数据分析及实践-实验四

PB20000326 徐海阳

Part 1 分类算法实践

1.1 算法主要流程

Step 1. 将lab3中特征工程得到的DIY.csv转换为.npy文件，并进行5折交叉验证所需要的train/test数据集划分

```
data = np.load('./data/DIY.npy')

data_part = cross_validation(data, 5)

for i in range(5):
    print('part: %d' % (i+1))
    train, test = split_train_test(data_part, i)
```

Step 2. 用logistic回归进行二分类。

```
class MLP(object):
    # ...
for i in range(5):
    print('part: %d' % (i+1))
    train, test = split_train_test(data_part, i)
    model = MLP(train, test, train.shape[1]-1, 1, 0.1, 5001)
    model.train_model()
```

1.2 算法关键技术

Step 1: 将DIY.npy数据分为5个part，其中四个concatenate成为train，剩余一个为test。

Step 2: 构造class: MLP，实现sigmoid，sigmoid_derivative（对sigmoid求导），forward（前向传播），judge（judge>0则代表predict结果正确），backward（反向传播），test_model（在test集上计算正确率），train_model （在train集上计算正确率）方法。

1.3 算法实现

关键代码结构如下。完整代码见附录。

```python
import numpy as np

def cross_validation(data, k):
    # split the data into k parts
    # return the k parts
    return data.reshape(k, -1, data.shape[1])

def split_train_test(data_part, k):
    # split the data_part into train and test
    # return the train and test
    test = data_part[k, :, :]
    train = np.delete(data_part, k, axis=0)
    train = np.concatenate(train, axis=0)
    return train, test

class MLP(object):
    def __init__(self, train, test, n_in, n_out, lr, epoch):
        # ...

    def sigmoid(self, x):
        # ...

    def sigmoid_derivative(self, x):
        # ...

    def forward(self, x): # (bsz, n_in)
        # ...

    def judge(self, y_hat, y):
        """
        if (y_hat > 0.5 and y = 1) or (y_hat < 0.5 and y = 0),then return positive: right predict!
        create a function to judge whether the prediction is correct
        """
        # ...

    def backward(self, x, y): # (bsz, n_in) , (bsz, n_out)
        # ...


    def test_model(self, data, label):
        # ...

    def train_model(self):
        # ...

def main():
    data = np.load('./data/DIY.npy')

    data_part = cross_validation(data, 5)

    for i in range(5):
        print('part: %d' % (i+1))
```

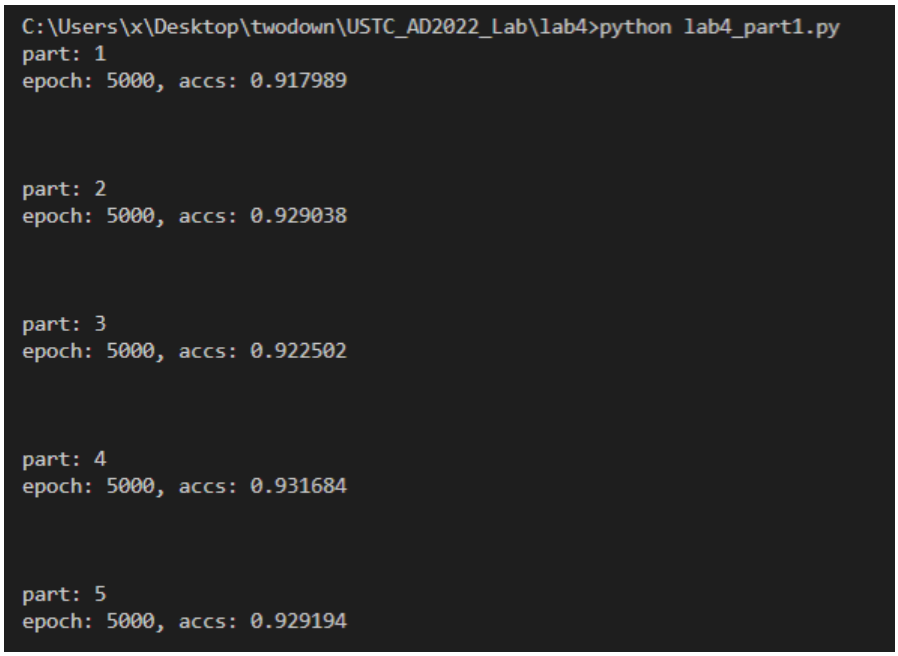
```
        train, test = split_train_test(data_part, i)
        model = MLP(train, test, train.shape[1]-1, 1, 0.1, 5001)
        model.train_model()
        print('\n\n')

if __name__ == '__main__':
    main()
```

1.4 实验记录

（k折交叉验证，4:1比例，共有5折）

在learning_rate=0.1，epoch=5001时的结果如下：



k	ACC
1	0.917989
2	0.929038
3	0.922502
4	0.931684
5	0.929194
平均值	0.926081

Part 2 预测算法实践

1. 尝试了lightgbm库
2. 尝试了sklearn库中的12种回归算法

2.1 lightgbm

附录

part 1完整代码及实现细节

```python
import numpy as np

def cross_validation(data, k):
    # split the data into k parts
    # return the k parts
    return data.reshape(k, -1, data.shape[1])

def split_train_test(data_part, k):
    # split the data_part into train and test
    # return the train and test
    test = data_part[k, :, :]
    train = np.delete(data_part, k, axis=0)
    train = np.concatenate(train, axis=0)
    return train, test

class MLP(object):
    def __init__(self, train, test, n_in, n_out, lr, epoch):
        # self.bsz = train.shape[0]
        # self.train = train
        # self.test = test
        self.n_in = n_in
        self.n_out = n_out
        self.lr = lr
        self.epoch = epoch
        self.train_data = train[:, :-1]
        train_label = train[:, -1]
        self.train_label = train_label.reshape(train_label.shape[0], 1)
        self.test_data = test[:, :-1]
        test_label = test[:, -1]
        self.test_label = test_label.reshape(test_label.shape[0], 1)
        self.w = np.random.randn(self.n_in, self.n_out)
        self.b = np.random.randn(self.n_out)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        return self.sigmoid(x) * (1 - self.sigmoid(x))

    def forward(self, x): # (bsz, n_in)
        o = np.dot(x, self.w) + self.b # (bsz, n_out)
        y_hat = self.sigmoid(o) # (bsz, n_out)
        return o, y_hat

    def judge(self, y_hat, y):
```

```
        """
        if (y_hat > 0.5 and y = 1) or (y_hat < 0.5 and y = 0),then return positive: right predict!
        create a function to judge whether the prediction is correct
        """
        s = (y_hat - 0.5) * (y - 0.5)
        return s

    def backward(self, x, y): # (bsz, n_in) , (bsz, n_out)
        bsz = x.shape[0]
        o, y_hat = self.forward(x) # (bsz, n_out)

        # 链式法则
        d_L_d_y_hat = -y/y_hat + (np.ones_like(y)-y)/(np.ones_like(y)-y_hat) # (bsz, n_out)
        d_y_hat_d_o = self.sigmoid_derivative(y_hat) # (bsz, n_out)
        d_o_d_w = x # (bsz, n_in)
        d_o_d_b = np.ones((bsz, 1)) # (bsz, 1)

        d_L_d_w = np.mean(d_L_d_y_hat * d_y_hat_d_o * d_o_d_w, axis=0) # (n_in,)
        d_L_d_w = d_L_d_w.reshape(self.n_in, 1) # (n_in, 1)
        d_L_d_b = np.mean(d_L_d_y_hat * d_y_hat_d_o * d_o_d_b, axis=0) # (1,)
        self.w = self.w - self.lr * d_L_d_w
        self.b = self.b - self.lr * d_L_d_b


    def test_model(self, data, label):
        # pdb.set_trace()
        o, y_hat = self.forward(data)
        total = data.shape[0]
        """
        correct are those whose prediction is correct, i.e. y_hat > 0.5 and y = 1
        """
        correct = np.sum(self.judge(y_hat, label)>0)
        accs = correct / total
        return accs

    def train_model(self):
        for i in range(self.epoch):
            self.backward(self.train_data, self.train_label)
            if i!=0 and i%5000 == 0:
                accs = self.test_model(self.test_data, self.test_label)
                print('epoch: %d, accs: %f' % (i, accs))

def main():
    data = np.load('./data/DIY.npy')

    data_part = cross_validation(data, 5)

    for i in range(5):
        print('part: %d' % (i+1))
        train, test = split_train_test(data_part, i)
        model = MLP(train, test, train.shape[1]-1, 1, 0.1, 5001)
        model.train_model()
        print('\n\n')

if __name__ == '__main__':
    main()
```