机器学习概论

第一次实验

PB21000302

张成

1. 总述

本次实验中我主要使用了数据的标准化(normalization)，十次十折检验，对缺失数据进行了用平均值填充的操作，同时对学习率(learning rate)，规范(restriction)进行调节，发现对这个数据集没有什么影响。

2. 数据处理

需要注意的是要将 df 中的 Loan_status 单独拿出来作为 y，同时要为 X 添加一个全部是 1 的维度作为截距

2.1. 编码特征（Encode categorical features）

因为这里的几乎每个 feature 都是用字符串来代表的，所以很自然的想到的就是 str.replace，将字符出转换为数字 0 和 1，但是有一点需要注意的是在 Education 中要先替换 Not Graduate 否则变成 Not 1

代码示例：

```
df['Gender']=df['Gender'].str.replace('Male','1')
df['Gender']=df['Gender'].str.replace('Female','0')
```

2.2. 处理缺失值（Deal with NULL rows）

这里我选择的是将它们用平均值替代，因为我认为用一个具体的数值比如-1，2 来代替 NAN 会导致整体平均值的偏移，对最后的标准化（normalization）有不好的影响

代码示例：

```
df[column].fillna(mean_val, inplace=True)
```

2.3. 正则化（Normalization）

这个数据集既有取值 01 的特征，也有取值 1e3 量级的，两级相差太大，需要正则化，用常见的 df_norm=(df-mean)/var 这样不至于特征之间数量级差距太大

代码示例：

```
for column in df_norms:
    #print(df_norm[column])
    mean_val=df_norms[column].mean()
    var_1=df_norms[column].var()
    df_norms[column]=(df_norms[column]-mean_val)/var_1
```

3. 训练算法

本次实验我采用普通的梯度下降法，激活函数为 sigmoid 函数，损失函数采用交叉熵

3.1. Sigmoid

代码示例：

```
def sigmoid(self, x):
    """The logistic sigmoid function"""
    z=1/(1+np.exp(-x))
    return z
```

3.2. 损失函数和梯度

显然这个任务是二分类，通过课程中提到的损失函数（交叉熵）的表达式和简单的求导就可以得到损失函数和这里需要注意的是用矩阵乘法会比用循环快很多。

至于约束，也是表达和梯度都很简单，即为参数θ的二范数的平方，导数即为θ本身

代码示例：

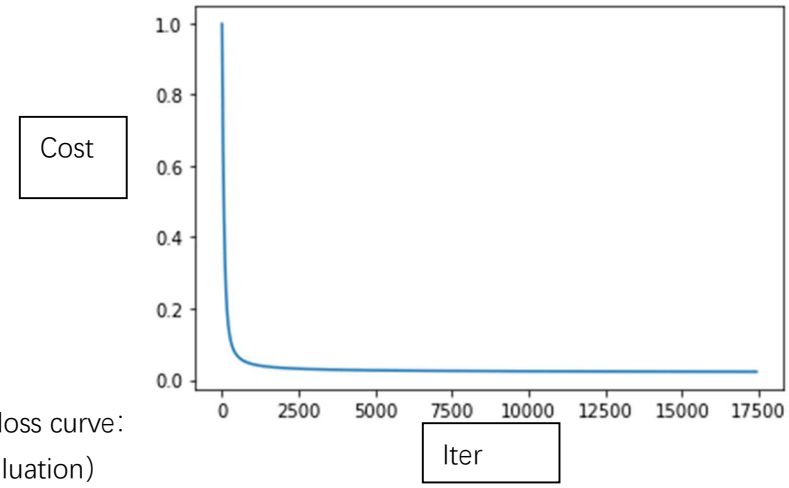
```
def cost_fun(self, theta,X,y,lambda_=0):
    m = y.size    # number of training examples
    n= theta.size
    grad = np.zeros(theta.shape)
    # for i in range(m):
    #       J+=-y[i]*np.log(self.sigmoid(np.dot(X[i],theta)))/m-(1-y[i])*np.log(1-self.sigmoid(np.dot(X[i],theta)))/m
    J=0
    J+=-np.dot(y,np.log(self.sigmoid(np.dot(X.T,theta))))/m-np.dot(1-y,np.log(1-self.sigmoid(np.dot(X.T,theta))))/m
    grad+=np.dot(X,(self.sigmoid(np.dot(X.T,theta))-y))/m
    J+=np.dot(theta,theta)*lambda_/2
    grad+=theta*lambda_
    return J, grad
```

3.3. 梯度下降

根据训练结果来看，iter=1e5 参数已经完全收敛，所以我将 baseline 中的 1e7 改编成 1e5

代码示例:

```
def fit(self, X, y, lr=0.01, tol=1e-7, max_iter=1e5,lambda_=0):
    theta=np.random.randn(12)#一共 11 个 feature 加上截距一共 12 个
    m = y.size    # number of training examples
    n= theta.size
    J_his=[ ]
    for i in range(int(max_iter)):
        theta_0=theta.copy()
        J,grad=self.cost_fun(theta_0,X,y,lambda_)
        theta-=lr*grad
        J_his.append(J)
        if (abs(J-J_his[i-1])<tol )&(i>=1000):
            return theta, J_his
    return theta,J_his
```



得到 baseline loss curve：

4. 评估（evaluation）

本次实验我对学习率 lr，约束 lambda_分别做改变，同时使用十次十折实验对模型整体性能做评估，结果如下

4.1. Lr 变化

对不同 lr，训练结果完全一致，结果如下：

lr= 0.001 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

lr= 0.003 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

lr= 0.01 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

lr= 0.03 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

lr= 0.1 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

lr= 0.3 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

是在情理之中的，因为算法收敛的很快，学习率变化没有什么本质区别

4.2. Lambda_变化

对不同 lambda，训练结果完全一致，结果如下：

lambda= 0 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

lambda= 1 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

lambda= 3 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

lambda= 10 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

lambda= 30 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

lambda= 0.1 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

lambda= 0.3 accuracy= 0.9666666666666667 recall= 1.0 precision= 0.8181818181818182

4.3. 十次十折测验

为了检验本次实验模型的准确性，我采用十次十折检验，检验准确率，recall，precision 和 F1

为了实现十次十折测验，每次要对已经处理好的 df 做打乱，因为数据共 614 个，抽取 0-60，61-121…作为测验组，剩余作为训练组，进行实验

代码示例：

```
df=df.sample(frac=1)
#打乱顺序
for i in range(10):
    X_temp_test=df.iloc[i*61:(i+1)*61]
    X_temp_train=pd.concat([df.iloc[0:i*61],df.iloc[(i+1)*61:]],axis=0)
#分割数据集
```

随后对每次产生的十个模型的预测结果做平均得到模型的平均表现，重复十次得到更可信的结果，发现训练算法得到的模型稳定在如下水平：

id= 0 acc= 0.9475409836065575 pr= 0.7291666666666667 re= 1.0 F1= 0.8433734939759037

id= 1 acc= 0.9491803278688525 pr= 0.7186077811077811 re= 1.0 F1= 0.8362673426796433

id= 2 acc= 0.9475409836065574 pr= 0.7252308802308802 re= 1.0 F1= 0.840734870377263

id= 3 acc= 0.9475409836065575 pr= 0.6938702963702964 re= 1.0 F1= 0.8192720515344697

id= 4 acc= 0.9475409836065574 pr= 0.7153769841269841 re= 1.0 F1= 0.8340755306228674

id= 5 acc= 0.9475409836065574 pr= 0.7206603119103119 re= 1.0 F1= 0.8376555290105113

id= 6 acc= 0.9491803278688524 pr= 0.7437268613739201 re= 1.0 F1= 0.8530313753243689

id= 7 acc= 0.9475409836065574 pr= 0.7119390331890332 re= 1.0 F1= 0.8317340972859523

id= 8 acc= 0.9475409836065575 pr= 0.7162824675324677 re= 1.0 F1= 0.8346906538785318

id= 9 acc= 0.9475409836065575 pr= 0.7124229691876751 re= 1.0 F1= 0.8320642528237382

acc_ave,pr_ave,re_ave 分别为 0.94754098, 0.71242297, 1. F1_ave 为 0.8362899197513249 表现符合预期

5. 总结与展望

本次实验应用了课上讲的逻辑回归和检验方法，得到了一个表现较好的模型，

但是本次实验检验 lr 和 lambda 对算法的影响中没有体现出这两个参数的作用，非常可惜，并且本次实验 recall 一直保持 1 而 precision 却才 0.7 到 0.8 并不是很让人满意，可能是因为正负样本数量不一样导致的，事实上正样例为 192 个，负样例为 422 个并不算相差悬殊，因而可能有其他原因。