课　　程：2019 软件工程综合实训

项　　目：　数据挖掘实训项目

院　　系：数据科学与计算机学院

专　　业：　　软件工程

学生姓名：　　陈明亮

学　　号：　　16340023

授课教师：　　郑子彬

2019 年 07 月 09 日

目 录

一、赛题简述.....3
1.1 赛事简介.....3
1.2 赛题解读与准备.....4
二、赛题分析.....4
2.1 数据探索.....4
2.2 数据预处理.....6
三、赛题解析过程.....7
3.1 特征工程 - 初步处理.....7
3.1.1 处理字符串数据.....7
3.1.2 处理用户安装的 APP 编号序列.....7
3.2 特征工程 - 特征生成.....8
3.2.1 利用 Stacking 思想生成特征.....8
3.2.2 统计用户安装的总 APP 数目生成特征.....9
3.2.3 利用 LDA 模型降维生成特征.....9
3.2.4 利用 RNN 生成特征.....10
3.3 模型选取.....11
3.3.1 单 LR 分类器.....11
3.3.2 lightGBM 模型.....12
四、比赛结果与总结.....13
参考文献.....14
代码展示.....14
提交成绩.....14

一、赛题简述

1.1 赛事简介

本次数据挖掘实训期末比赛，本人选择的是华为 Digix 杯的用户人口属性预测赛题，此赛题通过提供众多用户的基本信息，与对于智能手机的使用情况，要求参赛者建立模型，预测出对应的目标用户的年龄段，作为比赛结果。

下面是赛题官网上的比赛说明[1]：

1. 智能手机已成为人们工作生活中不可或缺的伙伴，当今社会几乎所有“衣食住行”都可以通过一部手机来解决。手机作为一种能够同时提供视频、游戏、音乐、阅读、摄影功能的全功能综合移动载体，正在逐步取代传统媒介（如电视、个人电脑、MP3、照相机等）。在此背景下，如何通过智能化、个性化应用以及精准推送服务来提升用户体验是每个手机厂商孜孜以求的目标，而数据则是实现这一目标的基石。

2. 用户的人口属性（如性别、年龄、常驻地等）数据一方面可以被用于个性化推荐服务，提升用户体验，另一方面可以用于手机用户群画像分析，帮助厂商了解产品的人群定位，优化产品设计。

3. 年龄是用户人口属性的重要维度，本次比赛任务为根据用户的手机使用行为习惯来预估用户所处的年龄段。每个用户（以唯一 ID 标识）对应唯一的年龄段。年龄段有 6 种划分，分别代表不同年龄段，分别为：小于等于 18 岁， 19-23 岁， 24-34 岁， 35-44 岁， 45-54 岁，大于等于 55 岁。参赛同学根据华为提供数据构建预测模型进行年龄段预估，在测试数据集上给出预估结果。

数据挖掘的赛事仍需要关心给出的数据表的字段属性对应的信息，以及具体数据表的作用，在某种程度上可以初步地决定该数据表的重要程度，方便之后对于特征的提取，以及生成。

给定的训练数据表有：

age_train.csv -- 训练集用户编号及其对应的年龄段序号
user_basic_info.csv -- 所有用户的基本信息数据集，包含用户人口属性描述
user_behavior_info.csv -- 所有用户的设备的使用行为汇总数据(30 天内)
user_app_actived.csv -- 所有用户手机上激活的 APP 编号序列(强特征数据)
user_app_usage.csv -- 用户 30 天内每天对单个 app 的打开次数和使用时长
app_info.csv -- 记录每个 app 所属的应用类型

1.2 赛题解读与准备

通过上述的赛题分析与说明，阅读之后我们可以大致概括出本次比赛的性质，以决定之后的数据挖掘模型的建立与实现方向：

1. 本次比赛要求建立多分类模型进行训练预测，输出结果。
2. 需要处理多张数据表，其中包含一张 20 多 G 的用户行为数据表，需要对此表进行特殊对待。
3. 此类用户人口属性预测的题目实际上有过很多先例，可以参考之前易观的比赛[2]获取特征提取的灵感。
4. 需要构建赛题任务图，以方便之后的实验过程。

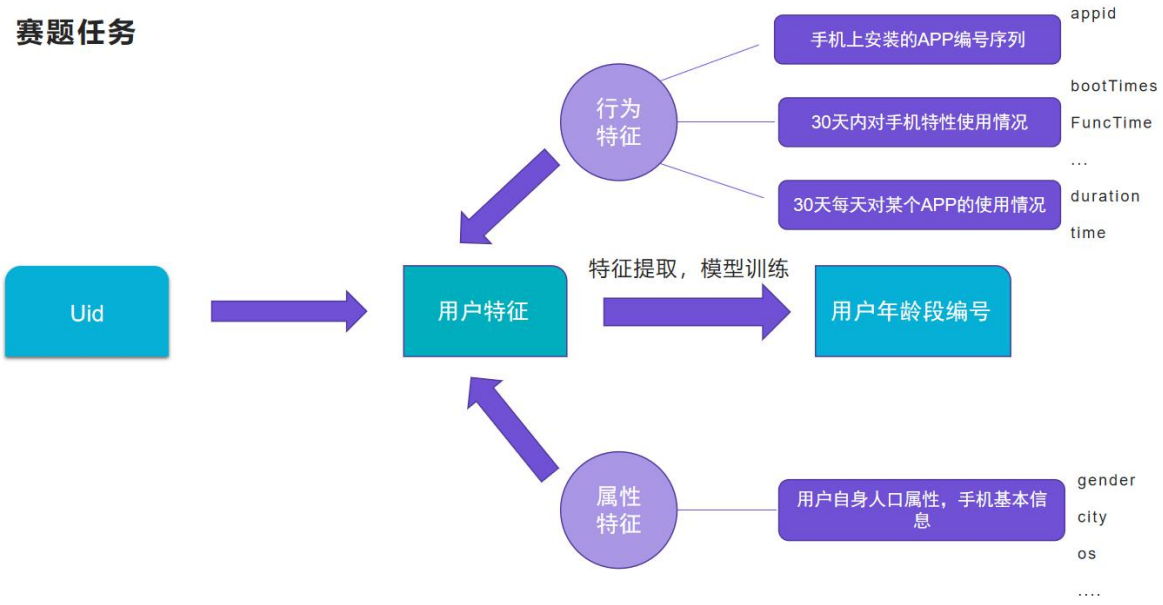


图 1. 赛题任务图

二、赛题分析

2.1 数据探索

关于此次比赛的数据探索，可以分为：数据缺失值情况探索，以及数据量级差距探索，和训练集的年龄段分布情况探索。

对于缺失值的问题向来都是预处理阶段需要首先考虑的，若不及时把输入的训练数据中的缺失值处理掉，那么在之后的模型训练中就会出错。分别对官网上提供的所有数据表进行缺失值统计，发现：用户基本人口属性信息表 user_basic_info.csv 存在缺失值情况，下面是对用户基本人口属性缺失值信息的统计图：

蓝色代表非缺失值，红色代表缺失值，纵轴为统计的出现次数，横轴为对应的字段名称。

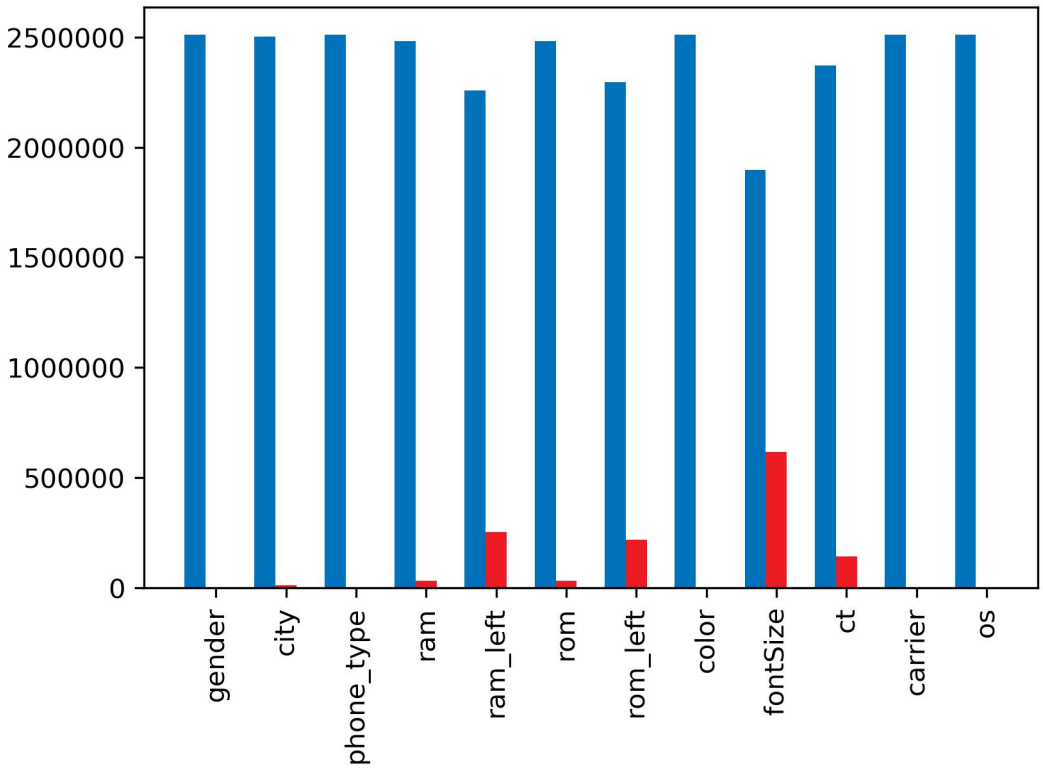


图 2. 用户基本信息表存在的缺失值统计情况

此外，对于用户的设备使用行为表，存在用户在使用不同特性时，特性对应字段下的数据量级差距过大的情况，详情可见下表：

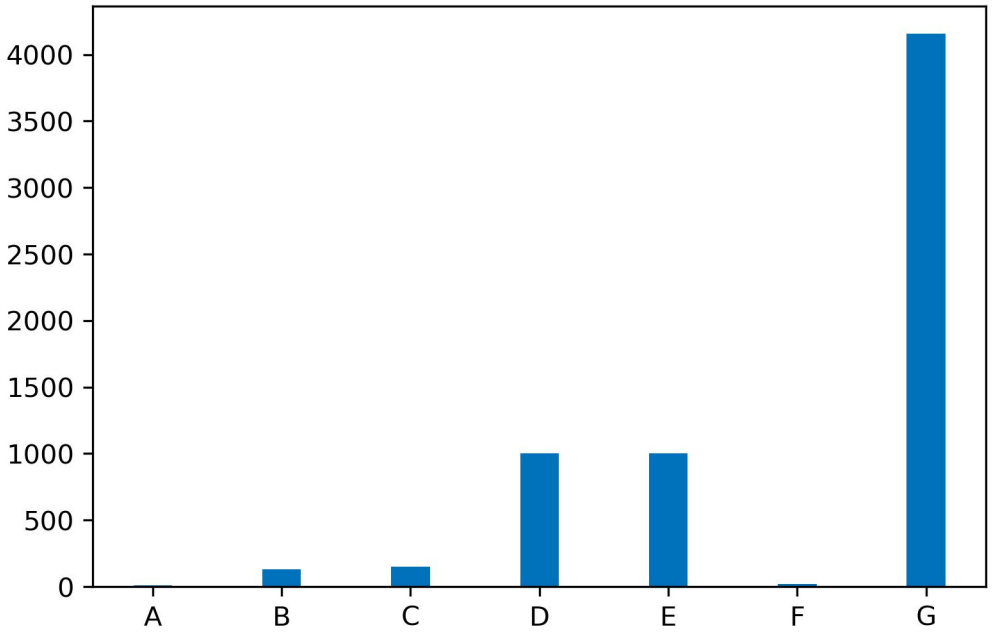


图 3. 用户使用手机行为表的特性平均值统计情况

A 到 G 分别代表用户使用手机的不同特性对应的编号，纵轴为对应特性的数据平均值，此处 G 特性还是做了千分之一处理的，否则图上是无法显示

出其他特性平均值数据的，因为量级差距实在过大，这种情况是会导致模型在处理特征时面对权重分配不均衡的情况的。

在数据探索阶段，我们还需要对作为训练数据的表进行查看，下面是训练表中，用户的年龄段分配情况：

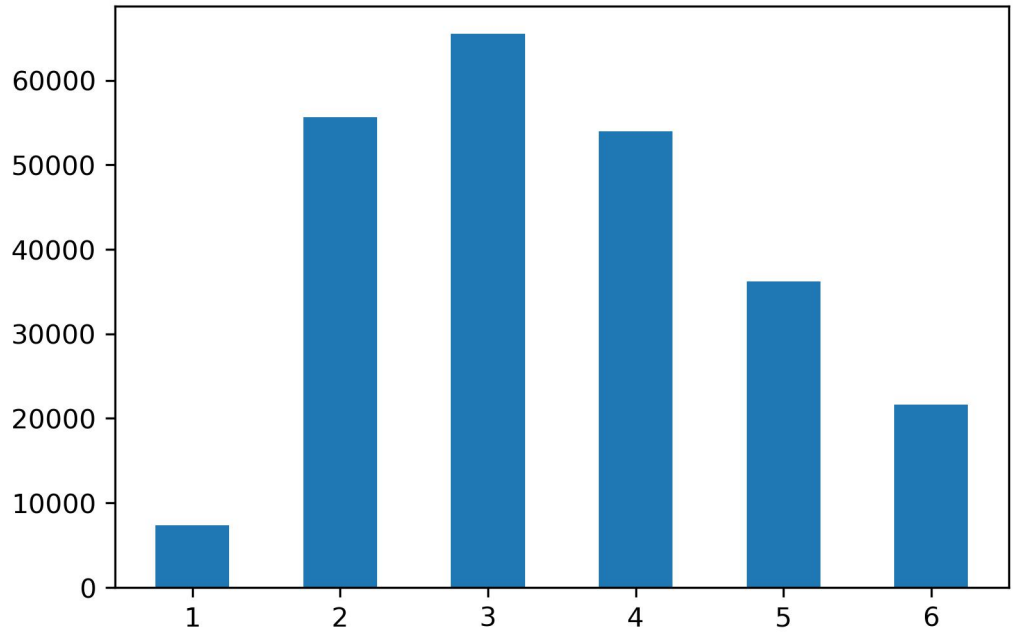


图 4. 训练集上的用户年龄分布情况图

1 到 6 分别对应于赛题中的不同年龄段，可以看到此处作为训练集的用户年龄分配没有出现比较极端的情况：某个年龄段的用户数量过大或者过小，而是整体呈近似的正态分布，这对于模型的训练来说是有利的事情，合理地提升了模型泛化能力。

2.2 数据预处理

针对上一节的数据探索情况，我们需要采取相应的措施，即数据预处理，在正式开始特征提取和模型构建步骤前，将一切潜在的问题解决。

1. 对于缺失值的处理方法：对于字符串型字段缺失值，直接使用空字符串进行缺失值填充；对于数字型字段缺失值，直接使用字段平均值进行填充。

2. 对于数据量级差距过大处理方法：使用 MinMaxScaler 进行行为特征数据的预处理，使其量级保持在相同的范围内

3. 利用训练集年龄分布情况：之后的模型融合中，也参考了训练集上的年龄分布情况，在 ensamble 过程分配权重，一定程度上是参考图 4 的。

三、赛题解析过程

3.1 特征工程 - 初步处理

3.1.1 处理字符串数据

方法：sklearn.preprocessing.LabelEncoder
sklearn.preprocessing.OneHotEncoder

首先结合 LabelEncoder 对原始的文本数据，进行离散化的 Label 映射，将字符串型的数据转换成离散型的整数数据。

结合 OneHotEncoder 对处理之后的离散整数数据进行 OneHot 向量的转换，扩充维度，最终的结果是每个原始字符串数据对应于一个单独的 OneHot 向量。

关于 LabelEncoder 与 OneHotEncoder 的原理与使用方法请自行查阅 sklearn 官网，此处不做赘述。我们需要关注的点是，在通过 LabelEncoder 转换之后，字符串变量成为整数变量，可以直接作为特征输入到模型中进行训练，优缺点如下：

优点：转换速度快，内存占用小

缺点：如果直接作为训练特征数据，会导致原始字段内的数据之间产生大小关系，降低模型训练效果

所以，我们往往会在机器条件允许的情况下(一般是内存条件)，采取 OneHotEncoder 对转换之后的整数型变量进行 OneHot 向量的映射，消除整数与整数之间的大小差距，而导致的模型效果不充分的问题，但此方法也存在优缺点：

优点：作为训练特征的效果明显优于前者

缺点：OneHot 编码转换的速度较慢，内存占用高，对硬件要求高

尤其是在本次比赛中，不同字段下对应到的 OneHot 向量的整体数量可能过于多，导致同学们的 PC 上可能会出现内存溢出(Memory Error)的问题，所以选择折中的 LabelEncoder 单处理应该是较好的特征处理方法。

3.1.2 处理用户安装的 APP 编号序列

通过对特征数据的研究，我们可以很容易地发现，用户安装的 APP 列表数据是本次比赛的强特征数据，即核心特征，之后的特征生成步骤也基本上是围绕着此数据表展开的。所以，我们需要先对字符型的 APP 编号进行上一节的整型化，然后再通过 Tfidf 进行安装列表的文本特征提取。

方法：sklearn.feature_extraction.text.CountVectorizer
sklearn.feature_extraction.text.TfidfVectorizer

CountVectorizer 是属于常见的特征数值计算类，是一个文本特征提取方法。对于每一个训练文本，它只考虑每种词汇在该训练文本中出现的频率，将输入文本中的词语转换成词频矩阵

TfidfVectorizer 除了考量某词汇在文本出现的频率，还关注包含这个词汇的所有文本的数量，能够削减高频没有意义的词汇出现带来的影响, 挖掘更有意义的特征，相比之下，文本条目越多，Tfidf 的效果会越显著

处理过程：

输入：字符串列表，此处对应于每个用户对应的 APP 安装序列，并且每个字符串内的单个字符之间需要做好分隔处理(一般是采用空格进行分割)

训练：fit()函数将训练文本中的词语转换为词频矩阵，矩阵元素 a[i][j] 表示 j 词在第 i 个文本下的词频，即各个词语出现的次数

输出: 训练完成的文本模型结合 transform()函数，将原始的测试文本数据，转换成对应的词频矩阵(一般维度较大，须以稀疏矩阵形式存储)

讲述完整体的处理方法之后，我们可以简单地了解 Tfidf 背后的原理[2]：

0. TF-IDF(Term Frequency-Inverse Document Frequency, 词频-逆文件频率).意为：一个词语在一篇文章中出现次数越多, 同时在所有文档中出现次数越少, 越能够代表该文章.

1. 词频 (term frequency, TF)指的是某一个给定的词语在该文件中出现的次数。这个数字通常会被归一化(一般是词频除以文章总词数), 以防止它偏向长的文件。

$$TF_w = \frac{\text{在某一类中词条}w\text{出现的次数}}{\text{该类中所有的词条数目}}$$

2. 逆向文件频率 (inverse document frequency, IDF) IDF 的主要思想是：如果包含词条 t 的文档越少, IDF 越大，则说明词条具有很好的类别区分能力。某一特定词语的 IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到。

$$IDF = \log(\frac{\text{语料库的文档总数}}{\text{包含词条}w\text{的文档数}+1})$$

3. 结合上述定义概念与计算公式，作者定义 TF-IDF 的计算公式为：

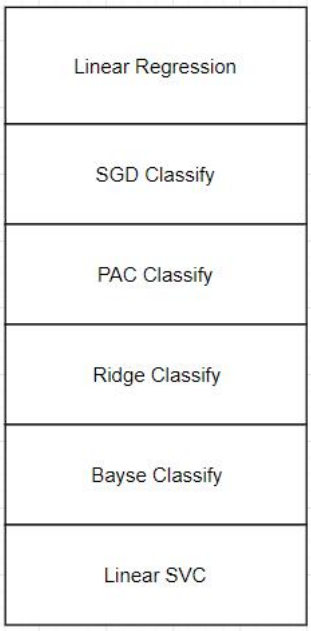
$$TF - IDF = TF * IDF$$

3.2 特征工程 - 特征生成

通过上一节的初步特征工程，我们目前可以掌握的特征数据有：整数化后的用户基本属性信息、用户 30 天内行为信息，和用户安装 APP 列表的 Tfidf 化后的稀疏词矩阵。当然，我们可以直接使用这些特征进行模型训练，但稀疏矩阵进行训练，时长过于长，我们需要在此基础上，提取生成新的特征，优化训练速度，思考新的思路。

3.2.1 利用 Stacking 思想生成特征

首先考虑的就是参考之前易观赛题的做法[3]，选取多个基础分类器，提取这些分类器在 3.1 节获取的特征基础上的分类结果，将这些分类结果作为新的生成特征保存。(下面是使用的基分类器情况，包括 LR, SGD, PAC 等)



步骤：

1. 初始化各项基分类器，将上一步获取的稀疏矩阵作为特征集，结合 train 和 test 表内的训练、预测用户 uid 进行特征集的划分
2. 将用于训练的特征集输入至分类器中进行训练
3. 获取不同分类器在特征集上的预测结果，每一种分类器的预测结果对应于一列，最终生成关于所有用户的，多个基分类器的预测结果特征

3.2.2 统计用户安装的总 APP 数目生成特征

此过程相对来说较为简单，只需要统计不同用户的 uid 对应的 APP 编号列表(以#分隔)，分割完之后的 APP 编号数量，统计并进行保存即可。

关于统计过程，此处可讲述 pandas 的一个有用特性 apply，它支持自定义 lambda 函数的执行，用户可以自己写计算分割后的 APP 数量的函数，并 apply 到原有的数据表上，即可生成新的一列特征。

```
def get_app_len(df):
    return len(df.split('#'))

app_number_feat['app_number'] = app_package['appid'].apply(lambda x: get_app_len(x), 1)
```

3.2.3 利用 LDA 模型降维生成特征

方法：sklearn.decomposition.LatentDirichletAllocation

针对 Tfidf 模型生成的词频矩阵转换成训练数据时，面临维度过大的问题，可以结合 LDA 主题模型进行维度上的降低。

LDA 是一种文档主题生成模型，也称为一个三层贝叶斯概率模型，包含词、主题和文档三层结构。所谓生成模型，就是说，我们认为一篇文章的每个词都是

通过“以一定概率选择了某个主题，并从这个主题中以一定概率选择某个词语”这样一个过程得到。文档到主题服从多项式分布，主题到词服从多项式分布

输入：TfidfVector 训练之后的词频稀疏矩阵，同时初始化 LDA 模型生成的主题数量
训练：结合用于训练的”文本-词频”对应信息，得出 LDA 模型的所有主题矩阵
输出：根据 transform()函数，获取当前文本在 LDA 模型的所有主题上的权重分布向量
(实际上，不同用户的 APP 安装信息对应于单个主题权重分布向量，由此可以生成 LDA 降维后的特征数据)

3.2.4 利用 RNN 处理 user_app_usage.csv 表，生成特征

方法：keras 实现 RNN 进行时序数据处理

预处理：结合 C++，读取庞大的 user_app_usage.csv，然后针对单个用户，以使用 APP 的时间先后，生成其对应的 APP 使用序列，包括编号序列与使用时间序列。

处理的详细过程为：通过 C++的 getline()函数，一行一行地读取 csv 表内的数据，在一定程度上避免了全表读入的内存不足问题，然后对每一行的用户行为数据做处理，针对同一 uid 的行数据做合并，合并依据为时间先后，即最终合成的是一个用户对不同 APP 的使用情况统计列表，列表内的单个元素以时间先后排序，元素的结构为：APP 编号+APP 使用时长，元素与元素之间以#号分隔。

整体过程步骤：

1. 序列词典化

结合 keras 的 Tokenizer 库处理上步骤获取的用户 APP 行为的时间序列，并且将其词典化，具体的处理方法为：

1) 词典化

```
tokenizer = Tokenizer(lower=False, char_level=False, split='#')
tokenizer.fit_on_texts(list(time_app_usage['app_list']))
```

2) 序列化

```
X_seq = tokenizer.texts_to_sequences(train['app_list'])
X_test_seq = tokenizer.texts_to_sequences(test['app_list'])
```

2. 固定长度化

为了使得词向量之间的维度保持一致，训练时不至于报错，此处需要对序列化之后的每个词向量进行固定长度化，不足长度的向量以 0 填充，超过的则截短，具体处理方法如下：

```
maxlen = 100
X = pad_sequences(X_seq, maxlen=maxlen, value=0)
X_test = pad_sequences(X_test_seq, maxlen=maxlen, value=0)
```

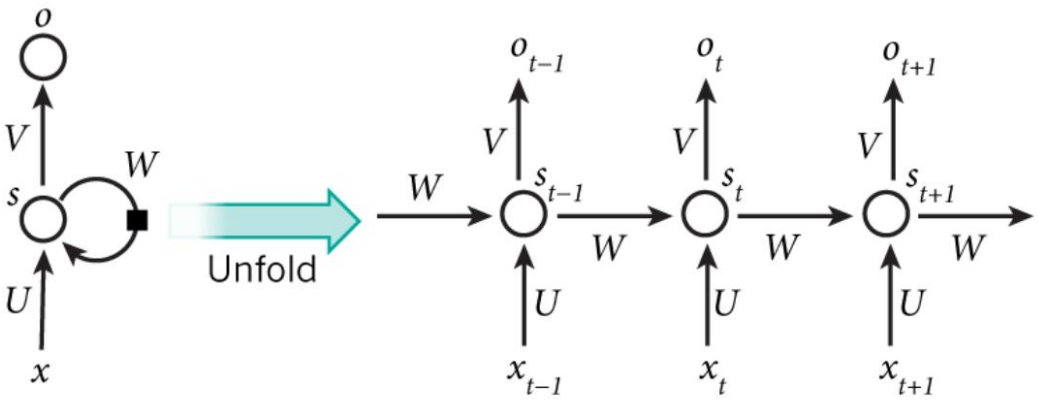
3. Word2Vec 转换为 Embedding 矩阵

产生 RNN 输入的最关键一步最在于产生 Embedding 矩阵，此过程借助了 gensim 库的 Word2Vec 接口，将之前的词向量进一步压缩转换为 embedding 矩阵，用于 RNN 训练过程的输入。

```
max_features = 30000
embedding_matrix = np.zeros((max_features, embedding_size))
for word in tokenizer.word_index:
    if word not in fast_model.wv.vocab:
        continue
    embedding_matrix[tokenizer.word_index[word]] = fast_model[word]
embedding_pd = pd.DataFrame(embedding_matrix)
embedding_pd.to_csv(dataPath + 'embedding_matrix.csv', index=False)
```

4. 输入 RNN 进行模型训练
5. 得到分类输出，产生新特征

(注：该图出自引用[4])

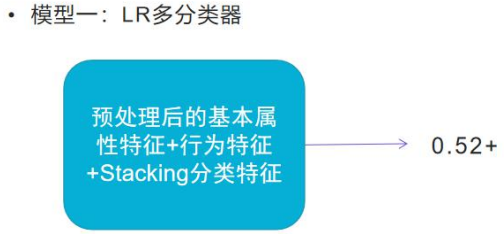


最终结果：由于原始表存在较多缺失值，同时通过实践表明 RNN 处理时间长，分类效果没有得到显著的提升，推荐将该庞大的数据表直接 drop 掉

3.3 模型选取

3.3.1 单 LR 分类器

获取上述特征工程生成的新特征，用于单 LR 分类器的输入，结果为：



考虑到 LR 分类器的内部结构简单，故尝试将原始的 Tfidf 转换后的稀疏矩阵进行特征数据输入到 LR 分类器中进行训练，发现耗时相对可接受(大约 5~6 小时)，结果也挺不错：

3.3.2 lightGBM 模型

lightGBM 模型相对来说内部结构复杂，此处尝试使用 Stacking 过程+统计 APP 总数特征+LDA 降维特征作为特征数据输入，结果如下：

考虑到模型的结果不算太好，最终仍然使用了原始的 Tfidf 稀疏矩阵，作为 lightGBM 的输入特征，耗时约为十几个小时，最终结果是几次提交以来最好的一次：

四、比赛结果与总结

首先总结一下本次比赛的结果：最终的最好成绩是 0.608147，还在 100 名榜上，详情可以查看下面的提交分数历史和排名截图。

然后就是本次数据挖掘比赛的整体历程，本人是在 TA 布置完任务之后的一个星期，开始分析赛题，初步地进行特征工程和模型训练的。一开始受到了易观比赛的优秀解决方案的影响，于是第一阶段的工作就集中在生成新特征，包括 Stacking 与 LDA 降维处理，第一阶段持续了一个多星期，其中也包括了对 pandas 的使用学习，以及 lightGBM 的模型使用。

第二阶段由于没找到更好的特征突破口，分数也没起来，于是就把目光投向原本的 Tfidf 转换后的稀疏矩阵数据，也了解到原来用 LR 分类器直接处理稀疏矩阵，不仅在速度上可以接受，结果上还算不错 -- 0.59+，名次也提升了很多。之后，使用 lightGBM 直接处理该稀疏矩阵，结合调参之后的 RNN 处理特征，可以达到 0.60+的分数。

整体来说，这次期末项目本人使用了接近一个月的时间进行，其中的过程是较为辛苦的，尤其是对于 python 各种库的使用，经常需要查阅网上的博客进行解决。但最后的结果是很好的，尤其是当每一次提交之后的分数上升和排名上升，都会让人觉得有成就感，付出是值得的。

最后，本次报告也预示着本学期数据挖掘实训课程的完全结束，在本学期的课程学习中，本人对数据挖掘的整体过程了解更加深厚，对不同类型的赛题研究，包括：分类问题，回归问题等的处理方法和模型认识有了飞跃的进步，这也需要感谢 TA 和老师对于课件的认真准备，以及课堂上的耐心讲解，和课后作业的用心批改，也看到了 TA 们对同学们的负责，尤其是多次地提醒同学们要早点动手开始做期末项目，是十分负责任的行为，因为数据挖掘比赛往往都是不能一蹴而就的。也希望能在之后的学业上有所建树，在数据挖掘这个方向上取得更大的进步!

参考文献

[1] 华为人工智能研究所. DiGiX 极客数据挖掘比赛 用户人口属性预测赛题简介[EB/OL]. 江苏：江苏省南京市华为研究所二期，2019.
(https://developer.huawei.com/consumer/cn/activity/devStarAI/algo/competition.html#/preliminary/info/digix-trail-02/introduction)

[2] Zrc199021. CSDN 博客论坛. TF-IDF 原理及使用[EB/OL]
(https://blog.csdn.net/zrc199021/article/details/53728499)

[3] dailidong. Github 开源仓库网站.
2018_Analysys-2nd_Algorithm_Competition
(https://github.com/analysys/2018_Analysys_2nd_Algorithm_Competition)

[4] AC_Best_. CSDN 博客论坛. RNN 原理小结[EB/OL]
(https://blog.csdn.net/ieyibinxu/article/details/80251781)

代码展示

本人代码 Github 仓库链接
(https://github.com/Palette25/Comprehensive_Ranking/tree/master/Final_Project)

提交成绩（按照示例截取完整提交历史）

1. 提交历史截图

2019-06-30 11:09:20	0.606223
2019-06-27 20:52:22	0.606597
2019-06-27 10:06:53	0.608147
2019-06-23 12:22:04	0.594567
2019-06-23 10:21:47	0.545930
2019-06-22 00:15:22	0.514259
2019-06-21 19:47:32	0.146693
2019-06-21 14:34:13	0.236295

2019-06-21 11:22:01	0.543453
2019-06-21 01:01:42	0.545956
2019-06-20 18:37:55	0.538217
2019-06-20 11:32:51	0.527021
2019-06-05 10:53:13	0.527620
2019-06-04 14:21:21	0.524147
2019-06-02 17:30:37	0.536209
2019-05-31 13:11:32	0.530627

2. 当天成绩在前 100 名排行榜上的截图