

华东师范大学计算机科学技术系上机实践报告

课程名称: 智能推荐系统

年级: 2018

上机实践成绩:

指导教师: 张伟

姓名: 张翼鹏

创新实践成绩:

实验编号: 02

学号: 10185102105

上机实践日期: 2021/05/06

实验名称: Content-based Recommendation

1 问题重述

本次实践要求使用基于内容的推荐算法预测用户对某页面中各条新闻进行点击的可能性作出预测, 给出的数据包括 `train.tsv/train_news.tsv` 和 `test.tsv/test_news.tsv`。

- 对于训练集: `train_news.tsv` 包含新闻 ID, 新闻一级二级分类、新闻标题和新闻摘要, `train.tsv` 包含用户 ID, 用户数据产生时间, 用户浏览历史和用户对当前页面中各条新闻的点击情况。
- 对于测试集: `test_news.tsv` 包含新闻 ID, 新闻一级二级分类、新闻标题和新闻摘要, `test.tsv` 包含用户 ID, 用户数据产生时间, 用户浏览历史和当前页面的各条新闻。

最终要求对测试集中用户对当前页面中的各条新闻点击的可能性以列表的形式输入到新的列 `Predict` 里 (那么显然有 $|Impression| = |Predict|$)。

2 算法原理

2.1 算法概述

对于该算法的概述, 首先来看《Recommender Systems Handbook》中的一段原文:

Content-based recommendation systems try to recommend items similar to those a given user has liked in the past. Indeed, the basic process performed by a content-based recommender consists in matching up the attributes of a user profile in which preferences and interests are stored, with the attributes of a content object (item), in order to recommend to the user new interesting items.

——《Recommender Systems Handbook》P73

我们可以把这个过程简要概括为: 根据用户过去喜欢的物品, 为用户推荐和他过去喜欢的物品相似的物品。

2.2 算法流程

一个完整的基于内容的推荐系统架构图 1 所示, 结合概述部分, 我把整个过程划分为三个部分, 并以此为依据实现该算法:

- **物品画像构建:** 通俗来讲也就是对待推荐的物品建模, 这里“物品”的概念是相对于“用户”的概念的, 就本次实践而言也就是从非结构化的文本内容中抽取一些属性来代表这条新闻, 最终结果可以是一个向量。

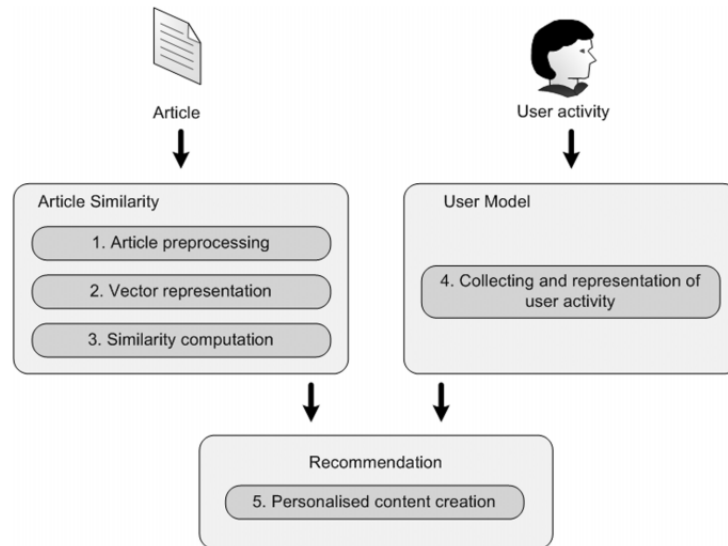


图 1: Proposed news recommendation method

- **用户偏好学习**: 对用户的偏好学习是基于内容的推荐系统的中间环节, 往往通过对用户数据的分析给用户“画像”, 用户画像由两个部分构成, 一部分是对用户端内容挖掘分析后的用户的属性特征, 另一部分是根据用户行为数据统计或学习到的对物品端结构化属性的偏好度量, 本次实践根据用户的浏览历史刻画其偏好。
- **生成推荐列表**: 通过比较前两部分得到的数据, 将物品与用户偏好进行比对, 推荐用户最有可能感兴趣的物品 (往往是按照相似度倒排序, 取相似度最大的前 N 个物品进行推荐), 本次任务要求计算用户点击当前页面某条新闻的可能性, 可以用余弦相似度来刻画。

接下来将从几个模型切入, 详述算法各个模块的实现。

3 TF-IDF 优化模型

3.1 新闻内容建模

在信息检索、文本挖掘以及自然语言处理领域, TF-IDF (term frequency-inverse document frequency, 词频-逆文档频率, 简称 TF-IDF) 算法都可以说是鼎鼎有名。虽然在这些领域中, 目前也出现了不少以深度学习为基础的新的文本表达和算分 (Weighting) 方法, 但是 TF-IDF 作为一个最基础的方法, 依然在很多应用中发挥着不可替代的作用。

TF-IDF 算法在基于内容的推荐系统中, 发挥的作用主要有两个, 一个是关键词提取, 来获取最基础的标签; 另外一个作用就是为所有属性标签赋值 (Weighting)。

记我们要表示的所有文章集合为 $D = \{d_1, d_2, \dots, d_n\}$, 而所有文章中出现的词 (对于中文文章, 首先得对所有文章进行分词) 的集合 (也称为词典) 为 $T = \{t_1, t_2, \dots, t_n\}$ 。也就是说, 我们有 N 篇要处理的文章, 而这些文章里包含了 n 个不同的词。我们最终要使用一个向量来表示一篇文章, 比如第 j 篇文章被表示为 $d_j = \{w_{1j}, w_{2j}, \dots, w_{nj}\}$, 其中 w_{ij} 表示第 i 个词 t_i 在文章 j 中的权重, 值越大表示越重要, d_j 中其他向量的解释类似。TF-IDF 就是在向量空间模型的假设下的一种更加复杂的赋值方式。

TF-IDF 如何计算呢, 最基础的模式, 顾名思义, 就是 TF 和 IDF 的乘积。

先来看整个公式：

$$TF-IDF Score_{(t_i, d_j)} = TF_{(t_i, d_j)} \times IDF_{(t_i)} = TF_{(t_i, d_j)} \times \log\left(\frac{N}{n_i + 1}\right)$$

其中, TF 表示词频 (Term Frequency), IDF 表示逆文档频率 (Inverse Document Frequency), $TF_{(t_i, d_j)}$ 表示第 i 个词 t_i 在第 j 篇文档 d_j 中出现的次数, N 为所有文档总数, n_i 表示有 n 篇文档包含第 i 个词, 也就是 DF (Document Frequency) 文档频率。

python 的 sklearn 库中提供了相关的函数来计算 TF-IDF, 可以直接调用, 对于该库的使用方法, 其源代码给出了一个例子:

```

1 Examples
2 -----
3 >>> from sklearn.feature_extraction.text import TfidfTransformer
4 >>> from sklearn.feature_extraction.text import CountVectorizer
5 >>> from sklearn.pipeline import Pipeline
6 >>> import numpy as np
7 >>> corpus = ['this is the first document',
8 ...           'this document is the second document',
9 ...           'and this is the third one',
10 ...          'is this the first document']
11 >>> vocabulary = ['this', 'document', 'first', 'is', 'second', 'the',
12 ...               'and', 'one']
13 >>> pipe = Pipeline([('count', CountVectorizer(vocabulary=vocabulary)),
14 ...                  ('tfidf', TfidfTransformer())]).fit(corpus)
15 >>> pipe['count'].transform(corpus).toarray()
16 array([[1, 1, 1, 1, 0, 1, 0, 0],
17        [1, 2, 0, 1, 1, 1, 0, 0],
18        [1, 0, 0, 1, 0, 1, 1, 1],
19        [1, 1, 1, 1, 0, 1, 0, 0]])
20 >>> pipe['tfidf'].idf_
21 array([1.          , 1.22314355, 1.51082562, 1.          , 1.91629073,
22        1.          , 1.91629073, 1.91629073])

```

从中我们可以看到, 要调用该库, 我们需要得到一个新闻语料库, 其中按照字符串的格式存储每条新闻, 最终得到 TF-IDF 矩阵, 每行表示不同的新闻向量, 也即是说, 我们从给出的新闻数据中只需要得到一个 corpus 列表, 其中 corpus[i] 表示第 i 条新闻的字符串, 该库可以自动提取字符串中的关键字 (显然我们需要将一些没意义的词去掉, 也就是 nltk 库中的 stopwords 停用词):

```

1 # 将文本中的词语转换为词频矩阵
2 vectorizer = CountVectorizer()
3 # 计算个词语出现的次数
4 X = vectorizer.fit_transform(corpus)
5 # 获取词袋中所有文本关键词

```

```

6 word = vectorizer.get_feature_names()
7 # 类调用
8 transformer = TfidfTransformer()
9 # 将词频矩阵  $X$  统计成  $TF-IDF$  值
10 tfidf = transformer.fit_transform(X)

```

我们上手处理数据后，问题接踵而至。可以看到，新闻提供的信息除了摘要文本，还包括了新闻的一级分类、二级分类和新闻标题，这些内容从权重上来讲和摘要文本是不能同等看待的。文献 [1] 将每篇新闻抽象成“六向量”，如下图所示：

Vector part	Weights
Title	transplantácia_0.5 tvár_0.5
TF of title words in the content	transplantácia_0.0178571428571429 tvár_0.0714285714285714
Category	Sme.sk_0.5 PRESS_FOTO_1.0
Keywords	klinika_0.0357142857142857 povrch_0.0178571428571429 nos_0.0178571428571429 zub_0.0178571428571429 nerv_0.0178571428571429 svalstvo_0.0178571428571429 pacientka_0.0178571428571429 rozsah_0.0178571428571429
Names/Places	Cleveland_1
CLI	0.2543

图 2：“六向量法”举例

由于数据量过大，我仍然觉得应该使用一维向量来表示一个新闻文本，于是在分词前，我想到了可以增加权重大的部分的计数，比如：

信息类型	权重
Category	4
Subcategory	3
Title	2
Abstract	1

此外，对于文本信息而言，我们很容易假设，在同等条件下，名词和动词能提供的信息比介词、连词等要重要的多（名词可能甚至更胜一筹），将这些“权重”要求也要纳入考量，比如：

词性	权重
名词	3
动词	2
其他	1

到此为止，如果一条新闻的分类为“动画片”，子分类为“海绵宝宝”，标题为“应聘”，摘要为“海绵宝宝信心饱满的准备去蟹堡王应聘”，那么按照我们前述的处理方式，最终该新闻 id 对应的字符串为：

“动画片 动画片 动画片 动画片 动画片 动画片 动画片 动画片 动画片 动画片
动画片 动画片 海绵宝宝 海绵宝宝 海绵宝宝 海绵宝宝 海绵宝宝 海绵宝宝 海绵宝宝



图 3: 海绵宝宝例子词云

同时，现实文本中很容易出现一个词的多种形式，比如”drive”和”drove”，”driven”很可能表达的是相同的意思，这就要求我们进行词型还原；在自然语言处理背景下，提到词型还原很容易联想到词干提取，不过查阅了很多资料后发现，在这个场景下，词干提取显得不是那么有必要，很有可能提取出一些没有意义的词干，因此我们这里只进行词型还原。

```

1 # 朴素算法中新闻各项所占权重
2 WEIGHT_OF_CATEGORY = 4
3 WEIGHT_OF_SUBCATEGORY = 3
4 WEIGHT_OF_TITLE = 2
5 WEIGHT_OF_ABSTRACT = 1
6
7 # 朴素算法中新闻各项所占权重
8 WEIGHT_OF_NOUN = 3
9 WEIGHT_OF_VERB = 2
10
11 # 读入一个字符串，返回分词后的字符串
12 def text_preprocessing(row_text):
13     token_words = nltk.word_tokenize(row_text.lower())
14     characters = ['.', ':', ';', '?', '(', ')', '[', ']',
15                  '&', '!', '*', '@', '#', '$', '\\%', '-', '...',
16                  '^', '{', '}']
17
18     # 分词、删除标点并小写
19     token_words = [str for str in token_words if str not in characters]
20     token_words = pos_tag(token_words) # 词性标注
21     # 词型还原
22     words_lematizer = []

```

```

23 wordnet_lematizer = WordNetLemmatizer()
24 for word, tag in token_words:
25     if tag.startswith('NN'):# n 代表名词
26         word_lematizer = wordnet_lematizer.lemmatize(word, pos='n')
27         for i in range(WEIGHT_OF_NOUN-1):
28             words_lematizer.append(word_lematizer)
29     elif tag.startswith('VB'):# v 代表动词
30         word_lematizer = wordnet_lematizer.lemmatize(word, pos='v')
31         for i in range(WEIGHT_OF_VERB-1):
32             words_lematizer.append(word_lematizer)
33     elif tag.startswith('JJ'):# a 代表形容词
34         word_lematizer = wordnet_lematizer.lemmatize(word, pos='a')
35     elif tag.startswith('R'):# r 代表代词
36         word_lematizer = wordnet_lematizer.lemmatize(word, pos='r')
37     else:
38         word_lematizer = wordnet_lematizer.lemmatize(word)
39     words_lematizer.append(word_lematizer)
40
41 words_list = [word for word in words_lematizer\
42               if word not in stopwords.words('english')]
43 return ' '.join(words_list)+' '

```

nlTK 库在做词形还原前应该先进行词性标注以提升还原的准确性，但是祈使句句首的动词由于首字母大写会被标记为名词，因此我们在实现的时候应该先转化为小写再进行处理。

3.2 用户偏好学习

本题给出了用户的浏览历史作为用户偏好集，我们可以对此进行训练，这里将用户浏览过的新闻记录到一个名为 history 的列表中，取其平均向量作为偏好向量，核心代码如下：

```

1 # 计算用户偏好向量
2 def get_user_pref(history):                                     # history 为用户浏览历史列表
3     total_history = len(history)
4     if(total_history == 0):
5         return []
6     history_sum = tfidf[history[0]]
7     for i in range(total_history-1):
8         history_sum += tfidf[history[i+1]]
9     return history_sum/total_history

```

3.3 模型评估

首先这个模型的训练需要一定的时间，为了增强使用者的体验，可以在循环迭代器外层套一个 rich 库里的 track 函数，该函数可以在终端显示一个进度条，实时监测迭代任务完成的百分比：

```
F:\PyCharm 2020.3.3\Projects\Project2>python news_preprocessing.py
新闻数据处理中... 38% 0:23:21
```

图 4: Caption

观察数据发现，大部分的预测值集中在 0 附近，这样的话显然不太准确，对于预测用户是否点击某条新闻，比较科学的做法是设定一个阈值，大于该值则预测会点击，否则不会，不过本次任务要求输出 0-1 之间的一个可能性，这就要求我们将这些结果映射到 0-1 之间。

很容易想到的一个办法是取出这些值的最大值和最小值，按照比例将其映射到 0-1 之间，不幸的是在进行了尝试之后我发现并没有任何变化，也就是最值就是 0 和 1，通过输出结果发现 1780 行出现了预测值为 1 的情况，检查以后发现，这是因为该用户的浏览历史只包含一条浏览数据，也就是说用户偏好就是以这条数据刻画的，而当前页面又出现了这条新闻，因此预测值必定为 1：

1779	U23378	11/12/2019	N18777	N8	N51255-0	N6890-0	N16096-0	N46607-0	N62386-0	N45704-0	N7821-0	N336
1780	U73096	11/11/2019	N47020	N55204-0	N13801-0	N63550-0	N63538-0	N35729-0	N59981-0	N47020-1	N1	
1781	U45453	11/14/2019	N59704	N3	N22257-0	N35576-0	N24109-0	N58264-0	N18409-0	N25165-0	N32005-0	N1

图 5: 预测值为 1 的特殊情况

所以按照极大值极小值将预测值放缩是没有效果的，又想到了取对数的方式，观察这些值，大部分分布在 0 附近，小部分增长到 1，这很容易想到对数函数图像，如果我们取该图像的其中的一部分，以此为依据对预测值进行放缩，可以一定程度上拉开这些预测值的距离：

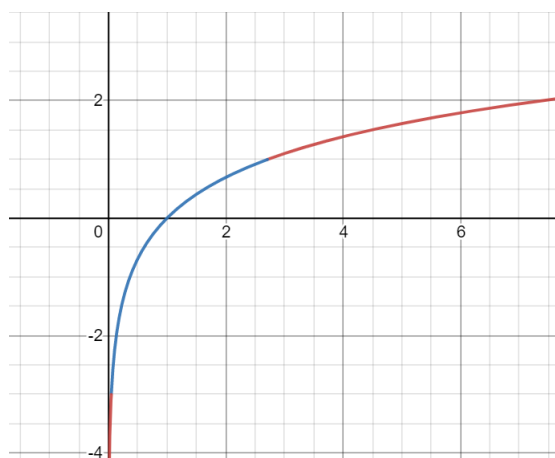


图 6: $y=\ln(x)$ 的部分图像

比如我们取 $(e^{-3}, -3)$ 到 $(e, 1)$ 这段函数的图像，变换方式：

$$[0, 1] \rightarrow [e^{-3}, e] \rightarrow [-3, 1] \rightarrow [0, 1]$$

对于一段区间 $[l_1, r_1]$ ，若想要把他变换到区间 $[l_2, r_2]$ ，可以先变为 $[0, 1]$ 区间，变换函数为：

$$f_1(x) = \frac{x - l_1}{r_1 - l_1}$$

再变换到 $[l_2, r_2]$ ，变换函数为：

$$f_2(x) = l_2 + x \times (r_2 - l_2)$$

综上所述, $[l_1, r_1]$ 到 $[l_2, r_2]$ 的变换函数为:

$$f(x) = l_2 + \frac{(x - l_1) \times (r_2 - l_2)}{r_1 - l_1}$$

最终对于每个用户, 输出 RMD 的平均值观察结果:

	A	B	C	D
1	Uid	Actual_val	Pred_val	Avg_RMD
2	U48196	[' N58410-C	[0.0659, 0	0.1328
3	U46641	[' N20270-C	[0.0011, 0	0.295725
4	U48973	[' N23446-C	[0.0, 0.0	0.0688964
5	U15980	[' N57713-C	[0.1364, 0	0.1115688
6	U48830	[' N53245-C	[0.0202, 0	0.0937212
7	U52471	[' N40109-C	[0.0064, 0	0.20625
8	U5658	[' N59267-C	[0.0, 0.0	0.1017556
9	U70453	[' N41220-C	[0.035, 0	0.2159667
10	U43436	[' N26130-C	[0.1883, 0	0.0935944

图 7: 模型测试结果

从这个角度看模型似乎还不错, 但实际上由于 TD-IDF 矩阵的维度很大, 多数预测值都是接近 0 的, 即使我们经过了上一步的放缩过程, 大部分也依然和 0 接近, 在实际用户点击情况中又多数是 0 (未点击), 所以 RMD 值也说明不了什么。

4 Doc2vec 模型

在课上老师给的 PPT6-advanced-content 的最后一页提到了 Doc2vec 方法, 参考文献 [2] 中也使用了该方法, 由于 gensim 库中已经封装好了模型的训练代码, 我对此方法进行了尝试。

4.1 Doc2vec 之我见

首先阅读了 Word2vec 的经典论文 [3] 和, 结合一些前人的总结讲下我对 Word2vec 的简单理解。

在自然语言处理领域, 最细粒度的是词语, 词语组成句子, 句子再组成段落、篇章、文档。所以处理 NLP 的问题, 首先就要从词语切入。

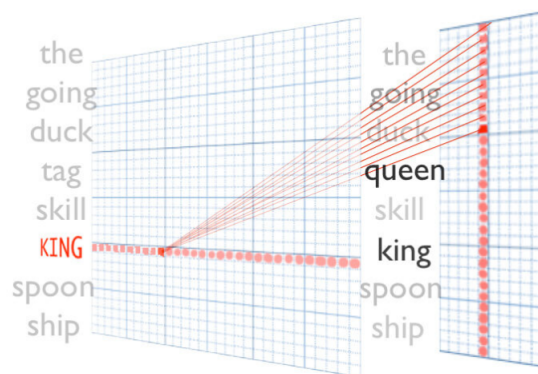


图 8: 词嵌入

举个简单例子，判断一个词的词性，是动词还是名词。用机器学习的思路，我们有一系列样本 (x,y) ，这里 x 是词语， y 是它们的词性，我们要构建 $f(x) \rightarrow y$ 的映射，但这里的数学模型 f （比如神经网络、SVM）只接受数值型输入，而 NLP 里的词语，是人类的抽象总结，是符号形式的（比如中文、英文、拉丁文等等），所以需要把他们转换成数值形式，或者说嵌入到一个数学空间里，这种嵌入方式，就叫词嵌入 (Word Embedding)。词嵌入表示模型最早应用在自然语言处理领域中，利用背景信息构建词汇的分布式表示。嵌入式表示模型往往简单而又相对比较有效，因此很快被人们应用到推荐系统中。其核心思想就是同时构建用户和物品的嵌入式表示，使得多种实体的嵌入式表示存在于同一个隐含空间内，进而可以计算两个实体之间的相似性。而 Word2vec，就是词嵌入 (word embedding) 的一种。

Doc2vec 模型是受到了 Word2Vec 模型的启发。Doc2vec 方法是一种无监督算法，能从变长的文本（例如：句子、段落或文档）中学习得到固定长度的特征表示。Doc2vec 也可以叫做 Paragraph Vector、Sentence Embeddings，它可以获得句子、段落和文档的向量表达，是 Word2Vec 的拓展，其具有一些优点，比如不用固定句子长度，接受不同长度的句子做训练样本。Doc2vec 算法用于预测一个向量来表示不同的文档，该模型的结构潜在的克服了词袋模型的缺点。

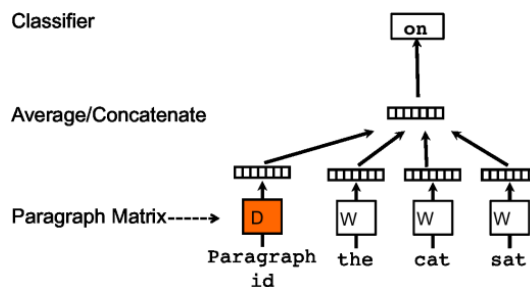
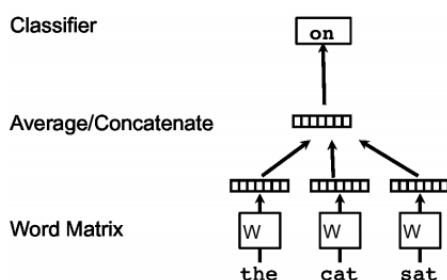


图 9: Framework for learning word vectors 图 10: Framework for learning paragraph vector

Word2Vec 里预测词向量时，预测出来的词是含有词义的，Doc2vec 中也是构建了相同的结构，所以 Doc2vec 克服了词袋模型中没有语义的缺点。假设现在存在训练样本，每个句子是训练样本，和 Word2Vec 一样，Doc2vec 也有两种训练方式，一种是分布记忆的段落向量 (Distributed Memory Model of Paragraph Vectors, PV-DM) 类似于 Word2Vec 中的 CBOW 模型，另一种是分布词袋版本的段落向量 (Distributed Bag of Words version of Paragraph Vector, PV-DBOW) 类似于 Word2Vec 中的 Skip-gram 模型。

遗憾的是不知是因为我代码哪里有问题亦或是电脑算力不足、数据量太大，这个模型我跑了

三次，每次设置电脑不休眠，每次大概都跑了十几个小时，一直卡在一个地方，希望以后有机会运用这个模型。

4.2 新闻内容建模

```
1 stop_words = set(stopwords.words('english'))
2 if not os.path.exists('d2v.model'):
3     news_tags = []
4     news_txt_data = []
5     r_news_tsv = 'train/train_news.tsv'
6     total_news = len(open(r_news_tsv, 'r', encoding='UTF-8').readlines())-1 # 新闻总数
7     with open(r_news_tsv, 'r', encoding='UTF-8') as tsv_in:
8         tsv_reader = csv.reader(tsv_in, delimiter='\t')
9         news_tsv_labels = tsv_reader.__next__()
10        for news_index in track(range(total_news), description='新闻文本读取中...'):
11            record = tsv_reader.__next__()
12            news_tags.append(record[0])
13            news_txt_data.append(record[3]+'.'+record[4])
14
15        tagged_data = [TaggedDocument(words=[word for word in word_tokenize\
16            (_d.lower()) if word not in stop_words], tags=news_tags)
17            for k, _d in enumerate(news_txt_data)]
18
19        # 模型训练参数
20        max_epochs = 100
21        vec_size = 20
22        alpha = 0.025
23
24        model = Doc2Vec(vector_size=vec_size,
25            alpha=alpha,
26            min_alpha=0.00025,
27            min_count=1,
28            dm=1)
29
30        model.build_vocab(tagged_data)
31        for epoch in track(range(max_epochs), description='模型训练中...'):
32            print('第 {0} 次迭代'.format(epoch))
33            model.train(tagged_data, epochs=model.epochs, total_examples=model.corpus_count)
34            model.alpha -= 0.0002
35            model.min_alpha = model.alpha
36
37        model.save("d2v.model")
38        print('模型训练完毕!')
```

4.3 用户偏好学习

如果该模型能够最终预测成功，我们可以用 Doc2vec 中封装好的 Similarity 方法计算两条新闻之间的相似度，对于用户浏览历史，我们既可以像前面那样取一个平均值，也可以对于要判断的新闻与历史中所有新闻分别计算相似度，取最大值作为预测结果。

5 运行方法

5.1 库要求

除了常见的 python 库，需要安装以下库：

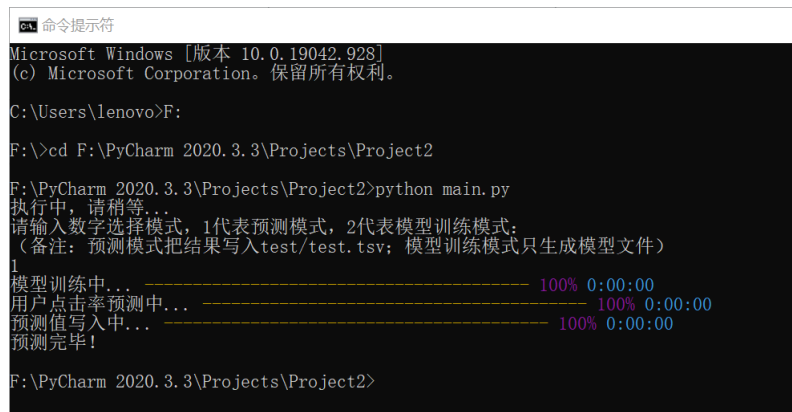
- os
- nltk
- rich
- numpy
- gensim

5.2 运行说明

程序支持点击率预测、模型训练等功能，在主函数中通过 os 库实现程序的调用：

```
1 import os
2 print('执行中，请稍等...')
3 print('请输入数字选择模式，1 代表预测模式，2 代表模型训练模式：')
4 print('（备注：预测模式把结果写入 test/test.tsv；模型训练模式只生成模型文件）')
5 op1 = int(input())
6 while(op1 != 1 and op1 != 2):
7     print('请输入数字选择模式，1 代表预测模式，2 代表模型训练模式：')
8     print('（备注：预测模式把结果写入 test/test.tsv；模型训练模式只生成模型文件）')
9     op1 = int(input())
10 if(op1 == 1):
11     os.system('pred.py')
12 else:
13     print('请输入数字选择模型：1 代表 TF-IDF 优化模型，2 代表 Doc2vec 模型：')
14     op2 = int(input())
15     while (op2 != 1 and op2 != 2):
16         print('请输入数字选择模型：1 代表 TF-IDF 优化模型，2 代表 Doc2vec 模型：')
17         op2 = int(input())
18     if(op2 == 1):
19         os.system('news_preprocessing_tfidf.py')
20     else:
21         os.system('news_preprocessing_doc2vec.py')
```

若要单独训练模型，在文件夹下命令行运行'python main.py'，输入数字'1' 即可：



```
命令提示符
Microsoft Windows [版本 10.0.19042.928]
(c) Microsoft Corporation。保留所有权利。

C:\Users\lenovo>F:

F:\>cd F:\PyCharm 2020.3.3\Projects\Project2

F:\PyCharm 2020.3.3\Projects\Project2>python main.py
执行中，请稍等...
请输入数字选择模式，1代表预测模式，2代表模型训练模式：
（备注：预测模式把结果写入test/test.tsv；模型训练模式只生成模型文件）
1
模型训练中... 100% 0:00:00
用户点击率预测中... 100% 0:00:00
预测值写入中... 100% 0:00:00
预测完毕！

F:\PyCharm 2020.3.3\Projects\Project2>
```

图 11: Caption

6 拓展思考

基于内容的推荐算法有很多值得推敲的细节，由于本题的题干和数据的限制很多因素没有纳入考虑，如果这次实践的要求是给出一个固定长度的推荐列表，或许还要做如下事情：

- 选择一个固定阈值作为是否推荐的标准。在计算完相似度以后，显然需要确定相似度在多少范围是可以被推荐的。
- 对于访问过的新闻，确定其是否需要被再次推荐。很容易想象，在真实场景下，如果用户点击过某条新闻，那这条新闻一定能较好的匹配我们刻画的用户偏好向量，但实际上，尤其在新闻的场景下，用户重复点击已经看过的新闻可能性不大。
- 我们在课上学习过基于内容的推荐系统容易出现“马太效应”，这就要求我们及时更新用户偏好向量，不仅是简单线性的将用户浏览历史进行累加，比如，是否可以监视用户对新闻的浏览时间，如果用户对推荐到的新闻的浏览时间远小于该用户的平均浏览一条新闻所用时间，显然这个推荐结果是失败的，那么就需要我们在用户偏好中剔除这类新闻，这个细节的实现似乎很有难度。
- 在本次实践的两个模型中，第一个模型的训练和预测用时大约在一小时左右，第二个没能成行的模型每次尝试都训练了 12 个小时以上，在现实应用下，用户往往刷新一次就要生成新的推荐，因为模型应该考虑如何加速。
- 或许可以为每一篇推荐的文章存储“文章年龄”。此数字表示文章推荐的时间。如果用户在定义的时间内未访问本文（推荐次数），则本文将会从推荐列表中删除。[1]

7 实践总结

本次实践完成了基于内容的推荐系统并据此进行用户对新闻点击的可能性预测，比较满意的是尝试了两种方法并在细节方面加入了自己的思考和修正，遗憾的是没能最终得到第二个模型，也很期待能有机会向其他同学学习。

参考文献

- [1] Kompan M., Bieliková M. (2010) Content-Based News Recommendation. In: Buccafurri F., Semeraro G. (eds) E-Commerce and Web Technologies. EC-Web 2010. Lecture Notes in Business Information Processing, vol 61. Springer, Berlin, Heidelberg.
- [2] Kevin Joseph and Hui Jiang. 2019. Content based News Recommendation via Shortest Entity Distance over Knowledge Graphs. In Companion Proceedings of The 2019 World Wide Web Conference WWW '19. Association for Computing Machinery, New York, NY, USA, 690–699.
- [3] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14). JMLR.org, II–1188–II–1196.
- [4] Wouter IJntema, Frank Goossen, Flavius Frasincar, and Frederik Hogenboom. 2010. Ontology-based news recommendation. In Proceedings of the 2010 EDBT/ICDT Workshops EDBT '10. Association for Computing Machinery, New York, NY, USA, Article 16, 1–6.