

SMPCUP2017用户画像技术评测比赛

数据集:

任务一: 博文关键词抽取

任务:

指标:

实验:

1. 预处理过程: (2件事)

1.1 扩充jieba的IDF词典

(QA:)

1.2 基于train训练集, 用Ida抽出关键词

1.2.1·参数背景:

1.2.2·调用Ida包抽取关键词并统计词频, 构建Ida-dic:

2. 基于规则的更新权值策略 (主函数将用到):

3. 关键词抽取主函数

3.1 增加文章标题权值----方法: 直接把标题重复12遍和博客内容拼接起来

3.2 从文章中提取英文单词增加到jieba的词典dictionary中, 并基于jieba.suggest_freq调节此字符词频

3.3 **利用jieba的tf-idf方法抽出关键词top20**, 根据单词长度更新它的权重----- $w=w*\text{float}(\text{len}(x))$

3.4 **利用jieba的textrank抽出top20**, 根据单词长度更新它的权重----- $w=w*\text{float}(\text{len}(x))$

3.5 挑选出tf-idf Top15∩textrank Top15∩train词典 放入u中

4. 结果

5. 任务一总结:

任务二: 给用户兴趣打标签

任务:

指标:

实验1 (方法1):

1. 预处理过程:

1.1 基于Word2Vec把100万篇博文的每个词语表示成300维向量

1.2 每篇文章取tf-idf的Top15来表示, 这些词对应的Word2Vec向量加和求平均, 作为一篇文章的表示。

1.3 把每个用户的所有文章对应的300维向量加和求平均, 即最后输入模型中的矩阵是 用户数量*300

2. Stacking模型:

2.1 Stacking模型的第一层: (基于sklearn, 调用了如下5个简单模型)

2.2 将它们进行5折交叉验证输出预测结果

2.3 将每个模型输出的预测结果合并为新的特征(矩阵5*5), 并使用新的模型SVM加以训练

3. 结果

实验2 (方法2): (后期改进---毕设)CNN+self-attention

Background: Self-Attention; scaled dot-Product attention; Multi-head attention(AAAI2018 大连理工IR实验室地刁宇峰)

完整框架 (CNN+self-attention)

实验结果: 准确率达到42%, 提高3个百分点。

任务三: 用户成长值预测

任务:

指标:

实验:

1. 预处理过程: (相关性分析, 特征抽取)

1.1 行为计数

1.2 行为增长

1.3 将行为计数和行为增长特征拼接, 即每个用户77维 (76维+1维=77维)

2. Stacking模型

2.1 Stacking模型的第一层: (PassiveAggressiveClassifier+GrandientBoosting)

PassiveAggressiveClassifier:

GBDT:

2.2 Stacking模型的第二层:

3. 结果

总结与展望

SMPCUP2017用户画像技术评测比赛

数据集：

由2015年的**15万8000个CSDN上的用户**的博文内容信息和行为信息组成；可以进一步分成三个部分，

(1)**用户博客100万篇**，信息形式即：用户ID \博文ID \标题 \博文内容

(2)6类用户行为信息，包括发帖、浏览、评论、投票顶、投票踩、收藏，信息形式即：博文ID\时间 年月日分；以及用户成长值得分，信息形式即：用户ID\分数

(3)用户之间的关系，具体指关注信息和私人记录。信息形式即：用户ID1\用户ID2\时间 年月日分

任务一：博文关键词抽取

任务：

需要从每个文档中提取3个能够很好地代表主题的关键字。

指标：

$$Score_1 = \frac{1}{N} \sum_{i=1}^N \frac{|K_i \cap K_i^*|}{|K_i|}$$

N=博文的数目； $K_i=5$ (标准答案给予每篇文章的5个主题词); $K=3$ （提交的3个主题词）

实验：

1.预处理过程：（2件事）

1.1扩充jieba的IDF词典

- 在jieba自带词典的基础上构建了基于train训练集的IDF词典，增大特殊词权重
- jieba本身词典形式是 [单词 词性 词频]
- 构建基于train训练集的IDF词典(文件)；某个词的idf值 = $\log((\text{博文总数}) / (\text{包含该词的博文数量} + 1))$ ；IDF词典文件的形式：词语，IDF分数

(QA:)

1.如何构造idf词典

[1对每个文档分词去重;

[2把这些文档（[1中的文档）合并，并统计每个词出现的频次。按照（word,count）存放到字典中

[3然后对于分词集([2中的文档)里的每一个词语w，求其idf值， $\text{idf} = \log(\text{文档总数} / (\text{含有w的文档数量} + 1))$ 。注意3中含有单词w的文档数量即=2中单词w的词频

[4最后以词典dic形式存到txt文件里（w单词, idf）

2.为什么要构造idf词典

通过增大特殊词的权重，提高关键词的合理性

因为jieba是基于自身的语料库来计算IDF值的。它范围很广泛。但是我们的任务是基于CSDN的后台开发或者算法类话题的。因此在这种特定场合，需要自定义语料来提高关键词的合理性

1.2基于train训练集，用lda抽出关键词

1.2.1·参数背景：

Task 2:需要生成3个标签来描述用户的兴趣，其中标签是从一个给定的候选集中产生的(一共是42个)，所以我们假设每篇文章可以用42个主题来描述。

1.2.2·调用lda包抽取关键词并统计词频，构建lda-dic:

1【输入样例】

词语1 词语2 词语3 词语4 词语5 词语6 词语7 词语8 词语9 词语1 词语2 词语3 词语4 词语5 词语1 词语2 词语3 词语4
词语5 词语6 词语7 每一行是一篇（已切好词的）博文，词语之间用空格分隔

2【需要调整的参数说明】

1.n_topics 42: 主题个数，即需要将这些文本聚成几类，这里是42类

2.n_iter 500: 迭代次数

3.twords 100 :用以描述该主题的词数，按照概率从高到低选取，这里设的是100

4.Random state 1: 随机种子(取值在1-100之间)

3【程序常用输出说明】

1.(doc-topic分布) 输出文章属于类的分布概率，文本一行表示一篇文章，概率1 概率2...概率42（这里是42个topic,每行就有42个概率） 2.(topic-word分布)输出所有词与类的分布概率（即每个topic内词的分布，包含这个词的概率/权重），是一个K*M的矩阵，K为设置分类的个数（这里是42行），M为所有文章的词的总数 3.model_twords.dat : 输出每个类前topN高频词（这里即每个topic内权重最高的100个词语） 4.每篇文本最可能的topic -----这里只用把42个类的每个类的前100个高频词输出即可

4【具体实现】

1. 读取已切好词的语料库所有词语，去重(其中一行代表一篇文章)
2. 生成词频矩阵X，一行一个文本，一列一个词语，数值等于该词语在当前文本中出现的频次

矩阵行数=文本总数，矩阵列数=语料库去重后词语总数，该矩阵是一个大的稀疏矩阵

3. 模型训练

```
model = lda.LDA(n_topics = 42, n_iter = 500, random_state = 1)
model.fit(X)
```

4. 输出每个topic内权重最高的100个词语,将这42*100个词写入文件42topic-word.txt

```
n = 100
print('=====topic top' + str(n) + ' word=====')
for i, topic_dist in enumerate(topic_word):
    topic_words = np.array(wordList)[np.argsort(topic_dist)][:(-n+1):-1]
    print('*Topic {}\n-{}'.format(i, ' '.join(topic_words)))
```

5. 统计4中的 42topic-word.txt中的词频，构建lda-dic词典

2.基于规则的更新权值策略（主函数将用到）：

通过规则（英语词权重增加）和属于LDA抽出来的词（权重调整）对词的tf-idf进行调整：

（1如果是英语词，权重=权重*1.7

（2如果是LDA主题词，权重=权重*（1+（1.0 / float(dicc[key]) - 1.0 / 42.0）），float(dicc[key])即这个词在LDA-dic中的词频

（3最后返回按照权值逆序排好序的词典，形式为（单词，权值）

3.关键词抽取主函数

对于测试集test的每一篇博客：

3.1 增加文章标题权值-----方法：直接把标题重复12遍和博客内容拼接起来

3.2从文章中提取英文单词增加到jieba的词典dictionary中，并基于jieba.suggest_freq调节此字符词频

3.3利用jieba的tf-idf方法抽出关键词top20,根据单词长度更新它的权重-----w=w*float(len(x))

通过2中：英文单词权重增加和属于LDA抽出来的词权重增加的规则，对tf-idf的前20个单词进行调整，再抽取出前15个，并按照权重大小降序排序

3.4利用jieba的textrank抽出top20,根据单词长度更新它的权重-----w=w*float(len(x))

函数： jieba.analyse.textrank(string, topK=20, withWeight=True, allowPOS=()) string：待处理语句 topK：关键字的个数，默认20 withWeight：是否返回权重值，默认false allowPOS：是否仅返回指定类型，默认为空

通过2中：英文单词权重增加和属于LDA抽出来的词权重增加的规则，对text-rank前20个单词进行调整：再抽取出前15个，并按照权重大小降序排序

3.5挑选出tf-idf Top15∩textrank Top15∩train词典 放入u中

如果交集数量<3：

对jieba-tfidf分出的前5个词，根据单词长度更新权重加入到final-dic词典中

对final-dic中的词：通过2中英文单词权重增加的规则和属于LDA抽出来的词权重调整的方法对权重调整

把词典按照权重逆序排序抽取出权重前3的词

QA1:为什么要和train构成的词典取交集

因为train和test的词是相似的，在加了这个条件之后，准确率增加了2个百分点（偏工程性）

QA2:为什么在二次迭代的时候，采用基于tf-idf抽取的关键词进行更新，而不是textrank

1. TextRank的效果并不优于TFIDF。
2. TextRank虽然考虑到了词之间的关系，但是仍然倾向于将频繁词作为关键词。
3. TextRank涉及到构建词图及迭代计算，所以提取速度较慢。

4.结果

得分 0.56

结果评估指标：

N：博客数量 Ki：抽取出的关键词(3个) Ki：标准标签关键词（5个） |Ki|:标准标签关键词的数目，这里是5

$$\text{score1} = (1/N) * \sum_{i=1 \sim N} |K_i \cap K_i^*| / |K_i|$$

5.任务一总结：

规则：

词长：词的权重*词长，提高长词和英语词组的权重

英语词（组）：英语词组权重增加, $w = w * 1.7$

标题：根据写作习惯，标题相比较正文权重增加

训练集优先：以训练集构建字典，字典中词组权重增加

改进：

别人组用的哪些方法可以借鉴？抽取关键词大体上有哪些方法

任务一大多数组用的是无监督方法，tf-idf, textrank, 根据数据特点建立一些规则去调整；

但是后来反思也可以尝试用监督学习算法，将关键词抽取过程视为二分类问题，先抽取出候选词，然后对于每个候选词划定标签，要么是关键词，要么不是关键词，然后训练关键词抽取分类器。当新来一篇文档时，抽取出所有的候选词，然后利用训练好的分类器，对各个候选词进行分类，最终将标签为关键词的候选词作为关键词。

任务二：给用户兴趣打标签

任务：

这个子任务的目标是用42个给定的标签中的3个标签来标记每个用户的兴趣。

指标：

$$\text{Score}_2 = \frac{1}{N} \sum_{i=1}^N \frac{|T_i \cap T_i^*|}{|T_i|}$$

N =用户的数目； $|T_i|=3$ (T_i : 标准答案给予每个用户的3个标签); $|T_i^*|=3$ (T_i^* : 提交的每个用户的3个标签)

实验1（方法1）：

1.预处理过程：

1.1基于Word2Vec把100万篇博文的每个词语表示成300维向量

把100万篇博文分词，并基于gensim包的word2vec把每个词表示成300维的向量

1.2每篇文章取tf-idf的Top15来表示，这些词对应的Word2Vec向量加和求平均，作为一篇文章的表示。

1.3把每个用户的所有文章对应的300维向量加和求平均，即最后输入模型中的矩阵是 用户数量*300

2.Stacking模型：

Stacking模型就是将几个简单的模型，一般采用将它们进行K折交叉验证输出预测结果，然后将每个模型输出的预测结果合并为新的特征，并使用新的模型加以训练。

2.1 Stacking模型的第一层：（基于sklearn，调用了如下5个简单模型）

SGD随机梯度下降、逻辑回归LR、SVM支持向量机、XGB,RF

以RF为例说明调用RF模型以及调节参数过程：

对Random Forest来说：**1.增加“子模型数”（n_estimators）可以明显降低整体模型的方差，且不会对子模型的偏差和方差有任何影响。模型的准确度会随着“子模型数”的增加而提高。**由于减少的是整体模型方差公式的第二项，故准确度的提高有一个上限。**2.调整“最大叶节点数”（max_leaf_nodes）或者“最大树深度”（max_depth），**可以粗粒度地调整树的结构：叶节点越多或者树越深，意味着子模型的偏差越低，方差越高；**3.同时，调整“分裂所需最小样本数”（min_samples_split）**可以更细粒度地调整树的结构：分裂所需样本数越少或者叶节点所需样本越少，也意味着子模型越复杂。**4.适当地减少“分裂时考虑的最大特征数”（max_features），**给子模型注入了另外的随机性**，同样也达到了降低子模型之间关联度的效果。但是一味地降低该参数也是不行的，因为分裂时可选特征变少，模型的偏差会越来越大。

调用RF模型过程：

```
def random_forest_classifier(train_x, train_y):  
    from sklearn.ensemble import RandomForestClassifier  
    model = RandomForestClassifier(n_estimators=40)  
    model.fit(train_x, train_y)  
    return model
```

具体调整参数过程：

1.调整过程影响类参数

首先，我们需要对过程影响类参数进行调整，而Random Forest的过程影响类参数只有“**子模型数**”（n_estimators）。“子模型数”的默认值为10，在此基础上，我们以10为单位，考察取值范围在1至201的调参情况。随着“子模型数”的增加，整体模型的方差减少，其防止过拟合的能力增强，故整体模型的准确度提高。当“子模型数”增加到40以上时，准确度的提升逐渐不明显。考虑到训练的效率，最终我们选择“子模型数”为200。

2.调整子模型影响类参数

在设定“子模型数”（n_estimators）为200的前提下，我们依次对子模型影响类的参数对整体模型性能的影响力进行分析。

对“分裂条件”（criterion）分别取值gini（最小基尼系数）和entropy（最小熵），分析调参结果可得，在此问题中，“**分裂条件**”保持默认值gini更加合适。

对“**分裂时参与判断的最大特征数**”（max_feature）以1为单位，设定取值范围为28至47，得到调参结果如下：“分裂时参与判断的最大特征数”的默认值auto，即总特征数的开方。通过提升该参数，整体模型的准确度得到了提升。可见，该参数的默认值过小，导致了子模型的偏差过大，从而整体模型的偏差过大。同时，我们还注意到，**该参数对整体模型性能的影响是近似单调的**：从28到38，模型的准确度逐步抖动提升。所以，我们可考虑将该参数纳入下一步的调参工作。

对“**最大深度**”（max_depth）以10为单位，设定取值范围为10到100，得到调参结果如下：随着树的深度加深，子模型的偏差减少，整体模型的准确度得到提升。从理论上来说，子模型训练的后期，随着方差增大，子模型的准确度稍微降低，从而影响整体模型的准确度降低。不妨以1为单位，设定取值范围为40到59，更加细致地分析：有点傻眼了，怎么跟预想的不太一样？为什么模型准确度的变化在40到59之间没有鲜明的“规律”了？要分析这个问题，我们得先思考一下，**少一层子节点对子模型意味着什么？若少的那一层给原子模型带来的是方差增大，则新子模型会准确度提高；若少的那一层给原子模型带来的是偏差减小，则新子模型会准确度降低。所以，细粒度的层次变化既可能使整体模型的准确度提升，也可能使整体模型的准确度降低。从而也说明了，该参数更适合进行粗粒度的调整。**在训练的现阶段，“抖动”现象的发生说明，此时对该参数的调整已不太合适了。

对“最大叶节点数”（max_leaf_nodes）以100为单位，设定取值范围为2500到3400，得到调参结果如下：类似于“最大深度”，该参数的增大会带来模型准确的提升，可是由于后期“不规律”的抖动，我们暂时不进行处理

3.调参总结：

还需要继续下一轮坐标下降式调参吗？一般来说没有太大的必要，在本轮中出现了两个发生抖动现象的参数，而其他参数的调整均没有提升整体模型的性能。还是得老调重弹：**数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已**。在测评竞赛中，与其期待通过对RandomForestClassifier调参来进一步提升整体模型的性能，不如挖掘出更有价值的特征，或者使用自带特征挖掘技能的模型（比如说神经网络）。

RF模型原理特点：

基本思想就是构造多棵相互独立的CART决策树，形成一个森林，利用这些决策树共同决策输出类别。随机森林算法秉承了bagging方法的思想，以构建单一决策树为基础，同时也是单一决策树算法的延伸和改进。

在整个随机森林算法的过程中，有两个随机过程：

1.输入数据是随机的：从全体训练数据中选取一部分来构建一棵决策树，并且是有放回的选取 2.每棵决策树的构建所需的特征是从全体特征中随机选取的

2.2将它们进行5折交叉验证输出预测结果

以XGB模型为例，把train分train1~train5,共5份，用其中4份预测剩下的那份,同时预测test数据，这样的过程做5次,生成5份train（原train样本数/5）数据和5份test数据。然后把5份预测的train数据纵向叠起来，把test预测的结果取平均。

其他模型和XGB模型一样，再来4次。这样就生成了5份train数据和5份test数据（XGB重新表达的数据、RF重新表达的数据、.....）



2.3将每个模型输出的预测结果合并为新的特征(矩阵5*5)，并使用新的模型SVM加以训练

然后用SVM模型，进一步做融合，得到最终的预测结果。

因为是多分类，数据量并不小，所以采用建立42个这样的SVM分类器，最后取max{top3}

3.结果

准确率 39%

实验2（方法2）：(后期改进---毕设)CNN+self-attention

Background: Self-Attention; scaled dot-Product attention; Multi-head attention(AAAI2018 大连理工IR实验室地刁宇峰)

以计算 “I really love nlp”为例，说明以下几种attention的区别

Self-Attention: (如何计算 love的 attention值)

1. 将每个词转化成词向量
2. 计算attention和其他词向量的点积运算 (或者计算余弦相似度)
3. 将这4个结果基于softmax归一化
4. 然后再分别和4个词本来的value (词向量) 点积运算 (相乘再相加) 得到4个数值
5. 将这4个数值相加即得到单词 love在这句话中的attention值 (即权重再分配)

Scaled dot-product attention:

在Self-Attention基础上进行修改:

在进行softmax归一化之前，点积运算之后，公式改一下，分子仍然不变（点积运算的结果），分母由1变成(根号下 d_k)， d_k 即为词向量的维度，目的是：调节内积的值防止过大或者过小，太大的话softmax后就非0即1了，不够“soft”了

。公式如下

本文对基本的attention模型进行了少许改进，提出了缩放点积attention (scaled dot-Product attention)。在使用点积运算进行相似度计算的基础上，缩小了 $\sqrt{d_k}$ 倍 (d_k 为词向量的维度)。其目的在于调节的作用，使得内积不易过大。

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

```
1 def scaled_dotproduct_attention(Q, K, V):
2     QKT = np.dot(Q, K.T)
3     d_k = np.sqrt(K.shape[0]) # scaled part
4     return np.dot(softmax(QKT / d_k), V)
```

multi-head attention: (这里拟定multi为h次)

在Scaled Attention基础上进行修改:

1. 将每个词转化成词向量，并进行一个线性变换
2. 将1的结果放到scaled-attention里面去
3. 1、2步骤重复做h次，每次的输入为线性变换后的原始输入。但是每次线性变换的参数w是不一样的（这里的h次操作类似于卷积网络里使用不同的卷积核多次进行卷积操作）
4. 将h次attention的结果进行拼接，将拼接后的模型再做一次线性变换 $xW_1 + b_1$ ，得到的值为多头attention的结果

多头attention (Multi-head attention) 的结构公式如图所示。首先，需要对query、key和value 进行一个线性变换；然后输入到缩放点积attention机制，重复做h次，每次的输入为线性变换后的原始输入，这里，多头就是指做多次attention之后进行拼接，每一次算一个头，每次Q、K和V的线性变换参数W是不一样的；最后，将拼接后的模型做一次线性变换，得到的值为多头attention的结果。可以看出，多头attention与传统的attention区别在于计算了h次，这样可以从不同的维度和表示子空间里学习到相关的信息，可通过attention可视化机制来验证。

multi-head attention的实现：

要注意的是：Multi-Head的意思虽然很简单——重复做几次然后拼接，但事实上不能按照这个思路来写程序，这样会非常慢。因为tensorflow是不会自动并行的，比如

```
a = tf.zeros((10, 10))
b = a + 1
c = a + 2
```

其中b,c的计算是串联的，尽管b、c没有相互依赖。因此我们必须把Multi-Head的操作合并到一个张量来运算，因为单个张量的乘法内部则会自动并行。

此外，我们要对序列做Mask以忽略填充部分的影响。一般的Mask是将填充部分置零，但Attention中的Mask是要在softmax之前，把填充部分减去一个大整数（这样softmax之后就非常接近0了）。这些内容都在代码中有对应的实现。

```
'''
Multi-Head Attention的实现
'''
def Attention(Q, K, V, nb_head, size_per_head, Q_len=None, V_len=None):
    #对Q、K、V分别作线性映射
    Q = Dense(Q, nb_head * size_per_head, False)
    Q = tf.reshape(Q, (-1, tf.shape(Q)[1], nb_head, size_per_head))
    Q = tf.transpose(Q, [0, 2, 1, 3])
    K = Dense(K, nb_head * size_per_head, False)
    K = tf.reshape(K, (-1, tf.shape(K)[1], nb_head, size_per_head))
    K = tf.transpose(K, [0, 2, 1, 3])
    V = Dense(V, nb_head * size_per_head, False)
    V = tf.reshape(V, (-1, tf.shape(V)[1], nb_head, size_per_head))
    V = tf.transpose(V, [0, 2, 1, 3])
    #计算内积，然后mask，然后softmax
    A = tf.matmul(Q, K, transpose_b=True) / tf.sqrt(float(size_per_head))
    A = tf.transpose(A, [0, 3, 2, 1])
    A = Mask(A, V_len, mode='add')
    A = tf.transpose(A, [0, 3, 2, 1])
    A = tf.nn.softmax(A)
    #输出并mask
    O = tf.matmul(A, V)
    O = tf.transpose(O, [0, 2, 1, 3])
    O = tf.reshape(O, (-1, tf.shape(O)[1], nb_head * size_per_head))
    O = Mask(O, Q_len, 'mul')
    return O
```

完整框架 (CNN+self-attention)

1. 嵌入层:用tf.embedding得到一个三维的张量，每个词向量仍然是300维，用PAD把句子补充完整
2. 卷积\池化：选取3*3的卷积核进行卷积，卷积核个数20个，通道数1，池化尺寸2 * 2，步长2
3. 卷积\池化：卷积核33，卷积核个数100个，通道数20个，池化尺寸20 * 20，步长2
(缩小卷积核尺寸，增加卷积核数目都会提高准确率，卷积核大小从5 * 5变成3 * 3，提高了1.9个百分点)
4. multi-self-attention层,然后把这些数据“拍平”，丢到Flatten层，

5. 然后把Flatten层的output放到full connected Layer（全连接）里，合成一个长向量，采用softmax对其进行分类。
6. dropout训练时设为0.5，即禁止用一些点防止过拟合；测试时就默认为1

实验结果：准确率达到42%，提高3个百分点。

任务三：用户成长值预测

任务：

根据每个用户的情况预测其未来6个月的增长趋势

过去一年的行为，包括短信、关系和与其他用户的互动。增长趋势需要缩放到[0,1]，其中0表示用户的退出。

指标：

$$Score_3 = 1 - \frac{1}{N} \sum_{i=1}^N \begin{cases} 0, & v_i = 0, v_i^* = 0 \\ |v_i - v_i^*| / \max(v_i, v_i^*), & otherwise \end{cases}$$

N=用户的数目；Vi: 用户预测成长值；Vi*: 用户实际成长值；0表示用户的退出

实验：

1.预处理过程：（相关性分析，特征抽取）

1.1行为计数

[1]按照月统计用户7种行为频次，共7*12=84维；

[2]经相关性分析，第38列相关性为0，故剔除第38列

相关性分析的具体方法：

基于numpy和pandas包,调用df.corr (numpy包),计算出84列分别和y（成长值）的皮尔逊相关系数。

用X、Y的协方差除以X的标准差和Y的标准差。从数值来看，协方差的数值越大，两个变量同向程度也就越大。反之亦然。如果是0，则表示无关

$$\rho = \frac{Cov(X, Y)}{\sigma_X \sigma_Y} :$$

[3]将12个月的行为取均值；+1后取log的值作为每个用户的一个行为计数

1.2行为增长

统计每个用户7种行为的月间增长率(GR(dt))，增长率可以反映行为的变化情况，共76维（77-1）

$$GR(d_t) = \frac{d_{t+1} - d_t}{d_t + 1}$$

d(t+1): 第t+1个月的用户某行为频次

d(t): 第t个月的用户某行为频次

1.3将行为计数和行为增长特征拼接，即每个用户77维（76维+1维=77维）

2.Stacking模型

Stacking模型就是将几个简单的模型，一般采用将它们进行K折交叉验证输出预测结果，然后将每个模型输出的预测结果合并为新的特征，并使用新的模型加以训练。

2.1Stacking模型的第一层：（PassiveAggressiveClassifier+GrandientBoosting）

PassiveAggressiveClassifier:

实现：

sklearn提供很多增量学习算法 例如sklearn.linear_model.PassiveAggressiveClassifier

其中对于回归问题，在第一次调用partial_fit时不需要通过classes参数指定分类的类别。

PassiveAggressive分类器的特点：

在于每次只能得到一个样本点，无法保留历史数据，对每一个新的样本点进行分析，根据分析的结果更新分类器。适合增量型数据

PassiveAggressive分类器分类原理步骤：以二分类为例

\1. 设置参数C (C>0)

\2. 设定W的初值 $w_i=(0,...,0)$

\3. 每接收一个样本 X_t , 计算 $Y_t=\text{sign}(W_t \cdot X_t)$: 其功能是取某个数的符号（正或负）：当 $x>0$, $\text{sign}(x)=1$;当 $x=0$, $\text{sign}(x)=0$; 当 $x<0$, $\text{sign}(x)=-1$

获取类别 y_t , 取值为-1或1

\4. 计算损失值: $l_t = \max\{0, 1 - Y_t\}$

\5. 更新权值:

5.1计算 T_t $T_t = l_t / \|X_t\|^2$

5.2更新权值: $W_{t+1} = W_t + T_t \cdot Y_t \cdot X_t$

GBDT:

实现: GradientBoostingRegression

参数里面 $n_estimators$ 和 max_depth 与gbdt的表达能力相关度很高

```
import numpy as np
from sklearn.ensemble import GradientBoostingRegressor
gbdt=GradientBoostingRegressor(
    loss='ls'
    , learning_rate=0.1#学习率 0.1或者更小
    , n_estimators=100#子模型的数量，默认是100
    , subsample=1#子采样率
    , min_samples_split=2#分裂所需最小样本数
    , min_samples_leaf=1#"叶节点最小样本数
    , max_depth=3#深度
    , init=None
```

```

, random_state=None
, max_features=None
, alpha=0.9
, verbose=0
, max_leaf_nodes=None
, warm_start=False
)
train_feat=np.genfromtxt("train_feat.txt",dtype=np.float32)
train_id=np.genfromtxt("train_id.txt",dtype=np.float32)
test_feat=np.genfromtxt("test_feat.txt",dtype=np.float32)
test_id=np.genfromtxt("test_id.txt",dtype=np.float32)
print train_feat.shape, train_id.shape, test_feat.shape, test_id.shape
gbdt.fit(train_feat,train_id)
pred=gbdt.predict(test_feat)
total_err=0
for i in range(pred.shape[0]):
    print pred[i],test_id[i]
    err=(pred[i]-test_id[i])/test_id[i]
    total_err+=err*err
print total_err/pred.shape[0]

```

参数调整：

1.GradientBoostingClassifier的过程影响类参数有“子模型数”（n_estimators）和“学习率”（learning_rate），

这里留了一个很大的陷阱：“子模型数”和“学习率”带来的性能提升是不均衡的，在前期会比较高，在后期会比较低，如果一开始我们将这两个参数调成最优，这样很容易陷入一个“局部最优解”。在目标函数都不确定的情况下（如是否凸？），谈局部最优解就是耍流氓，在此，我们先直觉地选择“子模型数”为60，“学习率”为0.1，此时的整体模型性能（平均准确度为0.822）不是最好，但是也不差，良好水准。

2.调整子模型影响类参数

子模型数	学习率	叶节点最小样本数	最大深度	子采样率	分裂时参与判断的最大特征数
n_estimators	learning_rate	min_samples_leaf	max_depth	subsample	max_feature
60	0.1	12	4	0.77	10

到此，整体模型性能为0.8343，与baseline相比，提升了约0.012

3.回马枪

还记得一开始我们对“子模型数”（n_estimators）和“学习率”（learning_rate）手下留情了吗？现在我们可以回过头来，调整这两个参数，调整的方法为成倍地放大“子模型数”，对应成倍地缩小“学习率”（learning_rate）。通过该方法，本例中整体模型性能又提升了约0.003。

GBDT模型原理

GBDT是第一棵树分类后的残差，作为第二棵树的输入；依次类推直到残差为0或者到达树深

以决策树（CART）为基学习器的GB算法

GBDT是GB和DT的结合。要注意的是这里的决策树是回归树，GBDT中的决策树是个弱模型，深度较小一般不会超过5，叶子节点的数量也不会超过10，对于生成的每棵决策树乘上比较小的缩减系数（学习率<0.1），有些GBDT的实现加入了随机抽样（subsample 0.5<=f <=0.8）提高模型的泛化能力。通过交叉验证的方法选择最优的参数。

2.2 Stacking模型的第二层：

通过对第一层的两个基本模型（Passive Aggressive和GBDT）的预测值取平均，我们将创建一个新特性并将其输入到叠加模型NuSVR中。由于基础模型的随机性，我们采用了10倍交叉验证的自检机制。

NuSVR是SVM的一种回归模型，可以基于 sklearn.svm.NuSVR() 调用

```
def stacking(base_models, X, Y, T):
    models = base_models
    folds = list(KFold(len(Y), n_folds=10, random_state=0))#10折交叉
    S_train = np.zeros((X.shape[0], len(models)))#训练集
    S_test = np.zeros((T.shape[0], len(models)))#测试集
    for i, bm in enumerate(models):
        clf = bm[1]
        S_test_i = np.zeros((T.shape[0], len(folds)))
        for j, (train_idx, test_idx) in enumerate(folds):
            X_train = X[train_idx]
            y_train = Y[train_idx]
            X_holdout = X[test_idx]
            clf.fit(X_train, y_train)
            y_pred = clf.predict(X_holdout)[:]
            S_train[test_idx, i] = y_pred
            S_test_i[:, j] = clf.predict(T)[:]
        S_test[:, i] = S_test_i.mean(1)
    nuss=NuSVR(kernel='rbf')#核函数，默认值
    nuss.fit(S_train, Y)
    yp = nuss.predict(S_test)[:]
    return yp
```

3.结果

0.854

总结与展望

用户之间的网络关系尚未被应用，可以考虑邻近用户的标签来对用户进行分类