

稳定度模型分析报告

介应奇

November 3, 2023

Contents

1	摘要	1
2	拆分后加权平均模型	1
2.1	数据准备与读取	2
2.2	数据处理	2
2.2.1	数据读取	3
2.2.2	数据预处理	4
2.3	子模型建立	5
2.3.1	离网模型获取	7
2.3.2	携转模型获取	7
2.3.3	降档模型获取	8
2.3.4	使用子模型综合预测结果	8
2.3.5	将概率值转换为是否离网的标称值	9
2.4	评估结果	10

3 不做区别的一个模型	11
3.1 数据预处理	11
3.2 训练模型	11
3.3 评估结果	12
4 对比分析与总结	13
4.1 对比分析	13
4.2 总结	14
5 鸣谢	14

1 摘要

根据要求，本报告完成下列三个任务：

- 1 读懂现有的代码里提供的模型，使用三个子模型分别预测然后综合预测结果给出是否离网的预测
- 2 写一个不做子模型区分的模型直接给出是否离网的预测
- 3 对比这两个模型的结果，并做出一定的解释

不包含对业务背景の説明。

2 拆分后加权平均模型

这部分模型的实现由老师和助教提供，下面我将结合提供的代码对这部分模型进行简要分析。

2.1 数据准备与读取

数据准备 该案例提供的数据文件为 csv 格式，一共有 4 个文件，其中 colnamepre.csv 和 colname.csv 文件中分别写了预测和训练时使用的数据属性，其中第一列为属性名称序列，第二列为数据序列；pre_test 文件里是测试时使用的具体数据，train-test 文件里是训练用的具体数据。在这个实验中我们只是用了其中的 colname.csv 和 train-test 这两个文件，因为我们要评估融合的模型和拆分离网类型的模型效果哪个好，因此需要有带标签的测试集，而 colname.csv 文件中只有属性名称序列，而 pre_test 文件中没有 flag 数据，因此我们不使用。

数据集理解 每个模型都将使用全部的这些数据特征，只是每个模型具有的标签值在 flag 属性中用不同的数字做了区分：0-正常未流失 1-离网模型 2-携转模型 3-降档模型，对每一种模型我们使用其中的一个流失标签类别对模型进行训练，由于不同的用户离网都有不同的标签，所以最终这三个模型相当于每一个面模型使用了其中的一部分数据进行了训练，在是自己的类型和正常之间给出一个二分类模型，最终得到在三个不同标签属性下训练出来的模型，然后最终的结果用这三个模型给出的结果做加权平均给出。

2.2 数据处理

由于原始提供的数据中测试集和训练集的属性值是基于真实业务的，因此测试集中没有 flag 标签，我们无法做模型效果的对比，因此我将原本的 pre_test 文件替换为了从 train-test 文件中切分出来的部分数据，这样就可以对比模型效果了。在测试集中，我将 flag 值中非零的值都用 1 来表示，统一用 1 表示离网，0 表示稳定，好对两个模型进行统一的评级。

下面是我切分数据的代码：

```

1 # 将所有数据都获取
2 allX=alldata[(alldata['flag']==0)|(alldata['flag']==1)|(alldata['flag']==2)|(alldata['flag']==3)][usecol_list]
3 ally=alldata[(alldata['flag']==0)|(alldata['flag']==1)|(alldata['flag']==2)|(alldata['flag']==3)]['flag']
4
5 allX_train, allX_test, ally_train, ally_test = train_test_split(allX, ally, test_size=0.2, random_state=7)

```

Figure 1: 数据读取部分代码

```

1 i = 0
2 for d in ally_test:
3     if d != 0:
4         ally_test[i] = 1
5     i += 1

```

Figure 2: 测试集数据转换部分代码

2.2.1 数据读取

下面代码完成了数据的读取操作，过程是，先读入列属性文件，然后根据这个信息将属性值划分到 3 个列表中，区分属性的类型，这三个属性分别是要使用的属性，也就是出了 none 类型和标签，然后又区分了数值型数据和标称类数据，方便后面数据预处理。最后读入具体的数据到


allcol_list 中，之后使用属性分组来获取其中对应的数据即可。

```
1 #写入数据列名称 设置字段类型
2 col_file = pd.read_csv("colname.csv", sep=',', header=None, index_col=None) #118个特征
3 allcol_list=[]
4 numcol_list=[]
5 catcol_list=[]
6 usecol_list=[]
7
8 for i in range(len(col_file)):
9     allcol_list.append(col_file[0][i])
10    if col_file[1][i] != 'none' and col_file[1][i] != 'flag': #num cat
11        usecol_list.append(col_file[0][i])
12    if col_file[1][i] == 'num':
13        numcol_list.append(col_file[0][i])
14    if col_file[1][i] == 'cat':
15        catcol_list.append(col_file[0][i])
16
17 #读取数据
18 alldata = pd.read_csv("train_test", sep=',', names=allcol_list, index_col=None)
19 s=alldata.copy()
20 alldata.describe()
21 print('数据读取完成')
```

Figure 3: 数据读取部分代码

2.2.2 数据预处理

下面代码完成了缺失值填充和类别数据的序列化



```
1 #缺失值用零填充
2 alldata = alldata.replace({'\N': np.nan})
3 alldata = alldata.fillna(value=0)
4
5 #cat编码序列化
6 for i in catcol_list:
7     alldata[i] = alldata[i].astype('category')
8     alldata[i] = alldata[i].cat.codes
```

Figure 4: 数据预处理部分代码

2.3 子模型建立


在这里我们每个模型都是用相同的模型 GradientBoostClassifier，只是数据不同，获得到了 X 和 Y 的做输入和输出数据后的操作都进行如下的公共操作：

```
1 #切分训练集
2 X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=7)
3
4 print('预处理完成')
5 #模型训练gbdt
6 model = GradientBoostingClassifier(learning_rate=0.4, n_estimators=50, subsample=0.75,
7                                     max_depth=3, max_features='log2',
7                                     random_state=49)
8 model.fit(X_train, y_train)
9 print('训练完成')
10
11
12 ### 保存模型到文件
13 import pickle
14 with open('model/model_3.pkl', mode='wb') as file:
15     pickle.dump(model, file)
```

Figure 5: 模型训练代码

下面根据 flag 标签获取相应的数据进行训练即可


2.3.1 离网模型获取



```
1 X=alldata[(alldata['flag']==0)|(alldata['flag']==1)][usecol_list]
2 Y=alldata[(alldata['flag']==0)|(alldata['flag']==1)]['flag']
```

Figure 6: 获取离网模型数据

2.3.2 携转模型获取



```
1 X=alldata[(alldata['flag']==0)|(alldata['flag']==2)][usecol_list]
2 Y=alldata[(alldata['flag']==0)|(alldata['flag']==2)]['flag']
```

Figure 7: 获取携转模型数据

2.3.3 降档模型获取

```
1 X=alldata[(alldata['flag']==0)|(alldata['flag']==3)][usecol_list]
2 Y=alldata[(alldata['flag']==0)|(alldata['flag']==3)]['flag']
```

Figure 8: 获取降档模型数据

2.3.4 使用子模型综合预测结果

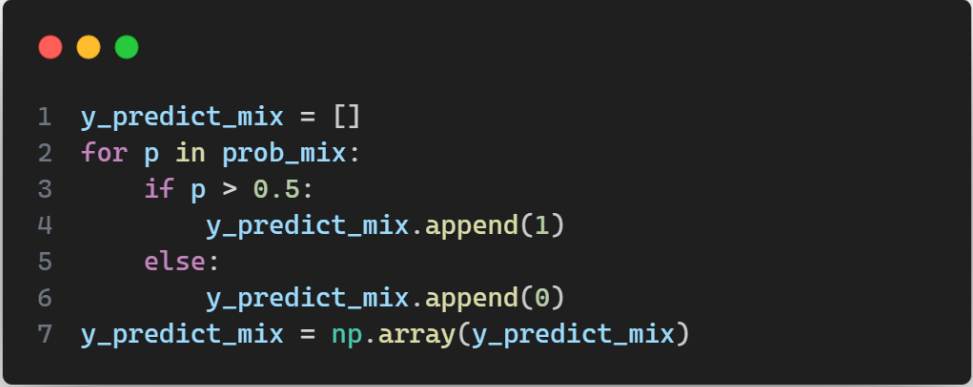
预测出每个模型给出的离网概率，并将他们的概率值取平均值，因为在我们假设训练集和测试集是独立同分布的，因此加权的过程我们可以使用子模型中离网类型的数量占比作为加权因子。

```
1 y_predict_proba1 = model1.predict_proba(allX_test)
2 y_predict_proba2 = model2.predict_proba(allX_test)
3 y_predict_proba3 = model3.predict_proba(allX_test)
4 prob_mix = (y_predict_proba1[:, 1] * len(allX_train[alldata['flag']==1])
5             + y_predict_proba2[:, 1] * len(allX_train[alldata['flag']==2])
6             + y_predict_proba3[:, 1] * len(allX_train[alldata['flag']==3])) / len
             (allX_train[(alldata['flag']==1)|(alldata['flag']==2)|(alldata['flag']==3)])
```

Figure 9: 预测离网概率

2.3.5 将概率值转换为是否离网的标称值

根据加权完的概率结果给出标签值, 当离网概率大于 0.5 时我们将其标为 1, 否则标为 0, 代码如下:



```
1 y_predict_mix = []
2 for p in prob_mix:
3     if p > 0.5:
4         y_predict_mix.append(1)
5     else:
6         y_predict_mix.append(0)
7 y_predict_mix = np.array(y_predict_mix)
```

Figure 10: 标注结果

2.4 评估结果

```
1 print(classification_report(ally_test, y_predict_mix))
2 print(confusion_matrix(ally_test, y_predict_mix))
```

Figure 11: 评估结果代码

	precision	recall	f1-score	support
0	0.57	0.92	0.70	1813
1	0.78	0.30	0.43	1811
accuracy			0.61	3624
macro avg	0.67	0.61	0.57	3624
weighted avg	0.67	0.61	0.57	3624

[[1661	152]
[1266	545]]

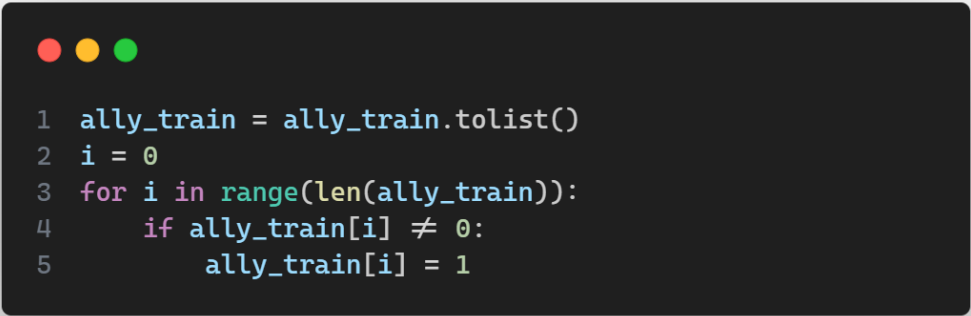
Figure 12: 评估结果

3 不做区别的一个模型

这个模型使用的数据和上面分割的是一致的

3.1 数据预处理

将训练集中的非零 flag 值都设置为 1

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains five lines of Python code:

```
1 ally_train = ally_train.tolist()
2 i = 0
3 for i in range(len(ally_train)):
4     if ally_train[i] != 0:
5         ally_train[i] = 1
```

Figure 13: 标签值更改

3.2 训练模型

使用与子模型相同的算法训练模型

```
1 #模型训练gbdt
2 model = GradientBoostingClassifier(learning_rate=0.4, n_estimators=50,
3                                     subsample=0.75, max_depth=3, max_features='log2',
4                                     random_state=49)
5 model.fit(allX_train, ally_train)
6 print('训练完成')
7
8 ### 保存模型到文件
9 import pickle
10 with open('model/model_only.pkl', mode='wb') as file:
11     pickle.dump(model, file)
```

Figure 14: 训练单个模型

3.3 评估结果

```
1 y_pre = model.predict(allX_test)
2 print(classification_report(ally_test, y_pre))
3 print(confusion_matrix(ally_test, y_pre))
```

Figure 15: 评估结果代码

	precision	recall	f1-score	support
0	0.70	0.67	0.68	1813
1	0.68	0.72	0.70	1811
accuracy			0.69	3624
macro avg	0.69	0.69	0.69	3624
weighted avg	0.69	0.69	0.69	3624
[[1208 605]				
[514 1297]]				

Figure 16: 评估结果

4 对比分析与总结

4.1 对比分析

从两个模型在测试集中的评估结果来看:

- 预测离网的精确率是第一个模型做的更好, $0.78 > 0.68$, 高出了 10%, 我认为这是由于子模型版本的模型在 flag 的多样性标签值中拟合到了更多的信息, 因此精确率更高
- 预测离网的召回率是第二个模型更好 $0.30 < 0.72$, 相差了 42%, 这是由于召回率评估的是模型找到流失用户的能力, 由于子模型拆分的模型将用户的流失单纯的归因为三种, 排除了其他的用户流失的可能性, 因此不如不做区分的融合模型
- 由于召回率相差过大, 且在精确率上第一个模型也没有远超第二个模型, 因此总体上来说, 针对区分用户是否流失这件事情还是第二个模型的效果更由于召回率相差过大, 且在精确率上第一个模型也没有远超第二个模型, 因此总体上来说, 针对区分用户是否流失这件事情还

是第二个模型的效果更好。

4.2 总结

虽然在区分是否离网这件事情上，融合模型比子模型好，但是这并不代表区分了离网类型的模型就没有价值；相反，正因为区分了离网的类型，预测出离网这一事件对于运营商才有了价值，因为，只有知道了用户具体是因为什么原因离网才能采取相应的策略进行挽回，或者针对不同的类型采取不同的策略，而在这个方面可以说融合模型几乎没有什么应用价值，但这也提醒了我们，如果使用分类别的离网预测模型，那么我们就需要注意离网与否的预测可能会有很大的疏漏，这也是运营商需要注意的地方，如果想要进行改良的话可以通过增加模型类别的方式进行，也就是提高信息密度的方式。

至此我们分析了使用两种模型的利与弊。

5 鸣谢

感谢袁汉宁老师和助教为本实验报告提供了代码框架，以及加权平均模型部分的大部分代码，以及 PPT 讲解。