Linear Regression & Closed-form& SGD

Abstract—We study two methods for solving the parameters in linear regression: closed-form solution and stochastic

I. Introduction

Linear regression wants to learn a linear model between predictor and the real value more precisely.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \theta^T x$$

And in this report we want to find the best value of θ^T to achieve the goal

II. Methods and Theory

A. Closed-form Solution:

Loss function:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y)$$

We can get the best θ by following formula:

$$\theta = (X^T X)^{-1} X^T y$$

B. Stochastic gradient descent

We require solution makes the J (theta) the smallest value of theta and gradient descent algorithm is probably thinking: we first give the theta an initialization value, and then change theta value decreased let J (theta) value, repeated change theta process made the J (theta) smaller until J (theta) is approximately equal to the minimum first we give theta a initial value, and then to let J (theta) change the direction of the largest update theta values, so the iterative formula is as follows:

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

Solve the derivation:

Random gradient descent selects a random data for calculation when calculating the direction of the fastest decline, instead of scanning all the training data sets. In this way, the iterative speed is accelerated. The random gradient descent does not converge along the direction of the fastest fall in J(theta), but rather approaches the minimum point in the way of oscillation

Loop {

  for i=1 to m, {

    $\theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j^{(i)}$ (for every j).

  }

}

III. Experiment

Data

In this experiment, we use the housing_scale dataset in the LIBSVM Data.

Each row represent a sample with 13 features.There are 506 samples in the dataset.Divide them into training set and validation set with the proportion of 3:1 (training set:379, validation set:127)

Implementation

Closed-form Solution:

Load and split the dataset

线性回归（闭式解）

1. 导入依赖

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
from scipy.sparse import linalg
```

2. 导入和分割数据集

```
Dataset = load_svmlight_file('housing_scale.txt', n_features=13)
X_train, X_val, y_train, y_val = train_test_split(Dataset[0], Dataset[1], test_size=0.2,
```

Loss functions:

3.构建线性模型

y=wx

损失函数

```
def L2Loss(y, y_):
    return ((y-y_)**2)

def L1Loss(y, y_):
    return np.abs(y-y_)

Loss = L2Loss
```

Calculate the loss of the origin loss:

6.在训练集上计算损失

```
pred_train = X_train.dot(W)
loss_train = Loss(pred_train, y_train)
print('mean(loss_train) = {}'.format(loss_train.mean()))
```

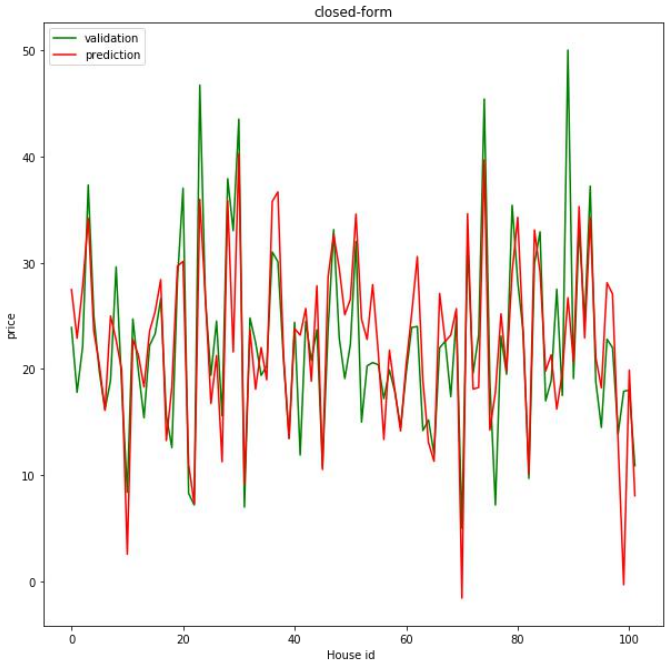
```
mean(loss_train) = 23.726942024006856
```

7. 计算验证集上的损失

```
pred_val = X_val.dot(W)
loss_val = Loss(pred_val, y_val).mean()
print('mean(loss_val) = {}'.format(loss_val.mean()))
```

```
mean(loss_val) = 27.20850718361358
```

```
plt.figure(figsize=[10,10])
plt.title('closed-form')
plt.xlabel('House id ')
plt.ylabel('price')
plt.plot(y_val,c='g', label='validation')
plt.plot(pred_val,c='r', label='prediction')
plt.legend()
plt.show()
```



随机梯度下降

1. 导入依赖

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
from scipy.sparse import linalg
```

2. 切分数据集

```
Dataset = load_svmlight_file('housing_scale.txt', n_features=13)
X_train, X_val, y_train, y_val = train_test_split(Dataset[0], Dataset[1], test_size=0.2,
```

损失函数

```
def L2Loss(y, y_):
    return ((y-y_)**2)

def L1Loss(y, y_):
    return np.abs(y-y_)

Loss = L2Loss
```

4. 在验证集上计算loss

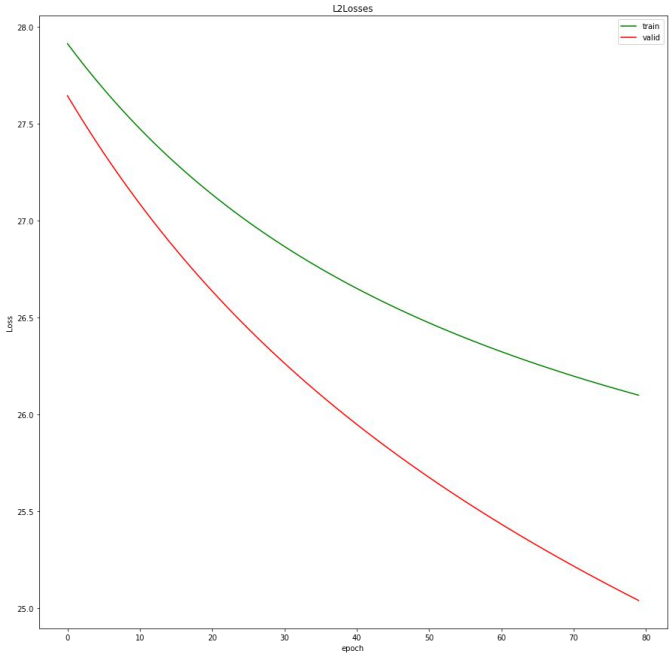
```
W = np.random.normal(1, 1, size=(13))
pred_init = X_train.dot(W)
loss_init = Loss(pred_init, y_train)
print('loss = {}'.format(loss_init.mean()))
```

```
loss = 681.6427696138634
```

```
losses_train, losses_val = [], []

batch_idxs_pool = np.arange(X_train.shape[0])

for epoch in range(EPOCH_NUM):
#     for step in range(X_train.shape[0] // BATCH_SIZE):
#         batch = np.random.choice(batch_idxs_pool, size=BATCH_SIZE)
    for batch in range(X_train.shape[0]):
        X, y = X_train[batch], y_train[batch]

        grad = X.T.dot(X.dot(W)-y) #+ PENALTY_FACTOR*w
        grad = -grad / X.shape[0]
        W += LR * grad

    losses_train.append(Loss(X_train.dot(W), y_train).mean())
    losses_val.append(Loss(X_val.dot(W), y_val).mean())
#     print('Epoch: #{}\tLoss_train: {}\tLoss_val: {}'.format(epoch+1, losses_train[-1],

print('Losstrain is {:.3f}, Loss_val is {:.3f}'.format(losses_train[-1], losses_val[-1]))
```

```
Losstrain is 26.100, Loss_val is 25.041
```

```
plt.figure(figsize=[15,15])
plt.title('L2Losses')
plt.plot(losses_train, c='g', label='train')
plt.xlabel('epoch')
plt.ylabel('Loss')
plt.plot(losses_val, c='r', label='valid')
plt.legend()
plt.show()
```



IV. CONCLUSION

It is a difficult start of machine learning. Although there are many difficulties, I am interested in exploring the linear regression algorithm. In addition, this experiment enhanced my code skills.