Homework 2: Multivariate Linear Regression

16340232 王泽浩

github repo：https://github.com/hansenbeast/2019-SYSU-DM/tree/master/HW2

Exercise 1

你需要用多少个参数来训练该线性回归模型？请使用梯度下降方法训练。训练时，请把迭代次数设成 1500000，学习率设成 0.00015，参数都设成 0.0。在训练的过程中，每迭代 100000 步，计算训 练样本对应的误差，和使用当前的参数得到的测试样本对应的误差。请画图显示迭代到达 100000 步、200000 步、... ... 1500000 时对应的训练样本的误差和测试样本对应的误差（图可以手画，或者用工具画 图）。从画出的图中，你发现什么？请简单分析。

算法概述

多元线性回归模型

在此多元线性回归模型中，自变量为一个向量，为训练样本的前两列数据，因变量为一个标量，为训练样本的第三列数据

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \epsilon$$
$$E(y) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

根据如上公式发现需要三个未知参数，其中 $[\beta_1, \beta_2]'$ 为斜率，β_0 为偏差项，即截距

由于观测值，即训练数据有若干个，因此把这些观测值按行叠加起来就成为了一个向量或者矩阵表示为

$$y=\begin{bmatrix}y_1\\y_2\\\vdots\\y_n\end{bmatrix},X=\begin{bmatrix}1&x_{11}&\cdots&x_{1p}\\1&x_{21}&\cdots&x_{2p}\\\vdots&\vdots&&\vdots\\1&x_{n1}&\cdots&x_{np}\end{bmatrix},\epsilon=\begin{bmatrix}\epsilon_1\\\epsilon_2\\\vdots\\\epsilon_n\end{bmatrix},\beta=\begin{bmatrix}\beta_0\\\beta_1\\\vdots\\\beta_p\end{bmatrix}$$

这时多元线性回归的表示就变成了

$$y=X\beta+\epsilon$$

其中，噪声误差 $\epsilon\sim N(0,\sigma^2)$

损失函数

引入最大似然估计 MLE，**似然函数**与概率非常类似但又有根本的区别，概率为在某种条件（参数）下预测某事件发生的可能性；而似然函数与之相反，为已知该事件的情况下**推测出该事件发生时的条件（参数）**；所以似然估计也称为参数估计，为参数估计中的一种算法

对于单个训练数据，回归模型也可以表示为：

$$y^{(i)}=\theta^T x^{(i)}+\varepsilon^{(i)}$$

$$Y\sim N(\theta^T x,\sigma^2)$$

$$p(y|x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(y-\theta^T x)^2}{2\sigma^2}\right)$$

假设数据集是独立同分布的，则联合概率密度函数为

$$p(\mathcal{D}) = \prod_{i=1}^{n} p(x_i, y_i)$$

$$= \prod_{i=1}^{n} \cancel{p(x_i)} p(y_i|x_i)$$

"1"

Discriminative Model

将 p（x，y）带入上式得

$$\mathcal{L}(\theta|\mathcal{D}) = \prod_{i=1}^{n} p_\theta(y_i|x_i)$$

$$= \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y_i - \theta^T x_i)^2}{2\sigma^2}\right)$$

$$= \frac{1}{\sigma^n (2\pi)^{\frac{n}{2}}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2\right)$$

因为log函数为单调递增的，不影响极值处理，取对数得

$$\log \mathcal{L}(\theta|\mathcal{D}) = -\log(\sigma^n (2\pi)^{\frac{n}{2}}) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2$$

移除不带 θ 的常数项后

$$\log \mathcal{L}(\theta) = -\sum_{i=1}^{n} (y_i - \theta^T x_i)^2$$

Monotone Function
(Easy to maximize)

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta \in \mathbb{R}^p} -\sum_{i=1}^{n} (y_i - \theta^T x_i)^2$$

最后得到损失函数为真实值和估计值的误差平方和

$$\hat{\theta}_{\text{MLE}} = \arg\min_{\theta \in \mathbb{R}^p} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2$$

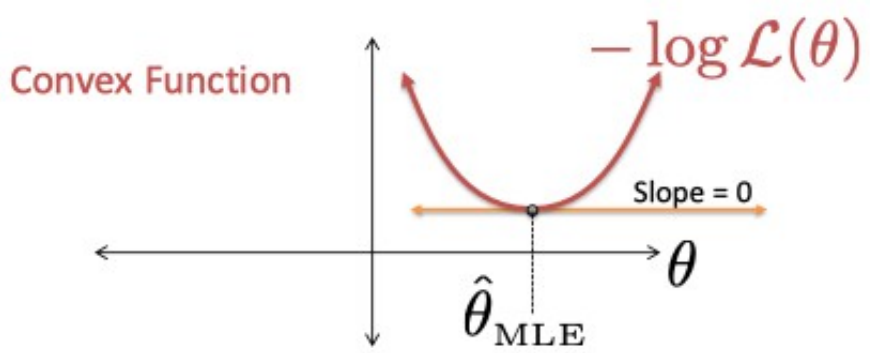
Minimize Sum (Error)²

当损失函数最小时，我们就能得到该数据集最吻合的正态分布对应的概率分布函数的总似然最大的情况，也就是我们想要的最优解。

梯度下降

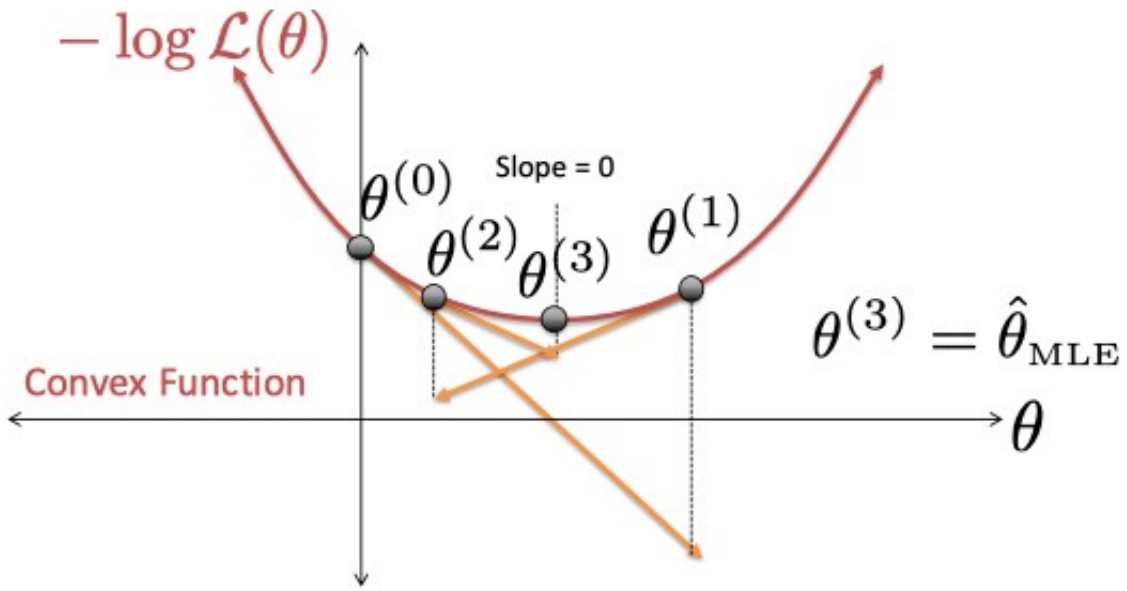
对凸二次函数使用链式法则进行梯度计算

$$i=1$$



$$-\nabla_\theta \log \mathcal{L}(\theta) = -2\sum_{i=1}^{n} y_i x_i + 2\sum_{i=1}^{n} (\theta^T x_i) x_i$$
$$= -2X^T Y + 2X^T X\theta$$

梯度决定了损失函数向着局部极小值下降的最快方向，学习率则为步长



每次迭代，更新 θ

For τ from 0 until *convergence*

$$\begin{aligned}\theta^{(\tau+1)} &= \theta^{(\tau)} - \rho(\tau)\nabla \log \mathcal{L}(\theta^{(\tau)}|D) \\ &= \theta^{(\tau)} + \rho(\tau)\frac{1}{n}\sum_{i=1}^{n}(y_i - \theta^{(\tau)T}x_i)x_i \quad O(np)\end{aligned}$$

数据标准化

由于以下原因

- 数值计算是通过计算来迭代逼近的，如果自变量量数量级相差太大，则很容易在运算过程中丢失，出现 nan 的结果
- 不同数量级的自变量对权重影响的不同

需要通过预处理，让初始的特征量具有同等的地位，这个预处理的过程为**数据标准化（Normalization）**

本次使用的是**特征缩放（feature scaling）**的方法进行标准化，即

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

具体代码

使用 python 的 tensorflow 框架进行训练

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing

# Try to find value for W and b to compute y_data = x_data * W + b

# Define dimensions
d = 2    # Size of the parameter space
N = 50 # Number of data sample

# Model parameters
W = tf.Variable(tf.zeros([d, 1], tf.float32), name="weights")
b = tf.Variable(tf.zeros([1], tf.float32), name="biases")

# Model input and output
x = tf.placeholder(tf.float32, shape=[None, d])
y = tf.placeholder(tf.float32, shape=[None, 1])

# hypothesis
linear_regression_model = tf.add(tf.matmul(x, W), b)
# cost/loss function
loss = tf.reduce_sum(tf.square(linear_regression_model - y))

# optimizer
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.00015)
train = optimizer.minimize(loss)

training_filename = "dataForTraining.txt"
testing_filename = "dataForTesting.txt"
training_dataset = np.loadtxt(training_filename)
testing_dataset = np.loadtxt(testing_filename)
dataset = np.vstack((training_dataset,testing_dataset))
min_max_scaler = preprocessing.MinMaxScaler()
dataset = min_max_scaler.fit_transform(dataset)

x_train = np.array(dataset[:50,:2])
y_train = np.array(dataset[:50,2:3])
x_test = np.array(dataset[50:,:2])
y_test = np.array(dataset[50:,2:3])

save_step_loss = {"step":[],"train_loss":[],"test_loss":[]}# 保存step和loss用于可视化操
作
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)  # reset values to wrong
    steps = 1500001
```

```
    for i in range(steps):
        sess.run(train, {x: x_train, y: y_train})
        if i % 100000 == 0:
            # evaluate training loss
            print("iteration times: %s" % i)
            curr_W, curr_b, curr_train_loss = sess.run([W, b, loss], {x: x_train, y:
y_train})
            print("W: %s \nb: %s \nTrain Loss: %s" % (curr_W, curr_b,
curr_train_loss))
            # evaluate testing loss
            curr_test_loss = sess.run(loss,{x:x_test,y:y_test})
            print("Test Loss: %s\n" % curr_test_loss)
            save_step_loss["step"].append(i)
            save_step_loss["train_loss"].append(curr_train_loss)
            save_step_loss["test_loss"].append(curr_test_loss)

#画图损失函数变化曲线
...
```

结果分析