中图分类号：TP3

论文编号： 10006GS11211B8

北京航空航天大學

专业硕士学位论文

微博垃圾信息过滤系统的设计与实现

作者姓名　　于俊超

学科专业　　软件工程

指导教师　　邵兵　　谭火彬

培养院系　　软件学院

Design and Implementation of Weibo Spam Filtering System

A Dissertation Submitted for the Degree of Master

Candidate：Yu Junchao

Supervisor：Shao Bing

College of Software Engineering

Beihang University, Beijing, China

中图分类号：TP3

论文编号：10006GS11211B8

硕 士 学 位 论 文

微博垃圾信息过滤系统的设计与实现

作者姓名 于俊超 申请学位级别 工程硕士

指导教师姓名 邵兵 职 称 副教授

学科专业 软件工程 研究方向 移动云计算

学习时间自 2011 年 9 月 20 日起至 2013 年 12 月 31 日 止

论文提交日期 2013 年 11 月 14 日 论文答辩日期 2013 年 12 月 14 日

学位授予单位 北京航空航天大学 学位授予日期 年 月 日

关于学位论文的独创性声明

本人郑重声明：所呈交的论文是本人在指导教师指导下独立进行研究工作所取得的成果，论文中有关资料和数据是实事求是的。尽我所知，除文中已经加以标注和致谢外，本论文不包含其他人已经发表或撰写的研究成果，也不包含本人或他人为获得北京航空航天大学或其它教育机构的学位或学历证书而使用过的材料。与我一同工作的同志对研究所做的任何贡献均已在论文中做出了明确的说明。

若有不实之处，本人愿意承担相关法律责任。

学位论文作者签名：_____ 日期：　　年　　月　　日

学位论文使用授权书

本人完全同意北京航空航天大学有权使用本学位论文（包括但不限于其印刷版和电子版），使用方式包括但不限于：保留学位论文，按规定向国家有关部门（机构）送交学位论文，以学术交流为目的赠送和交换学位论文，允许学位论文被查阅、借阅和复印，将学位论文的全部或部分内容编入有关数据库进行检索，采用影印、缩印或其他复制手段保存学位论文。

保密学位论文在解密后的使用授权同上。

学位论文作者签名：_____ 日期：　　年　　月　　日

指导教师签名：_____ 日期：　　年　　月　　日

摘　要

随着科技的迅速发展，尤其是大数据时代的到来，出现在我们身边的信息越来越丰富密集，来带的直接现象是我们会对于接收到的信息眼花缭乱。目前互联网的用户们所面临的问题已经不是以前科技不发达的时候的信息匮乏问题而是如今大量各种数据出现而导致的信息过载问题。所以如何对我们感兴趣的信息进行过滤，避免在我们获取信息的过程由于受到过多的干扰而浪费宝贵的时间这样的研究工作具备重要的意义。

本文对一种微博垃圾信息过滤系统的设计与实现进行了详细论述，同时对系统设计与实现中所采用的一种贝叶斯分类器的算法进行了研究优化。本文所介绍的整个过滤系统本质上是一套分类系统，重点在于如何过滤掉我们不想要的信息，而不在于推荐和挑选我们可能专注需要的信息。同时为了更好的扩展与适应未来的发展，文中所介绍的分类器的架构实验工具可以灵活的扩展并进行效果测试验证，使得这一套内容分类过滤解决方可以应用到短信、邮件甚至更多的文本分类应用。

有关过滤器的实验非常成功，目前使用这样系统验证下所产生的正式的产品已投入上线运行，可以在广告以及色情等信息爆发出现的时候自动开启并过滤删除这些信息，对于提高网站内容质量，杜绝漫天垃圾信息带来的危害，优化网站用户的阅读浏览体验具有十分重要的意义与价值。

关键词：文本过滤，分类器，朴素贝叶斯

Abstract

With the rapid development of science and technology, especially the big data time is coming, appears around us more and more varies of information, then we will see things a blur to the received information. Internet user now have problems of too much information come into sight instead of no much enough informations. So how about we are interested in information filtering, to avoid in the process we obtain information due to too much interference and waste valuable time has important significance of such research.

This article on the design of a micro-blog spam filtering system and implementation in detail, at the same time, a classifier mainly use Bayesian system design and implementation of the algorithm is studied to optimize. Essence of the whole filtering system described in this paper is a classification system, the key point in this article is how to filter out unwanted information but commendation and selection on focusing needed information. At the same time in order to extend and adapt to the future development, extended architecture experiment tool of the classifier can flexibly and effect test, making this a content classification filtering solution can be applied to text messages, emails and even more text classification should be used.

The research framework code on filters were very successful, the use of such a system, products have been put into operation, to improve the quality of site content, has the vital significance to the garbage information brings harm. Optimizing a website user browsing has significance and important value experience.

Key words: Document Filtering, Classifier, Na ïve Bayes

目 录

图目录

表目录

第一章 绪论

1.1 课题来源和意义

截至2011年，互联网已经走过了40多年历程，相对人类上下五千年文明史来讲，40年是极为短暂的，但仅仅是这40多年，互联网为整个世界带来了翻天覆地的变化。正因为互联网的普及，人们的生活才变得更加便利，更加丰富多彩，人与人、与社会、与世界之间的距离也越拉越近。据2011年1月19日中国互联网络信息中心（CNNIC）发布的《第27次中国互联网络发展状况统计报告》数据显示，截至2010年12月，中国网民规模达4.57亿人，互联网普及率在稳步上升，稳居世界第一位，其中手机网民规模达到3.03亿人，占整体网民的66.2%。由此可以看出，移动网络、手机终端在中国互联网发展中起着更加重要的作用。

搜索引擎、即时通信、网络视频、博客应用、论坛/BBS、电子邮件、网络购物等网络应用依然是广大网民首选，排在网民网络应用前十位。而社交网站和网络文学是2009年度新兴网络应用，微博客和团购是2010年新兴网络应用。微博客类应用反应了中国网络应用的新特点，使信息互通更加便利、快捷，也促进了各地区、各民族的沟通交流，推动了和谐社会的构建进程。

在这些Web 2.0运营模式下的网站，由于用户有足够的权限发布各种各样的内容，而这些内容，大多数互联网企业在企业发展初期为了节约成本不会投入太多的人力成本去监管这些内容，再加上现在网络营销概念的出现，导致我们每天在获取有效信息的时候会遇到大量的噪音信息，比如各种各样的广告，就会对我们的浏览过程造成大量的干扰，所以实现一种高效的文本过滤器筛掉我们不想要的信息，对于我们更高效更快捷的获取信息有着十分重要的意义。

在我实习的某世界500强公司所属的门户网站中，也出现了网络营销的现象：由于大型门户网站具备用户浏览量大的特征，同时发布评论等信息的门槛又特别低，只需要注册一个用户完成简单的认证流程就可以发布任意的文字信息，在这种情况下，利用门户网站的评论功能发布广告是成本非常低廉的营销手段，正所谓天下熙熙皆为利来，天下攘攘皆为利往，即使是大公司的门户网站也难逃被垃圾信息灌满的厄运，所以网站急需一套系统可以过滤网站的所有者们不希望在网站中看到的内容。在提高网站用户的阅

读浏览体验的同时，也维护了公司本身的利益。

1.2 课题国内外研究现状分析

1.2.1 文本分类器算法

国外对文本自动分类的研究开展较早，50 年代末，H. P. LUhn 在这个领域进行了开创性的研究，提出了基于词频统计思想的文本自动分类方法。1960 年，Maron 发表了关于自动分类算法的第一篇论文，随后以 K. Spark，G. Salton 以及 K. S. Jones 等人为代表的众多学者也在这一领域进行了很有成效的研究工作，目前国外的文本分类研究已经从实验性阶段进入到了实用化阶段，并在邮件分类、电子会议等方面取得了广泛的应用，其中较为成功的有麻省理工学院为白宫开发的邮件分类系统和卡内基集团为路透社开发的 const rue 系统。

相比于英文文本分类，中文文本分类的一个重要的差别在于预处理阶段:中文文本的读取需要分词，不像英文文本的单词那样有空格来区分。在很长一段时间内，中文文本分类的研究没有公开的数据集，使得分类算法难以比较。现在一般采用的中文测试集有：北京大学建立的人民日报语料库、清华大学建立的现代汉语语料库等。其实一旦经过预处理将中文文本变成了样本矢量的数据矩阵，那么随后的文本分类过程和英文文本分类相同，也就是随后的文本分类过程独立于语种。因此，当前的中文文本分类主要集中在如何利用中文本身的一些特征来更好地表示文本样本。国内对于文本自动分类的研究起步较晚，但从简单的查词典的方法，到后来的基于统计语言模型的分词方法，中文分词的技术已趋于成熟。

近年来，文本分类已成为众多领域研究者的热门研究课题 ,研究者们从不同的角度把越来越多的知识引入文本分类领域，推动着文本分类的不断发展，产生了许多新的方法。

(1) 基于群的分类方法。这种方法可以看作是进化计算的一个新的分支，它模拟了生物界中蚁群、鱼群和鸟群在觅食或者逃避敌人时的行为。纵观文献中对基于群的分类方法的研究。对 ACO 或者 PSO 在数据挖掘中应用的研究仍处于早期阶段，要将这些方法用到实际的大规模数据挖掘中还需要做大量的研究工作。

(2) 基于模糊粗糙集的文本分类模型。文本分类过程中由于同义词、多义词、近义

词的存在导致许多类并不能完全划分开来,造成类之间的边界模糊。粗糙集理论有机的结合了模糊集理论与粗糙集理论在处理不确定信息方面的能力。它们处理的是两种不同类别的模糊和不确定性，将两者结合起来的模糊粗糙集理论能更好地处理不完全知识。

(3) 多分类器融合的方法。实际应用的复杂性和数据的多样性往往使得单一的分类方法不够有效。因此学者们对多种分类方法的融合(fusion)进行了广泛的研究，取得了一系列研究成果。纵观文献中的研究，可以大致将多分类器的融合技术分为以下几类：投票机制(Voting)、行为知识空间方法(Behavior Knowledge Space BKS)、证据理论(Dempster shafer theory)、贝叶斯方法和遗传编程(Genetic programming GP)。

(4) 基于 RBF 网络的文本分类模型。把监督方法和非监督方法相结合 ,通过两层映射关系对文本进行分类，首先利用非监督聚类方法根据文本本身的相似性聚出若干个簇，使得每个簇内部的相似性尽可能高而簇之间的相似性尽可能低，并由此产生第一层映射关系即文本到簇的映射，然后通过监督学习方法构造出第二层映射关系，即簇集到目标类集合的映射，然后为每一个簇定义一个相应的径向基函数(Radial Basis Function，RBF)，并确定这些基函数的中心和宽度，利用这些径向基函数的线形组合来拟合训练文本，利用矩阵运算得到线性组合中的权值，在计算权值时，为了避免产生过度拟合的现象，采用了岭回归技术，即在代价函数中加入包含适当正规化参数的权值惩罚项，以保证网络输出函数具有一定平滑度。

(5) 潜在语义分类模型。潜在语义索引方法，已经被证明是对传统的向量空间技术的一种改良，可以达到消除词之间的相关性，化简文档向量的目的。

(6) K 近邻算法(KNN)的新发展：KNN 是一种有效的分类方法，但是它有两个最大的缺陷：第一，由于要存储所有的训练实例，所以对大规模数据集进行分类是低效的；第二，KNN 分类的效果在很大程度上依赖于 k 值选择的好坏。实验证明,基于 KNN 模型的方法在分类精确度上与 C5.0 和标准的 KNN 相当。另外，针对 KNN 方法的第一个缺陷，Nong Ye and Xiangyang Li 将聚类方法和经典的 KNN 方法结合起来，提出了一种新颖的分类方法，称为 CCA2S。CCA2S 能够处理大规模数据集，可伸缩性好，并且支持增量式学习。但 CCA2S 只能处理连续属性，而且只针对类别为两类的分类问题。如何扩展 CCA2S，以使其能够处理多类别的问题,还有待进一步研究。

(7) 支持向量机(SVM)方法的新发展: SVM 是进行分类、 聚类和时间序列分析的有

效数据挖掘工具。但是，由于 SVM 的训练时间会随着数据集的增大而增加，所以在处理大规模数据集时，SVM 往往需要较长的训练时间。而实际的数据挖掘应用往往包含了数以百万计的数据，这使得 SVM 很难发挥作用[12]。

1.2.2 中文分词算法

在国外，已经有很多诸如 Akimist 的厂商在做 Spam Detection 方面的产品。然而在中国，由于中文的特殊性，需要复杂的分词算法才能很好的分离出特征，所以目前尚没有出现成熟的中文垃圾信息过滤器的产品面世。

所以针对中文分词算法的调研之后，我们可以获得这样的信息，目前主流的分词算法，大概已经出现了以下几个广义的方向：

A. 基于词典的分词算法（机械分词）

算法主要依赖于词典匹配，主要分为以下几种算法

I. 正向最大匹配

II. 逆向最大匹配

III. 全二分最大匹配算法

这种算法的由于过于依赖词典，对于未登录的词的补充很难实现。

B. 基于统计的分词算法

目前基于统计的分词算法有很多种，较为常见的算法有如下：

I. 互信息的概率统计算法

II. N-Gram 模型算法

III. 组合度决策算法

这种算法也有局限，对于常用的词识别精度很差。

C. 基于规则的分类算法

这种算法由于中文语言的复杂性限制，导致目前还很不成熟[10]。

1.2.3 其他存在的问题

A. 目前现存的分类器虽然正确率较高，但是依然会导致很多误报，无法应用于实际产品当中，不具备足够的市场价值。

B. 中文特征的提取问题，诸如火星文广告，如何对这些信息进行处理又将是一个需要解决研究的问题。

1.2.4 结论

由于目前采取什么样的分词以及分类器方案效果最好并没有明确的定论，所以系统决定采取松耦合的方式，将目标任务分成若干个阶段，然后每个阶段实现不同的算法，然后通过实验确定最终最有效的算法策略。

1.3 课题研究目标及内容

1.3.1 研究目标

本文论述了微博垃圾文本信息过滤系统的设计实现过程，通过该系统的实施，解决了用户发布的垃圾信息充斥在微博平台的问题，同时中文文本分类系统原型的灵活架构可以对各种各样平台出现的内容进行实验测试，并可以针对这个渠道出现的文本进行特征分析提取，并形成新的分类策略适用于定制化的使用场景。

1.3.2 研究工作内容

针对上述用户发布的垃圾信息充斥在微博平台的问题，提出了利用机器学习方法应用与文本分类识别环境的方案，研究的最终目标是会实现一个中文文本分类器的原型系统，这个原型系统具备以下功能：

A. 特征提取模块：可以完成从大段的文本中整理出每个文本的信息，并使用中文分词算法提取出每个文本所具备的的特征，并将其存储到文件中以便观察。

B. 源数据分发模块：将已经分号类别的素材分配到不同用途，分别用来做训练数据和测试数据。

C. 多分类器实现模块：可以针对已经提取好特征和标记过类别的数据对分类器进行训练，并可以将训练结果输出，训练结果可以被其他分类器直接继承。

D. 结果统计报告模块：可以对一个训练好的数据分类器进行测试，并对分类器的准确性稳定性等特征进行统计报告。

通过这个系统，我们可以测试各种各样分类方法的准确度，并根据结果进行分析优化后可以达到更好的性能与效果。

1.3.3 个人在项目中承担的主要工作

根据实际的研究目标内容，个人主要负责整个研究工作的如下几个方面：

A．基础研究数据收集工作

B．数据人工标记分类工作

C．代码编写工作

D．第三方工具研究应用工作

E．测试收集数据工作

F．对结果中的偏差所做各种针对性优化的工作

1.4 本文组织结构

本论文分为五章，具体安排内容如下：

第一章是绪论部分，介绍课题的来源、研究背景、国内外研究现状以及论文的目标与主要内容。

第二章是系统的需求分析，对整个系统将要面临的环境进行了明确的阐述，并对接下来的设计实现进行了预估。

第三章是总体解决方案的设计，介绍了所有研究工作的流程设计以及实验源数据准备工作，详细讲述了实验数据收集的策略过程以及标记工具的建立与使用。

第四章是系统的详细设计实现，介绍了整个过滤系统的平台架构的设计以及子分类器的设计实现，介绍了每个细分的模块的详细设计以及对于的第三方库的引用与分类器实验工具适配结合。

第五章是系统测试与验证，介绍了整个过滤器对于实验数据的运行结果，已经后续如何针对实验结果进行的优化。

第六章是总结与展望，在本章中对本课题的设计和研究过程进行了全面的概括和总结，并对未来进行了展望。

第二章 系统需求分析

论文从本章开始，开始对该管理系统的开发进行正是描述。第二章主要通过对本系统所面向文本特点和环境进行了分析，分析了文本的特征环境，建立了分类模型，并提出在系统的设计与实施过程中必须解决的几个主要问题。

2.1 系统的原始需求概述

当下的互联网产业中，微博作为一个新兴的媒体模式在社会信息的传播中起着重大的作用。海量的信息通过微博在互联网用户与用户之间进行流通，同时为信息的的公开性开辟了一条崭新的道路。在我们经历的每一秒中，都会有数以万记的微博被创造出来，而内容也千奇百怪，人们在享受着微博带来的信息畅通无阻的沟通的同时，也产生了垃圾微博所带来的干扰性问题。

垃圾微博的定义主要是那些对用户没有价值的信息，主要的构成为广告微博，同时也包含了一定的色情诈骗等信息，这些微博投放的初衷只是为了吸引用户获得巨大的广告价值，而巨大的投放量导致其变成了严重伤害用户浏览体验的刽子手，同时也损害了平台提供商的利益。

微博过滤，正是要使用机器自动的方法完成这样过滤的功能。它不仅可以将需求的有用信息过滤出来，还可以对信息进行一定的分类整合，用机器自动的将分类信息提供给不同领域将会带来极大的商业价值。

2.2 系统的功能需求分析

既然论文是做文本分类，首先我们要分析一下我们要进行分类的文本环境的特征：

微博作为近几年来发展起来的一种流行的 SNS 模型，正受到越来越多的研究者的关注。微博中的内容相比于传统的文本环境具备以下几种很显著的特征点：

1）每一条微博的长度最多为 140 个汉字字符，所能表达的内容有限，所以同时具备可以被判断的信号量会特别的少，需要我们想尽办法挖掘特征。

2）微博的获取比较方便，国内的各大微博平台基本都有完整的对外开放机制，这相比于我们常见的邮件和网页来讲，对于数据的抓取会更加的方便。

3）语言风格的网络化以及表情的使用，使得微博中所包含的信息更加的丰富多样化，同时因为微博的实时性和娱乐性的大众趋向，所以在内容中还会出现会造成使用大量新生词汇或者出现错别字。

基于以上的比较明显的特征，即使微博的内容形式多种多样，而有用信息并不一定是从开头开始的，有一些有用的信息是从整段的后半部分，有一些就直接不包含有用信息，这对后续的分类工作产生了巨大的影响。

所以基于这种变化性的需求最好的办法就是在机器学习算法的基础之上设计灵活可用的架构，使得不管微博中出现什么样的特征都可以准确的识别并进行分类。

虽然实验数据的标记可以在实验室中进行，但是真正到了产品级的应用当中也依然需要用户来进行数据的报告来进行训练数据的积累，所以这个系统的用例虽然并不复杂但是却具备了相应的功能，如下图所示：

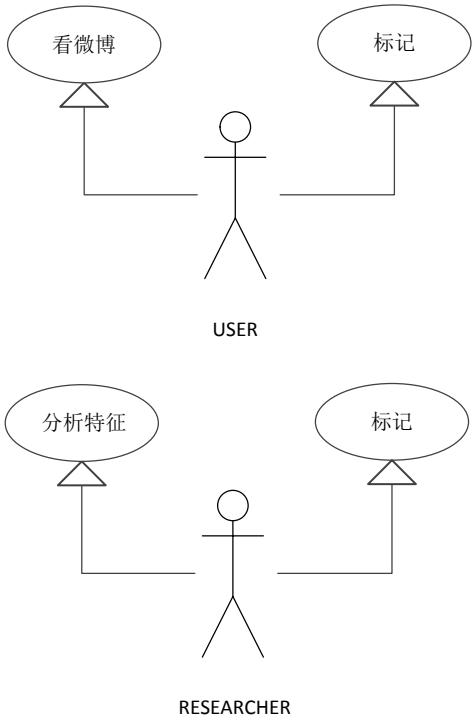

图 1　系统需求的用例图

2.3 系统的质量需求分析

2.3.1 健壮性需求

系统的健壮性主要是指在各种异常的使用场景之下，系统能够依然保证正常运行发

挥其原来应有作用的能力，健壮性包括两种能力，一个是容错能力，另一个则是出现错误之后的恢复能力，健壮性同时也是是软件质量中最基本的质量需求。

由于新浪微博的内容具有其独特的内容风格，同时也有它一定的限制，所以综合软件工程质量的需求，系统可以发挥价值首当其冲的前提就是不管在什么样的语言环境下都能够保证足够的稳定可运行并且完成预期交付的分类工作。如果实在是无法运行的环境，也应当有的相应的日志输出记录进行反馈，方便后续的程序维护和 Bug 修复。

2.3.2 扩展性需求

可扩展性主要指的是软件系统适应变化的需求的能力，可扩展性同时也是系统设计的阶段需要重点考虑的质量属性。

由于新浪微博的属性是用户产生内容，所以微博的内容也会是当今谁会中最前沿的语料库，各种各样的新词诸如“感累不爱，人艰不拆，十动然泼，来信砍……”，以及各种各样的特殊符号比如“ㄟ(▽)ㄏ”这样的颜文字的表达方式都会层出不穷，即使是垃圾微博的信息也一样，所以对于新内容的快速反应则需要在系统设计的阶段重点进行考虑。

2.3.3 正确性需求

在已经成熟的软件工程领域，正确性从来都是独一无的最重要的软件质量属性的需求。但是在最新的前沿领域，尤其是人工智能领域，对于客观世界进行赋予机器的智能的改造已经不可能达到百分之百的正确，所以现在在机器学习的领域里，我们经常用两个数据指标来衡量系统的正确性程度：一个是查全率，是指机器智能完成的任务在所有预期任务中占得比例，另一个是查准率，是指机器已经完成的正确任务在所有完成的任务当中的比例。通过这样两个指标我们可以衡量机器智能的优劣，而在微博垃圾信息过滤的系统设计中，我们对于查全率的要求远大于查准率，因为过滤系统的误报我们错过了重要的信息所带来的损失远大于有些垃圾信息没有被检测过滤出来的危害，所以在最后的系统算法及参数调整的时候，我们需要在保证查准率足够稳定高效的基础上进一步提高查全率。

2.3.4 性能需求

对于性能的需求主要指算法上的时间空间复杂度的需求，而不是仅仅指代软件的运行速度，而对于性能优化主要还是找出各个限制性的瓶颈，可以通过优化数据结构、算法和代码来提高软件运行时的性能。

而对于微博垃圾信息过滤系统来讲，最主要的不是系统的空间问题，因为训练集数量受到标记工作速度的限制所以不会是太大的瓶颈，但是对于系统的过滤过程当中的性能来讲，最大的瓶颈在于训练的计算过程以及分类计算所消耗的时间。

2.4 本章小结

本章对整个系统的需求进行了分析，也使开发所需要完成的目标进行了明确。另外，该章也明确了系统设计最开始所要解决的关键问题，并针对这些问题提出明确适当的解决方案。以上经过分析的内容以及实际的技术架构的设计与实现将在下一章进行阐述。

第三章 系统总体设计

3.1 系统总体结构设计

要想使用带有机器学习功能的分类器去完成信息的过滤，首先我们要知道需要被过滤的信息内容是什么，文章讲的是微博垃圾信息过滤系统，所以我们所要选的数据源会来自各大中文微博平台，实验数据的选择则肯定是使用一定数量的微博数据。其次，数据源选择之后我们要确定如何存储及标记这些数据，使得这些数据可以被当做训练数据集被分类器识别，这里就需要你个标记工具将已经标号的数据封装到文件当中，最后，利用已经完成的标记的训练数据进行测试，来检测整个系统的运行效果然后根据得到的实验数据展开进一步的分析和优化。

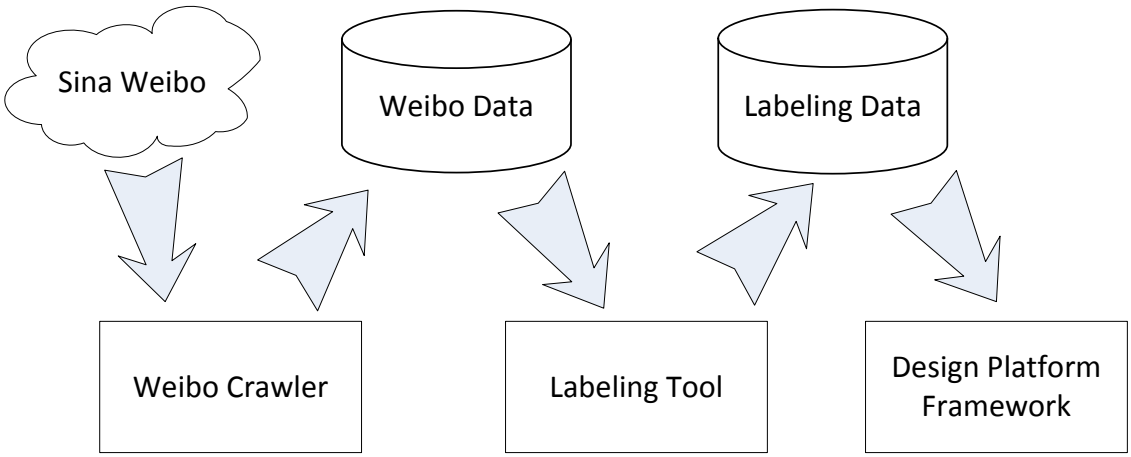

图 2 总体解决方案结构图

如上图所示，整个系统总体的实验工作步骤大概可以被分为下面几部：

首先我们要选择合适的微博的数据源，数据源选择的调研完毕之后则需要编写一个数据爬虫来对微博进行本地化永久存储以备随时实验工作的需要。

其次由于我们是要做的核心工作是设计实现一个查全率和查准率都足够能满足需求的分类系统，又因为整个分类系统将会使用监督学习能力的机器学习算法，所以被人工标记的数据是必需的，而人工标记的工作量繁重，这里就需要用一种机器学习的方法来完成机械重复的标记工作。

最后当所有的标记工作都已经完成了之后，我们就需要开始使用已经标记过的数据开始设计实现最关键的文本过滤系统，这个文本过滤系统将会具备足够的灵活性和健壮

性，以便更好的满足变化多端的文本环境的需求。

3.2 数据源的选择与抓取

当今时下，可以使用的微博类短文本数据源的种类繁多，比如国内的数据由新浪微博、腾讯微博等等，国外的微博类平台比如说 Twitter 等。而本文所选用的数据则采用当下流行的 SNS 新浪微博的 API 来获取相应的数据，这样的方式的优势在于可以更快更好的获取相应的数据。

新浪微博是一个由新浪网公司推出，提供微型博客服务的类 Twitter 网站。用户可以通过网页、WAP 页面、手机客户端、手机短信、彩信发布消息或上传图片。目前新浪微博开放平台用户身份鉴权使用 OAuth2.0 协议，同时提供对 Web，桌面以及移动应用程序的支持。同时新浪微博也在一直对自身的 API 开放平台的服务进行升级和产品的迭代工作，随着 API 的越来越完善，API 开放平台也变得越来越适合开发者针对平台进行实验，同时对于实验研究工作来讲，使用 API 获取数据远比网页的数据爬虫来的效率高得多。

针对上述的情况，实验决定基于新浪微博的数据进行接下来的工作。

选择则了合适的 API 之后，我们需要设计合适的规则去获得我们实验所需要的数据，由于工程测试所需要的实验训练集样本容量并不是海量数据，所以这里不使用关系型数据库作为存储策略，简单的硬盘文件存储序列化后的类就可以足够满足项目的需求。

3.2.1 数据抓取的策略

1）查找特定类型的用户并记录

a.收集可能大量发布广告信息的用户，这类用户的微博基本清一色的是广告信息

b.收集微博的大 V 列表，这类用户基本很少发布广告信息

c.收集有名的草根列表，这类用户胡偶尔发布高质量不容易分辨的广告

d.收集各种不活跃微博的用户列表填补训练集中可能出现的语料库的空缺

2）申请测试用实验账号

3）针对每一类用户抓取流程为：

A．取消当前所有实验账号所有的关注列表。

B．对固定类别的用户列表中的所有用户添加关注。

C．用 Crawler 中新浪微博的的 HomeTimeLine 方法抓去当前关注用户最新发布的所

有微博。

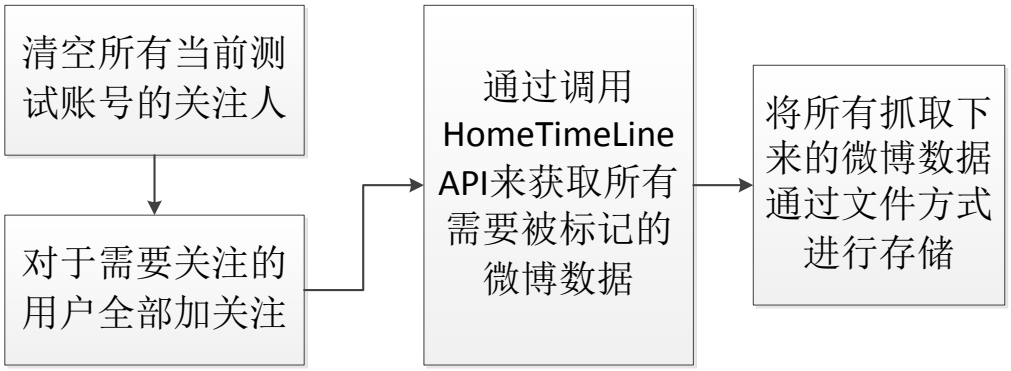

图 3　抓取数据的流程

3.2.2 数据抓取的过程

由于最新的新浪微博 API 已经不允许未经审核过的应用账号使用 UserTimeLine 获取其他用户的 TimeLine 信息，所以抓取过程中需要先将测试账号当前所有关注用户取消关注，清空之后再将所有的需要抓取的信息的账户全部关注，然后调用 HomeTimeLine 来获取信息。

下图则是数据抓取过程中控制台界面的的截图：

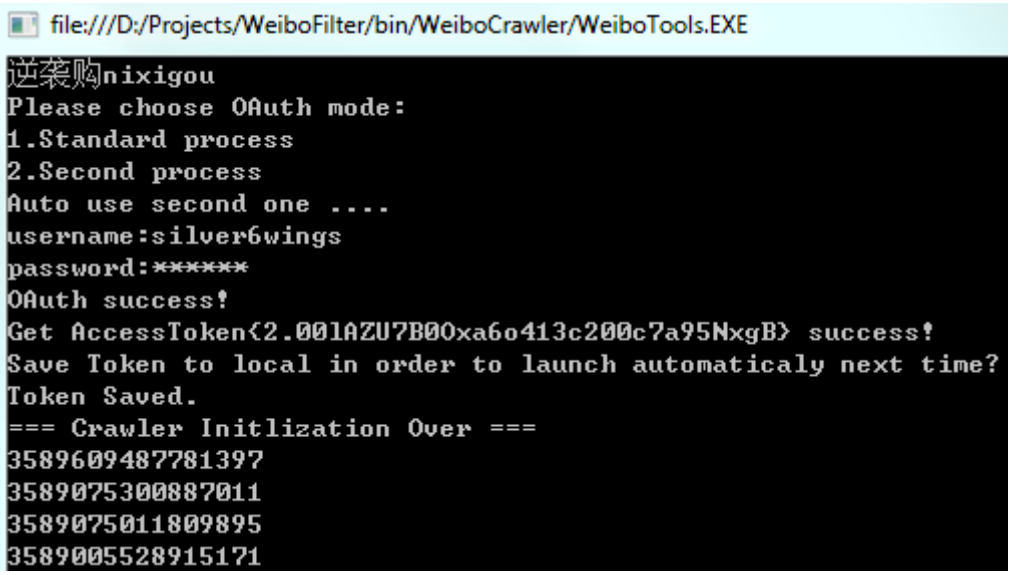

图 4　微博数据抓取的过程的控制台界面

3.3 微博数据授权与存储

新浪微博的数据能抓取下来的前提是我们已经经过了 APP 的授权拿到了相应的权限之后才可以进行的，接下来介绍新浪微博的授权机制，以及对于抓取下来的数据如何存储的问题。

3.3.1 新浪微博 OAuth 验证

为了使用新浪微博开放平台提供的 API(应用程序接口)，需要先注册一个应用。新浪微博会给每一个应用一个专属的 App Key 和 App Secret。Key 跟 Secret 的使用方式跟其他一些协议中的公钥私钥的方案相类似，可以使用自己所熟悉的编程语言将 key 和 secret 结合，为自己发出的每个请求添加签名，以此来向新浪微博开放平台表明自己身份的合法性。

Web 应用应该使用完整的 OAuth 来进行用户认证。桌面以及移动用户也应该使用 OAuth。当然，桌面和移动应用也可以使用 Basic Auth，一种简单的通过用户名密码的方式来进行认证的方式。

大部分 API 的访问如发表微博、获取私信，关注都需要用户身份，目前新浪微博开放平台用户身份鉴权有 OAuth2.0 和 Basic Auth（仅用于应用所属开发者调试接口），新版接口也仅支持这两种方式。OAuth2.0 较 1.0 相比整个授权验证流程更简单更安全，也是未来最主要的用户身份验证和授权方式。

OAuth 的申请过程如下所示：

1）请求签名

所有的 OAuth 请求使用同样的算法来生成(signature base string)签名字符基串和签名。base string 是把 http 方法名,请求 URL 以及请求参数用&字符连起来后做 URL Encode 编码。具体来讲，base string 由 http 方法名，之后是&，接着是过 url 编码(url-encoded)之后的 url 和访问路径及&。接下来，把所有的请求参数包括 POST 方法体中的参数，经过排序(按参数名进行文本排序，如果参数名有重复则再安参数值进行重复项目排序)，使用%3D 替代=号，并且使用%26 作为每个参数之间的分隔符，拼接成一个字符串。

2）获取 Request Token

获取 request token 是进行用户认证的第一步。这一步主要有两个目的：

第一，告诉新浪微博你将要做什么

第二，告诉新浪微博你在 callback 里要做什么

3）用户认证

这一步主要是发送你获取的 oauth_token，并且获得用户的授权。一般来说，WEB 应用会简单的重定向到相应的页面,桌面应用程序会给出 URL 并要求用户自行验证.

新浪微博开放平台的验证 URL 是 http://api.t.sina.com.cn/oauth/authorize

要求必须以 oauth 作为参数,一般来说请求格式如下:

http://api.t.sina.com.cn/oauth/authorize?oauth_token=8lXFOZH5tAwj6vzJYuLQpl0WUEYtWc

如果用户没有登录新浪微博，则会要求用户登录。否则将会出现一个页面，用户可以在此页面上一键同意或者拒绝对此应用授权。用户授权后，web 应用页面将会重定向至你指定的 oauth_callback，如果是桌面应用,将会显示 PIN 码，用户需要将 PIN 码输入你的应用中来完成授权过程。

4）获取 access_token

新浪微博开放平台 access token 请求地址为

http://api.t.sina.com.cn/oauth.access_token

新浪微博开放平台会返回应用需要的信息，包括用户名，oauth_token 以及 oauth_token_secret，然后就可以使用 access token 来 call 相应的 API 了。

图 5 新浪微博 OAuth 验证过程

3.3.2 新浪微博 API 的调用

论文的初期使用的是 2/status/user_timeline 来获取用户信息，此 API 的功能主要为：

获取某个用户最新发表的微博列表。

但是在 2013 年 7 月之后的接口升级过后，2/status/user_timeline 这条 API 就无法获得除自己之外的别人的 API 了，官方提示是建议使用 2/status/home_timeline 来获取当前关注用户最新的微博。接口返回的 JSON 示例如下所示：

```
{
    "statuses": [
        {
            "created_at": "Tue May 31 17:46:55 +0800 2011",
            "id": 11488058246,
            "text": "求关注。",
            "source": "<a href="http://weibo.com" rel="nofollow">新浪微博</a>",
            "favorited": false,
            "truncated": false,
            "in_reply_to_status_id": "",
            "in_reply_to_user_id": "",
            "in_reply_to_screen_name": "",
            "geo": null,
            "mid": "5612814510546515491",
            "reposts_count": 8,
            "comments_count": 9,
            "annotations": [],
            "user": {
                "id": 1404376560,
                "screen_name": "zaku",
                "name": "zaku",
                "province": "11",
                "city": "5",
                "location": "北京 朝阳区",
                "description": "人生五十年，乃如梦如幻；有生斯有死，壮士复何憾。",
                "url": "http://blog.sina.com.cn/zaku",
                "profile_image_url": "http://tp1.sinaimg.cn/1404376560/50/0/1",
                "domain": "zaku",
                "gender": "m",
                "followers_count": 1204,
                "friends_count": 447,
                "statuses_count": 2908,
                "favourites_count": 0,
                "created_at": "Fri Aug 28 00:00:00 +0800 2009",
                "following": false,
                "allow_all_act_msg": false,
                "remark": "",
                "geo_enabled": true,
                "verified": false,
                "allow_all_comment": true,
                "avatar_large": "http://tp1.sinaimg.cn/1404376560/180/0/1",
                "verified_reason": "",
                "follow_me": false,
                "online_status": 0,
```

```
                    "bi_followers_count": 215
                }
            },
            ...
        ],
        "ad": [
            {
                "id": 3366614911586452,
                "mark": "AB21321XDFJJK"
            },
            ...
        ],
        "previous_cursor": 0,
        "next_cursor": 11488013766,
        "total_number": 81655
    }
```


3.3.3 数据存储策略

由于实验所需的人工标记数据量并不是海量大数据，所以不使用关系型数据库而只是用文件存储就足以满足工程实验所需的数据需求。

当我们看到了 API 中的数据结构时，我们就能根据 API 返回的 JSON 数据，来进行存储结构的设计。

对于每个从JSON中截取出来的Status，根据实验的文本分类的需求就只采集其中相关需要的数据进行存储：Status的模型成员如下：

```
public string CreatedAt { get; internal set; }
public string ID { get; internal set; }
public string Text { get; internal set; }

public string Source { get; internal set; }
public string UserID { get; internal set; }
public bool Truncated { get; internal set; }

public string ThumbnailPictureUrl { get; internal set; }
public string MiddleSizePictureUrl { get; internal set; }
public string OriginalPictureUrl { get; internal set; }
```

```
public int RepostsCount { get; internal set; }
public int CommentsCount { get; internal set; }
public string RetweetedStatusID { get; internal set; }
```

同时，考虑到后续进一步扩展比如发布微博与发布广告之间用户特点的这一类功能试验的可能性，程序里也定义了JSON返回数据中User的存储模型，模型成员如下：

```
public string ID { get; internal set; }
public string Name { get; internal set; }
public string Province { get; internal set; }
public string City { get; internal set; }
public string Location { get; internal set; }
public string Description { get; internal set; }
public string Gender { get; internal set; }

public int FollowersCount { get; internal set; }
public int FriendsCount { get; internal set; }
public int StatusesCount { get; internal set; }

public bool Verified { get; internal set; }
public string VerifiedType { get; internal set; }
public string Lang { get; internal set; }
```

这些信息的容量已经足够完成论文中的实验要求所需。

3.4 数据标记工具

由于不使用关系型数据库来存储数据，所以使用文本编辑器对于存储的序列化文档编辑是一件非常危险的操作，比如不经意的随机误操作会导致数据读取的操作，这对于繁重标记的工作是一件毁灭性的打击，所以针对这种情况，需要一个短小精悍的工具来

解决这个问题，让数据充分的封装和严格控制数据输入输出的过程安全保障。

3.4.1 序列化模块

1）将类标记为序列化

为了将已经抓下来的数据可以以文件的形式存储到硬盘上，我们可以将类添加为可以序列化的特性标记[Serializable()]

同时会有一个序列化模块负责将相应的对象序列化的工作。

2）将标记过的类存入文件当中的模块

WriteStream：打开一个写文件流，向文件中写入数据。

WriteObject：向一个文件流中写入一个序列化的类数据

ReadStream：打开一个读文件流，从文件中读取数据。

ReadObject：从一个文件流中读取一个序列化的类数据

CloseStream：关闭一个文件流，解除文件占用。

3.4.2 操作界面设计和使用

为了更好更快的使用自动化方法标记实验用的训练数据集，这里做一个小软件来对已经抓去好的实验数据进行标记。

1）标记工具的界面

菜单栏：显示可以使用的功能

源文件路径显示窗口：显示正在读取的文件路径

微博信息显示栏窗口：显示当前一个待标记的微博的文本信息

已经标记的记录窗口：显示已经完成的标记的数量

标记按钮：点击按钮即可对当前微博进行标记并将标记结果记录在内存当中

同时为了更加的便捷，点击键盘按钮即可对当前的微博进行标记。

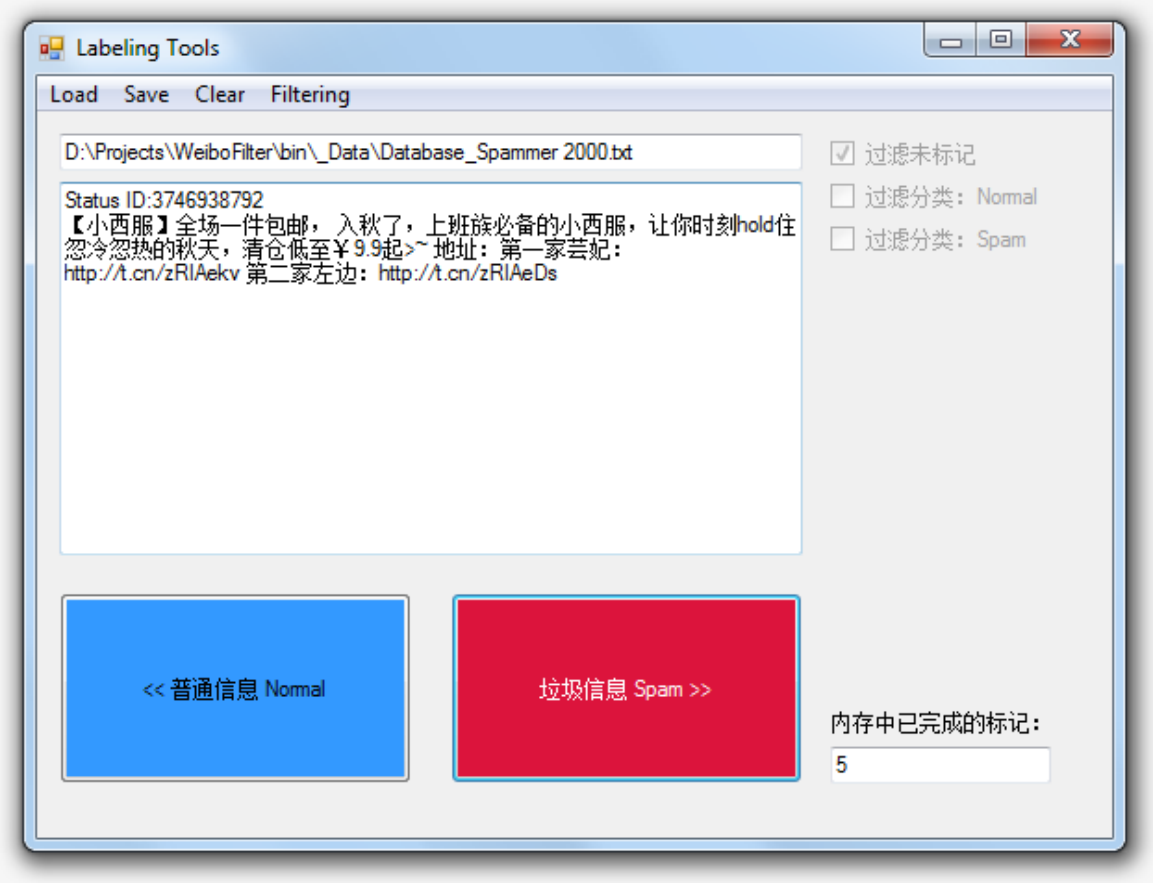

图 6　标记工具的运行界面

标记工具的操作使用

A .首先读取已经抓取的文件

B. 完成所有的微博数据的标记

C. 将已经标记好的数据存入文件中

D. 如果还需要标记另一个文件，则将先清空内存再载入新的文件。

3）标记的存储模型

由于标记所需的完整信息并不多，又因为实验主要做的文本分类，为了节约空间，这里只保存四个字段：被标记的文本的分类类别、被标记的文本、被标记的文本来自的微博 ID、被标记的文本的作者的用户 ID，有了这些数据，就可以获取一个标记的完整信息了。

```
public string Category { get; internal set; }
public string Text { get; internal set; }
public string StatusID { get; internal set; }
public string UserID { get; internal set; }
```

3.5 文本过滤实验平台工具

与传统的文本分类不一样，诸如广告这一类的垃圾信息是不会一成不变的等着去被删的，在这个利益博弈的过程中，垃圾信息会变换各种各样的方式去突破这套系统，所以单一的机器学习方法的过滤策略是不会一直有效的，所以产生了一个需求就是我们需要一个更加灵活可变的 Framework，将各个机器学习的方法融合起来，各取所长，面对变化多端的情况下可以以最快速度灵活地调整策略，以达到可以应对变化多端的文本环境的功能要求，下一章将重点讲述这方面的设计与实现。

3.6 本章小结

本章前半部分主要介绍了整个项目解决方案的设计以及整个工程的工作流程并简单阐述了项目的面临的环境需求以及项目自身实施的可行性，同时也阐明了项目的设计测试方案进行了描述，归纳出本课题需要重点解决的问题。后半部分重点介绍了实验数据的准备的过程，包括如何使用申请使用新浪微博 API 的权限，以及如何使用合适的 API 来获取想要的数据的策略以及最终如何存储这些数据的技术方案。当数据抓取结束之后，则需要对其进行标记，本章也简单介绍了一个数据标记的小软件，这个小软件简化了人工审阅复杂列表来判断标记的工作，使标记这种简单重复的工作变得更加简便。

第四章 系统详细设计与实现

4.1 系统整体架构

整个系统架构的 UML 图，如下图所示：

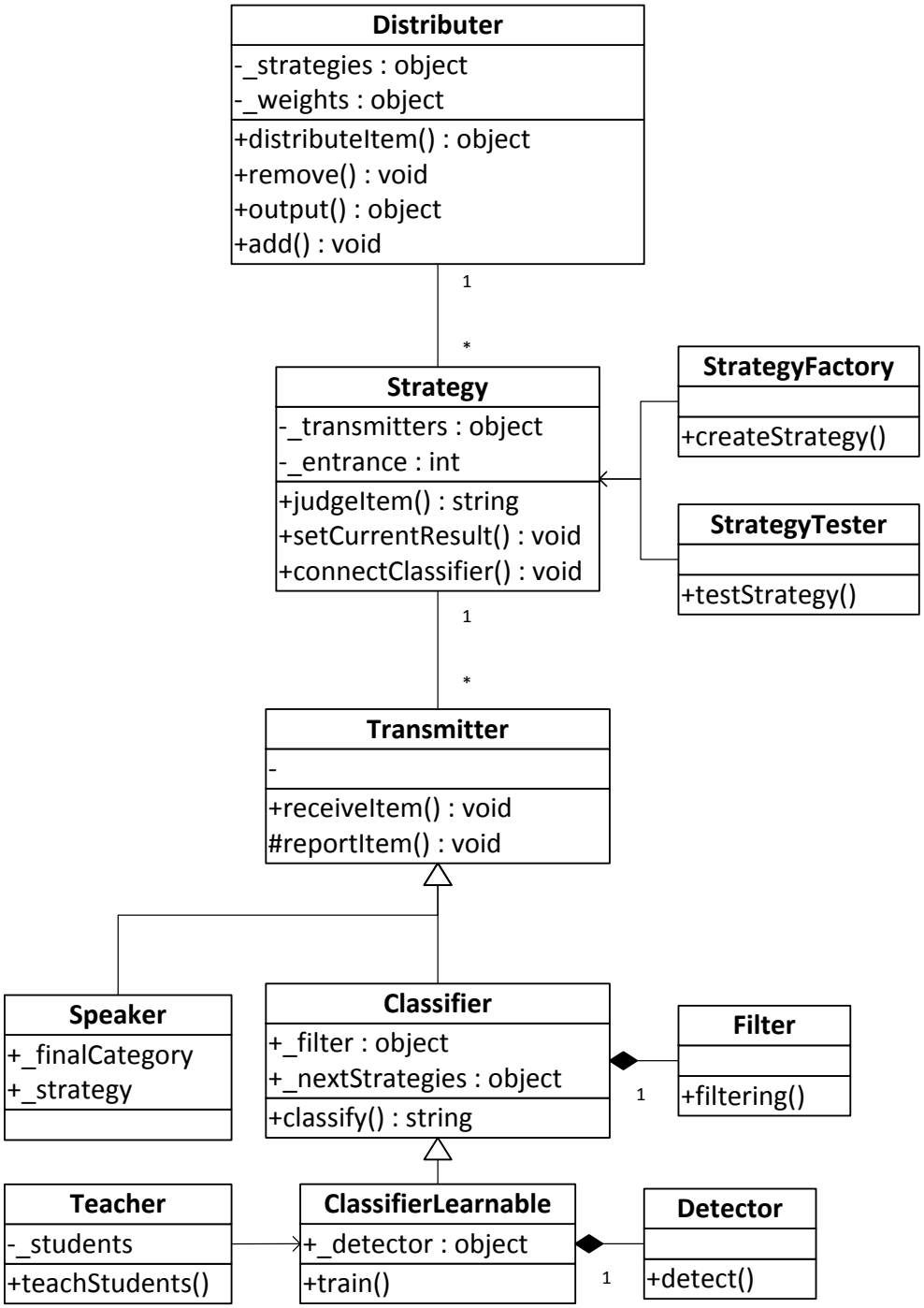

图 7 整个系统的 UML 图

4.1.1 每个模块的功能介绍

Distributer：接受一个未被分类的信息，并把它分发给各个分类策略，在每次判断一个结果的时候，会根据每个策略的权重给出多策略分类的一系列的数值。

Strategy：记录固定设计的分类策略，可以包含多个也可以只包含一个子分类器，基于决策树，多个分类器会构成一个树状的决策结构。每个 Strategy 包含一个数值，数值指向分类器的入口。

StrategyFactory：产生可以使用的各种各样的组合的分类器工厂

StrategyTester：对组合分类器测试精度结果的工具模块，为之后进一步自动检测精度调整单个策略权重而设计的测试模块。

Transmitter：传递器，将接收到的文本处理并传给下一个传递器是构成树节点的最基础模块，属于抽象类，不能被实例化。

Speaker：继承了传递器的抽象类，作用相当于整个策略对于某个分类结果的的发言人的角色。在分类过程中不起分类作用而是作为分类工作结束之后将最终分类的结果报告回所在的 Strategy，由 Strategy 处理相关的结果。

Classifer：继承了传递器的抽象类，作用相当于整个树状结构中的其中一个工作员，进行每一步的分类工作。Classifier 本身无法被实例化，需要定义 classify 方法的详细分类规则方能使 Classifier 的实例作为一个工作单元在 Strategy 中发挥作用。

Filter：负责给分类器过滤出可以辨认的信息，相当于在复杂的环境下给分类器带上一副工作用的特种面具或眼镜进行分类工作，对于分类器来说 Filter 的存在可有可无，这个是为了让框架在面临更加复杂的环境时依然保持足够的灵活性的设计存在。

ClassiferLearnable：继承了分类器的抽象类，具备学习能力的分类器， 可以对这样的分类器实例进行训练等工作，定义 ClassifierLearnable 接口中的 classify 方法可以实现具备机器学习能力的分类器，而 train 方法决定了一个分类器的 model 怎样被训练积累数据的策略。

Detector：负责给分类器解析出文本中可以被用来分类的特征值，同样是为了应付复杂的环境减少代码重复工作的设计， Detector 所解析出的特征就是分类器最终于用判断的依据。

Teacher：可学习分类器的助手，负责训练集处理等辅助工作，可以对自己所管理的具备学习能力的分类器集体进行训练的模块。

4.1.2 灵活的模型框架应用示例

系统的整体架构设计抽象出了机器学习算法的最基本特征，使得整个架构在应对新的策略场景的时候，可以灵活的实现各种各样的策略，当面对更加复杂的环境的时候，也可以使用架构结合多种分类器的算法优劣特点进行集群智慧的文本决策分类。

下图描述了一个可以运行的架构的整体示例，整个架构设计灵活，可以使用单策略单分类器进行分类，同时也可以进行多策略多分类器进行自动学习决策，同时也可以支持非学习型和拥有机器学习能力的分类器共同工作：

下图为框架应用的举例之一：

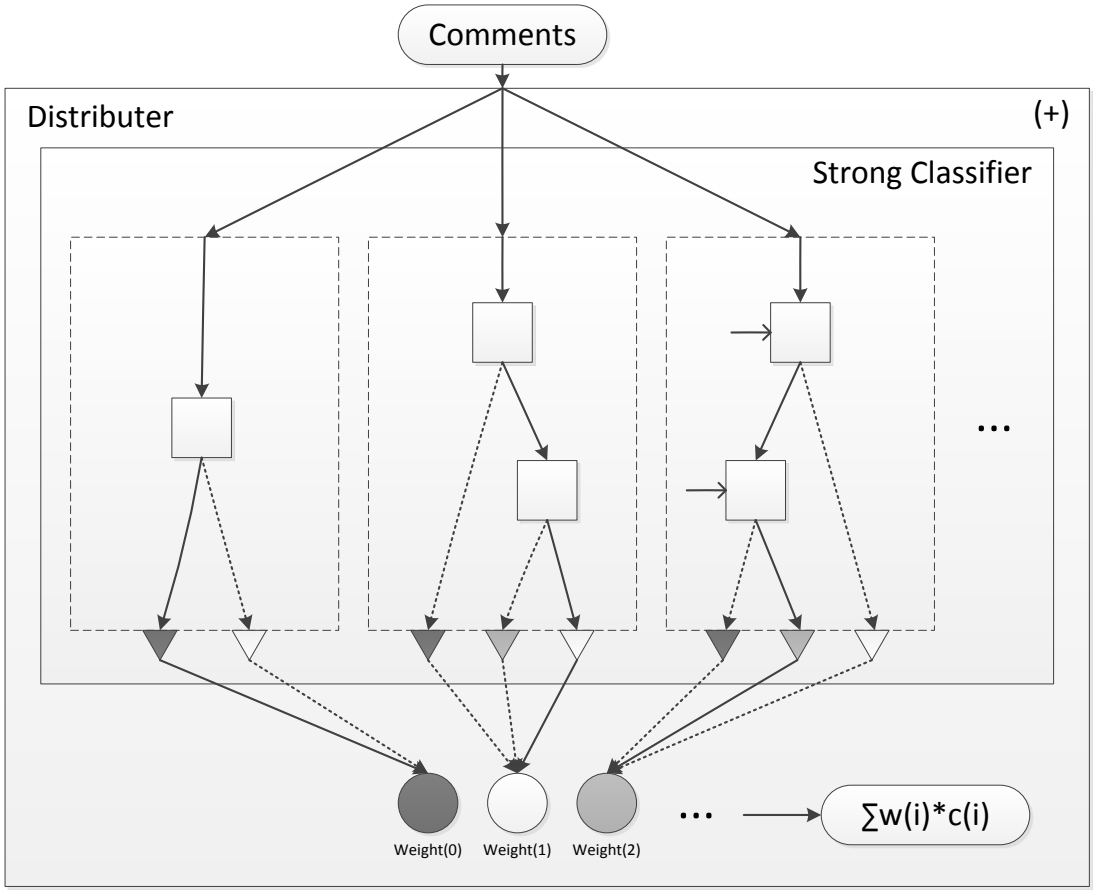

图 8　框架的应用示意图

4.2 灵活的系统设计应用于 Random Forest

4.2.1 Random Forest 简介

在机器学习中，随机森林是一个包含多个决策树的分类器， 并且其输出的类别是由

个别树输出的类别的众数而定。 Leo Breiman 和 Adele Cutler 发展出推论出随机森林的算法。 而 "Random Forests" 是他们的商标。 这个术语是 1995 年由贝尔实验室的 Tin Kam Ho 所提出的随机决策森林（random decision forests）而来的。这个方法则是结合 Breimans 的 "Bootstrap aggregating" 想法和 Ho 的"random subspace method"" 以建造决策树的集合。

随机森林可以很好的规避训练集分布不均衡给算法精确度带来的影响，同时也可以减少训练集当中噪音数据带来的准确性降低的影响。通过本论文建立的软件架构，可以迅速的迭代生成新的 Random Forest 方法为基础的分类器策略。

4.2.2 Random Forest 学习算法

根据下列算法而建造每棵树：

1. 用 N 来表示训练样例的个数，M 表示变量的数目。

2. Random Forest 需要一个固定的数值一个数 m ，m 用来决定当在一个节点上做决策时，会使用到多少个变量。（m < M）

3. 从 N 个训练样例中以可重复取样的方式，取样 N 次，形成一组训练集（本质上是 bootstrap 取样）。并使用这棵树来对剩余预测其类别，并评估其误差。

4. 对于每一个节点，随机选择 m 个基于此点上的变量。根据这 m 个变量，计算其最佳的分割方式。

5. 每棵树都会完整的成长而不会对其进行剪枝（Pruning）（这有可能在建完一棵正常树状分类器后会被采用）。

4.2.3 Random Forest 优点

1.在资料种类非常多的情况下，它可以产生高准确度的分类器。

2.Random Forest 可以处理具有大量规模输入的变量。

3.Random Forest 可以在预测类别的同时评估变量在做决策时的重要性。

4.在建造森林时，它可以在内部对于一般化后的误差产生不偏差的估计。

5.Random Forest 包含一个好方法可以估计弥补丢失的训练样例，而且在资料大部分缺失的情况下，分类其本身仍可以维持一定的准确度。

6.Random Forest 提供了一个实验方法，可以去侦测变量之间的相互影响。

7.对于不平衡的分类训练集来说，它可以平衡误差。

8.Random Forest 会计算各个样例数据中的亲近度，对于数据挖掘、侦测偏离者

（outlier）和处理图像化的数据也非常有用。

9. 使用上述所述的特性，Random Forest 同时也可以用于训练准备未标记的数据集上，这类数据通常是使用非监督式聚类，同时也可侦测偏离和观看资料。

10. 训练的过程所消耗的时间代价非常小。

4.2.4 Random Forest 缺点

1. 随机森林已经被证明在某些噪音较大的分类或回归问题上会过拟

2. 对于有不同级别的属性的数据，级别划分较多的属性会对随机森林产生更大的影响，所以随机森林在这种数据上产出的属性权值是不可信的。

4.2.5 Random Forest 与过滤器架构的结合使用

每个 Distributer 总管多个 Strategy，每个 Strategy 包含一个决策树，而对于多个决策树而言，每一个决策树中的训练集是随机选取的，这样就符合了随机森林的算法要求，进行每一次分类的时候，随机森林就会比简单的分类其产生更高的准确度。

下图为使用框架搭建的随机森林的示意图：

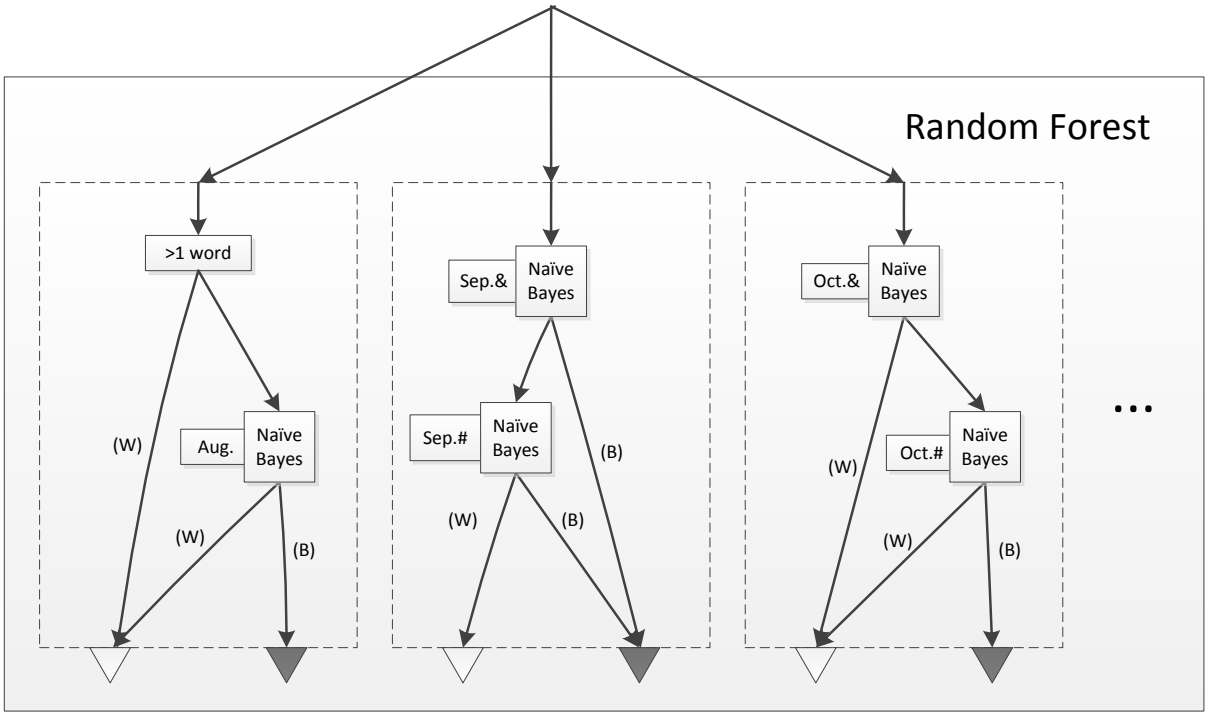

图 9 框架的 Random Forest 应用示意图

4.3 灵活的系统设计应用于 Ada Boost

4.3.1 Ada Boost 简介

在机器学习中，Ada boost 是一种迭代算法，其核心思想是针对同一个训练集训练不同的多个分类器(这里我们将每个训练出来的分类器称之为弱分类器)，然后把这些弱分类器分类的结果集合起来，构成一个更强的最终分类器(我这里我们组合形成的分类器称作强分类器)。其算法本身是通过改变训练集中使用的数据分布来实现的，它根据每次训练集之中每个样本的分类是否产生了正确的结果，以及上层分类器的总体分类的准确率，来确定每个样本数据的权值。将修改过权值的新训练集送给下层分类器进行训练，最后将每次训练得到的分类器最后融合起来，作为最终可以使用的决策分类器。使用 Ada boost 构建的分类器可以自动无视掉一些的训练集中的数据特征，并将核心的分类点放在最关键的训练数据上面。

AdaBoost 方法针对具备一定深度学习能力的分类器效果表现的十分的优异，利用已经实现的分类器模块配合整个架构灵活可调整的特性，可以很快的迅速迭代生成一套具备 AdaBoost 特性的高级文本分类器。

4.3.2 Ada Boost 主要解决的问题

目前，对 Ada Boost 算法的研究以及应用大多集中于分类问题，同时近年也出现了一些在回归问题上的应用。单从其应用的角度来讲 Ada Boost 系列的算法研究主要解决下面几个问题：两类分类问题、多类单标签的分类问题、多类多标签的分类问题、大类单标签的分类问题、回归分析的问题。解决问题的过程往往需要全部完整的训练样本。

4.3.3 Ada Boost 算法过程

算法的整个流程是一个弱分类器迭代积累强化的算法提升过程，这个过程通过一次次迭代不断的训练生成更多的弱分类器组合，可以提高整个分类器对数据的分类能力，整个过程如下图所示：

1. 先通过对 N 个训练样本的整体学习训练得到弱分类器，那弱分类器进行测验。

2. 将第一部分错的样本和其他的训练数据一起构成一个新的训练样本，通过对这个样本的学习再次得到一个弱分类器。

3. 将 1 和 2 都分错了的样本加上其他的新样本构成另一个新的训练集，通过对这个样本的学习得到第三个弱分类器。

4. 循环重复至新分类其精度提高小于某个阈值为止。

5. 组合所有的弱分类器构成一个提升的强分类器，每个弱分类器都需要某个权值来控制，即某个数据被分为哪一类要通过数个弱分类器的权值累加的多数表决。

4.3.4 Ada Boost 与过滤器架构的结合使用

从 Ada Boost 的角度来讲，Distributer 就是这个分类器中的强分类器，而每个训练的弱分类器就是一个 Strategy，每次迭代训练之后产生的新分类器可以被当做一个新的 Strategy 被加入 Distributer。

下图为使用框架搭建的 Ada Boost 的示意图：

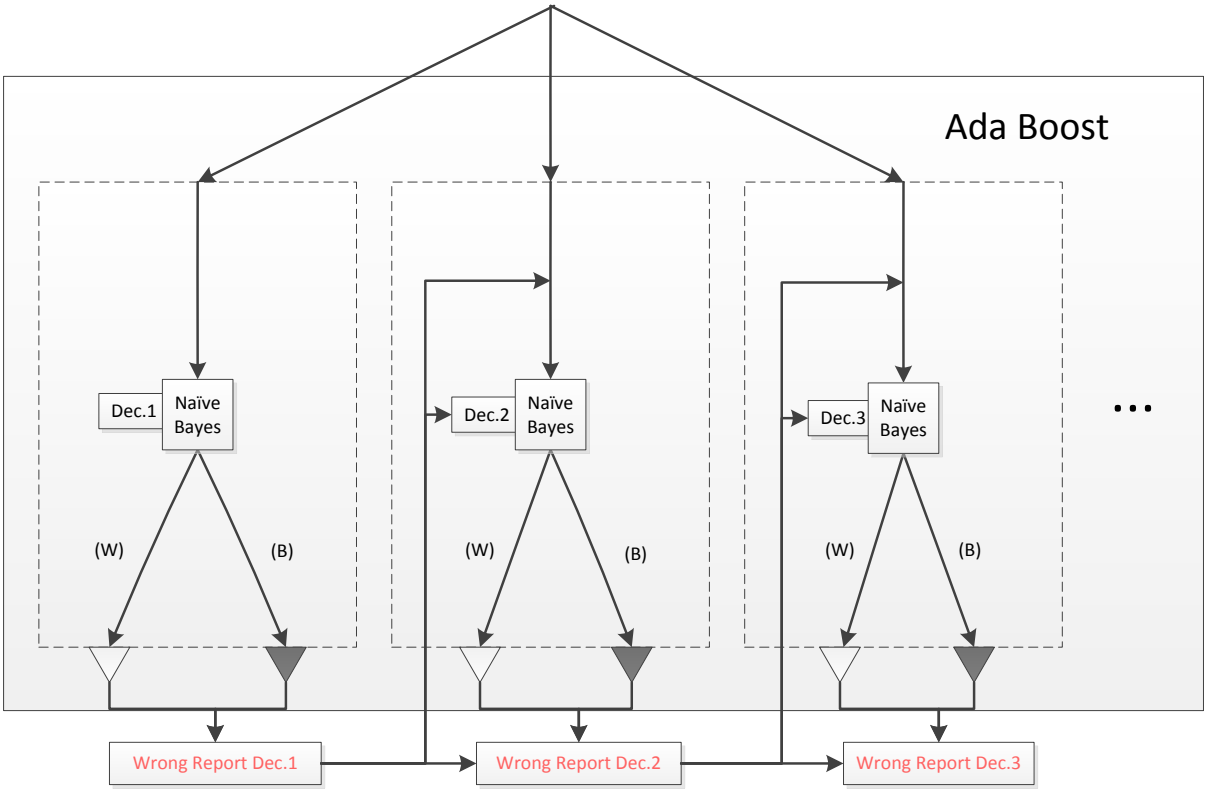

图 10 框架的 Ada Boost 应用示意图

4.4 子分类器-简单分类器的设计与实现

在本节中，将对一些简单的基于固定规则的分类器进行介绍，这一类简单的分类器虽然不及机器学习能力的分类器强大，却可以对内容进行初分类，对进一步提高下一步

机器学习的精准度有着很大的帮助。

4.4.1 抽象的结构

Classifier 的抽象结构如下图所示：

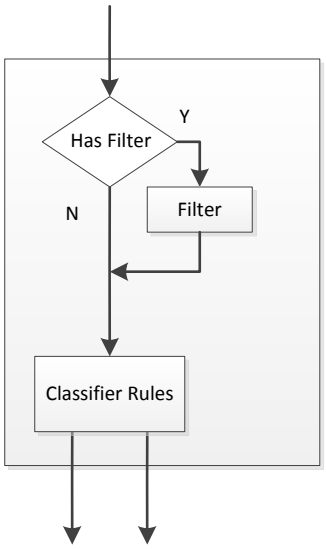

图 11 简单过滤器的结构

简单的 Classifier 类只完成一个工作，就是接受到传来的 item，然后判断自己是否已经有 filter,如果有就把 item 用 filter 过滤之后交给实现他 classsify 方法的实例，没有的话就直接交给 classify 方法进行判断，然后把判断之后的结果以及原 item 一起依据惯导名称决定交付给下一个 Classifier 继续分类辨别或者交给 Speak 向 Strategy 报告最终的裁决结果。

4.4.2 实例：微博中是否包含网址链接的规则预处理分类器

在新浪微博的许多的垃圾广告信息中，你会看到这样一种特征：广告信息中的文本有一大部分会包含短连接的信息，这种特征无需机器学习方法就能判断是否有很大可能是一个广告信息，所以在第一部粗分类的过程，可以将其归为需要高度警惕的信息，在接下来的机器学习方法过滤的过程中，可以使用高敏感度的分类器对其进行分辨，这样一来在进行过粗分类的基础之上我们可以逐步优化分类的过程来一步步的提高分类的精准度。

下两个图则是很经典的广告信息，包含了多条短连接：

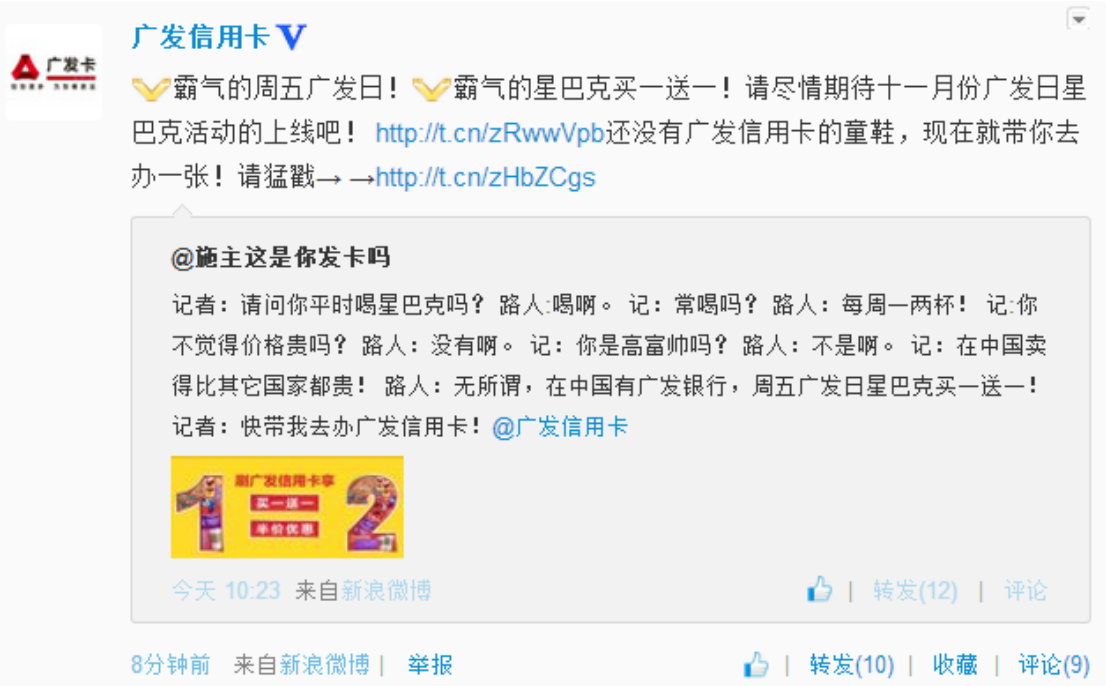

图 12　包含两条短链接的转发广告微博

图 13　包含三条短链接的原创广告微博

当然单纯凭借这样的特征并不能 100%完全断定这是一个垃圾信息，不然很多时候包含一个链接的信息也会很容易被“冤枉”成一个垃圾信息而被过滤掉，如果一个系统错判了太多好的信息，即使它对于垃圾信息过滤的精准度再高，也没有实际的应用意义与价值。

比如说下图就是个很常见的微博，它的文本信息中包含了短链接的信息，但是实际上他并不是一个垃圾信息：

图 14 包含短链接但不是垃圾信息的微博

4.4.3 实例：对于表情符号和好友@以及#标签过滤的分类器

通过对新浪微博许多信息的分析中我们可以得知，对于好友的@操作和表情转义符号的存在也会是一个非常重要的特征，对于这样的特征，依赖于架构的巨大的灵活性可以迅速实现一个特征的过滤，因为我们可以发现，表情这样的特征与是否是垃圾信息的关系并不大，所以我们设计的过滤器除了可以进行分类分发之外，还可以对于无关信息的干扰进行过滤净化。

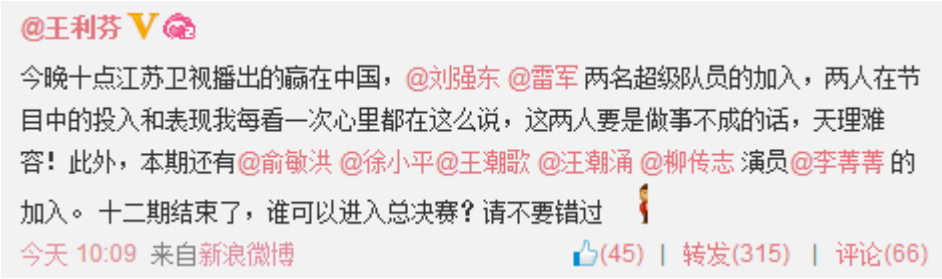

图 15 有@操作也有表情的普通微博

图 16 有#标签也有表情的广告微博

4.4.4 与垃圾信息博弈进化中的快速反应

如前面分析所得，对于垃圾信息的过滤与对文本的分类有着本质的区别：文本分类不会对文本本身的发布者的利益造成任何影响，而微博的垃圾广告信息过滤则会对广告发布者的行为产生深远的影响：因为在当今的眼球经济体系下，自己发布信息的目的是让更多的人看到自己发布的广告宣传，而垃圾过滤器则会阻止自己进一步达到自己的目的，所以发布者会尝试各种各样的新的方法来突破过滤规则本身，这是为了提高进一步的过滤效果就需要快速反应。

在下面的例子当中，由于链接这个特征会被用来判断是否为垃圾信息，所以广告的发布商们就会回避在微博中包含短链接的方法来发布信息，而在本条微博的评论中发布短连接信息。

比如下几张图中：

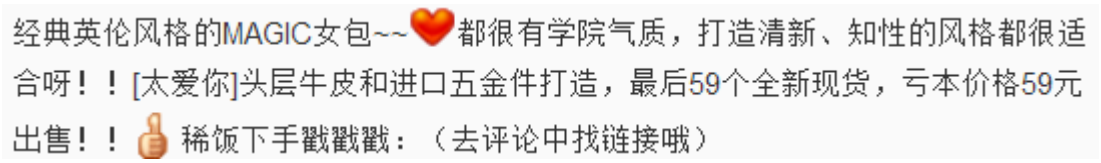

图 17 包含“评论中找链接”的微博

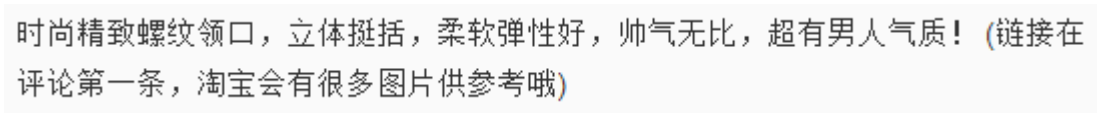

图 18 包含“链接在评论”的微博一

秋季最新爆款，立体花的设计还是那么大受大美女和小美女的热爱与追捧！那种会呼吸的美，那种清新又不张扬的美！(链接在评论第一条，淘宝会有很多图片供参考哦)

图 19 包含“链接在评论”的微博二

所以这时候就可以快速发现这些广告信息新的共同点，我们可以把最新的 feature 定义如下：

当文本中同时出现了“链接”和“评论”两个词，并且这两个词之间的中文字符少于等于五个时，我们认为这条信息是垃圾信息的嫌疑是非常大的。

Strategy 的对抗结构就可以从旧的策略中进化到新的结构，如下图所示：

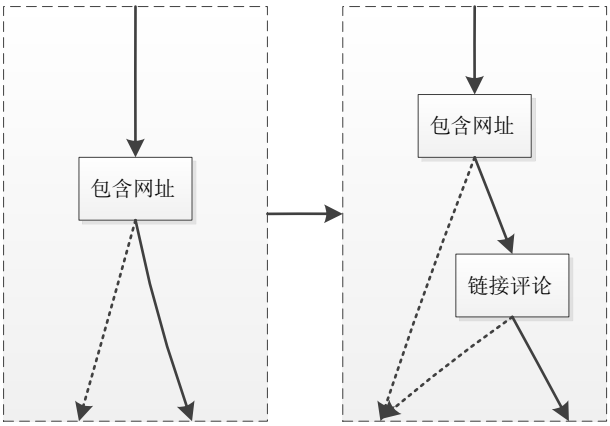

图 20 简单过滤器的树形叠加

这样以来，即使在后续变化的环境当中，整个分类策略依然可以保证足够高的分类精确度。

4.5 机器学习算法调研

通过上面的需求分析，我们了解到了系统的性能需求中，对于算法的时间复杂度的要求比较高，所以我们需要对算法进行调研，然后选择适应系统需求的算法。

4.5.1 机器学习概述

机器学习是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构使之不断改善自身的性能。

4.5.2 K 近邻算法的应用分析

K 最近邻分类学习算法，目前已经是是一个理论上比较成熟的方法之一，同时也是技术上实现最简单的机器学习算法之一。

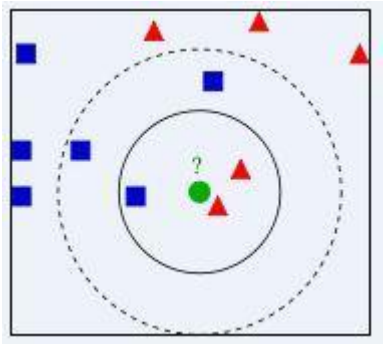

图 21　K 近邻算法的示意图

K 近邻算法的核心思想基于一种“近朱者赤，近墨者黑”的假设：如果一个样本在特征空间中的 K 个距离最相近的样本中的最多数属于某一个类别，则该样本本身就也属于这个类别，K 的取值需要事先给定。

K 近邻的优势在于算法实现特别简单易懂，但是有两个缺点非常的明显：其中一个是对于不平衡的训练集过于敏感，因为 K 近邻算法只计算距离最近的邻居样本，某一类的样本数量很大，那么或者这类样本并不接近目标样本，或者这一类样本很靠近目标样本但是样本由于过于稀疏导致还是被归为了其他的类别。而另一个明显的缺点则在于性能方面的瓶颈过于明显，因为每一个待分类的样本都要计算寻找它与到全体已知样本的距离之后，才能有足够的信息求得它的 K 个最近邻点，这样消耗的计算资源过于庞大，如果没有对算法优化和训练集进行剪辑之后就投入使用的情况下满足性能需求的代价会相当的大。

4.5.3 决策树算法的应用分析

决策树分类算法是另一种具备机器学习能力的算法，它通过一种典型的逼近离散函数值的方法对数据进行处理，利用相应的归纳算法生成具备一定可读能力的同时具备一定规则的可以用来进行决策分析的树状结构图，然后使用新生成决策树对未来出现的样本进行分析预测。从本质上来讲，决策树算法的过程实际上是通过一系列规则建立来对数据进行分类的过程。

决策树构造一般是被分成两步进行的：第一步，决策树的生成阶段，将训练集进行

计算学习并且生成决策树的过程。对于训练集的数据要求则是不能有过大的数据不平衡，同时尽可能的覆盖各个属性与各个节点的分析用数据集，这样生成的决策树可以有更好的预测覆盖率。第二步，决策树的剪技阶段，对于决策树的剪枝过程来说，需要完成的最主要的任务是是对上一阶段生成的决策树进行进一步检验校正以及修正的过程，校验的主要关键点在于决策树生成过程中产生的初步规则，在剪枝的过程中还需要将那些影响预衡准确性的分枝剪除。

决策树的算法在面临少量属性决策的时候显得很有效，但是却不适用于自然语言处理的环境下，因为即使是传统的语言环境树的深度已经会变得非常的深到影响训练计算的速度。而新浪微博的特殊环境下，在目前的分词算法尚未发展到可以自动识别新的衍生词的阶段的时候，使用决策树所产生的的效果对于系统正确性的满足程度将会非常的有限。

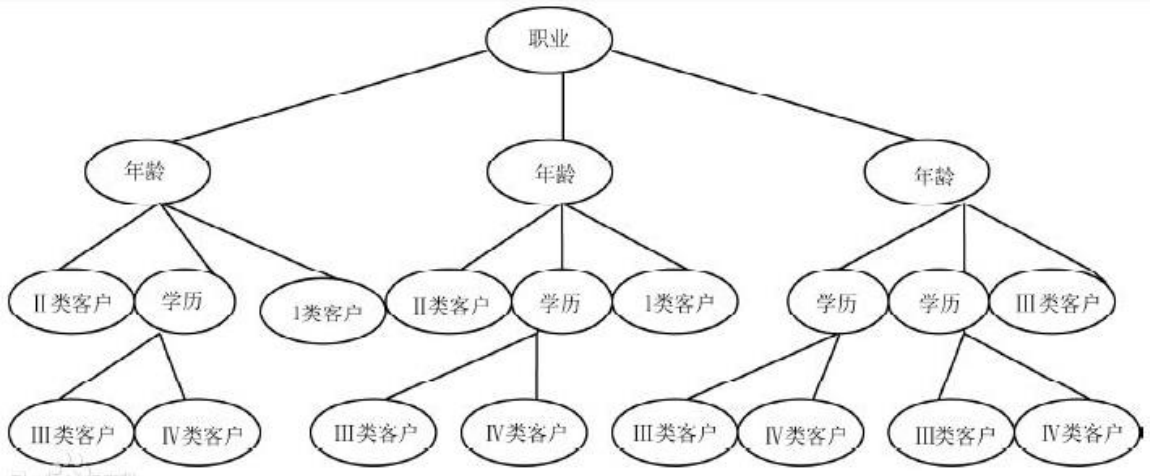

图 22 由算法生成的决策树

4.5.4 SVM 算法的应用分析

SVM 算法(支持向量机)的理论基础主要来自于建立在统计学习理论的 VC 维理论以及结构风险最小原理，SVM 算法的主要思想是根据有限的样本信息对特定训练样本的学习精度和学习能力之间寻求最佳平衡点并且以此为目标进行计算学习积累并且对未来出现的样本进行预测。

SVM 的主要算法过程是通过将每一个样本抽象出来的可计算向量映射到一个更高维的空间里的同时在这个空间里寻找一个具备最大间隔特征的超平面。与此同时在分开数据的超平面的两边各自建立两个互相平行的超平面，基于这两个建立方向合适的分隔超平面使两个与之平行的超平面间的距离最大化。使用这种算法的假设前提为，平行超

平面间的几何距离越大，分类器所产生的的总误差将会越小。

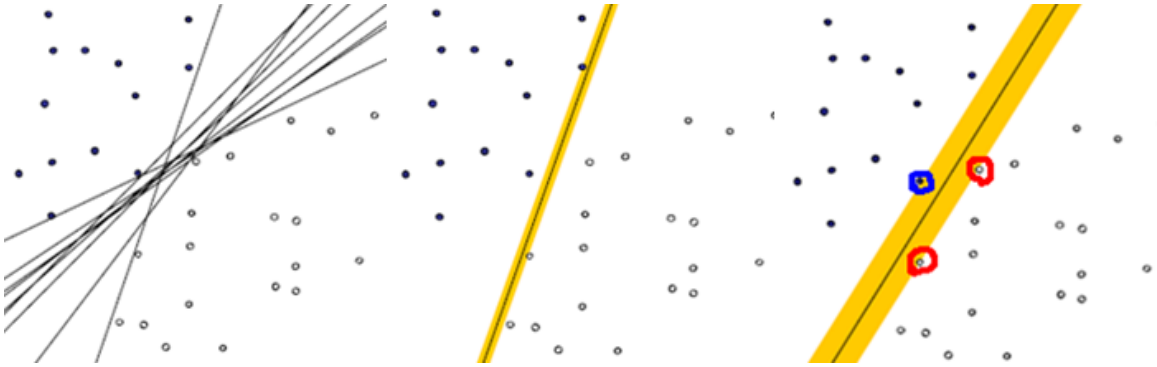

图 23　超平面获取的示意图

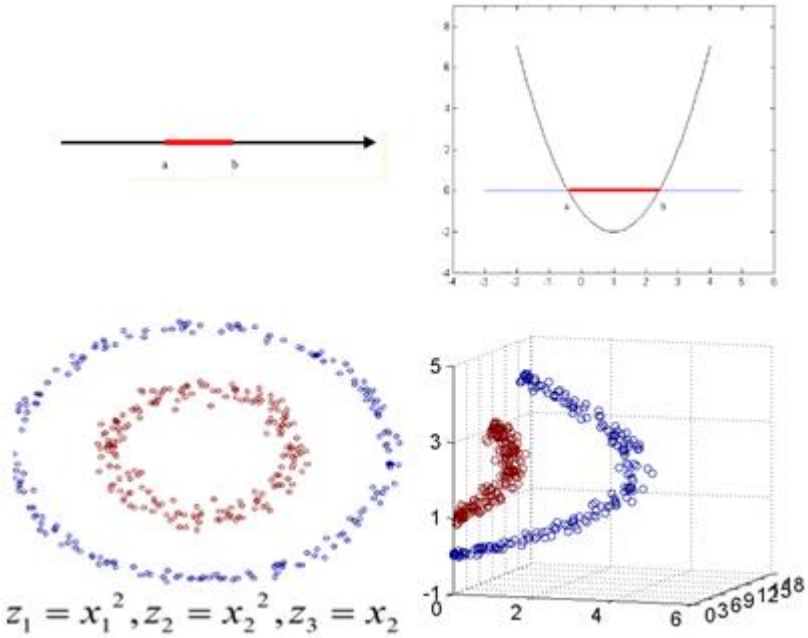

图 24　SVM 通过升维方法进行分类的示意图

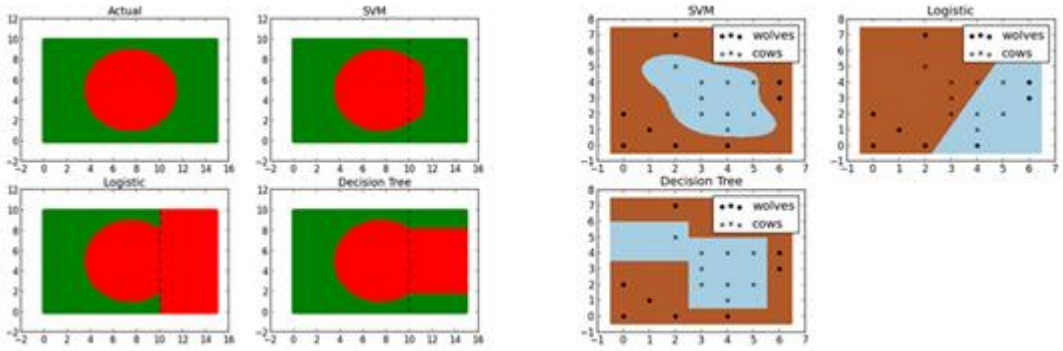

图 25　SVM 进行实际的分类效果的示意图

SVM 对于分类的正确效果相对于其他算法会有相当高的精确度，但是超高的正确度所消耗的代价就是 SVM 需要大量的消耗计算资源会带来速度上的实现瓶颈之外，还有十分复杂的实现难度，而目前可以使用的第三方工具 Libsvm 的速度也不尽人意，所以在投入使用的时候，SVM 需要很慎重的考虑。

4.5.5 朴素贝叶斯算法的应用分析

贝叶斯定理这个在 250 多年前发明的算法，在信息科技的整个领域内有着至高无上的地位。贝叶斯分类算法实际上不是一个单一的算法而是一系列分类算法的总称，这类算法均以贝叶斯定理的假设作为坚实的基础，故统称为贝叶斯分类算法。而众多贝叶斯算法当中，朴素贝叶斯算法是其中应用最为广泛和被熟知的的分类算法。

而在各种各样的算法应用于文本分类算法的场景中时，朴素贝叶斯所展现出来各方面的综合的效果是最佳的，无论是从算法的精准正确程度还是性能速度方面的考究。

4.5.6 针对核心问题选择算法

综合所有上面讲过的算法看来，最佳的解决方案不是选择其中一个算法进行实现而是将多个算法以及多个优化过的训练及结合起来使用，站在巨人的肩膀上进行创新才会产生更好的效果，当各方面资源有限的情况下，会优先使用贝叶斯算法辅以决策树的模型思想进行系统的具体实现，从而更好地满足各方面的需求。

4.6 具备机器学习能力的分类器的架构与实现

4.6.1 抽象的结构

具有机器学习能力的 Classifier 与简单的 Classifier 最普通的区别在于，机器学习是通过统计方法通过对训练数据集进行一定的计算处理后的模型为依据来对将来可能出现的事物进行预测，所以使用升级后的 ClassifierLearnable，要首先对其进行训练或者载入已经经过计算处理过的数据模型，然后再进行预测。

由于分类器做的是中文分类，所以这里需要强调的是在进行训练和判断的过程中，都是需要经过 Detector 将接收到的 Item 拆解成数个 Feature 字符串再处理，所以，Detector 的选择会直接影响算法最终判断的的准确度。

由于实验的时间资源有限，所以一部分算法的实现工作这里则需要依赖外部的工具包来简化实现所需要的工作。比如中文分词所使用的 Pangu 分词，在这里为其适配所写

的胶水代码如下：

```
public static List<string> panguDivide(String s)
{
    Segment segment = new Segment();

    ICollection<WordInfo> words = segment.DoSegment(s);
    List<string> wordsResult = new List<string>();

    foreach (WordInfo wordInfo in words)
    {
        if (wordInfo == null) continue;
        wordsResult.Add(wordInfo.Word);
    }

    return wordsResult;
}
```

4.6.2 朴素贝叶斯分类器的存储实现与流程结构

1）存储模型的结构

对于朴素贝叶斯来讲，Model的结构相对比较简单，只需要维护一个词频列表和各个类别出现的Documents的数量记录就可以实现，以下是为了维护这个Model所必须存储记录的信息容器模型：

```
// <Category>
private List<string> _categoryNames;

// <Category, Count of Items>
private Dictionary<string, long> _itemsCountInCategory;

// <Category, <Feature, Count>>
private Dictionary<string, Dictionary<string, long>> _featuresCountInCategory;
```

但仅仅存储这些信息对于工程性的要求是远远不够的，我们还需要用空间换时间，记录一些无需重复计算的信息以加快每次进行判断的速度：

```
// just for performance
private long _featuresCountTotal;

// just for performance <Category, Count Sum of features in this category>
private Dictionary<string, long> _allFeaturesCountInCategory;

// just for performance <Feature, Count Sum of features In All Category>
private Dictionary<string, long> _featuresCountInAllCategory;
```

2）分类继承模型的结构

NBC（Naïve Bayes Classifier）分类器的继承关系如下：

ClassifierBayes -> ClassifierLearnable -> Classifier -> Tranismmiter

所以整体的工作流程就如下图所示：

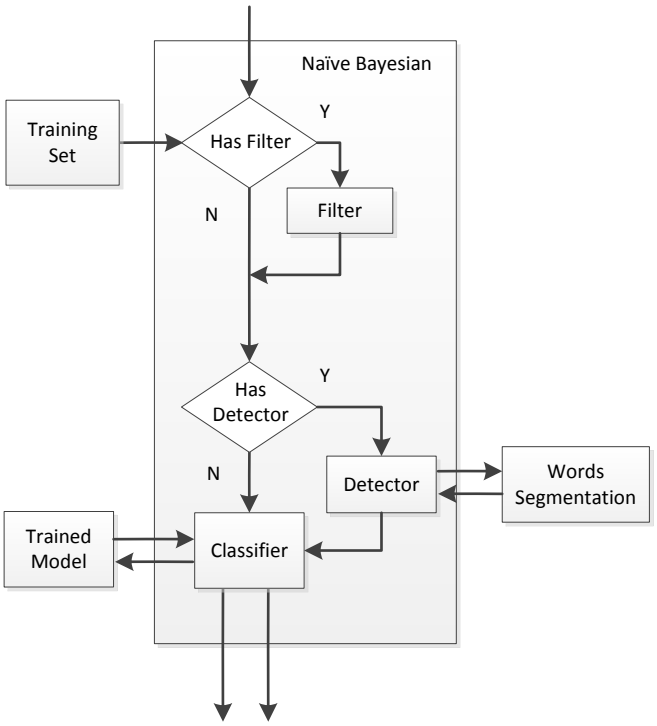

图 26 具备学习能力的分类器的结构示意图

a）在正式将分类器投入使用之前我们要首先定义如何去拆解一个文本，把它变成数个可以执行的特征，这里的 Detector 只需要将文本单纯的打散分词即可，并不需要进一步使用 TF-IDF 这样复杂的算法来对文本进行处理。

b）其次就是需要定义使用的训练集然后要对分类器进行训练，训练会单纯的积累词频已经文本的数量，本项目中所实现的分类器可以在初始化的时候定义三种分类计算方法，每个计算方法所使用的计算方式不同但是使用的 Model 可以是同一个。

c）当 Detector 定义和训练都结束之后，我们就可以把这个分类器的实例放入 Strategy 作为其中的一个分类节点来使用了。

3）多种贝叶斯算法的设计实现

如前面所说，贝叶斯分类器实现了至少两种算法，为了保留各种实验的效果，最后的分类器模块依然保留了没有经过优化的朴素贝叶斯的算法，所以分类器可以在初始化

的时候就指定本身需要使用的算法。

可以指定的算法类型枚举如下面代码所示：

```
enum ClassifierLearnableBayesType{
    Naive,
    Fisher,
    Revised
}
```

4）贝叶斯算法的训练过程

对于贝叶斯算法来讲，无论是朴素贝叶斯还是优化过的贝叶斯，其核心都是对于词频的积累，所以训练的过程也相对简单并且可以支持增量训练和减量训练。不论是从空间复杂度还是时间复杂度都会有很高的表现，具体实现代码如下：

```
public override void train(string comment, string category)
{
    // Add category name list
    if (!_categoryNames.Contains(category)) _categoryNames.Add(category);

    // Add items count
    if (!_itemsCountInCategory.ContainsKey(category))
    {
        _itemsCountInCategory.Add(category, 0);
    }
    _itemsCountInCategory[category]++;

    // Add feature count
    if (!_featuresCountInCategory.ContainsKey(category))
_featuresCountInCategory.Add(category, new Dictionary<string, long>());


    string[] features = _detector.detect(comment);
    foreach (string feature in features)
    {
        // Add feature Count in all Category
        if (!_featuresCountInAllCategory.ContainsKey(feature))
        {
            _featuresCountInAllCategory.Add(feature, 0);
        }
        _featuresCountInAllCategory[feature]++;
```

```
                // Add feature Count in that Category
                if (!_featuresCountInCategory[category].ContainsKey(feature))
                {
                    _featuresCountInCategory[category].Add(feature, 0);
                }
                _featuresCountInCategory[category][feature]++;
            }
        }
```

4.6.3 朴素贝叶斯分类器的理论基础及代码实现

朴素贝叶斯分类器模型是当下应用最广泛的分类模型，朴素贝叶斯分类器基于一个简单的假定：给定目标值时属性之间相互条件独立。

贝叶斯公式为：

$$\Pr(A|B) = \Pr(B|A) * \frac{\Pr(A)}{\Pr(B)}$$ （公式 4.1）

通过公式 4.1 和“朴素”的假定，设 A = Category, B = Document 则会有

$$\Pr(Cat|Doc) = \Pr(Doc|Cat) * \frac{\Pr(Cat)}{\Pr(Doc)}$$ （公式 4.2）

由公式 4.2 来看，我们假设每个 Documents 有一定概率可以被分成若干个 Category，在每个 Document 出现的时候 P（Document）是相同的没有可比较性，所以在这里忽略掉，剩下的就是首先要计算 P（ Category ）的概率，在朴素贝叶斯当中:

$$\mathrm{P}(\text{category}) = \frac{\text{total(documents in category)}}{\sum \text{total(documents)}}$$ （公式 4.3）

通过公式 4.3 我们可以看到，一个大概率可以通过出现在这个 Category 中的 Documents 的数量来计算得出。具体的代码实现如下所示：

```
    private double getCategoryProbByItems(string category)
    {
        if (getItemsTotalAll() > 0)
        {
            return (double)getItemsTotalInCategory(category) / getItemsTotalAll();
        }
        return 0.0;
    }
```

而对于每个文本来讲，特征可以被分成若干不同的词语，而这些词语是独立分布的，在给定的 Category 类文本中第 i 个单词出现的概率可以表示为如下：

$$\mathrm{P}(w|category) = \frac{count(word\ in\ doc\ of\ cat)}{count(all\ doc\ of\ cat)}$$ （公式 4.4）

通过这种处理，我们进一步简化了工作，假设每个词语是在文中是随机分布的，也就是单词不依赖于文本的长度，同时也不依赖于与其他词语出现在文中的位置，或者其他文本内容在文中的位置。代码实现如下：

```
private double getFeatureProbInItemCategory(string category, string feature)
{
    if (getItemsTotalInCategory(category) > 0)
    {
        return (double)getFeatureTotalInCategory(category, feature) /
getItemsTotalInCategory(category);
    }
    return 0.0;
}
```

对于一个给定类别 Category，单词 w_i 的文本 Document，概率表示为

$$P(doc|cat) = \prod wp(w_i \mid cat) = P(w_1 \mid cat) * P(w_2 \mid cat) * \ldots \text{（公式 4.5）}$$

通过公式 4.5 我们会发现一个严重的问题：由于每个单词的概率的边界条件的情况下会出现 0 的情况，所以对于每个词语的概率，我们需要做一个优化，使得在边界情况下，不会因为一个词语概率为零而导致整个算式的结果为 0 进而使得判断失效：

$$P(w|category) = \frac{weight*baseprob+\ total(w)*P(w|category)}{weight+total(w)} \text{（公式 4.6）}$$

$$weight = 1.0\ \ assumedprob = \frac{weight}{total(category)} \text{（公式 4.7）}$$

公式 4.6 和公式 4.7 对公式 4.5 中可能带来的问题进行了优化，代码实现如下：

```
private double getFeatureWeightedProbByNaive(string category, string feature)
{
    double weight = 1.0;
    double allProb = weight / getCategoryNamesCount();

    double basicProb = getFeatureProbInItemCategory(category, feature);

    double totals = getFeatureTotalInAllCategory(feature);

    return (weight * allProb + basicProb * totals) / (weight + totals);
}
```

这样一来整个公式所需要计算的内容都已经准备完成了，下一步我们来探讨怎么利用计算出来的结果来进行分类判断：

假设现在只有两个相互独立的类别，S 和¬S（垃圾信息和普通信息），这里每个元素要么是垃圾信息，要么就是普通信息。所以根据贝叶斯公式我们可以算出：

$$P(D|S) = \prod_i P(w_i|S) \quad 和 \quad P(D|\neg S) = \prod_i P(w_i|\neg S) \quad （公式 4.8）$$

应用上述公式 4.8 的结果，最后的贝叶斯计算方法可以写成：

$$P(S|D) = \frac{P(S)}{P(D)} \prod_i P(w_i|S) \quad 和 \quad P(\neg S|D) = \frac{P(\neg S)}{P(D)} \prod_i P(w_i|\neg S) \quad （公式 4.9）$$

将公式 4.9 两者相除得到公式 4.10：

$$\frac{P(S|D)}{P(\neg S|D)} = \frac{P(S)\prod_i P(wi|S)}{P(\neg S)\prod_i P(wi|\neg S)} \quad （公式 4.10）$$

这样概率比 P(S|D) / P(¬S|D)可以表达为似然比。实际的概率 p(S|D)可以很容易通过 log (P(S|D)/P(¬S|D))计算出来，基于 P(S|D)+P(¬S|D)=1。

最后文本可以分类，当 P(S|D) > P(¬S|D)或者 ln (P(S|D)/P(¬S|D)) > 0 时判定为垃圾信息，否则为普通信息。

假设我们面临的文本环境需要有多个分类，则在计算出多个分类概率的情况下，我们认为最终结果为概率值最大的那个，具体计算方法见公式 4.11。

$$Result = \max\{ P(cat_1|doc), P(cat_2|doc), \ldots\} \quad （公式 4.11）$$

最终的结果判断的代码实现如下所示：

```
private double getProbNaive(string category, string[] features)
{
    // get category Probability
    double catProb = getCategoryProbByItems(category);

    // count document Probability
    double docProb = 1.0;

    foreach (string feature in features)
    {
        docProb *= getFeatureWeightedProbByNaive(category, feature);
    }

    return catProb * docProb;
    //return docProb;
}
```

4.6.4 贝叶斯分类器的 Fisher 优化及代码实现

在完成朴素贝叶斯分类之后的测试里，我们会发现分类器的精准度会受到训练集容量比例很大的影响，假如用于训练的垃圾信息比重过高的情况下，误判也会随着训练集中比例的失衡而导致严重的增加，而过多的误判对于算法本身来讲是不具备实际应用价

值的。

所以根据这个特质，在贝叶斯分类器当中我们会有一种称之为 Fisher Method 的优化，也就是 Fisher 准侧分类其思想的应用：

从上面的公式我们可以看出，由于相互独立的词频概率是被一个个主次相乘起来而获得的最终的结果，所以，为了把这些概率受到训练集比例的影响去除掉，我们需要将公式增加新的计算：

朴素贝叶斯优化 feature weight 的计算方法见公式 4.12 和公式 4.13

$$totals(feature) = \sum count(feature\ in\ all\ category) \quad \text{（公式 4.12）}$$

$$prob(feature|cat) = \frac{count(feature\ in\ doc\ of\ cat)}{count(all\ doc\ of\ cat)} \quad \text{（公式 4.13）}$$

对于新的公式，我们需要进一步的进行优化计算，优化计算的思路则是将所有的概率加起来再进行一轮权重迭代计算：

$$cprob(feature|cat) = \frac{prob(feature|category)}{\sum prob(feature\ in\ all\ category)} \quad \text{（公式 4.14）}$$

$$wprob(w|cat) = \frac{weight*baseprob+total(w)*cprob(feature|cat)}{weight+total(w)} \quad \text{（公式 4.15）}$$

通过上述公式 4.14 到公式 4.15 的优化计算，我们就能得出一系列的数据，保证在训练及比例不同的情况下，依然可以保证较高级别的精准度。

cprob 和 wprob 的计算的代码实现如下所示：

```
private double getFeatureWeightedProbByFisher(string category, string feature)
{
    double weight = 1.0;
    double allProb = 1.0 / getCategoryNamesCount();

    double basicProb = getFeatureProbByFisher(category, feature);

    double totals = getFeatureTotalInAllCategory(feature);

    return (weight * allProb + basicProb * totals) / (weight + totals);
}

private double getFeatureProbByFisher(string category, string feature)
{
    double fpsum = 0.0;

    foreach (string catName in _categoryNames)
    {
```

```
            fpsum += getFeatureProbInItemCategory(catName, feature);
        }
        if (fpsum > 0.0)
        {
            return getFeatureProbInItemCategory(category, feature) / fpsum;
        }
        return 0.0;
    }
```

由于我们已经进行了一次权重迭代的计算，所以不再需要公式中的 P(B|A)来进行概率的预判断值，于是概率的计算公式就被简化如公式 4.16 所示：

$$Pr(cat|doc) = \prod wprob(feature, cat) \quad \text{（公式 4.16）}$$

Fisher 优化还包含另外一个环节，就是由于相互独立的词频概率是被一个个主次相乘起来而获得的最终的结果，数值将会变得非常非常小以至于难以直观的比较概率的的大小。对于下面的公式 4.17 来讲：

$$-2 * \log(\Pr(cat|doc)) \quad \text{（公式 4.17）}$$

这样计算之后的结果符合卡方分布，所以我们需要将卡方分布变为线性分布，具体的计算过程代码实现如下：

```
    private double invchi2(double chi, int df){
        double m = chi / 2.0;
        double term = Math.Exp(-m);
        double sum = term;
        for (int i = 1; i < df / 2; i++){
            term *= m / i;
            sum += term;
        }
        return sum > 1.0 ? 1.0 : sum;
    }
```

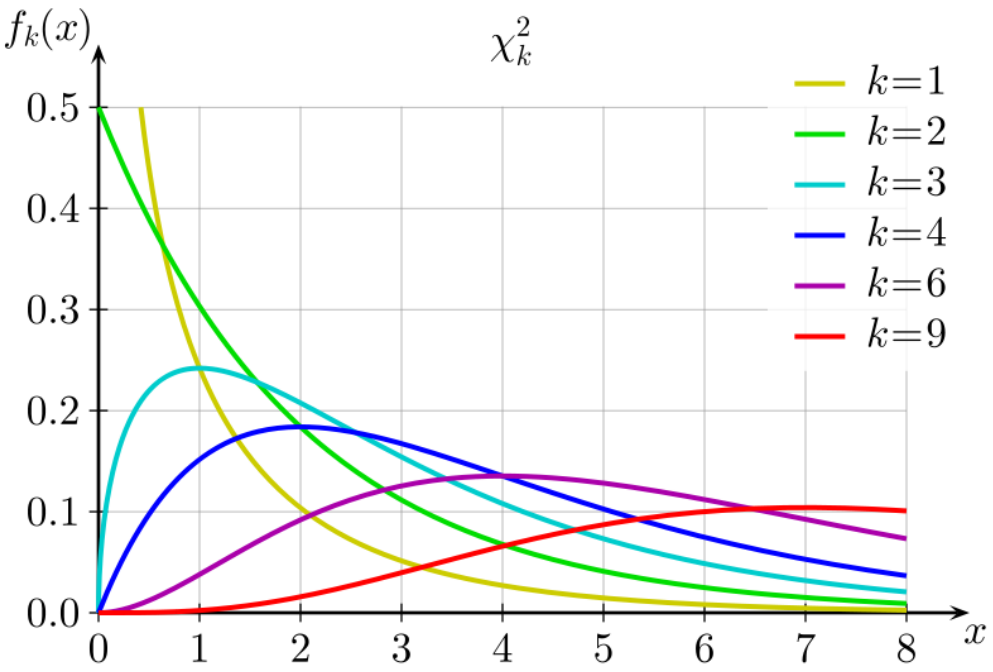

图 27　卡方分布

所以对于已经这样线性化的结果来看的，最终的分类是很容易直观的用程序简单的比较分辨出最终的分类结果的。进行最终分类判断的代码实现如下：

```
private double getProbFisher(string category, string[] features)
{
    double p = 1.0;

    foreach (string feature in features)
    {
        p *= getFeatureWeightedProbByFisher(category, feature);
    }

    double fscore = -2 * Math.Log(p);

    return invchi2(fscore, (features.Length+1) * 2);
}
```

4.6.5 贝叶斯分类器的 Revised 优化及代码实现

在项目进行的过程当中，由于现有的成熟的朴素贝叶斯分类器算法的准确度依然还存在可优化的空间，所以，除却 Fisher 方法对于训练集不平衡的优化，在做项目的过程中还发现了另外一种对于精准度方面的优化，首先是对于公式 4.3 的优化：

$$P(category) = \frac{total(feature\ in\ category)}{\sum total(features)}$$ （公式 4.18）

通过公式 4.18 我们可以看得出，对于先验概率的计算，通积累 feature 的数量来计算概率比积累 document 的数量来得更加的精细，在小规模训练集的适用场景下，使用 feature 积累的效果更加明显。具体实现代码如下：

```
private double getCategoryProbByFeatures(string category)
{
    if (_featuresCountInCategory.ContainsKey(category))
    {
        return (double)getAllFeaturesTotalInCategory(category) /
getAllFeaturesTotalInAllCategory();
    }
    return 0.0;
}
```

其次测试最重要的地方在于对 weighted prob 的计算（公式 4.4 和公式 4.6）优化，见如下公式：

$$P(w|category) = \frac{total(w\ in\ this\ category)}{total(w\ in\ all\ category)}$$ （公式 4.19）

$$P(w|category) = \frac{weight*baseprob+\ total(w)*P(w|category)}{weight+total(w)}$$ （公式 4.20）

通过这层优化，基本消除了 document 中出现一次还是多次 feature 对于最后结果的巨大影响，通过直接计算 feature 出现在 category 中的概率的成绩，来衡量 feature 出现在 category 中的概率。具体实现代码如下：

```
private double getFeatureWeightedProbByRevised(string category, string feature)
{

    double weight = 1.0;
    double allProb = weight / getCategoryNamesCount();

    double basicProb = getFeatureProbByFeatureCount(category, feature);
    double totals = getFeatureTotalInAllCategory(feature);

    return (weight * allProb + basicProb * totals) / (weight + totals);
}

private double getFeatureProbByFeatureCount(string category, string feature)
{
    if (_featuresCountInAllCategory.ContainsKey(feature))
    {
        return (double)getFeatureTotalInCategory(category, feature) /
```

```
getFeatureTotalInAllCategory(feature);
            }
            return 0.0;
        }
```

对于最终的 category 判断来讲，与其他贝叶斯分类器的算法比较并没有使用多少优化，因为通过概率来判断最终的类别的理论基础依然是最普通的朴素贝叶斯定理，具体实现代码如下：

```
        private double getProbRevised(string category, string[] features)
        {

            double catProb = getCategoryProbByFeatures(category);

            double docProb = 1.0;
            foreach (string feature in features)
            {
                docProb *= getFeatureWeightedProbByRevised(category, feature); ;
            }
            return catProb * docProb;
        }
```

4.7 本章小结

本章的前半部分给出了过滤系统的总体架构设计，采用了多分类器多层树状过滤的结构，这样的结构好处在于可以完整当今各种各样的分类过滤需求。

本章的后半部分对每个大型的Strategy下面的小分类器的结构进行了详细的讲述，无论是从技术的实现角度还是从算法理论实施计算的角度都进行了详尽的介绍，最后讲述了整个贝叶斯分类器的模块结构，根据分类器的需求结论设计了该模块的模型，最终实现了一个机器学习的分类功模块能。

第五章 系统测试和运行效果分析

在本系统的设计与实现过程中，软件测试过程是用来确认一个分类策略的效果品质或性能是否符合之前设计所提出的一些要求，验证是否达到预期的目标。

为了使系统质量得到有力的保证，对本系统采用了同训练集不同分类策略的测试。本章通过对软件测试的相关研究，结合本系统的实际情况，确定测试技术及方案。通过测试执行，并对测试结果分析优化。本系统主要是对效果进行了测试，论文也在此方面进行了论述。

5.1 系统测试策略

5.1.1 测试目标

测试针对的目标为整个系统的全面测试，既包含各个子模块的健壮性（建立起来的分类树能否进行有效的分类）方面的测试，也包含功能性测试（每个模块能否进行有效地拆解利用），又包括对整个系统架构的安全性及各种性能参数的验证。

5.1.2 测试原则

对于分类系统的测试而言，从不同的角度来看会产生两种不同的测试原则：

第一、从用户的角度来看，用户希望的是在不误删正常信息的前提下，尽可能地将垃圾信息过滤掉，因为假如因为过滤垃圾信息这样的次重要事件影响到了重要信息的接受，那么也没有必要使用垃圾过滤系统，这个系统设计的使用价值就会非常低非常低，背离了初衷，等于因小失大。

第二从研究测试的角度出发的角度出发，则我们就是希望不管是 Precision 还是 Recall 都可以尽可能的提高百分比，这样一来证明系统算法的有效性会越来越高，效果会越来越好，但是这样的目标会背离用户利益的初衷，也不是十全十美。

所有系统进行测试时都须有一定的标准与原则，测试原则是为测试者提供依据，起到约束的作用，有利于促进软件的测试。在测试时所采取的主要原则如下：

1. 所有的测试都应追溯到用户需求。软件测试的目标在于揭示缺陷。从用户角度来看，最严重的缺陷是那些导致程序无法满足需求的缺陷。

2. 独立的第三方来构造测试来保证测试的质量。第三方测试最大的优点在于可以保证测试具备足够的专业性、独立性、客观性和公正性。

3. 在测试工作真正开始前的较长时间内就进行测试计划。测试计划可以在需求模型一完成就开始，详细的测试用例定义可以在设计模型被确定后立即开始。因此，测试理应在编写代码之前就可以进行计划。

4. 对于整个系统的完全测试是不可能的，测试在需要的时候需要适可而止。测试某种程度上无法显示软件潜在的缺陷。在测试中不可能枚举出运行路径的每一种组合功能树。然而，充分覆盖程序逻辑的同时确保程序设计中已经使用了所有的条件却是有可能的。

5. 尽量避免测试的随意性。测试计划应包括所有系统的安全性测试，完整性测试以及性能方面的测试。

6. 每次程序修改后必须要进行要回归测试。

7. 应一直保留 Testcase，直至整个系用废弃为止。

根据以上的测试标准，微博垃圾过滤系统的测试在项目早期就已经设计好了相关的测试计划和测试过程。

5.2 系统测试方案

5.2.1 测试环境配置

由于微博垃圾信息过滤系统实现的代码由 C#编写，在 VS2012 下编译运行可以看到系统的运行结果,，在测试时对测试环境配置上的要求只需要.NET Framework 4.0 以上的环境支持就足以完成执行测试的要求。

5.2.2 测试工具配置

由于程序的复杂性极高，对于程序的测试工具都由自己来进行编写，工具会报告分类器进行训练之后的模型，以及进行分类测试后Precision和Recall的报告工具，都由自己来亲自进行编写。

5.2.3 测试用例设计

微博垃圾信息过滤系统主要的测试范围主要在于分类的准确性方面，对于已经标记好的数据，测试的主要办法基本对于不同的模块测试有两种测试方法：

1） 直接测试

对于只具备简单分类器的策略模块，我们只需要把规则建立好然后把标记好的数据进行分类就可以判断出分类的有效性。

2） 抽样训练测试

对于具备机器学习能力的分类器的策略模块，我们则需要从标记好的实验数据中抽取一定百分比的实验数据先对分类策略中的每个子分类器进行训练之后再对余下的数据进行分类准确度进行测试。

5.2.4 测试实例说明

假设我们有一个策略需求只是一个单一的朴素贝叶斯分类器，分类器的 Filter 是将所有的字符转变成小写字符，而 Detector 我们则配置成可以进行中文分词的特征拆解模块，然后我们就可以用下面的方法测试训练这个分类策略，测试代码如下：

```
// 训练过程
ClassifierMLBayes c = (ClassifierMLBayes)tempT[0];
c._filter = new FilterToLowcase();
c._detector = new DetectorPangu();

c.train("篱笆上棕色的小浣熊，萌翻了嗷嗷", "Good");
c.train("可爱动物明星们的视频集锦", "Good");
c.train("保护我们的地球，珍惜淡水资源", "Good");
c.train("精致的小清新搭配，淘宝女装限时八折", "Spam");
c.train("快速赚钱秘笈，请点击 http://t.aRsic", "Spam");
c.showModelInfo(null, true);
```

训练之后模型文件的输出如下所示：

WORD	all	S	G	b(s)	b(g)	f(s)	f(g)	r(s)	r(g)
秘笈	1	1	0	50.00%	25.00%	75.00%	25.00%	75.00%	25.00%
女装	1	1	0	50.00%	25.00%	75.00%	25.00%	75.00%	25.00%
限时	1	1	0	50.00%	25.00%	75.00%	25.00%	75.00%	25.00%
赚钱	1	1	0	50.00%	25.00%	75.00%	25.00%	75.00%	25.00%
清新	1	1	0	50.00%	25.00%	75.00%	25.00%	75.00%	25.00%
淘宝	1	1	0	50.00%	25.00%	75.00%	25.00%	75.00%	25.00%
的	3	1	2	50.00%	62.50%	44.64%	55.36%	37.50%	62.50%
精致	1	1	0	50.00%	25.00%	75.00%	25.00%	75.00%	25.00%
点击	1	1	0	50.00%	25.00%	75.00%	25.00%	75.00%	25.00%
明星	1	0	1	25.00%	41.67%	25.00%	75.00%	25.00%	75.00%
淡水	1	0	1	25.00%	41.67%	25.00%	75.00%	25.00%	75.00%
资源	1	0	1	25.00%	41.67%	25.00%	75.00%	25.00%	75.00%
动物	1	0	1	25.00%	41.67%	25.00%	75.00%	25.00%	75.00%
嗷	2	0	2	16.67%	61.11%	16.67%	83.33%	16.67%	83.33%
篱笆	1	0	1	25.00%	41.67%	25.00%	75.00%	25.00%	75.00%
棕色	1	0	1	25.00%	41.67%	25.00%	75.00%	25.00%	75.00%
浣熊	1	0	1	25.00%	41.67%	25.00%	75.00%	25.00%	75.00%

……

最终我们可以看到在五个训练集训练完成之后的Model会产生一长串的词频列表的统计，词频记录了每个feature出现的总次数和分类中的次数，以及三种不同的算法对于这个单个词频概率权重的数值，从测试结果来看，这些数值完全符合当初算法实现的预期。

接下来我们可以用一个测试用例来测试经过训练之后的文本：

```
Strategy strategySingleNaiveBayes = StrategyFactory.createDemoSingleBayesStrategy();
Console.WriteLine(strategySingleNaiveBayes.judgeItem("棕色的"));
```

样例输出：

Probability(Good) = 0.156250 (0.6 * 0.4166… * 0.625)

Probability(Spam) = 0.050000 (0.4 * 0.25 * 0.5)

得出的结果依然与算法预期的结果是一样的，说明程序对于算法实现的正确性是没有问题的。

5.3 测试结果分析

对于最后的测试而言，我们需要看到的指标主要在于两个方面：

1）查准率（Precision）：也就是机器判断所有文本中属猪某个 Category 的数量确实同人类标记的真正的 Category 是属于相同的类别。

2）查全率（Recall）：也就是机器正确检测出来的类别覆盖了这个类别所有数量的比例，而且这里不仅要看对于垃圾信息的过滤的覆盖率，而且基于用户需求来讲，对于非垃圾信息的过滤的覆盖率也很重要，因为这个数值直接决定了有多少可能会成为重要信息的无辜信息被误判。

5.3.1 简单策略的效果测试

如之前所讲述的其中一个策略那样设计的分类器的策略在真实的实验数据中的表现在最终的测试中的数据就像之前所讲的那样，只使用简单过滤器的策略虽然会有一定的识别能力，但是必然不会有很好的效果，以下这组数据说明了问题，只有不到 20%的垃圾信息数据被过滤了出来，剩下的 80%垃圾信息没有被很好的识别而且与此同时，符合这个固定特征的非垃圾信息也被误判为垃圾信息，这样的分类器离我们的预期是相差甚远，下表为数据报告：

表 1 简单策略的效果测试数据

Man \ Com	Spam	Normal	Total	Recall
Spam	260	1218	1478	**17.59%**
Normal	160	9005	9165	**98.25%**
Total	420	10223	10643	
Precision	**61.90%**	88.09%		87.05%

5.3.2 单朴素贝叶斯分类器使用正反例对等的数据集合

在完成了朴素贝叶斯的算法之后，我们要测试整个算法的可行性，由之前的论述可以得到，仅仅是使用了朴素贝叶斯算法的分类器会受到训练集的比例的严重影响，为了保守起见，首先采用的训练数据是 1:1 的训练集，经过训练之后对于测试数据的分类情况如下表所示，我们可以看到我们的分类器对于的数据的识别能力相对于简单策略下的识别能力已经有了飞跃般的提升，但是效果依然不尽人意，因为误判的数量实在是太多了，这样的分类器虽然有一定的识别能力但是没有任何的应用价值，所以我们需要进一步对分类器进行优化。

表 2　单贝朴素叶斯等比例测试数据

Man \ Com	Spam	Normal	Total	Recall
Spam	1218	260	1478	**82.41%**
Normal	988	8177	9165	**89.22%**
Total	2206	8473	10643	
Precision	**55.21%**	96.92%		88.27

5.3.3 单朴素贝叶斯分类器使用接近真实比例的数据集合

因为从分类的角度来言，我们给予概率来判断的时候有时候训练集应该更接近于真实比例的时候，根据已经发生的事件作出的下一步预测才会特别的准确，所以在使用 1:5 的数据做训练集之后就会看到下表所示的结果，结果显示在调整了训练集比例之后，对于垃圾信息的过滤能力有一定程度的下降，但是误删的比例同时也下降表示算法本身更加安全，此时算法的实用价值提高了，但是高于 5%的误判率依然是一个比较高的问题所在。

表 3　单朴素贝叶斯接近真实比例测试数据

Man \ Com	Spam	Normal	Total	Recall
Spam	1194	284	1478	**80.78%**
Normal	632	8533	9165	**93.10%**
Total	1826	8817	10643	
Precision	**65.39%**	96.78%		91.39%

5.3.4 训练集比例反向破坏性测试

为了同时验证分类器在训练集反差比较大的情况下会危险到一个什么样的程度，在这里同时做了一组反向测试，观察训练集反向为 5:1 之后的结果，如下表中所示，在观察过过多的垃圾信息之后的分类器本身的“世界观也变得非常的黑暗”，所以近半数的无辜信息被误判成垃圾信息。

表 4 单朴素贝叶斯反比例破坏性测试数据

Man \ Com	Spam	Normal	Total	Recall
Spam	1320	158	1478	**89.31%**
Normal	3725	5440	9165	**59.36%**
Total	5045	5598	10643	
Precision	**26.16%**	97.18%		63.52%

5.3.5 使用 Fisher 优化的算法分类器测试

前文介绍了一种叫做 Fisher 的优化方法，使用了 Fisher 算法的分类器可以无视训练集比例的影响，如下表所示的结果，我们可以看到不管是识别率和安全性都有提高，只是还需要进行更多的优化才能达到让人满意的效果。

表 5 使用 Fisher 优化之后的测试数据

Man \ Com	Spam	Normal	Total	Recall
Spam	1212	266	1478	**82.00%**
Normal	677	8488	9165	**92.61%**
Total	1889	8754	10643	
Precision	**61.16%**	96.96%		91.14%

5.3.6 之前的基础上加入了二审裁决优化的

之所以设计这样的架构，就在于当我们进行一步分类之后还有更加优化的可能，我们可以在已经过滤出来的“嫌疑垃圾信息”中在用一个比较保守的分类器进行二审保证不“错杀一个好人”，我们可以看得出，虽然在二审中放过了很多无辜的垃圾信息，但是却导致了覆盖率下降了，不过即使是这样的情况，算法的应用价值却得到了极大的提高，这种级别的算法基本可以保证不错漏掉重要的信息。

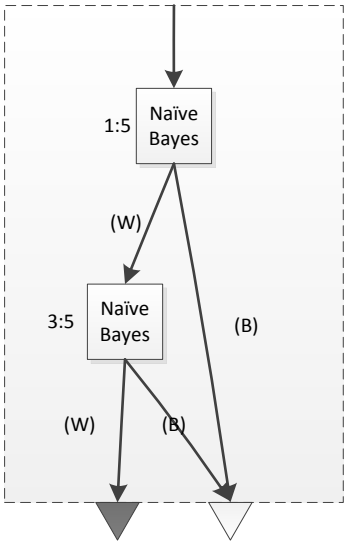

图 28　二级优化的示意图

下表则为经过二级优化后的测试结果：

表 6　加入二级优化后的测试数据

Man \ Com	Spam	Normal	Total	Recall
Spam	924	554	1478	**62.51%**
Normal	6	9159	9165	**99.93%**
Total	930	9713	10643	
Precision	**99.35%**	94.30%		94.74%

5.3.7 混合了多种优化分类器策略之后的结果

那在这个基础上我们是不是可以设计更加灵活的分类器来进一步提高呢？下面就是添加了多层分类器的结果，把覆盖率提高了近 10%，同时误判率也没有太大的下降，这样就是一个具有很高应用价值的分类策略算法了。

表 7　多策略混合优化后的测试数据

Man \ Com	Spam	Normal	Total	Recall
Spam	1080	398	1478	**73.07%**
Normal	27	9138	9165	**99.71%**
Total	1107	9536	10643	
Precision	**97.56%**	95.83%		96.01%

5.3.8 不同测试数据的测试结果

在试验了另一批实验数据的时候，结果却不如之前的分类器那么理想，除去新的测试数据中的噪音数据的元素，也因为前面的部分优化是根据固定的训练集研究进行的，当环境变化的时候，有些过度的优化或许并不能产生足够的优势，所以这个实验平台存在的另一个意义就在于可以根据文本环境的变化快速的进行创新实验迭代，实验进化然后新策略投入使用，由人类来监督指导的机器来弥补目前的缺陷是监督学习中最好的方法。

表 8 不同测试数据的测试数据

Man \ Com	Spam	Normal	Total	Recall
Spam	1768	389	2157	**81.97%**
Normal	371	5901	6272	**94.08%**
Total	2139	6290	8429	
Precision	**82.66%**	93.82%		90.98%

综上所述，各项测试已经能说明分类系统满足预计的功能要求。

5.4 本章小结

本章描述了本项目的测试设计，在实际测试中通过可测的数据和操作尽可能的达到了全面覆盖，一系列的第三方程序的测试输出的数据已经显示出分类效果的足够优秀，这些数据足以证明本项目已经达到了初始设定的目标，同时为项目之后的扩展打下了坚实的架构基础，为未来的项目投入时间的研究应用开了一个好头。

总结与展望

1）结论

在本项目的研发过程中，主要研究了对机器学习数据挖掘学科的相关理论和应用案例。重点是对机器学习中的分类系统的实现进行了研究和实践。通过对某微博垃圾信息过滤系统的研究与分析，运用 OOP 编程语言实现了一套非常灵活的架构。然后对微博的数据的结构进行了分析，并编写了爬虫程序进行数据的抓取，进而为了给抓取到的程序进行快速准确的人工分类而书写了短小精悍的标记工具软件，而后对系统各模块的主要功能的流程及数据结构进行代码的设计实现。最后对系统测试的原则和方案进行了设计与规划，并根据。

项目基本达到了对垃圾信息过滤的目标，而垃圾信息最主要的是铺天盖地的广告信息，而项目的价值在于可以灵活扩展与垃圾信息的发布者进行对抗，而且不只可以运用于微博垃圾信息的过滤，而且还可以运用于各种各样的比如说短信这样的短文本环境。

2）个人收获

通过本项目，本人对机器学习以及数据挖掘有了更为全面的理解，对设计模式的驾驭更加的熟练，对面向对象的软件开发技术有了更为深入的掌握，在项目管理过程中也养成了良好的个人工作作风习惯，能够有能力安排好工作实习与研究的任务，同时协调好两边的时间分配将两边都不耽误的保质保量把任务完成。

3）展望

本论文的研究工作虽然已经完成了一套中文文本分类系统的研发，虽然通过优化之后的分类系统可以定制化的在特定文本的环境下达到很高很好的效果，但是依然存在两点不足：

第一点是这种优化需要人工去分析拆解文本的各种特征问题，虽然整个分类架构可以融合当今各种各样的机器学习算法思路，汇百家之长，但是却没有提出一种固定可行的可以衡量各种机器学习算法在某种文本环境下的效果，这样就为新策略的设计运用提出了很大的难度要求。

第二点就是算法无法自动的适应新文本环境的变化，只能根据固定的环境和训练集进行分类，不具备适应和成长能力，如何让新的分类器可以进行自我训练和进化成长，会是今后很大的努力方向。

最后，总而言之，在互联网蓬勃发展信息爆炸的时代，我们对于一个分类器预先对

我们可能想要的信息进行与处理的需求会一直存在，这边文章设计的架构只不过是开了一头，后续还有很多优化值得我们去改进去挖掘，相信技术的发展可以让我们的生活变得越来越幸福美好。

参考文献

[1]杨挚诚，基于机器学习的文本分类算法研究，广西大学，2007

[2]Nitin Jindal and Bing Liu, Opinion Spam and Analysis, University of Illinois at Chicago, 2008

[3]Guan Wang, Sihong Xie, Bing Liu, Philip S. Yu, Review Graph based Online Store Review Spammer Detection, University of Illinois at Chicago, 2011

[4]Fangtao Li, Minlie Huang, Yi Yang and Xiaoyan Zhu, Learning to Identify Review Spam, Tsinghua National Laboratory for Information Science and Technology Dept. of Computer Science and Technology, Tsinghua University, 2011

[5]Youcef Begriche, Ahmed Serhrouchni, Bayesian Statistical Analysis for Spams, Institut Telecom, Telecom ParisTech, 2010

[6]Biju Issac, Wendy Japutra Jap, Jofry Hadi Sutanto, Improved Bayesian Anti-Spam Filter - Implementation and Analysis on Independent Spam Corpuses, School of Computing and Design Swinburne University of Technology, 2009

[7]Safvan Vahora, Mosin Hasan, Reshma Lakhani, Novel Approach: Naïve Bayes with Vector, Space Model for Spam Classification.

[8]Chun Chao Yeh, Soun Jan Chiang, Revisit Bayesian Approaches for Spam Detection, National Taiwan University.

[9]Fuchun Peng, Fangfang Feng, Andrew McCallum, Chinese Segmentation and New Word Detection using Conditional Random Fields, University of Massachusetts Amherst.

[10]孙铁利、刘延吉，中文分词技术的研究现状与困难，东北师范大学计算机学院，2009

[11]Toby Segaram, Programming Collective Intelligence, 2007

[12]高淑琴, Web 文本分类技术研究现状述评, 江苏徐州师范大学图书馆, 2008

[13]苏淑玲, 机器学习的发展现状及其相关研究, 江门职业技术学院, 2007

[14]吴雅娟、柳培林、丁子睿，基于统计分词的中文文本分类系统，大庆石油学院计算机与信息技术学院，2005

[15]石洪波、王志海、黄厚宽，贝叶斯文本分类方法研究，山西财经大学学报(高等教

育版)，2002 年增刊，87-88

[16]耿新青、陶凤梅、黄宏光，一种基于近邻匹配的中文分词算法 Jlppeccz，鞍山师范学院数学系，2010

[17]别玉玉、刘飞、张书伟，贝叶斯垃圾邮件过滤算法的改进与实现，中国矿业大学，2010

[18]方辉、王倩，支持向量机的算法研究，长春师范学院学报(自然科学版)，2007.6，90-91

[19]王潇、胡鑫，三种文本分类算法的比较，石河子大学学报(自然科学版)，2005.12，770-771

[20]崔争艳，中文短文本分类的相关技术研究，河南大学，2011

致谢

在此衷心感谢邵兵和谭火彬两位导师在我的两年半的研究生学习生活中，对我的悉心教导和大力帮助，给予我在学术研究方向、项目实践中的指导。在我的论文开题、项目实施和论文撰写过程中，他认真批阅，多次给予去富有启发性和建设性的意见。同时也在此感谢康一梅老师在论文审阅过程当中给我的建议和指导。

同时要感谢我所在的实习公司，微软亚洲互联网工程院社交计算平台组的领导及同事，特别是我的企业导师林刚给了我很多专业技术上的指导。在为期一年多的课题研究实施中，从中我学到了对我而言完全陌生的互联网软件工程领域的管理及业务知识。了解了这一领域广阔的前景，掌握了软件工程的实施规范，真正地将书本中的理论知识实际运用到商业软件开发中，懂得了作为一名职业研发人应该遵守的职业操守。这一切，为我即将走上的工作岗位打下了良好的基础。

特别感谢亲爱的父母，谢谢父母辛辛苦苦培养我成人，谢谢父母无微不至的关爱，谢谢父母对我无怨无悔的支持。

感谢一直以来给我帮助的同学、朋友，分享了许许多多的欢乐，也分担了许许多多的忧愁，我们在欢笑与泪水中互相成长。

最后向世间所有的相遇感恩，世界那么大，能遇见你们多么的不容易。