Predicting Boston Housing Prices

Homework Explanation

罗宇矗

Monday, 24 July 2017

编程练习 1：基础统计运算

Right Code

```
minimum_price = np.min(prices)
maximum_price = np.max(prices)
mean_price = np.average(prices)
median_price = np.median(prices)
std_price = np.std(prices)
```

应使用 NumPy 进行运算

编程练习 1：基础统计运算

标准差（Standard deviation）

std_price = np.sqrt(np.var(prices))

std_price = prices.std()

std_price = np.std(prices)

- 样本标准差 (Sample std)

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}.$$

- 总体标准差 (Population std)

$$SD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

WARN: Pandas 中的 std 方法求的是样本标准差

编程练习 2：数据分割与重排

不能使用额外的 Package

Wrong Code

```
from sklearn.cross_validation import train_test_split

def generate_train_and_test(X, y):
    """打乱并分割数据为训练集和测试集"""
    return train_test_split(X, y, test_size = 0.20, train_size = 0.80, random_state = 100)

X_train, X_test, y_train, y_test = generate_train_and_test(features, prices)
```

编程练习 2：数据分割与重排

Right Code

```
    LEN, seed = 490, 2
def generate_train_and_test(X, y):
    """打乱并分割数据为训练集和测试集"""
    np.random.seed(seed)
    shuffled_list = np.random.permutation(LEN)
    split_point = int(LEN * 0.8)
    train_list, test_list = shuffled_list[:split_point], shuffled_list[split_point:]
    return X[train_list], X[test_list], y[train_list], y[test_list]
```

问题 2：训练及测试

Wrong Answer

训练用的数据集用来生成模型，测试用的数据集可以根据结果，分辨模型的好坏，以便进一步调整模型

问题 2：训练及测试

Wrong Answer

没有数据的模型一般来说是没有参考价值的，也就是我们练习的时候可能拿不到具体的数据，所以可以练习的数据集来完成项目的调试，而如果没有数据让我们去完成一个项目的话其实得到的并没有太大的意义。而分成训练用的数据集和测试用的数据集也能更好的促进学生来完成训练的调试和测试时候的严谨性

问题 2：训练及测试

Wrong Answer

如果得到的模型没有真实数据进行测试，那并不能知道这个模型的可靠性。一部分作出模型，一部分测试模型，查看二者的平均误差值可以体现这个模型是不是好的

问题 2：训练及测试

Wrong Answer

我们的模型参数很有可能是在test data上过拟合，如果我们在test data上做模型参数选择，又用它做效果评估，显然不是那么合理,所以我们通常会把训练数据分为两个部分，一大部分作为训练用，另外一部分就是所谓的cross validation数据集，用来进行模型参数选择的。

问题 2：训练及测试

Right Answer

防止过拟合

编程练习 4：梯度下降

Suppose that θ has two variables $\{\theta_1, \theta_2\}$

Randomly start at $\theta^0 = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix}$

$$\nabla L(\theta) = \begin{bmatrix} \partial L(\theta_1)/\partial \theta_1 \\ \partial L(\theta_2)/\partial \theta_2 \end{bmatrix}$$

$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix} - \eta \begin{bmatrix} \partial L(\theta_1^0)/\partial \theta_1 \\ \partial L(\theta_2^0)/\partial \theta_2 \end{bmatrix} \Rightarrow \theta^1 = \theta^0 - \eta \nabla L(\theta^0)$$

$$\begin{bmatrix} \theta_1^2 \\ \theta_2^2 \end{bmatrix} = \begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} - \eta \begin{bmatrix} \partial L(\theta_1^1)/\partial \theta_1 \\ \partial L(\theta_2^1)/\partial \theta_2 \end{bmatrix} \Rightarrow \theta^2 = \theta^1 - \eta \nabla L(\theta^1)$$

Input: X, y, theta_init, alpha

Output: optimal theta

Not good enough

```
for i in range(num_iters):
    predictions = X.dot(theta)
    for it in range(theta_size):
        temp = X[:, it]
        temp.shape = (m, 1)
        errors_x1 = (predictions - y) * temp
        theta[it][0] = theta[it][0] -
            alpha * (1.0 / m) * errors_x1.sum()
```

Not good enough

```
while not collect:
    g0 = 1.0/m * sum([(t0 + t1*x[i,0] + t2*x[i,1] + t3*x[i,2]- y[i]) for i in range(m)])
    g1 = 1.0/m * sum([(t0 + t1*x[i,0] + t2*x[i,1] + t3*x[i,2]- y[i]) *x[i,0] for i in range(m)])
    g2 = 1.0/m * sum([(t0 + t1*x[i,0] + t2*x[i,1] + t3*x[i,2]- y[i]) *x[i,1] for i in range(m)])
    g3 = 1.0/m * sum([(t0 + t1*x[i,0] + t2*x[i,1] + t3*x[i,2]- y[i]) *x[i,2] for i in range(m)])

    t0 = t0 - lr * g0
    t1 = t1 - lr * g1
    t2 = t2 - lr * g2
    t3 = t3 - lr * g3

    e = sum([(t0 + t1*x[i,0] + t2*x[i,1] + t3*x[i,2]- y[i])**2 for i in range(m)])

    if np.all(abs(l-e) <= ep):    #完成了聚集
        collect = True

    l = e
    iter += 1

    if iter == maxcycle:          #达到上限次数
        collect = True
```

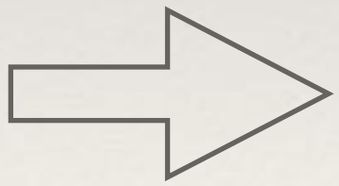
Good enough

```
for iteration in range(iterations):
    hypothesis = np.dot(X, theta)
    loss = hypothesis - y
    gradient = np.dot(X_T, loss) / m
    theta = theta - alpha * gradient
```

Excellent code

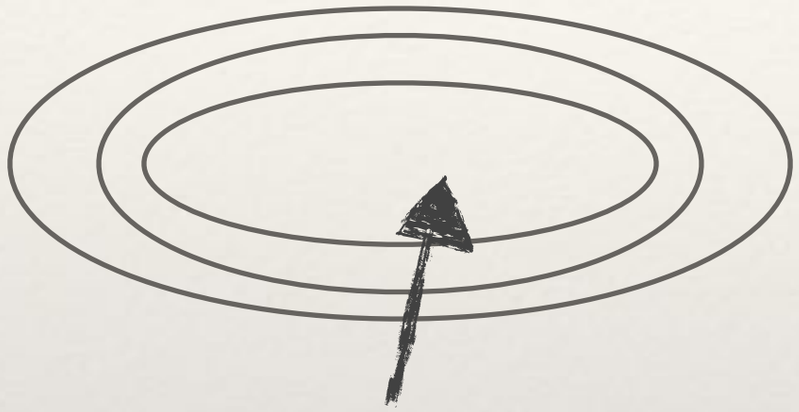
```
for it in iter_li:
    for lr in lr_li:
        for reg in reg_li:
            linear_reg = LinearRegression()
            score = 0.0
            for k in range(num_fold):
                tmp_X_val = X_fold[k]
                tmp_y_val = y_fold[k]
                tmp_X_train = np.concatenate((X_fold[(k+1)%num_fold],X_fold[(k+2)%num_fold],X_fold[(k+3)%num_fold])
                                             ,axis=0)
                tmp_y_train = np.concatenate((y_fold[(k+1)%num_fold],y_fold[(k+2)%num_fold],y_fold[(k+3)%num_fold])
                                             ,axis=0)

                linear_reg.train(tmp_X_train,tmp_y_train,learning_rate=lr,regulization=reg,num_iters=it)
                prices = linear_reg.predict(tmp_X_val)

                score += r2_score(tmp_y_val,prices)

            score /= num_fold
            if(score > best_score):
                best_score = score
                best_linear_reg = linear_reg
```

- object-oriented code
- K Fold Cross-Validation
- Grid-search

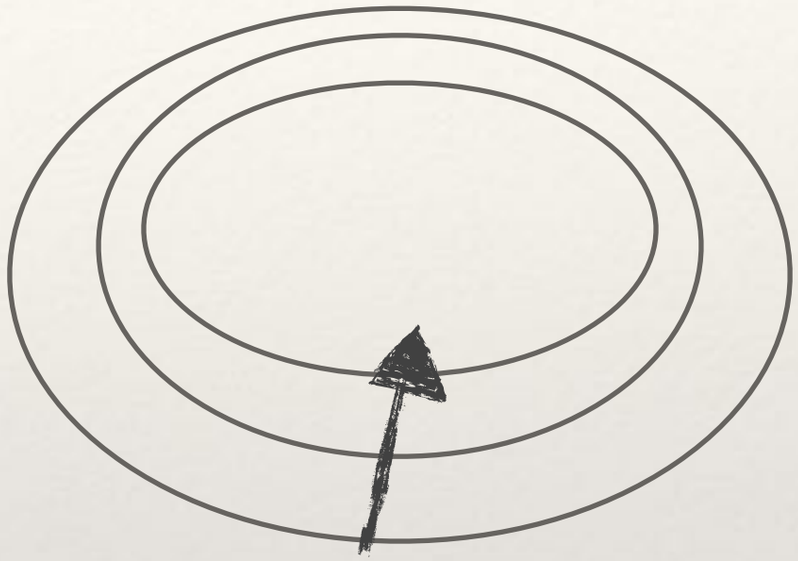


Best Performance

Common Issue: No Feature Scaling



1, 2, 3, …



100, 200, 300, …

Use z Score

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

Common Issue: Poor Code Style

```
X_1 = X_copy[:78]
X_2 = X_copy[78:156]
X_3 = X_copy[156:234]
X_4 = X_copy[234:312]
X_5 = X_copy[312:390]
y_1 = y_copy[:78]
y_2 = y_copy[78:156]
y_3 = y_copy[156:234]
y_4 = y_copy[234:312]
y_5 = y_copy[312:390]
```

```
t0 = t0 - lr * g0
t1 = t1 - lr * g1
t2 = t2 - lr * g2
t3 = t3 - lr * g3
…
return t0,t1,t2,t3
```

Other Issues

- 回答简单，不全面
- 没有仔细阅读问题上下文
- R2 Score
- 使用了额外的库
- 提交无关文件
- 没有按格式命名作业

QA