**实验报告**

**16337341 朱志儒**

1. **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

1. **流程**
2. **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。

1. **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。

1. **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。

1. **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。

1. **模块内容**
2. **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)))

1. **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)

1. **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。

1. **模型的原理**
2. **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，其中，，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的几何间隔，SVM模型欲找到具有最大间隔的划分超平面，即

可重写为

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：

解出最优解

计算

选择的一个分量满足条件，计算

求分离超平面

分类决策函数：

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。

1. **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。