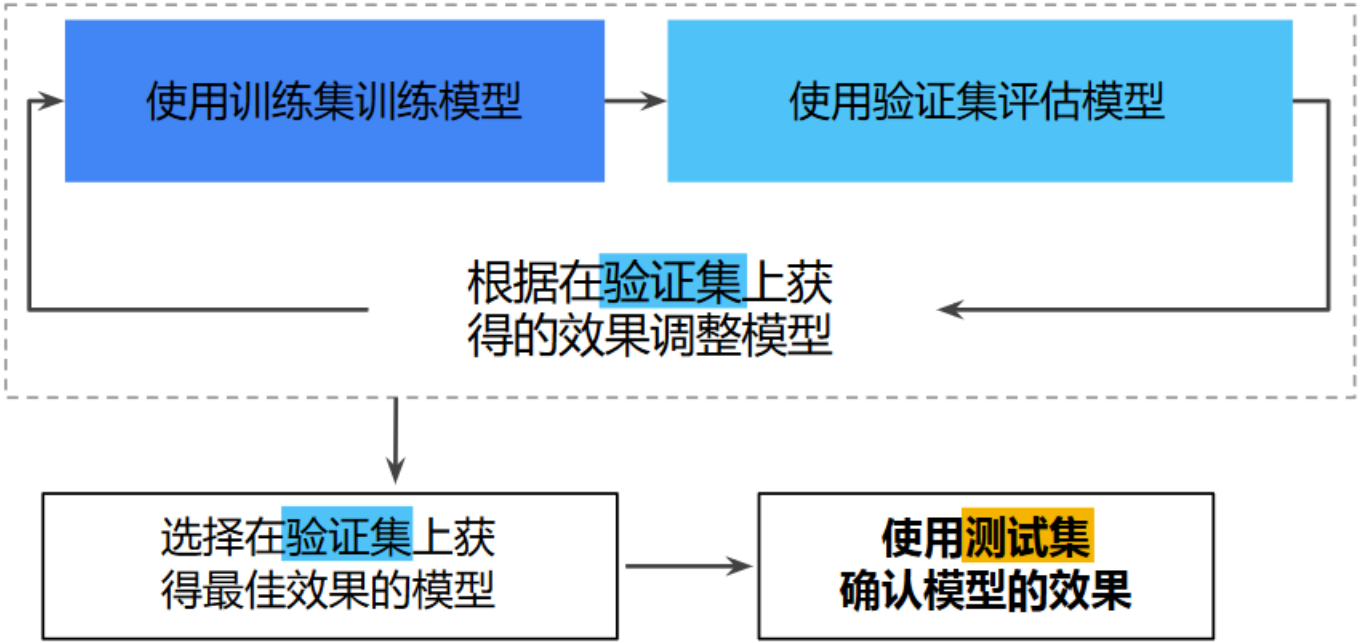
新的数据划分

通过将数据集划分为三个子集，可以大幅降低过拟合的发生几率：

使用**验证集**评估训练集的效果。

在模型“通过”验证集之后，使用测试集再次检查评估结果

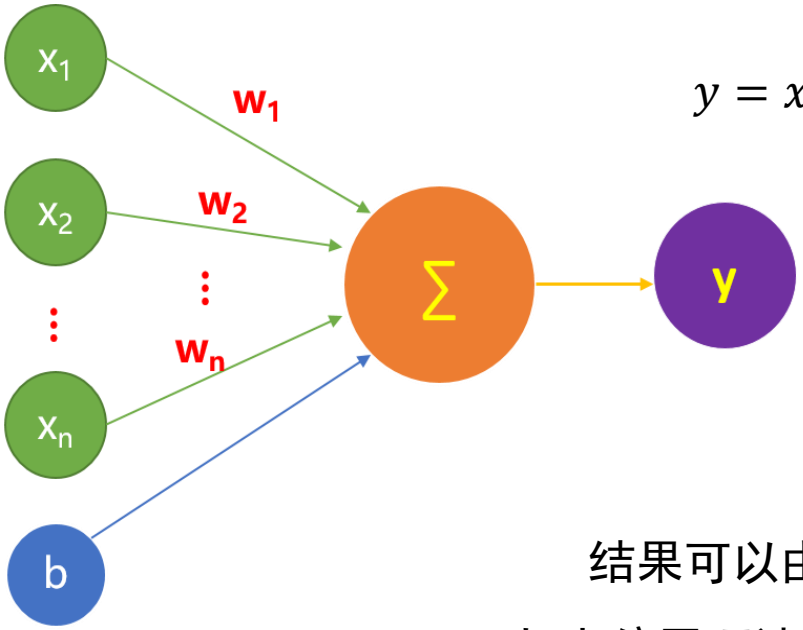
新的工作流程


使用训练集训练模型
使用验证集评估模型
根据在验证集上获得的效果调整模型
选择在验证集上获得最佳效果的模型
使用测试集确认模型的效果

准备建模

多元线性回归模型

房价和多个特征变量相关，本讲尝试使用多元线性回归建模

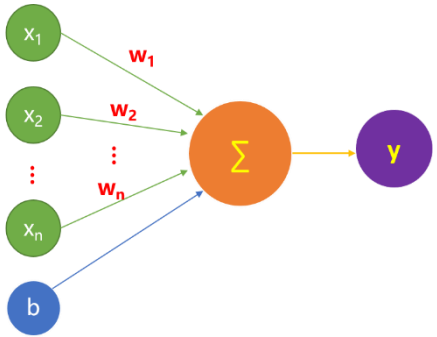


$$y = x_1 * \omega_1 + x_2 * \omega_2 + \cdots + x_n * \omega_n + b$$

结果可以由不同特征的输入值和对应的权重相乘求和，加上偏置项计算求解

多变量线性方程的矩阵运算表示

$$y = x_1 * \omega_1 + x_2 * \omega_2 + \cdots + x_n * \omega_n + b$$



$$y = \sum_{i=0}^{n} x_i * \omega_i + b$$

$$(x_1 * \omega_1 + x_2 * \omega_2 + \cdots + x_n * \omega_n) = [x_1 \quad x_2 \quad \cdots \quad x_n] * \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix}$$

$$y = [x_1 \quad x_2 \quad \cdots \quad x_n] * \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix} + b$$

矩阵运算是机器学习的基本手段，必须掌握！

机器学习中的基本线性代数

线性代数的数学对象

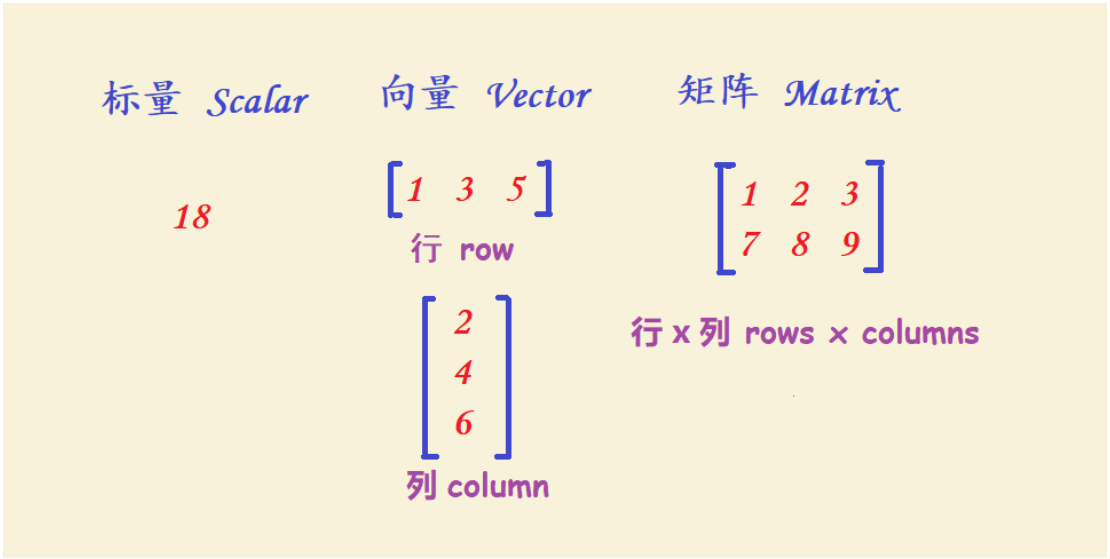
数学对象

标量
标量只是一个单一的数字

向量
向量是一个有序的数字数组
可以在一行或一列中

矩阵
矩阵是一个有序的二维数组，它有两个索引。第一个指向该行，第二个指向该列

向量也是一个矩阵，但只有一行或一列



线性代数基本计算规则

1 矩阵标量运算

如果矩阵乘，除，或者加、减一个标量，即：对矩阵的每一个元素进行数学运算

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} * 2 = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{bmatrix}$$

$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \end{bmatrix} * n = \begin{bmatrix} a11*n & a12*n & a13*n \\ a21*n & a22*n & a23*n \end{bmatrix}$$

线性代数基本计算规则

2 矩阵-矩阵加法和减法

矩阵-矩阵加法和减法要求是矩阵具有相同的尺寸，并且结果将是具有相同尺寸的矩阵。只需在第一个矩阵中添加或减去第二个矩阵的每个值及其对应的值

$$\begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix} + \begin{bmatrix} b11 & b12 \\ b21 & b22 \end{bmatrix} = \begin{bmatrix} a11+b11 & a12+b12 \\ a21+b21 & a22+b22 \end{bmatrix}$$

线性代数基本计算规则

3 矩阵-矩阵点乘（点积）

矩阵-矩阵点乘要求是矩阵具有相同的尺寸，矩阵各个对应元素相乘

$$\begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix} \bullet \begin{bmatrix} b11 & b12 \\ b21 & b22 \end{bmatrix} = \begin{bmatrix} a11 \bullet b11 & a12 \bullet b12 \\ a21 \bullet b21 & a22 \bullet b22 \end{bmatrix}$$

线性代数基本计算规则

4 矩阵矩阵相乘（叉乘）

如果第一个矩阵列的数量与第二个矩阵行数要相等，才能将矩阵相乘
结果矩阵具有与第一个矩阵相同的行数和与第二个矩阵相同的列数

$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \end{bmatrix} * \begin{bmatrix} b11 & b12 \\ b21 & b22 \\ b31 & b33 \end{bmatrix} = \begin{bmatrix} a11*b11+a12*b21+a13*b31 & a11*b12+a12*b22+a13*b33 \\ a21*b11+a22*b21+a12*b31 & a21*b12+a22*b22+a23*b33 \end{bmatrix}$$

A	B	A*B=C
(2, 3)	(3, 2)	(2, 2)
m * n	n * k	m * k

线性代数基本计算规则

5 矩阵-向量乘法

看作矩阵-矩阵叉乘的特列

$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \end{bmatrix} * \begin{bmatrix} b11 \\ b21 \\ b31 \end{bmatrix} = \begin{bmatrix} a11*b11+a12*b21+a13*b31 \\ a21*b11+a22*b21+a12*b31 \end{bmatrix}$$

A	B	A*B=C
(2, 3)	(3, 1)	(2, 1)
m * n	n * 1	m * 1

线性代数基本计算规则

6 向量-向量乘法（列向量-行向量）

看作矩阵-矩阵叉乘的特列

$$\begin{bmatrix} b11 \\ b21 \\ b31 \end{bmatrix} * \begin{bmatrix} a11 & a12 \end{bmatrix} = \begin{bmatrix} b11*a11 & b11*a12 \\ b21*a11 & b21*a12 \\ b31*a11 & b31*a12 \end{bmatrix}$$

B　　A　　B*A=C

(3, 1)　　(1, 2)　　(3, 2)

n * 1　　1 * m　　n * m

线性代数基本计算规则

7 向量-向量乘法（行向量-列向量）

看作矩阵-矩阵叉乘的特列中的特例

$$\begin{bmatrix} a11 & a12 & a13 \end{bmatrix} * \begin{bmatrix} b11 \\ b21 \\ b31 \end{bmatrix} = \begin{bmatrix} a11*b11+a12*b21+a13*b31 \end{bmatrix}$$

A	B	A*B=C
(1, 3)	(3, 1)	(1, 1)
1 * n	n * 1	1 * 1

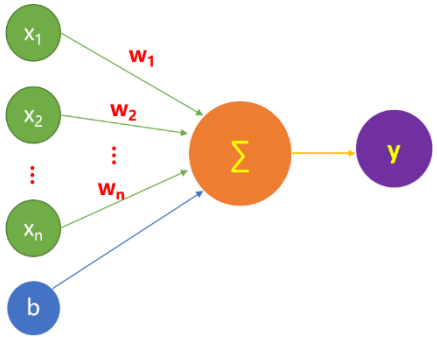
多变量线性方程的矩阵运算表示

$$y = x_1 * \omega_1 + x_2 * \omega_2 + \cdots + x_n * \omega_n + b$$

$$y = \sum_{i=0}^{n} x_i * \omega_i + b$$

$$(x_1 * \omega_1 + x_2 * \omega_2 + \cdots + x_n * \omega_n) = [x_1 \quad x_2 \quad \cdots \quad x_n] * \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix}$$

$$y = [x_1 \quad x_2 \quad \cdots \quad x_n] * \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix} + b$$



线性代数基本计算规则

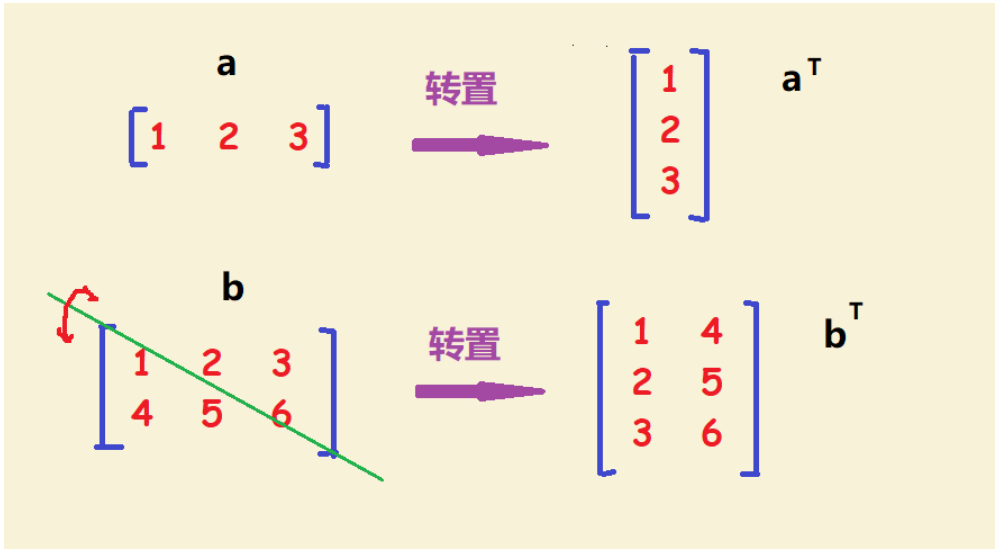
8 矩阵转置

第一列变成转置矩阵的第一行，第二列变成了矩阵转置的第二行

一个m * n矩阵被转换成一个n * m矩阵

a的a_{ij}元素等于转置矩阵a^T的a_{ji}元素

转置矩阵像沿着45度轴线的矩阵镜像



数据准备

读取数据

通过Pandas导入数据

```
df = pd.read_csv("data/boston.csv", header=0)
```

数据探索

```
print (df.describe())
```

```
              CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677082   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO       LSTAT  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534   12.653063
std     28.148861    2.105710    8.707259  168.537116    2.164946    7.141062
min      2.900000    1.129600    1.000000  187.000000   12.600000    1.730000
25%     45.025000    2.100175    4.000000  279.000000   17.400000    6.950000
50%     77.500000    3.207450    5.000000  330.000000   19.050000   11.360000
75%     94.075000    5.188425   24.000000  666.000000   20.200000   16.955000
max    100.000000   12.126500   24.000000  711.000000   22.000000   37.970000

             MEDV
count  506.000000
mean    22.532806
std      9.197104
min      5.000000
25%     17.025000
50%     21.200000
75%     25.000000
max     50.000000
```

通过pandas读取数据文件，列出统计概述

数据准备

```
# 获取数据集的值
ds = df.values     # df.values以np.array形式返回数据集的值
```

```
print(ds.shape)     # 查看数据的形状
```

```
(506, 13)
```

```
print(ds)     # 查看数据集的值
```

```
[[6.3200e-03 1.8000e+01 2.3100e+00 ... 1.5300e+01 4.9800e+00 2.4000e+01]
 [2.7310e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 9.1400e+00 2.1600e+01]
 [2.7290e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 4.0300e+00 3.4700e+01]
 ...
 [6.0760e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 5.6400e+00 2.3900e+01]
 [1.0959e-01 0.0000e+00 1.1930e+01 ... 2.1000e+01 6.4800e+00 2.2000e+01]
 [4.7410e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 7.8800e+00 1.1900e+01]]
```

划分特征数据和标签数据

```
# x_data 为 归一化后的前12列特征数据
x_data = ds[:,:12]

# y_data 为最后1列标签数据
y_data = ds[:,12]
```

```
print('x_data shape=', x_data.shape)
print('y_data shape=',y_data.shape)
```

```
x_data shape= (506, 12)
y_data shape= (506,)
```

划分训练集、验证集和测试集


```
train_num = 300          #训练集的数目
valid_num = 100          #验证集的数目
test_num = len(x_data) - train_num - valid_num     #测试集的数目 = 506 - 训练集的数目 - 验证集的数目

# 训练集划分
x_train = x_data[:train_num]
y_train = y_data[:train_num]

# 验证集划分
x_valid = x_data[train_num:train_num+valid_num]
y_valid = y_data[train_num:train_num+valid_num]

# 测试集划分
x_test = x_data[train_num+valid_num:train_num+valid_num+test_num]
y_test = y_data[train_num + valid_num : train_num + valid_num + test_num]
```

转换数据类型

转换为tf.float32数据类型，后面求损失时要和变量W执行tf.matmul操作

```
x_train = tf.cast(x_train, dtype=tf.float32)
x_valid = tf.cast(x_valid, dtype=tf.float32)
x_test = tf.cast(x_test, dtype=tf.float32)
```

构建模型

定义模型

多元线性回归模型仍然是个简单的线性函数，其基本形式还是$y = w * x + b$，只是此处w和b不再是一个标量，形状会不同。根据模型定义，执行的是矩阵叉乘，所以此处调用的是tf.matmul()函数。

```
def model(x, w, b):
    return tf.matmul(x, w) + b
```

创建待优化变量

```
W = tf.Variable(tf.random.normal([12, 1],mean=0.0, stddev=1.0, dtype=tf.float32))
B = tf.Variable(tf.zeros(1) , dtype = tf.float32)
```

```
print(W)
print(B)
```

```
<tf.Variable 'Variable:0' shape=(12, 1) dtype=float32, numpy=
array([[ 0.4603383 ],
       [-0.00832263],
       [ 0.8700424 ],
       [ 0.22785938],
       [ 0.99086046],
       [-0.53891313],
       [ 0.36209586],
       [ 0.02487743],
       [-0.7888076 ],
       [-1.3149714 ],
       [ 2.18732   ],
       [ 0.59125024]], dtype=float32)>
<tf.Variable 'Variable:0' shape=(1,) dtype=float32, numpy=array([0.], dtype=float32)>
```

模型训练

设置超参数

本列将采用小批量梯度下降算法MBGD进行优化

```
training_epochs = 50 # 迭代次数
learning_rate = 0.001 # 学习率
batch_size = 10 # 批量训练一次的样本数
```

设置了一个**batch_size**超参数，用来调整每次进行小批量训练优化的样本数

定义损失函数

定义均方差损失函数

```
# 采用均方差作为损失函数

def loss(x, y, w, b):
    err = model(x, w, b) - y #  计算模型预测值和标签值的差异
    squared_err = tf.square(err)     #  求平方，得出方差
    return tf.reduce_mean(squared_err) # 求均值，得出均方差.
```

定义梯度计算函数

定义梯度计算函数

```
# 计算样本数据[x,y]在参数[w,b]点上的梯度
def grad(x, y, w, b):
    with tf.GradientTape() as tape:
        loss_ = loss(x, y, w, b)
    return tape.gradient(loss_, [w, b])# 返回梯度向量
```

选择优化器

使用tf.keras.optimizers.SGD()声明了一个梯度下降优化器（Optimizer），其学习率通过参数指定。

优化器可以帮助根据计算出的求导结果更新模型参数，从而最小化损失函数，具体使用方式是调用其apply_gradients()方法。

```
optimizer = tf.keras.optimizers.SGD(learning_rate) # 创建优化器，指定学习率
```

迭代训练


```
loss_list_train = [] # 用于保存训练集loss值的列表
loss_list_valid = [] # 用于保存验证集loss值的列表
total_step = int(train_num/batch_size)

for epoch in range (training_epochs):
    for step in range(total_step):
        xs = x_train[step*batch_size:(step+1)*batch_size,:]
        ys = y_train[step*batch_size:(step+1)*batch_size]

        grads = grad(xs, ys, W, B)# 计算梯度
        optimizer.apply_gradients(zip(grads, [W,B])) # 优化器根据梯度自动调整变量w和b

    loss_train = loss(x_train, y_train, W, B).numpy()    # 计算当前轮训练损失
    loss_valid = loss(x_valid, y_valid, W, B).numpy()    # 计算当前轮验证损失
    loss_list_train.append(loss_train)
    loss_list_valid.append(loss_valid)
    print("epoch={:3d} ,train_loss={:.4f},valid_loss={:.4f}".format(epoch+1, loss_train, loss_valid))
```

模型训练

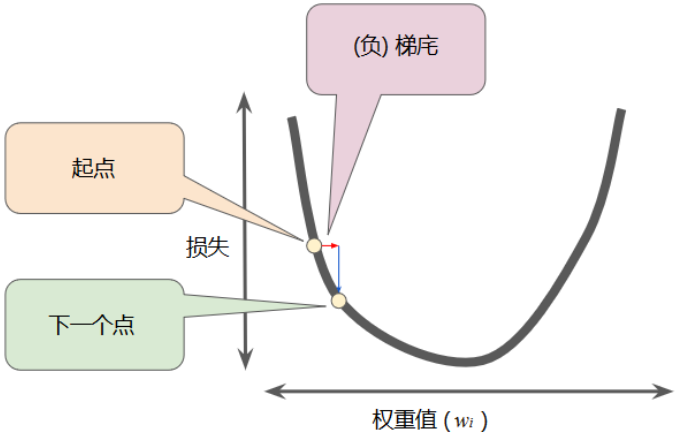
浙江大学城市学院 ZHEJIANG UNIVERSITY CITY COLLEGE

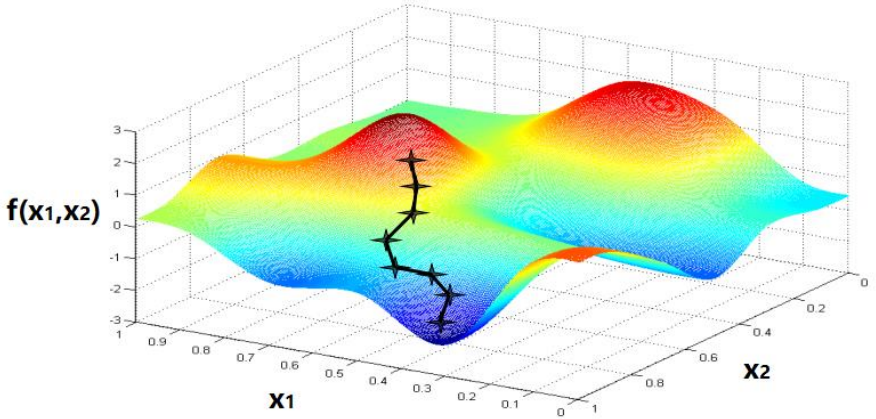
训练过程输出

```
epoch=  1 ,train_loss=nan,valid_loss=nan
epoch=  2 ,train_loss=nan,valid_loss=nan
epoch=  3 ,train_loss=nan,valid_loss=nan
epoch=  4 ,train_loss=nan,valid_loss=nan
epoch=  5 ,train_loss=nan,valid_loss=nan
epoch=  6 ,train_loss=nan,valid_loss=nan
epoch=  7 ,train_loss=nan,valid_loss=nan
epoch=  8 ,train_loss=nan,valid_loss=nan
epoch=  9 ,train_loss=nan,valid_loss=nan
epoch= 10 ,train_loss=nan,valid_loss=nan
epoch= 11 ,train_loss=nan,valid_loss=nan
epoch= 12 ,train_loss=nan,valid_loss=nan
epoch= 13 ,train_loss=nan,valid_loss=nan
epoch= 14 ,train_loss=nan,valid_loss=nan
epoch= 15 ,train_loss=nan,valid_loss=nan
epoch= 16 ,train_loss=nan,valid_loss=nan
```

探究训练结果异常的原因：从梯度下降讲起

梯度下降





单变量（一元）

两个变量（二元）

梯度下降

要考虑不同特征值取值范围大小的影响

杭州笋干老鸭汤

原材料的配比：鸭、火腿、笋干、青菜、水、油、盐、酱油、葱、姜、蒜、胡椒粉.....

梯度下降

要考虑不同特征值取值范围大小的影响

	CRIM	ZN	INDUS	CHAS	NOX	RM	\
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	
75%	3.677082	12.500000	18.100000	0.000000	0.624000	6.623500	
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	

	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	\
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	
mean	68.574901	3.795043	9.549407	408.237154	18.455534	12.653063	
std	28.148861	2.105710	8.707259	168.537116	2.164946	7.141062	
min	2.900000	1.129600	1.000000	187.000000	12.600000	1.730000	
25%	45.025000	2.100175	4.000000	279.000000	17.400000	6.950000	
50%	77.500000	3.207450	5.000000	330.000000	19.050000	11.360000	
75%	94.075000	5.188425	24.000000	666.000000	20.200000	16.955000	
max	100.000000	12.126500	24.000000	711.000000	22.000000	37.970000	

归一化

(特征值-特征值最小值)/ (特征值最大值 - 特征值最小值）

0-1

版本2：特征数据归一化

特征数据归一化

划分特征数据和标签数据

```
# x_data 为 归一化后的前12列特征数据
x_data = ds[:,:12]

# y_data 为最后1列标签数据
y_data = ds[:,12]
```

特征数据归一化

```
# 对特征数据 【0到11】列 做（0-1）归一化
for i in range(12):
    x_data[:,i]=(x_data[:,i]-x_data[:,i].min())/(x_data[:,i].max()-x_data[:,i].min())
```

其他代码保持不变

特征数据归一化（另一种方案）

在sklearn.preprocessing里提供了scale()方法可以直接应用。在scale()里执行的转换公式如下面公式所示

$$x_i' = \frac{x_i - \mathrm{mean}(x)}{std(\mathrm{x})}$$

可以将原来做数据类型转换的3句代码修改为一下代码

```
x_train = tf.cast(scale(x_train), dtype=tf.float32)
x_valid = tf.cast(scale(x_valid), dtype=tf.float32)
x_test = tf.cast(scale(x_test), dtype=tf.float32)
```

其他代码保持不变

数据归一化后模型运行结果

浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

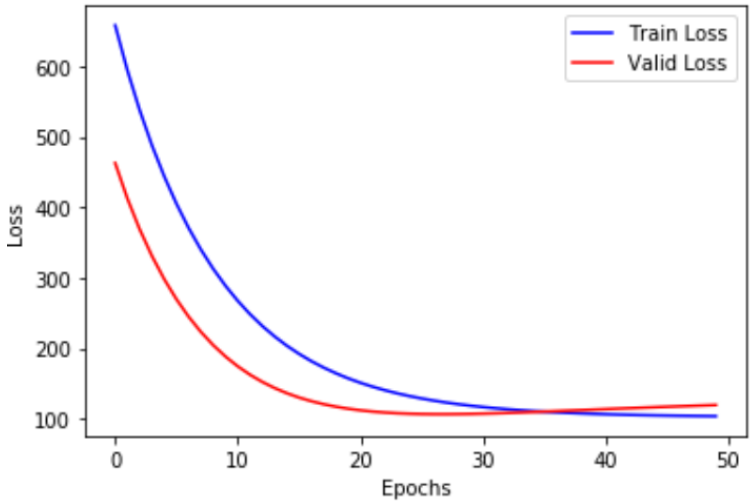
Oh，Yeah！

```
epoch=  1 ,train_loss=658.3249,valid_loss=462.6978
epoch=  2 ,train_loss=594.1818,valid_loss=412.2452
epoch=  3 ,train_loss=537.7859,valid_loss=368.6745
epoch=  4 ,train_loss=488.0065,valid_loss=330.8622
epoch=  5 ,train_loss=443.9688,valid_loss=297.9776
epoch=  6 ,train_loss=404.9637,valid_loss=269.3693
epoch=  7 ,train_loss=370.3961,valid_loss=244.5029
epoch=  8 ,train_loss=339.7545,valid_loss=222.9245
epoch=  9 ,train_loss=312.5926,valid_loss=204.2407
epoch= 10 ,train_loss=288.5175,valid_loss=188.1062
epoch= 11 ,train_loss=267.1812,valid_loss=174.2155
epoch= 12 ,train_loss=248.2750,valid_loss=162.2975
epoch= 13 ,train_loss=231.5246,valid_loss=152.1116
epoch= 14 ,train_loss=216.6863,valid_loss=143.4440
epoch= 15 ,train_loss=203.5434,valid_loss=136.1052
epoch= 16 ,train_loss=191.9033,valid_loss=129.9269
epoch= 17 ,train_loss=181.5950,valid_loss=124.7604
epoch= 18 ,train_loss=172.4669,valid_loss=120.4744
```

```
epoch= 34 ,train_loss=112.0141,valid_loss=108.6624
epoch= 35 ,train_loss=110.8636,valid_loss=109.2946
epoch= 36 ,train_loss=109.8453,valid_loss=109.9585
epoch= 37 ,train_loss=108.9441,valid_loss=110.6446
epoch= 38 ,train_loss=108.1466,valid_loss=111.3449
epoch= 39 ,train_loss=107.4409,valid_loss=112.0530
epoch= 40 ,train_loss=106.8166,valid_loss=112.7630
epoch= 41 ,train_loss=106.2643,valid_loss=113.4702
epoch= 42 ,train_loss=105.7758,valid_loss=114.1707
epoch= 43 ,train_loss=105.3438,valid_loss=114.8613
epoch= 44 ,train_loss=104.9619,valid_loss=115.5393
epoch= 45 ,train_loss=104.6242,valid_loss=116.2026
epoch= 46 ,train_loss=104.3260,valid_loss=116.8497
epoch= 47 ,train_loss=104.0625,valid_loss=117.4791
epoch= 48 ,train_loss=103.8298,valid_loss=118.0900
epoch= 49 ,train_loss=103.6245,valid_loss=118.6817
epoch= 50 ,train_loss=103.4434,valid_loss=119.2537
```

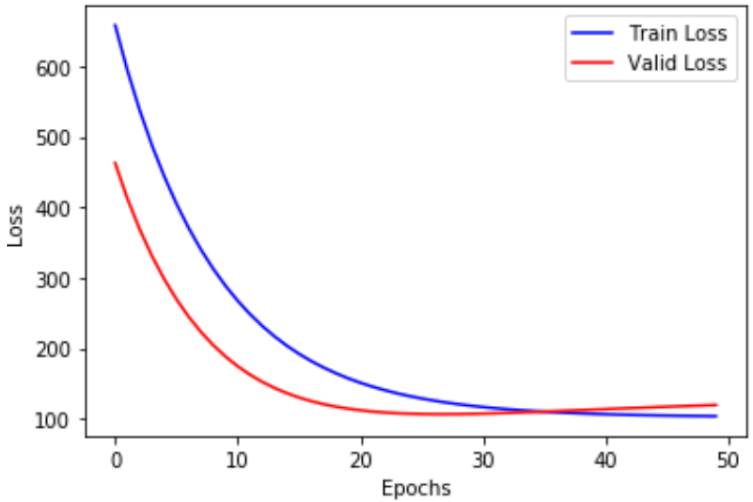
可视化损失值

```
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.plot(loss_list_train,'blue',label="Train Loss")
plt.plot(loss_list_valid,'red',label="Valid Loss")
plt.legend(loc=1)# 通过参数loc指定图例位置
```



模型训练

思考：根据损失值的变化，该如何调整超参数？



查看测试集损失值

查看测试集的损失

```
print("Test_loss: {:.4f}".format(loss(x_test, y_test, W, B).numpy()))
```

```
Test_loss: 112.8085
```

模型应用

模型应用

测试集里随机选一条

应用模型

```
test_house_id = np.random.randint(0,test_num)
y = y_test[test_house_id]
y_pred = model(x_test,W,B)[test_house_id]
y_predit=tf.reshape(y_pred,()).numpy()
print("House id",test_house_id,"Actual value",y,"Predicted value ",y_predit)
```

```
House id 82 Actual value 25.0 Predicted value  28.429281
```