智能推荐系统第三次作业报告

Content-based Recommendation

宋朝芸 10215001419

Kaggle id：NaCloudy

2024 年 5 月 8 日

目录

1 算法介绍

1.1 基于内容的推荐算法

基于内容的推荐算法（Content-based Recommendation）是一种依赖用户历史偏好的物品内容信息进行推荐的算法，物品的内容信息可以是文本、视频、图像等多模态信息。本次作业只有物品的文本信息，因此核心算法与自然语言处理领域相关性很强。这种算法的优势是不需要协同信息，因此不会面临冷启动问题；由于推荐结果只受用户历史偏好的影响，推荐的内容也会更符合用户本身的兴趣，而不会受到用户群体中其他用户的影响。

本次作业要求完成的任务是新闻的序列推荐。每条推荐记录包含用户 ID、用户历史交互新闻 ID、本次推荐的新闻 ID 及其交互情况。目标是对于本次待推荐的多条新闻，根据预测的用户交互优先级进行排序。

算法的要点为:

- 文本预处理
- 文本向量化表示
- 用户向量化表示
- 排序推荐

1. 文本预处理

为了得到更好的文本向量化表示，需要对物品的内容信息进行预处理，所用的方法包括 Tokenization、Normalization、Stemming 和 Remove Stop Words. 许多方法都有常用库，比如英文可以选择 NLTK 或 SpaCy，中文则使用 jieba.

Tokenization（分词）的作用是将字符序列转为 tokens，包括词粒度、句粒度、子词（subword）粒度 [1]。其中最常见的是词粒度，需要利用词的语义信息，可以使用词表匹配来实现。

Normalization（标准化）的作用是将语义相同的 tokens 转为形式一致的结果。比如，将英文语料统一转为小写表示，或者去除标点符号和数字。

Stemming（词干化）主要针对英语文本，去除前缀、后缀，将不同词性、时态、语态的词汇转为统一表示。比如将”running,run” 转为”run,run”。

Remove Stop Words（去除停用词）是将文本中一些意义不大的词去掉，一般是去除介词、冠词，比如中文里的“了”、“的”，英文里的“the”、“a”等。

处理完之后，文本就变成了多个 token 组成的有序列表，以待进一步处理。

2. 文本向量化表示

为了便于统计分析，需要将文本向量化，又称为 embedding. 这一操作可以分为不同的尺度，包括 word 维度和 document 维度（多个 words）。

2.1 word 维度

2.1.1 TF-IDF（Term-Frequency - Inverse Document Frequency）模型

这是一种基础的模型，能够完成 word 维度的表示。给定关键词 i 和文档 j，可以得到一个表示向量。简单来说，TF 表示词汇频度，关键词 i 在文档 j 中出现得越多，表明它越可以代表文档 j 的特性；IDF 代表逆文档频度，关键词 i 在所有文档中出现得越多，表明它越不能代表文档 j 的特性（比如”the” 就是一个例子）。因此，TF-IDF(i,j) = TF(i,j) * IDF(i) 是在词汇频度和特有性上做了折衷。

对于文档 j 而言，将词典中所有词在本篇文章中的 TF-IDF 值拼接为向量，即为本篇文章的向量表示。对于用户 u 而言，可以将历史所有文档的向量表示进行加权平均，得到用户的向量表示。

此外，该模型的一大问题是高维稀疏表示。如果想要表示一篇文档，其特征向量的维度是词典的大小，这会导致内存和计算上存在非必要的巨大开销，可以选择进行降维之后再进行存储或运算。PCA 就是一种常见的降维方法，通过协方差矩阵的特征值分解性质，将数据投影到低维空间，同时尽可能保证数据的方差最大，以继续保持良好的可区分性。但需要注意的是，PCA 的降维维度是需要指定的，可以利用交叉验证法选定超参数，或者使用碎石图等方式保留一定的方差以确定合适的维数。

2.1.2 Okapi BM25 模型

该模型的目的是希望向量表示对词汇频度和文本长度都敏感，因此改进了 TF-IDF 模型中 TF 的部分，使用文档长度进行标准化，考虑了因为文档长度比较长而出现较大 TF 的情况。

2.2 document 维度

2.2.1 概率主题模型

其思想是认为每篇文章背后有一个随机的主题的概率分布，文章中每一个词都是从其中之一

的主题里产生。这一类模型的发展路径是 LSA——PLSA——LDA。其中，LDA 模型引入了先验的狄利克雷分布，可以将每篇文档的主题按照概率分布的形式给出，更加符合直觉。

2.2.2 Doc2Vec

TF-IDF 模型的表示方式是词袋化表示，没有考虑词汇的顺序信息、只考虑了是否出现，而 Doc2Vec 模型就对这点进行了改进。Doc2Vec 采用滑动窗口，使用上下文窗口对目标词进行预测，从而学习到了文本中的语序信息。训练结束后，能够得到训练样本中所有的词向量和每句话对应的句子向量。在预测环境，Doc2vec 随机初始化 Paragraph vector，放入模型中不断迭代，其收敛值即为句子向量。

3. 用户向量化表示

为了量化用户的内容偏好，需要结合用户历史交互信息和文本向量化表示来表征用户。

一种直接的思路是使用加权平均，将用户所有历史文档的向量加权平均，得到用户向量。也可以沿用 Doc2Vec 思想，将用户的所有历史文档拼接，并将该文档使用 Doc2Vec 模型进行表示。此外，还可以将文档降维成概率最大的主题，然后简单统计用户历史文档的主题分布。

要注意的是，以上这些都是在用户存在历史记录的情况下。如果是新用户，则可以借助协同过滤思想，使用用户群体向量的加权平均（平均偏好）来表示新用户。

4. 排序推荐

在得到多篇待推荐新闻的向量表示和用户历史信息的向量表示之后可以根据用户的历史偏好，对待推荐新闻进行排序。

对于使用加权平均或者 Doc2Vec 的用户向量表示而言，排序的依据可以是向量的相似度。由于新闻的向量表示和用户的向量表示是在同一个向量空间中，可以计算余弦相似度，越接近的表明用户可能越感兴趣。这里选择余弦相似度而非欧氏距离，是因为文章中的关键词可能多次出现，这种高频词会使得向量表示的模长增加，但并不意味着在相似度上有很大差异。因此，使用角度而非距离进行度量是更为合理的。

此外，如果文本采用了概率主题模型，直接根据主题的偏好程度来排序是一种更直接的路径。对于每篇新闻，可以取概率最大的主题作为该新闻的标签。对于每位用户，统计用户历史上浏览过的不同主题个数，并根据主题的频度进行新闻推荐。尽管新闻本身也有 category 和 subcategory 的标签，但是这些标签的数量是给定的，而利用概率主题模型可以人为设定主题的数量，这一超参数也可以使用交叉验证法来确定。

1.2 所用算法描述

本次实验主要采用算法是使用 LDA 模型来量化文本表示。由于 title 和 abstract 的文本篇幅较长，希望能有更多数据进行建模和分析，因此每篇新闻拼接 title 和 abstract 作为新闻文本。

此外还需要构建用户表示：每条推荐记录都有 history 字段和 impressions 字段，分别表示用户的历史点击行为（可认为是正例）和待推荐的新闻列表（包含正例和负例）。对测试集而言，用户可以分为三类。

1. 如果用户在训练集中出现，意味着训练集中存在该用户相关联的 impressions 字段（以及可能非空的 history 字段）。将这些字段的正例取出，加上测试集 history 的新闻，可以构成完整的用户正向内容偏好。如果这些字段都不存在正例，则退化为第 2 类或者第 3 类。

2. 如果用户不在训练集中出现，但是测试集 history 字段非空。可以将测试集 history 字段作为用户正向内容偏好。

3. 如果用户不在训练集中出现，且测试集 history 字段为空。这表明用户存在冷启动。可以利用协同过滤的思想，简单地把所有用户的向量加权平均，得到该用户的向量表示。

最后使用历史主题偏好的分布进行排序推荐. 假定只预测推荐记录字典 R' 的第一条记录 r，其用户 id 为 $user\_id$, 推荐新闻字典为 I', 目标是获得推荐新闻序列 S，伪代码见算法 1.

- 训练数据结构：用户历史新闻字典 H, 以用户 id 为键，以历史新闻列表为值. 推荐记录字典 R, 以推荐 id 为键，以用户 id、推荐新闻字典 I 为值. 推荐新闻字典 I, 以推荐新闻 id 为键，以交互情况为值. 新闻信息字典 N, 以新闻 id 为键，以新闻文本为值.

- 预测数据结构：用户历史新闻字典 H', 以用户 id 为键，以历史新闻列表为值. 推荐记录字典 R', 以推荐 id 为键，以用户 id、推荐新闻字典 I' 为值. 推荐新闻字典 I', 以推荐新闻 id 为键，值为空. 新闻信息字典 N', 以新闻 id 为键，以新闻文本为值.

- 超参数：主题数量 num.

此外，还使用了另一种算法作为对比。使用了 Doc2Vec 模型来表示文本，用户表示同算法 1，并使用余弦相似度进行推荐排序。具体见算法 2. 其训练数据结构和预测数据结构同算法 1，超参数是向量大小 $size$, 最小词频 $count$, 迭代轮数 $epoch$.

Algorithm 1 LDA+history+rank

初始化新闻信息字典 $\bar{N}$

for (key,value)$\in N \cup N'$ **do**

value = Process(value), 其中 Process(·) 表示对字符串使用文本预处理成 word 列表.

if key$\notin \bar{N}$ **then**

将 (key,value) 加入 $\bar{N}$.

end if

end for

用新闻文本集合 $value(\bar{N})$ 和指定主题数量 num 训练 LDA 模型.

对每一条新闻 news_id，计算其 LDA 表示 $LDA(news)$，并将分量最大的下标取出作为 topic_news_id.

初始化用户正向内容偏好 User_pref 和平均内容偏好 Avg_pref.

for (key,value)$\in H \cup H'$ **do**

if key$\notin$ User_pref **then**

for news_id in value **do**

User_pref(key)(topic_news_id) += 1

Avg_pref(topic_news_id) += 1

end for

end if

end for

for (key,value)$\in R$ **do**

if value(user_id)$\notin$ User_pref **then**

for (news_id,interaction) in I **do**

if interaction == 1 **then**

User_pref(user_id)(topic_news_id) += 1

end if

end for

end if

end for

用户兴趣主题 User_dict 为 User_pref(user_id) 或 Avg_pref（如果前者为空），新闻主题 Item_dict.

for (news_id,interaction) in I' **do**

Item_dict(news_id) = topic_news_id

end for

将 User_dict 按 value 降序排序，将 Item_dict 按 value 在 User_dict 的顺序的降序排序

return Item_dict 的键为推荐新闻序列 S;

Algorithm 2 Doc2Vec+history+cosine similarity

初始化新闻信息字典 $\bar{N}$

for (key,value)$\in N \cup N'$ **do**

value = Process(value).

if key$\notin \bar{N}$ **then**

将 (key,value) 加入 $\bar{N}$.

end if

end for

用新闻文本集合 $value(\bar{N})$，向量大小 $size$, 最小词频 $count$, 迭代轮数 $epoch$ 训练 Doc2Vec 模型.

初始化用户正向内容偏好 User_pref 和平均内容偏好 Avg_pref.

for (key,value)$\in H \cup H'$ **do**

if key$\notin$ User_pref **then**

for news_id in value **do**

User_pref(key) += Doc2Vec(news_id)

Avg_pref += Doc2Vec(news_id)

end for

end if

end for

for (key,value)$\in R$ **do**

if value(user_id)$\notin$ User_pref **then**

for (news_id,interaction) in I **do**

if interaction == 1 **then**

User_pref(user_id) += Doc2Vec(news_id)

end if

end for

end if

end for

用户向量 User_vec 为 User_pref(user_id) 或 Avg_pref（如果前者为空）, 待推荐新闻矩阵 Item_vec.

for (news_id,interaction) in I' **do**

Item_vec[news_id] = Doc2Vec(news_id)

end for

计算 Cosine_Sim(User_vec,[Item_vec]), 将对应的 news_id 根据相似度降序排序为 S

return 推荐新闻序列 S;

2 代码说明

2.1 环境与依赖

本次实验的编程语言是 Python 3.8.18，所用第三方库包括:

- pandas 2.0.3
- numpy 1.24.3
- sklearn 1.3.0
- nltk 3.8.1（文本处理）
- gensim 4.3.2 （LDA、Doc2Vec）
- tqdm 4.65.0
- matplotlib 3.7.1

2.2 代码思路

代码的总体思路分为模型选择和结果预测两步。

模型选择时，将**训练数据**划分为**训练集**和**验证集**。在**训练集**和**验证集**上获得模型，在**训练集**上计算用户平均内容偏好，在**验证集**上计算用户历史内容偏好。然后在**验证集**上计算 NDCG@10 以选择合适的模型超参数.

结果预测时，使用选定的超参数，在所有**训练数据**和**测试数据**上获得模型，并在**测试数据**上输出预测结果。

下面主要以算法 1 的实现为例说明核心代码，算法 2 在模型训练和数据准备方面的代码与算法 1 差异较大，也将说明。参数选择、结果可视化均复用作业 1、2 的代码，因此不再展示。主要函数如下:

- 数据预处理 `merge_title_abstract`和`preprocess_text`等
- LDA 模型的预测和验证 `pred_news_series`和`get_ndcg`
- Doc2Vec 模型的预测和验证 `pred_news_series_d2v`和`get_ndcg_d2v`

2.3 核心代码

2.3.1 merge_title_abstract 和 preprocess_text

本节包含数据的初步处理模块，处理完的数据适合用于模型训练和预测。

由于 title 和 abstract 的文本篇幅较长，希望能有更多数据进行建模和分析，因此每篇新闻将 title 与 abstract 拼接作为新闻文本。

```
def merge_title_abstract(row):
    # 如果title为空
    if pd.isna(row['Title']):
        return row['Abstract']
    # 如果abstract为空
    elif pd.isna(row['Abstract']):
        return row['Title']
    # 如果均非空
    else:
        return row['Title'] + ' ' + row['Abstract']
```

拼接后的文本只是字符串，还需要经过文本预处理，得到 tokens 列表。

```
def preprocess_text(row):
    # 设置停用词
    stop_words = stopwords.words('english')
    # 设置词干函数
    wnl = WordNetLemmatizer()
    # 对于text列进行处理
    text = row["text"]
    # 大小写
    text = text.lower()
    # 英文标点符号
    for i in string.punctuation:
        text = text.replace(i, ' ')
    # 数字
    text = re.sub('[\d]','',text)
    # 分词
    text = text.split()
    # 去除停用词
    text = [w for w in text if not w in stop_words]
    # 词干化
    text = [wnl.lemmatize(w) for w in text]
```

```
    return text
```

此外，为了让数据变成容易理解和处理的结构，分别对几个原文件也进行了处理，并另存为 pickle 格式文件，保证数据再次读取后格式的一致性。

处理了两个 news_info 文件，主要是利用前面的两个函数进行处理：

```
# 读入新闻信息数据
train_news_info = pd.read_csv("train_news_info.csv")
test_news_info = pd.read_csv("test_new_info.csv")
# 训练和测试集拼接去重
news_info = pd.concat([train_news_info, test_news_info],axis=0)
news_info = news_info.drop_duplicates(subset = "news_id").reset_index()
# 将title和abstract拼接并进行文本预处理
news_info['text'] = news_info.apply(merge_title_abstract, axis=1)
news_info['words'] = news_info.apply(preprocess_text, axis=1)
merged_info = news_info[["news_id","Category","SubCategory","words"]]
# 存储
merged_info.to_pickle("merged_info.pkl")
```

处理了两个 history 文件，主要是将同一个字段的新闻存为字典，便于后续读取。

```
# 读入训练用户数据
train_history = pd.read_csv("train_history.csv")
# 将新闻提取成列表或字典
train_history['history'] = train_history['history'].apply(lambda x: x.split() if
                                        isinstance(x, str) else [])
train_history['impressions'] = train_history['impressions'].apply(lambda x:
    {item.split('-')[0]: int(item.split('-')[1]) for item in x.split() if
                                            isinstance(x, str) and '-' in item}
                                            if isinstance(x, str) else {})
# 存储
train_history.to_pickle("train_history_processed.pkl")

# 读入测试用户数据
test_history = pd.read_csv("test_history.csv")
# 将新闻提取成列表
test_history['history'] = test_history['history'].apply(lambda x: x.split() if
                                        isinstance(x, str) else [])
test_history['impressions'] = test_history['impressions'].apply(lambda x: x.
                                        split() if isinstance(x, str) else [])
```

```
# 存储
test_history.to_pickle("test_history_processed.pkl")
```

还需要将数据处理为适合 LDA 模型训练和 Doc2Vec 模型训练的格式，在此略过。

2.3.2 get_ndcg 和 pred_news_series

`get_ndcg`函数完成了算法 1 的前半段：根据输入的数据和超参数训练 LDA 模型，并预处理计算用户主题偏好和新闻主题，以便于后续进行预测。

先训练 LDA 模型：

```
def get_ndcg(train_history, validation_history, all_corpus, num_topics = 18, k =
                                                10):
    # 训练模型
    lda = LdaModel(all_corpus, num_topics, random_state=42)
    print("LDA model training success")
```

再计算每条新闻的主要主题，避免后续重复计算。

```
    # 预处理：为每个新闻ID计算其主题分布和主要主题
    news_to_main_topic = {}
    for index, row in tqdm(merged_info.iterrows(), desc="Processing news", total
                                                =len(merged_info)):
        news_id = row['news_id']
        distribution = lda[all_corpus[index]]
        max_topic = max(distribution, key=lambda x: x[1])[0]
        news_to_main_topic[news_id] = max_topic
    # 保存到 pickle 文件
    filename = "lda"+str(num_topics)
    with open(filename+"_main_topic.pkl", "wb") as f:
        pickle.dump(news_to_main_topic, f)
```

计算用户群体的偏好和每个用户的历史偏好，以主题个数来统计：

```
    # 预处理：用户群体偏好、每个用户的历史信息
    ## 训练集用户（历史、impression）的平均情况
    all_distribution = defaultdict(float)
    user_history = defaultdict(float)
    for _, row in tqdm(train_history.iterrows(), total=len(train_history), desc=
                                                "Preprocessing train user"):
```

```
        user = row["user_id"]
        if(not user in user_history): # 新用户
            user_history[user] = defaultdict(float)
            # 历史都是正例
            history = row["history"]
            for item in history:
                max_topic = news_to_main_topic[item]
                all_distribution[max_topic] += 1
                user_history[user][max_topic] += 1
        # impression有正有负
        impression = row["impressions"]
        for key in impression:
            if(impression[key] == 1):
                max_topic = news_to_main_topic[key]
                all_distribution[max_topic] += 1
                user_history[user][max_topic] += 1
    ## 测试集用户（历史）的平均情况
    for _, row in tqdm(validation_history.iterrows(), total=len(
                                            validation_history), desc="
                                            Preprocessing val user"):
        user = row["user_id"]
        if(not user in user_history): # 新用户
            user_history[user] = defaultdict(float)
            # 历史都是正例
            history = row["history"]
            for item in history:
                max_topic = news_to_main_topic[item]
                all_distribution[max_topic] += 1
                user_history[user][max_topic] += 1
    # 保存到 pickle 文件
    with open(filename+"_all_distribution.pkl", "wb") as f:
        pickle.dump(all_distribution, f)
    with open(filename+"_user_history.pkl", "wb") as f:
        pickle.dump(user_history, f)
```

最后，基于以上数据，调用接下来要分析的`pred_news_series`函数，预测验证集并计算ndcg@10 的均值.

```
    # 评估ndcg表现
    ndcgs = []
    for _, row in tqdm(validation_history.iterrows(), total=len(
```

```
                                                validation_history), desc="
                                                Processing validation data"):
        ndcgs.append(pred_news_series(row, news_to_main_topic, all_distribution,
                                                    user_history, predict = 0))
    avg_ndcg = sum(ndcgs)/len(ndcgs)

    return avg_ndcg
```

`pred_news_series`函数基于计算得到的用户主题偏好和新闻主题，对用户进行预测，是算法 1 的后半段。此外，为了在模型选择步和模型预测步对函数进行复用，增加了选项参数 predict.

先根据是预测还是选模型，从数据中提取所需元素。

```
def pred_news_series(row, news_to_main_topic, all_distribution, user_history,
                                            predict = 0, k = 10):
    user = row["user_id"]
    # 如果做预测
    if(predict == 1):
        impression = row['impressions']
    # 如果选模型
    elif(predict == 0):
        real_impression = row['impressions']
        impression = list(real_impression.keys())
    else:
        print("error")
        return 0
```

然后根据用户的偏好，对新闻进行排序推荐。

```
    # 获取主题偏好
    if((not user in user_history) or (user_history[user] == defaultdict(float)))
                                                :
        user_distribution = all_distribution
    else:
        user_distribution = user_history[user]
    # 根据value排序
    sorted_interest = sorted(user_distribution.items(), key=lambda item: (-item[
                                                1], item[0]))
    # 根据主题偏好对新闻进行排序
    sorted_news = sorted(impression, key=lambda item: next((v for v, _ in
```

```
                                                sorted_interest if
                                                news_to_main_topic[item] == v),
                                                float('-inf')), reverse=True)
```

最后输出预测结果，或者返回 ndcg 值。

```
    # 如果做预测
    if(predict == 1):
        pred_news = " ".join(sorted_news)
        return pred_news
    # 如果选模型
    elif(predict == 0):
        interaction = [real_impression[item] for item in sorted_news]
        rank = [len(sorted_news) - i for i in range(len(sorted_news))]
        y_true = np.asarray([interaction[:k]])
        y_score = np.asarray([rank[:k]])
        return ndcg_score(y_true, y_score)
    else:
        print("error")
        return 0
```

2.3.3 get_ndcg_d2v 和 pred_news_series_d2v

此外，还实现了算法 2 的代码。本节代码在上一节代码的基础上修改而来，因此只展示不同的部分。`get_ndcg_d2v`函数实现了算法 2 的前半部分，在模型训练上有差别：

```
def get_ndcg_d2v(train_history, validation_history, tagged_documents,
                                        merged_info, vector_size_ = 20,
                                        min_count_ = 2, epochs_ = 10):
    # 创建模型
    model = Doc2Vec(vector_size = vector_size_, min_count = min_count_, epochs=
                                            epochs_, workers = 1)
    model.build_vocab(tagged_documents)
    # 训练模型
    model.train(corpus_iterable=tagged_documents, total_examples=model.
                                            corpus_count, epochs=model.epochs)
    # 模型保存
    filename = "./d2v/d2v_" + str(vector_size_) + "_" + str(min_count_) + "_" +
                                            str(epochs_)
    model.save(filename + ".model")
```

每个新闻的向量表示，使用模型的 model.dv 函数即可得到：

```
#预处理：为每个新闻ID计算其向量表示
news_to_topic_dist = {}
for index, row in tqdm(merged_info.iterrows(), desc="Processing news", total
                                                =len(merged_info)):
    news_id = row['news_id']
    distribution = model.dv[index]
    news_to_topic_dist[news_id] = distribution
```

此外，在遍历训练集和验证集得到用户群体偏好和历史偏好时，不需要取出最大分量的下标，而是直接将向量表示加和即可，如：

```
for item in history:
    all_distribution += news_to_topic_dist[item]
    user_history[user] += news_to_topic_dist[item]
```

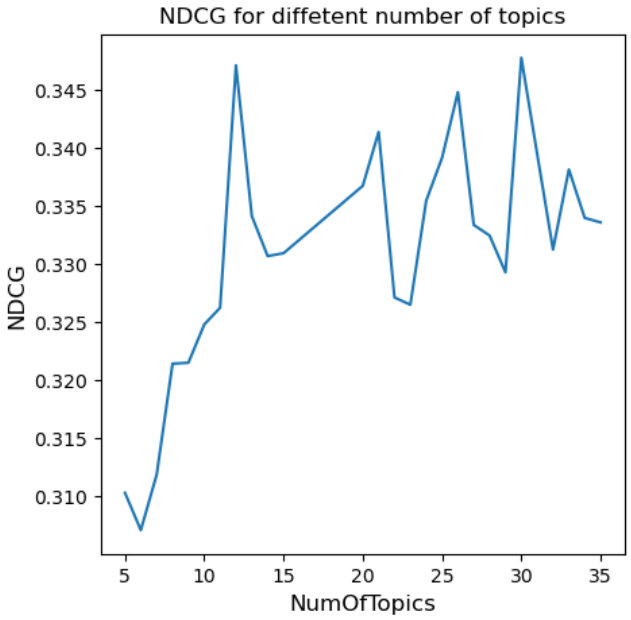
`pred_news_series_d2v`函数实现了算法 2 的后半部分，区别主要在于排序之前需要计算余弦相似度：

```
# 获取用户历史偏好
if((not user in user_history) or (user_history[user] == np.zeros(length))):
    user_vector = all_distribution
else:
    user_vector = user_history[user]

# 待推荐的新闻信息向量
item_vectors = [news_to_topic_dist[item] for item in impression]

# 计算相似度
similarities = cosine_similarity([user_vector], item_vectors)

# 根据相似度进行倒序排列，并返回排序后的待推荐物品向量
sorted_indices = np.argsort(similarities)[::-1]
sorted_news = [impression[i] for i in sorted_indices[0]]
```

3 结果分析

3.1 参数选择

先来分析算法 1 的模型性能。LDA 模型的超参数仅有主题个数 num_topics，接下来使用验证集方法估计测试集上的 NDCG，绘制 NDCG 折线图结果见图 1。推荐序列的 NDCG 指标取值范围在 $[0, 1]$，NDCG 越大，表明模型表现越好。



(a)

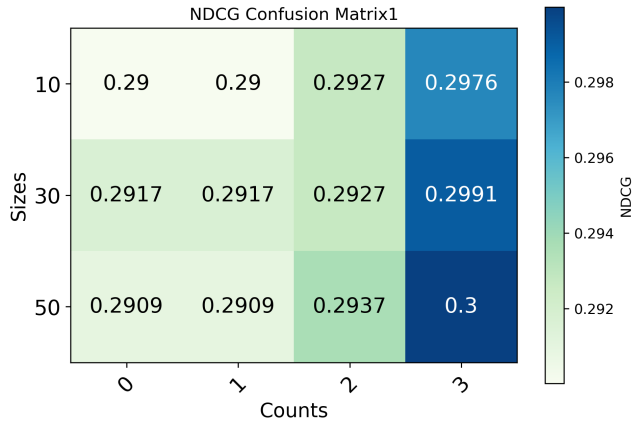
图 1: LDA 模型不同参数下的 NDCG 折线图

原始数据中"Category" 字段的不同值有 18 个，而此处在 num_topics=30 时出现了局部最优情况，NDCG@10 为 0.3475. 考虑到笔记本性能和合理的主题数量，不再进行 num_topics>35 的尝试，故选定 **num_topics=30**.

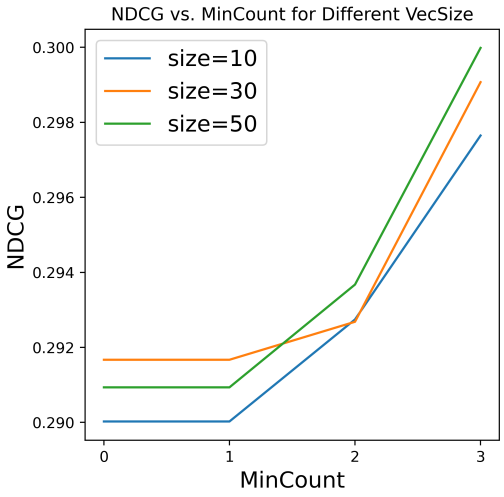
再来分析算法 2 的模型性能。Doc2Vec 的模型超参数包含向量大小 *size*, 最小词频 *count*, 迭代轮数 *epoch*. 其中，源码建议 *epoch* 取值为 10，最小词频表示在训练模型时考虑的词至少要在所有文本中出现 *count* 次，考虑到文本不包含正文，这个值不应该取太小。因此实验 1 取 *epoch*=10，并在 *size*=10、30、50，*count*=0、1、2、3 的情况下进行实验。绘制 NDCG 热力图和折线图如图 2(a)(b)，发现模型在 *count*=3、*size*=50 处取得局部最优值 0.3，但是离算法 1 的最优 NDCG@10=0.36 相差太远。

为此，考虑到可能是迭代轮数太少导致模型未收敛，因此设置迭代轮数 *epoch*=100，进行实验 2. 取 *epoch*=100，并在 *size*=10、30、50、70、100，*count*=1、2、3、4 的情况下进行实验，

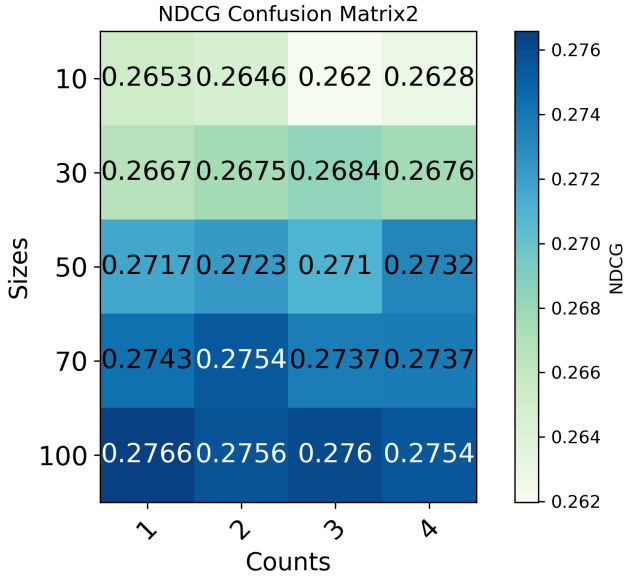
绘制 NDCG 热力图和折线图如图 2(c)(d)，发现模型在 *count*=1、*size*=100 处取得局部最优值 0.2766，比算法 1 和算法 2 的实验 1 都相差太远。因此不再考虑该模型。



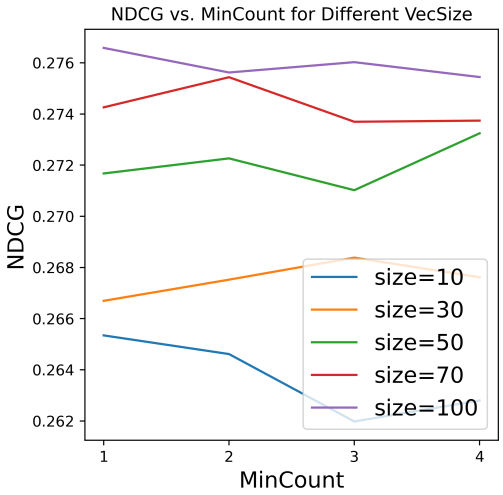
(a) 实验 1 的 NDCG 热力图



(b) 实验 1 的 NDCG 折线图



(c) 实验 2 的 NDCG 热力图



(d) 实验 2 的 NECG 折线图

图 2: D2V 模型的效果

3.2 预测结果

使用 **num_topics=30** 的 LDA 模型，模型表现如下：

- 训练数据上的 $\hat{\text{NDCG}} = 0.3477$，本地模型训练时间为 2m.
- 使用完整训练数据进行训练，在 Kaggle 平台测试集上 NDCG = 0.366（Kaggle id：Na-Cloudy），本地模型训练时间为 2m.

4 提交文件列表

压缩包`宋朝芸_10215001419_3`包含以下这些内容，部分文件夹为空是因为文件夹上传大小有要求，已经清空。但为了体现作业逻辑，仍然保留文件夹：

```
|— project_report.pdf
|— prediction_results
|       5.3-exp-lda30-0.3478.csv
|— source_code
    |— d2v        # Doc2Vec模型（算法2）
    |   |— exp    # 实验结果和可视化
    |   |       Confusion_ndcg_d2v.png
    |   |       Confusion_ndcg_d2v_2.png
    |   |       exp1_ndcg.csv
    |   |       exp2_ndcg.csv
    |   |       Lines_ndcg_d2v.png
    |   |       Lines_ndcg_d2v_2.png
    |   |— model # 存储了实验中所有不同超参数值模型及其中间结果，已清空
    |— lda        # LDA模型（算法1）
    |   |— exp    # 实验结果和可视化
    |   |       LDA-output.png
    |   |       lda_10_exp1_ndcg.csv
    |   |— model  # 存储了实验中所有不同超参数值模型及其中间结果，已清空
    |   |— result # 预测结果
    |           5.3-exp-lda30-0.3478.csv
    |— original_data # 原始数据，已清空
    |— preprocess  # 数据预处理后的中间结果，已清空
```

5 参考文章

[1] 薛定谔没养猫, 《NLP 中的 Tokenization》, 知乎.

[2] 刘启林, 《LSA 潜在语义分析的原理、公式推导和应用》, 知乎.

[2] 华年 ss, 《< 高级机器学习 > 第三讲 (下) 概率主题模型》, 知乎.