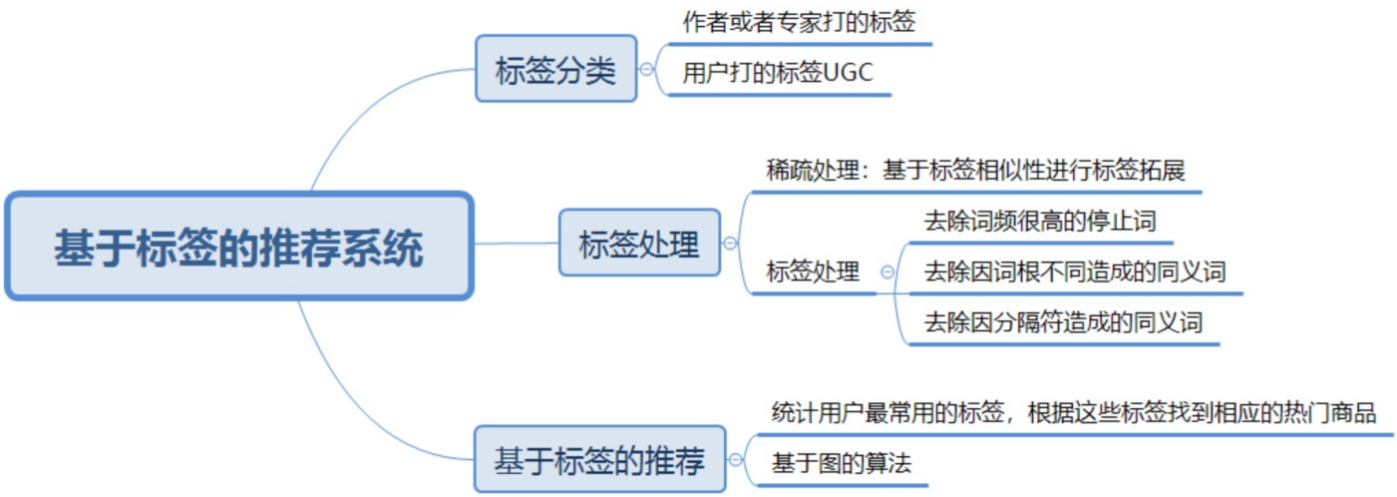
知乎 首发于 Data Analyst



推荐系统（3）-基于标签的推荐系统


Alan
数据分析、挖掘、机器学习
关注他

33 人赞同了该文章

一、基于标签的推荐系统定义

标签推荐系统系统：以标签作为媒介，构建推荐系统。

二、基于标签的推荐系统意义

可解决冷启动问题：新用户APP下载后，选取感兴趣的关注标签，系统可自动推送筛选。

例如：

- 豆瓣的电影标签、书籍标签；
- 网易云音乐的音乐标签；
- bilibili视频标签；
- 抖音等短视频APP；

关键词是指能够反映文本语料主题的词语或短语。在不同的业务场景中，词语和短语具有不同的意义。例如：

从电商网站商品标题中提取标签时，词语所传达的意义就比较突出。

从新闻类网站中生成新闻摘要时，短语所传达的意义就比较突出。

3.1数据标注

数据标注即利用人工或AI（人工智能）技术对数据（文本、图像、用户或物品）进行标注。

标注有许多类型，如:

分类标注：即打标签，常用在图像、文本中。一般是指，从既定的标签中选择数据对应的标签，得到的结果是一个封闭的集合。

框框标注：常用在图像识别中，如有一张环路上的行车照片，从中框出所有的车辆。

区域标注：常见于自动驾驶中。例如从一张图片中标出公路对应的区域。

其他标注：除了上述常见的标注类型外，还有许多个性化需求。例如，自动摘要、用户或商品的标签（因为其中总有一些未知标签，当然也可以看成是多分类）。

数据标注的一般步骤为：

（1）确定标注标准：设置标注样例和模板（如标注颜色时对应的比色卡等）。对于模棱两可的数据，制定统一的处理方式。

（2）确定标注形式：标注形式一般由算法人员确定。例如，在垃圾问题识别中，垃圾问题标注为1，正常问题标注为0。

（3）确定标注方法：可以使用人工标注，也可以针对不同的标注类型采用相应的工具进行标注。

3.2标签的分类

在推荐系统中，不管是数据标注还是关键词提取，其目的都是得到用户或物品的标签。但是在不同场景下，标签的具体内容是不定的。例如，同样是分类标注，新闻的类别里可以有军事、科技等，但音乐的类别里就很少会涉及军事或科技了。

- 标识对象的创建者或所有者。例如博客文章的作者署名、论文的作者署名等。
- 标识对象的品质和特征。例如“有趣” “幽默”等。
- 用户参考用到的标签。例如“myPhoto” “myFavorite”等。
- 分类提炼用的标签。用数字化标签对现有分类进一步细化，如一个人收藏的技术博客，按照难度等级分为“1” “2” “3” “4”等。
- 用于任务组织的标签。例如“to read” “IT blog”等。

当然以上7种类别标签是一个通用框架，在每一个具体的场景下会有不同的划分。

3.3基于TF-IDF提取标题中的关键词

TF-IDF（Term Frequency–Inverse Document Frequency）是一种用于资讯检索与文本挖掘的常用加权技术。TF-IDF算法的主要思想是：如果某个词或短语在一篇文章中出现的频率TF高，并且在其他文章中很少出现，则认为此词或短语具有很好的类别区分能力，适合用来分类。TF-IDF实际是TF*IDF。

四、基于标签的推荐系统原理

标签是用户描述、整理、分享网络内容的一种新的形式，同时也反映出用户自身的兴趣和态度。标签为创建用户兴趣模型提供了一种全新的途径。

基于标签的用户如何进行兴趣建模

4.1标签评分算法

用户对标签的认同度可以使用二元关系表示，如“喜欢”或“不喜欢”；也可以使用“连续数值”表示喜好程度。

二元表示方法简单明了，但精确度不够，在对标签喜好程度进行排序时，也无法进行区分。所以，这里选用“连续数值”来表达用户对标签的喜好程度。

为了计算用户对标签的喜好程度，需要将用户对物品的评分传递给这个物品所拥有的标签，传递的分值为物品与标签的相关度。

4.2用户对标签的依赖程度

如图6-13所示，用户u对艺术家A的评分为5星，对艺术家B的评分为3星，对艺术家C的评分为4

艺术家B与标签1、2、3的相关度分别为：0.3，0.6，0.9；

艺术家C与标签1、2、3的相关度分别为：0.5，0.7，0.6。

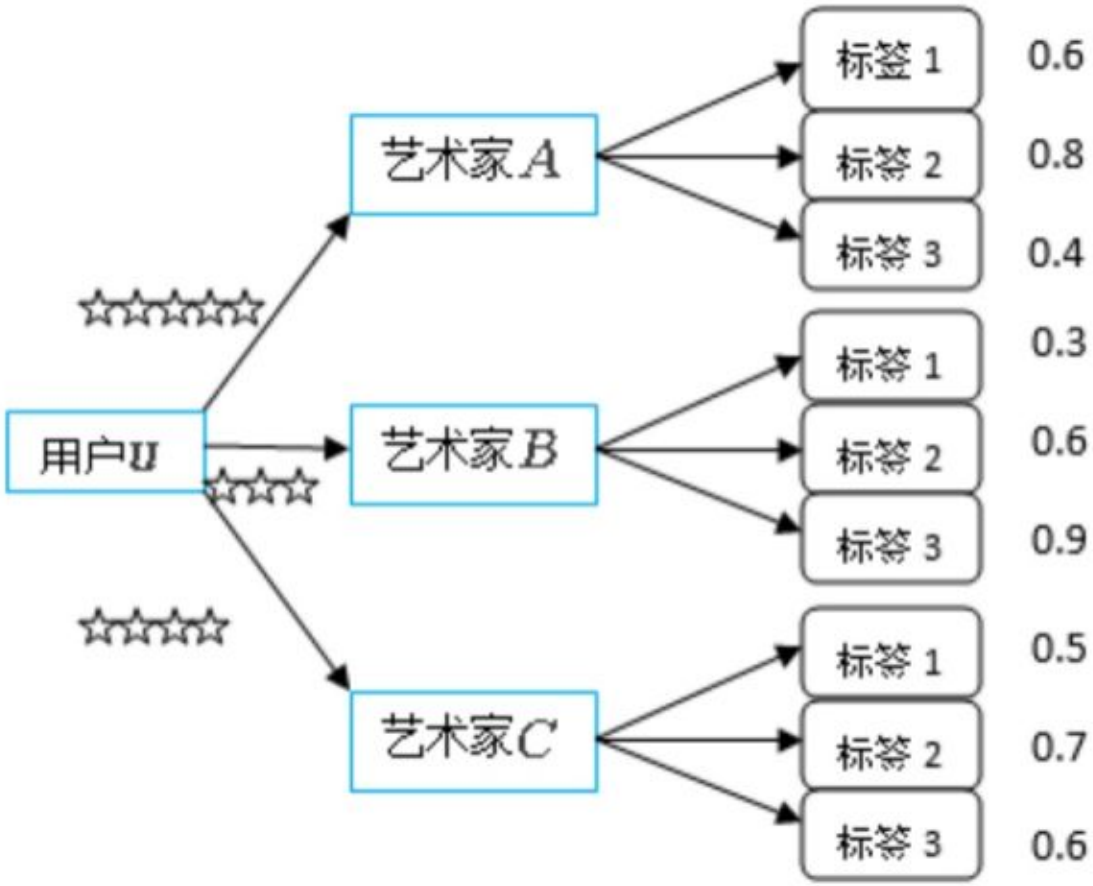


图 6-13 用户 u 对艺术家的评分和艺术家与标签的相关度

对应的用户（u）对标签（t）的喜好程度计算公式为：

$$\text{rate}(u,t)=\frac{\sum_{i\in I_u}\text{rate}(u,i)\times\text{rel}(i,t)}{\sum_{i\in I_u}\text{rel}(i,t)}$$

式中：

根据式（6.4）计算出用户u对标签1的喜好程度为：

（50.6+30.3+40.5）/（0.6+0.3+0.5）=4.21

同理可以计算出用户u对标签2的喜好程度为 4.10，对标签3的喜好程度为3.74。

4.3优化用户对标签的喜好程度

如果一个用户的评分行为较少，就会导致预测结果存在误差。那么该如何改进呢?

标签评分算法改进

这里使用**TF-IDF算法**来计算每个标签的权重，用该权重来表达用户对标签的依赖程度。

TF-IDF算法在6.3.1节中进行了介绍，这里不再赘述。每个用户标记的标签对应的TF值的计算公式为：

$$\mathrm{TF}(u,t)=\frac{n(u,t)}{\sum_{t_i \in T} n(u,t_i)}$$

式中：

- n(u,ti)表示用户u使用标签ti标记的次数。
- 分母部分表示用户u使用所有标签标记的次数和。
- TF(u,t)表示用户u使用标签t标记的频率，即用户u对标签t的依赖程度。

4.4优化用户对标签的依赖程度

在社会化标签的使用网站中存在“马太效应”，即热门标签由于被展示的次数较多而变得越来越热门，而冷门标签也会越来越冷门。大多数用户标注的标签都集中在一个很小的集合内，而大量长尾标签则较少有用户使用。

每个用户标记的标签对应的IDF值的计算公式为：

$$\mathrm{IDF}(u,t)=\lg\frac{\sum_{u_i\in U}\sum_{t_j\in T}n(u_i,t_j)}{\sum_{u_i\in U}n(u_i,t)+1}$$

- 分子表示所有用户对所有标签的标记计数和。
- 分母表示所有用户对标签t的标记计数和。
- IDF(u, t)表示t的热门程度，即一个标签被不同用户使用的概率。

对于一个标签而言，如果使用过它的用户数量很少，但某一个用户经常使用它，说明这个用户与这个标签的关系很紧密。

用户对标签的兴趣读

综合式上述，用户对标签的依赖度为：

$$\text{TF-IDF}(u,t)=\mathrm{TF}(u,t)\times\mathrm{IDF}(u,t)$$

在之前分析了用户对标签的主观喜好程度，本节分析了用户对标签的依赖程度，综合可以得到用户u对标签的兴趣度为：

$$\mathrm{Pre}(u,t)=\mathrm{rate}(u,t)\times\text{TF-IDF}(u,t)$$

4.5标签基因

标签基因是GroupLens研究组的一个项目。

在社会化标签系统中，每个物品都可以被看作与其相关的标签的集合，rel(i, t)以从0（完全不相关）到1（完全正相关）的连续值衡量一个标签与一个物品的符合程度。

例如图6-13中：

采用标签基因可以为每个艺术家i计算出一个标签向量rel(i)，其元素是i与T中所有标签的相关度。这里，rel(i)相当于以标签为基因描绘出了不同物品的基因图谱。形式化的表达如下：

$$rel(i)=[rel(i,t_1),rel(i,t_2),\ \cdots,rel(i,t_p)],\forall tk\in T$$

例如，图中，艺术家A的标签基因为：rel(艺术家A)=[0.6,0.8,0.4]。

选用标签基因来表示标签与物品的关系有以下三个原因：

（1）它提供了从0到1的连续数值；

（2）关系矩阵是稠密的，它定义了每个标签t∈ T与每个物品 i∈ I的相关度；

（3）它是基于真实数据构建的。

4.6用户兴趣建模

根据训练数据，可以构建所有商品的标签基因矩阵Ti和用户最终对标签的兴趣度Tu，则用户对商品的可能喜好程度为：

$$T(u,i)=T_u\times T_i^{\mathrm{T}}$$

式中：

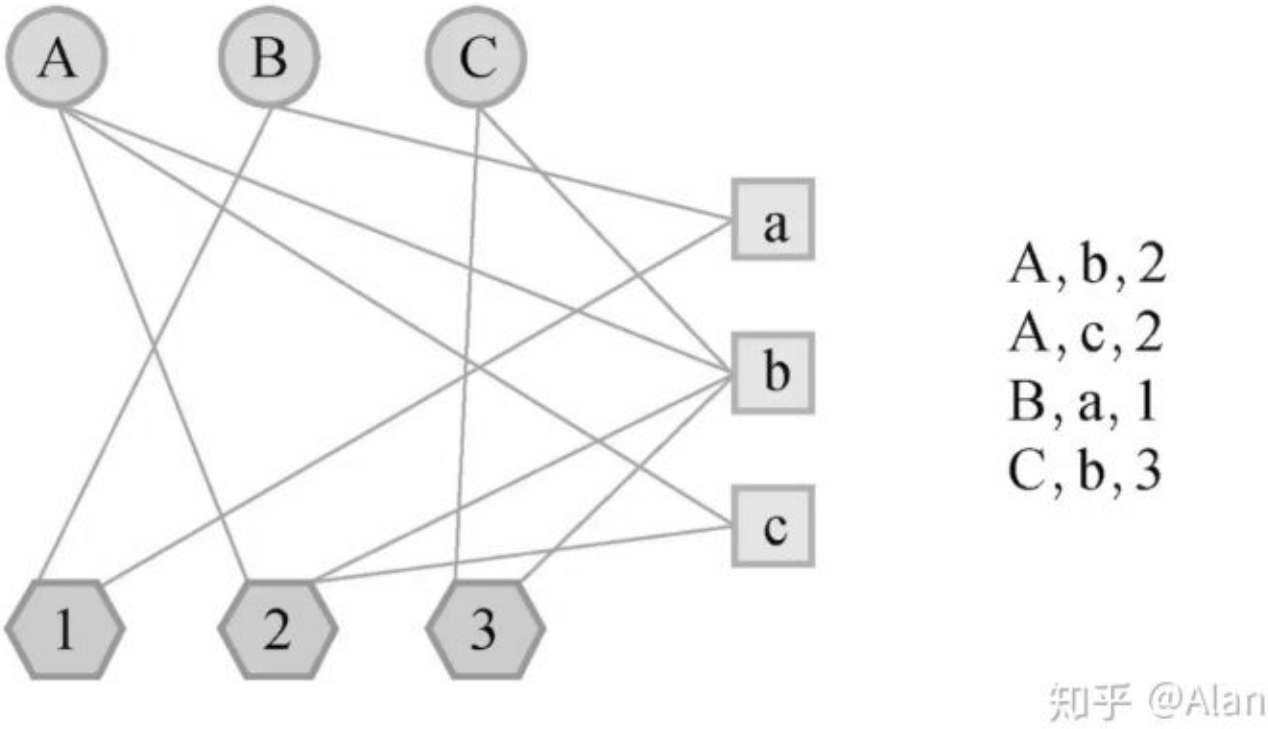
- Tu：用户u对所有标签的兴趣度矩阵（1行m列，m为标签个数）。
- Ti^T：所有商品的标签基因矩阵Ti的转置矩阵（m行n列，m为标签个数，n为商品个数）。
- T(u,i)：用户u对所有商品的喜好程度矩阵（1行n列，n为商品个数）。

最终从计算结果中选取前K个推荐给用户。

4.7补充：基于图的推荐算法（知识图谱）

首先，我们需要将用户打标签的行为表示到一张图上，我们知道，图是由顶点、边和边上的权重组

的顶点v(u)和物品i对应的顶点v(i)之间增加一条边(如果这两个顶点已经有边相连，那么就应该将边的权重加1)，同理，在v(u)和v(b)之间需要增加一条边，v(i)和v(b)之间也需要边相连接。
下图是一个简单的用户—物品—标签图的例子。该图包含3个用户(A、B、C)、3个物品(a、 b、c)和3个标签(1、2、3)。在定义出用户—物品—标签图后，我们可以用基于随机游走的PersonalRank算法计算所有物品节点相对于当前用户节点在图上的相关性，然后按照相关性从大到小的排序，给用户推荐排名最高的N个物品。



五、代码实例：基于标签推荐算法实现艺术家推荐

基于标签推荐算法实现艺术家推荐

利用标签推荐算法实现一个艺术家推荐系统，即，根据用户已经标记过的标签进行标签兴趣建模，进而为用户推荐喜好标签下最相关的艺术家。

这里使用Last.fm数据集中的数据作为基础数据，该数据集在3.3节有相关的介绍。该实例的具体实现思路如下：

（3）计算用户最终对每个标签的兴趣度；

（4）进行艺术家推荐和效果评估。

代码实例：

```
# -*-coding:utf-8-*-

"""
    Author: Alan
    Desc:
        实例：  利用标签推荐算法实现艺术家的推荐
"""

import pandas as pd
import math

class RecBasedTag:
    # 由于从文件读取为字符串，统一格式为整数，方便后续计算
    def __init__(self):
        # 用户听过艺术次数文件
        self.user_rate_file = "../data/lastfm-2k/user_artists.dat"
        # 用户打标签信息
        self.user_tag_file = "../data/lastfm-2k/user_taggedartists.dat"

        # 获取所有的艺术家ID
        self.artistsAll = list(
            pd.read_table("../data/lastfm-2k/artists.dat", delimiter="\t")["id"].value
        )
        # 用户对艺术家的评分
        self.userRateDict = self.getUserRate()
        # 艺术家与标签的相关度
        self.artistsTagsDict = self.getArtistsTags()
        # 用户对每个标签打标的次数统计和每个标签被所有用户打标的次数统计
        self.userTagDict, self.tagUserDict = self.getUserTagNum()
        # 用户最终对每个标签的喜好程度
        self.userTagPre = self.getUserTagPre()

    # 获取用户对艺术家的评分信息
    def getUserRate(self):
```

```
            if not line.startswith("userID"):
                userID, artistID, weight = line.split("\t")
                userRateDict.setdefault(int(userID), {})
                # 对听歌次数进行适当比例的缩放，避免计算结果过大
                userRateDict[int(userID)][int(artistID)] = float(weight) / 10000
        return userRateDict

    # 获取艺术家对应的标签基因,这里的相关度全部为1
    # 由于艺术家和tag过多，存储到一个矩阵中维度太大，这里优化存储结构
    # 如果艺术家有对应的标签则记录，相关度为1，否则不为1
    def getArtistsTags(self):
        artistsTagsDict = dict()
        for line in open(self.user_tag_file, "r", encoding="utf-8"):
            if not line.startswith("userID"):
                artistID, tagID = line.split("\t")[1:3]
                artistsTagsDict.setdefault(int(artistID), {})
                artistsTagsDict[int(artistID)][int(tagID)] = 1
        return artistsTagsDict

    # 获取每个用户打标的标签和每个标签被所有用户打标的次数
    def getUserTagNum(self):
        userTagDict = dict()
        tagUserDict = dict()
        for line in open(self.user_tag_file, "r", encoding="utf-8"):
            if not line.startswith("userID"):
                userID, artistID, tagID = line.strip().split("\t")[:3]
                # 统计每个标签被打标的次数
                if int(tagID) in tagUserDict.keys():
                    tagUserDict[int(tagID)] += 1
                else:
                    tagUserDict[int(tagID)] = 1
                # 统计每个用户对每个标签的打标次数
                userTagDict.setdefault(int(userID), {})
                if int(tagID) in userTagDict[int(userID)].keys():
                    userTagDict[int(userID)][int(tagID)] += 1
                else:
                    userTagDict[int(userID)][int(tagID)] = 1
        return userTagDict, tagUserDict

    # 获取用户对标签的最终兴趣度
    def getUserTagPre(self):
```

```
    for line in open(self.user_tag_file, "r", encoding="utf-8").readlines():
        if not line.startswith("userID"):
            userID, artistID, tagID = line.split("\t")[:3]
            userTagPre.setdefault(int(userID), {})
            userTagCount.setdefault(int(userID), {})
            rate_ui = (
                self.userRateDict[int(userID)][int(artistID)]
                if int(artistID) in self.userRateDict[int(userID)].keys()
                else 0
            )
            if int(tagID) not in userTagPre[int(userID)].keys():
                userTagPre[int(userID)][int(tagID)] = (
                    rate_ui * self.artistsTagsDict[int(artistID)][int(tagID)]
                )
                userTagCount[int(userID)][int(tagID)] = 1
            else:
                userTagPre[int(userID)][int(tagID)] += (
                    rate_ui * self.artistsTagsDict[int(artistID)][int(tagID)]
                )
                userTagCount[int(userID)][int(tagID)] += 1

    for userID in userTagPre.keys():
        for tagID in userTagPre[userID].keys():
            tf_ut = self.userTagDict[int(userID)][int(tagID)] / sum(
                self.userTagDict[int(userID)].values()
            )
            idf_ut = math.log(Num * 1.0 / (self.tagUserDict[int(tagID)] + 1))
            userTagPre[userID][tagID] = (
                userTagPre[userID][tagID]/userTagCount[userID][tagID] * tf_ut * id
            )
    return userTagPre

# 对用户进行艺术家推荐
def recommendForUser(self, user, K, flag=True):
    userArtistPreDict = dict()
    # 得到用户没有打标过的艺术家
    for artist in self.artistsAll:
        if int(artist) in self.artistsTagsDict.keys():
            # 计算用户对艺术的喜好程度
            for tag in self.userTagPre[int(user)].keys():
                rate_ut = self.userTagPre[int(user)][int(tag)]
```

```
                    )
                    if artist in userArtistPreDict.keys():
                        userArtistPreDict[int(artist)] += rate_ut * rel_it
                    else:
                        userArtistPreDict[int(artist)] = rate_ut * rel_it
            newUserArtistPreDict = dict()
            if flag:
                # 对推荐结果进行过滤，过滤掉用户已经听过的艺术家
                for artist in userArtistPreDict.keys():
                    if artist not in self.userRateDict[int(user)].keys():
                        newUserArtistPreDict[artist] = userArtistPreDict[int(artist)]
                return sorted(
                    newUserArtistPreDict.items(), key=lambda k: k[1], reverse=True
                )[:K]
            else:
                # 表示是用来进行效果评估
                return sorted(
                    userArtistPreDict.items(), key=lambda k: k[1], reverse=True
                )[:K]

        # 效果评估 重合度
        def evaluate(self, user):
            K = len(self.userRateDict[int(user)])
            recResult = self.recommendForUser(user, K=K, flag=False)
            count = 0
            for (artist, pre) in recResult:
                if artist in self.userRateDict[int(user)]:
                    count += 1
            return count * 1.0 / K

    if __name__ == "__main__":
        rbt = RecBasedTag()
        print(rbt.recommendForUser("2", K=20))
        print(rbt.evaluate("2"))
```

编辑于 2019-12-13