LB5 机器学习概论

PB19151769　马宇骁

目录

1 实验要求

- 数据预处理：需要注意，提供数据包含大量冗余随机特征、outlier 数据以及 Null 数据，你需要综合运用所学的知识进行数据降维、降噪、补缺、特征提取、编码以及必要的其他数据预处理工作。
- 数据划分：你需要将所提供的 train 数据集按照所学的方法拆分成训练集以及测试集。
- 模型训练：你需要分别使用本课程所学习的线性回归模型、决策树模型、神经网络模型、支持向量机以及 XGBoost 等分类模型来完成标签预测任务。
- 模型验证：你需要将 test_feature.csv 的数据输入到一个你认为性能最佳的模型中，然后仿照 train_label.csv 的文件格式生成对应标签数据文件，命名为 test_label.csv，并将它包含在你所提交的压缩包中。
- 实验分析：你需要仔细撰写实验报告以及相关分析。

2 实验原理

2.1 多分类逻辑回归

在多类别逻辑回归中，因变量是根据一系列自变量（就是我们所说的特征、观测变量）来预测得到的。具体来说，就是通过将自变量和相应参数进行线性组合之后，使用某种概率模型来计算预测因变量中得到某个结果的概率，而自变量对应的参数是通过训练数据计算得到的，有时我们将这些参数成为回归系数。

在二分类逻辑回归的基础上扩展，softmax 回归输出的 K 种类别的概率。模型公式如下：

$$h_\theta\left(x^{(i)}\right)=\begin{bmatrix} p\left(y^{(i)}=1 \mid x^{(i)};\theta\right) \\ p\left(y^{(i)}=2 \mid x^{(i)};\theta\right) \\ \vdots \\ p\left(y^{(i)}=k \mid x^{(i)};\theta\right) \end{bmatrix}=\frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}}\begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

参数 θ 是一个矩阵，矩阵的每一行可以看做是一个类别所对应分类器的参数，总共有 k 行，输出的 K 个数就表示该类别的概率，总和为 1。这样，softmax 回归模型对于一个测试样本，可以得到多个类别对应的概率值，模型选取概率最高的类别作为最终判定结果。

2.2 多分类决策树

划分数据集可以选择很多特征，那么关键问题在于现在使用哪个特征能得到最好的划分结果，因此需要对特征进行评估！根据该特征进行数据集划分会得到很多分支，如果某个分支下的数据属于同一类型，则已经正确分类，如果不属于同一类型，继续选取此时的最好特征继续划分！最后：所有具有相同类型的数据均在同一个数据子集内。

决策树构造过程总结为：

1. 评估最好特征
2. 划分分支
3. 对分支继续评估最好特征 + 划分分支

2.3 神经网络

主要使用 sklearn 库中的 MLPClassifier 直接进行模型构建。以下是其的几个重要参数说明：

- hidden_layer_sizes : 元组形式, 长度 n_layers-2, 默认 (100,), 第 i 元素表示第 i 个神经元的个数
- activation: ‘identity’, ‘logistic’, ‘tanh’, ‘relu’, 默认”relu”
- solver:‘lbfgs’, ‘sgd’, ‘adam’, default ‘adam’. sgd：随机梯度下降; lbfgs：quasi-Newton 方法的优化器; adam：Kingma, Diederik, and Jimmy Ba 提出的机遇随机梯度的优化器
- alpha:float, 可选的，默认 0.0001, 正则化项参数
- learning_rate: ‘constant’，‘invscaling’, ‘adaptive’, 默认 ‘constant’，用于权重更新，只有当 solver 为’ sgd ‘时使用
- max_iter: int，可选，默认 200，最大迭代次数。
- tol：float, 可选，默认 1e-4，优化的容忍度

2.4 支持向量机

支持向量机（support vector machines, SVM）是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM 还包括核技巧，这使它成为实质上的非线性分类器。SVM 的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM 的的学习算法就是求解凸二次规划的最优化算法。

与实验一相同（在预测的时候多分类训练和预测是利用 k(k-1)/2 个 1v1 来实现）

2.5 XGBoost

Boosting 方法，通过将多个弱学习器集成起来形成一个强学习器。XGBoost(eXtreme Gradient Boosting) 是 Boosting 算法中的一种，是一种提升树模型，将很多树的模型集成起来。其以正则化提升（Regularized Boosting）技术而闻名，通过代价函数里加入正则项，控制模型的复杂度，防止过拟合。可以实现并行处理，相比 GBM 有了很大的速度提升。

3 实验实现

3.1 数据预处理

由于训练数据集总共有 10000 条，且有 120 个 feature，其中存在空值和异常数据，因此考虑对原始数据集进行预处理。

3.1.1 空值填充

按总行数来看，空值由于占比过高（45.14%），因此不直接去除，采用上一行的值填充。

```
data.fillna(method='pad', axis=0)
```

3.1.2 异常值

通过绘制'feature_116' 的箱线图（图 1）发现 异常值确实存在。因此，对于过大以及过小值

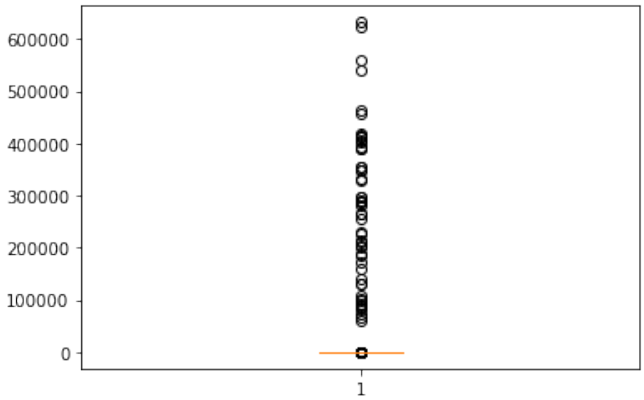

图 1: 116 异常值

做如下处理：

```
# 如果出现异常数据就用5%和95%的分位数取取边界值
for obj in datanames:
    # 肉眼观察最大值都大于0
    if (max(data1[obj])>=0 and np.percentile(data1[obj], 95)>=0) and max(data1[obj]) >
                                                        np.percentile(data1[obj], 95)*
                                                        100:
        data1[obj][data1[obj]>np.percentile(data1[obj], 95)] = np.percentile(data1[obj
                                                            ], 95)
    # 最小值分类处理
    if (min(data1[obj])>=0 and np.percentile(data1[obj], 5)>=0) and np.percentile(
                                                        data1[obj], 5)-min(data1[obj]) >
                                                        np.percentile(data1[obj], 95)/100
                                                        :
        data1[obj][data1[obj]<np.percentile(data1[obj], 5)] = np.percentile(data1[obj]
                                                            , 5)
```

```
    if (min(data1[obj])<0 and np.percentile(data1[obj], 5)>=0) and np.percentile(data1
                                                    [obj], 5)+min(data1[obj]) < min(
                                                    data1[obj])/100:
        data1[obj][data1[obj]<np.percentile(data1[obj], 5)] = np.percentile(data1[obj]
                                                        , 5)
    if (min(data1[obj])<0 and np.percentile(data1[obj], 5)<0) and np.abs(np.percentile
                                                    (data1[obj], 5)-min(data1[obj]))
                                                    > np.abs(min(data1[obj])/100):
        data1[obj][data1[obj]<np.percentile(data1[obj], 5)] = np.percentile(data1[obj]
                                                        , 5)
```

3.1.3 去噪以及降维

- 小波变换去噪

将信号经小波分解后小波系数较大，噪声的小波系数较小，并且噪声的小波系数要小于信号的小波系数，通过选取一个合适的阀值，大于阀值的小波系数被认为是有信号产生的，应予以保留，小于阀值的则认为是噪声产生的，置为零从而达到去噪的目的。

选用 Daubechies8 小波，将信号进行小波分解，再将噪声滤波，最后将信号进行小波重构。

```
## 小波变换去噪
import pywt

w = pywt.Wavelet('db8')  # 选用Daubechies8小波
maxlev = pywt.dwt_max_level(len(data1), w.dec_len)
threshold = 0.2  # Threshold for filtering
for obj in datanames:
    lis = list(data1[obj])
    coeffs = pywt.wavedec(lis, 'db8', level=maxlev)  # 将信号进行小波分解
    for i in range(1, len(coeffs)):
        coeffs[i] = pywt.threshold(coeffs[i], threshold*max(coeffs[i]))  # 将噪声滤波
    datarec = pywt.waverec(coeffs, 'db8')  # 将信号进行小波重构
    data1[obj] = datarec
```

- 奇异值分解降维

若要对行数据进行压缩，我们从矩阵 A 的奇异值分解式子 $A = U\Sigma V^T$ 入手。将等式两边同时乘以左奇异矩阵的转置矩阵 U^T，得到 $U^T A = U^T U\Sigma v^T = \Sigma V^T$. 左侧表达

式 U^TA 表示把矩阵 A 的 n 个 m 维列向量并排放置：

$$[u_1,u_2,u_3,\dots u_m]^T A = \begin{bmatrix} u_1^T \\ u_2^T \\ u_3^T \\ \cdots \\ u_m^T \end{bmatrix} [\mathrm{col}\, A_1, \mathrm{col}\, A_2,\ \mathrm{col}\ A_3,\ \mathrm{clo}\ A_n]$$

$$= \begin{bmatrix} u_1^T \mathrm{col}\, A_1 & u_1^T\ \mathrm{col}\ A_2 & u_1^T \mathrm{col}\, A_3 & \dots & u_1^T col A_n \\ u_2^T \mathrm{col}\, A_1 & u_2^T \mathrm{col}\, A_2 & u_2^T \mathrm{col}\, A_3 & \dots & u_2^T col A_n \\ u_3^T \mathrm{col}\, A_1 & u_3^T \mathrm{col}\, A_2 & u_3^T \mathrm{col}\, A_3 & \dots & u_3^T \mathrm{col}\, A_n \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ u_m^T \mathrm{col}\, A_1 & u_m^T \mathrm{col}\, A_2 & u_m^T \mathrm{col}\, A_3 & \dots & u_m^T \mathrm{col}\, A_n \end{bmatrix}$$

此时，可以把每一列看作一个样本，各行是样本的不同特征，各行之间彼此无关，此时可以选择最大的 k 个奇异值对应的 k 个标准正交向量，形成行压缩矩阵

$$U_{k\times m}^T = \begin{bmatrix} u_1^T \\ u_2^T \\ u_3^T \\ \cdots \\ u_k^T \end{bmatrix}$$

通过式子

$$U_{k\times m}^T \mathrm{col}\, A_i = \begin{bmatrix} u_1^T \mathrm{col}\, A_i \\ u_2^T \mathrm{col}\, A_i \\ u_3^T \mathrm{col}\, A_i \\ \cdots \\ u_k^T \mathrm{col}\, A_i \end{bmatrix}$$

实现了列向量从 m 维降低到 k 维，完成主成分提取。

对实验数据，经过 SVD 提取，其中奇异值序列归一之后如图 2. 因此，大概可以选择保留的奇异值阶数 11，进行数据重构，将数据集最后转换成 10000*11 的维数。

3.2 实验调试

3.2.1 多分类逻辑回归

规定当相邻两次迭代的差异小于 1e-3，或者超过最大迭代次数 100 时停止拟合，最终拟合过程如下：

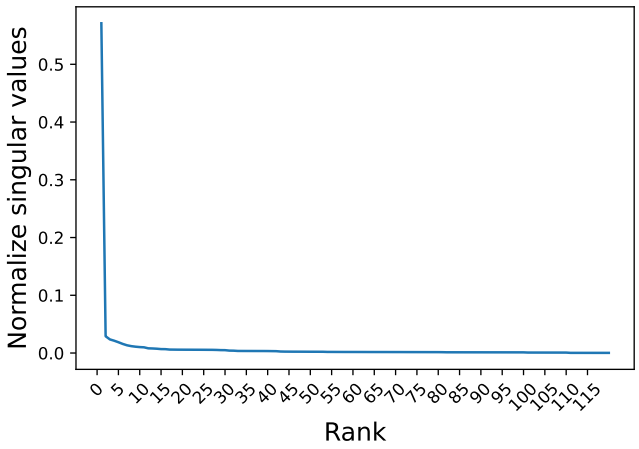

图 2: 奇异值序列

```
Processing:  82%|       | 82/100 [06:40<01:28,  4.94s/it]
一共用时：488.5729990005493s
```

预测准确率为：0.249

3.2.2 决策树

最大深度设置为 6，自己撰写构建决策树的拟合过程如下（此结果还是 120 列时候的情况（因为虽然大幅度降维之后会减少运行时间，但不想等了））：

```
Processing: 100%|      | 120/120 [53:52<00:00, 26.94s/it]
Processing: 100%|      | 120/120 [02:07<00:00,  1.06s/it]
Processing: 100%|      | 120/120 [00:05<00:00, 23.31it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 489.80it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 1084.13it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 1410.85it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 1026.40it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 3115.89it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 2307.73it/s]
Processing: 100%|      | 120/120 [00:04<00:00, 28.22it/s]
Processing: 100%|      | 120/120 [00:03<00:00, 35.15it/s]
Processing: 100%|      | 120/120 [00:02<00:00, 42.05it/s]
```

```
Processing: 100%|      | 120/120 [00:00<00:00, 1587.78it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 845.01it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 1263.17it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 3000.07it/s]
Processing: 100%|      | 120/120 [01:21<00:00,  1.48it/s]
Processing: 100%|      | 120/120 [00:05<00:00, 21.35it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 810.63it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 827.77it/s]
Processing: 100%|      | 120/120 [00:04<00:00, 25.14it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 294.90it/s]
Processing: 100%|      | 120/120 [00:03<00:00, 37.53it/s]
Processing: 100%|      | 120/120 [00:41<00:00,  2.92it/s]
Processing: 100%|      | 120/120 [00:03<00:00, 31.50it/s]
...
Processing: 100%|      | 120/120 [00:00<00:00, 3438.54it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 4287.67it/s]
Processing: 100%|      | 120/120 [00:01<00:00, 87.72it/s]
Processing: 100%|      | 120/120 [00:01<00:00, 94.32it/s]
Processing: 100%|      | 120/120 [00:00<00:00, 4283.07it/s]
一共用时：9171.37307024002s
```

预测准确率为：0.252

3.2.3 神经网络

做如下参数设置：

```
netmodel = MPC(verbose=False,solver='sgd', alpha=1e-5, activation='tanh',
                                                hidden_layer_sizes=(11,32,64,32,4,2),
                                                 max_iter=5000, tol=1e-5,
                                                learning_rate='adaptive')
```

tanh 的激活函数，6 层神经网络的拟合信息如下：一共用时：5.553001403808594s

```
print(netmodel.n_iter_)
print(netmodel.loss_)
print(netmodel.score(np.array(X_test), np.array(y_test)))
print(netmodel.out_activation_)
```

```
118
1.3862243682913666
0.245
softmax
```

预测准确率为：0.245

3.2.4 支持向量机

将自己写的 MultiSVM 参数设置为:max_iter = 100, epsilon=1e-5, kernel_type='linear', C=1.0

对于此数据集，共有 6 个 1v1 的 svm 二分类器，对于同一条数据所有判断为一个的类进行统计，最多的作为预测值（相同时取第一个（np.argmax 的逻辑））。拟合过程如下：

```
Processing: 100%|      | 100/100 [01:08<00:00,  1.46it/s]
Processing: 100%|      | 100/100 [01:08<00:00,  1.46it/s]
Processing: 100%|      | 100/100 [01:05<00:00,  1.52it/s]
Processing: 100%|      | 100/100 [01:06<00:00,  1.50it/s]
分类器：:  67%|       | 4/6 [04:29<02:14, 67.13s/it]
一共用时：403.68699979782104s
```

预测准确率为：0.262

3.2.5 XGBoost

做如下参数设置：

```
params = {
  'booster':'gbtree',
  'objective':'multi:softmax',   # 多分类问题
  'gamma':0.1,    # 用于控制是否后剪枝的参数，越大越保守，一般0.1 0.2的样子
  'max_depth':10,  # 构建树的深度，越大越容易过拟合
  'lambda':2,  # 控制模型复杂度的权重值的L2 正则化项参数，参数越大，模型越不容易过拟合
  'subsample':0.7, # 随机采样训练样本
  'silent':0,  # 设置成1 则没有运行信息输入，最好是设置成0
  'eta':0.001,  # 如同学习率
  'seed':1000,
  'nthread':7,  #CPU线程数
  #'eval_metric':'auc'
}
```

预测准确率为：0.255

3.3 预测结果

最终，综合比较下来，决定使用我的 MultiSVM 作为对预测集的模型。将预测结果附在同一文件夹中。