实验报告

16337341 朱志儒

一、 环境说明

系统：windows 10 64 位

处理器：Intel® Core™ i5-6300HQ CPU @ 2.30GHz 2.30GHz

内存：8.00GB

GPU：NVIDIA GeForce GTX 950M

语言：Python

调用的库：pandas-0.24.2 xgboost-0.82 numpy-1.15.4 sklearn-0.20.3

二、 流程

1. 数据处理方法

训练集中的数据存在空缺值，数据类型有 float 型、int 型和字符串类型，所以需要对这些数据进行预处理。读入训练集后，将整列为 NaN 的列删除；对于数值类型的列，将该列的平均值填入该列中的空缺位置；对于非数值类型的列，将该列中出现次数最多的值填入列中的空缺位置，再使用 sklearn.preprocessing.LabelEncoder 将该列的数据全部转为数值类型。填补所有的空缺值后，再将训练集中的所有数据 min-max 标准化。由于整个训练集数据只有 200 多维，计算的时间成本不会太高，为了保留原始数据中的各种特征，我没有选择降维。

2. 选择模型

一开始我就想使用线性 SVM，所以我使用 sklearn 库中的 LinearSVC 分类算法，但实际预测结果并不是特别理想，提交后得分只有 0.86323。LinearSVC 的预测输出是 0、1 标签，

不是输出属于 0 标签的概率和属于 1 标签的概率, 而实验的预测结果打分函数是计算 AUC, 所以最后结果输出的是属于正标签的概率的话，可以获得更高的 AUC 值，这一点也说明 LinearSVC 并不适合本次实验。

当然，我也尝试过使用 sklearn 库中的 SVC 分类算法，参数 kernel 设置为 linear，使用线性核；参数 probability 设为 True，最终的预测结果采用概率估计；对于参数 C，遍历[1, 2]中的值，使用训练数据训练模型，进行 5 折交叉验证，最后得到最优值为 1.5，提交最后的预测结果得分为 0.94539，这个结果还是不太令人满意。

上述的 SVM 模型与距离相关，对训练集数据十分敏感，如果数据预处理不当的话，分类的效果可能会很差，所以我尝试使用 xgboost 模型。初次使用时，将参数设为默认值，预测结果就能够取得较好的成绩，提交后得分为 0.97699，这比使用 SVC 分类算法有较大的提升。于是，便确定使用 xgboost 模型，接下来就是调参使得模型达到最优。

3. 调参方法

当我使用 sklearn 库中的 SVC 分类算法时，由于使用线性核函数，所以只有参数 C 可以调节，参数 C 越大，表示对错误分类的惩罚越大，C 设置过大可能会导致过拟合。我使用网格搜索，让 C 在[1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2]中取值，每取一个值训练模型后都要进行 5 折交叉验证，设置评分方法为“roc_auc”，运行结束后，最优的 C 值为 1.5，对应的得分为 0.94478，将其对应的测试集预测结果提交后得到的分数为 0.94539。

当我使用 xgboost 模型时，该模型中可调节的参数如下。同样，进行网格搜索时都要经过 5 折交叉验证，评分方法设置为“roc_auc”。

objective：定义需要被最小化的损失函。本次实验是二分类问题，所以这个参数的取值为“binary:logistic”。

n_estimators：弱学习器的最大迭代次数。或者说最大的弱学习器的个数。n_estimators

太小会出现欠拟合，n_estimators 太大计算量会显著增大，但 n_estimators 增大到一定程度后模型只有细微提升，所以 n_estimators 需选择一个合适的值。我让 n_estimators 在[600, 800, 1000, 1200, 1400]中取值, 最后的输出结果显示最佳取值为 1400, 最佳得分为 0.97847。我觉得 n_estimators 取 1400 已经足够大了，如果再提高对整个模型也没有大的提升，所以将参数 n_estimators 的值定为 1400。

min_child_weight：决定最小叶子节点样本权重和。这个参数用于避免过拟合。取值较大时，可避免模型学习到局部的特殊样例。但取值过高时，会导致模型欠拟合。

max_depth：树的最大深度。max_depth 越大，模型能学到更局部更具体的样本。我将 min_child_weight 和 max_depth 放在一起调节, min_child_weight 的取值范围为[1, 2, 3, 4, 5], max_depth 的取值范围为[6, 7, 8, 9, 10]，进行网格搜索，最后的输出结果显示 min_child_weight 的最佳取值为 1, max_depth 的最佳取值为 9, 对应的最佳得分为 0.98151。

gamma：节点分裂所需的最小损失函数下降值。gamma 越大，模型越保守。将 gamma 的取值范围设为[0.1, 0.2, 0.3, 0.4, 0.5]，进行网格搜索，最后的输出结果显示 gamma 的最佳取值为 0.2，对应的最佳得分为 0.98190。

subsample：控制每棵树随机采样的比例。这个值越小，模型会更为保守，可避免过拟合，但值过小，可能会导致欠拟合。

colsample_bytree：控制每棵树随机采样的列数的占比，每一列是一个特征。作用与 subsample 相似。我将 subsample 和 colsample_bytree 放在一起调节, 设置 colsample_bytree 的取值范围为[0.5, 0.6, 0.7, 0.8, 0.9, 1.0]，subsample 的取值范围为[0.5, 0.6, 0.7, 0.8, 0.9, 1.0]，进行网格搜索，最后的输出结果显示 subsample 和 colsample_bytree 的最佳取值均为 0.7，对应的最佳得分为 0.98234。

reg_lambda：L2 正则化项的权重。这个值越大就越可以惩罚树的复杂度。设置

reg_lambda 的取值范围为[0.5, 1, 1.5, 2, 3]，进行网格搜索，最后的输出结果显示 reg_lambda 的最佳取值为 1，对应的最佳得分为 0.98342。

scale_pos_weight：如果仅仅关注预测问题的排序或者 AUC 指标，则可以调节此参数，但想得到真正的预测概率，则不能够通过此参数来平衡样本。设置 scale_pos_weight 的取值范围为[2, 4, 6, 8, 10]，进行网格搜索，最后的输出结果显示 scale_pos_weight 的最佳取值为 6，对应的最佳得分为 0.98358。

max_delta_step：决定最小叶子节点样本权重和。当它的值较大时，可以避免模型学习到局部的特殊样本。但是如果这个值过高，会导致欠拟合。设置 max_delta_step 的取值范围为[0.5, 1, 1,5, 2, 3],进行网格搜索，最后的输出结果显示 max_delta_step 的最佳取值为 1，对应的最佳得分为 0.98458。

learning_rate：学习率。梯度提升中的系数，值越小，下降的越慢但越精确。设置 learning_rate 的取值范围为[0.01, 0.02, 0.05,0.1, 0.15, 1],进行网格搜索，最后的输出结果显示 learning_rate 的最佳取值为 0.05，对应的最佳得分为 0.98558。

将上述所有参数设置为最佳取值后，训练模型，预测测试集数据，将最后的结果提交后得分为 0.98298。

4. 防止过拟合

xgboost 在代价函数里加入了正则项，用于控制模型的复杂度，参数 reg_lambda 就是 L2 正则的惩罚系数，在之前的参数调节过程中，经过网格搜索可以确定其最佳取值为 1，这可以防止模型过拟合。

本次实验类别分布极不平衡，容易出现过拟合，调节参数 max_delta_step 使 xgboost 在其更新过程中更加保守，防止模型过拟合。

在调参过程中，将参数 subsample 设置为 0.7，也就是说 xgboost 从整个样本集合中随

机抽取 70%的子样本建立树模型，这也可防止模型过拟合。同样，通过网格搜索确定参数 min_child_weight 的值，可以防止 xgboost 模型过拟合。

三、 模块内容

1. 数据处理

```
# 将整列为 NaN 的列删除
x.dropna(axis=1, how='all', inplace=True)
for col in x.columns:
    if x[col].dtype != 'object':
        # 将数值类型的列中平均值填入该列中的空缺位置
        x[col].fillna(x[col].mean(), inplace=True)
    else:
        # 将非数值类型的列中出现次数最多的值填入列中的空缺位置
        x[col].fillna(x[col].mode()[0], inplace=True)
        # 将该列的数据全部转为数值类型
        lbl = preprocessing.LabelEncoder()
        x[col] = lbl.fit_transform(list(x[col].values))
# 填补所有的空缺值后，再将训练集中的所有数据 min-max 标准化
x = x.apply(lambda x: (x - np.min(x)) / (np.max(x)-np.min(x)))
```

2. 选择模型

LinearSVC 模型:

```
# LinearSVM 模型，参数 C 为 1.5
classfilter = LinearSVC(C=1.5, dual=False)
# 使用训练集数据训练模型
classfilter.fit(x, y.values.ravel())
# 预测测试集数据的标签
pred = classfilter.predict(test)
```

SVC 模型：

```
# SVC 模型，采用线性核函数，参数 C 为 1.5
classfilter = SVC(kernel='linear', C=1.5, probability=True)
# 使用训练集数据训练模型
classfilter.fit(x, y.values.ravel())
# 预测测试集数据标签的概率
pred = classfilter.predict_proba(test)
```

xgboost 模型：

```
# xgboost 模型，选择最优参数建立模型
classfilter = xgb.XGBClassifier(max_depth=9,learning_rate=0.05,
                                objective='binary:logistic', n_jobs=-1,
                                min_child_weight=1, gamma=0.2,
                                scale_pos_weight=6, reg_lambda=1,
                                subsample=0.7,n_estimators=1400,
                                colsample_bytree=0.7, max_delta_step=1)
# 使用训练集数据训练模型
classfilter.fit(x, y.values.ravel(), verbose=True)
# 预测测试集数据标签的概率
pred = classfilter.predict_proba(test)
```

3. 调参方法

网格搜索参数 n_estimators：

```
# 搜索的参数取值
cv_params = {'n_estimators': [600, 800, 1000, 1200, 1400]}
# 其他参数
other_params = {'max_depth': 9, 'learning_rate': 0.05,
                'gamma': 0.2, 'reg_lambda': 1,
                'scale_pos_weight': 6, 'subsample': 0.7,
                'min_child_weight': 1, 'colsample_bytree':
                  0.7, 'objective': 'binary:logistic',
                'n_jobs': -1, 'silent': 0}
```

```
# 建立模型
model = xgb.XGBClassifier(**other_params)
# 进行网格搜索
optimized = GridSearchCV(estimator=model, param_grid=cv_params,
scoring='roc_auc', cv=5, verbose=True, n_jobs=-1)
# 使用训练集数据训练模型
optimized.fit(x, y.values.ravel())
# 获得最佳参数取值，以及对应得分
print('参数的最佳取值：{0}'.format(optimized.best_params_))
print('最佳模型得分:{0}'.format(optimized.best_score_))
```

网格搜索参数 min_child_weight 和 max_depth：

```
# 搜索的参数取值
cv_params = {'min_child_weight': [1, 2, 3, 4, 5],
             'max_depth': [6, 7, 8, 9 , 10]}
# 其他参数
other_params = {'max_depth': 9, 'learning_rate': 0.05,
                'gamma': 0.2, 'reg_lambda': 1,
                'scale_pos_weight': 6, 'subsample': 0.7,
                'min_child_weight': 1, 'colsample_bytree':
                0.7, 'objective': 'binary:logistic',
                'n_jobs': -1, 'silent': 0}
# 建立模型
model = xgb.XGBClassifier(**other_params)
# 进行网格搜索
optimized = GridSearchCV(estimator=model, param_grid=cv_params,
scoring='roc_auc', cv=5, verbose=True, n_jobs=-1)
# 使用训练集数据训练模型
optimized.fit(x, y.values.ravel())
# 获得最佳参数取值，以及对应得分
print('参数的最佳取值：{0}'.format(optimized.best_params_))
print('最佳模型得分:{0}'.format(optimized.best_score_))
```

网格搜索 gamma、subsample 等参数与上述的过程相似就不再赘述。

四、 模型的原理

1. 支持向量机：

支持向量机（SVM）是一种二分类模型，它最基本的想法就是在基于训练集的样本空间

中找到一个超平面，将不同类别的样本分开，这个超平面是几何间距最大的分离超平面。

假设训练集数据集$\mathrm{T}=\{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$，其中$\mathrm{x_i}\in R^n$，$\mathrm{y_i}\in\{1,0\}$，1 表示正例，0 表示负例。假设训练集数据是线性可分的，定义超平面与样本点$(\mathrm{x_i},y_i)$的几何间隔$\gamma_\mathrm{i}=y_i\left(\frac{w}{\|w\|}\cdot x_i+\frac{b}{\|w\|}\right)$，SVM 模型欲找到具有最大间隔的划分超平面，即

$$\max_{w,b}\frac{2}{\|w\|}$$
$$\text{s.t.}\quad \mathrm{y_i}(w^Tx_i+b)\geq 1,\mathrm{i}=1,2,3,\ldots,\mathrm{m}$$

可重写为

$$\min_{w,b}\frac{1}{2}\|w\|^2$$
$$\text{s.t.}\quad \mathrm{y_i}(w^Tx_i+b)\geq 1,\mathrm{i}=1,2,3,\ldots,\mathrm{m}$$

上式即为支持向量机的基本型。这是一个含有不等式约束的凸二次规划问题，可以使用拉格朗日乘子法得到其对偶问题：

$$\max_a\sum_{i=1}^m a_i-\frac{1}{2}\sum_{i=1}^m\sum_{j=1}^m a_ia_jy_iy_jx_i^Tx_j$$
$$\text{s.t.}\quad \sum_{i=1}^m a_iy_i=0$$
$$a_i\geq 0,\mathrm{i}=1,2,..\mathrm{m}$$

解出最优解

$$\mathrm{a'}=(\mathrm{a_1'},a_2',\ldots,a_m')^T$$

计算

$$\mathrm{w}=\sum_{i=1}^m a_i'y_ix_i$$

选择$\mathrm{a'}$的一个分量$\mathrm{a_j'}$满足条件$0<\mathrm{a_j'}<C$，计算

$$\mathrm{b'}=\mathrm{y_j}-\sum_{i=1}^m a_i'y_i(x_i\cdot x_j)$$

求分离超平面

$$\mathrm{w'}\cdot\mathrm{x}+\mathrm{b'}=0$$

分类决策函数：

$$f(x) = sign(w' \cdot x + b')$$

训练完成后，大部分的训练样本都不需要保留，最终模型仅与支持向量有关。

2. xgboost：

xgboost 是 Boosting 算法的其中一种，Boosting 算法的工作机制是：先从初始训练集训练出一个基学习器，再根据基学习器的表现对训练样本分布进行调整，使得之前基学习器分类错的训练样本在后续得到更多的关注，然后基于调整后的样本分布训练下一个基学习器。如此重复进行，直至基学习器的数量达到指定的值 T，最终将这 T 个基学习器加权结合。

xgboost 的思想是不断地添加树，不断地进行特征分裂来生长一棵树，每次添加一个数，就是学习一个新的函数去拟合上次预测的残差。当训练完成得到 k 颗数，要预测一个样本的分数，就是根据该样本的特征，在每棵树中回落到对应的一个叶子节点，每个叶子节点对应一个分数，最后只需将每棵树对应的分数求和可得到该样本的预测值。

xgboost 相比于 SVM 的优点：

本次实验的训练样本比较多，SVM 的运行效率不是很高，而 xgboost 的弱分类器的构造十分简单，运行效率较高。

对于本次实验的数据，xgboost 的分类精度比 SVM 较高。

xgboost 在应对异常数据时，与 SVM 相比有更强的稳健性和可扩展性，也不易于发生过拟合。