上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

BACHELOR'S THESIS

论文题目 学术大数据中会议关系研究

学生姓名 李一玚

学生学号 5130309721

指导教师 伍民友

专　　业 计算机科学与技术

学院（系）电子信息与电气工程学院

Submitted in total fulfilment of the requirements for the degree of

Bachelor

in Computer Science

Connection between Conference in Academic Big Data

Yiyang Li

Advisor

Prof. Minyou Wu

Depart. of Computer Science and Engineering, School of Electronic Information and Electrical Engineering

Shanghai Jiao Tong University

Shanghai, P.R.China

May. 31st, 2017

上海交通大学
毕业设计（论文）学术诚信声明

本人郑重声明：所呈交的毕业设计（论文），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：_____

日　期：_____年 ____月 ____日

上海交通大学
毕业设计（论文）版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保　密 □，在 ___ 年解密后适用本授权书。

本学位论文属于

不保密 □。

（请在以上方框内打“✓”）

作者签名：_____　　指导教师签名：_____

日　期：_____年 ___月 ___日　　日　期：_____年 ___月 ___日

学术大数据中会议关系研究

摘　　要

学术搜索是指针对目前学术论文数据量十分巨大，难以直接获取所需要的文献的情况下，在现有会议论文数据中建立合适的索引和联系，以结构化的信息方便查找选择。类似传统搜索引擎，为用户提供关键字或分类查找等方便的搜索方式。

而与传统搜索引擎使用者不同的是，学术搜索引擎使用者往往需要对整个领域有一个较为宏观的掌握才能够比较有效的使用搜索功能查找所需要的文献。而在同一会议中发表的论文之间，往往具有较强的相关性。但即使在一个特定领域，也存在复数个会议。如何在会议之间进行论文的推荐就是比较困难的问题。目前国内外的这一领域里，均缺少比较深入的研究和完善的解决方案。即使是在大数据十分火热的现今，对于学术大数据的特殊研究，特别是会议之间的研究获得的关注并不多。而对于很多研究者来说，对于某个会议的论文的研究，或者是同一领域内数个会议之间论文的比较，会带来对研究领域的比较清晰和完备的认识，对研究带来收获。

在现有的学术论文数据基础上, 主要考虑利用会议之间文章的相似度来定义会议之间的关系，基于各种自然语言处理方法，例如词嵌入以及神经网络语言模型等 NLP 方法来对文章进行分析，在辅助以例如引用以及作者之间关系等信息，来实现协同过滤的推荐方法。

关键词： 学术搜索, 数据挖掘, 词嵌入, 自然语言处理, 推荐系统

Connection between Conference in Academic Big Data

ABSTRACT

The academic search refers to deal with the large amount of academic papers,

KEY WORDS: acadamic search, BigData, word embedding, NLP, recommendation system

目　　录

主要符号对照表 **vi**

第一章　绪论 **1**

1.1　学术搜索现状简介 1

1.2　系统简介 2

第二章　相关技术介绍 **6**

2.1　自然语言处理技术介绍 6

2.1.1　综述 6

2.1.2　统计语言模型与词嵌入介绍 6

2.1.3　N-gram 语言模型 8

2.1.4　Word2Vec 9

2.2　聚类分析介绍 12

2.2.1　相似性衡量 13

2.2.2　聚类算法 15

2.2.3　数据简化 17

2.3　推荐算法介绍 20

2.3.1　基于协同过滤的推荐算法 21

2.3.2　基于内容的推荐算法 24

2.3.3　其他推荐算法 28

2.4　本章总结 29

第三章 系统架构 **31**

3.1 系统环境 31

3.1.1 Tensorflow 32

3.1.2 Gephi 34

3.2 系统模块 35

3.2.1 词向量生成 35

3.2.2 词聚类与文章向量生成 40

3.2.3 会议向量生成与相关性分析 42

3.2.4 推荐系统及可视化 45

全文总结 **49**

参考文献 **50**

致谢 **51**

插图索引

2–1 Word2Vec 模型 10

3–1 工具框架示例 34
3–2 可视化全景 46
3–3 可视化细节 47

主要符号对照表

$\mathbf{V}_{word}$	词向量
$\mathbf{V}_{doc}$	文本向量
D_M	闵氏距离

第一章　绪论

1.1　学术搜索现状简介

网络信息时代下, 各个学科的学术研究成果的交流自然而然变得更加频繁且不可或缺. 其中在我们日常研究生活中最为经常应用到的部分就是灌注了其他研究者智慧果实和研究成果的学术论文. 获取一篇想要阅读的论文通常需要自费购买或者依靠于对应的学术机构集体购买. 例如 IEEE 的数据库[1]以及国内的中国知网[2]. 当然也有类似 arXiv[3]的网站可以免费阅读下载上面的论文. 然而这些数据库的问题在于其中存储的论文数据相互之间几乎没有交集. 导致如果要寻找某一篇论文有时候必须遍历数个数据库才能找到具体想要的论文. 这样的行为明显增加了学术研究中不必要的时间开销. 而类似的情况以前也曾经出现过, 那就是在互联网的发展的早期, 各个网站之间并不完全相通, 缺乏一个合适的流量引导的工具导致网络的相对闭塞. 而门户网站以至于之后的搜索引擎的出现正是填补了这个空缺, 让整个网络真正的互相连接起来了, 也方便了在互联网上浏览的所有用户. 而在网络中的学术论文这个领域, 相对应出现的工具就是学术搜索引擎了.

在当今世界上, 被应用得比较广泛, 使用者数量比较多的学术搜索引擎应该要数美国 Google 公司开发的 Google Scholar 网站[4]. 这个网站也是目前公开的, 数据量最大, 最方便的学术搜索引擎. 这个搜索引擎的使用方式和普通的网站搜索引擎类似. 支持按照关键词搜索, 通过作者检索, 通过高级搜索进行更加精确

[1] http://ieeexplore.ieee.org
[2] http://cnki.net
[3] https://arxiv.org
[4] https://scholar.google.com

地搜索等功能 [1]. Google Scholar 也提供了学术论文作者的个人资料页面, 其中可以看到该作者相关的很多信息, 包括按年份划分的被引用数以及一些统计数据等. 例如 **h-index**, 意义是被引用数超过该数字次数的文章数. 从这些个人页面也可以很快地获得对此人研究成果的大致认知. 可以说也从侧面加强了网站整体的功能性以及易用性.

与 Google Scholar 相似的, 还有微软公司的微软学术搜索[5]. 是由微软亚洲研究院 (MSRA) 开发的免费学术搜索引擎。特点是使用了对象级别的信息检索途经, 可以通过微软自己的数据库来找到某个学术领域内的领先的研究者和顶尖的会议和期刊, 甚至还可以研究领域兴盛和发展的具体过程. 还可以用来发现一些学术研究学科内的研究热点和正在上升期的学术新星等. 虽然这个搜索引擎目前主要收录了与计算机科学和信息科学相关的学术数据, 但这些更为高级的应用功能无疑是对于用户的使用体验有很大提高的.

除了这些国外公司开发的学术搜索引擎以外, 国内也有高校开发出自己的学术搜索引擎, 例如清华大学的 AMiner[6]和上海交通大学的 Acemap[7]. 这些高校搜索引擎的特点是虽然数据量无法和 Google Scholar 那样的大体量比肩, 但开发者更注重用户在使用学术搜索引擎时可能需要的一些个性化的功能. 例如 Acemap 之中, 用户可以通过可视化的方式, 由大及小很轻松的选择出自己感兴趣的学术研究领域, 来检索其中的论文. 更可以通过合作者地图, 学术机构地图等方式来贴合各种个性化的需求, 来提高寻找所需要的学术论文的效率.

1.2 系统简介

由于在实际的学术研究中, 相对于关注某个知名研究者或者研究机构的研究成果, 我们往往会更加关注某一个领域的过往以及最新的研究成果也即是在

[5] http://cn.bing.com/academic
[6] https://aminer.org
[7] http://acemap.sjtu.edu.cn

该领域发表的学术论文. 而且不同的人关注的领域往往并不会宽泛到某个一级学科甚至于二级学科, 而是具体到某个很细分的方向. 例如对于同样是关注机器学习这个领域的研究者, 有些人关注的重点在于监督学习而有些人则会对非监督学习更为感兴趣. 还有现在大火的增强学习, 也是一个截然不同的领域. 更进一步地, 在相似的一些算法上, 也有不同的侧重点在. 这样就给我们在寻找一系列感兴趣的论文时增加了难度. 仅仅依靠几个关键词无法很好的找到所需要的论文, 更无法发现一些提出的崭新的思想和概念. 而通过作者和合作者甚至于引用关系来检索也有很大的局限性, 因为每个研究者的研究兴趣和方向也是会随着时间的变化而有相应的变化, 无法仅仅通过锁定数个研究者来获得感兴趣的方向和领域的论文. 很自然地, 我们就产生了一个想法, 如何通过现有的数据, 来改善我们的学术搜索使用的体验, 来更好的去切合我们去查找某个细分领域内所有优质论文的需求呢?

这乍一看是一个并没有表面上那么简单的想法. 论文的思想和方向无法简单地通过数个关键词来定义. 而具体涉及到论文的内容, 要通过自然语言处理的方法来直接进行文本分类, 来进一步地将我们所寻找的相似领域的论文来划分到一起, 先不提其效果比较难以检验, 学术论文作为研究者的思维结晶和学术成果, 作为文本, 它的单位信息量是大于我们常见的新闻, 小说等文本类型的. 而且论文文本之中结构也十分复杂, 为了能够很好地将自己的研究成果向其他研究者阐述清楚, 研究者们往往需要在学术论文之中附上大量的图标, 数据, 算法. 这些并不标准的部分也增加了分析学术论文整个文本的难度. 目前为止, 有关于学术论文文本分类的研究比较少, 想必也有这个方面的原因. 所幸这个想法能够很自然地落到实处, 通过自然语言处理方法来进行对应的学术论文分类并不是一个非常糟糕的选择. 但具体应用到学术论文上时, 我们考虑采取一些其他的手段来辅助我们的分类.

首先考虑到学术论文的特殊性在于其结构化较为明显, 大部分发表的学术

论文, 基本上都包含了摘要, 绪论, 正文等几个部分. 这也体现了学术论文与其他文本的区别的一点在于其作者都是经过了充分训练的科学工作者, 其整体结构的严谨性自然能够得到保证. 其次则是学术论文大多通过会议和期刊来发表. 我们很自然地想到, 在学术论文从写作到投稿这一过程中, 研究者对于投稿的期刊和会议的选择, 其实就是一次人为的标签行为. 因为不同的期刊和会议, 所接受的论文的研究主题和范围都是有所限定的. 虽然有时候会随着时间的经过, 世界上研究热点的变化而稍有变化, 但依然能作为一个判定地依据. 特别是学术会议, 作为一个有时效性的活动, 其中发表的学术论文的选题和方向通常都具有很强的时代性和针对性. 与此相对的是许多期刊因为是长期发表的刊物, 其内容受时间和时代的影响就比较明显. 任何一个期刊里刊登在 2016 年的文章和 2006 年的文章关注的热点肯定是截然不同的. 很显然, 与其直接研究学术论文文本的分类, 研究学术会议整体之间的关系, 再通过相似的会议来进行学术论文的推荐就是一个比较容易且高效的方法.

那么对于一个了解数据挖掘的人来说, 这个问题就集中到如何去在机器能够认识的领域内去定义一个学术会议了, 简单地来说就是如何对学术会议去做特征提取. 这时候又有一个很自然的想法, 既然学术会议是其中学术论文的一个隐含的特征, 如果我们把学术会议看做是学术论文的集合, 那么我们显然可以通过其中包含的学术论文反过来定义会议本身. 如何定义一篇学术论文呢? 常见的文本特征抽取的方法有从简单的词频,**TF-IDF** 到 **N-gram** 算法, 模拟退火算法等较为复杂的方法. 这里我们考虑到我们最终的目的在于理清学术会议之间一个相对的关系, 注重的是相对. 所以我采取了具有很好线性特征的 Word2vec[2, 3] 来实现最终的文本向量化.

具体的做法是利用单层的神经网络语言模型 **Skip-gram** 来训练词向量 $\mathbf{V}_{word}$, 再将训练好的词向量进行聚类. 利用聚类来对文本向量化. 聚类算法也有很多中选择, 包括层次化聚类算法, 划分式聚类算法以及基于网格和密度的聚类

算法等 [4], 这里我们选择使用划分式聚类算法中的 **K-Means** 算法来进行聚类. 经过聚类之后的词向量, 每个类型都代表了学术论文中比较接近的一类词语, 这时候我们只需要简单的统计出不同类型的词语的词频, 就可以获得代表文章的向量 $\mathbf{V}_{doc}$.

得到了文本向量之后, 我们将进一步考虑利用文本向量去生成对应会议向量. 这时候我们主要考虑到会议向量应该代表了该学术会议中学术论文的主要趋势, 以及会议向量之间能够通过某种方式来比较之间的相关程度. 在上一步中, 我们采用了不同类型词语的词频来代表文本向量, 那么将一个学术会议中所有学术论文的文本向量取平均, 然后通过余弦相似度来进行比较就是一个很容易想到的方法. 实际上这种方法也确实能够得到非常令人满意的结果.

第二章　相关技术介绍

2.1　自然语言处理技术介绍

2.1.1　综述

自然语言处理是计算机科学的一个细分领域, 与人工智能和计算语言学息息相关. 自然语言处理主要关注的重点在于人机交互之中的语言交互, 也即是如何让计算机能够理解人类日常使用的语言文字. 细分领域包括了底层的词法分析与解析, 文章断句, 到语义层的机器翻译, 自然语言生成与识别, 语境分析, 主题识别, 文本分类等, 涉及范围十分广泛. 自然语言处理在日常生活中也是一个不可或缺的支持. 许多我们已经习惯的技术, 像是学习外语时用到的各种翻译软件, 到各个手机厂商提供的各种个人助手, 甚至于每天上网都很有可能打开的各个搜索引擎等等, 自然语言处理的技术已经渗透到我们日常生活的各个角落之中, 让我们已经渐渐离不开它了. 现在势头正热的人工智能技术, 其中备受瞩目的强人工智能的实现, 自然语言处理技术就是它少不了的一块基石. 自然语言处理与计算机视觉, 迁移学习等都是让计算机中跳动的高低电压逐渐贴近人类思维的一种方式.

2.1.2　统计语言模型与词嵌入介绍

自然语言处理技术的分类十分广泛, 也拥有非常多的理论基础. 研究者们关注的重点在于如何为我们日常使用的语言文字进行合适的建模. 目前比较主流的方法是采取统计方法对语言进行建模 [5]. 对于一个语言系统, 例如语音识别系统, 在给定了音频信号 a 和对应语言句子集合 S 的前提下, 这个系统需要解决的问题就是

$$\arg\max_{s\in S} P(s|a) \tag{2–1}$$

也就是确定信号 a 所对应的概率最大的句子 s, 其中 s 为一系列词语 w_i 的集合. 那么根据贝叶斯公式, 我们有

$$\arg\max_{s\in S} P(s|a) = \arg\max_{s\in S} \frac{P(a|s) * P(s)}{P(s)} \tag{2–2}$$

在这之中, 先验概率 $P(a)$ 与 s 的选择显然无关, 而 $P(a|s)$ 体现了被选择的词句与采集的信号之间的相关程度, 我们称之为采样模型.$P(s)$ 则是所谓的语言模型, 在这个语音识别的例子中, 它表示了该语言中各个语句的概率分布情况.

从以上例子之中我们可以发现, 统计语言模型关注的重点就在于如何得到语言之中的基本单位 —-例如音素, 单字, 单词, 词组甚至句子 —-的分布函数. 这个分布函数就能用来描述这个语言基于统计学的生成规则. 在实际的应用中, 一般选取的基本单位是单词, 而将句子看做单词的组合, 把句子的概率拆分成其中包含着的每个单词的条件概率的乘积, 也就是

$$P(s) = \prod_{i=1}^{n} P(w_i|c_i) \tag{2–3}$$

其中 w_i 和 c_i 分别代表了句子中第 i 个词语和其所对应的上下文. 有关于上下文的选择不同的语言模型是不同的, 但通常是由该词语在句子中前后的词语的序列所组成. 由此我们可以发现, 在实际的应用过程中, 统计语言模型要做的事情就是在给出句子上下文的前提下, 去猜测当前的词语是什么. 当然, 在某些模型中也可以反其道而行之, 通过某个词语去猜测其对应的上下文. 比较常见的统计语言模型有 **N-gram** 语言模型, 决策树语言模型以及最大熵语言模型等等.

2.1.3 N-gram 语言模型

N-gram 语言模型是一个早在 1980 年就被提出的, 历史悠久, 相关研究比较成熟的语言模型. 其本质是根据马尔科夫假设对语言进行建模. 其中的 N 代表的是被选取做为一个词组整体的词语的数量. 对于一个 **N-gram** 语言模型来说, 其对应的就是一条 $N-1$ 的马尔科夫链 [6]. 定义长度为 t 的序列 $\mathbf{w} = w_1, w_2, w_3, ..., w_t$, 我们可以根据贝叶斯公式来对其概率分布 $P(\mathbf{w})$ 进行分解

$$P(\mathbf{w}) = P(w_1)P(w_2|w_1)...P(w_n|w_1...w_{t-1}) = \prod_{i=1}^{t} P(w_i|w_1...w_{i-1}) \tag{2–4}$$

然而, 在序列长度 t 不断增加, 这个分解的过程需要计算的条件概率数量会无限制地上升, 最后导致实际上无法计算. 所以为了在实际中能够应用这一模型, 我们将序列长度限制在 n, 也就是说对于任意一个词, 只计算他最多 $n-1$ 个前驱单词, 由此我们可以将原式子变换成

$$P(\mathbf{w}) \approx \prod_{i=1}^{t} P(w_i|w_{i-n+1}, ..., w_{i-1}) \tag{2–5}$$

从时间序列的角度上来看, 这样的模型就相当于通过前 $n-1$ 个已知的前驱词语的历史信息来进行预测. 整体的概率分布可以通过其各个部分的概率来直接计算获得, 也就是由必须作为前提的条件概率分布组成了这个统计语言模型. 与其相似的隐式马尔科夫模型相比, 这个模型比较看重的并不是利用模型来进行对应位置值的预测, 而是代表了这整个模型的条件概率分布本身. 这也是很多统计语言模型所具有的特点.

subsec:word2vec

2.1.4 Word2Vec

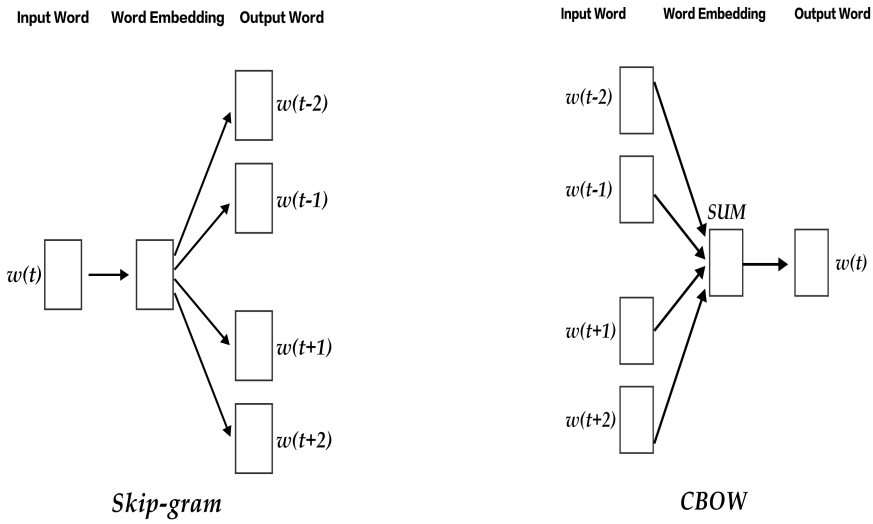
Word2Vec 是由 Tomas Mikolov 在 Google 的团队设计的用来产生词嵌入的一个浅层神经网络语言模型. 其基本特征是 **N-gram** 语言模型在神经网络上的实现. 整个模型的核心思路为利用 **N-gram** 语言模型与神经网络去实现自然语言中词语的向量化, 也即词嵌入 (word embedding).

首先要了解何为词嵌入. 在自然语言处理中, 词嵌入是指把一个维度数目等于单词表大小的高维离散空间嵌入映射到另外一个拥有较低维数的连续向量空间中. 在 Word2Vec 之前, 也出现过其他的词嵌入技术. 其中最具有代表性的, 就是直接采用原单词空间向量的单热点式词嵌入 (one-hot embedding). 这种方法得到的词向量, 是一个维数巨大的, 仅有一个维度为 1, 其余维度为 0 的向量. 显然这对于词语特征的进一步利用以及计算都是巨大的挑战.

其中一个比较显著的原因是这种词向量所有的信息都被集中到了某一个维度上, 信息量过于集中, 导致反而失去了在实际应用中信息本身的意义. 随后提出的分布式词嵌入 (distributed word embedding) 就是为了解决这一点. 例如在 2003 年,Y. Bengio 等人就首先实现了由神经网络训练得到的神经语言模型 [7]. 由机器去自主学习词语的分布式特征的方法就是我们如今所谓的词嵌入方法了. 比较常见的词嵌入方法有人工神经网络训练法, 词频矩阵降维法以及一些概率模型等等.

Word2Vec 由两种模型组成,CBOW 和 Skip-gram. 这两种模型是并列且互补的关系. 有着相似的作用原理, 也有不同的侧重处. 可以从图2–1中看出两种模型的具体结构. 两种模型虽然其中的神经网络的输入输出有些许区别, 但思路都是通过单词向量和上下文向量来互相训练, 最后得到训练后的向量来作为模型的输出词向量.

CBOW(Continuous Bags of Words) 模型, 又称词袋模型. 其思路为以目标单



(a) Skip-gram 模型　　(b) CBOW 模型

图 2–1 Word2Vec 模型

词 w 前后 n 个单词为上下文, 作为输入向量, 去训练网络输出一个预测的对应单词向量. 上下文向量通常由包含的全部单词对应的词向量求和而得. 因此 CBOW 模型对于上下文的顺序不敏感. 在训练过程中, 上下文所有的单词都一视同仁地平均分摊损失函数的影响.

Skip-gram 模型与词袋模型相反, 是通过目标位置的单词去预测其上下文的模型. 通过将目标位置单词对应词向量作为神经网络输入, 训练网络输出对应 $2*n$ 个上下文词向量, 其中每个位置与每个词向量会一一对应, 会对单词出现的顺序敏感.

让我们以 CBOW 模型的训练过程为例. 首先我们得到位置在 t 处, 长度为 $2*n$ 的上下文单词向量集合 $V_c = \{v_i | i \in [t-n-1, t-1] \bigcup [t+1, t+n+1]\}$, 将这 $2n$ 个向量作为输入层输入.

然后再以这 $2n$ 个向量取加和作为投影层, 我们能得到

$$\mathbf{v}_{project} = \sum_{v \in V_c} v \tag{2–6}$$

最后只需要将投影层加上激活函数输出, 就完成了整个神经网络. 至于训练过程,Word2Vec 提供了两种训练方法:Hierarchical Softmax 以及 Negative Sampling. 其中 Hierarchical Softmax 是通过将单词集合组织成哈弗曼树的形式, 通过将从树根到正例之中经过的路径上的词节点作为负例, 来对网络进行训练. 这种方法是为了改 e 良原本在整个集合上的 Softmax 训练法. 但在实际应用中, 这种方法也会消耗很多时间, 我们比较多的会采用 Negative Sampling 来进行训练.

顾名思义,Negative Sampling 就是通过固定次数的采样, 来取得负样本. 采样所依据的分布就是平滑后的词频. 这种方法在采样数趋向正无穷时, 其损失函数等价于原语言模型概率分布在神经网络下的损失函数 [8], 根据实验, 在通常的使用过程中, 采样率取 5 到 10 即可获得非常好的效果.

我们可以很轻松的推导出网络的损失函数

$$\mathbf{L}(w,u) = \delta^w(u) \cdot log[\sigma(v^T\theta^u)] + [1 - \delta^w(u)] \cdot log[1 - \sigma(v^T\theta^u)] \tag{2–7}$$

其中 u 为对应单词,w 为上下文单词, θ^u 为对应单词对应的网络参数, 而 σ 为激活函数, 这里采用的是 **sigmoid**. 然后利用随机梯度上升法, 能得到关于 θ^u 的梯度

$$\frac{\partial \mathbf{L}(w,u)}{\partial \theta^u} = [\delta^w(u) - \sigma(v_w^T\theta^u)]v_w \tag{2–8}$$

则关于 θ^u 的更新公式为

$$\theta^u := \theta^u + \eta[\delta^w(u) - \sigma(v_w^T\theta^u)]v_w \tag{2–9}$$

显然由于 θ^u 和 v_w 的对称性可以简单地求得 v_w 的更新公式. 可以说 Word2Vec 的网络模型是非常简洁且优美的. 并且两种神经网络语言模型的思路都是十分接近, 并且很对称的, 通过神经网络训练过程之中的副产物, 也即是代表了网络本身的一部分参数作为最终的成果词向量而输出.

这个想法既大胆, 然而又符合情理. 语言模型本身所关注的点就是作为语言基本单元的单词的上下文条件概率分布. 那么我们显然可以将这语言模型本身视为以单词和其上下文为参数的概率分布, 那么以词向量本身作为神经网络中的参数来进行训练就是一个非常符合逻辑的事情了. 可能正因为这思路大胆却符合逻辑,Word2Vec 才会既高效而又准确.

2.2 聚类分析介绍

聚类分析是针对于统计数据来进行分析, 从而获得数据结构, 组成等信息的一项技术. 在包括了机器学习, 数据挖掘, 模式识别以及计算机视觉, 生物特征提取等领域有着非常广泛的应用. 聚类算法的目的在于按照某个特定的标准将在某些方面相似度较高的个体数据对象通过静态地分类方法分成独立的组别或者子集, 使得同一组别或者子集下的所有个体成员对象都拥有较为近似的一些属性, 而使得不同的组别和子集之间的个体成员对象都拥有较为明显的区别.

主要的聚类算法可以划分为划分式聚类算法, 层次化聚类算法, 基于网格, 基于密度以及基于模型的聚类算法等等. 但在具体的应用过程中, 聚类算法的选择还是要取决于实际的数据情况和情况. 各个类型的聚类算法之中都有一些被比较广泛地应用的著名的算法, 例如划分式聚类算法中的 **K-Means** 算法, 层次化聚类算法中的凝聚层层次聚类算法, 还有基于模型方法之中的神经网络聚类

算法.

虽然聚类分析的方法方案种类非常多, 应用也十分广泛, 但本质上几乎所有的聚类方法都是三个要素, 或者说三个步骤的组合.

- 相似性衡量
- 聚类算法
- 数据简化

在进行聚类分析时, 通常的步骤是根据我们不同应用场景的需要和数据的类型结构, 分别在每个步骤选择合适的方法, 组合在一起就是一套完整的聚类方案了. 接下来分别就各个阶段中比较典型的方法来进行介绍

2.2.1 相似性衡量

相似性衡量就是指针对于给出的数据集, 特别是对于特定的一个数据二元组 $(\mathbf{x}, \mathbf{y})$ 我们选择一种直接或者间接的方法, 或者说一套评分标准, 来评价两者之间的相似程度, 以作为之后聚类算法实施比较的基准.

最简单, 也是最常见的方法就是求闵氏距离. 闵氏距离就是定义在赋范向量空间的度规. 根据范数的不同, 闵氏距离分别可以退化到曼哈顿距离, 欧式距离或者切比雪夫距离等等. 如果给定范数 p, 那么对于任意两个向量 $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, 我们可以按照下式计算其闵氏距离

$$D_M = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{2–10}$$

相似系数也是一种常见的相似性衡量指标. 所谓相似, 可以类比平面几何之中对于相似三角形的定义, 其最为核心的优势就在于不受线性变换的影响, 并且

可以很自然地转化为距离来进行计算. 主要的相似系数有余弦相似度, 和相关系数. 其中余弦相似度的计算就是对两个向量求其夹角余弦

$$C_{x,y} = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||\,||\mathbf{y}||} \tag{2–11}$$

而相关系数中最常见的要数皮尔逊相关系数. 其定义为两个向量之间协方差与标准差的商值.

$$\rho_{x,y} = \frac{cov(\mathbf{x}, \mathbf{y})}{\sigma_x \sigma_y} \tag{2–12}$$

需要注意的是, 在通常情况下, 相关系数的计算要比距离慢, 特别是在维度较大的时候. 这是因为在计算协方差矩阵的时候, 复杂性会与向量维数的平方成正比. 因此在选择的时候需要注意这一点.

除了上面提到的, 核函数也是一种被广泛应用的方法.$K(x, y)$ 就是定义在 $\mathbb{R}^d \times \mathbb{R}^d$ 上的二元函数, 其功能在于将数据从低维空间投影到高维空间, 这样可以解决很多线性不可分的问题. 我们假设在原向量空间内的两个向量 $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, 其内积表示为

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \cdot \mathbf{y} = \sum_{i=1}^{d} x_i y_i \tag{2–13}$$

那么若存在映射 $\phi : \mathbb{R}^d \to \mathbb{R}^D, D > d$, 则我们可以定义该映射对应的核函数为

$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle \tag{2–14}$$

可以很容易的发现, 我们只要找到了一个合适的核函数, 在只需要知道映射后向量内积的场合, 例如一些分类问题以及我们所关注的相似度分析时, 我们是

不需要去计算原来的映射甚至于都不需要知道原映射具体是什么样的. 这给我们的计算带来了很多便利. 这也是为什么核函数这么受欢迎, 有那么广泛的应用的原因. 当然在实际的使用中, 核函数必须要满足 Mercer 条件[1], 并不是可以随便定义使用的. 一个常见的核函数是径向基函数核, 这个核函数也是支持向量机中经常使用的核函数, 它对应的映射将原向量映射到无限维度的向量空间中. 这个函数的定义为

$$K(\mathbf{x}, \mathbf{y}) = exp\left(-\frac{||\mathbf{x} - \mathbf{y}||_2^2}{2\sigma^2}\right) \tag{2–15}$$

其中 $||\mathbf{v}||_2$ 的意义为向量的 L2 范数, 即 $||\mathbf{x} - \mathbf{y}||_2^2$ 等价于两向量之间的平方欧几里得距离.

2.2.2 聚类算法

在选择了定义数据相似度的方法后, 重要的就是如何根据数据之间的相似程度来对其进行聚类. 这时候就应该让各个聚类算法发挥作用了.

最常见的聚类算法应该要数划分式的聚类算法了. 这类聚类算法的中心思想在于, 首先确定好需要分的类别的数目, 然后以类别之内的数据点都足够接近, 而类别之间的数据点都足够远. 这类算法通常采用的方法是给定初始化的类别中心点, 然后根据不同的启发式算法来对数据集进行迭代, 一直到形成足够好的结果为止.

这类算法的代表是 **K-Means** 聚类算法. 正如之前所介绍的那样,**K-Means** 算法的执行过程是迭代的启发式算法, 每一次迭代分为两个阶段, 指派阶段和更新阶段. 算法会在收敛时停止.

算法 2.1 (K-means 算法)**.** *对于给定的向量集合 $V, |V| = m$ 以及分成的聚类数 k,*

[1] https://en.wikipedia.org/wiki/Mercer%27s_theorem

首先指派 k 个初始均值点 $m_1^{(0)}, m_2^{(0)}, ..., m_k^{(0)}$

第 i 次迭代:

指派阶段:

对于 $\forall v_j \in V$, 我们定义其指派 $A_j^{(i)}$ 为

$$A_j^{(i)} = q \iff \forall p, 1 \leq p \leq k, |v_j - m_q^{(i-1)}|^2 \leq |v_j - m_p^{(i-1)}|^2 \tag{2–16}$$

更新阶段:

根据指派阶段, 得到新的聚类 $C_1^{(i)}, C_2^{(i)}, ..., C_k^{(i)}$, 其中

$$C_j^{(i)} = \{v_q | A_q = j\} \tag{2–17}$$

若对于 $\forall j, 1 \leq j \leq k$, 有 $C_j^{(i)} = C_j^{(i-1)}$, 则算法收敛, 结束迭代, 否则计算新的聚类均值点

$$m_j^{(i)} = \frac{1}{|C_j^{(i)}|} \cdot \sum_{v_q \in C_j^{(i)}} v_q \tag{2–18}$$

可以看到,**K-Means** 算法的一个非常显著的特征是算法的步骤非常简单, 并且易于理解. 而且由于对于一个有限集合, 其指派的方案是一个有限集合, 算法必然能够收敛. 但是这种算法也有两个比较明显的缺点. 第一个缺点由于算法只能从初始点出发到达某个局部最优点, 而并不能保证一定到达全局最优, 所以算法的效果一般对初始化的均值点非常敏感.

第二个缺点就是通过欧式距离来决定指派以及通过最小二乘估计来更新新的聚类中心, 导致算法无法解决线性不可分的聚类问题, 例如在二维空间中的异或以及嵌套的圆环等数据集合结构. 为了解决这些问题, 还有很多的算法是根据这个算法衍生出来的, 例如 **K-Modes** 算法和 Kernel **K-Means** 算法等等, 在这里

就不再赘述.

第二类常见的聚类算法就是层次式聚类算法. 这种聚类算法的思路主要分为两个方向, 自顶向下和自底向上. 两种思路的执行过程也非常符合直觉, 自顶向下法就是首先将所有的数据对象分为同一类, 然后根据数据个体与类型整体的相关程度来将区别较大的个体列为别的分类. 而自底向上则是首先将所有的数据对象自己分为一类, 然后根据相关程度的远近来合并每个类.

很显然, 这种方法最后会得到的是一颗叶子节点代表每个数据对象, 其余节点代表不同等级的聚类的二叉树. 如果我们要得到对原数据集合的一个划分, 只需要确定一个合适的深度, 取出这个深度的所有节点, 就得到了原数据集合的一个划分. 根据深度的不同, 我们可以得到不同精度的划分, 或者说不同规模的聚类. 这也是层次式聚类的优点所在, 得到的结果信息量较大, 可以适用于很灵活的应用. 当然其代价就是在做聚类时, 计算量较大, 通常每轮迭代需要将目前剩余的所有数据聚类进行比较, 然而却只能新增或者减少一个聚类.

在实际的应用过程中, 为了提高使用的效率, 也是存在相应的手段的. 例如从数据中提取出能代表原数据集合中连续信息的一个特征集合, 再将其映射到某个上近似空间, 以此可以得到具有一定相似性的粗分类的上近似 [9]. 这种基于模糊集的近似方法中, 由于一个数据对象可以属于多个粗分类, 因此可以在一次迭代中合并多个聚类, 可以有效的提高层次聚类的运行速度.

除此之外, 还有基于网格的聚类算法, 基于密度的聚类算法以及基于模型的聚类算法等. 在这里就不全部介绍了.

2.2.3 数据简化

在确定了相似性度规以及聚类方法后, 为了实际应用中的效率和方便, 我们通常会对原数据集合进行一定的处理, 来简化之后的运算过程. 比较常见的简化方法有降维和抽样等, 针对一些特殊的数据类型, 还可以对其做合适的变换.

首先是降维, 降维是指在一定的前提下, 将数据中的维度数, 或者说变量数降低, 得到一组不相关, 或者说相互正交的主要变量的过程. 具体可以通过舍弃去除原有数据变量, 或者去创造新的数据变量等方法来实现.

常见的降维方法比如主成分分析, 就是一种通过多元统计分析方法来简化数据集, 使得数据集维数降低的技术. 它的特点是在使数据集合简化, 维数降低的同时, 还可以尽可能的保持数据集中对整体方差贡献最大的维度, 或者说贡献最大的特征. 这是通过保留低阶的数据主要成分, 而舍弃掉高阶成分而实现的.

主成分分析的算法具体步骤是非常简单的. 在已知数据集矩阵 M_{data} 的前提下, 首先将数据中心化, 也即是使得数据集每个维度都均匀的分布在原点附近. 具体来说就是求出每个变量的均值, 然后对数据集每一列都减去对应的均值操作. 这样得到的数据集合就是所有维度均值都为 0 的新数据集 M_0.

在新数据集上, 我们来求特征的协方差矩阵. 所谓协方差矩阵, 就是衡量特征之间的相关程度的矩阵. 具体以两维的特征数据集来举例的话, 如果数据集合由变量 x, y 组成, 那么其协方差矩阵为

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) \\ cov(y,x) & cov(y,y) \end{pmatrix} \tag{2–19}$$

在得到数据集协方差矩阵后, 我们下一步可以求协方差矩阵 C 特征向量 $\{v_i\}$ 和其对应的特征值 $\{\lambda_i\}$. 特征值的大小就代表了对应维度对数据集整体方差的贡献大小, 我们从中选取特征值最大的 k 组特征向量 V_k, 再将原数据集合投影到这 k 个特征向量上, 就得到了新的降维过后的数据集了.

$$M_{pca} = M_0 \cdot V_k \tag{2–20}$$

由于主成分分析的算法非常简单, 但是处理后的数据效果却往往都比较不错, 也让人不禁好奇其中蕴含了怎样的数学原理. 这也要从它的中心思想来说起, 也就是主成分分析本质上是保留了数据变量中方差较大的部分, 而舍弃掉了方差较小的变量. 这么做的原因在于在信号处理理论中, 我们认为在信号与噪声之间比较, 信号的方差较大而噪声的方差较小. 所以我们会选择以对方差的贡献大小为标准来保留维度.

那么为什么可以通过维度对应的特征值来进行这种方差贡献的比较呢? 简单地来说, 就是在寻找一个新的向量作为数据集投影并使得在该向量上数据集投影方差最大时, 将这个问题转化为寻找数据集矩阵对应线性变换所能造成最大模长变化的向量. 那么显然, 所有模长变化都必须小于该矩阵所对应的最大特征值, 这样一来两者就很轻松的联系到了一起, 也让人不得不感慨数学的美妙以及主成分分析作者的天才了.

除了主成分分析, 奇异值分解这种将数据作为矩阵操作的线性算法, 还有一大类降维算法就是流型学习了. 所谓流型, 就是指在局部可以近似退化为欧式空间的特别的拓扑空间. 简单的来说, 任何一个球面都可以认为是一个简单的流型, 因为当对应的区域足够小时, 球面会退化成一个平面, 或者说我们可以通过一个简单的映射, 来通过一个平面上的点来表示对应球面上的点. 具体来说对应整个球面, 我们可以将其分为上下两个半球, 然后将其分别映射到一个正方形区域上, 这样的两个映射我们分别称为流型的一个坐标图, 而其组合起来则称为能够描述这个流型的图册.

然后我们发现, 在这个简单的流型里, 我们通过将具有曲率的球面近似成平面, 将三维坐标成功的压缩成了两维, 这不正是一次降维操作吗? 没错, 所谓流型学习, 正是通过一些数学方法来对原本流型内蕴空间进行的近似, 在近似下来求所有点之间的相似性矩阵, 最后通过等距重构来将原数据集合映射到一个低维空间的一系列算法的统称.

其中一个比较常见使用的算法叫做局部线性嵌入算法 (Locally Linear Embedding).**LLE** 所关注的重点在于哪些数据点在原来的流型空间中距离比较近, 我们就会在新空间中重构时让他们继续保持足够近的距离. 而对于距离比较远的数据点来说, 我们并不关注他们在投影到新空间之后的位置关系.**LLE** 通过足够接近的样本之间的关系进行互相之间的重构.

$$v_i = \sum_{Sim(v_i,v_j)<T} w_{ij} v_j \tag{2–21}$$

其中 $Sim(x, y)$ 为计算两者之间相似度的函数, 具体可以参照2.2.1中介绍的方法. 而 T 在这里表示人为控制的阈值, 可以是固定的值, 也可以是按照距离顺序排序得到的一个变化值.**LLE** 的计算可以通过拉格朗日法来进行, 具体的目标优化函数为

$$\arg\min_{w} \sum_{i} ||v_i - \sum_{Sim(v_i,v_j)<T} w_{ij} v_j||_2^2 \tag{2–22}$$

除了降维, 我们还可以对较大的数据集进行抽样, 只要采取合适的抽样方法, 可以在明显有效地降低数据集大小, 减少计算复杂度的同时, 还能较好地保持原有数据的特征, 使得最后的结果可以与原本数据集得出的结果近似相同, 在这里就不赘述具体的应用.

2.3 推荐算法介绍

所谓的推荐算法, 是任何一个推荐系统的核心部分, 也是为什么称其为推荐系统的理由. 一个推荐算法会对现有的数据信息进行过滤, 从而用于对用户对目标对象的偏好或者说是评分. 其核心目的在于为不同的用户, 提供个性化的服务. 在这个信息量与日俱增的时代里, 无论是怎样的信息媒介甚至与物品, 都有着繁

星一般众多的分类. 视频, 音乐, 图片, 文章, 这种种的媒体, 在我们浏览因特网的时候, 以各种各样的形式出现在我们眼前, 而我们也依据自己的喜好, 在它们之间做出选择. 打开任何一个文章网站, 或者是视频网站, 通常都会在阅读或者浏览的位置之外, 来给出你可能会感兴趣的文章或是视频的推荐. 去电商网站上购物时, 通常在首页上也会出现网站认为你会比较感兴趣的商品, 来辅助你的选择. 就连很多的普通网站上的广告, 也是综合了搜索引擎中的使用记录, 来得出你可能会接受会感兴趣的广告种类.

那么, 既然在我们的互联网生活中, 推荐系统, 推荐算法的身影这么频繁的出现, 它们背后的原理又是怎么样的呢? 这样一个神奇的功能究竟是怎样实现的呢? 如何让一台普通的机器居然就能够读懂人心了呢?

常见的推荐算法大致可以分为五类

- 基于协同过滤的推荐算法
- 基于内容的推荐算法
- 混合推荐算法
- 流行度推荐算法
- 基于深度学习的推荐算法

2.3.1 基于协同过滤的推荐算法

在一系列的推荐算法中, 其中最常见的算法就是协同过滤算法了. 所谓的协同过滤, 就是一种从众心理的具体体现, 也是对于集体智慧的一次实际应用. 早在维多利亚时代的英国, 就有实验利用对牛的体重估计证明了集体思维在平均意义上具有很高的准确度.

在协同过滤中, 主要的影响因素有两个, 用户和项. 用户指的自然就是我们这些推荐系统的使用者. 在推荐系统中, 我们认为用户对所有的项拥有自己的偏好, 并且会根据自己的偏好进行评分和选择. 而所谓的项, 根据推荐系统应用的不同也会不同. 在视频网站推荐系统中, 所谓的项就是一个一个视频, 在文章网站推荐系统里, 项自然指的就是一篇一篇的文章了. 协同过滤通常分为基于领域的协同过滤和基于模型的协同过滤. 其中基于领域的协同过滤主要考虑的是两个关系, 用户与用户之间的关系以及项与项之间的关系.

这实际上也是非常符合我们人类的思维直觉的. 举个浅显的, 现实的例子, 例如我们要出去吃饭, 要在众多的餐馆中进行选择. 这时候我们就是用户, 而项就是待选的所有餐馆. 那么我们会用怎样的思路去考虑我们应该去哪一家吃饭呢? 掷筛子当然是一个简单的选择, 但同样的也是一个选到我们满意的餐馆期望最低的选择. 一个最直接的想法是, 我们可能会询问周围的人, 哪一家餐馆可能是我们会喜欢的. 那么这个时候, 你是会去问经常一起出去吃饭, 一起玩的同龄人好朋友呢, 还是会去问一个素不相识的, 甚至于来自一个文化和风俗习惯都和你所在的环境截然不同的陌生人呢? 显然前者的推荐是有比较高的可信度的. 当然这里举出来的例子可能比较极端, 但是在做选择时的整体思路, 就是去考虑怎样的人, 也就是用户可能和我拥有接近的喜好. 那么根据他们的喜好做出的推荐, 就更有可能同样地符合我自身的喜好. 基于用户的领域协同过滤就是通过对项的选择喜好, 来定义一个用户在用户空间的位置和用户之间的相似程度. 然后来根据相似用户的口味, 来为用户推荐可能符合其口味的项.

同样的, 我们也会考虑项与项之间的关系. 这时候举例子就需要做一个比较强的假设了, 如果我们可以知道每家餐厅每天接待了谁, 每家餐厅都有哪些常客. 这时候我们就可以根据这些常客们之间的交集来对不同的餐厅之间的关系有比较好的认识. 这层逻辑也是非常浅显易懂的, 设想如果喜欢一家老牌川菜馆的人们, 居然大多都喜欢去另一家新疆菜馆, 那么我们自然会猜测这家新疆菜馆里是

不是有什么招牌菜让人们能辣的乐不可支不能自拔. 那么这时, 如果我也是那家川菜馆的老主顾, 我自然也比较可能同样地会喜欢那家神奇的新疆餐厅. 基于项的领域协同过滤考虑的就是根据项同时出现在用户选择中的情况, 来计算项之间的相似程度, 从而根据用户对项的选择, 来猜测其他可能接近用户喜好的项.

不管是基于项或是用户, 考虑的角度是从做出选择的那方还是被选择的那方, 一个核心的组件是不会变的, 那就是由用户和项共同组成的用户评分矩阵, 我们称为评价矩阵 M_a. 两种方法的区别在于我们怎样利用这个矩阵作为原始数据来处理得到我们实际上来应用推荐的新矩阵.

在基于用户的邻域协同过滤中, 我们可以通过这个矩阵来得到用户之间的相似度矩阵. 一个简单的思路是将 M_a 中的每一行代表用户的数据视为一个向量, 然后来根据一系列的比较算法来计算两个向量之间的相似度. 最为简单的方法就是求两者的余弦相似性. 那么从矩阵整体的角度来看我们可以发现这就相当于去进行一次矩阵乘法

$$M_u = M_a M_a^T \tag{2–23}$$

得到的新矩阵中, 每一个元素就代表了两个用户之间的相似程度. 那么再推荐的时候, 我们就可以首先从这个矩阵中找到和目标推荐用户最为接近的 k 个用户, 将他们所选择的项综合起来, 再去掉当前用户曾经选择过的项, 这样就获得了相应的推荐.

$$R(u) = \left(\bigcup_{v \in N_u} I_v \right) \backslash I_u \tag{2–24}$$

其中 N_u 表示与用户 u 接近的其他用户集合, 而 I_v 表示用户 v 的喜欢的项的集合.

基于项的邻域协同过滤推荐也是类似的, 只是此时我们关注的是项与项之

间的相关度, 那么我们相似度矩阵的求得就应该由这个公式来进行.

$$M_i = M_a^T M_a \tag{2–25}$$

得到的相似矩阵就描述了任意两个项之间的相似程度. 在实际推荐时, 我们通过寻找与用户喜欢的项接近的项来做出推荐.

$$R(u) = \bigcup_{v \in I_u} N_v \tag{2–26}$$

有趣的是, 虽然基于用户和基于项的邻域协同过滤算法无论是在思想上还是具体的算法步骤上都非常相似, 甚至于关键的相似性矩阵的计算仅仅是同一个矩阵不同方向的内积, 但在实际应用中, 这两个算法却往往会带来不同的推荐结果. 所以在实际应用中, 往往会同时采取这两种算法, 能够为用户带来不同的体验.

基于邻域的协同过滤算法由于其实现的简易以及效率被广泛的采用, 然而它也有相应的弱点. 特别是在评价矩阵过于稀疏时, 往往难以得到合理的结果. 意思是当可选择的项很多, 但用户和用户的喜好比较有限的时候, 两个用户之间往往很难有多大的重合性, 这样子在寻找邻域的时候就会出现很多的误差. 还有一点就是这种基于用户行为的算法, 太过于依靠评价矩阵 M_a 作为整个算法的基石, 在一些新的网站上就难以启用. 这种困难的局面被称为" 冷启动" 问题, 也是基于协同过滤的推荐算法常常要面对的.

2.3.2 基于内容的推荐算法

在基于协同过滤的推荐算法之外, 第二常用的推荐算法就是基于内容的推荐算法了. 这两者都非常符合我们人类的思考直觉. 在做一个自己没有把握的决定时, 要么参照与自己会做出相似选择的人, 要么参照之前做过的相似的选择.

这基于内容的推荐算法的中心思想就是后者.

和基于项的协同过滤的思想类似, 基于内容的推荐算法总是推荐与用户之前所选择的项类似的项. 但这两者的区别在于, 基于内容的推荐算法并不依赖于评价矩阵 M_a, 它对两个项之间相似度做出评价的标准是项本身的内容特征而非用户对项的偏好与评分.

在简单的例子中, 一个项的标签可以通过很多彼此之间独立的而各自又十分显而易见的特征来表征. 例如一个电影, 我们可以通过电影的标题, 电影的种类, 电影的年代, 电影里的主演和导演, 甚至是电影的时长和票房等来构建其特征向量.

当然实际上的处理肯定没有这么简单. 以上面的例子来说, 建立好了特征向量好, 如何去评价两个向量之间的相似度也是一个很麻烦的问题. 在这里, 每个变量各自的分布是不同的, 甚至有些本身就难以比较, 这让两个向量整体之间的比较显得更加困难. 所以基于内容的推荐算法的要点往往在于找到合适的提取项的特征向量, 并加以比较度量的方法.

那么以文章推荐为例, 我们的每一项就是一篇文章, 怎么去将文章进行特征提取从而使其向量化呢? 最常见的算法要数 **tf-idf** 了.tf-idf 是一种将统计学方法应用到文章集合或者语料库中, 从而来评价一个词语在一篇文章中的重要程度. 然后我们就可以根据由这些词所作为变量所组成的向量来比较文章之间的相似性了.

tf-idf 算法可以很简单的分为两个部分. 其中的 tf 表示词频, 是表示一个给定的单词在目标文章中出现的频率. 之所以用频率而非频次是为了减少文件长度对这个算法的影响. 我们可以通过以下公式来计算词语 w_j 在文章 A_i 的重要程度.

$$\mathbf{tf}_{i,j} = \frac{n_{i,j}}{|A_i|} \tag{2–27}$$

其中 n_j 表示 w_j 在 A_i 中出现的次数, 而 $|A_i|$ 则是文章所包含单词的总数.

而 idf 则表示逆向文件频率, 是用来衡量一个单词在整个语料库或者文章集合中的重要性的度规. 计算逆向文件频率的公式可以写作

$$\mathbf{idf}_j = \log \frac{|A|}{1 + |\{d_i | w_j \in A_i\}|} \tag{2–28}$$

即用文章集合中所有文本的个数除以有 w_i 出现的文章的个数, 然后在取对数. 加上 1 是为了防止某个词在所有文章中均没有出现.

在得到了 $\mathbf{tf}_{i,j}$ 和 $\mathbf{idf}_j$ 之后, 我们可以计算 w_j 对于 A_i 的重要程度, 也即是其 tf-idf 指标

$$\mathbf{tfidf}_{i,j} = \mathbf{tf}_{i,j} \times \mathbf{idf}_j \tag{2–29}$$

经过简单的思考我们可以发现,tf-idf 指标倾向于将仅仅出现在目标文章以及其他少数文章中的单词当做重要的单词, 而会把在所有文章中都很常见的单词忽略. 这也是很符合我们的直觉的一种方法. 在得到了所有单词对于一篇文章的 tf-idf 指标后, 我们就得到了这篇文章的 tf-idf 向量. 我们可以通过简单的余弦相似性来对两个文章的 tf-idf 向量进行比较从而来判断两篇文章的相似程度.

除了 **tf-idf** 算法之外, 之前在2.1.4中也有介绍了一种将词进行向量化的技术叫做 Word2Vec. 而以词向量为基础的 **Doc2Vec** 也是有很广泛的应用场景的.

Doc2Vec 通常假定已经获得词向量集合 W_i. 那么对于一篇文章, 如何运用它所包含的单词的词向量就是将文章转化为向量的关键了. 最简单的思路是对所有包含单词的词向量取加权平均, 权值可以是最简单的词频, 也可以是其 tf-idf

的值.

$$D_j = \sum_{w_i \in D_j} (W_i \times \mathbf{tfidf}_{i,j}) \tag{2–30}$$

这种方法的好处在于简单易于实现, 并且利用上了所有的信息. 特别是在使用 tf-idf 的时候, 往往能够获得比仅仅使用 tf-idf 来作为特征向量好很多的结果.

这种方法可能存在的问题就是对词向量的训练结果比较敏感, 特别是对于一篇文章中的高频词, 重要的词, 其 tf-idf 值很高, 但如果所对应的向量训练结果不好的话, 对于整个文章向量的结果影响就会很大.

那么为了不让少数的词向量造成过多的影响, 我们有另外一种方法. 就是将词向量聚类之后, 以词聚类为单位变量来表示文章向量.

首先, 为什么能够将 Word2Vec 生成的词向量聚类呢, 这是因为 Word2Vec 的模型生成的词向量有很好的性质. 可以很方便的用欧式距离来衡量两个词的关系, 甚至还可以进行一定程度的四则运算. 那么得到了词聚类之后, 我们再生成文章向量时, 使用的就不是简单的词向量配上词频或者 tf-idf 值. 我们直接对词聚类来求频率或者 tf-idf 值.

这种方法的好处很多. 首先, 和直接的 tf-idf 对比,tf-idf 仅仅反应了词与文章的关系, 词与词之间的关系并不能很好的体现出来. 同时出现在一篇文章中的两个重要词, 究竟是在意义上非常接近的两个词, 实际上是比较重复的意义表达呢, 还是两个互相补充意义的单词, 综合起来表达出一个与单独出现时不同的意义呢? 通过 tf-idf 的方法完全无法体现出来这种区别. 而将词先行聚类之后在进行 tf-idf 算法的计算, 首先就通过词向量的性质保证了不会将同一意义的单词重复在两个分量上进行计算, 无形之中增加了生成的文章向量所蕴含的信息量.

其次, 与简单的词向量加和或者 tf-idf 加权相比, 并不直接运用所得到的词向量使得文章向量具有更好的鲁棒性. 并不会受词向量中的特异点很大的影响.

而且比起在加和之中隐藏了一部分的信息, 将不同类型的词所代表的信息分散地分布在各个维度上也更好的保留了文章所包含的单词所蕴含的信息. 这也是加和作为一个非单射所不可避免的弱点, 我们通过维度的分散很好的解决了这个问题.

2.3.3 其他推荐算法

除了两类应用最为广泛的推荐算法之外, 目前比较常见的还有思路和实现起来最为简单的流行度推荐算法, 以及将两个或以上的推荐算法结合起来的混合推荐算法. 近几年, 随着神经网络, 深度学习等概念的普及和火热, 也出现了基于深度学习的推荐算法.

流行度推荐算法就是最简单的从众心理, 也就是我们打开搜索引擎时推荐的热搜词, 打开电商网站时给出的最热商品. 流行度推荐算法作为一个最基础的推荐算法, 无论是其思想还是具体的实现都是非常简单的. 正因为如此, 其优缺点也是非常明显地让人一眼能看透. 首先是优点, 这种算法由于不需要关注于被推荐用户个人的偏好, 所以在新用户加入时, 可以解决用户数据缺乏的冷启动问题. 还有由于实现的简单以及易于解释, 流行度推荐算法往往可以和其他推荐算法放在一个系统中, 在进行适当的互补的同时, 也提供了一个很好的基线. 在其他推荐算法获得足够的数据来给出足够好的服务之前, 往往可以使用流行度推荐算法来增强原有的推荐系统, 以此来获得足够的前期用户活跃度和使用率.

而混合推荐算法, 也是针对于原有的基于协同过滤的推荐算法或是基于内容的推荐算法中存在的不足而创造出来的. 针对于原有算法的不足, 通过引入另外一个算法的方式, 来给予弥补. 以基于协同过滤的方法为例, 由于算法整体过于依赖评价矩阵, 所有的计算都是基于 M_a 来进行的, 所以算法对于矩阵中的空白行就没有任何的应对手段. 从实际的角度来说, 如果在系统中引入一个新的项, 那么在没有用户选择过这个项, 给项评分之前, 这个项就无法得到推荐, 反过来

又会让这个项得不到评分. 这就是协同过滤中的新项问题.

我们为了解决这个新项问题, 可以通过基于内容的推荐算法的方式. 当新的项进入原有系统时, 我们可以通过基于内容的推荐方式来将其推荐给用户. 甚至于我们可以以新项为中心, 寻找可能对其感兴趣的用户, 推荐给他们以此来获得反馈. 这样, 我们就通过两个不同的推荐算法的协作, 解决了原本单个算法很难解决的问题.

除了这种根据不同情况选择不同推荐算法的方式, 混合推荐算法还有很多其他的方式来组合两种或以上的推荐算法. 例如直接将不同算法得到的推荐结果同时呈现给用户的混杂型, 通过设置固定的或者是通过学习得到的权值来综合不同的推荐算法给项目的评分的加权型, 将一个推荐算法的结果作为推荐目标再次放到另一个推荐算法中以此来综合两个推荐算法的级联型等等. 虽然每种综合手段各不相同, 原理可能也都有不小差别, 但总体的思路都是大体一致的, 都是通过某种手段来使得最后的结果能够更合理的将来自不同算法的不同推荐结果综合到一起. 包括最近出现的深度学习推荐算法, 也是将深度学习作为一个特征提取的手段, 来进行推荐的, 可谓是大同小异.

2.4 本章总结

本章主要为背景知识介绍章节, 主要目的是为之后的系统架构以至于整体的介绍做必要的铺垫. 由于本系统所涉及到的领域比较广, 其中也有很多的细节知识, 所以必须在完整地介绍整个系统之前做适当的介绍. 在本章节中, 从系统设计的几个基础出发, 介绍了系统前端所应用的一些自然语言处理技术, 顺便介绍了当下最经常应用的统计语言模型以及其应用. 还有十分火热的 Word2Vec 的一些相关模型和原理.

对于用到的聚类分析技术, 本章节中从其各个阶段, 相似性计算, 数据聚类以及数据简化进行了介绍, 着重介绍之后将要用到的一些算法步骤时, 也对聚类

算法的整体情况进行了一定的解说, 让读者能对之后的应用, 以及为什么这样选择能有更好的理解.

对于推荐算法这一目前在互联网应用中非常热的热点, 本章节中也是花了不少篇幅来进行介绍. 对于各种各样的推荐算法思路, 从原理上进行了一定的解析, 也对各种不同的思路方向进行了对比, 让读者能够理解这不同算法中存在的差异和优劣, 以及各种各样合适的应用场景和限制等.

在本章节之后, 相信读者可以对系统整体的架构有个大体的猜测, 从而在阅读之后的章节时, 能够更快的理解整体的结构, 并且更好的判断出每个关键节点的地方在做出选择时所面对的抉择以及这样选择的原因. 相信也能够对本文以至于整个系统有着更加全面深刻的理解吧.

第三章　系统架构

3.1　系统环境

本文章所对应实现的主要目的有两个, 第一点是从学术大数据中提取学术会议之间的相关关系, 并将其以可视化的形式表现出来. 第二点则是利用学术大数据以及其中学术会议之间的相关关系, 为在阅读查找学术论文时, 想要寻找主题相近的文章的人提供帮助的一个推荐算法.

数据来源为微软公司截止至 2016 年 2 月所收集的论文的各种信息, 包括题目, 作者, 会议, 摘要等等. 选作为数据源的论文为筛选出的包含论文数量最多的前一千个会议. 论文总数为 1514255 篇, 用于训练词向量的文本总大小为 1.2GB.

整个项目基本上由 Python 语言编写而成, 训练词向量的环境为 Ubuntu 14.04 LTS, Python 2.7, Tensorflow 1.0 版本, 通过 GeForce GTX 1080 显卡 GPU 进行训练.

从词聚类到会议向量生成一直到推荐系统应用以及可视化数据的生成也是通过 Python 进行. 主要原因是系统主要处理对象为大量格式化数据, 而又夹杂了包括神经网络在内的一系列工程方面的实现,Python 作为一个新兴的面向对象的脚本语言, 生态环境良好. 既拥有以 Numpy,Pandas 为首的一系列支持数据处理的优秀的包, 也同时是几个主要的神经网络框架 Theano[1],Tensorflow[2]等选择的接口平台, 还可以很方便的搭建工程, 管理系统, 可以说是用来搭建本系统的最好选择. 并且笔者对 Python 的特性较为了解, 能够很迅速地利用 Python 中带有的各种语法糖来编写简短有效的代码段, 易于开发调试.

[1] http://deeplearning.net/software/theano/
[2] https://www.tensorflow.org/

3.1.1 Tensorflow

Tensorflow 是由 Google 公司旗下 Google Brain 的研究小组开发的一种利用数据流图来进行数值计算的一个开源的软件包. 从它的名字中可以看出, 在这里的数据流是以张量的形式来参与计算的. 所谓的张量, 简单来说也就是有固定”形状” 的多维数组. 严格的来说,Tensorflow 并不是一个仅仅能用于训练神经网络的框架, 但 Tensorflow 拥有设计优秀使用简便的 C++/Python 接口, 配合上它固有的数据流图的使用特性, 会让搭建个性化的神经网络变得十分简单, 而对多核 CPU 以及显卡等辅助计算工具的良好支持又使得它在提升效率上能给予我们很多帮助, 所以 Tensorflow 目前也成为搭建训练神经网络的首选框架之一.

在 Tensorflow 之中, 搭建好的神经网络被称为一个” 图”. 而图是由节点和边构成的.Tensorflow 中的图里的节点就是一个一个的张量, 也就是多维数组. 这些多维数组通常是在搭建图的时候就决定好了其尺寸和具体数值, 可以是固定的值常量 constant, 也可以是在训练中能够改变的值变量 Variable, 但也有特殊的情况. 在我们的输入层和最后输出层计算损失函数时就需要用到训练数据, 这部分在模型搭建时是不知道的, 但我们可以提前知道它在网络中的位置和后续的操作, 我们利用占位符 placeholder 来代表这一类的节点.

而节点与节点之间的连接关系称为边, 在 Tensorflow 中, 每一条边都代表了一个张量操作. 可以是简单的与标量的加减乘除四则运算, 也可以是复杂的矩阵运算, 还可以在张量上应用激活函数, 算结果与标签的交叉熵, 取各种统计量等做法. 采用 Tensorflow 的优势就在于我们不用自己去实现复杂的矩阵操作, 而是可以将整个张量视作整体, 来在比较宏观的角度来思考我们所搭建的网络结构, 这样能够很大程度上提升网络搭建的效率, 而不是自己仔细地去写一个个复杂的循环. 在图3.1(a)中有搭建简单网络的示意图, 可以在从图中大致了解网络的组成和训练的流程.

以下是一个简单的通过 Tensorflow 搭建神经网络训练识别手写字符

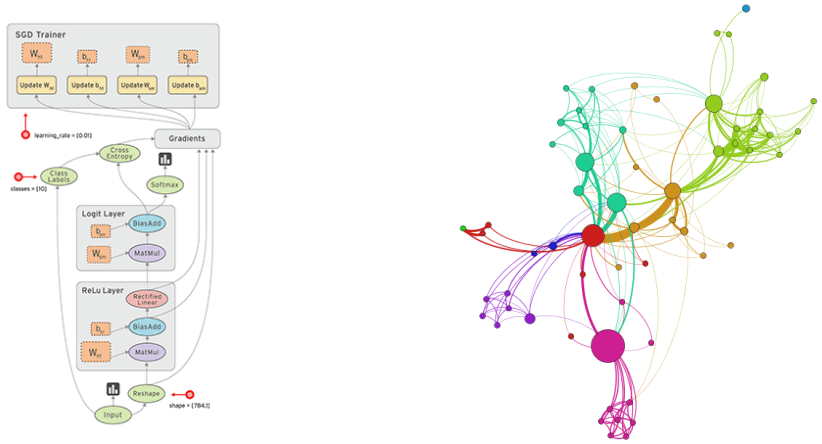
(MINST) 的示例代码

```
import tensorflow as tf
# 训练变量定义
x = tf.placeholder("float", [None, 784])
y_ = tf.placeholder("float", [None,10])
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))

# 网络搭建
y = tf.nn.softmax(tf.matmul(x,W) + b)
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(
                                                cross_entropy)
init = tf.initialize_all_variables()

# 开始训练
with tf.Session() as sess:
    sess.run(init)
    for i in range(1000):
      batch_xs, batch_ys = mnist.train.next_batch(100)
      sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

可以看到, 我们搭建的网络是由一层简单的全连接层组成, 输入为 784(28*28) 而输出为 10, 激活函数采用 Softmax, 损失函数采用的是经常被用于多标签分类问题的交叉熵, 采用梯度下降算法进行训练. 一个网络的搭建仅仅需要几行代码, 并且通过读取代码中的函数名称等, 基本上就能掌握网络的整体结构, 训练也可以很方便的通过调用一个 run 函数来实现, 实在是搭建以及训练神经网络的利器.



(a) Tensorflow 节点示意　　(b) Gephi 图示意

图 3–1 工具框架示例

3.1.2 Gephi

Gephi 是一款开源免费的跨平台网络分析软件. 主要应用于对网络和各种复杂系统之中的各类大数据集的动态, 分层交互可视化以及探测. 常见用途可以是用来对科研项目中实验数据进行分析, 或是对统计信息进行分析研究, 例如新闻分析, 微博, 推特分析等等.

Gephi 是用 Java 编程语言, 基于 NetBeans 平台开发的, 通过 JVM 可以实现跨平台. 其可视化引擎是完全基于 OpenGL 的 API, 并且由于其完全开源, 使用者完全可以通过应用 OpenGL 的标准 API 来自己定制所需要的插件, 编写符合自己要求的新功能或者对旧的显示选项进行修改等等, 总而言之是一款非常好用且灵活的大数据可视化软件.

在 Gephi 中, 我们通过定义节点和边以及它们之间的关系来定义一幅图. 节点之间可以通过有向边或者是无向边来相互连接, 而节点和边本身又可以携带很多信息, 例如节点编号, 节点标签, 边的编号, 标签, 种类, 权重等等. 我们可以很方便的自己添加各种信息, 来在图中体现各种各样的数据集合. 同时, 我们定义好的图可以也就可以通过二元组 (V, E) 来表示, 其中节点和边的信息可以作

为 csv 文件导入导出, 使用非常简便, 也方便 Gephi 程序本身与后端处理数据的 Python 代码对接. 从图3.1(b)中可以看到用 Gephi 作为可视化工具画出来的图. 非常美观大方, 又可以清晰的看到数据点之间的关系.

3.2 系统模块

整个系统分为两大部分, 会议关系可视化以及基于会议关系的论文推荐系统. 其中两个系统的前端是相通的, 进行处理的基本思路是相同的, 或者可以说会议关系可视化是对基于会议关系的论文推荐系统的一个补充, 二者在功能上相辅相成, 基本原理也是一样的.

前端则是从最开始的词向量生成一直到会议向量生成. 可以将整个前端理解成一个独立的系统, 这个系统以学术论文数据库为输入, 最终能够生成有良好性质的会议向量以及各个文章向量来辅助我们进行数据分析.

3.2.1 词向量生成

根据在前文2.1.4中所提到的方法, 我们采用 Word2Vec 的方法来实现从论文数据到词向量的转化. 首先我们来观察我们所拥有的数据情况. 在微软公司提供的数据库中, 我们可以根据一篇论文的编号, 得到其标题, 作者, 发表的会议以及其摘要, 下载地址等信息. 那么我们应该采用哪些数据, 又应该放弃哪些数据来进行词向量生成呢?

这个时候我们就应该把目光从当前的词向量放得更远一些, 想清楚我们为什么需要这些词向量, 我们最后会怎么应用这些词向量, 怎么样才能使得生成的这些词向量更符合我们之后所需要的应用呢? 这样一想我们就豁然开朗了. 我们的根本目的是生成能够很好比较相似性, 甚至于一些更复杂的相关关系的会议向量. 而这么做的好处是我们可以为寻找关注于某一类论文的用户提供更好的推荐服务. 那么, 显然我们应该让词向量能够更好的反应这篇文章所关注的重点,

所归属的学术分类. 更具体地说, 是需要找到一个词, 在整个语料库, 整个学术论文数据库中, 所代表含义的综合体. 通过这个综合的意义, 来组成代表这个词的词向量.

所以在经过分析之后, 我们考虑的是将每篇论文的摘要和标题作为训练数据来训练词向量. 首先论文正文数据量过于庞大, 并且比较杂乱, 里面会夹杂一些没有用, 或者无法体现出文章特点的词语, 数据等等, 稀释了训练的成果. 而其余的数据并没有办法直接地拿来放到神经网络中去训练一个词向量. 所以我们就得到了以所有文章的摘要组成的训练语料.

词向量的训练通过在 Tensorflow 上实现 Word2Vec 算法来执行. 值得一提的是, 由于在搭载 NVidia GeForce GTX 1080 显卡的机器上进行训练, 所以在训练时采用了 GPU 模式, 将训练速度提高到了原来的 20 到 30 倍, 这也要归功于 Tensorflow 前后端分离的架构, 能让在显卡 GPU 上的部署变得非常方便.

网络搭建部分的代码如下示例

```
# From word2vec.py
def build_graph(self):
    opts = self._options

    (words, counts, words_per_epoch, self._epoch, self._words, examples
                                            , labels) = word2vec.
                                            skipgram(
            filename=opts.train_data,
            batch_size=opts.batch_size,
            window_size=opts.window_size,
            min_count=opts.min_count,
            subsample=opts.subsample
            )
    (opts.vocab_words, opts.vocab_counts,
     opts.words_per_epoch) = self._session.run([words, counts,
```

```
                                            words_per_epoch])
    opts.vocab_size = len(opts.vocab_words)


    self._examples = examples
    self._labels = labels
    self._id2word = opts.vocab_words
    for i, w in enumerate(self._id2word):
        self._word2id[w] = i
    true_logits, sampled_logits = self.forward(examples, labels)
    loss = self.nce_loss(true_logits, sampled_logits)
    self._loss = loss
    self.optimize(loss)

    # Properly initialize all variables.
    tf.global_variables_initializer().run()
```

由于 Word2Vec 本身的结构较为简单, 仅仅是拥有一个中间层的神经网络, 所以在 Tensorflow 上进行网络的搭建本身是非常简单的事情. 不过要考虑到一个训练批次的问题. 所谓的训练批次, 就是在利用损失函数更新网络参数时, 如果每次只计算一组数据带来的损失, 计算效率过低不说, 还会大大地降低运行的并行度, 让利用 GPU 变得毫无意义. 所以我们每次更新网络参数的时候会综合多组数据计算出的损失, 这多组数据一起就叫一个训练批次. 现在流行的神经网络的实现, 通常都采用以批次为单位来训练的方法.

那么下面的问题, 就变成我们应该选择同时将多少组数据当做一个批次, 进行并行的训练了. 选择的批次过小可能会导致训练数据之间相互影响抵消, 导致结果不能很好的收敛, 而训练批次过大会让算法收敛的很快, 但很有可能会收敛到一个并不是很好的局部最优点, 而几乎不可能摆脱出来. 理由也很好解释. 在批次过小时, 在神经网络的训练过程中, 选择合适的训练批次大小也是一个非常

有意义的话题, 它们都从属于超参数学习的研究范围, 在这里就不赘述了.

以下是 Word2Vec 训练词向量的部分的代码.

```
# From word2vec.py
def main(_):
    opts = Options()
    mConfig = tf.ConfigProto(allow_soft_placement=True)
    mConfig.gpu_options.allocator_type = 'BFC'
    mConfig.gpu_options.per_process_gpu_memory_fraction=0.8
    with tf.Graph().as_default() as graph, tf.Session(config=mConfig)
                                            as session:
        with tf.device("/gpu:0"):
            model = Word2Vec(opts, session)

        for epoch in xrange(opts.epochs_to_train):
            model.train()
            model.save_vec(FLAGS.vector_path, epoch)

        # Perform a final save.
        model.saver.save(session,
                os.path.join(opts.save_path, "model.ckpt"),
                global_step=model.global_step)
```

可以看到还是非常简单的步骤, 设定好一些参数之后就可以进行训练和结果的保存. 可以稍微一提的是, 这里我自己实现了一个支持多线程的 Skip-gram 模型数据读取转换工具, 是为了支持在自己的 CPU 机器上进行高速的并行训练. 不过因为最终训练的场所是 GPU 机器, 这里就没有用上, 也是一种遗憾吧. 这里就稍微贴上一部分代码可以参考一下.

```
#!/usr/bin/python
import threading
class SkipgramReader:
```

```
    """Self-Customed Skipgram Reader"""
    def __init__(self, batch_size, window_size, file_path, word2id):
        self.window_size = window_size
        self.file_path = file_path
        self.batch_size = batch_size
        self._word2id = word2id
        self.EOF = True
        self.batch_lock = threading.Lock()
        self.init_lock = threading.Lock()
        self.initial()

    def get_next_batch(self):
        self.batch_lock.acquire()
        next_batch = []
        for i in xrange(self.batch_size):
            status, res = self.get_next_pair()
            if not status:
                next_batch += [[0,0] for _ in xrange(self.batch_size)]
                break
            next_batch.append(res)
        self.batch_lock.release()
        return self.EOF, next_batch

    def initial(self):
        self.init_lock.acquire()
        if self.EOF:
            self.fd = open(self.file_path, 'r')
            self.cur_char = ' '
            self.word_queue = list()
            self.cur_pos, self.cur_off = 0, 0
            for i in xrange(2*self.window_size+1):
                self.insert_next_word()
            self.EOF = False
```

```
37            self.init_lock.release()
38
39    ...
```

3.2.2 词聚类与文章向量生成

在获得了由 Word2Vec 利用了学术论文数据库中的摘要而训练处词向量之后, 我们接下来考虑的就是如何利用这一组词向量来构成文章向量. 那么和在生成词向量之前同样的, 我们就必须在这么做之前考虑应该怎么做, 为什么要这么做的问题. 首先为什么要生成文章向量呢? 那是因为考虑到我们的目的, 是为了最后的学术会议的相关性研究以及以此为基础的推荐算法的实现, 所以我们在之前的章节2.3中介绍的算法里面考虑过后, 发现基于内容的推荐算法是比较适合我们这整个系统的实现的.

第一点就是基于协同过滤的算法的评价矩阵在这个场景下是很难取得的, 学术搜索引擎的使用者比起电商网站和各种文章网站的使用者来说, 目的性更为强烈. 而基于内容的推荐算法由于只需要考虑目标本身的数据, 加上我们已经借由 Word2Vec 模型获得了整个数据集的词向量, 以此来进行文章的向量化特征提取就是一件水到渠成的事情了.

那么我们怎么做到这件事情呢? 在2.3.2中我们也曾经研究过这一点, 借由之前的结论我们可以发现, 做词聚类后的文章向量生成应该是一个比较合理的方案. 首先, 学术论文中专业名词的比例会远远高于普通论文, 特别是在需要在很短的篇幅中让人清楚地大体认识到整个文章内容的摘要里面, 就更是如此了. 那么我们经由摘要训练出的词向量, 聚类之后的结果显然是会很具有代表性的. 其次, 词聚类还能避免很多词向量之间的影响导致信息的流失, 这一点之前也有提出过了, 就不再赘述.

至于对聚类算法的选择, 这里我们在2.2.2中已经讨论过了各个不同种类的

聚类算法之间的差异和优劣比较. 作为比较大数据集合, 并且数据分布在高维空间中, 有很好的区分度的时候, 我们就会采取最为直接的划分式聚类算法. 这里采用的就是普通的 **K-Means** 算法, 由于算法过于简单, 就不在这里再过多描述了.

在得到了词聚类之后, 我们就采用 **tf-idf** 的方法来进行文章向量的构建, 作为变量的单位当然是聚类. 以下是供参考的部分代码片段.

```
# From doc2vec.py
from sklearn.preprocessing import scale


def load_clusters(path):
    word2cluster = dict()
    file_list = os.listdir(path)
    for each in file_list:
        if each in ['.', '..']: continue
        with open(os.path.join(path, each), 'r') as f:
            for line in f:
                try:
                    cluster_index, word = line.strip().split(',')
                    word2cluster[word] = int(cluster_index)
                except ValueError as detail:
                    print("ValueEroro in loading clusters: ", detail)
                    pass
    return word2cluster
word2cluster = load_clusters(path_cluster)
multiple_space = re.compile(r'\s+')
illegal_char = re.compile(r'[^A-Za-z]')

# Clean illegal characters and redundant space
def wash_doc(raw):
    # Wash dirty data
```

```
    clean = multiple_space.sub(" ", illegal_char.sub(' ', raw))
    return clean

def gen_doc_vec(doc):
    global word2cluster
    new_doc_vec = [0.0 for _ in xrange(opt.vec_dim)]
    for word in doc:
        new_doc_vec[word2cluster.get(word, 0)] += 100
    new_doc_vec_scaled = scale(new_doc_vec).tolist()
    return new_doc_vec_scaled

def write_doc_vec(doc_vecs, path):
    assert len(doc_vecs) > 0
    delim = " "
    with open(path, 'w') as f:
        for vec in doc_vecs:
            f.write(delim.join(map(str, vec)))
            f.write('\n')
```

3.2.3 会议向量生成与相关性分析

在得到了文章向量之后, 我们要做的最后一步就是生成对应的会议向量了. 当然到了最后一步我们仍然不能盲目地尝试, 还是要思考我们的目的以及为了达成目的而选择的手段. 得到了以词聚类频率为分量的文章向量, 那么可以说我们的系统已经完成了一大半. 因为这个时候的文章向量, 配合上词聚类本身的意义, 是非常容易解读的. 什么样子的单词在这篇文章的摘要里出现, 很直接地就决定了这篇文章背后所蕴含的学术成果的方向. 很明显地是, 这个时候我们自然大可以像一些文章网站所采用的简单的推荐系统那样, 直接利用文章向量之间的相似度, 来做基于内容的推荐系统. 但那样做会碰到两个问题.

第一, 跳过会议向量的生成, 直接进行推荐, 是无法达成我们想要研究学术

会议之间的相关性的目的的. 通过分散的学术论文, 我们很难很直观地去描述它们所对应的那个学术会议究竟是怎么样的存在. 或者说, 很难用一种机器的方式, 去直接通过文章向量描述一个学术会议.

第二, 会议中的文章和文章所投搞发表的会议之间所关注的主题, 我们不应该看做是一个单方向的蕴含关系. 并不仅仅是由发表的文章决定了某个学术会议所关注的主题. 同样的, 研究者们在投稿的时候, 当然很自然地会去考虑他们的文章应该往哪一个学术会议去投稿发表. 学术会议本身在召开前, 邀请投稿的时候, 也都会给出自身所需稿件的研究主题的范围. 这二者是一个相互影响的关系, 而仅仅采用文章向量去描述一个会议, 就会损失掉这部分的信息.

那么我们的选择就是, 将一个学术会议中的所有文章向量都集合起来, 综合地表示一个会议. 之后在考虑反过来利用这个会议向量, 来去做文章之间的推荐. 那么怎么去生成这个会议向量呢? 这时候我们就可以采用很简单的向量均值的方法了. 因为这时候需要的是将所有文章的方向都综合到一个向量里去, 这样平均出来的向量, 自然就反映出了不同会议之间不同的侧重点. 而根据之前生成文章向量的方法, 在2.2.1中介绍的度量方法里, 经过整体平均后的会议向量之间, 余弦相似性也依然是一个很好的度量距离的方法.

那么这样我们就得到了一个能够计算所有会议两两之间关系的方法了. 但是这样还不够好, 因为仅仅是这样, 我们得到的关系集合就会是一个非常巨大的稠密矩阵. 而事实上, 不仅仅在学术会议之间, 在很多足够大的集合里, 对象之间的关系矩阵往往都是比较稀疏的. 我们往往不会去考虑一个关注生物制药的会议和一个历史学会议之间的关系. 同样的, 我们也不需要一个数字去度量两个关系很远的, 相似度很低的会议之间的关系, 我们完全可以当他们没关系嘛.

回忆起2.2.3中介绍的方法, 我们本质上也是一种降维的行为, 在不考虑远距离的关系的时候, 采用流型学习中的 **LLE** 方法就很符合我们的需求. 当然实际上我实现的是我自己根据之后可视化和推荐的需求以及学术会议之间相关的实

际情况定制的一种根据 LLE 改编的算法.

这么做的原因同样有两个. 第一是在进行对前一步得到的会议向量的检查时我发现了, 有时候仅仅是相关度的绝对值并没有办法完全的反映两者之间的关系, 可能存在某个会议节点, 它仅仅与另外的一个会议关系比较接近, 但同时这个另外的会议又和其他的几个会议关系更为紧密, 那么这时候我们单单根据绝对值的大小而舍弃掉这个独立的节点的话, 实际上从这个会议的角度上来看是没有什么道理的. 并且在现实中, 也不会有完全独立的学术会议, 和其他学术会议完全没有交集.

第二则是可视化时希望能够清楚地将相似的一类会议聚在一起, 形成比较明显的类簇. 但同时也希望保持类别之间的连接.

考虑到这两点, 我最后在实现会议节点之间相似性分析时, 是同时从两个节点的角度来考虑这个问题. 为了更好的度量之间的关系, 我舍弃掉了余弦相似性的绝对值, 而是利用了最接近的邻居排名, 两者相互影响, 来决定相似的程度.

参考代码片段如下

```
# From genconfervec.py
def cosine(a, b):
    assert len(a) == len(b) and type(a[0]) == type(b[0]) == type(0.0)
    A, B = map(lambda e: math.sqrt(sum(map(lambda c:c*c, e))), [a,b])
    A_B = sum(map(lambda c: c[0] * c[1], zip(a,b)))
    try:
        res = A_B / (A * B)
    except ZeroDivisionError:
        return 0.0
    return res

def cosine_dis(tar, vecs, tags):
    assert  type(tar[0]) == type(vecs[0][0]), "Type Error"
    distances = list()
```

```
    for i, each in enumerate(vecs):
        cos_dis = cosine(tar, each)
        distances.append(cos_dis)
    res = zip(distances, tags)
    res.sort(key=lambda c:c[0], reverse=True)
    return map(lambda c:c[1], res[1:11])

def gen_edges():
    weights = [pow((10-i)/10.0, 0.5) for i in xrange(10)]
    threshold = 0.5
    confer_vec = read_data().tolist()
    confer_tags = read_tag()
    confer_tags = map(lambda c:c[0], confer_tags)
    tag2num = dict([[b, a] for a, b in enumerate(confer_tags)])
    edges = dict()
    for confer in confer_tags:
        src_id = tag2num.get(confer)
        target = confer_vec[src_id]
        neighbors = cosine_dis(target, confer_vec, confer_tags)
        for index, neighbor in enumerate(neighbors):
            dst_id = tag2num.get(neighbor)
            edge_id = min(src_id, dst_id) * 10000 + max(src_id, dst_id)
            weight = weights[index]
            if edges.get(edge_id, -1) != -1:
                edges[edge_id] = (edges[edge_id] * 1.8) *  weight
            else:
                edges[edge_id] = weight / 1.8
```

3.2.4 推荐系统及可视化

终于来到了最后一步, 得到了会议之间相似性度量的系统已经可以说是基本完成, 只差最后的临门一脚了. 我们根据上一步生成的这些经过简化降维后的

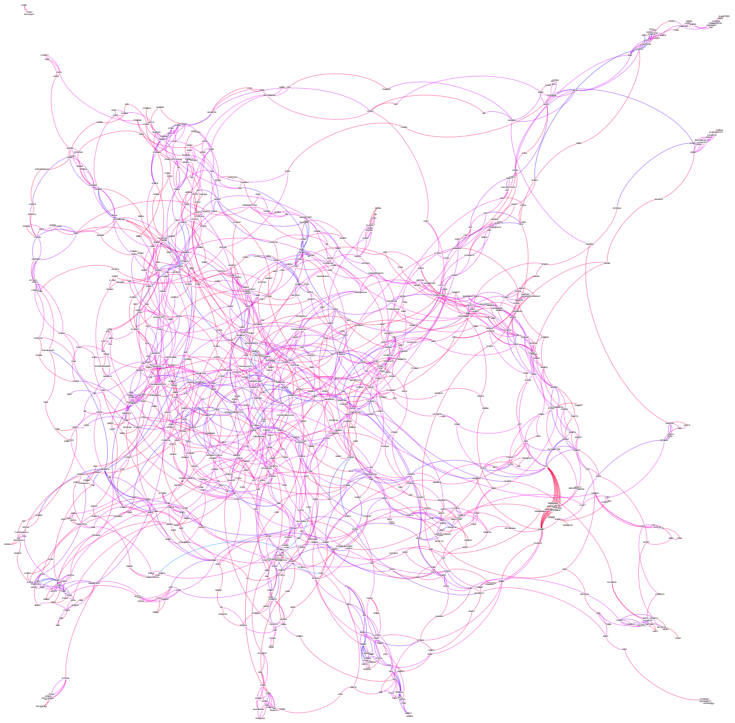


图 3–2 可视化全景

相似度矩阵, 可以直接利用 Gephi 生成学术会议节点图. 整体效果如图3–2

如果我们把镜头拉近, 可以看到节点之间的关系, 脉络清晰可见, 而且相关度的展示也很符合实际情况.

图3–3展示了具体的细节, 选择放大的区域是以 MOBICOM,INFOCOM 等顶级会议为首的一系列网络, 移动互联网相关会议. 很显然, 研究这些方向的研究者, 对于这些学术会议应该都是比较熟悉的. 而在阅读了其中一个学术会议论文

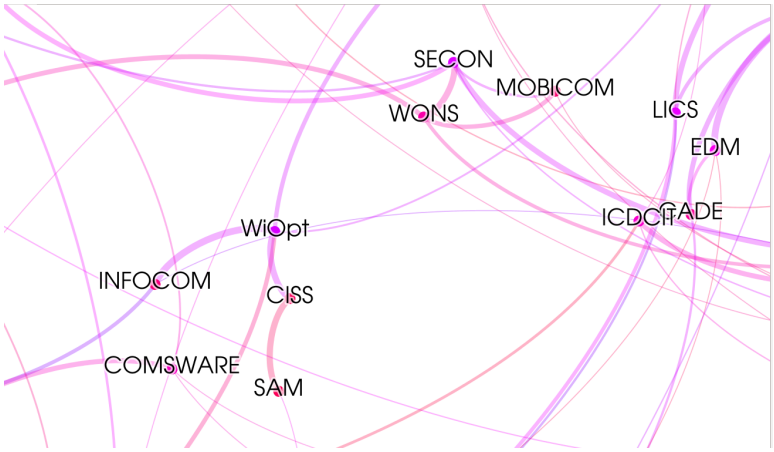


图 3–3 可视化细节

后, 那么我们也很自然地可以认为他对周围这几个会议中发表的学术论文会更有兴趣.

那么我们的推荐系统就可以围绕这一点来进行. 一下就能想到的做法是按照论文所属会议, 锁定距离比较接近的一些会议后, 再来根据文章向量进行限定区域的论文推荐. 这当然是一种合理的做法, 但是还不够好, 因为同样的会议中, 也是包含了很多的方向的论文的. 那么这些论文的方向, 本身可能会根据不同的学术会议, 会有少许的区别, 这也是我们之前提到的会议本身对论文方向的影响. 那我们要如何从这文章中剔除掉这里的影响呢?

看起来似乎很难, 因为我们一直做的都是利用文章去构建会议, 反过来怎么做就很让人发愁了. 所幸, 我们用于构建学术会议向量的方法, 是去取文章向量的平均值. 当时我们利用了聚类生成文章时词聚类频分布表示, 可以进行向量加减的性质. 那么同样的, 我们何不反过来, 从文章向量里减掉会议向量, 不就很好的去掉了会议的影响了吗?

实际上, 推荐系统也确实是按照这个思路实现的. 我们在根据会议向量找到候选会议之后, 再分别的求出其中每篇文章的相对向量

$$\mathbf{V}_{\text{relative}} = \mathbf{V}_{\text{paper}} - \mathbf{V}_{\text{conference}} \tag{3–1}$$

然后再来根据这个相对向量来进行推荐. 到这里, 我们就完成了整个系统. 推荐系统的参考代码片段如下

```python
# From recomm.py

def _vec_dis(a, b):
    assert len(a) == len(b) != 0, "Dimension Error in _vec_dis"
    A, B = map(lambda e: math.sqrt(sum(map(lambda c:c*c, e))), [a,b])
    A_B = sum(map(lambda c: c[0] * c[1], zip(a,b)))
    try:
        res = A_B / (A * B)
    except ZeroDivisionError:
        return 0.0
    return res

def recommend(cur_paper_id, k):
    cur_doc_vec = look_up_doc_vec(cur_paper_id)
    cur_confer = paper2confer[cur_paper_id]
    confers = get_confer_neighbor(cur_confer)
    relative_doc_vecs = gen_relative_doc_vec(confers)
    vec_dis = lambda a: _vec_dis(gen_relative_doc_vec(cur_paper_id), a)
    k_recomms = sorted(
        key=lambda c:c[1],
        map(vec_dis, relative_doc_vecs),
        reverse=True
        )[:k]
    return map(lambda c:c[0], k_recomms)
```

全文总结

本文主要介绍从头实现一个学术大数据推荐系统的过程. 该推荐系统利用了学术大数据中的会议关系并且将其可视化了, 以此来提供了更加个性化的推荐服务.

参考文献

[1] 夏旭. 基于 Google 学术搜索的引文检索研究 [J]. 情报理论与实践, 2006, 29(6):697–701.

[2] Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality[C]//Advances in neural information processing systems. .[S.l.]: [s.n.] , 2013:3111–3119.

[3] Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.

[4] 孙吉贵, 刘杰, 赵连宇, et al. 聚类算法研究 [J]. 软件学报, 2008, 19(1):48–61.

[5] 邢永康, 马少平. 统计语言模型综述 [J]. 计算机科学, 2003, 30(9):22–26.

[6] Fink G A. n-Gram Models[M]//Markov Models for Pattern Recognition.[S.l.]: Springer, 2014:107–127.

[7] Bengio Y, Ducharme R, Vincent P, et al. A neural probabilistic language model[J]. Journal of machine learning research, 2003, 3(Feb):1137–1155.

[8] Gutmann M, Hyvärinen A. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models.[C]//AISTATS. .[S.l.]: [s.n.] , 2010, 1:6.

[9] Kumar P, Krishna P R, Bapi R S, et al. Rough clustering of sequential data[J]. Data & Knowledge Engineering, 2007, 63(2):183–199.

致　　谢

感谢上海交通大学！

感谢所有测试和使用交大硕士学位论文 LaTeX 模板的同学！

感谢那位最先制作出博士学位论文 LaTeX 模板的交大物理系同学！

感谢 Jianwen(水源 ID：shinkansen) 为此模板做出的贡献！

感谢 William Wang 同学对模板移植做出的巨大贡献！

感谢 Wang 同学对推动模板官方化所做的工作！

Connection between Conference in Academic Big Data

An imperial edict issued in 1896 by Emperor Guangxu, established Nanyang Public School in Shanghai. The normal school, school of foreign studies, middle school and a high school were established. Sheng Xuanhuai, the person responsible for proposing the idea to the emperor, became the first president and is regarded as the founder of the university.

During the 1930s, the university gained a reputation of nurturing top engineers. After the foundation of People's Republic, some faculties were transferred to other universities. A significant amount of its faculty were sent in 1956, by the national government, to Xi'an to help build up Xi'an Jiao Tong University in western China. Afterwards, the school was officially renamed Shanghai Jiao Tong University.

Since the reform and opening up policy in China, SJTU has taken the lead in management reform of institutions for higher education, regaining its vigor and vitality with an unprecedented momentum of growth. SJTU includes five beautiful campuses, Xuhui, Minhang, Luwan Qibao, and Fahua, taking up an area of about 3,225,833 m2. A number of disciplines have been advancing towards the top echelon internationally, and a batch of burgeoning branches of learning have taken an important position domestically.

Today SJTU has 31 schools (departments), 63 undergraduate programs, 250 masters-degree programs, 203 Ph.D. programs, 28 post-doctorate programs, and 11 state key laboratories and national engineering research centers.

SJTU boasts a large number of famous scientists and professors, including 35 academics of the Academy of Sciences and Academy of Engineering, 95 accredited professors and chair professors of the ”Cheung Kong Scholars Program” and more than 2,000 professors and associate professors.

Its total enrollment of students amounts to 35,929, of which 1,564 are international students. There are 16,802 undergraduates, and 17,563 masters and Ph.D. candidates. After more than a century of operation, Jiao Tong University has inherited the old tradition of ”high starting points, solid foundation, strict requirements and extensive practice.” Students from SJTU have won top prizes in various competitions, including ACM International Collegiate Programming Contest, International Mathematical Contest in Modeling and Electronics Design Contests. Famous alumni include Jiang Zemin, Lu Dingyi, Ding Guangen, Wang Daohan, Qian Xuesen, Wu Wenjun, Zou Taofen, Mao Yisheng, Cai Er, Huang Yanpei, Shao Lizi, Wang An and many more. More than 200 of the academics of the Chinese Academy of Sciences and Chinese Academy of Engineering are alumni of Jiao Tong University.