# 基于内容的电影推荐系统

2120150987 郭媛　　2120151010 林仁俊
2120151074 周盼　　2120151070 赵树阳

**摘要：**互联网的飞速发展产生了＂信息过载＂问题，人们获取信息数量的爆炸性增长使得电影观众群体受到“信息迷航”问题的困扰。为解决这一问题，电影推荐系统应运而生。文章针对该系统的关键部分即电影特征值提取和用户画像做了深入的研究。采用了 TFIDF 进行电影分词及特征值提取，将电影用空间向量模型表示并利用 PU Learning 来解决用户画像时负反馈数据难以得到的问题。最后以证明了该方法的可行性。

**关键词：**推荐系统；特征抽取；词频-逆文档概率；用户画像；负反馈数据；PU 学习。

## I. 简介

随着网络信息量的爆炸性增长，推荐系统成为研究热点，个性化电影推荐得到了人们的重视，个性化电影推荐系统纷纷出现。目前比较主流推荐算法有基于协同过滤的推荐和基于内容的推荐等。由于协同过滤是根据用户对电影的访问记录来进行推荐的，只有被观看过的电影才能被推荐，然而新上映的电影观看量会远远不及老电影，用户的访问矩阵会相当稀疏，这对于时效性要求比较高的电影推荐系统是相当严重缺陷，所以采用基于内容的推荐。

基于内容推荐，对分别对电影和用户建模，然后把与用户历史上观看的电影相似的电影推荐给用户。一般来说电影和用户建模有两种方式：向量空间模型和浅层语义模型。向量空间模型有词袋模型和词频-逆文档概率 TFIDF (TERM FREQUENCY INVERT DOCUMENT FREQUENCY)，浅层语义模型有概率潜在语义索引 PLSI (PROBABILISTIC LATENT SEMANTIC INDEXING)和潜在狄利克雷分布 LDA(LATENT DIRICHLET ALLOCATION)。

## II. 问题陈述

电影推荐系统中基于内容的电影特征（以下简称电影推荐特征）并不完全等同于视频检索、分割中的电影特征，后者需要对某一电影信息的完整表达，而电影推荐系统需要获取其整体性特征为个性化推荐提供依据。因此，电影推荐特征实际可以看作个体电影相对于整个电影库的个性描述，此外还要求特征维数不能太大，以保证在线处理的能力。

介绍的算法首先通过设定电影的内容特征，构建电影库的特征记录集；之后采用分形理论对记录集进行属性约简，得到 n 维矢量构成电影推荐特征向量，最后定义向量之间的距离度量获得不同电影之间的相似度，实现视频的归类和推荐。电影推荐系统流程图如下。

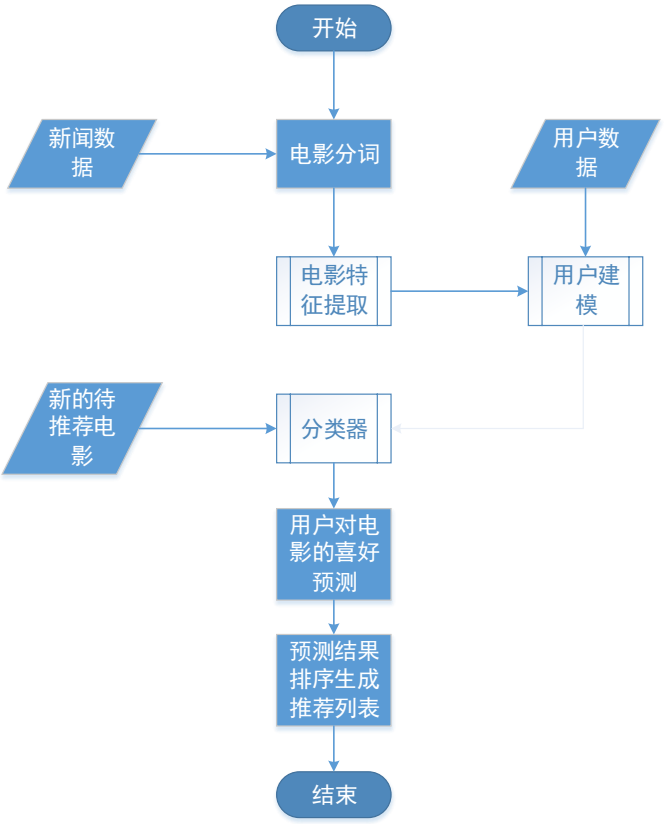


图 1 电影推荐系统流程图

### A. *电影特征提取*

大多数的基于内容的推荐系统在处理 item 特征时，都会尝试将内容映射到量空间模型 VSM（Vector Space Model），在模型中，每一部电影都被表示为一个 n 维的向量，每一维都对应词典中的一个特征词，这时电影会被表示成为一个包含每个词的权重的向量。电影特征向量生成流程如图 2 所示。

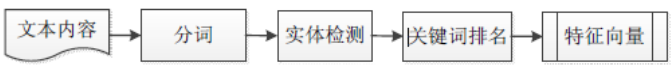


图 2 电影特征向量生成流程

为了得到电影的特征值，首先要对电影进行分词处理，将其划分成若干词条的组合。将电影表示为向量空间模型

带来了两个问题，即每个词的权重和向量之间的相似度计算，词频-逆文档概率 TF-IDF 被普遍用在处理各种自然语言的应用中来解决这个问题。

### B. 用户画像生成

学习一个用户画像就是为用户建模,在这里它可以被看作一个二值分类过程，每一个电影都被分类为喜欢和不喜欢。因此，我们有了一个分类记号 C ={c+,c-} ，其中 c+表示的是正例文本类，c- 表示的是负例文本类。我们利用用户对电影的历史数据对电影画像。

如果我们拥有用户的显式反馈，那传统的监督学习方法就能应付，但是往往用户不会提供很多显示反馈，如何使用隐式反馈来做推荐是基于内容的推荐系统的难题，半监督学习在被 Bing Liu 等发明出来用以在仅有隐式反馈时对数据进行训练。PU Learning 是一系列的概率推导，首先要生成合理的负数据，然后用这些负数据进行分类。有很多生成负数据的方法，如从无标注数据中随机的选取一些作为负数据，基于此构建分类器，如果负数据选择的足够随机，得到的效果往往都不错。

其中最有名的方法是 SPY 间谍算法，该方法将正数据中的很小一部分当作负数据来做分类，在这样的数据上应用一些常见的分类器，将无标注的数据进行分类。最后通过比较设定阈值来得到负数据。

### C. 推荐生成

通过计算前面得到的用户画像与候选电影的特征相似度，为此用户推荐一组用户喜好最相近的电影。推荐是应用用户画像中得到的分类器应用到未知电影的过程。通过将用户兴趣预测值高于某一阈值的电影推荐给用户就可以达到很好的效果。

## III. 技术方案

### A. 电影特征提取

真实应用中的 item 往往都会有一些可以描述它的属性。这些属性通常可以分为两种：结构化的（structured）属性与非结构化的（unstructured）属性。所谓结构化的属性就是这个属性的意义比较明确，其取值限定在某个范围；而非结构化的属性往往其意义不太明确，取值的限制也很少，不方便直接使用。比如在交友网站上，item 就是人，一个 item 会有结构化属性如身高、学历、籍贯等，也会有非结构化属性（如 item 自己写的交友宣言，博客内容等等）。对于结构化数据，我们自然可以直接使用；但对于非结构化数据，比如本系统中的电影，我们往往要先把它转化为结构化数据后才能在模型里加以使用。下面我们就详细介绍下如何把非结构化的内容结构化。

下面介绍的表示技术其来源也是信息检索，其名称为向量空间模型（Vector Space Model，简称 VSM）。

记我们要表示的所有电影集合为

$$D = \{d_1, d_2, ..., d_N\}$$

而所有电影中出现的片段、特征词的集合（也称为词典）为

$$T = \{t_1, t_2, ..., t_n\}$$

也就是说，我们有 N 部要处理的电影，而这些电影里包含了 n 个不同的特征词。我们最终要使用一个向量来表示一部电影，比如第 j 部电影被表示为

$$d_j = (w_{1j}, w_{2j}, ..., w_{nj})$$

其中 $w_{1j}$ 表示第 1 个特征 $t_1$ 在电影 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。所以，为了表示第 j 部电影，现在关键的就是如何计算 $d_j$ 各分量的值了。例如，我们可以选取 $w_{1j}$ 为 1，如果词 $t_1$ 出现在第 j 部电影中；选取为 0，如果 $t_1$ 未出现在第 j 部电影中。我们也可以选取 $w_{1j}$ 为词 $t_1$ 出现在第 j 部电影中的次数（frequency）。但是用的最多的计算方法还是信息检索中常用的词频-逆文档频率（term frequency－inverse document frequency，简称 tf-idf）。第 j 篇文章中与词典里第 k 个词对应的 tf-idf 为：

$$\text{TF-IDF}(t_k, d_j) = \underbrace{\text{TF}(t_k, d_j)}_{\text{TF}} \cdot \underbrace{log\frac{N}{n_k}}_{\text{IDF}}$$

其中 $\text{TF}(t_k, d_j)$ 是第 k 个特征在电影 j 中出现的次数，而 $n_k$ 是所电影中包括第 k 个特征的电影数量。

最终第 k 个特征在电影 j 中的权重由下面的公式获得：

$$w_{k,j} = \frac{\text{TF-IDF}(t_k, d_j)}{\sqrt{\sum_{s=1}^{|T|} \text{TF-IDF}(t_s, d_j)^2}}$$

做归一化的好处是不同文章之间的表示向量被归一到一个量级上，便于下面步骤的操作。

### B. 用户画像生成

当用户 u 已经对一些 item 给出了自己的喜好判断，喜欢其中的一部分 item，不喜欢其中的另一部分。那么，这一步要做的就是通过用户 u 过去的这些喜好判断，为他产生一个模型。有了这个模型，我们就可以根据此模型来判断用户 u 是否会喜欢一个新的 item。所以，这一部分要解决的是一个典型的有监督分类问题，理论上机器学习里的分类算法在此处都适用。

下面介绍常几个常用的学习算法：

1. 最近邻方法 （k-Nearest Neighbor，简称 kNN）

对于一个新的 item，最近邻方法首先找用户 u 已经评判过并与此新 item 最相似的 k 个 item，然后依据用户 u 对这 k 个 item 的喜好程度来判断其对此新 item 的喜好程度。这种做法和 CF 中的 item-based kNN 很相似，差别在于这里的 item 相似度是根据 item 的属性向量计算得到，而 CF 中是根据所有用户对 item 的评分计算得到。

对于这个方法，比较关键的可能就是如何通过 item 的属性向量计算 item 之间的两两相似度。对于结构化数据，相似度计算可以使用欧几里得距离；而如果使用向量空间模型（VSM）来表示 item 的话，则相似度计算可以使用 cosine。

2. Rocchio 算法

Rocchio 算法是信息检索中处理相关反馈（Relevance Feedback）的一个著名算法。比如在搜索引擎里搜“苹果”，最开始搜索这个词时，搜索引擎不知道你到底是要能吃的水果，还是要不能吃的苹果，所以它往往会尽量呈现给你各种结果。当用户看到这些结果后，会点击一些觉得相关的结果（这是相关反馈的一个过程）。在用户翻页查看第二页的结果时，搜索引擎可以通过用户给的相关反馈，修改查询向量取值，重新计算网页得分，将与刚才点击的结果相似的结果排前面。例如，最开始搜索“苹果”时，对应的查询向量是{“苹果”:1}。而当你点击了一些与 Mac、iPhone 相关的结果后，搜索引擎会把你的查询向量修改为{“苹果”:1,“Mac”:0.8,“iPhone”:0.7}，通过这个新的查询向量，搜索引擎就能比较明确地知道用户所搜索的词的实际语义。Rocchio 算法的作用就是用来修改这一查询向量：{“苹果”:1} --> {“苹果”:1,“Mac”:0.8,“iPhone”:0.7}。

在基于内容的推荐系统中，可以类似地使用 Rocchio 算法来获得用户 u 的画像 $\overrightarrow{w_u}$：

$$\overrightarrow{w_u} = \beta \cdot \frac{1}{|I_r|} \sum_{\overrightarrow{w_j} \in I_r} \overrightarrow{w_j} - \gamma \cdot \frac{1}{|I_{nr}|} \sum_{\overrightarrow{w_k} \in I_{nr}} \overrightarrow{w_k}$$

其中 $\overrightarrow{w_j}$ 表示 item j 的属性，$I_r$ 与 $I_{nr}$ 分别表示已知的用户 u 喜欢与不喜欢的 item 集合；而 $\beta$ 与 $\gamma$ 为正负反馈的权重，它们的值由系统给定。

在获得 $\overrightarrow{w_u}$ 后，对于某个给定的 item j，我们可以使用 $\overrightarrow{w_u}$ 与 $\overrightarrow{w_j}$ 的相似度来代表用户 u 对 j 的喜好度。

Rocchio 算法的一个好处是 $\overrightarrow{w_u}$ 可以根据用户的反馈实时更新，其更新代价很小。

3. 决策树算法（Decision Tree，简称 DT）

当 item 的属性较少而且是结构化属性时，决策树一般会是个好的选择。这种情况下决策树可以产生简单直观、容易让人理解的结果。而且我们可以把决策树的决策过程展示给用户 u，告诉他为什么这些 item 会被推荐。但是如果 item 的属性较多，且都来源于非结构化数据（如 item 是文章），那么决策树的效果可能并不会很好。

4. 线性分类算法（Linear Classifer，简称 LC）

对于这里的二类问题，线性分类器（LC）尝试在高维空间找一个平面，使得这个平面尽量分开两类点。也就是说，一类点尽可能在平面的某一边，而另一类点尽可能在平面的另一边。

仍以学习用户 u 的分类模型为例。$\overrightarrow{w_j}$ 表示 item j 的属性向量，那么 LC 尝试在 $\overrightarrow{w_j}$ 空间中找平面 $\overrightarrow{c_u} \cdot \overrightarrow{w_j}$，使得此平面尽量分开用户 u 喜欢与不喜欢的 item。其中的 $\overrightarrow{c_u}$ 就是我们要学习的参数了。最常用的学习 $\overrightarrow{c_u}$ 的方法就是梯度下降法了，其更新过程如下：

$$\overrightarrow{c_u}^{(t+1)} := \overrightarrow{c_u}^{(t)} - \eta(\overrightarrow{c_u}^{(t)} \cdot \overrightarrow{w_j} - y_{uj})\overrightarrow{w_j}$$

其中的上角标 t 表示第 t 次迭代，$y_{uj}$ 表示用户 u 对 item j 的打分（例如喜欢则值为 1，不喜欢则值为-1）。$\eta$ 为学习率，它控制每步迭代变化多大，由系统给定。

5. 朴素贝叶斯算法（Naive Bayes，简称 NB）

NB 经常被用来做文本分类，它假设在给定一篇文章的类别后，其中各个词出现的概率相互独立。用户画像生成问题中包括两个类别：用户 u 喜欢的 item，以及他不喜欢的 item。在给定一个 item 的类别后，其各个属性的取值概率互相独立。我们可以利用用户 u 的历史喜好数据训练 NB，之后再用训练好的 NB 对给定的 item 做分类。

*C. 推荐生成*

如果上一步用户画像生成中使用的是分类模型（如 DT、LC 和 NB），那么我们只要把模型预测的用户最可能感兴趣的 n 个 item 作为推荐返回给用户即可。而如果用户画像生成中使用的直接学习用户属性的方法（如 Rocchio 算法），那么我们只要把与用户属性最相关的 n 个 item 作为推荐返回给用户即可。其中的用户属性与 item 属性的相关性可以使用如 cosine 等相似度度量获得。

## IV. 实现和实验结果

为了便于实现，首先我们需要获取训练电影资源，并将其存放到电影服务器端。然后，训练模块从电影服务器端获取训练指令，开始其训练周期。训练中需要调用内部的电影训练模块和用户训练模块。

电影训练模块需要将读取电影串，构建电影字典，完成对电影的分词，转化到词袋模型，利用 TFIDF 转化到

TagSpace,最后利用 SVD 矩阵分解完成特征向量转化。用户训练模块需要将用户的数据根据历史观看电影数据构建出用户矩阵。根据用户矩阵进行训练，构建出分类器，该分类器被用于推荐模块给用户进行推荐。

推荐模块需要及时和电影服务器进行交互，与服务器的交互模块利用的是 socket 编程，为推荐模块和训练模块分别创建一个 socket，等待电影服务器与其通信。

电影服务器将采集的待推荐电影交递给推荐模块。推荐模块利用训练模块准备好的分类器对用户进行推荐并把推荐结果存放到数据库，数据库将推荐结构反馈到电影服务器端，电影服务器端根据此给客户端以反馈。

在上一部分中介绍了一些基于内容的推荐系统用到的关键技术和基本算法，本系统在特征提取部分即用到了上面的空间向量模型构建,而用户画像生成选取了 Rocchio 算法。

**A. 关键代码**

1. 空间向量模型构建关键代码

```
public class TFIDFModel implements Serializable {
   private static final long serialVersionUID = 1L;

   private final Map<String, Long> tagIds;
   private final Map<Long, SparseVector> itemVectors;

   TFIDFModel(Map<String,Long> tagIds, Map<Long,SparseVector> itemVectors) {
      this.tagIds = tagIds;
      this.itemVectors = itemVectors;
   }

   public MutableSparseVector newTagVector() {
      return MutableSparseVector.create(tagIds.values());
   }

   public SparseVector getItemVector(long item) {
      // Look up the item
      SparseVector vec = itemVectors.get(item);
      if (vec == null) {
         // We don't know the item! Return an empty vector
         return SparseVector.empty();
      } else {
         return vec;
      }
   }
}
```

2. 用户画像生成关键代码

```
   public void score(long user, @Nonnull MutableSparseVector output) {
      // Get the user's profile, which is a vector with their 'like' for each tag
      SparseVector userVector = makeUserVector(user);

      // Loop over each item requested and score it.
      // The *domain* of the output vector is the items that we are to score.
      for (VectorEntry e: output.fast(VectorEntry.State.EITHER)) {
         // Score the item represented by 'e'.
         // Get the item vector for this item
         SparseVector iv = model.getItemVector(e.getKey());
         // TODO Compute the cosine of this item and the user's profile, store it in the output vector
         // Cosine b/n iv and userVector
         double numerator = userVector.dot(iv);
         double denominator = userVector.norm() * iv.norm();
         double cosine = numerator / denominator;
         output.set(e.getKey(), cosine);

      }
   }

   private SparseVector makeUserVector(long user) {
      // Get the user's ratings
      List<Rating> userRatings = dao.getEventsForUser(user, Rating.class);
      if (userRatings == null) {
         // the user doesn't exist
         return SparseVector.empty();
      }

      // Create a new vector over tags to accumulate the user profile
      MutableSparseVector profile = model.newTagVector();
      // Fill it with 0's initially - they don't like anything
      profile.fill(0);

      // Iterate over the user's ratings to build their profile
      double ratingSum = 0;
      int counter = 0;
      for (Rating r: userRatings) {
         // In LensKit, ratings are expressions of preference
         Preference p = r.getPreference();

         counter++;
         ratingSum += p.getValue();
      }
      double avgRating = ratingSum/counter;
      for(Rating r: userRatings){
         Preference p = r.getPreference();
         double ratingValue = p.getValue();
         double multiplier = ratingValue - avgRating;

         long itemId = r.getItemId();
         SparseVector itemVector = this.model.getItemVector(itemId);
         for(VectorEntry v: itemVector.fast()){
            long vKey = v.getKey();
            double vValue = v.getValue();
            double sum = vValue * multiplier + profile.get(vKey);
            profile.set(vKey, sum);
         }
```

}

## B. 实验结果

在本文实现的电影推荐系统中，总共有 1500 部电影，1200 个用户，总共 6500 个评分，以此为测试条件。设定不同的推荐列表长度 L，Hits 和 diversity 指标的值如下表所示。

| 推荐列表长度 L | Hits | Diversity |
|---|---|---|
| **50** | 0.7520 | 0.7232 |
| **20** | 0.7568 | 0.7126 |
| **10** | 0.7143 | 0.7012 |
| **5** | 0.6686 | 0.7731 |
| **1** | 0.5346 | 0.9293 |

表 1 不同推荐列表长度系统性能评价

从上表可以看出，推荐系统中不同列表长度 L 下系统性能是相对稳定的。

下面展示系统运行效果界面：

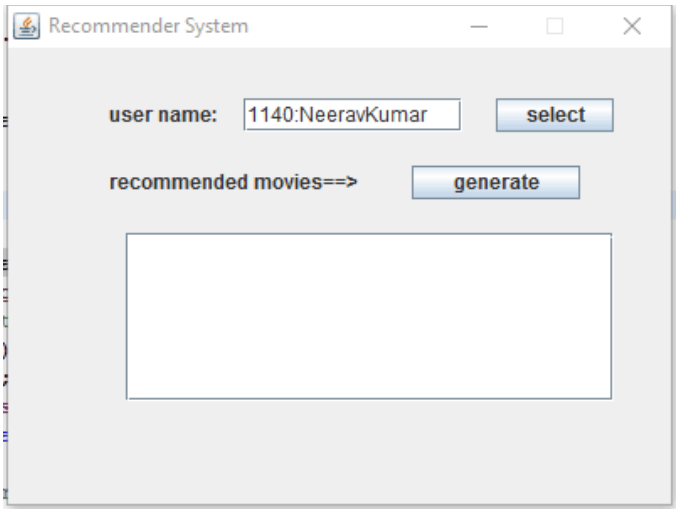


图 3 主界面

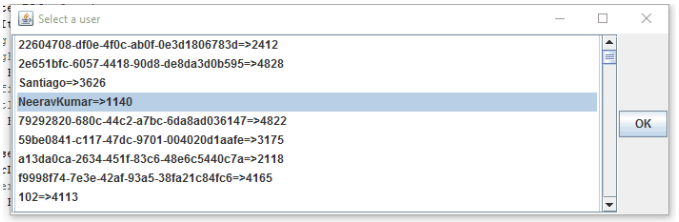


图 4 从用户列表选择

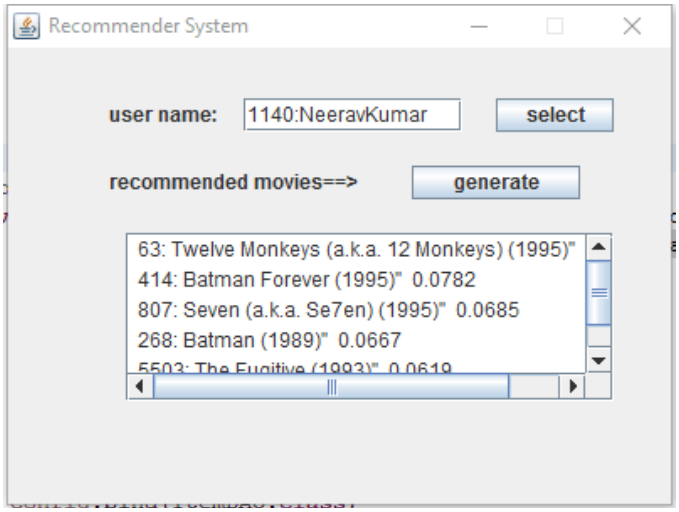


图 5 推荐结果

## 参考文献


[1] Balabanovic M, Shoham Y. Fab: Content－Based, collaborative recommendation. Communications of the ACM, 1997, 40(3): 66－72.

[2] Pazzani M,Billsus D. Content － based recommendation systems. The Adaptive Web Lecture Notes in Computer Science. Springer－Verlag, 2007, 4321: 325－341.

[3] Somlo G, Howe A. Adaptive lightweight text filtering. In: Proc. of the 4th Int′ I Symp. on Intelligent Data Analysis. Berlin, Heidelberg: Springer－Verlag, 2001: 319－329.

[4] Zhang Y, Callan J, Minka T. Novelty and redundancy detection in adaptive filtering. In: Proc. of the 25th Annual Int′ I ACM SIGIR Conf. New York: ACM Press, 2002: 81－88

[5] Robertson S. Threshold setting and performance optimization in adaptive filtering. Information Retrieval, 2002, 5 (2－3): 239-256.

[6] Zhang Y, Callan J. Maximum likelihood estimation for filtering thresholds. In: Proc. of the 24th Annual Int′ l ACM SIGIR Conf. New York: ACM Press, 2001: 294－302.

[7] Basu C, Hirsh H, Cohen W. Recommendation as classification: Using social and content －based information in recommendation. In: Proc. of the AAAI′ 98. Menlo Park: AAAI Press, 1998: 714－720.