# Contents

线性回归 **2**

背景介绍 . . . . . 2

效果展示 . . . . . 2

模型概览 . . . . . 2

模型定义 . . . . . 2

训练过程 . . . . . 4

数据准备 . . . . . 4

数据预处理 . . . . . 5

提供数据给 PaddlePaddle . . . . . 7

模型配置说明 . . . . . 7

数据定义 . . . . . 7

算法配置 . . . . . 7

网络结构 . . . . . 8

训练模型 . . . . . 8

应用模型 . . . . . 8

总结 . . . . . 9

参考文献 . . . . . 9

# 线性回归

让我们从经典的线性回归（Linear Regression [1]）模型开始这份教程。在这一章里，你将使用真实的数据集建立起一个房价预测模型，并且了解到机器学习中的若干重要概念。

## 背景介绍

给定一个大小为 $n$ 的数据集 $\{y_i, x_{i1}, ..., x_{id}\}_{i=1}^{n}$，其中 $x_{i1}, \ldots, x_{id}$ 是第 $i$ 个样本 $d$ 个属性上的取值，$y_i$ 是该样本待预测的目标。线性回归模型假设目标 $y_i$ 可以被属性间的线性组合描述，即

$$y_i = \omega_1 x_{i1} + \omega_2 x_{i2} + \ldots + \omega_d x_{id} + b, i = 1, \ldots, n$$

例如，在我们将要建模的房价预测问题里，$x_{ij}$ 是描述房子 $i$ 的各种属性（比如房间的个数、周围学校和医院的个数、交通状况等），而 $y_i$ 是房屋的价格。

初看起来，这个假设实在过于简单了，变量间的真实关系很难是线性的。但由于线性回归模型有形式简单和易于建模分析的优点，它在实际问题中得到了大量的应用。很多经典的统计学习、机器学习书籍 [2,3,4] 也选择对线性模型独立成章重点讲解。

## 效果展示

我们使用从UCI Housing Data Set获得的波士顿房价数据集进行模型的训练和预测。下面的散点图展示了使用模型对部分房屋价格进行的预测。其中，每个点的横坐标表示同一类房屋真实价格的中位数，纵坐标表示线性回归模型根据特征预测的结果，当二者值完全相等的时候就会落在虚线上。所以模型预测得越准确，则点离虚线越近。

## 模型概览

### 模型定义

在波士顿房价数据集中，和房屋相关的值共有 14 个：前 13 个用来描述房屋相关的各种信息，即模型中的 $x_i$；最后一个值为我们要预测的该类房价格的中位数，即模型中的 $y_i$。因此，我们的模型就可以表示成：


Figure 1: 预测值 V.S. 真实值

$$\hat{Y} = \omega_1 X_1 + \omega_2 X_2 + \ldots + \omega_{13} X_{13} + b$$

$\hat{Y}$ 表示模型的预测结果，用来和真实值 $Y$ 区分。模型要学习的参数即：$\omega_1, \ldots, \omega_{13}, b$。

建立模型后，我们需要给模型一个优化目标，使得学到的参数能够让预测值 $\hat{Y}$ 尽可能地接近真实值 $Y$。这里我们引入损失函数（Loss Function，或 Cost Function）这个概念。输入任意一个数据样本的目标值 $y_i$ 和模型给出的预测值 $\hat{y_i}$，损失函数输出一个非负的实值。这个实质通常用来反映模型误差的大小。

对于线性回归模型来讲，最常见的损失函数就是均方误差（Mean Squared Error，MSE）了，它的形式是：

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{Y_i} - Y_i)^2$$

即对于一个大小为 $n$ 的测试集，$MSE$ 是 $n$ 个数据预测结果误差平方的均值。

训练过程

定义好模型结构之后，我们要通过以下几个步骤进行模型训练 1. 初始化参数，其中包括权重 $\omega_i$ 和偏置 $b$，对其进行初始化（如 0 均值，1 方差）。2. 网络正向传播计算网络输出和损失函数。3. 根据损失函数进行反向误差传播（backpropagation），将网络误差从输出层依次向前传递, 并更新网络中的参数。4. 重复 2~3 步骤，直至网络训练误差达到规定的程度或训练轮次达到设定值。

## 数据准备

执行以下命令来准备数据:

```
cd data && python prepare_data.py
```

这段代码将从UCI Housing Data Set下载数据并进行预处理，最后数据将被分为训练集和测试集。

这份数据集共 506 行，每行包含了波士顿郊区的一类房屋的相关信息及该类房屋价格的中位数。其各维属性的意义如下：

| 属性名 | 解释 | 类型 |
|---|---|---|
| CRIM | 该镇的人均犯罪率 | 连续值 |
| ZN | 占地面积超过 25,000 平方呎的住宅用地比例 | 连续值 |
| INDUS | 非零售商业用地比例 | 连续值 |
| CHAS | 是否邻近 Charles River | 离散值，1= 邻近；0= 不邻近 |
| NOX | 一氧化氮浓度 | 连续值 |
| RM | 每栋房屋的平均客房数 | 连续值 |
| AGE | 1940 年之前建成的自用单位比例 | 连续值 |
| DIS | 到波士顿 5 个就业中心的加权距离 | 连续值 |
| RAD | 到径向公路的可达性指数 | 连续值 |
| TAX | 全值财产税率 | 连续值 |
| PTRATIO | 学生与教师的比例 | 连续值 |
| B | 1000(BK - 0.63)^2，其中 BK 为黑人占比 | 连续值 |
| LSTAT | 低收入人群占比 | 连续值 |
| MEDV | 同类房屋价格的中位数 | 连续值 |

数据预处理

连续值与离散值　观察一下数据，我们的第一个发现是：所有的 13 维属性中，有 12 维的连续值和 1 维的离散值（CHAS）。离散值虽然也常使用类似 0、1、2 这样的数字表示，但是其含义与连续值是不同的，因为这里的差值没有实际意义。例如，我们用 0、1、2 来分别表示红色、绿色和蓝色的话，我们并不能因此说“蓝色和红色”比“绿色和红色”的距离更远。所以通常对一个有 $d$ 个可能取值的离散属性，我们会将它们转为 $d$ 个取值为 0 或 1 的二值属性或者将每个可能取值映射为一个多维向量。不过就这里而言，因为 CHAS 本身就是一个二值属性，就省去了这个麻烦。

属性的归一化　另外一个稍加观察即可发现的事实是，各维属性的取值范围差别很大（如图 2 所示）。例如，属性 B 的取值范围是 [0.32, 396.90]，而属性 NOX 的取值范围是 [0.3850, 0.8170]。这里就要用到一个常见的操作-归一化（normalization）了。归一化的目标是把各位属性的取值范围放缩到差不多的区间，例如 [-0.5,0.5]。这里我们使用一种很常见的操作方法：减掉均值，然后除以原取值范围。

做归一化（或 Feature scaling）至少有以下 3 个理由：- 过大或过小的数值范围会导致计算时的浮点上溢或下溢。- 不同的数值范围会导致不同属性对模型的重要性不同（至少在训练的初始阶段如此），而这个隐含的假设常常是不合理的。这会对优化的过程造成困难，使训练时间大大的加长。- 很多的机器学习技巧/模型（例如 L1，L2 正则项，向量空间模型-Vector Space Model）都基于这样的假设：所有的属性取值都差不多是以 0 为均值且取值范围相近的。

整理训练集与测试集　我们将数据集分割为两份：一份用于调整模型的参数，即进行模型的训练，模型在这份数据集上的误差被称为训练误差；另外一份被用来测试，模型在这份数据集上的误差被称为测试误差。我

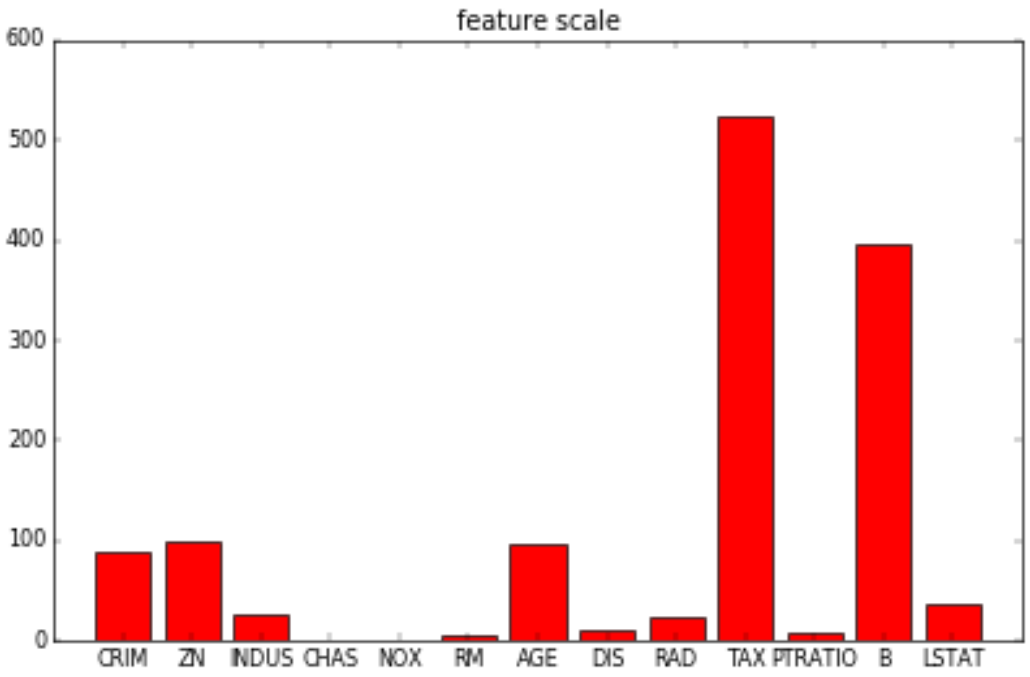


Figure 2: 各维属性的取值范围

们训练模型的目的是为了通过从训练数据中找到规律来预测未知的新数据，所以测试误差是更能反映模型表现的指标。分割数据的比例要考虑到两个因素：更多的训练数据会降低参数估计的方差，从而得到更可信的模型；而更多的测试数据会降低测试误差的方差，从而得到更可信的测试误差。一种常见的分割比例为 8 : 2，感兴趣的读者朋友们也可以尝试不同的设置来观察这两种误差的变化。

执行如下命令可以分割数据集，并将训练集和测试集的地址分别写入 train.list 和 test.list 两个文件中，供 PaddlePaddle 读取。

```
python prepare_data.py -r 0.8 # 默认使用 8:2 的比例进行分割
```

在更复杂的模型训练过程中，我们往往还会多使用一种数据集：验证集。因为复杂的模型中常常还有一些超参数（Hyperparameter）需要调节，所以我们会尝试多种超参数的组合来分别训练多个模型，然后对比它们在验证集上的表现选择相对最好的一组超参数，最后才使用这组参数下训练的模型在测试集上评估测试误差。由于本章训练的模型比较简单，我们暂且忽略掉这个过程。

提供数据给 **PaddlePaddle**

准备好数据之后，我们使用一个 Python data provider 来为 PaddlePaddle 的训练过程提供数据。一个 data provider 就是一个 Python 函数，它会被 PaddlePaddle 的训练过程调用。在这个例子里，只需要读取已经保存好的数据，然后一行一行地返回给 PaddlePaddle 的训练进程即可。

```
from paddle.trainer.PyDataProvider2 import *
import numpy as np
# 定义数据的类型和维度
@provider(input_types=[dense_vector(13), dense_vector(1)])
def process(settings, input_file):
    data = np.load(input_file.strip())
    for row in data:
        yield row[:-1].tolist(), row[-1:].tolist()
```

## 模型配置说明

### 数据定义

首先，通过 `define_py_data_sources2` 来配置 PaddlePaddle 从上面的 `dataprovider.py` 里读入训练数据和测试数据。PaddlePaddle 接受从命令行读入的配置信息，例如这里我们传入一个名为 `is_predict` 的变量来控制模型在训练和测试时的不同结构。

```
from paddle.trainer_config_helpers import *

is_predict = get_config_arg('is_predict', bool, False)

define_py_data_sources2(
    train_list='data/train.list',
    test_list='data/test.list',
    module='dataprovider',
    obj='process')
```

### 算法配置

接着，指定模型优化算法的细节。由于线性回归模型比较简单，我们只要设置基本的 `batch_size` 即可，它指定每次更新参数的时候使用多少条数据计算梯度信息。

```
settings(batch_size=2)
```

网络结构

最后，使用 `fc_layer` 和 `LinearActivation` 来表示线性回归的模型本身。

```
# 输入数据，13 维的房屋信息
x = data_layer(name='x', size=13)

y_predict = fc_layer(
    input=x,
    param_attr=ParamAttr(name='w'),
    size=1,
    act=LinearActivation(),
    bias_attr=ParamAttr(name='b'))

if not is_predict: # 训练时，我们使用 MSE，即 regression_cost 作为损失函数
    y = data_layer(name='y', size=1)
    cost = regression_cost(input=y_predict, label=y)
    outputs(cost) # 训练时输出 MSE 来监控损失的变化
else: # 测试时，输出预测值
    outputs(y_predict)
```

## 训练模型

在对应代码的根目录下执行 PaddlePaddle 的命令行训练程序。这里指定模型配置文件为 `trainer_config.py`，训练 30 轮，结果保存在 `output` 路径下。

```
./train.sh
```

## 应用模型

现在来看下如何使用已经训练好的模型进行预测。

```
python predict.py
```

这里默认使用 `output/pass-00029` 中保存的模型进行预测，并将数据中的房价与预测结果进行对比，结果保存在 `predictions.png` 中。如果你想使用别的模型或者其它的数据进行预测，只要传入新的路径即可：

```
python predict.py -m output/pass-00020 -t data/housing.test.npy
```

## 总结

在这章里，我们借助波士顿房价这一数据集，介绍了线性回归模型的基本概念，以及如何使用 PaddlePaddle 实现训练和测试的过程。很多的模型和技巧都是从简单的线性回归模型演化而来，因此弄清楚线性模型的原理和局限非常重要。

## 参考文献

1. https://en.wikipedia.org/wiki/Linear_regression
2. Friedman J, Hastie T, Tibshirani R. The elements of statistical learning[M]. Springer, Berlin: Springer series in statistics, 2001.
3. Murphy K P. Machine learning: a probabilistic perspective[M]. MIT press, 2012.
4. Bishop C M. Pattern recognition[J]. Machine Learning, 2006, 128.