UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

THIAGO WINKLER ALVES

Applying Link-Based Spamdexing Detection Techniques

Final Report presented in partial fulfillment
of the requirements for the degree of
Bachelor of Computer Science

Prof. Luciana Salete Buriol
Advisor

Prof. Viviane Pereira Moreira
Co-advisor

Porto Alegre, July 2010

CIP – CATALOGING-IN-PUBLICATION

Alves, Thiago Winkler

Applying Link-Based Spamdexing Detection Techniques / Thiago Winkler Alves. – Porto Alegre: UFRGS, 2010.

51 f.: il.

Final Report (Bachelor) – Universidade Federal do Rio Grande do Sul. Curso de Ciência da Computação, Porto Alegre, BR–RS, 2010. Advisor: Luciana Salete Buriol; Co-advisor: Viviane Pereira Moreira.

1. Spamdexing. 2. Link analysis. 3. Adversarial information retrieval. 4. Evaluation. I. Buriol, Luciana Salete. II. Moreira, Viviane Pereira. III. Applying Link-Based Spamdexing Detection Techniques.

“Man is a genius when he is dreaming.”
— AKIRA KUROSAWA

ACKNOWLEDGMENTS

I would like to thank my parents, sister and grandmothers, for giving me all the love they could, and for always doing the impossible to provide me with the best possible education, helping me to reach this point of my life and becoming who I am. Specifically for this work, I would like to cite some names who were fundamental for its development.

Eduardo Gastal, a colleague and friend of mine that, during some difficult moments, took the time to help me think about the problems I was dealing with, even though the area of study of this work has nothing to do with his.

Carlos Castillo, co-author of the Truncated PageRank algorithm, who has helped me in the understanding of some concepts and also gave me valuable tips, answering my emails with a fantastic timing.

Christian Northfleet, also a colleague and friend of mine, who has made me a favor by revising every single line of this work (even this one). A big favor, since English is not my native language.

Larissa Nebesnyj, my girlfriend, who has also helped me with some corrections and has done some wonderful work making my figures look prettier.

My little cat, Lilica, who accompanied me in every long night I have stayed awake while producing this work.

And finally, my two advisors, Luciana Buriol and Viviane Moreira, whose great, valuable advices and support were fundamental for this work to be fulfilled.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS 7

LIST OF SYMBOLS 8

LIST OF FIGURES 9

LIST OF TABLES 10

ABSTRACT 11

RESUMO 12

1 INTRODUCTION 13
1.1 Goals 14
1.2 Organization 14

2 WEB GRAPH, AND EVALUATION MEASURES 15
2.1 Web Graph 15
2.2 Evaluation Measures 16

3 RANKING ALGORITHMS 17
3.1 PageRank 17
3.2 Functional Ranking 19
3.3 Truncated PageRank 19
3.4 TrustRank 21
3.5 Inverted TrustRank 23

4 IMPLEMENTATION DETAILS 24
4.1 WebGraph Framework 24
4.2 Dataset 26
4.3 Truncated PageRank 27
4.4 TrustRank 30
4.4.1 Classic TrustRank 30
4.4.2 Inverted TrustRank 31
4.5 Extra Attributes 32
4.5.1 Assortativity 33
4.5.2 Average in-links in out-neighbors 33
4.5.3 Average out-links in in-neighbors 34
4.5.4 In-degree and Out-degree 34
4.5.5 Reciprocity 34

4.6 Input and Output Formats 35
4.6.1 Input Formats 35
4.6.2 Output Formats 36

5 EXPERIMENTS AND RESULTS 37
5.1 Weka 37
5.2 Preparing the Test Data 38
5.2.1 Label Sets and Validation 38
5.2.2 Score Calculation 39
5.2.3 Conventions 40
5.3 Classifiers 40
5.3.1 Standard Classifiers 40
5.3.2 Classifiers with Bagging 45
5.4 Final Evaluation 47

6 CONCLUSIONS 49

REFERENCES 50

LIST OF ABBREVIATIONS AND ACRONYMS

AIR	Adversarial Information Retrieval
GUI	Graphical User Interface
IR	Information Retrieval
ITR	Inverted TrustRank
ML	Machine Learning
PR	PageRank
TPR	Truncated PageRank
TR	TrustRank

LIST OF SYMBOLS

ω	damping factor
A_{NxN}	adjacency matrix in a Web graph
C	normalization constant
E	set of edges in a Web graph
$F1$	F-measure
FN	False negatives
FP	False positives
G	Web graph
N	number of pages in a Web graph
P	Precision
R	Recall
S	TrustRank's seed
S^+	subset of good pages in S
V	set of pages in a Web graph
U	row-normalized version of A
$branching(c)$	branching contribution of a path c
$damping(t)$	decreasing function on t
$i(p)$	number of in-links of a certain page p
$o(p)$	number of out-links of a certain page p

LIST OF FIGURES

Figure 3.1: PageRank calculation example 18
Figure 3.2: Normal link structure of a group of Web pages 20
Figure 3.3: Link farm 20
Figure 3.4: Web graph example - good pages in white; bad pages in black 21

Figure 4.1: Web graph example - same of figure 3.4 25
Figure 4.2: PageRank *versus* Truncated PageRank with T = 1 29
Figure 4.3: PageRank *versus* Truncated PageRank with T = 4 30

Figure 5.1: Precision and Recall of the classifiers 48
Figure 5.2: Error rates of the classifiers 48

LIST OF TABLES

2.1 Confusion matrix for a given *spamdexing* detection technique 16

4.1 Evaluation measures for TR based classifier 27

5.1 Confusion matrix for TPR-2 based classifier 41
5.2 Evaluation measures for TPR-2 based classifier 41
5.3 Confusion matrix for TPR-3 based classifier 41
5.4 Evaluation measures for TPR-3 based classifier 41
5.5 Confusion matrix for TPR-4 based classifier 41
5.6 Evaluation measures for TPR-4 based classifier 42
5.7 Confusion matrix for TPR-X based classifier 42
5.8 Evaluation measures for TPR-X based classifier 42
5.9 Confusion matrix for TR based classifier 42
5.10 Evaluation measures for TR based classifier 43
5.11 Confusion matrix for ITR based classifier 43
5.12 Evaluation measures for ITR based classifier 43
5.13 Confusion matrix for TR & ITR based classifier 43
5.14 Evaluation measures for TR & ITR based classifier 43
5.15 Confusion matrix the Combined classifier 44
5.16 Evaluation measures for the Combined classifier 44
5.17 Confusion matrix the Combined classifier with extra features 44
5.18 Evaluation measures for the Combined classifier with extra features . 45
5.19 TPR-2: with bagging *versus* without bagging 45
5.20 TPR-3: with bagging *versus* without bagging 45
5.21 TPR-4: with bagging *versus* without bagging 45
5.22 TPR-X: with bagging *versus* without bagging 46
5.23 TR: with bagging *versus* without bagging 46
5.24 ITR: with bagging *versus* without bagging 46
5.25 TR & ITR: with bagging *versus* without bagging 46
5.26 Combined: with bagging *versus* without bagging 46
5.27 Combined with extra features: with bagging *versus* without bagging . 47
5.28 Final comparison between classifiers 47

ABSTRACT

Spamdexing techniques have "haunted" search engines for more than a decade now, and are still an issue nowadays. Many content-based techniques to detect such methods were already proposed in the literature, but the research on link-based spamdexing detection techniques is recent. This work focuses on the study and implementation of two such link-based techniques, measuring their quality with well-known Information Retrieval metrics, and comparing their results. The obtained results show that even though the problem cannot be a hundred percent eliminated yet, a lot may already be done to improve search engines response to user submitted queries.

Keywords: Spamdexing, link analysis, adversarial information retrieval, evaluation.

Aplicando Técnicas de Detecção de Spamdexing Baseadas em Links

RESUMO

Técnicas de *spamdexing* têm "assombrado"os motores de busca por mais de uma década e ainda são um problema hoje em dia. Muitas técnicas baseadas em conteúdo para detectar esses métodos já foram propostas na literatura, mas a pesquisa sobre técnicas de detecção de *spamdexing* baseadas em *links* é recente. Este trabalho tem como foco o estudo e implementação de duas dessas técnicas baseadas em *links*, medindo suas qualidades com métricas de Recuperação de Informações bem conhecidas, e comparando seus resultados. Os resultados obtidos mostram que, embora o problema ainda não possa ser cem por cento eliminado, muito já pode ser feito para melhorar a resposta dos motores de busca para as consultas submetidas por usuários.

Palavras-chave: Spamdexing, análise de links, recuperação de informações contraditórias, avaliação.

1 INTRODUCTION

In recent years, the term *spam* has been associated with unwanted (and usually of commercial nature) messages sent out in bulks. Although the most common and known form of *spam* is simply junk e-mail, there are many other forms of spam, such as credit card seller operators. The focus of this work is to deal with another kind of *spam*, one that has slightly "disappeared" (from the point of view of the users) some years ago, but that now, with different and evolved techniques, is forcing search engines to re-adapt once more: *spamdexing*.

Also known as *Web spam*, the first citation of the term *spamdexing* was made by Eric Convey, in his article *"Porn sneaks way back on Web"* (The Boston Herald, May 1996), in which he wrote: *"the problem arises when site operators load their Web pages with hundreds of extraneous terms so search engines will list them among legitimate addresses"*. The term *"spamdexing"* is a combination of the words *"spam"* and *"indexing"*.

The definition above serves as a characterization of the first large class of *spamdexing* techniques: *content spam* (a.k.a. *term spam*). This class of *spamdexing* was the main cause for making the most popular 1990's search engines unreliable. That is because, at that time, search engines used to base their ranking algorithms solely on the content of the Web sites they indexed, matching the content of the pages with the queries submitted by their users.

Google's success came with their ability to combat this kind of *spamdexing* techniques by using a ranking algorithm based on the link structure between the pages on the Web: the famous PageRank (PR) (PAGE et al., 1998). Even though Google did not become unreliable as its ancestors, it has not remained immune to more sophisticated methods of *spamdexing*.

The second large class of *spamdexing* techniques is known as *link spam*. *Spam farming* and *Google bombing* are examples of new ways of manipulating results, which involve adding hyperlinks that affect the ranking of Web sites, making use of the properties of algorithms such as Google's PR.

With these two classes of *spamdexing* technique in mind, a more general definition of the term is given in (GYONGYI; GARCIA-MOLINA, 2005): *"any deliberate action that is meant to trigger an unjustifiably favorable relevance or importance for some Web page, considering the page's true value"*. That is, pages whose creators had the evident intention of misleading the results of search engines, showing the users unsolicited content (*spam content*) among the contents they had requested, or even preventing them from finding what they were looking for.

A lot of research has been made in the search of content-based techniques for detecting *spam pages*, such as (DAVISON, 2000), and (NTOULAS et al., 2006), by studying relevant features such as page size or keyword distribution. However, the use of link-based techniques for detecting *spam pages* is a relatively new area of research. The leading studies emerged only in 2004 and were reported in (GYONGYI; GARCIA-MOLINA; PEDERSEN, 2004), and in (BECCHETTI et al., 2006).

After a preliminary analysis of the current state of the art in link-based techniques for detecting *spam pages*, this work focuses on studying the two references cited above, since they are the most well-known and well-documented works on the subject. The proposed techniques were implemented, and evaluated, so that an analysis of the quality of the results given by the current state of art can be presented.

Indexing and retrieving information from collections that have been maliciously manipulated is known in the literature as Adversarial Information Retrieval (AIR). This area is dedicated to detecting and combating such manipulation.

1.1 Goals

The goals of this work are:

a) to study two different link-based *spamdexing* detection techniques already proposed in the literature;

b) to analyze and implement the studied techniques;

c) to evaluate and compare the results of the implemented techniques;

d) to combine the results of the techniques and analyze such a combination.

1.2 Organization

The remainder of this work is organized as follows:

a) Chapter 2 contains the first part of a literature review, presenting a model to view the Web as a graph, and the evaluation measures used in this work;

b) Chapter 3 follows the second Chapter as a second part of a literature review, showing the main algorithms used during this work;

c) Chapter 4 explains the implementation details of each studied technique and the dataset used;

d) Chapter 5 defines the analysis tool and presents the experiments and results. Comparisons are made in order to evaluate the significance of each result;

e) Chapter 6 reports the final conclusions of this work.

2 WEB GRAPH, AND EVALUATION MEASURES

In this Chapter, a model to describe the Web as a graph is explained, as well as the measures used to evaluate each of the link-based *spamdexing* detection techniques.

2.1 Web Graph

A Web graph is the graph representation of any portion of the (or of the whole) Web, where each node represents a page, and the hyperlinks between pages are the edges. Formally, we have a graph $G = (V, E)$ consisting of a set V of N pages and a set E of directed links (edges) that connect pages. In practice, self hyperlinks are removed, and, even though a page p may have multiple hyperlinks to a page q, they are usually collapsed into a single link $(p, q) \in E$.

Two functions, $i(p)$ and $o(p)$, may also be described as the number of in-links and out-links of a certain page p. Furthermore, this graph may also be represented as an adjacency matrix A_{NxN}, where $a_{i,j} = 1$ if and only if there is a link from page i to page j.

It is also important to notice that this model carries through to the case where graph vertices are entire sites (host graphs). In this case, all the hyperlinks between two hosts are merged, and possibly weighted - although such weighting system is not implemented in this work.

2.2 Evaluation Measures

The evaluation of the techniques presented in this work are done based on a set of measures commonly used in Machine Learning (ML) and Information Retrivel (IR) (BAEZA-YATES; RIBEIRO-NETO et al., 1999), considering the spam detection task. For a given technique, we consider its confusion matrix:

		Prediction	
		Non-spam	Spam
True Label	Non-spam	w	x
	Spam	y	z

Table 2.1: Confusion matrix for a given *spamdexing* detection technique

where each letter represents:

- w: the non-spam examples that were correctly classified;
- x: the non-spam examples that were wrongly classified as spam;
- y: the spam examples that were wrongly classified as non-spam;
- z: the spam examples that were correctly classified.

Two types of errors of the spam classification are then measured:

$$\text{False positives: } FP = \frac{x}{x+w}$$

$$\text{False negatives: } FN = \frac{y}{y+z}$$

As well as the following IR evaluation measures:

$$\text{Precision: } P = \frac{z}{x+z}$$

$$\text{Recall, or true positive rate: } R = \frac{z}{y+z}$$

And finally:

$$\text{F-measure: } F1 = 2\frac{P*R}{P+R}$$

where the F-measure $F1$ consists of a way of summarizing both precision and recall, so that it can be used as the primary measure of comparison between *spamdexing* detection techniques.

3 RANKING ALGORITHMS

The general idea behind the studies in (BECCHETTI et al., 2006), and in (GYONGYI; GARCIA-MOLINA; PEDERSEN, 2004), is to create an algorithm which ranks the pages on a Web graph, according to some properties. After, a classifier is built, which uses the results from these algorithms along with a small set of human-labeled hosts[1] to generate parameters for an automatic classification. This process is presented in a more detailed way in Chapters 4 and 5.

The two ranking algorithms are called: Truncated PageRank (TPR) (BECCHETTI et al., 2006) and TrustRank (TR) (GYONGYI; GARCIA-MOLINA; PEDERSEN, 2004). Since the two may be implemented as slightly modified versions of the best known PageRank, PR is the first ranking algorithm presented.

3.1 PageRank

As stated in the first Chapter, before the idealization of PageRank by Larry Page and Sergey Brin (Google founders) (PAGE et al., 1998), the main search engines then used to calculate the importance of a page by looking at how many words matched those entered by a user search query. That is not completely true, because some of them already used link counting whilst trying to bias their databases in favor of some pages, which would probably happen to be the most important pages. The idea was the same as in classic IR systems (based primarily on the study of retrieving information from scientific work): counting the number of links that pointed to a page (a.k.a. the in-links of this page) should suggest its importance, much like the number of citations to an article usually does.

The problem is that citation counting on the Web does not work as it used to work on academic citation databases, since the Web has characteristics of its own. For example, if a page p is being pointed to by NASA[2], it should receive a higher rank than a page q that is linked by many hobbyist sites about space traveling. Google's PR is based on this simple idea: that we should not only count the number of in-links a certain page has, but also consider the importance of each link that is pointing to it.

As in (PAGE et al., 1998), the intuitive description for PageRank is: *"a page has high rank if the sum of the ranks of its backlinks is high. This covers both the case when a page has many backlinks* - or in-links - *and when a page has a few highly ranked backlinks"*.

[1] Hosts labeled as spam or non-spam. Both the dataset and the tool used to build such classifiers are shown in Chapter 4 and 5, respectively.

[2] http://www.nasa.gov

Let ω be a damping factor between 0 and 1 - typically 0.85, as suggested in the original PR's proposal -, used for normalization, the above description can be expressed by the following formula:

$$r(p) = \omega \sum_{q;(q,p)\in E} \frac{r(q)}{o(q)}$$

That is, the rank of a page p is based on a percentage of the rank of all pages that point to it. If done this way, a page that is highly linked will receive a lot of "importance", and a page with lots of out-links will distribute smaller portions of its rank. This focuses on the idea that, usually, important pages do not point to many pages, but are pointed by a high number of pages. Figure 3.1 exemplifies such ranking system.

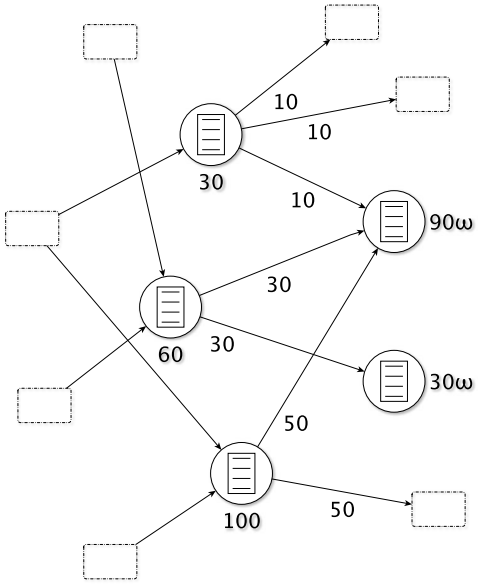

Figure 3.1: PageRank calculation example

Actually, a more complete version of PR's formula may be described as the following:

$$r(p) = \omega \sum_{q;(q,p)\in E} \frac{r(q)}{o(q)} + (1-\omega)\frac{1}{N}$$

The equation is recursive, so that there is mutual reinforcement between pages, where the importance of a certain page influences and is influenced by the importance of some other pages. It may be computed by starting with any set of ranks and iterating the computation until it converges. What usually happens is that the ranks are uniformly initialized ($\frac{1}{N}$), and instead of waiting until it converges, the number of iterations is fixed.

For the formula above, the equivalent matrix equation form is:

$$r = \omega U r + (1-\omega)\frac{1}{N}1_N$$

where U is the row-normalized version of the adjacency matrix A_{NxN}, such that all rows add up to one, and rows of zeros are replaced by rows of $\frac{1}{N}$ to avoid the effect of

rank "sinks". Pages with no out-links are called "sinks" because their accumulated rank is usually wasted.

While the first component of the score of a certain page p comes from pages that point to it, the second component is equal for all web pages. This part of the equation, as pointed by Page and Brin in the original PageRank paper, is called *"the random surfer model"*, in other words, the probability of a certain Web surfer to stop following links and redirecting to any other page on the Web. Also in this part, it is possible to substitute the $\frac{1}{N}1_N$ for a vector d: a static scored distribution vector, of arbitrary, non-negative entries adding up to one, that is, a way to bias the ranking of a pre-selected group of pages. This will also cause this "extra score" to be spread for the pages this pre-selected group of pages points to.

3.2 Functional Ranking

Given a path, that is, a sequence of links in a Web graph $u = \{p_1, p_2, ..., p_k\}$, such that page p_i has a link to page p_{i+1}, a branching contribution is as follows:

$$branching(u) = \frac{1}{o(p_1)o(p_2)...o(p_{k-1})}$$

A functional ranking (BAEZA-YATES; BOLDI; CASTILLO, 2006) is a link-based ranking algorithm formalized as:

$$W_i = \sum_{u \in Path(-,i)} \frac{damping(|u|)}{N} branching(u)$$

where $Path(-,i)$ is the set of all paths into page i, $|u|$ is the length of path u, and $damping(|u|)$ is a decreasing function on $|u|$.

Equivalently, we have:

$$W = \sum_{t=0}^{\infty} damping(t) \frac{1}{N} 1_N U^t$$

where $(U^t)_{i,j}$ contains the sum of the branching contributions of all paths of length t from i to j. In particular, PR's $damping(t) = (1-\omega)\omega^t$, as also explained in (BAEZA-YATES; BOLDI; CASTILLO, 2006), where they also point *"This way of expressing the PageRank of a node is essentially obtained as a weighted sum of contributions coming from every path entering into the node, with weights that decay exponentially in the length of the path"*. This also gives as us an important starting point for the first of the two ranking algorithms used in this work: the Truncated PageRank.

3.3 Truncated PageRank

The Truncated PageRank algorithm is a slightly modified version of the PageRank algorithm with the intention of avoiding the impact of a special kind of link-based structure

commonly used by spammers to mislead link-based ranking algorithms such as PR: link farms - densely connected sets of pages. In (ZHANG et al., 2004), this is called *"collusion"*, and is defined as the *"manipulation of the link structure by a group of users with the intent of improving the rating of one or more users in the group"* .

For a better understanding of what a link farm actually is, a comparison between a normal link structure and a link farm is made in Figures 3.2 and 3.3.

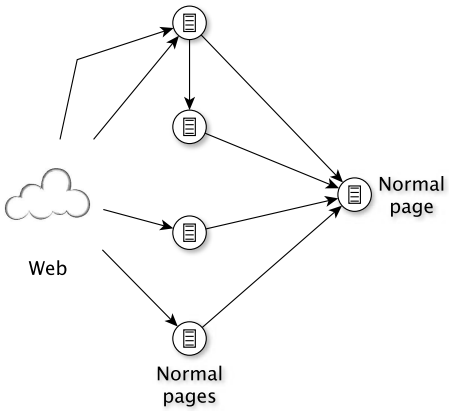

Figure 3.2: Normal link structure of a group of Web pages

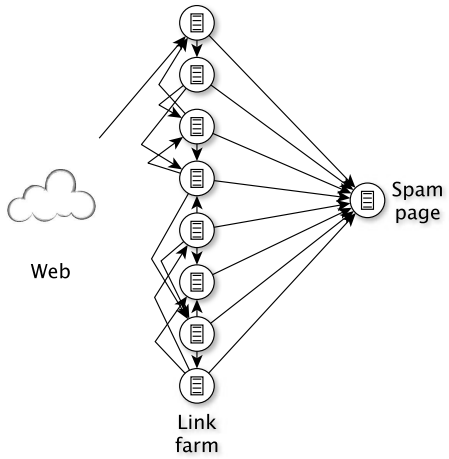

Figure 3.3: Link farm

"Link farms can receive links by buying advertising, or by buying expired domains used previously by legitimate Web sites" (BECCHETTI et al., 2006).

Studies presented in the original paper, based on the calculation of the PageRank of pages from a $spamdexing - free$ portion of the Web, have shown that, normally, highly-ranked pages have a large number of supporters from a farther distance in the Web graph, while low-ranked pages do not. Spam pages participating in link farms will not follow this rule, since they still have a high rank, even though they share little relationship with the rest of the Web graph.

Based on these ideas, the TPR algorithm does what its name already suggests: it truncates the normal PageRank algorithm by ignoring the contribution of the first levels of links in the paths that lead to the pages, so that pages participating in link farms (that is, that have a high PageRank value because of their nearest neighbors) "lose their value", actually making them assume their real value.

Using the functional ranking version of the PageRank, we can achieve the expected behavior by adjusting its $damping(t)$ function, as follows:

$$damping(t) = \begin{cases} 0 & t \leqslant T \\ C\omega^t & t > T \end{cases}$$

where T is the length of the truncated path, C is a normalization constant, such that $\sum_{t=0}^{\infty} damping(t) = 1$ - so $C = \frac{1-\omega}{\omega^{T+1}}$ -, and ω is the damping factor used for PageRank.

3.4 TrustRank

The assumption underlying TrustRank (GYONGYI; GARCIA-MOLINA; PEDERSEN, 2004) is what the authors of the original paper call *"approximate isolation of the good set"*, which means that trustful pages tend to point exclusively to other genuine pages.

Lets then call the non-spam pages "good" and the spam pages "bad". Given the set of pages of the Web graph from Figure 3.4, where the good pages are in white and the bad pages in black, lets suppose we do not know the quality of all of those pages, just from a small portion of them. A sample of pages, called "seed", which was previously labeled by humans as "spam" or "non-spam".

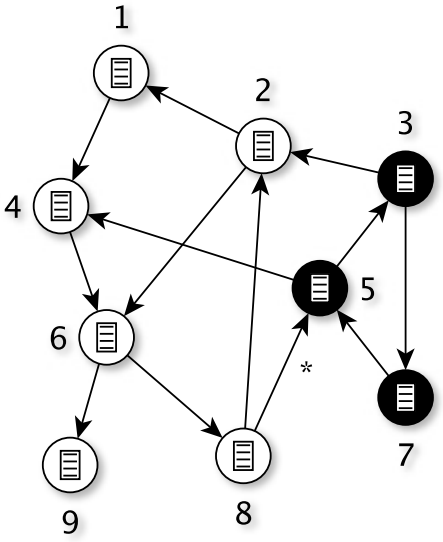

Figure 3.4: Web graph example - good pages in white; bad pages in black

Then, the first attempt to compute Trust with a seed is through an Ignorant Trust Function. For any given page p in our Web graph, where S denotes our seed set:

$$T_0(p) = \begin{cases} s(p) & \text{if } p \in S \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

And:

$$s(p) = \begin{cases} 1 & \text{if } p \text{ was labeled as "non-spam"} \\ 0 & \text{if } p \text{ was labeled as "spam"} \end{cases}$$

With $S = \{1,2,7\}$, we should have:

$$t_0 = \left\{1, 1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}\right\}$$

The next step is to propagate these Trust scores by applying a K-Step Trust Function:

$$T_K(p) = \begin{cases} s(p) & \text{if } p \in S \\ 1 & \text{if } p \notin S \text{ and } \exists q \in S^+ : Path(q,p) \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

where S^+ denotes the subset of good pages in S, and $Path(q,p)$ denotes the existence of a path between pages q and p.

For $K = 3$, the same seed S, and the same Web graph, then we have:

$$t_1 = \left\{1, 1, \frac{1}{2}, 1, \frac{1}{2}, 1, 0, \frac{1}{2}, \frac{1}{2}\right\}$$

$$t_2 = \left\{1, 1, \frac{1}{2}, 1, \frac{1}{2}, 1, 0, 1, 1\right\}$$

$$t_3 = \left\{1, 1, \frac{1}{2}, 1, 1, 1, 0, 1, 1\right\}$$

Note that, until the second step, all the pages were correctly assigned, but in the third step, page 5 receives a score of 1 due to the link from good page 8 to bad page 5 (marked with an asterisk on Figure 3.4). Since bad pages are created to mislead search engines, there is little or no reason for a good page to point towards a bad page. But, sometimes, they do.

The authors of good pages may be "tricked" into pointing to bad pages. The most simple example is the creator of a bad page adding a link to that page in a well-known forum on the Web. Another example is the creation of a "honey pot", which is a set of pages that provide some useful information along with hidden links to spam pages. The "honey pot" can then attract people to point to these pages, helping them boost the rank of spam pages.

To solve this problem, a Trust attenuation is applied, that is, a reduction of Trust as we move further and further away from the good seed pages on the Web graph (as $|Path(q,p)|$ becomes higher). This attenuation may be done in different ways, but the authors of the algorithm suggest the use of damping, splitting, or both. By using damping, at each level, instead of propagating 1's, we will be propagating $goodScore * \beta$, where β is a damping factor, so, we will be actually propagating β^t, where $t = |Path(q,p)|$. By using splitting, at each level, instead of propagating 1's, we will be propagating $\frac{goodScore}{o(p)}$, where p is the current page.

TrustRank may be easily implemented by using PageRank as its basis. The only modification is to use the specialization vector d instead of a vector 1_N, where the values are assigned according to the values in the seed S, with the values that are unknown being assigned to 0. We also use this vector to initialize our vector of ranks, and, other than this, we just need to multiply the vector by $\frac{1}{N^+}$ instead of $\frac{1}{N}$, as it is in the original PageRank formula, where N^+ corresponds to the number of good pages in the seed. So, the matrix equation for TrustRank is the following:

$$r = \omega U r + (1-\omega)\frac{1}{N^+}d$$

where the values of the vector d are as described above. This way, at each iteration, the Trust score of a page is split among its neighbors, and dampened by the same factor as the PageRank. The only main difference, though, is that this U does not have its rows of zeros replaced by rows of $\frac{1}{N}$, since the TrustRank calculation does not need to avoid the effect of rank "sinks": we do not want the trustfulness of a page being propagated to every other page in the graph.

The functional ranking form of TrustRank should then look like this:

$$W = \sum_{t=0}^{\infty} damping(t)\frac{1}{N^+}dU^t$$

After this brief explanation of the fundamental ideas underlying each of the studied algorithms, Chapter 4 follows with the implementation details of each of them. The use of the scores produced by such algorithms to detect spamdexing is presented in Chapter 5.

3.5 Inverted TrustRank

An alternative for the original TrustRank would be its inverted calculation, which is here named Inverted TrustRank (ITR). It follows the same idea underlying the TR, but instead of propagating the goodness of a page forwards, the badness of a page is propagated backwards.

The assumption is exactly the opposite of the one used by TrustRank, that is, the *"approximate isolation of the bad set"*, which means that pages that may not be trusted tend to be pointed exclusively by other non-genuine pages.

4 IMPLEMENTATION DETAILS

In this Chapter, the implementations of the Truncated PageRank and TrustRank are shown, as well as the implementation of a variation of the "classic" TrustRank, the Inverted TrustRank, and of a series of algorithms that calculate different attributes presented in the pages of a Web graph. The necessary tools and data to enable such implementations are also detailed.

All implementations were done in the Java programming language, using the Truncated PageRank implementation of its original authors[1] as the base source code. This code has suffered only minor updates in this work, but its functionality is explained anyway, for the sake of completeness.

4.1 WebGraph Framework

WebGraph is a framework made to assist in the study and research of large graphs, like the ones presented on the Web: *"It provides simple ways to manage very large graphs, exploiting modern compression techniques"*[2]. It is totally implemented in the Java programming language[3], and a complete documentation also accompanies the package.

The graph available with the dataset that is presented in Section 4.2 is in the BVGraph compressed format, which is part of the WebGraph framework, and further explained in (BOLDI; VIGNA, 2004). Also, the framework provides the ASCIIGraph class, which allows us to convert a graph in text format to the BVGraph compressed format.

An example of how to use the ASCIIGraph class is presented with the use of the graph shown in Figure 3.4, which is now repeated in Figure 4.1 to facilitate the understanding of the procedure. Note that now the labels on the nodes range from 0 to 8, instead of 1 to 9 as in Figure 3.4.

[1] Available under http://213.27.241.151/webspam/code

[2] http://webgraph.dsi.unimi.it

[3] A version in C++ is also available, but, by the time this work was done, it was considered still in alpha stage.

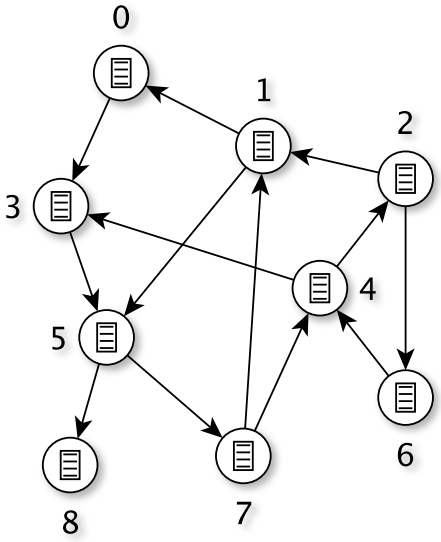

Figure 4.1: Web graph example - same of figure 3.4

A file named *"example.graph-txt"* to represent the graph from Figure 4.1 would look like the following:

```
9
3
0 5
1 6
5
2
7 8
4
1 4
_
```

where:

- the first line indicates the number of nodes in the graph;
- the second line represents the out-links of node 0;
- the third line represents the out-links of node 1, and so on;
- nodes with no out-links are represented with blank lines (the underscore character in the example above).

Also, an important observation is that the out-links list must be sorted in numerical order for a better compression rate when converting to the BVGraph compressed format.

After creating such file, all one needs to do is use the following command (considering that the libraries of the framework have been properly installed):

```
java it.unimi.dsi.webgraph.BVGraph -g ASCIIGraph example bvexample
```

The outputted file, in the BVGraph compressed format, is then named *"bvexample.graph"*. Along with it, other two files are created: *"bvexample.offsets"* and *"bvexample.properties"*.

4.2 Dataset

The dataset used in this work is the *WEBSPAM-UK2006* collection[4] detailed in (CASTILLO et al., 2006). As the title of the original paper suggests, it is a reference collection for the testing of *spamdexing* detection techniques.

It is a snapshot of the *.uk* domain pages that was made starting in May 2006. *"These pages were downloaded at the Laboratory of Web Algorithmics*[5] *at the Università degli Studi di Milano. The crawl was done using the UbiCrawler (BOLDI et al., 2004) in breadth-first-search mode for cross-host links (depth-first exploration was adopted for local links), starting from a large seed of over 190,000 URLs in about 150,000 hosts under the .uk domain listed in the Open Directory Project*[6]*. The crawler was limited to the .uk domain and to 8 levels of depth, with no more than 50,000 pages per host: these restrictions were such that indeed only a part of the hosts contained in the seed were actually crawled"* (CASTILLO et al., 2006). With such procedure, the collection includes 77.9 million pages from 11,402 hosts, and over $3 * 10^9$ edges.

The available files are:

- *"uk-2006.05.graph.gz"* (734MB, expands to 1GB): the graph itself in the BVGraph compressed format;
- *"uk-2006.05.offsets.gz"* (67MB): the offsets of the graph in the BVGraph compressed format;
- *"uk-2006.05.properties"* (1KB): the properties of the graph in the BVGraph compressed format;
- *"uk-2006.05.urls.gz"* (572MB): contains one URL per line, sorted lexicographically (which is also the same node access order when using the graph with the WebGraph framework - the first URL is identified with the number 0);
- *"uk-2006-05.graphtxt.gz"* (1.5GB, expands to >10GB): the graph in text format, as exemplified in Section 4.1;
- *"uk-2006-05.hostnames.txt.gz"* (97KB): a text format graph where each line is formatted as follows: "$src -> dest_1 : nlinks_1 dest_2 : nlinks_2, ..., dest_k : nlinks_k$" in which *src* is the source host, *dest* is the destination host, and *nlinks* the number of page-to-page links between the two hosts.
- *"uk-2006-05.hostnames.txt.gz"* (1.9MB): just like the *"uk-2006.05.urls.gz"* file, but only with the host pages included.

[4] Available under http://213.27.241.151/webspam/datasets/uk2006
[5] http://law.dsi.unimi.it
[6] http://www.dmoz.org

The content of the pages is not available for download, but may be requested with a signed agreement. Since the algorithms studied in this work make use only of the link structure of the Web graph for their calculations, such request was not necessary.

Using the "bow-tie" model of the Web (BRODER et al., 2000), the components of the host graph of the *WEBSPAM-UK2006* dataset are the following:

Region	CORE	IN	OUT	TENDRILS	DISC	Total
Size	7945	135	3,100	64	165	11,402

Table 4.1: Evaluation measures for TR based classifier

Which shows that most of the pages contained in the dataset are part of the strongly connected components set (CORE) of the graph they are in (no matter if it is the Web graph or the host graph), and, also, that a big portion of them is on the OUT set. The OUT set is formed by the pages that are linked by the CORE set, but do not link to any page in the CORE set (which does not mean that they do not have links between each other). These characteristics show that this collection may indeed serve as a really good dataset for testing purposes.

4.3 Truncated PageRank

In this Section, the implementation details for the Truncated PageRank algorithm are presented. For a better comprehension, its pseudocode is also shown.

This work used the TPR implementation of its original authors[7] with minor updates. The original version already used the WebGraph framework to access the graph data, but since then, updates to the framework caused the original source code to not work anymore, which has been corrected in this work. Nevertheless, the original algorithm remains the same.

The algorithm starts by testing whether or not we are truncating the PageRank scores. Note that if this algorithm is executed with truncation $T = 0$, the normal PR is calculated. With this test, the normalization constant c is assigned to be either the one used by the PR or the one used by its truncated version.

Afterwards, the R and *Score* vectors are initialized. R is an auxiliary accumulator which keeps the contribution of a certain iteration to the rank of each page. It is important to keep both *Score* and R separated in the calculation, since the first levels are discarded, or we may end up with only zeros in the output. All values of the vector R are initialized to $\frac{C}{N}$, where N is the total number of pages in the graph. If we are to calculate the normal PageRank, then *Score* is initialized with the same values as R.

iteration is initialized to 1, and, until the scores converge, the algorithm is executed (although, usually, the number of iterations is fixed). The *Aux* vector is set to contain only 0's and *dangling_score* is also initialized to 0. The *dangling_score* variable is responsible for accumulating the scores from "sinks".

Then, for every page in the graph, if the page has no out-links, its R is divided by N,

[7] Available under http://213.27.241.151/webspam/code

the total number of pages in the graph, and added to the *dangling_score* accumulator. If the page has out-links, then every out-neighbor of this page receives a portion of its R (which corresponds to its R divided by the number of out-links this page has - or simply put, its out-degree).

Finally, the damping factor is applied, and R is updated with a new value. Then, if there is no need to truncate anymore, R is added to *Score*.

Truncated PageRank pseudocode:

Require: N: number of nodes, $0 < \omega < 1$: damping factor, $T \geqslant 0$: distance of truncation

```
1   if T > 0 then
2       C = (1 − ω) / (ω^(T+1))
3   else
4       C = (1 − ω)
5   end if
6   for j : 1 ... N do //Initialization
7       R[j] = C / N
8       if T > 0 then
9           Score[j] = 0
10      else //Calculate normal PageRank
11          Score[j] = R[j]
12      end if
13  end for
14  iteration = 1
15  while not converged do
16      Aux = 0
17      dangling_score = 0
18      for src : 1 ... N do //Follow links in the graph
19          if src has no links then
20              dangling_score = dangling_score + (R[src] / N)
21          end if
22          for all link from src to dest do
23              Aux[dest] = Aux[dest] + R[src] / o(src)
24          end for
25      end for
26      for j : 1 ... N do //Apply damping factor ω
27          R[j] = (Aux[j] + dangling_score) * ω
28          if iteration > T then //Add to ranking value
29              Score[j] = Score[j] + R[j]
30          end if
31      end for
32      iteration = iteration + 1
33  end while
34  return Score
```

Formally:

$$R^{(0)} = \frac{C}{N}$$

$$R^{(t)} = \omega R^{(t-1)} U$$

And the Truncated PageRank is calculated using:

$$W = \sum_{t=T+1}^{\infty} R^{(t)}$$

The complexity of the implementation above is analyzed as follows. The initialization part, from line 1 to line 14, has a cost of $O(N)$, where N is the number of pages in the graph. Lines 16-32 has a cost of $O(N+M)$, where M is the number of links. Thus, the Truncated PageRank implemented has complexity $O(K(N+M))$, where K is the number of iterations of the algorithm that depends on the stopping criterium of the algorithm. In our implementation the stopping criterium is a given number of iterations.

The complexity of such implementation may be easy calculated. The initialization part, from line 1 to line 14, has a cost of $O(N)$, where N is the number of pages in the graph. As of the rest of the code, the complexity is $O(k*(N+M))$, where k is the number of needed iterations, and M is the number of links in the graph.

Let us now calculate the PageRank and the Truncated PageRank - with truncations $T = 1$, and $T = 4$ - of the pages presented in the host graph of the *WEBSPAM-UK2006* dataset. The relation of both measures is shown in Figures 4.2, and 4.3.

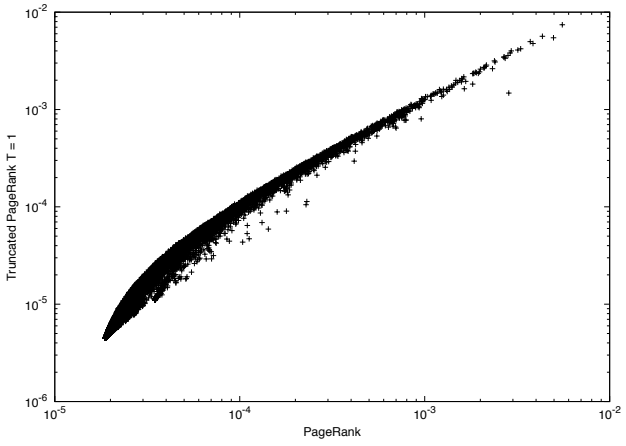

Figure 4.2: PageRank *versus* Truncated PageRank with T = 1

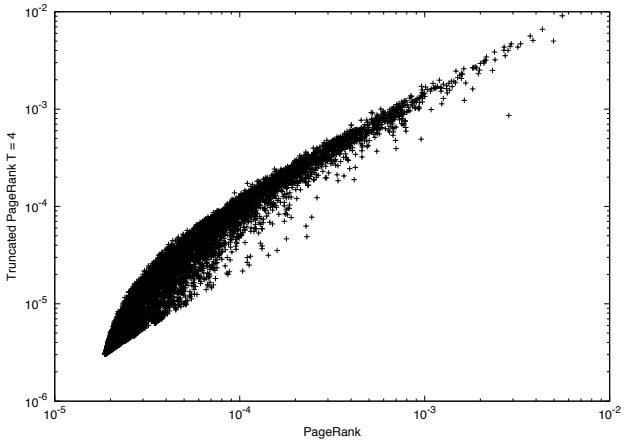

Figure 4.3: PageRank *versus* Truncated PageRank with T = 4

It is possible to notice that, as expected, both measures are highly correlated, and that this correlation decreases as more levels are truncated. That is, if $T = 0$ was used instead, something near to a line would be seen in the graphics above, and, as we truncate more and more the PageRank calculation, the TPR values of the pages (with a certain PR value) begin to decrease.

4.4 TrustRank

This Section focuses on explaining the implementation of both TrustRank and Inverted TrustRank. For differentiation, the first is referred as the "classic" TrustRank.

4.4.1 Classic TrustRank

Considering the functional ranking form of the TrustRank, it is possible to use the TPR implementation as the basis for a TR implementation. The details of such implementation are presented in this Subsection, along with its pseudocode.

First of all, the normalization constant C is initialized in the same way as the PageRank. R is an auxiliary accumulator which keeps the contribution of a certain iteration to the rank of each page, and *Score* will contain the final score of each page. The values of both these vectors are initialized with $\frac{C}{N^+}$, where N^+ is the number of good pages in the seed, if the corresponding page is a good page in the seed, and with 0 if it is not.

As explained in Chapter 3, what follows is a normal PR calculation. $iteration$ is initialized to 1, and, until the scores converge, the algorithm is executed (although, usually, the number of iterations is fixed) like in the Truncated PageRank implementation. The *Aux* vector is also set to contain only 0's.

Then, for every page in the graph, if it has out-links, every out-neighbor of this page receives a portion of its R (which corresponds to its R divided by the number of out-links this page has - or simply put, its out-degree). The damping factor is applied, and R is

updated with a new value, which is then added to *Score*.

Classic TrustRank pseudocode:

Require: N: number of nodes, $0 < \omega < 1$: damping factor, S^+: subset of the seed containing only the good pages, N^+: number of good pages in the seed

```
1   C = (1 − ω)
2   for j : 1 ... N do //Initialization
3       if j ∈ S+ then
4           R[j] = C / N+
5       else
6           R[j] = 0
7       end if
8       Score[j] = R[j]
9   end for
10  while not converged do
11      Aux = 0
12      for src : 1 ... N do //Follow links in the graph
13          for all link from src to dest do
14              Aux[dest] = Aux[dest] + R[src] / o(src)
15          end for
16      end for
17      for j : 1 ... N do
18          //Apply damping factor ω
19          R[j] = Aux[j] * ω
20          //Add to ranking value
21          Score[j] = Score[j] + R[j]
22          end if
23      end for
24  end while
25  return Score
```

The complexity of such implementation is the same of the TPR's implementation: $O(K(N+M))$, where K is the number of iterations, N is the number of pages in the graph, and M is the number of links in the graph.

4.4.2 Inverted TrustRank

The algorithm that is detailed now is quite the same as the TrustRank's, but "inverted". Its pseudocode is also provided.

Instead of repeating the same explanation given in Subsection 4.4.1, lets focus on the differences between the TR and the ITR. The first one is that the values of vectors R and *Score* are initialized with $\frac{C}{N^-}$, where N^- is the number of bad pages in the seed, if the corresponding page is a bad page in the seed, and with 0 if it is not.

Furthermore, in the step responsible for following every link in the graph, instead of considering if a page has out-links, the in-links are considered. Then, for every page in the graph, if it has in-links, every in-neighbor of this page receives a portion of its R (which corresponds to its R divided by the number of in-links this page has - or simply put, its in-degree).

Inverted TrustRank pseudocode:

Require: N: number of nodes, $0 < \omega < 1$: damping factor, S^-: subset of the seed containing only the bad pages, N^-: number of bad pages in the seed

```
1   C = (1 − ω)
2   for j : 1 ... N do //Initialization
3       if j ∈ S⁻ then
4           R[j] = C / N⁻
5       else
6           R[j] = 0
7       end if
8       Score[j] = R[j]
9   end for
10  while not converged do
11      Aux = 0
12      for dest : 1 ... N do //Follow links in the graph
13          for all link to dest from src do
14              Aux[src] = Aux[src] + R[dest] / i(dest)
15          end for
16      end for
17      for j : 1 ... N do
18          //Apply damping factor ω
19          R[j] = Aux[j] * ω
20          //Add to ranking value
21          Score[j] = Score[j] + R[j]
22          end if
23      end for
24  end while
25  return Score
```

With the same complexity of both Truncated PageRank and TrustRank.

4.5 Extra Attributes

Along with the algorithms explained in Chapter 3, five other page-level attributes were computed. These extra attributes were calculated as an attempt to make the results from the experiments described in Chapter 5 become even better. They are:

- Assortativity: the coefficient of the degree of a page divided by the average degree of both its in-neighbors and out-neighbors. Degree in this case is undirected (for a page p, $degree(p) = i(p) + o(p)$);
- Average in-links in out-neighbors: for a page p, $ave\_in\_out(p) = \sum_{q \in (p,q)} \frac{i(q)}{o(p)}$;
- Average out-links in in-neighbors: for a page q, $ave\_out\_in(q) = \sum_{p \in (p,q)} \frac{o(p)}{i(q)}$;
- In-degree: for a page p, $i(p)$ as defined in Chapter 2;
- Out-degree: for a page p, $o(p)$ as defined in Chapter 2;
- Reciprocity: the fraction of out-links that are also in-links of a page. For instance, if a page p has five out-links, and three of those out-neighbors links back to p, $reciprocity(p) = \frac{3}{5}$. If p has no out-links, $reciprocity(p) = 0$.

A short explanation of the implementation details of each of them is given now.

4.5.1 Assortativity

The first step is to calculate the degree of every page, by adding the out-degree and the in-degree of each of them. Then, the average degree of the neighbors (*ave_degree* vector in the pseudocode) of every page is then computed. Finally, the ratio between the degree of every page and the average degree of its neighbors is then calculated.

If the average degree of the neighbors of a page p is equal to 0, it also means that $degree(p) = 0$. In this case, an assortativity of 1 is given.

Assortativity pseudocode:

Require: N: number of nodes

```
1   for j : 1 ... N do
2       degree[j] = i(j) + o(j)
3   end for
4   for src : 1 ... N do //Follow links in the graph
5       for all links from src to dest do
6           ave_degree[src] = ave_degree[src] + (degree[dest] / degree[src])
7           ave_degree[dest] = ave_degree[dest] + (degree[src] / degree[dest])
8       end for
9   end for
10  for j : 1 ... N do
11      if ave_degree[j] ≠ 0 then
12          assortativity[j] = degree[j] / ave_degree[j]
13      else
14          assortativity[j] = 1
15      end if
16  end for
17  return assortativity
```

4.5.2 Average in-links in out-neighbors

The average in-links in out-neighbors computation is done by simply passing over the graph, and adding up the ratio between the in-degree of the out-neighbors of a page p and the out-degree of p.

Average in-links in out-neighbors pseudocode:

Require: N: number of nodes

```
1   ave_in_out = 0
2   for src : 1 ... N do //Follow links in the graph
3       for all links from src to dest do
4           ave_in_out[src] = ave_in_out[src] + (i(dest) / o(src))
5       end for
6   end for
7   return ave_in_out
```

4.5.3 Average out-links in in-neighbors

The average out-links in in-neighbors computation is done by simply passing over the graph, and adding up the ratio between the out-degree of the in-neighbors of a page p and the in-degree of p.

Average out-links in in-neighbors pseudocode:

Require: N: number of nodes

```
1  ave_out_in = 0
2  for dest : 1 ... N do //Follow links in the graph
3      for all links to dest from src do
4          ave_out_in[dest] = ave_out_in[dest] + (o(src) / i(dest))
5      end for
6  end for
7  return ave_out_in
```

4.5.4 In-degree and Out-degree

The Out-degree of every page may be found with a simple call to a method from the WebGraph framework. The In-degree is found by passing through all the out-neighbors of a page and adding 1 to its in-degree variable.

4.5.5 Reciprocity

To calculate the reciprocity of every page in a graph, it is necessary to follow all the links in the graph and check if there is a "back-link" for each of them. So, for each link from page p to page q, if there is also a link from page q to page p (the "back-link" of the link from page p), $\frac{1}{o(p)}$ is added to the reciprocity of p.

Require: N: number of nodes

```
1  reciprocity = 0
2  for src : 1 ... N do //Follow links in the graph
3      for all links from src to dest do
4          if ∃ link from dest to src then
5              reciprocity[src] = reciprocity[src] + (1 / o(src))
6      end for
7  end for
8  return reciprocity
```

4.6 Input and Output Formats

The input and output formats used in this work is explained in the two Subsections below.

4.6.1 Input Formats

Most of the implemented algorithms used only the BVGraph compressed format of the graph as input. The basename of the graph was then given as an argument for their executions.

For example: if the files for the graph are *"uk-2006-05.graph"*, *"uk-2006-05.offsets"*, and *"uk-2006-05.properties"*, the basename is then only *"uk-2006-05"*.

The TrustRank implementation also needs to receive a seed file as argument, with the following format (for the graph in Figure 3.4):

```
2
1
1
_
_
_
_
0
_
_
_
```

where:

- the first line indicates the number of good pages in the graph;
- the second line represents the trustfulness of page 0 (1 if good, 0 if bad);
- the third line represents the trustfulness of page 1 (1 if good, 0 if bad), and so on;
- pages that are not in the seed are represented with black lines (underscore characters in the example above).

The Inverted TrustRank implementation receives as argument a seed file with a similar format, but with the first line indicating the number of bad pages in the graph instead.

4.6.2 Output Formats

All implemented algorithm output files look the same: it is a consecutive list of values, where each line indicates the score for the page which the id is equal to $line\_number - 1$. So, the first 15 lines of the output file from the Truncated PageRank with truncation $T = 3$, for the Web graph of the *WEBSPAM-UK2006* dataset should look like:

```
2.2992026551612757E-8
2.283874110873647E-8
2.214069898303004E-8
2.214069898303004E-8
1.1984591420336383E-9
2.283874110873647E-8
2.283874110873647E-8
2.283874110873647E-8
2.283874110873647E-8
2.283874110873647E-8
2.283874110873647E-8
2.283874110873647E-8
2.205961027994608E-8
1.2451718963401025E-9
2.205961027994608E-8
. . .
```

where:

- the first line represents the Truncated PageRank with truncation $T = 3$ score for page 0;
- the second line represents the Truncated PageRank with truncation $T = 3$ score for page 1, and so on;

5 EXPERIMENTS AND RESULTS

This Chapter presents the experiments conducted during this work. Each of the classifiers built is presented together with their results and a comparative analysis. The features used by each classifier were calculated using the algorithms discussed in Chapter 4.

In Section 5.1, Weka, the tool used to create the classifiers is briefly presented. In Section 5.2, the necessary steps to be taken before the generation of such classifiers is described. Section 5.3 details the results of each of the proposed classifiers. Finally, Section 5.4 shows a comparison between the results of the two studied *spamdexing* detection techniques and an analysis of the combination of both techniques (with and without the use of the extra attributes, presented in Chapter 4, as features).

5.1 Weka

Weka (WITTEN; FRANK, 2005), as presented in its official Web site[1], *"is a collection of machine learning algorithms for data mining tasks"*. These algorithms can either be applied directly to a dataset, with the use of an available Graphical User Interface (GUI), or called from a code in the Java programming language.

For this work, the Weka's implementation of C4.5 (QUINLAN, 1993) decision trees is used. The choice of C4.5 was because it is the most usual way of experimenting with this kind of data, as seen in most papers of this area of research - like (BECCHETTI et al., 2006) and (CASTILLO et al., 2007). These decision trees may be used as classifiers to determine, based on a set of features, whether a page is or is not a spam page. Describing such classifiers in detail is not the focus of this work, and more information about them may be found in (WITTEN; FRANK, 2005).

[1] http://www.cs.waikato.ac.nz/ml/weka

As an example, the classifier generated by Weka featuring the TrustRank algorithm results (which is further detailed in Section 5.3) is shown:

```
log_trustrank_div_pagerank <= -3.58793
|   log_pagerank <= -10.2313
|   |   trustrank_div_pagerank <= 0.00382
|   |   |   log_pagerank <= -10.8684
|   |   |   |   trustrank_div_pagerank <= 0.001275
|   |   |   |   |   log_pagerank <= -10.8778: normal
|   |   |   |   |   log_pagerank > -10.8778: spam
|   |   |   |   trustrank_div_pagerank > 0.001275: normal
|   |   |   log_pagerank > -10.8684
|   |   |   |   log_pagerank <= -10.5393: spam
|   |   |   |   log_pagerank > -10.5393
|   |   |   |   |   log_pagerank <= -10.521: normal
|   |   |   |   |   log_pagerank > -10.521
|   |   |   |   |   |   log_pagerank <= -10.2725: spam
|   |   |   |   |   |   log_pagerank > -10.2725: normal
|   |   trustrank_div_pagerank > 0.00382
|   |   |   log_pagerank <= -10.8153: normal
|   |   |   log_pagerank > -10.8153
|   |   |   |   trustrank_div_pagerank <= 0.014271: spam
|   |   |   |   trustrank_div_pagerank > 0.014271
|   |   |   |   |   log_pagerank <= -10.4121: normal
|   |   |   |   |   log_pagerank > -10.4121: spam
|   log_pagerank > -10.2313
|   |   trustrank_div_pagerank <= 0.015484: spam
|   |   trustrank_div_pagerank > 0.015484
|   |   |   log_pagerank <= -9.94297: spam
|   |   |   log_pagerank > -9.94297
|   |   |   |   pagerank <= 0.000097: normal
|   |   |   |   pagerank > 0.000097
|   |   |   |   |   trustrank_div_pagerank <= 0.023442
|   |   |   |   |   |   trustrank_div_pagerank <= 0.016887: normal
|   |   |   |   |   |   trustrank_div_pagerank > 0.016887: spam
|   |   |   |   |   trustrank_div_pagerank > 0.023442: normal
log_trustrank_div_pagerank > -3.58793: normal
```

A tree of size 35 and 18 leaves.

5.2 Preparing the Test Data

In this Section, the steps taken before the creation of the classifiers shown in Section 5.3 are presented.

5.2.1 Label Sets and Validation

As a part of the *WEBSPAM-UK2006* dataset (CASTILLO et al., 2006), two sets of human-labeled hosts are provided: the first one is provided as a training set for tools like Weka, and the second one is provided as a testing set for the classifiers generated for such tools. The distribution of labels in these sets are as follows:

- *set*1: 4948 (88%) non-spam labels *versus* 674 (12%) spam labels;
- *set*2: 601 (32.5%) non-spam labels *versus* 1250 (67.5%) spam labels.

The problem is that, preliminary tests in these sets have shown that these label balances are not the best, since the significant difference in the amount of each sample (non-spam, and spam) caused the generated classifiers to misclassify too many entrances. To solve this problem, a mixture of the two sets was proposed: the non-spam labels from *set*1 are joined with the spam labels from *set*2, and the non-spam labels from *set*2 are joined with the spam labels from *set*1, which causes the creation of two whole new sets:

- *set*3: 4948 (80%) non-spam labels *versus* 1250 (20%) spam labels;
- *set*4: 601 (47%) non-spam labels *versus* 674 (53%) spam labels.

Tests using both new sets as training sets were performed. Even though the classifiers using *set*4 for training presented, in general, better F-measures than the ones built with *set*3, they have also shown a False positive rate three to six times bigger than the classifiers that used *set*3 as training set. The reason for this overall better results with *set*3 may be strongly related to the reality of the Web, where the number of non-spam pages really outnumbers the number of spam pages (even though the later are the most noticed by the users). For this reason, it was opted to use *set*3 as the training set for the classifiers that are presented in Section 5.3.

Furthermore, the almost 50%/50% ratio achieved in the creation of *set*4 gave the opportunity to use it as the seed for both TrustRank and Inverted TrustRank algorithms, since it has a fair distribution of good and bad pages. A fair distribution which would cause an unfair evaluation for both of the algorithms if *set*4 was also used as the testing set for the generated classifiers. Two options then emerged: either breaking *set*3 in two different sets, with the same label balance as the one presented in *set*3, so that one could be used as the training set and the other as the testing set; or using a K-fold cross-validation.

K-fold cross-validation is a technique that divides the data in K approximately equal partitions, then each part is held out as the validation data for the testing model, and the remaining K-1 folds are used as training data. The cross-validation process is then repeated K times, each time using a different partition as the validation data. The K error estimates are then averaged to produce a single error estimate. Using 10 as fold number is the most common choice for the cross-validation method, and was also the option used in this work.

5.2.2 Score Calculation

In order to calculate the scores produced by the studied ranking algorithms, the host graph of the *WEBSPAM-UK2006* dataset was used instead of using its whole Web graph. The reason for this is that preliminary tests have shown that the difference in the accuracy of the classifiers generated with the page-level scores and with the host-level scores is small, but the time taken to calculate the same scores for a 77-million-node graph is much higher than the time it takes to calculate them for a 11-thousand-node graph. Also, the labels provided are at host-level, so it would not be possible to use all the calculated page-level scores anyway.

For doing this, the procedure explained in Section 4.1 was used to create a BVGraph compressed format graph from the *"uk-2006-05.hostnames.txt.gz"* file provided (though a text-level conversion was necessary - and made using one of the the auxiliary scripts presented in appendix A). The weights of the links were ignored.

It is also important to notice that the Truncated PageRank scores were calculated using truncations $T = 2,3,4$, since $T = 1$ would be just based on in-degree, and $T > 4$ would start to truncate more than just link farms, which are the main targets of this algorithm. Moreover, the PageRank used as feature of all classifiers was calculated using the Truncated PageRank implementation, with truncation $T = 0$.

The number of iterations used during the score calculation of each of the implemented algorithms were the same as the ones suggested by their original authors: 50 iterations for the Truncated PageRank and 20 iterations for the TrustRank and Inverted TrustRank score calculations. The extra attributes, that were also calculated, may be easily computed in one to three steps over the Web graph, as shown in their pseudo codes in Chapter 4.

5.2.3 Conventions

Some of the features used by the classifiers that are shown in Section 5.3 were made by using the logarithm of some of the calculated scores, as well as the ratio between some of them. When computing these features, some conventions were used to avoid null values:

- $log(x) = -50$ if $x \leqslant 0$
- $\frac{x}{0} = 1$ if $x = 0$
- $\frac{x}{0} = 0$ if $x \neq 0$

5.3 Classifiers

This Section presents the 18 different classifiers which were built using the scores computed by the implemented algorithms of Chapter 4. First, we deal with the straightforward application of the calculated scores as the features for the C4.5 decision trees implementation of Weka, the Standard Classifiers. The results of the ensemble of classifiers created with the help of the bagging technique are shown afterwards.

5.3.1 Standard Classifiers

Before listing the generated classifiers, it is important to explain the use of the logarithm of the calculated scores as features: the use of the raw scores as the only features for the classifiers was not properly taken into account by the decision tree algorithm. Since the computed scores seems to be distributed according to a log-normal distribution, then their logarithms are distributed according to a normal distribution, which this implementation of C4.5 decision trees seems to handle better. Also, the ratio of the scores with the PageRank was used as a comparison feature: to show the relation of the score of a certain page with its PR value.

For abbreviation purpose, the Truncated PageRank with truncations T = 2, 3, 4 are referred to as TPR-2, TPR-3, and TPR-4 respectively.

- **TPR-2 based classifier**: uses as features the TPR-2 scores, the PR scores, the TPR-2 scores divided by the PR scores, and the logarithm of these three values, totaling 6 features.

		Prediction	
		Non-spam	Spam
True Label	Non-spam	4570	378
	Spam	554	696

Table 5.1: Confusion matrix for TPR-2 based classifier

Precision	Recall	F-measure	FP rate	FN rate
0.65	0.56	0.60	7.6%	44%

Table 5.2: Evaluation measures for TPR-2 based classifier

- **TPR-3 based classifier**: uses as features the TPR-3 scores, the PR scores, the TPR-3 scores divided by the PR scores, and the logarithm of these three values, totaling 6 features.

		Prediction	
		Non-spam	Spam
True Label	Non-spam	4593	355
	Spam	549	701

Table 5.3: Confusion matrix for TPR-3 based classifier

Precision	Recall	F-measure	FP rate	FN rate
0.66	0.56	0.60	7.2%	44%

Table 5.4: Evaluation measures for TPR-3 based classifier

- **TPR-4 based classifier**: uses as features the TPR-4 scores, the PR scores, the TPR-4 scores divided by the PR scores, and the logarithm of these three values, totaling 6 features.

		Prediction	
		Non-spam	Spam
True Label	Non-spam	4533	415
	Spam	479	771

Table 5.5: Confusion matrix for TPR-4 based classifier

Precision	Recall	F-measure	FP rate	FN rate
0.65	0.62	0.63	8.4%	38%

Table 5.6: Evaluation measures for TPR-4 based classifier

At this point, it is possible to notice that even though the performance of the classifiers is getting better as we truncate more and more the PageRank calculation, the number of False positives is also increasing. This means that the applied truncation begins to "to cut" a larger number of non-spam pages. This may also indicate that the Truncated PageRank with truncation T = 3 is be the most appropriate for the host graph we are dealing with, since it also presents the best Precision of all three classifiers presented until now.

For a better use of the Truncated PageRank scores, with all truncation levels, a combined classifier, using the features of all the three classifiers above was tested:

- **TPR-X based classifier**: uses all features of the previous classifiers, totaling 14 features.

		Prediction	
		Non-spam	Spam
True Label	Non-spam	4498	450
	Spam	436	814

Table 5.7: Confusion matrix for TPR-X based classifier

Precision	Recall	F-measure	FP rate	FN rate
0.64	0.65	0.65	9.1%	35%

Table 5.8: Evaluation measures for TPR-X based classifier

The overall performance (F-measure) improved again, as well as the number of spams what were correctly classified, although the FP rate continues to rise. Depending on the intention when using this kind of classification, the False positive rate (if not too high) may be completely ignored in favor of detecting more spam pages, but then, the use of this kind of technique alone is not indicated.

The next classifiers use the TrustRank and the Inverted TrustRank scores as features.

- **TR based classifier**: uses as features the TR scores, the PR scores, the TR scores divided by the PR scores, and the logarithm of these three values, totaling 6 features.

		Prediction	
		Non-spam	Spam
True Label	Non-spam	4614	334
	Spam	350	900

Table 5.9: Confusion matrix for TR based classifier

Precision	Recall	F-measure	FP rate	FN rate
0.73	0.73	0.73	6.8%	28%

Table 5.10: Evaluation measures for TR based classifier

- **ITR based classifier**: uses as features the ITR scores, the PR scores, the ITR scores divided by the PR scores, and the logarithm of these three values, totaling 6 features.

		Prediction	
		Non-spam	Spam
True Label	Non-spam	4614	334
	Spam	350	900

Table 5.11: Confusion matrix for ITR based classifier

Precision	Recall	F-measure	FP rate	FN rate
0.63	0.50	0.56	7.2%	51%

Table 5.12: Evaluation measures for ITR based classifier

With these results, the *spamdexing* detection technique based on TrustRank proves itself as the best one so far, even with the use of a seed that corresponds to only 5% of the whole host graph. On the other hand, the use of the Inverted TrustRank scores as features for a classifier have shown the worst results until now. This behavior is, nevertheless, expected, since the majority of pages of this host graph are non-spam pages, and the spam pages are also supposed to be near each other, so that it is more difficult to propagate a non-trust factor than a trust factor.

Combining the last two classifiers above gives us the following classifier:

- **TR & ITR based classifier**: uses all features of both TR and ITR based classifiers, totaling 10 features.

		Prediction	
		Non-spam	Spam
True Label	Non-spam	4633	315
	Spam	363	887

Table 5.13: Confusion matrix for TR & ITR based classifier

Precision	Recall	F-measure	FP rate	FN rate
0.74	0.71	0.72	6.4%	29%

Table 5.14: Evaluation measures for TR & ITR based classifier

Even though this combined classifier is using both of the features from TR and ITR based classifiers, the F-measure results are worse than the one based only on TR. However, it has improved Precision and a lower False positive rate with the use of the ITR based classifier features.

If we use all the features, from all the classifiers presented to this point, we may then construct a combined classifier.

- **Combined classifier**: uses as features all the features used in both TPR-X and TR & ITR based classifiers, totaling 22 features.

		Prediction	
		Non-spam	Spam
True Label	Non-spam	4641	307
	Spam	356	894

Table 5.15: Confusion matrix the Combined classifier

Precision	Recall	F-measure	FP rate	FN rate
0.74	0.72	0.73	6.2%	29%

Table 5.16: Evaluation measures for the Combined classifier

Which shows itself as the classifier with the best balance in its evaluation measures, what is expected, since we used more information in its learning scheme.

With the sole purpose of improving even more the overall performance of this Combined classifier, the algorithms to calculate the extra attributes of the pages, shown in Chapter 4, were implemented. The objective is to try and explore the properties of the pages. So, a classifier using a total of 34 features was generated.

- **Combined classifier with extra features**: uses as features the TPR-2, TPR-3, TPR-4, TR, and ITR scores, as well as the PR score, and all the previous scores divided by the corresponding PR score. Furthermore, the assortativity, average in-links in out-neighbors, average out-links in in-neighbors, in-degree, out-degree, and reciprocity of the labeled pages were also used as features. The logarithm of all these values formed the other half of the features.

		Prediction	
		Non-spam	Spam
True Label	Non-spam	4657	291
	Spam	334	916

Table 5.17: Confusion matrix the Combined classifier with extra features

Precision	Recall	F-measure	FP rate	FN rate
0.76	0.73	0.75	5.9%	27%

Table 5.18: Evaluation measures for the Combined classifier with extra features

These good results show that all these extra attributes may be indeed used as a help factor to characterize spam pages over non-spam pages.

5.3.2 Classifiers with Bagging

As suggested in (CASTILLO et al., 2007), the bagging technique may be used to improve the results of the proposed classifiers: *"bagging is a technique that creates an ensemble of classifiers by sampling with replacement from the training set to create N classifiers whose training sets contain the same number of examples as the original training set, but may contain duplicates. The labels of the test set are determined by a majority vote of the classifier ensemble"*. In this work, this *N* was set to 10.

Bagging improved the overall results of the classifiers by reducing the False positive rate while increasing Precision and F-measure. So, for each of the previously presented classifiers:

- **TPR-2 based classifier**:

	Precision	Recall	F-measure	FP rate	FN rate
Without bagging	0.65	0.56	0.60	7.6%	44%
With bagging	0.69	0.55	0.61	6.2%	45%

Table 5.19: TPR-2: with bagging *versus* without bagging

- **TPR-3 based classifier**:

	Precision	Recall	F-measure	FP rate	FN rate
Without bagging	0.66	0.56	0.61	7.2%	44%
With bagging	0.69	0.56	0.62	6.3%	44%

Table 5.20: TPR-3: with bagging *versus* without bagging

- **TPR-4 based classifier**:

	Precision	Recall	F-measure	FP rate	FN rate
Without bagging	0.65	0.62	0.63	8.4%	38%
With bagging	0.70	0.60	0.65	6.6%	39%

Table 5.21: TPR-4: with bagging *versus* without bagging

- **TPR-X based classifier**:

	Precision	Recall	F-measure	FP rate	FN rate
Without bagging	0.64	0.65	0.65	9.1%	35%
With bagging	0.70	0.64	0.67	6.9%	36%

Table 5.22: TPR-X: with bagging *versus* without bagging

- **TR based classifier**:

	Precision	Recall	F-measure	FP rate	FN rate
Without bagging	0.73	0.72	0.73	6.8%	28%
With bagging	0.75	0.70	0.73	5.8%	29%

Table 5.23: TR: with bagging *versus* without bagging

- **ITR based classifier**:

	Precision	Recall	F-measure	FP rate	FN rate
Without bagging	0.63	0.50	0.56	7.2%	51%
With bagging	0.69	0.46	0.55	5.1%	54%

Table 5.24: ITR: with bagging *versus* without bagging

- **TR & ITR based classifier**:

	Precision	Recall	F-measure	FP rate	FN rate
Without bagging	0.74	0.71	0.72	6.4%	29%
With bagging	0.77	0.70	0.73	5.3%	30%

Table 5.25: TR & ITR: with bagging *versus* without bagging

- **Combined classifier**:

	Precision	Recall	F-measure	FP rate	FN rate
Without bagging	0.74	0.72	0.73	6.2%	29%
With bagging	0.77	0.73	0.75	5.6%	27%

Table 5.26: Combined: with bagging *versus* without bagging

- **Combined classifier with extra features**:

	Precision	Recall	F-measure	FP rate	FN rate
Without bagging	0.76	0.73	0.75	5.9%	27%
With bagging	0.79	0.76	0.78	5.0%	24%

Table 5.27: Combined with extra features: with bagging *versus* without bagging

Noticeably, when applying the bagging technique in the Combined classifiers (with and without extra features), there is a gain in all evaluation measures used in this work.

5.4 Final Evaluation

For a final comparison between the two studied *spamdexing* detection techniques, the version with bagging of the generated TPR-X and TR & ITR based classifiers are used, as well as the Combined classifier, with and without the use of the extra features.

	Precision	Recall	F-measure	FP rate	FN rate
TPR-X	0.70	0.64	0.67	6.9%	36%
TR & ITR	0.77	0.70	0.73	5.3%	30%
Combined	0.77	0.73	0.75	5.6%	27%
Combined + extras	0.79	0.76	0.78	5.0%	24%

Table 5.28: Final comparison between classifiers

The Truncated PageRank based classifier presents a satisfactory performance, with a good Precision, Recall, and with acceptable False positive and False negative rates. Still, it is worse than the TrustRank based classifier, which is the main responsible for the very good results obtained by the combined classifier, as shown in Table 5.28.

Both single-technique based classifiers are able to detect, alone, more than 60% of the spams present in the dataset used in this work, and while doing this, they incorrectly classify less than 7% non-spam pages as spam pages. Since, usually, it is much better to discard normal pages than having a lot of spam pages infecting our rank results, this rates safely allow both studied *spamdexing* detection techniques to be used in real-life environments.

Furthermore, the combined use of both techniques has proven to be even better, detecting 73% of the presented spam-mass, and only mistaking 5.6% of non-spam pages as spam. Also, although not attempted, an implementation that calculates all the scores at the same time could be made, reducing the implementation costs of using the combined classifier to the same cost of using a single-technique based classifier.

The Combined classifier with extra features was generated mainly with the purpose to be a proof of concept that, by using more characteristics of the pages, it is possible to achieve highly accurate results in *spamdexing* detection. This also shows that even though

spam pages try to disguise themselves as non-spam pages, their most simple attributes are quite different.

A comparison between the four classifiers listed above is also shown in Figures 5.1 and 5.2.

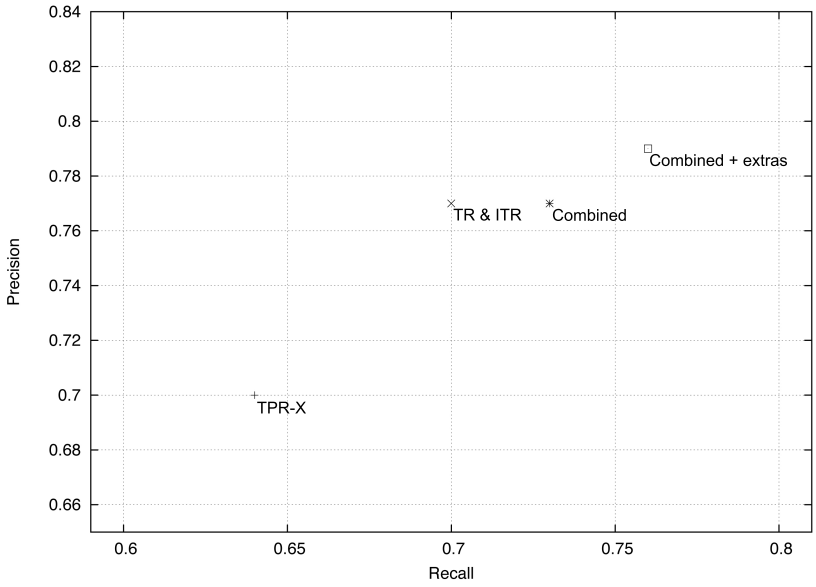

Figure 5.1: Precision and Recall of the classifiers

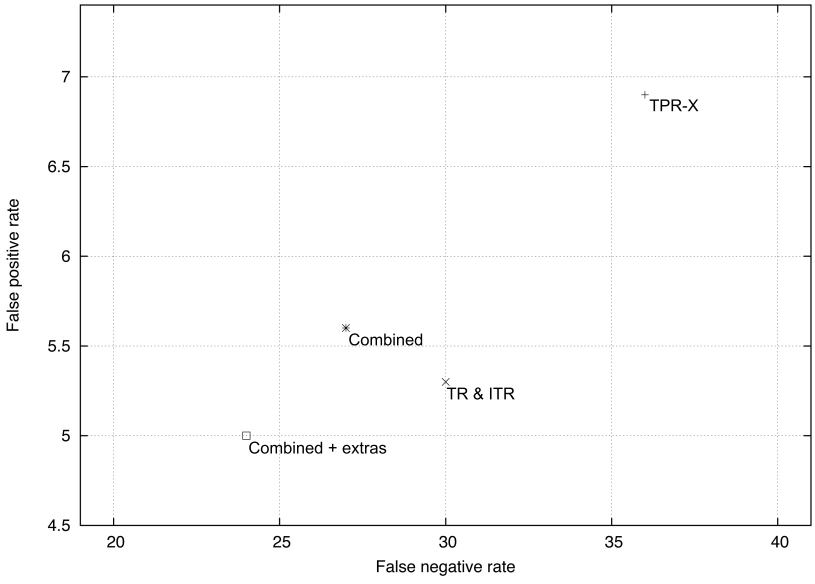

Figure 5.2: Error rates of the classifiers

6 CONCLUSIONS

The present work has evaluated the quality of two *spamdexing* detection techniques, as well as their combination. The goal was to check whether or not the current state of the art is good enough in terms of detecting the largest possible number of spam pages in a collection of Web sites, also considering the resulting mistakes. The conclusion is that the results were very good in all cases, specially when the two techniques were combined.

During this work, the two selected *spamdexing* detection techniques were studied and implemented, along with a small set of extra page-level attributes. The results of all implemented algorithms were then used as features to generate decision trees with the help of Weka (WITTEN; FRANK, 2005). The performance of each of the 18 built classifiers was then evaluated with well-known Information Retrieval metrics, such as Precision and Recall.

Even though the results achieved in the performed evaluations cannot be compared with the quality of the spam filters found in modern email solutions, they are still very acceptable, even when considering the False positive and False negative percentages. The reason for this is that, while losing an important email is really bad for end-users, misclassifying a non-spam page as spam is not that problematic. If the misclassified non-spam page is really relevant, it can still be found by following links, or it can simply be moved away from the first result page of search engines.

Also, the combined classifier with extra features proved that, when adding more features, the results may still be improved. So, the combination of these link-based *spamdexing* detecting techniques with content-based *spamdexing* detection techniques may lead to results with an even higher quality than the ones produced in this work.

Both Truncated PageRank and TrustRank algorithms are strongly based on the PageRank algorithm. A possible idea for a future work could be the study of other ranking algorithms, such as HITS (KLEINBERG, 1999) and SALSA (LEMPEL; MORAN, 2000), trying to modify them in the same fashion as the modifications applied to the PR. It would be really interesting to see how this could be done, and also to compare such modifications with the PageRank-based ones.

REFERENCES

BAEZA-YATES, R.; BOLDI, P.; CASTILLO, C. Generalizing PageRank: damping functions for link-based ranking algorithms. In: ANNUAL INTERNATIONAL ACM SIGIR CONFERENCE ON RESEARCH AND DEVELOPMENT IN INFORMATION RETRIEVAL, 29. **Proceedings...** ACM, 2006. p.315.

BAEZA-YATES, R.; RIBEIRO-NETO, B. et al. **Modern Information Retrieval**. [S.l.]: Addison-Wesley Reading, MA, 1999.

BECCHETTI, L. et al. Using Rank Propagation and Probabilistic Counting for Link-Based Spam Detection. In: WORKSHOP ON WEB MINING AND WEB USAGE ANALYSIS (WEBKDD). **Proceedings...** Citeseer, 2006.

BOLDI, P. et al. Ubicrawler: a scalable fully distributed web crawler. **Software: Practice and Experience**, [S.l.], v.34, n.8, p.711–726, 2004.

BOLDI, P.; VIGNA, S. The Webgraph Framework I: compression techniques. In: INTERNATIONAL CONFERENCE ON WORLD WIDE WEB, 13. **Proceedings...** ACM, 2004. p.595–602.

BRODER, A. et al. Graph Structure in the Web. **Computer networks**, [S.l.], v.33, n.1-6, p.309–320, 2000.

CASTILLO, C. et al. A Reference Collection for Web Spam. In: ACM SIGIR FORUM. **Anais...** ACM, 2006. p.24.

CASTILLO, C. et al. Know your Neighbors: web spam detection using the web topology. In: ANNUAL INTERNATIONAL ACM SIGIR CONFERENCE ON RESEARCH AND DEVELOPMENT IN INFORMATION RETRIEVAL, 30. **Proceedings...** ACM, 2007. p.430.

DAVISON, B. Recognizing Nepotistic Links on the Web. **Artificial Intelligence for Web Search**, [S.l.], p.23–28, 2000.

GYONGYI, Z.; GARCIA-MOLINA, H. Web Spam Taxonomy. **Adversarial Information Retrieval on the Web**, [S.l.], 2005.

GYONGYI, Z.; GARCIA-MOLINA, H.; PEDERSEN, J. Combating Web Spam with TrustRank. In: THIRTIETH INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES. **Proceedings...** [S.l.: s.n.], 2004. p.587.

KLEINBERG, J. Authoritative Sources in a Hyperlinked Environment. **Journal of the ACM (JACM)**, [S.l.], v.46, n.5, p.604–632, 1999.

LEMPEL, R.; MORAN, S. The Stochastic Approach for Link-Structure Analysis (SALSA) and the TKC Effect. **Computer Networks**, [S.l.], v.33, n.1-6, p.387–401, 2000.

NTOULAS, A. et al. Detecting Spam Web Pages Through Content Analysis. In: INTERNATIONAL CONFERENCE ON WORLD WIDE WEB, 15. **Proceedings...** ACM, 2006. p.92.

PAGE, L. et al. **The Pagerank Citation Ranking**: bringing order to the web. [S.l.]: Technical report, Stanford Digital Library Technologies Project, 1998.

QUINLAN, J. **C4. 5**: programs for machine learning. [S.l.]: Morgan Kaufmann, 1993.

WITTEN, I.; FRANK, E. **Data Mining**: practical machine learning tools and techniques. [S.l.]: Morgan Kaufmann Pub, 2005.

ZHANG, H. et al. Making Eigenvector-Based Reputation Systems Robust to Collusion. **Algorithms and Models for the Web-Graph**, [S.l.], p.92–104, 2004.