机器学习第五次实验报告——线性回归

April 23, 2018

1 问题描述

使用 sklearn 中的 dataset 读入数据集, 自行划分训练集和测试集 (测试集占 20% 左右)，对数据集特征进行归一化或标准化处理，用梯度下降算法进行线性回归，并对测试集进行预测，输出均方误差。

2 原理简介

2.1 预测

首先定义如下数学符号

- 特征：x_i
- 特征向量:x, 特征向量由特征组成，$(x_i)^j$ 表示第 i 个特征向量的第 j 个特征。
- 输出向量：y^i，表示第 i 个输入所对应的输出。
- 假设：也成为预测函数，比如线性预测函数为

$$h_\theta(x) = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2 + \theta_3 * x_3 + ... + \theta_n * x_n = \theta^T * x$$

上面的表达式也成为回归方程，θ 为回归系数

2.2 误差评估

为了评估真实值与预测值之间的差异，通过最小均方来描述误差

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

评估误差的函数也成为代价函数 (cost function)

2.3 梯度下降

引入代价函数的目的是解决有手段评估学习正确性问题，下面开始解决当学习效果不佳时，有手段能纠正学习策略的问题。

通常使用梯度下降的方法来不断调节 θ 以使得误差减小:

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

其中 α 为学习率。

梯度方向是函数值下降最为剧烈的方向，沿着 J(θ) 的梯度方向走，就能接近更高的预测精度。学习率 α 标识了延梯度方向下降的速度。

3 解决方案

3.1 数据预处理

首先定义函数 get_Boston_data() 生成数据集并进行预处理。

```
# Load the raw data
boston = load_boston()
data = np.array(boston['data'])
target = np.array(boston['target'])
```

boston 为一个字典变量，键 *data* 和 *target* 对应的值分别为样本数据集的特征向量和目标数据。

获取数据集之后，加快梯度下降的步伐以及增加模型的精度，对样本特征进行归一化处理 (feature scaling)。此处采用 *zscore* 的方法进行归一化，使得每维特征的均值为 0, 标准差为 1，具体做法是每维特征减去均值并除以标准差。

```
mean = np.mean(data,axis = 0)
std = np.std(data, axis = 0)
data = (data - mean)/std
```

为了加入偏执项，在样本数据中加入一维值均为 1 的特征，这为特征不计算入正则损失，也不用进行归一化处理。

```
data = np.column_stack((data,np.ones(num_train)))
```

下面对样本数据进行测试集，训练集以及交叉验证集的划分。此处取全部样本数据的 20% 作为测试集,10% 作为验证集，70% 作为训练集。使用 scikit 包中的函数 train_test_split 进行随机划分。

```
X_train2,X_test,y_train2, y_test = train_test_split(data, target,  test_size = 0.2, random_state = 0)
X_train,X_val,y_train, y_val = train_test_split(X_train2, y_train2,  test_size = 0.1, random_state = 0)
```

对数据进行预处理之后，查看训练集，测试集以及交叉验证集的维度

```
Training set shape: (363, 14)
Test set shape: (102, 14)
Validation set shape: (41, 14)
```

3.2 生成线性回归模型

下面定义类 *LinearRegression* 生成线性回归模型

3.2.1 计算损失及梯度

首先定义函数 $\text{loss}(X, y, reg)$, 在正则化权重为 reg 时，对于数据集 X 和目标值 y, 计算损失和损失相对于参数的梯度。其中 X 的维度为 $N * D$,N 为样本数，D 为特征维度，y 为长度为 N 的列向量。

```
def loss(self, X, y, reg):
        """
        Compute the loss function and its derivative.
        Inputs:
        - X: A numpy array of shape (N, D) containing  N
          data points; each point has dimension D.
        - y: A numpy array of shape (N,) containing targets for the set.
        - reg: (float) regularization strength.

        Returns: A tuple containing:
        - loss as a single float
        - gradient with respect to self.W; an array of the same shape as W
        """
        num_train = X.shape[0]
        pred = X.dot(self.W)
        #the mean square loss
        loss_ms = ((pred - y).T).dot((pred - y)) /num_train/2
        #the regularization loss
        reg_W = self.W[:-1]
        loss_reg = reg*np.sum(reg_W*reg_W)
        #the total loss
        loss = loss_ms + loss_reg
        grad_ms = (X.T).dot(pred-y)/num_train
        grad_reg = 2*reg*self.W
        grad_reg[-1] = 0
        grad = grad_reg + grad_ms
        return loss,grad
```

其中将估计值与真实值做差后点积并除以二倍的样本数得到损失的均方误差部分，将参数出去最后一行按元素平方求和得到损失的正则部分，最后总的损失为这两部分相加。

在求梯度时，对于损失均方误差部分的梯度，根据

$$\frac{\partial}{\partial \theta_j} * J(\theta) = \frac{1}{m} * \sum_{i=1}^{m} (x_i\theta - y_i) * x_i^j$$

设参数梯度向量为 $d\theta$, 将上式转换为矩阵形式为

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

对于正则部分损失的梯度，设为 d'_θ，则有

$$d\theta' = 2 * reg * \theta$$

由于偏置部分不计入正则化损失，还要把最后一维设为 0

$$d\theta'[-1] = 0$$

最后的总梯度为上述两部分之和

$$d\theta = d\theta + d\theta'$$

3.2.2 训练模型

下面定义函数 train$(self, X_train, y_train, X_val, y_val, learning_rate, reg, num_iters)$ 进行模型的训练。其中 X_train, y_train 为测试集的数据和标签，X_val, y_val 为验证集的样本和标签，$learning\_rate$ 为学习率，reg 为正则化权重，num_iters 为迭代次数。

```
def train (self, X_train, y_train,X_val,y_val, learning_rate, reg, num_iters):
    """
    Train this linear classifier using batch gradient descent.

    Inputs:
    - X: A numpy array of shape (N, D) containing training data; there are N
      training samples each of dimension D.
    - y: A numpy array of shape (N,) containing target;
    - learning_rate: (float) learning rate for optimization.
    - reg: (float) regularization strength.
    - num_iters: (integer) number of steps to take when optimizing

    """
    num_train, dim = X_train.shape
    num_val = X_val.shape[0]
    if self.W is None:
      # lazily initialize W
      self.W = 0.001 * np.random.randn(dim)

    # Run batch gradient descent to optimize W
    loss_history = []
    train_rmse_history = []
    val_rmse_history = []
    for it in range(num_iters):
      # evaluate loss and gradient
      loss, grad = self.loss(X_train, y_train, reg)

      loss_history.append(loss)
      # perform parameter update
      # Update the weights using the gradient and the learning rate.          #
      self.W = self.W - grad*learning_rate
      if it % 100 == 0:
            # Check accuracy
            y_diff_train = self.predict(X_train) - y_train
            train_rmse = math.sqrt(np.sum(y_diff_train.T.dot(y_diff_train))/num_train)

            y_diff_val = self.predict(X_val) - y_val
            val_rmse = math.sqrt(np.sum(y_diff_val.T.dot(y_diff_val))/num_val)

            train_rmse_history.append(train_rmse)
            val_rmse_history.append(val_rmse)
            print('iteration %d / %d: loss %f '%(it,num_iters,loss))
            print('training_rmse: %f val_rmse: % f' % (training_rmse, val_rmse ))
```

首先随机初始化参数 W, 循环遍历每一次迭代，调用函数 loss(X_train, y_train, reg) 计算当前参数下的损失和梯度，根据学习率进行参数更新. 每迭代 100 次时，分别在训练集和交叉验证集上用已迭代出的参数计算均方误差评估模型。评估模型时调用的函数 predict(X_train) 的定义如下：

```
    def predict(self, X):

        """
        Use the trained weights of this linear classifier to predict values for
        data points.

        Inputs:
        - X: A numpy array of shape (N, D) containing training data; there are N
          training samples each of dimension D.

        Returns:
        - y_pred: Predicted values for the data in X. y_pred is a 1-dimensional
          array of length N.
        """
        y_pred = np.zeros(X.shape[0])
        y_pred = X.dot(self.W)
        return y_pred
```

X 为待评估的数据集，输出 y 为评估结果。

3.2.3 调节参数

定义线性回归模型后，不断调整学习率以及正则化权重来训练模型，观察模型输出。
例如，在学习率为 0.00001, 正则化权重为 0.2，迭代次数为 10000 次时，训练模型的输出为：

```
iteration 0 / 10000: loss 236.609293 training_rmse: 21.738648 val_rmse: 21.102457
iteration 1000 / 10000: loss 231.596773 training_rmse: 21.504577 val_rmse: 20.856865
iteration 2000 / 10000: loss 226.749096 training_rmse: 21.275644 val_rmse: 20.617119
iteration 3000 / 10000: loss 222.055359 training_rmse: 21.051513 val_rmse: 20.382847
iteration 4000 / 10000: loss 217.505799 training_rmse: 20.831881 val_rmse: 20.153713
iteration 5000 / 10000: loss 213.091662 training_rmse: 20.616480 val_rmse: 19.929411
iteration 6000 / 10000: loss 208.805076 training_rmse: 20.405065 val_rmse: 19.709665
iteration 7000 / 10000: loss 204.638955 training_rmse: 20.197419 val_rmse: 19.494222
iteration 8000 / 10000: loss 200.586897 training_rmse: 19.993344 val_rmse: 19.282854
iteration 9000 / 10000: loss 196.643114 training_rmse: 19.792663 val_rmse: 19.075354
```

下面分别取不同的正则化权重及学习率训练模型，观察模型在交叉验证集上的均方误差：

```
for learning_rate in [1e-5,1e-6,1e-7]:
    for reg in [0.1,0.15,0.2,0.25,0.3]:
        train,val =  classifier.train(X_train, y_train, X_val, y_val,
                    learning_rate,
                    reg,num_iters=10000)
        print('learning rate: %f, reg: %f'%(learning_rate, reg))
        print('train_rmse:%f ,val_rmse:%f '%(train,val))
```

输出结果为

```
learning rate: 0.000010, reg: 0.100000, train_rmse:5.275013 ,val_rmse:3.998296
learning rate: 0.000010, reg: 0.150000, train_rmse:5.187406 ,val_rmse:3.903182
learning rate: 0.000010, reg: 0.200000, train_rmse:5.127140 ,val_rmse:3.835167
learning rate: 0.000010, reg: 0.250000, train_rmse:5.089705 ,val_rmse:3.788492
learning rate: 0.000010, reg: 0.300000, train_rmse:5.072106 ,val_rmse:3.760255
learning rate: 0.000001, reg: 0.100000, train_rmse:5.063487 ,val_rmse:3.750287
learning rate: 0.000001, reg: 0.150000, train_rmse:5.057234 ,val_rmse:3.743602
learning rate: 0.000001, reg: 0.200000, train_rmse:5.052744 ,val_rmse:3.738555
learning rate: 0.000001, reg: 0.250000, train_rmse:5.049950 ,val_rmse:3.735052
learning rate: 0.000001, reg: 0.300000, train_rmse:5.048806 ,val_rmse:3.733039
learning rate: 0.000000, reg: 0.100000, train_rmse:5.048036 ,val_rmse:3.732162
learning rate: 0.000000, reg: 0.150000, train_rmse:5.047437 ,val_rmse:3.731528
learning rate: 0.000000, reg: 0.200000, train_rmse:5.047002 ,val_rmse:3.731042
learning rate: 0.000000, reg: 0.250000, train_rmse:5.046730 ,val_rmse:3.730701
learning rate: 0.000000, reg: 0.300000, train_rmse:5.046620 ,val_rmse:3.730507
```

观察此时的模型训练得到的参数：

```
The value of the parameter is [-0.66648075  0.6464507  -0.56099297  0.65819112 -0.37907314  2.5018936
 -0.21252501 -1.06385585  0.08115488 -0.63869192 -1.56723949  0.66250341
 -2.41121533 21.78050679]
```

用参数估计测试集数据

```
num_test = X_test.shape[0]
y_diff_test = classifier.predict(X_test) - y_test
test_rmse = math.sqrt(np.sum(y_diff_test.T.dot(y_diff_test))/num_test)
```

输出的均方误差为

```
test_rmse: 6.406453
```

4 实验总结

在本次实验中，采用线性回归模型拟合 Boston 房价的数据，首先对数据进行特征归一化处理，使得每一维度的特征的均值为 0, 标准差为 1。加入一维特征为偏执项，值均为 1. 再对数据进行训练集，测试集，交叉验证集的划分。训练模型时，模型的损失函数由两部分组成，第一部分为均方误差损失，即预测值与实际目标值求差后平方加和除以二倍的样本数，第二部分为正则损失，正则损失为了防止模型过于复杂而出现过拟合现象，提升模型的泛化能力。正则损失的计算为参数 (除偏置参数外) 的平方加和。最后总的损失为上述两部分损失之和，正则损失的权重为模型的超参数。训练过程采用梯度下降的方式，计算损失在参数上的梯度并不断更新梯度，以使得目标函数的值最小。更新函数值时更新部分为学习率乘以梯度，梯度决定了参数更新的方向，而学习率决定了更新的幅度，学习率仍是模型的超参数之一。最后通过不断改变模型的超参数并计算在交叉验证集的均方误差来确定学习率以及正则权重，并用训练后的模型在测试集上测试。