

# 互联网广告分类器设计与训练

胡玮玮(2006011293) 孔祥欣(2006011299) 徐恺(2006011281)

清华大学计算机科学与技术系计 62 班

{huww06, kxx006, jaguarxk000}@gmail.com

## 摘要:

文本分类是将自然文本根据内容自动分为预先定义的一个或几个类别的过程。本文介绍了数据挖掘课程中提出的互联网广告分类器设计与训练实验的实施过程。在该实验中,搭建了一个比较完善的实验系统,可以进行再次利用。实现了特征频率 **TF** 和  $x^2$  统计量 **CHI** 特征选择算法以及 **FOIL** 算法和 **PRM** 分类算法,通过采取不同算法配对并设置不同参数的方式进行实验结果比较,并进行一定的分析。

**关键字:** 文本分类; 特征选择; 分类算法;

## 0 引言

近年来随着互联网的不断发展,产生了各种各样的电子数据,如何在海量的数据中挖掘出有用的信息成为了一个新的课题。在这个背景下,我们学习了数据挖掘课程,并利用所学的知识设计与实现一个基于文本的分类器,旨在能够对符合一定特征的数据进行分类。

## 1 实验内容

数据挖掘课程提出了互联网广告分类器设计与训练项目。项目提供了一个数据集,其中包括了在互联网页面上呈现出的可能广告,并提供了若干属性,对于离散的属性值来说只有 0 和 1 两种分类结果。还包括了三个连续的属性,包括图片长度、宽度、比例等。除此以外提供了一些数据,针对每一个记录(instance)将其定义为广告(ad)或者非广告(nonad)。

项目要求从这些提供的数据中挖掘出适当的有用信息,以这些有用的信息作为依据,对其他给定的若干数据进行分类,判定是否属于广告。

项目建议使用 weka 工具进行实验,同时鼓励自己编程实现数据挖掘的过程,或者对相关算法进行适当的改进。

## 2 算法分析

我们首先对这个项目的要求进行分析,了解项目的目的,以及提供的数据集的格式。进而熟悉数据挖掘的流程,经过课堂的学习和查阅相关论文,我们认识如下:

数据挖掘过程中存在如下两个名词,一个是训练集,一个是测试集以及相应的标准答案,我们需要做的事情是从训练集中训练出一个分类器,进而使用这个分类器对测试集中的数据进行测试,与标准答案比较即可得到该分类器的效果。一个典型的分类过程包括如下几个步骤:数据预处理→特征选择→训练分类器→分类。因此我们将从以上的四个部分来对我们的算法进行介绍。

### 2.1 数据预处理

在进行数据预处理之前,我们首先需要认识项目已经提供给我们的数据文件格式。我们并没有使用助教已经构造好的 ad.arff 文件,相反我们使用了 ad.names 和 ad.data 两个文件。下面进行分别介绍:

ad.name 文件:该文件的主要作用是提供了数据的若干属性,其中一条典型的语句如下格式:

url\*likesbooks.com: 0,1.

其中“\*”号之前的“url”表示该属性的类别是一个 url,“\*”号与“:”之间的 likesbooks.com 表示

该属性的名称，而“:”以后的值以“,”为间隔分别表示该属性的取值范围。

**ad.data** 文件：该文件的主要作用是提供了所有的相关数据，其中一条典型的语句如下格式：

125,125,1.0,1.0+,1+,ad.

假设在 **ad.name** 文件中属性个数为  $n$ ，那么在以上的一条数据中，以“,”为分隔符，应该可以得到  $n+1$  个值，其中前  $n$  个值分别对应每一个属性的取值，最后一个值标记该条记录是否是广告，“ad”表示该记录为广告，“nonad”表示该记录非广告。

了解完了文件格式以后，下面我们要进行数据处理，在导入文件的过程中，我们进行了如下处理：

首先读取 **ad.name**，我们定义了一个 `Vector<String> attribute`；用来保存所有属性，读完以后得到一个属性数目 `attributes`。进而读取 **ad.data**，我们定义了 `Vector<String> dataString`；来保存原始的数据，读完以后得到一个记录条数 `instances`。

然后我们针对 `dataString` 中的内容进行处理，对于其中的每一条记录，首先根据后缀判断该条记录是否是广告，并将该记录保存到 `Vector<Integer> type`；中。然后通过“,”分词，当值为“?”即表示该值缺失，那么将其设置为“0”，由于前三列为连续值属性，所以对其进行离散化，采用在最大与最小值之间进行均分的方式。当这些都处理完毕以后将数据保存至 `int [][]data` 中。到此为止，数据处理结束，已经给下一步骤提供了比较符合规范的数据。

## 2.2 特征选择

以上数据经过处理以后虽然比较规范，但是可以发现它们具有很高的维度，如果以这样的训练集来训练分类器的话那么对大大影响训练的效率。因此需要对数据进行降维操作。传统的特征选择算法有特征频率、期望交叉熵、信息增益、 $\chi^2$  统计量等方法。在本次实验中我们实现的算法有特征频率 **TF** 和  $\chi^2$  统计量 **CHI** 算法。下面进行详细介绍。

### 2.2.1 特征频率 TF

特征频率 (Term Frequency, TF) 指文本集中特征词  $t$  出现的次数，是最简单的特征选择方法。它认为特征在文本集中出现次数越多，对文本分类的贡献越大。在本次实验中，通过 `int []frequency` 数组来对各个特征词出现的次数进行统计，设置域值 `TFYIELD`，当特征词出现次数大于该域值时选择此特征项。本实验中选择 `Vector<Integer>[] processedData`；来保存经过特征选择以后的数据，`Vector<Integer> result`；用来

保存经过选择以后的特征词的序号。

### 2.2.2 $\chi^2$ 统计量 CHI

$\chi^2$  统计量 (Chi-square, CHI) 的主要思想是：认为词条与类别之间没有独立性，并可类比为在一个自由度的  $\chi^2$  分布， $\chi^2$  统计量的值越高，词条和类别之间的独立性越小，相关性也就越强。特征词  $t$  对类别  $c_i$  的 CHI 值公式如下：

$$\chi^2(t, c_i) = \frac{N * (AD - CB)^2}{(A + C) * (B + D) * (A + B) * (C + D)}$$

其中  $A$  表示属于  $c_i$  类且包含  $t$  的文档次数， $B$  表示不属于  $c_i$  类但包含  $t$  的文档次数， $C$  表示属于  $c_i$  类但不包含  $t$  的文档次数， $D$  表示不属于  $c_i$  类也不包含  $t$  的文档次数， $N$  为训练语料中的文档总数，且  $N=A+B+C+D$ ，同时满足要求  $A*D>B*C$ 。分别计算  $t$  对每一类的 CHI 值，再用下式计算词条  $t$  对于整个训练集的 CHI 值。

$$\chi^2_{\text{arg}} = \sum_{i=1}^c P(c_i) \chi^2(t, c_i)$$

其中  $c$  表示总类别数， $P(c_i)$  表示样本属于  $c_i$  类的频率。在本次实验中设置域值 `CHIYIELD`，根据以上公式计算的结果，过滤低于该域值的词条，保留高于该域值的词条作为特征项，然后对数据的维数进行降维即可。

## 2.3 训练分类器

文本分类系统就是按照文本的主题以及之前选择好的特征值，将具体的文本划归为适当类别的计算机系统，该系统的一个核心部分就是现在所说的分类器。分类器是针对已经降维以后的训练集，采用适当的分类算法，得到一个分类函数。该分类函数输入一个测试数据以后能够返回是否属于某一类。在这个过程中，采用的分类算法是一个关键，常用的分类算法主要分为两类：一是基于统计的方法，如简单贝叶斯、 $k$  最近邻方法、类中心向量方法、回归模型、支持向量机等，还有一种是基于规则的方法，如决策树、FOIL 等。在本次实验中，我们采用的是 FOIL 算法以及对 FOIL 算法改进以后的 PRM 算法，具体算法设计如下：

### 2.3.1 FOIL 算法

FOIL(First Order Inductive Learner)算法是属于贪心型的规则集产生算法。该算法的主要思想是通过不断的尝试条件，找到信息增益最大的条件加入到当前规则中。通过不断的重复条件的产生过程，最终找到

满足覆盖率条件的规则集。

该算法可以用如下伪码描述：

Procedure FOIL

```
rule set  $R \leftarrow \Phi$ 
while  $|P| > 0$ 
     $N' \leftarrow N, P' \leftarrow P$ 
    rule  $r \leftarrow \text{empty\_rule}$ 
    while  $|N'| > 0$  and
         $r.\text{length} < \text{max\_rule\_length}$ 
        find the literal  $p$  that brings most gain
            according to  $P'$  and  $N'$ 
        append  $p$  to  $r$ 
        remove from  $P'$  all examples not satisfying  $r$ 
        remove from  $N'$  all examples not satisfying  $r$ 
    end
     $R \leftarrow R \cup \{r\}$ 
    Remove from  $P$  all examples satisfying  $r$ 
end
return  $R$ 
```

其中，增益(gain)的计算方法如下：

假设当前的正元组为 $P$ ，负元组为 $N$ ，当将一个新的条件 $p$ 加入到规则 $r$ 中后，新的正元组为 $P^*$ ，负元组为 $N^*$ ，则

$$\text{gain}(p) = |P^*| \left( \log \frac{|P^*|}{|P^*| + |N^*|} - \log \frac{|P|}{|P| + |N|} \right)$$

由于每次产生规则后都将满足该规则的数据移出正元组，因此该算法每次循环数据规模都在减小，计算过程中的速度将逐渐加快，整体的运行时间较小。

### 2.3.2 PRM 算法

FOIL 算法在每次计算之后，都会将已经使用过的数据从当前的正元组中移除，也就说已经参与过规则产生的数据将不再参与数据产生。但已经产生的规则对于这些数据来说并不一定是最好的，因此，在 PRM 算法中，对每个数据设置一个权值，参与过规则产生的数据的权值将按一定比例减小，而不是直接移除，使得规则长生更合理。

PRM 算法的伪码如下：

Procedure Predictive Rule Mining

```
set the weight of every example to 1
rule set  $R \leftarrow \Phi$ 
 $\text{totalWeight} \leftarrow \text{TotalWeight}(P)$ 
while  $\text{TotalWeight}(P) > \delta \cdot \text{totalWeight}$ 
```

$N' \leftarrow N, P' \leftarrow P$

rule  $r \leftarrow \text{emptyrule}$

while true

find best literal  $p$  according to  $A'$

if  $\text{gain}(p) < \text{min\_gain}$  then break

append  $p$  to  $r$

for each example  $t$  in  $P' \cup N'$  not satisfying  
r's body

remove  $t$  from  $P'$  and  $N'$

end

end

$R \leftarrow R \cup r$

for each example  $t$  in  $P$  satisfying  $r$ 's body

$t.\text{weight} \leftarrow \alpha \cdot t.\text{weight}$

end

end

return  $R$

当然，没有将规则移除将导致每一次循环的规模是恒定的，使得算法的运行时间增长，同时由于已经使用过的数据没有删除，产生的规则的数量将显著增加，进一步影响了算法的效率。

## 2.4 分类

分类是本次挖掘过程的最后一步，针对训练集采用分类算法训练以后得到分类器，实际上分类器保存的就是很多规则，根据这些规则可以判断某数据是否属于某类。对测试数据进行一定的处理，主要是降维工作，然后将处理之后的数据作为参数传入到分类器中，分类器即可返回一个测试结果。将此测试结果与标准答案集相比较，经过适当计算，可以得到本次挖掘过程的准确率与覆盖率。

## 3 实验过程

本次实验中助教推荐了 weka 这个工具，但是我们并没有使用，我们是自己编程实现了整个挖掘过程，包括数据预处理、特征选择、训练分类器、分类等步骤。

### 3.1 实验环境

#### 3.1.1 编程语言：JAVA

#### 3.1.2 操作系统

MS Windows Vista Home Premium

MS Windows XP Professional SP2

### 3.1.3 编程环境

JDK 1.5.0.0

MyEclipse 7.0

Eclipse 3.4

### 3.1.4 SVN

Google Code

TortoiseSVN 1.5

## 3.2 实验系统搭建

本实验系统主要分为分类内核和操作界面两部分，分类内核即为挖掘过程的主要实现部分，而操作界面是为了方便助教重现结果而搭建的。该实验系统目前来看整体框架已经比较完整，可扩展性也比较好的，我们现在实现的是两个特征选择算法和两个分类算法，如果以后需要修改的话，只需要再添加新的函数即可以实现新的算法，可以继续为以后数据挖掘方面的实验服务。下面主要介绍该系统的如下两个方面：

### 3.2.1 分类内核

分类内核是数据挖掘过程的主要实现部分，包括了数据预处理、特征选择、训练分类器、分类等主要部分。各个部分在我们的内核中都有所体现。

#### (1) 数据预处理

本系统中数据预处理部分主要在 `PreProcess` 类中实现，其中包括了读入训练集文件以及测试文件，我们采用的读入文件不是 `arff` 类型的，但是我们读入以后可以输出这样的一个将属性值和数据合并到一起的 `arff` 文件，方便我们使用 `weka` 工具来验证我们的结果，因为 `weka` 使用的数据格式是 `arff` 的。虽然我们并没有使用 `weka`，但是我们还是可以输出这样的一个文件。

#### (2) 特征选择

本系统中特征选择部分主要在 `FeatureSelection` 类中实现，其中 `TF()`和 `CHI()`两个函数分别实现了特征频率和  $\chi^2$  统计量两个特征选择算法。经过特征选择以后的数据保存在 `Vector<Integer>[] processedData` 中，选择的属性序号保存在 `Vector<Integer> result` 中。

#### (3) 训练分类器

本系统中训练分类器主要就是两个算法，都在包 `org.kde9.algorithm` 中被实现，其中有两个类，一个是 `Foil` 类实现了 `FOIL` 算法，一个是 `PRM` 类实现了 `PRM` 算法。由于 `PRM` 是对 `FOIL` 算法的适当改进，所以我们采取的方式是让 `PRM` 类继承了 `FOIL` 类，重载了一些函数，以体现了两个算法之间的联系与不同。

#### (4) 分类

在 `org.kde9.entrance` 包里的 `main` 函数中，我们实现了一个无界面的主程序测试。

### 3.2.2 操作界面

本系统中操作界面部分主要集中在 `org.kde9.view` 包内，其中实现了一个简单的操作界面，可以帮助用户进行一个简单的数据挖掘过程，并查看结果。

## 3.3 实验测试

## 4 实验结果

## 5 实验总结

## 参考文献

- [1] Xiaoxin Yin and Jiawei Han, CPAR: Classification based on Predictive Association Rules;
- [2] 高亚波，文本分类系统的设计与实现，北京交通大学专业硕士学位论文，2008年6月；
- [3] 张俊丽，文本分类中的关键技术研究，华中师范大学硕士学位论文，2008年5月；
- [4] 余俊英，文本分类中特征选择方法的研究，江西师范大学硕士学位论文，2007年5月；