exp 5 - 大数据风控实验

数据预处理

1. 将数据中含有60个及以上的非Nan值的样本留下，其余去除。
2. 将标签值为Nan的样本删去。
3. 其余使用均值填充。
4. 使用sklearn中的train_test_split函数将训练集与测试集分为0.8/0.2份。

LR模型

不做归一化

进行LR模型训练与预测，调用库函数*roc_auc_score()*计算auc值：

```
clf = LogisticRegression(random_state=0).fit(x_train, y_train.values.ravel())
prey = clf.predict(x_test)
```

```
roc_auc_score(prey, y_test)
```

```
0.688429225331711
```

做归一化

使用StandardScaler库函数进行归一化：

```
# standard
x_train_arr = np.array(x_train)
x_test_arr = np.array(x_test)
s1 = StandardScaler()
s2 = StandardScaler()
s1.fit(x_train_arr)
x_train_stand = s1.transform(x_train_arr)
s2.fit(x_test_arr)
x_test_stand = s2.transform(x_test_arr)
```

进行LR模型训练与预测，计算auc值：

```
clf = LogisticRegression(random_state=0).fit(x_train_stand, y_train.values.ravel())
prey = clf.predict(x_test_stand)
```

```
print("auc_score: %s" % roc_auc_score(prey, y_test))
```

```
auc_score: 0.6435890923921836
```

发现auc值与没有归一化并没有很大差异。

WOE编码

将data中的X13特征即“网龄”进行分箱并计算woe值，由于网龄为离散值，且范围不大，故可以直接将其作为箱的界对woe值进行计算：

```
woe = {}
woel = []
good_t = sum(data['Y']==1)
bad_t = sum(data['Y']==0)
```

```
for i, v in data.X13.items():
    good = 0
    bad = 0
    if woe.get(v) != None:
        woel.append(woe[v])
    else:
        for j, vj in data.X13.items():
            if vj == v:
                if data.Y[j] == 1:
                    good += 1
                else:
                    bad += 1
        woel.append(math.log((good/good_t+0.1)/(bad/bad_t+0.1)))
        woe[v] = math.log((good/good_t+0.1)/(bad/bad_t+0.1))
```

然后与data合并成为一个新的特征并进行LR模型训练与预测，计算auc值：

```
clf = LogisticRegression(random_state=0).fit(x_train_stand, y_train.values.ravel())
prey = clf.predict(x_test_stand)
```

```
print("auc_score: %s" % roc_auc_score(prey, y_test))
```

```
auc_score: 0.6529384464391272
```

auc值比上述“0.6435”略大，可能是由于该特征计算得到的woe值代表性不强，对回归预测没有太大效果。

特征交叉

特征交叉规则：将两个特征的特征值分别相加。

特征交叉：'X23'(用户违约次数),'X34'(联系人中黑名单人数计数)：将该用户的违约次数与其联系人汇总黑名单人数计数，刻画该用户的形象。

'X36'(用户信用等级),'X37'(用户信用分数)：将用户信用的两项指标进行交叉，使特征更明显。

进行LR模型训练与预测，计算auc值：

```
clf = LogisticRegression(random_state=0).fit(x_train_stand, y_train.values.ravel())
prey = clf.predict(x_test_stand)
```

```
print("auc_score: %s" % roc_auc_score(prey, y_test))
```

```
auc_score: 0.6636655882598781
```

相比上述“0.6529”略大，在进行第一次交叉时auc值为“0.6562”，第二次为“0.6636”，即多次交叉特征可能对回归模型有提升，同时选择不同的特征进行交叉、效果也不同。

GBDT模型

不做归一化

```
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,max_depth=1, random_st
prey = clf.predict(x_test)
```

```
print("auc_score: %s" % roc_auc_score(prey, y_test))
```

```
auc_score: 0.6561652109954529
```

做归一化

使用StandardScaler库函数进行归一化：

```
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,max_depth=1, random_st
prey = clf.predict(x_test)
```

```
print("auc_score: %s" % roc_auc_score(prey, y_test))
```

```
auc_score: 0.6751411461082123
```

发现auc值有所提升。

WOE编码

将data中的X13特征即“网龄”进行分箱并计算woe值，由于网龄为离散值，且范围不大，故可以直接将其作为箱的界对woe值进行计算,然后与data合并成为一个新的特征并进行LR模型训练与预测，计算auc值：

```
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,max_depth=1, random_state=0).fit(x_train, y_train.values.rav
prey = clf.predict(x_test)
```

```
print("auc_score: %s" % roc_auc_score(prey, y_test))
```

```
auc_score: 0.6637180576365195
```

发现auc值下降，说明woe编码的操作出现问题，或者特征选取的较差。

特征交叉

特征交叉规则：将两个特征的特征值分别相加。

特征交叉：'X23'(用户违约次数),'X34'(联系人中黑名单人数计数)：将该用户的违约次数与其联系人汇总黑名单人数计数，刻画该用户的形象。

'X36'(用户信用等级),'X37'(用户信用分数)：将用户信用的两项指标进行交叉，使特征更明显。

进行LR模型训练与预测，计算auc值：

```
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,max_depth=1, random_state=
prey = clf.predict(x_test)
```

```
print("auc_score: %s" % roc_auc_score(prey, y_test))
```

```
auc_score: 0.6723336543657679
```

发现auc值略有提高，但无大差异。

lightgbm模型

按上述特征构造进行该模型测试。

利用GridSearchCV进行参数优化【参数‘max_depth’, ‘num_leaves’】

```
参数的最佳取值:{'max_depth': 6, 'num_leaves': 30}
最佳模型得分: 0.7494869565071922
```

再进行其他参数优化【参数‘learning_rate’, ‘min_child_samples’】

```
参数的最佳取值:{'learning_rate': 0.1, 'min_child_samples': 25}
最佳模型得分: 0.7514022180210994
```

使用最佳参数，不进行归一化，得到auc值为：

```
prey = gsearch.predict(x_test)
```

```
print("auc_score: %s" % roc_auc_score(prey, y_test))
```

```
auc_score: 0.6631646375701262
```

进行归一化，得到auc值：

```
prey = gsearch.predict(x_test)
```

```
print("auc_score: %s" % roc_auc_score(prey, y_test))
```

```
auc_score: 0.6756097473041328
```

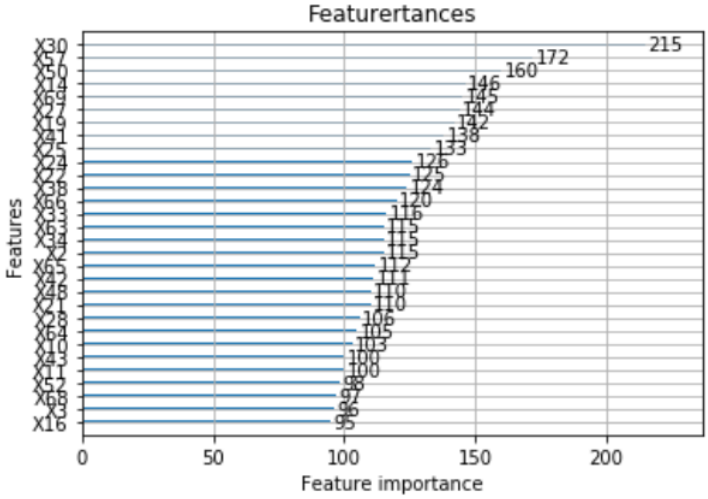
可以得出，归一化后有所提升，但是对提升的0.01是由波动造成还是归一化造成不能明确区分。

特征重要性

在lgb模型中调用matplotlib.pyplot进行特征重要性的分析与展示：

```
lgb_parameters = {
    'objective':'binary',
    'is_unbalance': True,
    'metric': 'auc',
    'max_depth': 6,
    'num_leaves': 30,
    'learning_rate': 0.1,
    'feature_fraction': 0.7,
    'min_child_samples': 25,
    'min_child_weight': 0.001,
    'bagging_fraction': 1,
    'bagging_freq': 2,
    'reg_alpha': 0.001,
    'reg_lambda': 8,
    'cat_smooth': 0,
    'num_iterations': 200
}
lgb_train = lgb.Dataset(x_train, y_train)
model = lgb.train(lgb_parameters, lgb_train)
plt.figure(figsize=(12,6))
lgb.plot_importance(model, max_num_features=30)
plt.title("Featurertances")
plt.show()
```

特征重要性可视化图如下：



使用自定义auc值计算

设计auc值计算函数（按照概率模型计算）：

```
def calcAUC(labels, probs):
    N = 0
    P = 0
    neg_prob = []
    pos_prob = []
    for _,i in enumerate(labels):
        if (i == 1):
            P += 1
            pos_prob.append(probs[_])
        else:
            N += 1
            neg_prob.append(probs[_])
    number = 0
    for pos in pos_prob:
        for neg in neg_prob:
            if (pos > neg):
                number += 1
            elif (pos == neg):
                number += 0.5
    return number / (N * P)
```

LR模型：

按特征处理后结果计算：

```
clf = LogisticRegression(random_state=0).fit(x_train_stand, y_train.values.ravel())
prey = clf.predict(x_test_stand)
```

```
print("auc_score: %s" % calcAUC(y_test.Y,prey))
```

```
auc_score: 0.6148069111308395
```

得到auc值略小于调用库函数结果。

GBDT模型：

按特征处理后结果计算：

```
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,max_depth=1, random_state=
prey = clf.predict(x_test)
```

```
print("auc_score: %s" % calcAUC(y_test.Y,prey))
```

```
auc_score: 0.6344319579100536
```

得到auc值略低于调用库函数结果。

lightgbm模型：

按特征处理后结果计算：

```
prey = gsearch.predict(x_test)
```

```
print("auc_score: %s" % calcAUC(y_test.Y,prey))
```

```
auc_score: 0.6812870753574978
```

得到auc值略高于调用库函数结果。

对于以上结果，LR模型与GBDT模型自定义auc值计算略低，而对于lightgbm自定义auc值计算略高。

讨论

对于是否应该归一化数据，在LR模型中，可以看到，数据归一化对效果的提升不大，而GBDT和lightgbm模型在数据归一化后有微小的提升。

在查阅资料后得知，对于概率模型，特征没必要做归一化，因为概率模型与变量的值没有关系，而与变量的分布以及变量的概率有关，所以一般不需要归一化。而对于采用基于梯度更新的学习方法(如线性回归、逻辑回归、支持向量机、神经网络等)对模型求解过程中，未归一化的数值特征在学习时，梯度下降较为抖动，模型难以收敛；而归一化之后的数值特征可以使梯度下降较为稳定，进而减少梯度下降次数，易于收敛。

即如果算法或相关优化函数受特征量纲的影响较大，则需要做归一化，而对于只与变量分布于概率有关的算法则不需要做归一化。由此可得，在实验中得到的实验结果或许是由于某些扰动造成的，我们对LR模型使用归一化与不使用归一化所需的训练时间做记录：

不使用归一化：

```
start_time = time.time()
clf = LogisticRegression(random_state=0).fit(x_train, y_train.values.ravel())
end_time = time.time()
prey = clf.predict(x_test)
print("time: %s" % (end_time-start_time))
```

```
time: 0.688910961151123
```

使用归一化：

```
start_time = time.time()
clf = LogisticRegression(random_state=0).fit(x_train_stand, y_train.values.ravel())
end_time = time.time()
prey = clf.predict(x_test_stand)
print("time: %s" % (end_time-start_time))
```

```
time: 0.5023133754730225
```

发现时间缩短了0.1 s，推测应该是归一化使得LR训练速度加快，符合上述理论。

> 在所有实验结果中，区别只有0.01或者0.1数量级,由于缺乏经验，对这样的数值区别不能确定是正常波动还是真实差异。但是在使用相同方法重复做实验得到的结果的差距往往小于不同方法间的结果差异，故在实验中按真实差异得结论。