Ranking Documents Using Degrees of Separation Analysis from a Large Dataset of Semantic Relationships

by

Matthew R. Howell

(Under the direction of Kang Li)

Abstract

In this paper, a ranking mechanism is presented that ranks documents based on their Semantic Association Similarity, which is defined as the close-ness (based on degrees of separation) of associations between the entities found in each document. A large semantic knowledge base with over 1.6 million entities and 24 million associations is used as the backend dataset for comparison. Multiple ranking techniques are evaluated and speed concerns are addressed. Bloom filters are used to improve ranking speed while introducing a small percentage of false positives. A real world example of spam page identification is investigated.

Index words: Semantic Similarity, Semantic Associations, Semantic Relationships, Document Comparison, Degrees of Separation, Semantic Web, Bloom Filter, Relation Discovery, Web Spam

Ranking Documents Using Degrees of Separation Analysis from a Large Dataset of Semantic Relationships

by

Matthew R. Howell

B.S., The University of Georgia, 2003

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Science

Athens, Georgia

2010

Ranking Documents Using Degrees of Separation Analysis from a Large Dataset of Semantic Relationships

by

Matthew R. Howell

Approved:

Major Professor: Kang Li

Committee: Prashant Doshi
Lakshmish Ramaswamy

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2010

Acknowledgments

I'd like to thank Dr. Kang Li for his input during my long research process. His insights and encouragement during our talks always left me energized and enthusiastic about my research. I'd also like to thank Zhenyu Zhong and Magnus Skjegstad for their willingness to provide code and help with my development of Java based Bloom filters to aid the ranker.

I also owe a debt of gratitude to the many people dedicated to the development of DBpedia and other semantic datasets. Their data mining and further development of Wikipedia provide an excellent source of knowledge for use throughout computer applications. Yahoo! research was also key in my research because they provided a large, unbiased and classified set of pages that was vital in the evaluation of the ranker.

Thanks to Shawn Baker for his statistical expertise and analysis. Lastly, thanks to Chiahsun Ho (Alex) and ManChon U (Kevin) for reviewing my work and providing me with critical comment and feedback.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ranking documents is a key function of computer applications, especially in the context of search engines. Many ranking techniques have been analyzed and used in practice, including analysis of text, link structure, and semantic meaning of content. A high search ranking has a direct correlation to the number of visits to a site and therefore can directly affect revenue. Therefore, web spammers have an incentive to manipulate their pages in attempt to raise their page rank, the process also know as *spamdexing*. The exact techniques used by search engines are not publicly available so that spammers do not have direct knowledge of their ranking methods. However, an analysis of search results show that the techniques used by search engines to identify web spam are not completely effective, as a recent search[1] for `kardashian pharmacy` that resulted in 5 of the top 10 results being web spam shows. Identifying and removing web spam from indexes is vital to the integrity of ranking techniques.

1.1 Motivation

The impact of spamdexing is felt by search engine users, legitimate companies, and the search engines themselves. Search engine users may feel frustration and dissatisfaction when they inadvertently arrive at a spam page from search engine results, but more importantly they could be subjected to malicious content that exploits browser or plug-in flaws. Malicious content could have disastrous results, including stolen private information and/or installation of unauthorized software, such as a virus or worm.

[1] On October 16, 2010

Spamdexing also affects legitimate companies because they may experience less traffic when spam pages that exploit ranking techniques have higher rankings. [1] states that users only look at the first page of search results 85% of the time. Therefore, the difference between being ranked 10 and 11 (if there are 10 results per page) is substantial. More site visits would result in more revenue for companies and it is imperative for invalid pages to be removed from indexes, or at least be ranked lower than legitimate pages.

Search engines also are impacted by web spam. In addition to the degraded search results that users will experience, search engines will also incur expenses for crawling, indexing and storing web spam. The expenses include consumption of computing and network resources as well as hard drive space. Salary costs are also incurred for the programming of spam detection techniques and manual classification if needed.

These costs are not insignificant when viewed on a large scale and a successful classification method that stops spam from being indexing and stored would benefit each of the three parities listed above. As larger knowledge bases of semantic data and relationships become available, such as structured extractions of Wikipedia[2], there is potential that this knowledge will help provide alternative and better ranking mechanisms.

1.2 Contributions

This paper discusses a Semantic Association Ranking application that ranks the relationships between documents based on the close-ness of the entities found in each document. The research can have many applications throughout the electronic world, including improving search engine results and web spam identification. Ranking mechanisms using a large dataset, i.e. DBpedia's knowledge base, and based on semantic association similarity are analyzed and evaluated. A real world example of ranking a list of URLs for similarity to a given spam page, in hopes of identifying spam candidates is evaluated by using the `WEBSPAM-UK2007` dataset. Figure 1.1 shows a diagram of the application with inputs of a classified page, i.e. a page

[2]http://www.wikipedia.org

whose spam/nonspam status is known, and a set of unclassified pages. With these inputs, the ranking application outputs a ranked list of pages that are the most similar based on semantic closeness of the associations between entities found on the pages. While the application alone is not found to discern spam from nonspam, when used in conjunction with other methods, it may be a valuable addition. Speed concerns of the ranker are highlighted and Bloom filters are used to provide a reliable, accurate, small and fast ranking method.

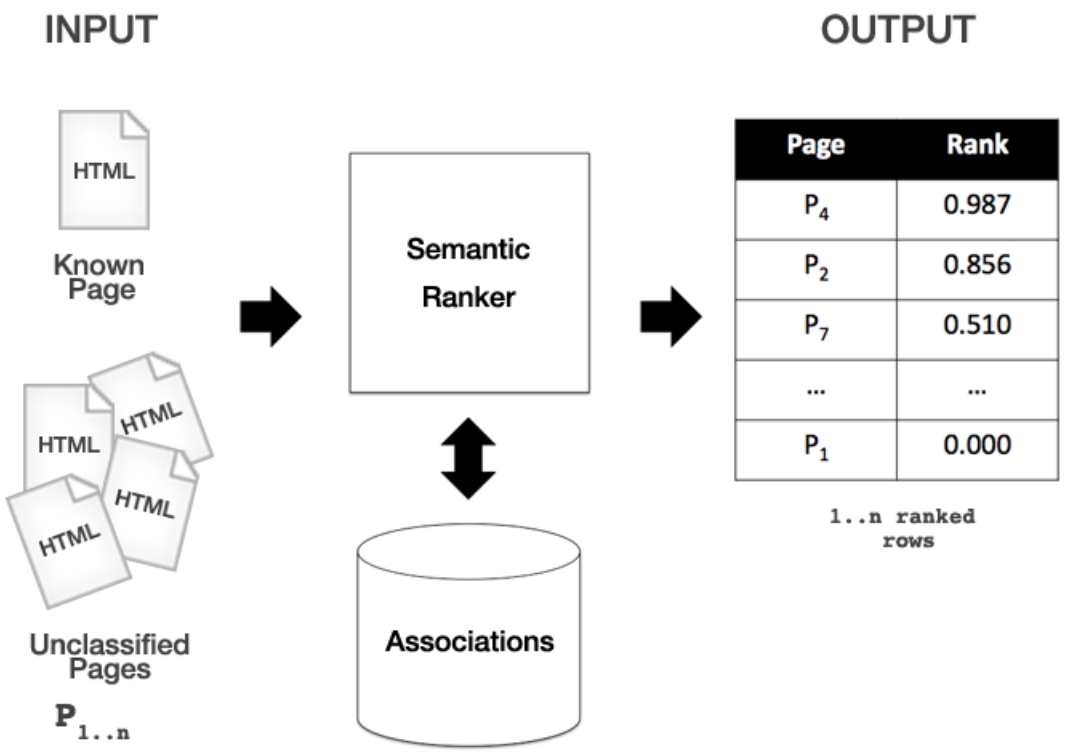


Figure 1.1: Overall System Architecture

Chapter 2

Related Work

Several research efforts focus on analyzing and ranking Semantic Associations with growing knowledge bases, such as DBpedia[1] and Freebase[2]. [2, 3] explore DBpedia's data by providing user interfaces to find connections between entities. Associations between two entities are provided in real time with intermediate entities and relationships defined. In [2], The dataset is treated as an undirected graph and pre-processed to retrieve the maximal connected subgraphs. These subgraphs are used to immediately determine if two entities have a semantic association. If two entities have an association, the user is able to view the associations by providing a maximum distance for the associations. [3] uses a SPARQL endpoint to query for relationships at run time, but does not treat the graph as undirected. It only allows one change in direction of an association throughout the path to improve performance. So for two entities, `A` and `B`, it will find associations that have a path directly from `A` to `B`, `B` to `A` or with an intermediate node, `C`, where there is a path from `A` to `C` and `B` to `C` or from `C` to `A` and `B`.

The application presented in this paper also uses a pre-processed version of the DBpedia infobox dataset, but does not focus on the ability to explore how entities are related, only the shortest path between them. Also, this application will perform many lookups for each pair of documents examined and therefore the shortest path between all entities was computed beforehand.

[4] ranks entities on Wikipedia from a given passage into a hierarchy of categories, including most important, important related, unrelated and not sure, using several ranking

[1] http://www.dbpedia.org
[2] http://www.freebase.org

techniques. They found that a good determinate of the importance of an entity is to use an *inverse entity frequency (ief)* method, which calculates how rare the entity is compared to the total number of sentences in the passage. This method reduces the value of more general entities and could be generated for all entities in the DBpedia dataset. The ranking methods used by this application are simple and do not account for the rank of entities in associations. However, it is likely that a more robust ranking algorithm that combined taking *ief* with the association ranking methods used in this application, would improve the results.

As more information is added to Wikipedia, DBpedia's knowledge base will continue to grow and the number and types of associations known will increase. Improvements to Wikipedia proposed by [5] where links to other articles are given a Semantic annotation would give DBpedia the ability to glean more information out of Wikipedia's pages. Other applications, such as [6, 7], make use of multiple online knowledge bases and ontologies to discover relationships at run time. The application presented here could use more than one ontology for relation discovery, however the ontologies would have to be incorporated during pre-processing as the time and space required to discover the many associations found between two documents would not be feasible in real time without pre-processing the relation discovery step.

[8] focuses on ranking individual semantic associations between entities using various semantic and statistical metrics, including a user-driven context and path length. A human subjects evaluation was performed and one result was that humans tended to rank shorter associations between entities higher. This application will allow for ranking documents based on a context of 5 general categories, i.e. people, places, organizations, works, or other. However, it could be extended to include more specific contexts because type information is available for each entity in the dataset.

Bloom filters, devised by Burton H. Bloom in [9], are used frequently with speed critical applications to reduce intensive set membership queries, while allowing a percentage of errors

due to false positives. Popular applications, such as Google's BigTable[3] database system and the Squid Proxy cache[4], use Bloom Filters to reduce expensive lookups [10]. [11] uses an extended bloom filter to speed spam ranking and provides a 6x speed improvement over two popular spam filters.

Combating Web spam is an ongoing issue and has been addressed in numerous research efforts. In an analysis of the prevalence of spam pages, [12] found that 13.8% of English pages were spam. Techniques such as a trust rank based on link structure analysis and semantic analysis of spam page content have been researched [12, 13, 14, 15].

In [15], semantic analysis of page's content was not able to effectively identify spam pages. This work, however, aims to analyze the effectiveness of analyzing Semantic Associations between content found in multiple documents, or pages. Two methods that spammers use to attempt to influence search engine rankings are to include a large number of distinct terms in a document and by repeating some "targeted" terms [16]. [17] found that overall web pages change slowly, on a weekly basis only 35% of pages will have any modifications and less than 1% will be changed completely. Because pages do not change very frequently and if a spammer uses a common set of terms on several pages and one of his spam pages has been identified, this application may help to flag other pages created by the spammer as candidates for spam detection using other techniques.

[3]http://labs.google.com/papers/bigtable.html
[4]http://www.squid-cache.org

Chapter 3

Dataset

The Web consists of documents discussing highly varied topics and so for this application to be effective, a large, thorough dataset was essential. Two sources of semantic data were evaluated for use as a dataset, DBpedia and Freebase. Both sources provide information about resources, also known as entities, and link resources together by providing associations between them. DBpedia extracts structured information from Wikipedia and provides links to other popular datasets including Freebase. Freebase combines information from Wikipedia and directly entered user input. Both of these sources make their data available for download. DBpedia has knowledge of over 2.6 million entities, including 198,000 people, 328,000 places, 20,000 companies, 101,000 musical works, and 34,000 films and is easily accessible, including dumps of the entire dataset [19].

In this case, DBpedia had advantages over Freebase. [2] states that DBpedia is a densely connected dataset and there is an average of 5.67 outgoing relationships for each entity. DBpedia datasets are also divided into smaller files based on the information they contain. Freebase's dataset, as of this writing, is output as a single file that is over 26 GB when uncompressed. Only a portion of the datasets will be needed for this application and therefore DBpedia's division of files was preferable. Also, because DBpedia provides a mapping between Freebase and its entities, it would be possible in the future to combine the two efforts. In an analysis of Wikipedia pages, [20] found that a page is modified an average of 1.2 times a month. This shows that a DBpedia dump of Wikipedia knowledge does not change drastically over a short period of time. Analyzing the types of relationships between

entities shows that they are usually long lasting and will not change frequently, e.g. The fact that an actor stared in a movie will not change.

DBpedia's downloadable datasets are provided in n-triple format. Version 3.4 which was created in September 2009 was used. Below is an example of a statement that would be found in the DBpedia dataset:

```
<http://dbpedia.org/resource/Atlanta_Falcons>
<http://dbpedia.org/ontology/FootballTeam/city>
<http://dbpedia.org/resource/Atlanta%2C_Georgia> .
```

The above statement asserts that the `Atlanta Falcons` are located in the `city` of `Atlanta, Georgia`. This application focuses on the associations between resources, and the distance of these associations, and therefore the application will process the above statement as: The `Atlanta Falcons` resource has an association to the `Atlanta, Georgia` resource via the property `FootballTeam/city` and the distance of this association is one hop.

Five files from the DBpedia dataset were used:

- infobox data (the boxes of information found in the top right on a Wikipedia page),
- person data,
- DBpedia `rdf:type` definitions,
- YAGO `rdf:type` definitions,
- redirects (variations in spelling, etc.)

The associations between resources are defined in the person and infobox data files (10 million triples combined). The other files are used for secondary purposes during the filtering and mapping steps described below. Two files providing type definitions (`rdf:type`) for entities were used: DBpedia and YAGO[1], which is another large semantic knowledge base (8.9 million triples total). The redirects file (3.8 million triples) which maps multiple URIs to

[1]http://www.mpi-inf.mpg.de/yago-naga/yago/

a single URI, will be used to provide aliases, e.g. there are 103 different URIs in the dataset for the Nintendo Wii console based on different spellings and naming conventions. There are 22.7 million triples in these files and a combined file size of 3 GB when uncompressed.

Chapter 4

Pre-Processing & Relation Discovery

Due to the large size of the dataset, several pre-processing steps were performed that focused on computing associations and removing unnecessary data. Without these steps, the time required to find shortest-path relationships for several pairs of entities would have been a major constraint to the possibility of running this application in realtime. For example, the DBpedia Relationship Finder[1], discussed in [2], discovers relationships in realtime with a small amount of pre-processing and is an excellent way to view and analyze relationships between entities. However, this would not work in realtime when many pairs of entities are examined at once. Therefore these pre-processing steps are vital to the performance of the application.

Shortest path ρ-Path and ρ-Join Associations for each pair of entities are found [21, 22]. For two entities, `e1`, `e2`, a ρ-Path association is one where there is a one-way path from `e1` to `e2` or vice versa. A ρ-Join association is where there is an intermediate entity, `e3`, and there is an association from `e1` and `e2` to `e3` or vice versa. ρ-Iso associations, defined as for any two pairs of associations, the relationships between each intermediate entity is the same for both pairs, are not found.

4.1 Filtering and Tagging

A filter was used to remove triples from the dataset files that are not applicable, including triples that contain literal values, e.g. birth dates, album titles or links to external web pages and disambiguation pages, which as described in [24] provide links, i.e. associations, to many

[1] http://relfinder.dbpedia.org/relfinder.html

varying entities that have a similar name. There were also many entities that do not have a type definition from either DBpedia or YAGO, such as lists. These lists cover varying topics, e.g.

```
List of discontinued Volkswagen engines between 90 - 115 horsepower,
```

and point to other entities in the dataset. After examination, it was determined that the value of each list would have to be determined on a case by case basis; this was not trivial and therefore they were removed from the dataset. In the case above, two engines contained in the list will also be related by a common manufacturer and the omission of the list will not result in a loss of knowledge in the dataset. While traversing the `rdf:type` definition files, each entity that does have type information is tagged with a numeric identifier, which will be used in mapping multiple URIs to the same entity (see Section 4.2 for more information).

After filtering, the files contained 17.2 million triples and a combined file size of 2.3 GB (uncompressed).

4.2 Mapping

In Wikipedia (and therefore in DBpedia), each entity has one page describing it, identified by a URI, and zero or more URIs that will redirect to this main URI. These redirect URIs can be used to provide aliases and make the system more robust by providing multiple terms that reference the same entity. The DBpedia redirects file contains n-triple statements as follows:

```
<http://dbpedia.org/resource/Wii_Console>
<http://dbpedia.org/property/redirect>
<http://dbpedia.org/resource/Wii> .
```

In order to use aliases, a mapping is provided by a lookup table that maps URIs to an identifier. The lookup table is populated by traversing the redirects file and giving the subject of each n-triple statement the same identifier as the object of the statement. For example, in the above n-triple statement, if `<http://dbpedia.org/resource/Wii>` had been given

the identifier 4 during the tagging process, <http://dbpedia.org/resource/Wii_Console> would also now have the identifier 4.

After filtering and mapping, the lookup table, stored in MySQL, contained 3,822,455 entries that pointed to 1,407,692 resources.

4.3 The 1-Hop associations file

Armed with the lookup table for entities and the files containing associations between entities, i.e. the infobox data and person data files, a new file, called the 1-Hop associations file, is now generated containing every pair of entities that have at least one association. From a graph point of view, entities are treated as nodes and associations are undirected edges connecting nodes. The type of association is ignored, but any unwanted types of associations could be removed during the Filtering and Tagging steps (see Section 4.1). Also, entities that have more than one 1-hop association between them will only be listed once.

For example, for the following statement:

```
<http://dbpedia.org/resource/Atlanta_Falcons>
<http://dbpedia.org/ontology/FootballTeam/city>
<http://dbpedia.org/resource/Atlanta%2C_Georgia> .
```

If `lookup_id()` returns the identifier for a URI and

```
lookup_id(<http://dbpedia.org/resource/Atlanta_Falcons>) = 28454
lookup_id(<http://dbpedia.org/resource/Atlanta%2C_Georgia>) = 100751
```

The 1-Hop associations file would contain the following lines (once and only once):

```
28454 100751
100751 28454
```

The resulting file containing all entities that are connected in 1 hop contained 7,035,286 lines (representing at least 3,517,643 associations) and had a file size of 99 MB. Of the initial 1,407,692 million entities that had `rdf:type` definitions, 466,417 did not have any associations with other entities in the set. Therefore, 941,275 entities are represented in the 1-Hop association file and each entity is connected to an average of 7.00 other entities.

4.4 Relation Discovery - Finding More Associations

Due to the potential large amount of time and memory it would take to determine the association distance between two entities on the fly, it was necessary to extend the 1-Hop association file by finding all entities that are associated in longer distances. However, because the densely-connected nature of this dataset, it was not feasible to maintain all associations of longer distances, i.e. 2–4 hops, in a single file. Therefore, each entity would have `n` files from `1..n` hops that contain all entities that can be reached in that number of hops. A hierarchical directory structure was used so that the maximum number of files possible in a directory would not be exceeded.

To find associations that are longer than 1 hop, a breadth-first algorithm is used. The psuedo-code for this algorithm is:

```
INPUT: entities - list of entities
       hops     - number of hops to find associations for

1 forall e in entities do
2     associations = {};
3     associated_entities = getAssociations(e, hops-1);
4
5     forall associated_entity in associated_entities do
6         assocations.add(getAssociations(associated_entity, hops));
7
8     removeEntitiesThatCanBeFoundInLessHops(associations, hops);
9
10     if |associations| > 0 do
11         sort(associations);
12         writeFile(associations, e, hops);
```

The operation of this algorithm can be described as follows: for each entity, `e`, get all entities that are associated with `e` in `n` hops, `associated_entities`. For each entity in `associated_entities`, `associated_entity`, get all entities associated with `associated_entity` in `1` hop and add these to the set of new associations of `n` hops for `e`. Once the new associations of length `n` are found, sort the set and remove all items that can found in less hops. Then the new associations can be saved out to a file.

For this application, all associations in 4 or less hops were found. However, this algorithm can be used to find as many hops as necessary with respect to storage space constraints. Table 4.1 shows statistics for associations at different distances. It took approximately 60 days to discover all associations for the entities up to 4 hops.

Table 4.1: Statistics for Associations at different hop distances

Hops	Total Associations	Average	Minimum	Maximum	File Length
1	7,035,296	7	1	87,549	47.304 MB
2	20,496,054,170	21,774	1	195,834	141.413 GB
3	52,909,550,866	56,210	1	599,658	362.194 GB
4	271,945,607,286	288,912	1	680,184	1871.633 GB
Total	345,358,247,618	366,905	1	894,080	2.3196 TB[2]

[2]File length is not necessarily equal to disk space used, e.g. minimum size of files stored on disk on the computer used for this application was 4 KB. The total disk space used to hold the files and the directory structure containing those files was 2.4689 TB

Chapter 5

Semantic Association Ranker

Using the pre-processed information from Chapter 4, the Semantic Association Ranker application is presented that with an input of a target document and a list of documents for comparison, returns a similarity ranking for each of the documents in the list based on the closeness of named entities in the context of the target document. This chapter describes the details of this application. Figure 5.1 shows an example of ranking the New York Times[1] website in the context of CNN[2].

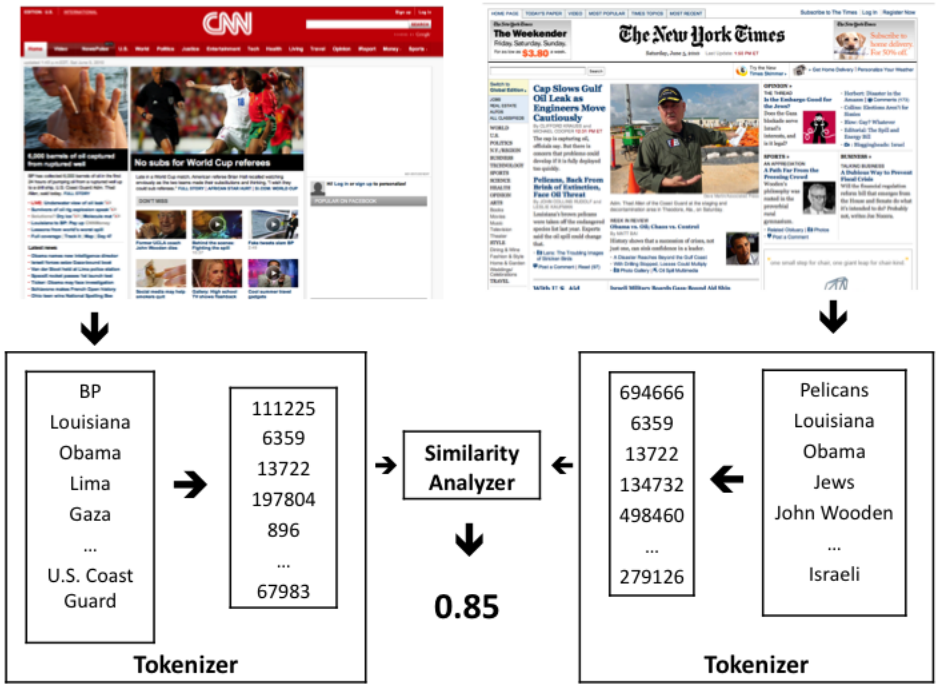


Figure 5.1: Semantic Association Ranking Example

[1]http://www.nytimes.com
[2]http://www.cnn.com

5.1 Algorithm

The Semantic Association Ranker is divided into two components: a tokenizer and a similarity analyzer. The tokenizer analyzes a document, finds entities that are found in the dataset and converts them to their identifiers. The analyzer compares two sets of tokens and returns a similarity value based on how close the tokens in each set are to each other. The psuedocode for the ranker is as follows:

```
INPUT:  document
        doc_list - A list of documents to be ranked based on similarity
OUTPUT: an ordered list of documents from the doc_list

1    tokens = find_tokens(document);
2    forall doc in doc_list do
3        doc_tokens = find_tokens(doc);
4        compute_similarity(tokens, doc_tokens);
```

5.2 Proper Noun Tokenizer

The proper noun tokenizer analyzes a document, finds entities by using the lookup table and returns a set of ids for the entities found in the document. For this application, documents are assumed to HTML pages and a HTML parser/scrubber is used to strip HTML tags and extract the text from pages before passing it to the tokenizer. However, since the input to the tokenizer is just string of text, other types of content, e.g. Microsoft Word documents and Adobe PDF documents, could be used as long as a suitable pre-processor is available to convert the content to a raw string.

The method used to find entities from the text is relatively naive and focuses on finding proper nouns. Since proper nouns will generally be capitalized, it looks for candidates by identifying sequences of capitalized words and then queries the lookup table for each of the candidates. If a candidate in not found in the lookup table, then either the entity represented by the candidate is not in the dataset, or the candidate does not represent an entity, i.e. the first word in a sentence, e.g. During, Using, Two (words used in this document).

During the ranking process, statistics for the tokenizer were taken for 10,200 pages[3]. Based on these pages, on average the tokenizer finds around 24 candidates per page; 14 of which are found in the lookup table and used for ranking. Table 5.1 shows lookup statistics for the proper noun tokenizer.

Table 5.1: Proper Noun Tokenizer Statistics Per Page

Candidates Identified	Found in Data Set	% Matched
24.147	14.071	58.272%

Also during the lookup phase, each entity is classified into one of five categories: people, places, organizations, works (e.g. books, movies, etc), and other (entities that do not match any of the other categories). The entities are classified by using the `rdf:type` definitions for the entity that are stored with the lookup table. Using these categories, documents can be classified on these more specific sets of entities, for instance a set of documents could be ranked on the closeness of the places mentioned in them and the a target document.

5.3 Searching Files for Associations

To determine if two entities have an association at a certain distance, the association files, created during pre-processing in Sections 4.3 and 4.4, are searched using a psuedo-binary search algorithm. The algorithm is not a true binary search because the number of entities in the file and their starting positions are not known, only the file length. When seeking to a position in the file, it is possible for the cursor to point to the middle of an entity and then it is necessary to seek to the beginning of the entity by decreasing the cursor's position one character at a time. The psuedo-binary search algorithm is listed below.

```
1 function hasAssociation(e1, e2, hops)
2    file = getFile(e1, hops);
3    low = 0;
4    high = |file|;
3
```

[3]Taken from the `WEBSPAM-UK2007` dataset which will be discussed in Chapter 6

```
4    while (low <= high) do
5      position = (low + high) / 2;
6      token = getToken();
7      if alreadyFound(token) return NO MATCH;
8      entitiesFound.add(token);
9
10     if (e2 < token) high = position - 1;
11     else if (e2 > token) low = position + 1;
12     else return MATCH;
13
14   return NO MATCH;
```

In the algorithm, `getToken()`, line 6, seeks the cursor to `position` and then backtracks until the beginning of a token. Because the cursor could be moved, a list of entities that have already been found in the file is maintained, lines 7 and 8, so `NO MATCH` can be returned if an entity is found twice. Without the list, the algorithm could run indefinitely because `getToken()` could repeatedly return the same token and leave the file cursor, `position`, in the same location.

Using the `hasAssociation()` algorithm, an algorithm to determine the shortest distance between two entities is given as:

```
1 function hopsSeparating(e1, e2)
2    hops = 1;
3    while hops <= MAX_HOPS do
4        if hasAssociation(e1, e2, hops) return hops;
5        hops++;
6
7    return MAX_HOPS + 1;
```

5.4 Similarity Analyzer

The similarity analyzer takes two sets of tokens provided by the proper noun tokenizer (see Section 5.2) and returns a similarity value between `0.0` and `1.0`. Two methods can be used to analyze the sets and return a similarity value: shortest distance and average distance.

5.4.1 Shortest Distance Similarity

The shortest distance similarity measure is computed by finding the average shortest distance between two sets. The algorithm to generate an average shortest value for two sets is as follows:

```
1 function shortest_similarity(set1, set2):
2    value = 0
3    foreach element1 in set1 do
4        if element1 in set2: continue;
5        min = MAX_HOPS + 1;
6        foreach element2 in set2 do
7            hops = hopsSeparating(element1, element2);
8            if (hops < min) min = hops;
9        value += min;
10
11   avg_value = value / |set1|;
12   normalized_value = avg_value / (MAX_HOPS + 1);
13   return 1 - normalized_value;
```

The `hopsSeparating` function, defined in Section 5.3, returns the number of hops separating two entities. Some key properties of this algorihtm include:

- If two entities are the same, `hopsSeparating` will return `0`
- If there are no associations between two entities within `MAX_HOPS`, `hopsSeparating` will return `MAX_HOPS + 1`[4]
- If there is no relationship between any of the entities in the set, the resulting similarity value will be `0.0`
- If `set1` $\subseteq$ `set2`, then the resulting similarity value will be `1.0`[5]

[4]For this application, `MAX_HOPS = 4`, so `hopsSeparating` will return `5` if there is no association between two entities.

[5]If `set1` and `set2` are disjoint, then `shortest_similarity(set1, set2) = shortest_similarity(set2, set1)`.

5.4.2 Average Distance Similarity

Instead of using the shortest distance separating entities, the average distance similarity measure computes the average of the average distance between each entity for two sets. The algorithm is as follows:

```
1 function avg_similarity(set1, set2):
2    value = 0.0
3    foreach element1 in set1 do
4        distance = 0;
5        if element1 in set2: continue;
6        foreach element2 in set2 do
7            distance += hopsSeparating(element1, element2);
8        avg_distance = distance / |set2|;
9        value += avg_distance;
10
11   avg_value = value / |set1|;
12   normalized_value = avg_value / (MAX_HOPS + 1);
13   return 1 - normalized_value;
```

The same properties listed under the shortest distance similarity algorithm (see Section 5.4.1) hold for this algorithm, with the additional property that:

$$\texttt{shortest\_similarity(set1, set2)} \geq \texttt{avg\_similarity(set1, set2)}, \forall \texttt{ set1, set2} \tag{5.1}$$

5.4.3 Similarity Example

Figure 5.2 shows an example of associations between entities found on two pages. The dashed edges are weighted with the minimum number of hops required to connect the two entities and the intermediate nodes are not shown. If there is no edge between entities, then the weight of the association is 5. Table 5.2 shows the values that would be calculated when comparing the similarity of $\texttt{P}_1$ to $\texttt{P}_2$ for the shortest distance and average distance similarity measures. The three E columns, $\texttt{E}_1$, $\texttt{E}_2$, $\texttt{E}_3$, show the distance values for the associations for each entity in $\texttt{P}_2$. The normalized distance ensures that the distance value is between 0.0 and 1.0 where 1.0

would mean an average distance of 5. The overall value is `1 - NORMALIZED_DISTANCE`, so that a higher overall value means a lower distance. As the table shows, the average distance method is effected by longer associations or one's greater than `MAX_HOPS`.

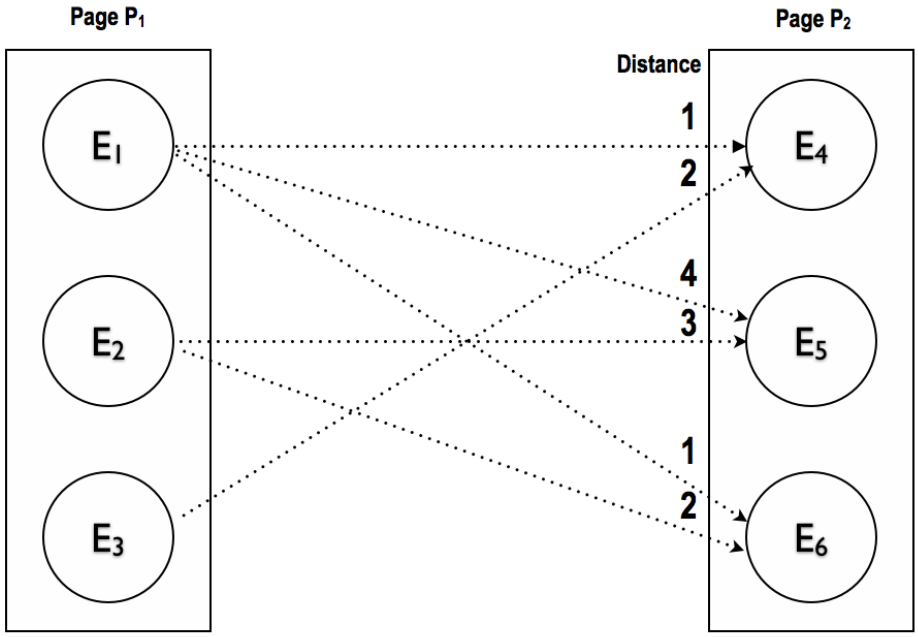


Figure 5.2: Example of Semantic Associations between Pages

Table 5.2: Simliarity Values Example

Method	E_1	E_2	E_3	Avg Distance	Normalized Distance	Overall Value
Shortest	1	2	2	1.667	0.333	0.667
Average	2	3.33	4	3.110	0.622	0.378

Chapter 6

Evaluation and Results

To test the effectiveness of the Semantic Association Ranker application, a portion of the `WEBSPAM-UK2007`[1] data set was used [25]. [26] highlights the fact that many analytical spam research papers used a biased set of data, while the `WEBSPAM-UK` dataset is a large, clean, i.e. has a small amount of classification errors, uniform, broad, i.e. in the amount of spam techniques, and openly available.

A set of training and testing data is available for download with URLs in the set being classified into one of three categories: *nonspam, spam, or undecided*. Deciding whether a page is spam or not is a subjective issue. Pages that are thought of as spam by some people may be useful to others. In the case of the `WEBSPAM-UK` data set, spam pages are defined as pages "used for spamming or receives a substantial amount of its score from other spam pages" [26]. [27] also gives two valuable insights for spam pages: "any attempt to deceive a search engine's relevancy algorithm" and "anything that would not be done if search engines did not exist". Figure 6.1 shows an example of a page that is classified as spam in the `WEBSPAM-UK2007` dataset.

The `WEBSPAM-UK2007` dataset contains links to more than 140,000 URLs under the `.uk` domain and the testing and training data made available classifies 6,479 of these pages. Of the 6,479 pages, 794 were no longer available and they were removed from the data set. Table 6.1 contains a breakdown of the classifications from the testing and training sets.

The evaluation data set was generated by randomly choosing a page that was classified as spam from the `WEBSPAM-UK2007` testing and training data as our input page and then

[1] http://www.yr-bcn.es/webspam/datasets/uk2007/

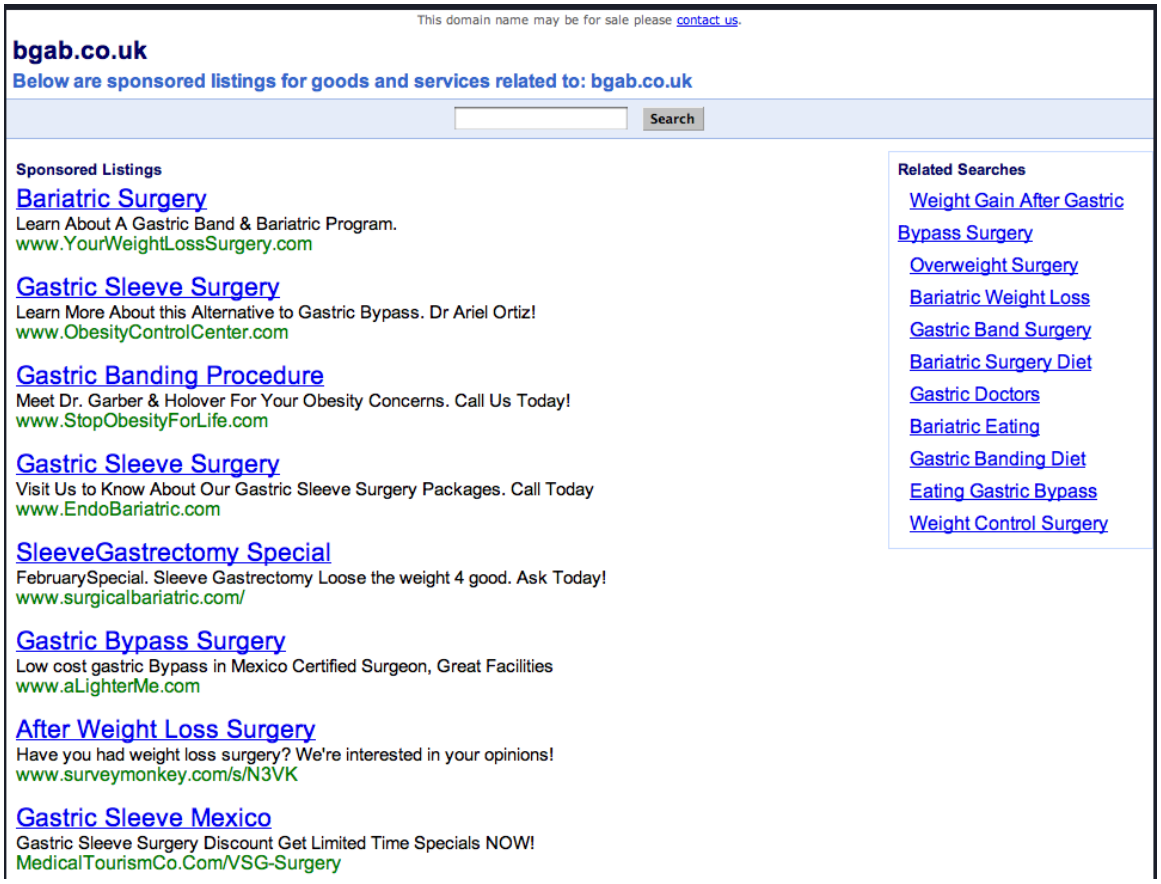


Figure 6.1: Example of a Page Classified as Spam by the WEBSPAM-UK2007 Dataset

Table 6.1: Spam Classification for the WEBSPAM-UK2007 Testing and Training Data Sets

Data Set	NonSpam	Spam	Undecided	Total
Original	5,709	344	426	6,479
Pages Actually Available	4,915	237	315	5,467

compiling a list of 100 test nonspam (including those classified by `WEBSPAM-UK2007` as undecided) and spam URLs as the input URLs to rank in the context of the input page. [12] found that around 13% of English pages were spam, but only 5% of pages under the `.uk` domain were spam. An analysis of the pages in this dataset also showed that 5% were classified as spam. Since the `WEBSPAM-UK2007` classification set is a large, unbiased set of pages, as well as being one of the few publicly available, and it is surmised that the focus and content of spam pages does not change significantly across domains, this set of classified pages is assumed to be a suitable set for the testing of spam recognition. The 100 URLs are broken down into 95 randomly chosen nonspam URLs and 5 randomly chosen spam URLs from the combined testing and training classifications.

This entire process was repeated 100 times and the resulting values were analyzed to see how well the semantic ranking methods rank the spam pages versus nonspam. Of the 100 iterations run, containing 9,500 nonspam and 500 spam pages, 18 times no entities were found on the spam input page. To evaluate the Semantic Association Ranker ranking mechanisms, the following analysis was undertaken:

- The ranking position of spam pages versus nonspam pages
- The average ranking score of spam versus nonspam pages

Figures 6.2 and 6.3 display the frequency of ranked positions of spam pages (from 1 to 100) for the shortest distance and average distance methods respectively. The shortest distance method performed slightly better than the average distance method. Spam pages were ranked in the top-5 18% and 15% for the shortest and average distance methods respectively. For all positions, spam pages were ranked number 1 most often, 26 and 22 times (out of 82) for the shortest and average distance methods respectively. There is no other noticeable pattern of the ranking of spam pages except that no spam pages were ranked lower than 82nd, but at such low ranking levels, the similarity values are all close to 0.0. Also, due to differences

in the purpose of spam pages, e.g. drugs, celebrities, etc., the fact that some spam pages are not related to other pages is to be expected.

A closer analysis of spam pages ranked in the first position in reference to a given spam page, it was found that almost all pages ranked in the first position also had a similarity value close to `1.0`. In almost all cases, the pages were from a common domain name parking company, such as Trigfega LTD[2] or Sedo[3]. In other cases, the input spam page only had a few identified tokens that were very common, e.g. United Kingdom, and therefore other pages in the test set that also had those same 1 or 2 tokens had a very high similarity score.

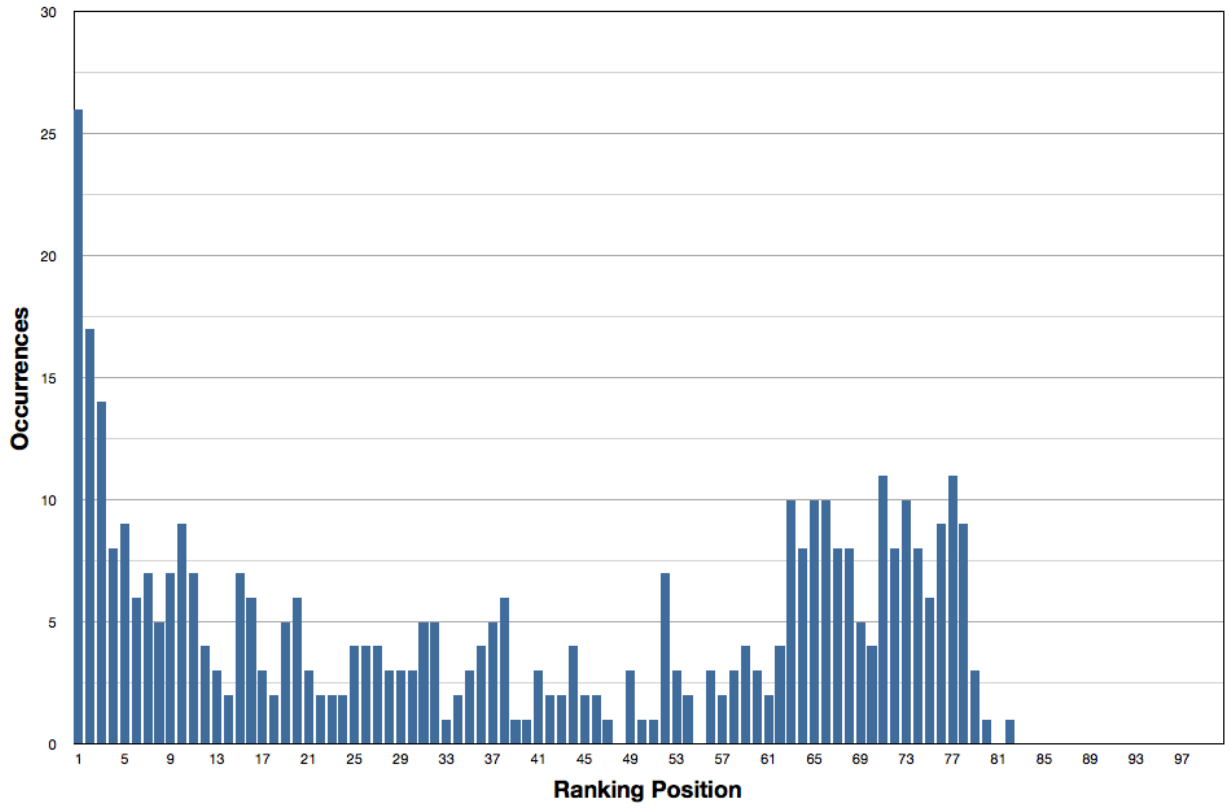


Figure 6.2: Ranking of Spam Pages Using Shortest Distance Similarity Rank

[2]http://www.trifega.com
[3]http://www.sedo.co.uk

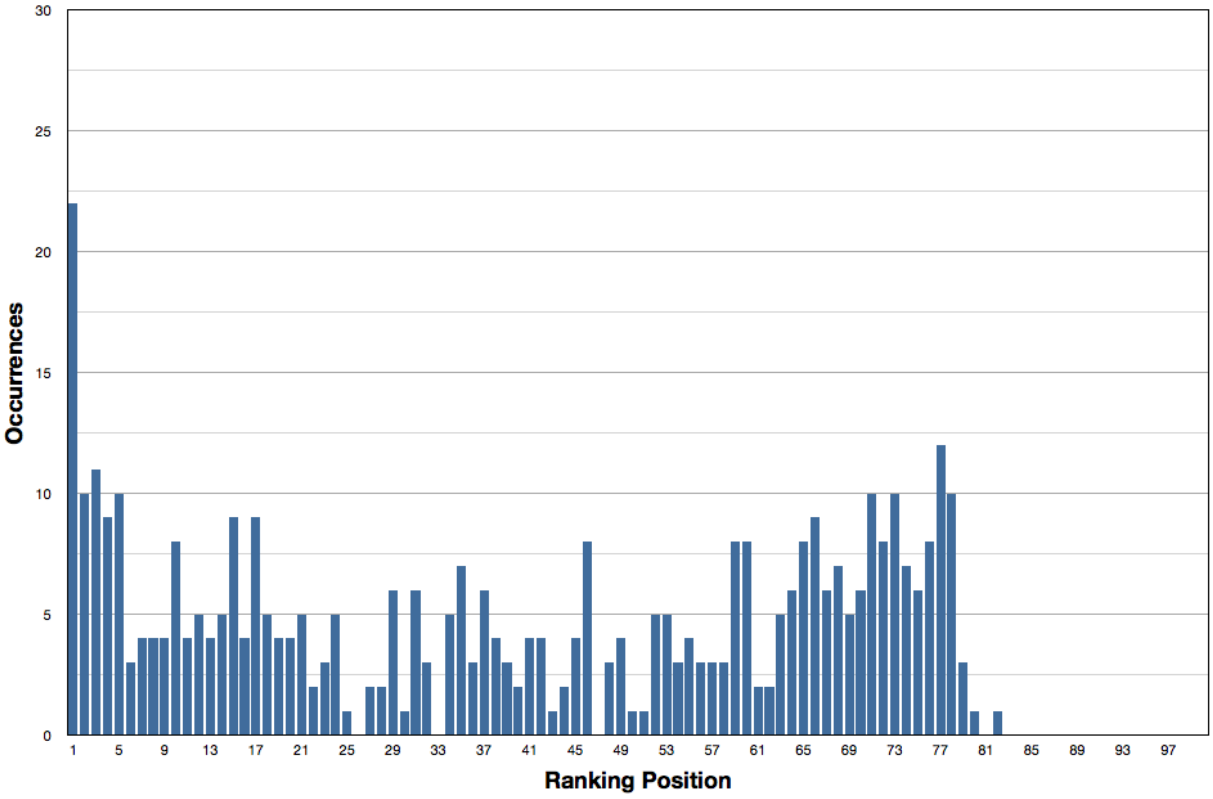


Figure 6.3: Ranking of Spam Pages Using Average Distance Similarity Rank

6.1 Shortest Distance Similarity Values Evaluation

The ranking values of spam and nonspam pages were also analyzed to see if any differences or patterns arose. Figures 6.4 and 6.5 display the percentage of classification values using the shortest distance similarity method for spam and nonspam pages respectively. The best fit line, represented by red in all the figures, is very similar for both spam and nonspam showing that there is not a large deviation in values between spam and nonspam pages using this method. In both cases, there is a large percentage of zero similarity (46.5% for both sets) and 90% of pages have a similarity value of less than `0.53` for spam and `0.5` for nonspam. Both spam and nonspam pages do not have a high or perfect similarity often. Spam pages had a `1.0` similarity 2.6% and nonspam around 1.5%. Table 6.2 displays statistics for these sets. While spam pages do have a higher average similarity value, the variance between the values shows that in this case, this would not be a reliable method to differeniate spam from nonspam.

Table 6.2: Similarity Values Statistics to a Spam Page Using Shortest Distance Method for Spam and Nonspam pages

Set	Average	Variance	STDEV	90th Percentile
Spam	0.2102	0.0654	0.2558	0.533
Nonspam	0.1884	0.0480	0.2231	0.500
All	0.1895	0.0506	0.2249	0.500

Since a large percentage of pages exhibited a zero similarity value, Figures 6.4 and 6.5 were recreated on a log scale in Figure 6.6. As shown in the figure, the similarity values for spam and nonspam at values other than `0.0` do not exhibit any major variance.

6.2 Average Distance Similarity Values Evaluation

The average distance similarity method exhibited similar overall results to the shortest distance method, but with lower overall values. Figures 6.7 and 6.8 display the percentage of classification values using the average distance similarity method for spam and nonspam

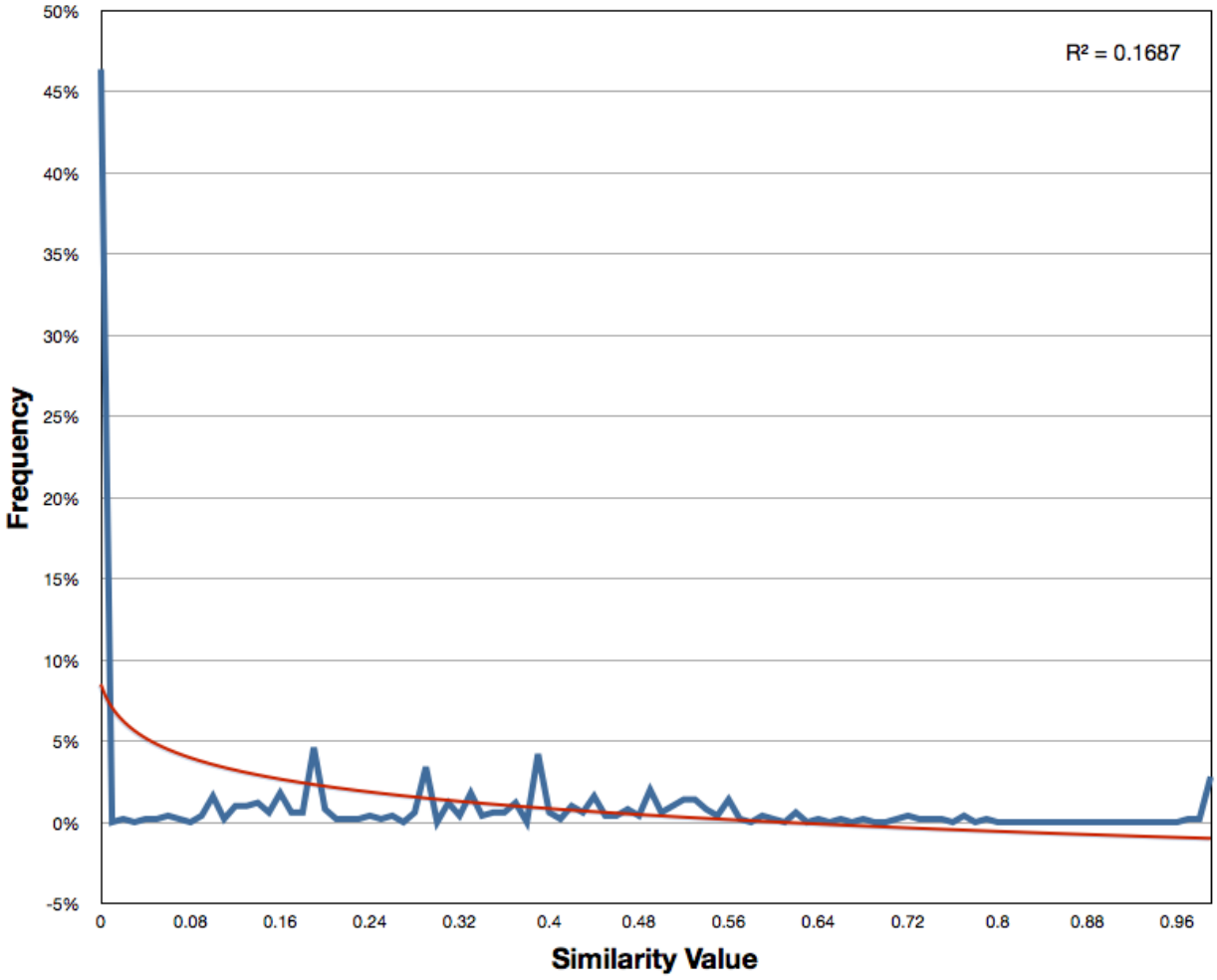


Figure 6.4: Frequency of Similarity Values Using **Shortest Distance** Similarity For **Spam Pages** with a **Spam Page** as Input

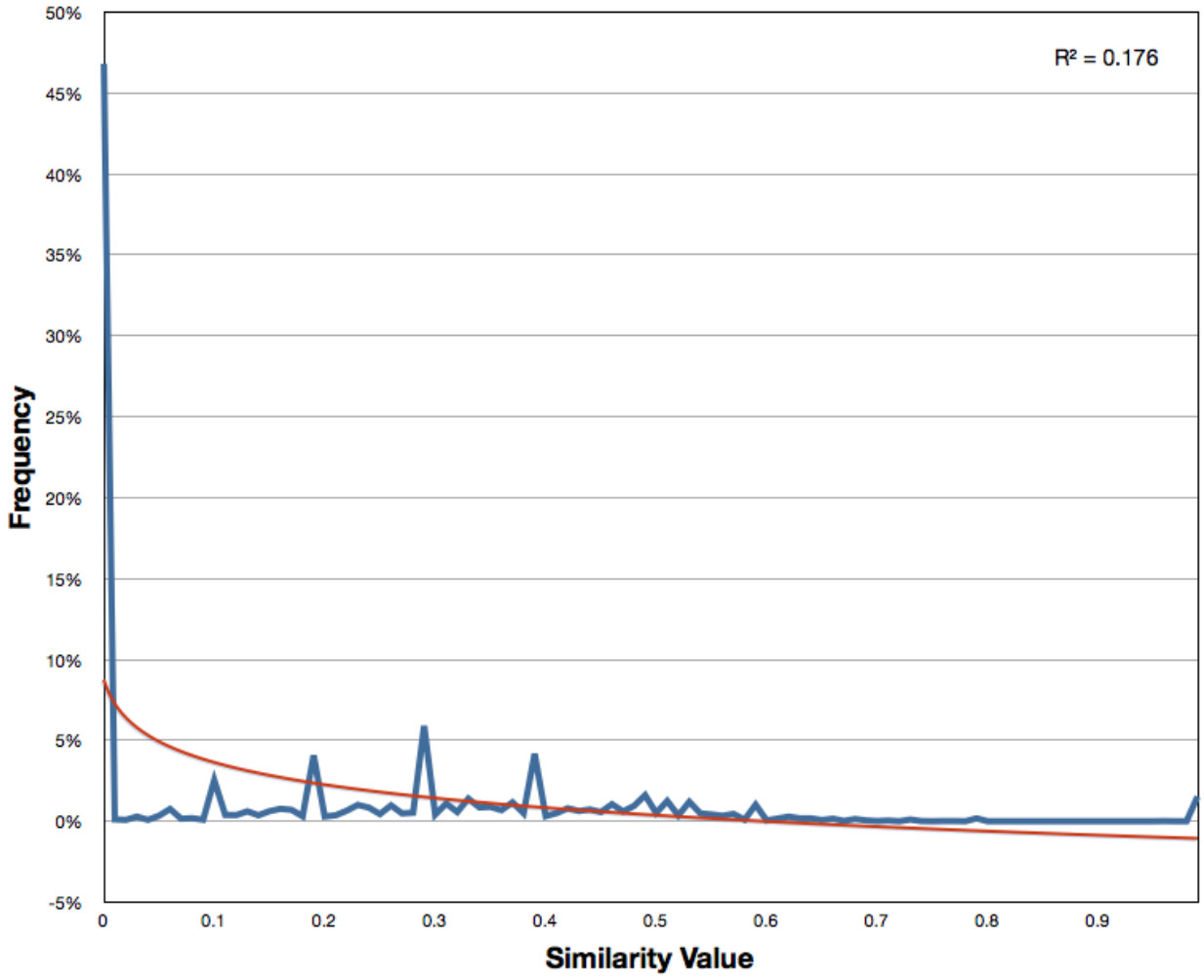
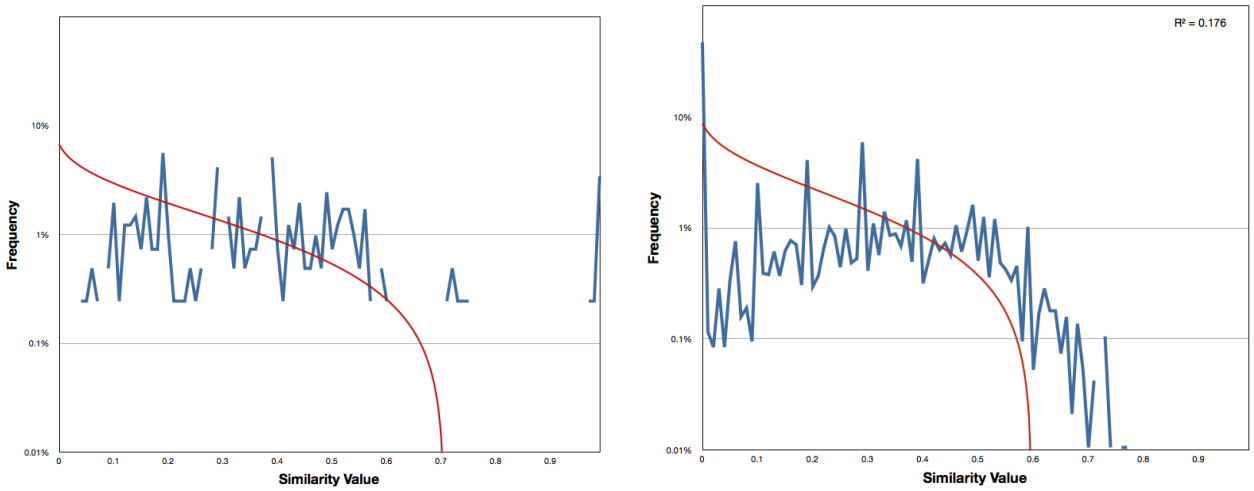


Figure 6.5: Frequency of Similarity Values Using **Shortest Distance** Similarity For **Non-Spam Pages** with a **Spam Page** as Input



(a) Spam (b) NonSpam

Figure 6.6: Shortest Distance Similarity on a Log Scale (Figures 6.4 and 6.5)

pages respectively. As with the shortest distance method a large percentage of zero similarity occurs (46.6% for both sets) and 90% of pages have a similarity value of less than `0.258` for spam and `0.243` for nonspam. Also like the shortest distance method, spam pages had a `1.0` similarity 2.6% and nonspam around 1.5%. The same conclusion is reached for this method, as with the shortest distance method, that no significant statistical difference was identified between spam and nonspam pages with this method. Table 6.3 displays statistics for these sets.

Table 6.3: Similarity Values Statistics to a Spam Page Using Average Distance Method for Spam and Nonspam pages

Set	Average	Variance	STDEV	90th Percentile
Spam	0.1158	0.0374	0.1934	0.2579
Nonspam	0.0966	0.0236	0.1535	0.2426
All	0.0975	0.0243	0.1558	0.2439

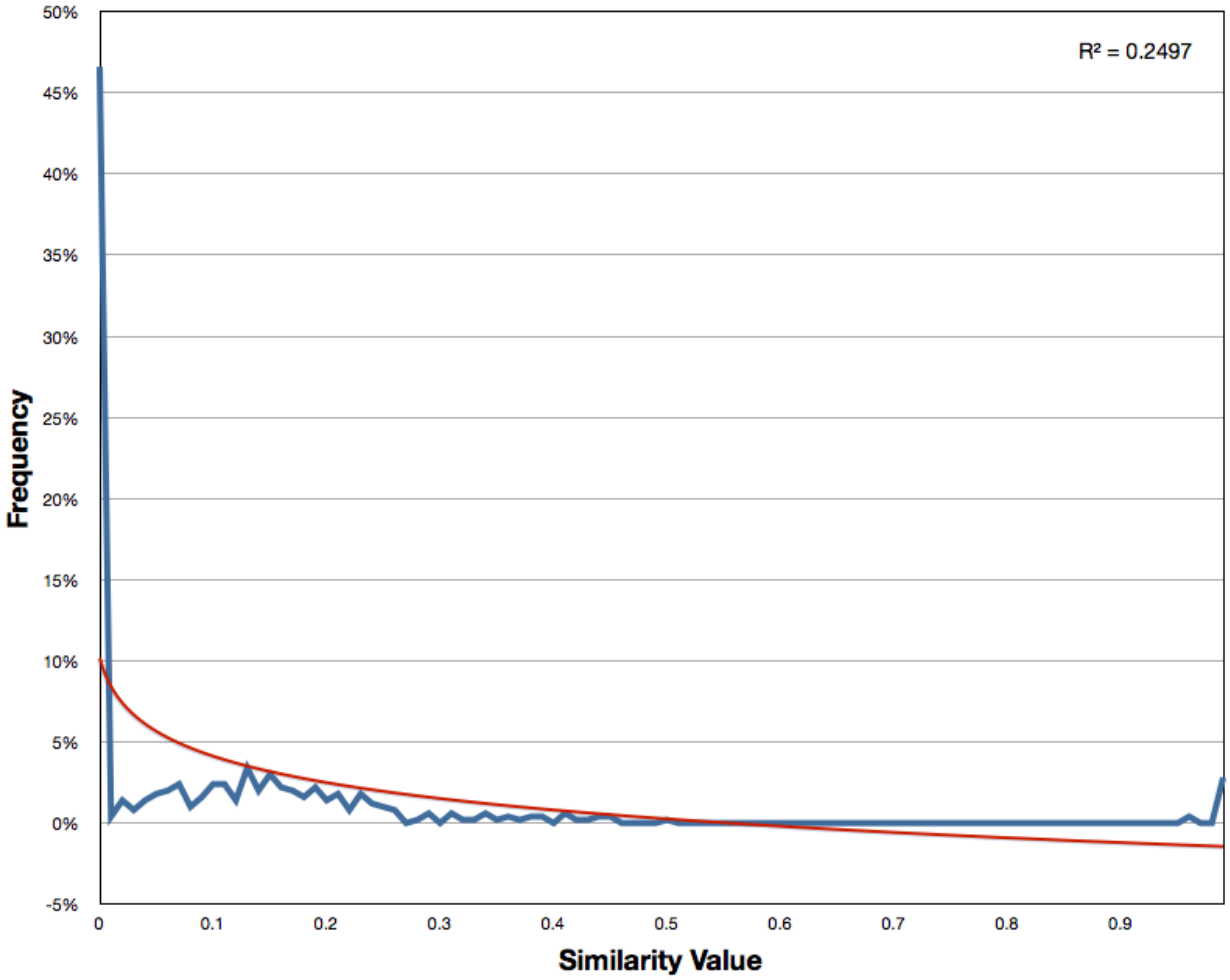


Figure 6.7: Frequency of Similarity Values Using **Average Distance** Similarity For **Spam Pages** with a **Spam Page** as Input

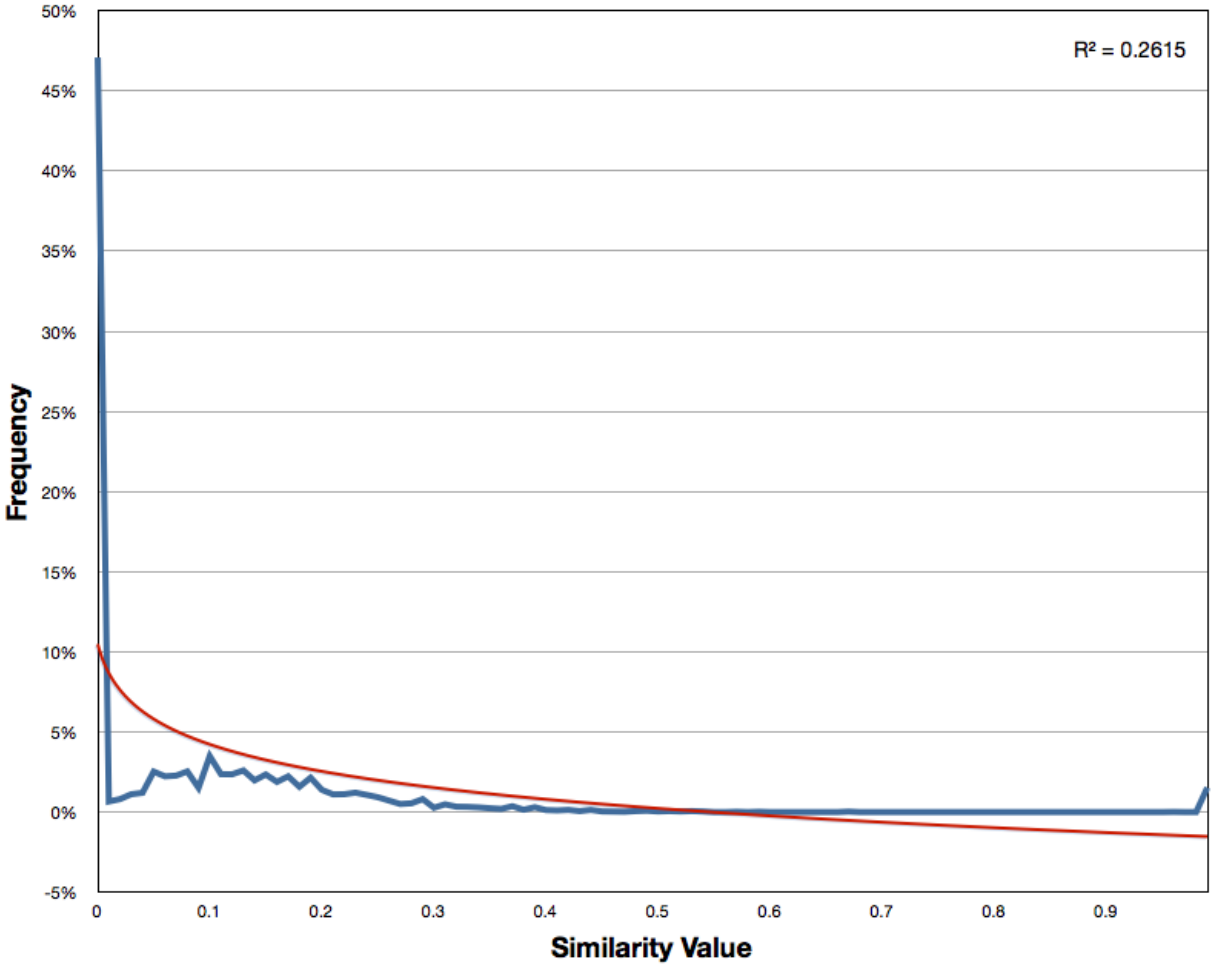
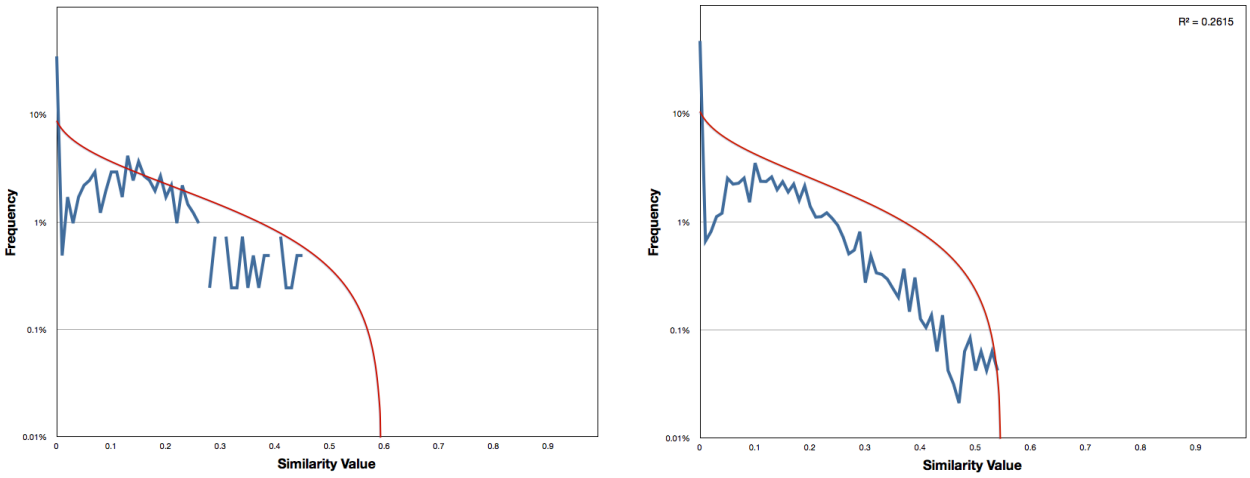


Figure 6.8: Frequency of Similarity Values Using **Average Distance** Similarity For **Non-Spam Pages** with a **Spam Page** as Input



(a) Spam (b) NonSpam

Figure 6.9: Average Distance Similarity on a Log Scale (Figures 6.7 and 6.8)

As with the shortest distance method, a log scale representation of Figures 6.7 and 6.8 were recreated in Figure 6.9. The similarity values for spam and nonspam at values other than `0.0` also do not exhibit any major variance.

6.3 Using a NonSpam Page As Input

For comparison, another test set was generated to analyze the relation of nonspam pages to spam and other nonspam pages for both shortest and average distance methods. The test data set makeup was identical to the previous set, i.e. 100 sets of 95 nonspam and 5 spam pages, with the exception that the input page was classified as nonspam by the `WEBSPAM-UK2007` dataset. Figures 6.10 and 6.11 display the results of using the shortest distance method to compare a nonspam page with spam and nonspam pages, respectively.

As shown in Table 6.4, nonspam pages did have a slightly higher average ranking than spam pages, which was the opposite of the previous test, but once again there are no statistically significant patterns that occur to reliably differeniate spam from nonspam pages.

Once again, there is a large percentage of zero similarity (45% for both sets) and the 90% ranking of pages is close to those with a spam page as input, `0.5` for both spam and nonspam. Both spam and nonspam pages do not have a high or perfect similarity often. Spam pages had a `1.0` similarity 0.4% and nonspam around 0.9%.

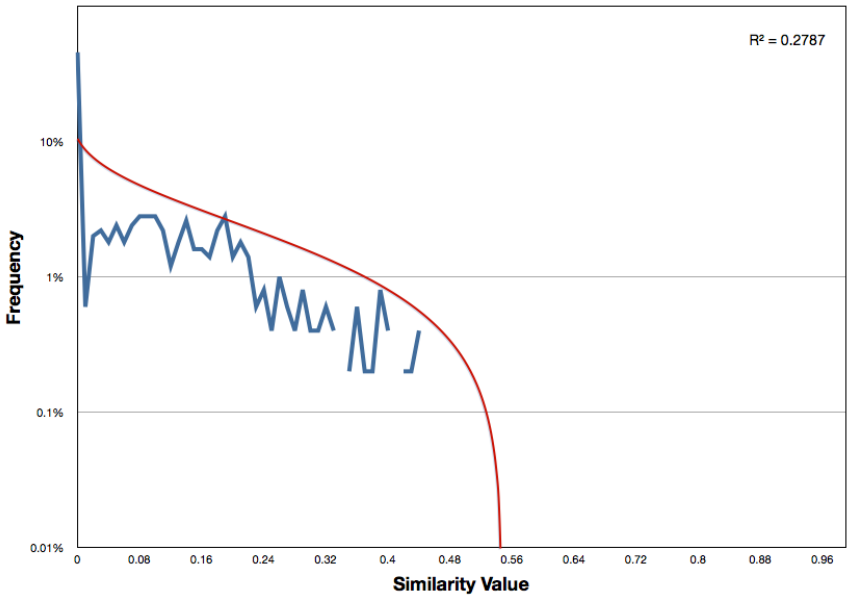


Figure 6.10: Frequency of Similarity Values Using **Shortest Distance** Similarity For **Spam Pages** with a **NonSpam Page** as Input (Log Scale)

Table 6.4: Similarity Values Statistics to a NonSpam Page Using Shortest Distance Method for Spam and Nonspam pages

Set	Average	Variance	STDEV	90th Percentile
Spam	0.1778	0.0445	0.2110	0.5
Nonspam	0.1882	0.0486	0.2204	0.5
All	0.1877	0.0484	0.2200	0.5

The average distance similarity method was also analyzed. Figures 6.12 and 6.13 show the results for a nonspam page against spam and nonspam pages, respectively. As with the shortest distance method, nonspam pages did have a slightly higher average ranking than spam pages, but as has been the case with all of the tests, there are no statistically significant

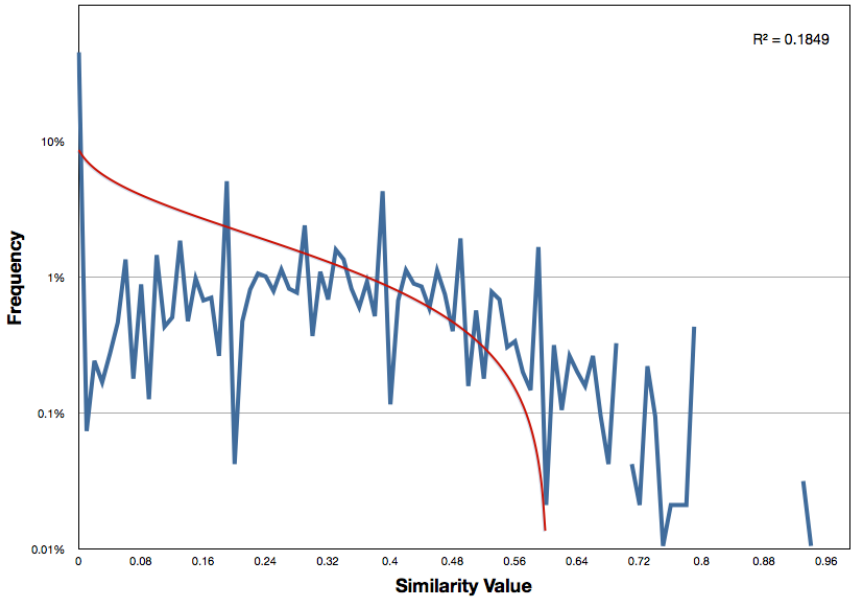


Figure 6.11: Frequency of Similarity Values Using **Shortest Distance** Similarity For **Non-Spam Pages** with a **NonSpam Page** as Input (Log Scale)

patterns that occur. As shown in Table 6.5, the 90% is similar to the spam page test with the average distance method, `0.244` for spam and `0.263` for nonspam.

Table 6.5: Similarity Values Statistics to a NonSpam Page Using Average Distance Method for Spam and Nonspam pages

Set	Average	Variance	STDEV	90th Percentile
Spam	0.0913	0.0172	0.1310	0.244
Nonspam	0.0984	0.0218	0.1477	0.263
All	0.0981	0.0216	0.1469	0.261

6.4 Speed Evaluation

Overall Speed of ranking was also analyzed. It took almost 15 minutes to compare and rank each set of 100 pages against the target page, which equates to 9 seconds for a page to page comparison. The time required to rank pages using these association files would be a

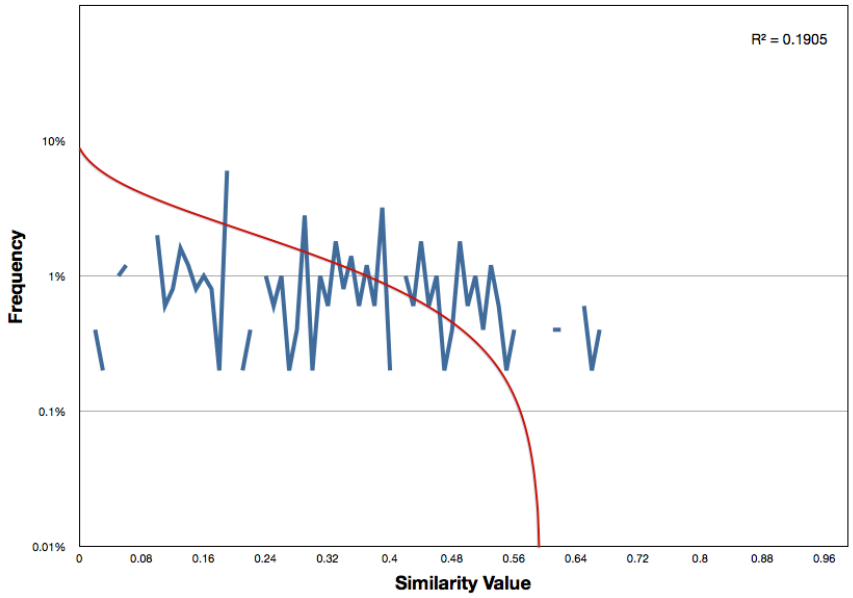


Figure 6.12: Frequency of Similarity Values Using **Average Distance** Similarity For **Spam Pages** with a **NonSpam Page** as Input (Log Scale)

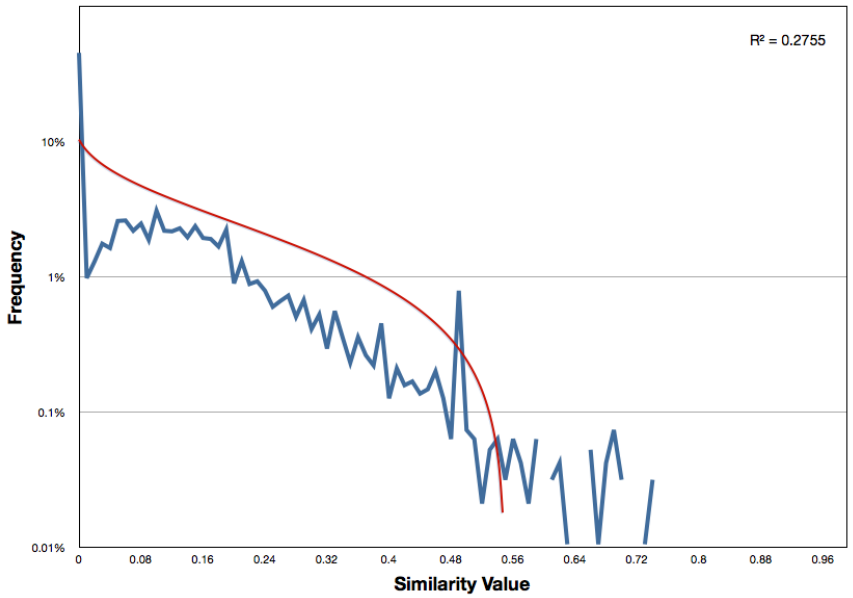


Figure 6.13: Frequency of Similarity Values Using **Average Distance** Similarity For **Non-Spam Pages** with a **NonSpam Page** as Input (Log Scale)

limitation to it being a real-time application. Other options are now analyzed that aim to improve speed and reduce disk space required.

Chapter 7

Improvements Using Bloom Filters

A Bloom filter is a probabilistic, compact data structure, originally devised by Burton H. Bloom in [9], that can be used to query for membership in a set. A Bloom filter makes use of a bit vector, of length `m`, which contains the membership information for the set. The filter is first trained by adding each of the set members as shown in Figure 7.1[1]. During training, the following procedure occurs: for each member in the set, run `k` hash functions on the member to output `k` locations in the bit vector and for set the bit to `1` at each of these locations. It is possible for the bit located at a location to already be `1` due to hash conflicts from previous insertions and in this case, the bit is still left as `1`. The time required for each insertion is *O(k)*.

Once the filter has been trained, querying for membership in a filter is a similar process. Given a token to test for membership, perform the same `k` hash functions and retrieve `k` bit vector positions. Perform a bit-wise AND on the bits at these positions and if the result is `1`, then the token is in the set. Otherwise, the token in not in the set. The time required to check for membership is also dependent on the running time of the hash functions and is therefore also *O(k)*. Figure 7.2 illustrates the process of of querying for set membership for a given token.

If the token was in the initial set, then the Bloom filter will always return a true value for this token, i.e. *no false negatives*. However, it is possible that during training previous tokens set all of the bit vector positions output by the `k` hash functions for this token to `1`, indicating that the token is a member in the set, when it really is not, i.e. a *false positive*. The

[1]In Figures 7.1 and 7.2, the circular bullets equate to an unknown value at that position in the bit vector, i.e. either 1 or 0.

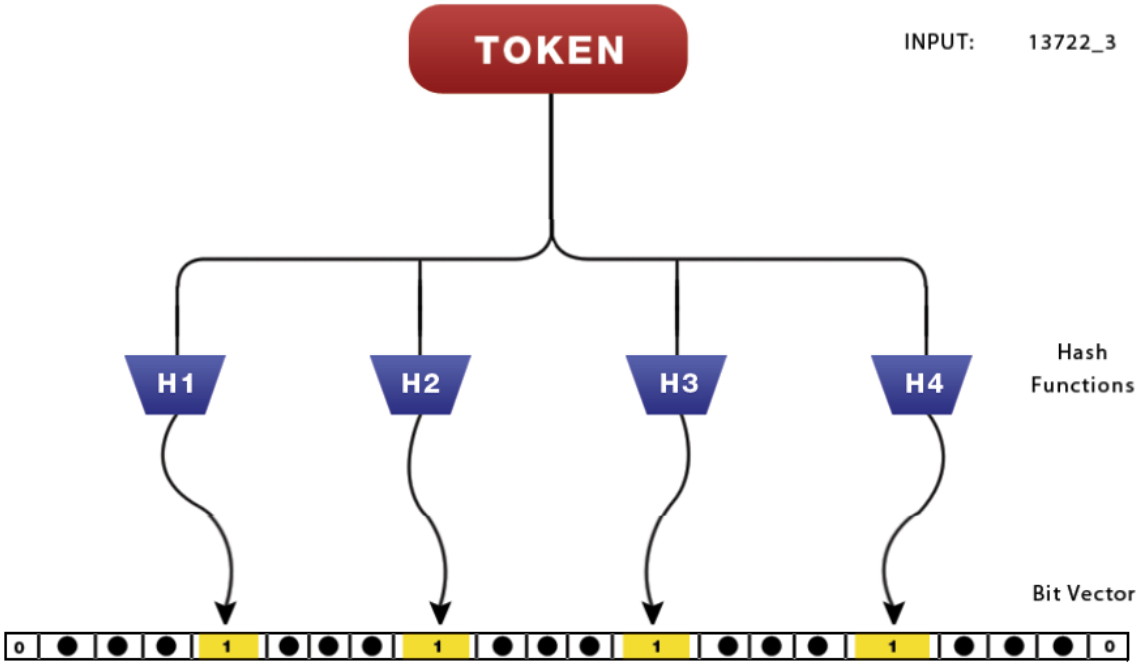


Figure 7.1: Training a Bloom Filter

false positive probability for a filter is determined by the size of `k`, the size of the bit vector, denoted by `m`, and the number of tokens inserted into the filter, denoted by `n`. Assuming perfect hash functions, then the probability of false positives can be lowered by increasing the size, `m`, of the bit vector. As defined in [11], the false positive probability, `P`, can be given as:

$$P = (1 - (1 - \frac{1}{\mathbf{m}})^{\mathbf{n*k}})^{\mathbf{k}} \tag{7.1}$$

Assuming that an optimal number for `k` is used and `n` and an expected false positive percentage, `P` is known ahead of time, a size for the bit vector, $\grave{m}$, that should produce a percentage close to the expected false positive value, can be found using the following

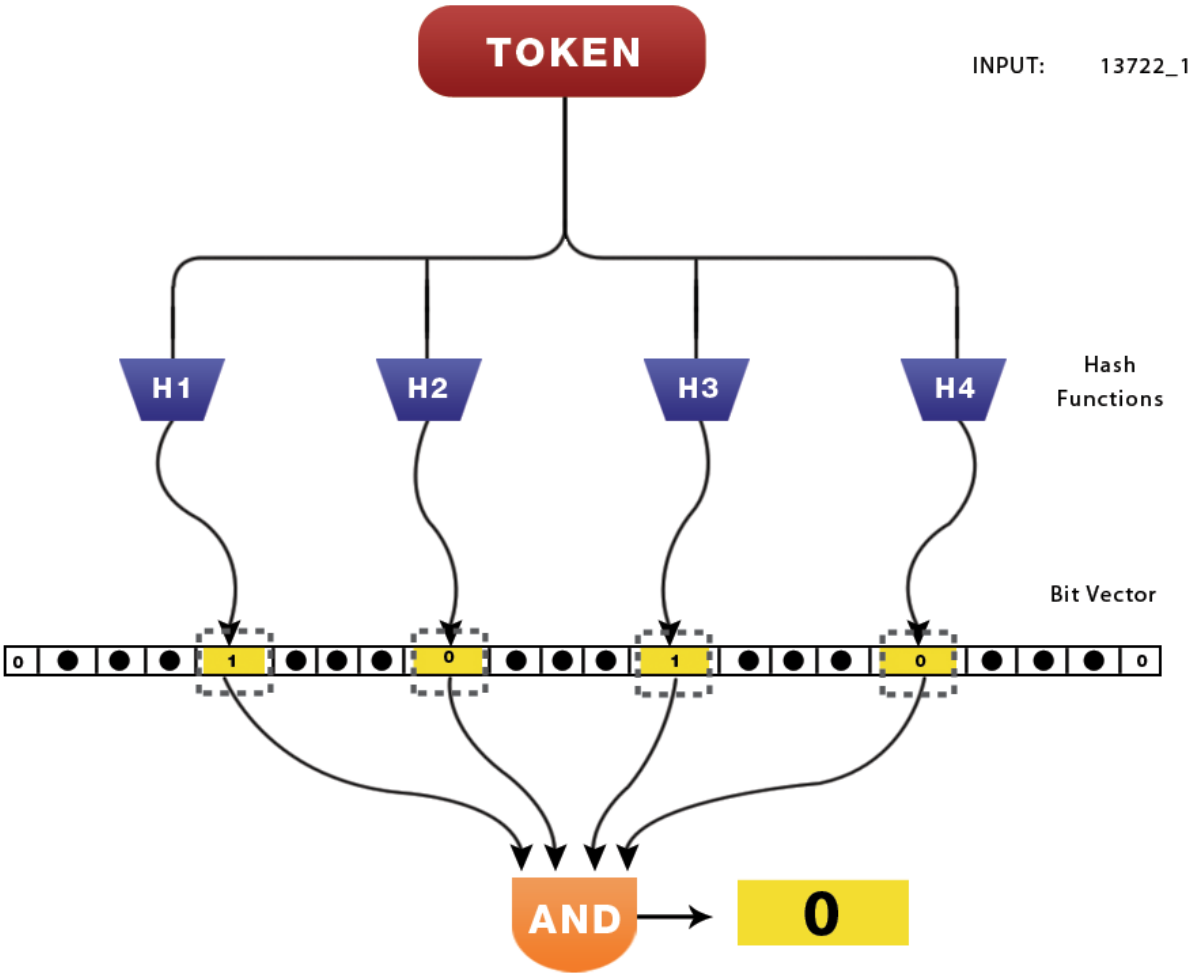


Figure 7.2: Querying for membership in a Bloom Filter

equation, defined in [23]:

$$\grave{m} \geq n \log_2 e * \log_2(\frac{1}{P}) \tag{7.2}$$

For example, based on Equation 7.2, if `n = 10000` and a false positive percentage close to `5%` is required, the bit vector size should be at least `62,352` bits long.

7.1 Replacing the Association files with Bloom filters

Since Bloom filters are compact and testing for membership is *O(k)*, it is hypothesized that using Bloom filters in place of the association files will improve the Semantic Association Ranker application because the time to find associations will be reduced (see Section 5.3) and the disk space required to store the filters will be less than the space required for the files (see Section 4.4).

7.1.1 Training

A Bloom filter was created for each entity in the association files, totaling 935,696 entities. Each filter contains the entities that are associated with that entity in 1–4 hops. For each association, the entity's id and distance are concatenated into a key, separated by an underscore, which is stored in the filter. For example, during training for entity `6359`, if it and `13722` are associated in `3` hops, the key `13722_3` will be stored in the filter for `6359`. To reduce the space needed while not losing any information about associations, only one side of the association is saved. This is done by only storing associations that are larger than the entity being trained, i.e. `6359_3` will not be stored in the Bloom filter for entity `13722` since that association has already been added to the filter for `6359`[2].

Seven Bloom filter sets were generated in order to test the speed and accuracy at different disk sizes. The Bloom filter sets total disk size started at 4 GB and doubled in size up to 256 GB. The `k` value for each set, i.e. number of hash functions, was determined by testing

[2]Of the 941,275 entities stored in files, 5,579 could be removed because their associations were stored in other filters, i.e. all of their associations were with entities whose id was numerical lower than theirs, therefore they were unnecessary.

the association data and choosing the `k` with the lowest false positive percentage. Table 7.1 shows basic information for each of the Bloom filter sets.

Table 7.1: Bloom Filter Sets

Set	Disk Size	K[3]	% of Files
Bloom-4	4 GB	1	0.32%
Bloom-8	8 GB	1	0.63%
Bloom-16	16 GB	1	1.27%
Bloom-32	32 GB	1	2.53%
Bloom-64	64 GB	2	5.06%
Bloom-128	128 GB	4	10.13%
Bloom-256	256 GB	8	20.25%

If the file space required to store the association files is reduced by half (since both sides of an association are stored, this results in no loss of knowledge, i.e. if entity `1` and `2` are associated, this information will be stored in both the `1` and `2` files.)[4], the Bloom filter sets would be between `0.32%` and `20.25%` of the file space taken up by the association files. Table 7.2 displays association statistics for all entities.

Table 7.2: Association Statistics for Entities

Entities	Total Associations	AVG	MIN	MAX
935,696	172,679,123,809	184,546	1	885,750

The Bloom filter sets use between `0.20` and `6.37` bits per association from smallest to largest Bloom filter sets. The low number of bits per association for the smaller Bloom filter sets is an indication that there will be a large percentage of false positives for those sets. On average, all associations for an entity use between `35,959` and `2,301,372` bits for the smaller to largest Bloom filter sets respectively. Table 7.3 displays bit vector statistics for the filter sets.

[3] `k` is the number of hash functions used.

[4] The association files will be $\approx$ 1.2344 TB if all duplicate associations are removed.

Table 7.3: Bit Vector Statistics for Bloom Filter Sets

Set	Total Bits	Bits/Association	AVG/Entity	MIN	MAX
Bloom-4	34,358,713,489	0.19900	35,959	1	172,590
Bloom-8	68,717,426,978	0.39800	71,918	1	345,178
Bloom-16	137,434,853,956	0.79591	143,836	1	690,356
Bloom-32	274,869,707,912	1.59182	287,672	2	1,380,712
Bloom-64	549,739,415,824	3.18364	575,343	3	2,761,425
Bloom-128	1,099,478,831,648	6.36728	1,150,686	6	5,522,849
Bloom-256	2,153,385,179,486	12.47044	2,301,372	12	11,045,698

7.1.2 Querying for Set Membership

Querying if two entities have an association is a simple and fast operation. The `hopsSeparating()` algorithm (defined in Section 5.3) can be now written as:

```
1 function hopsSeparating(e1, e2)
2    switch entities if e1 > e2
3    hops = 1;
4    filter = loadFilter(e1);
5    while hops <= MAX_HOPS do
6        key = e2 + "_" + hops
7        if filter.contains(key) return hops;
8        hops++;
9
10   return MAX_HOPS + 1;
```

`hopsSeparating()` will always return a value between 1 – 5. Line 2 ensures that `e1` is the entity with the smallest id and `loadFilter()` loads a Bloom filter from a file. Then, a key, generated from `e2` and the number of hops, is used to query the filter checking if that key exists. If there is an association between these two entities, then `hopsSeparating` will always return a value $\leq$ the actual number of hops[5]. If no association exists between two entities, then the majority of the time, `MAX_HOPS + 1` will be the result. However, the higher

[5]Because Bloom filters do not allow false negatives, the result will never be greater than the actual distance. However, a smaller distance than actually exists could be returned due to hash collisions.

the false positive percentage, the more likely `hopsSeparating` will return a value lower than expected.

7.2 Limitations

In the case that a new association needed to be added to the Bloom filter sets, having the association files would be necessary to recompute the associations for all entities in the range of 1-4 hops. Once all the necessary additions were computed, an assertion into the filter would be trivial to write and perform. New associations that related two entities at a shorter distance than previous found would cause an additional insert into the filter increasing the false positive probability as well. If the false positive probability for an individual filter reached a high percentage too high, the filter could be recreated with a larger bit size.

The removal of an association, however, due to the nature of Bloom filters would not be an easy solution. It is a simple matter to determine the locations in the bit vector that were affected by the insertion of an entity, but it is not correct to set each of these positions to 0 because they could have also been affected by another insertion. Clearing the value at these positions might cause the filter to incorrect state that an association does not exists when it actually does, i.e. a *false negative*, which is not allowed. The best solution would most likely be to regenerate the filters from the association files, which means that the association files would need to contain the newest information, so anytime there is an insertion into the filters, an equivalent update to the association files would be necessary.

7.3 Results

The performance of the Bloom filter sets was analyzed by using the same process for the file lookup method described in Chapter 6. The time results for only the lookup/querying for an association is shown in Table 7.4 based on 2,736,616 lookups[6].

[6]For the Bloom filter sets, this does not include the time to load the filter into memory. This information is added into the overall running time as shown in Figure 7.3.

Table 7.4: Lookups Per Second for All Data Sets

Data Set	Lookups/sec	Speed Improvement
Files	61.045	-
Bloom-4	995.965	16.3x
Bloom-8	1386.696	22.7x
Bloom-16	1788.454	29.3x
Bloom-32	2004.305	32.8x
Bloom-64	2412.982	39.5x
Bloom-128	3358.128	55x
Bloom-256	3668.745	60.1x

Figure 7.3 shows the overall performance of the filter sets and the file lookup method averaged for both the shortest and average distance methods. At least a 10x improvement overall can be seen for the Bloom filter sets over the file lookup method and as the Bloom filter sizes increase, the overall time increases as well. Smaller Bloom filter sets have a higher percentage of false positives, i.e. more than `81%`, and the largest Bloom filter set (`Bloom-256`) exhibits a false positive percentage of less than `1%` as shown in Table 7.5 for 2,736,616 lookups.

Table 7.5: False Positive Percentages for Bloom filter sets

Filter Set	False Positives	False Positive %
Bloom-4	2,238,607	81.802 %
Bloom-8	2,227,551	81.398 %
Bloom-16	2,156,755	78.811 %
Bloom-32	1,885,583	68.902 %
Bloom-64	1,200,444	43.866 %
Bloom-128	328,202	11.993 %
Bloom-256	18,691	0.683 %

Figure 7.4 shows the average values for all data sets using the shortest distance similarity method. Due to the large number of false positives for smaller data sets, the difference between average similarity values for spam and nonspam pages is negligible and the average similarity value is much higher for smaller data sets due to false positives. The average values calculated when using the average distance similarity method with smaller data sets is also

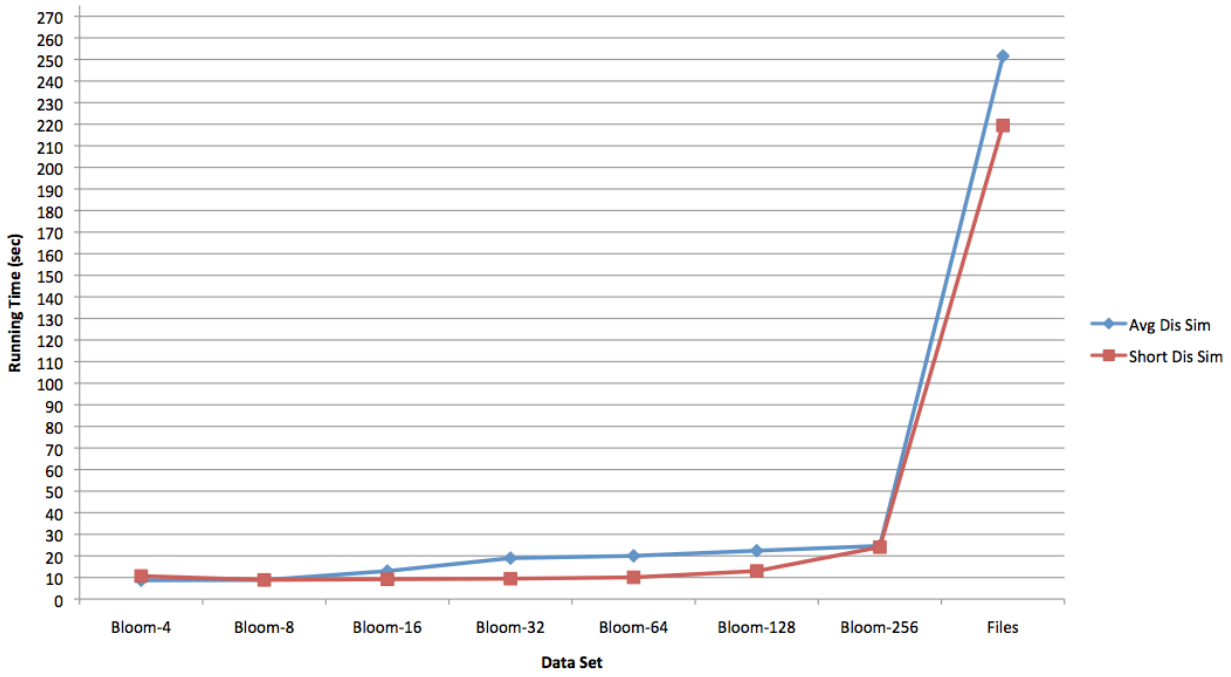


Figure 7.3: Average Running Time for All Data Sets

much higher than the actual value, but there is still a noticeable difference in similarity value between spam and nonspam pages (Figure 7.5).

`Bloom-256`'s characteristics, i.e. faster and smaller than the files and having a very low percentage of false positives, make it the best choice of the eight lookup sets for this application. The association files would need to be kept if any future changes needed to be made; However, they could be kept elsewhere. The total disk space required of 256 GB is large, but with several hard drives available in excess of 1 TB and costing less than $200 should not be a major limitation.

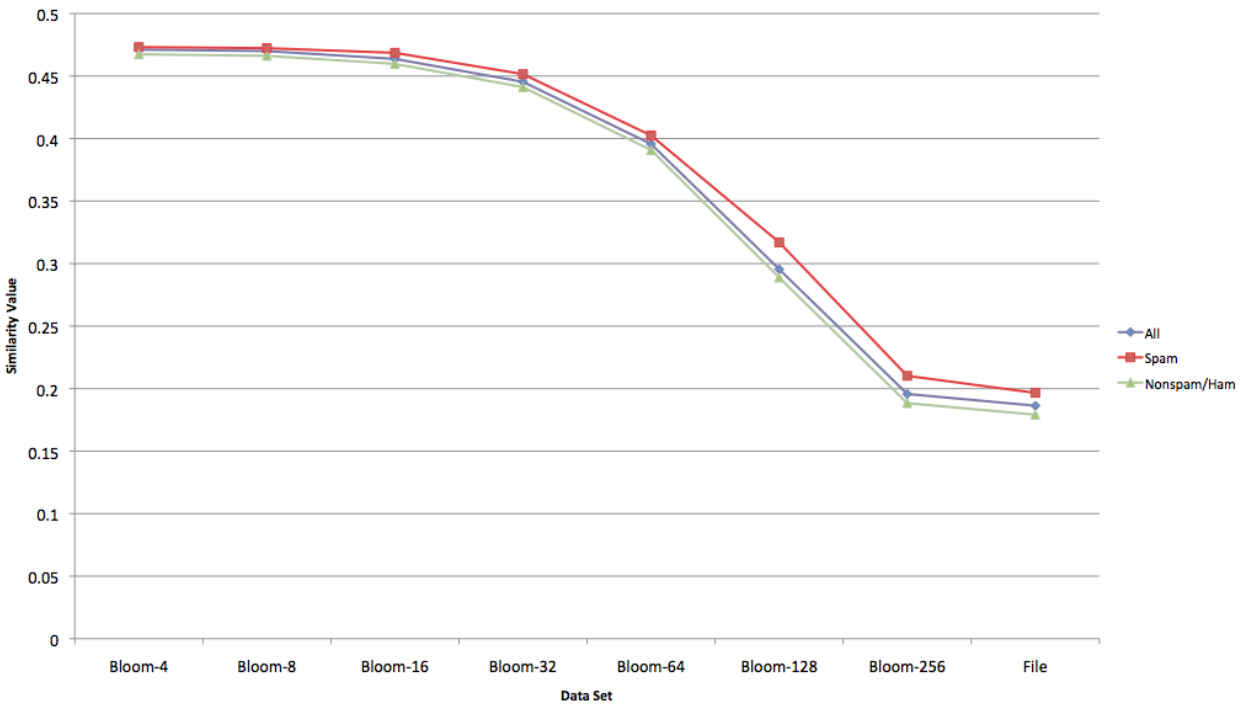


Figure 7.4: Average Value of Shortest Distance Similarity for All Data Sets With a **Spam Page** as Input

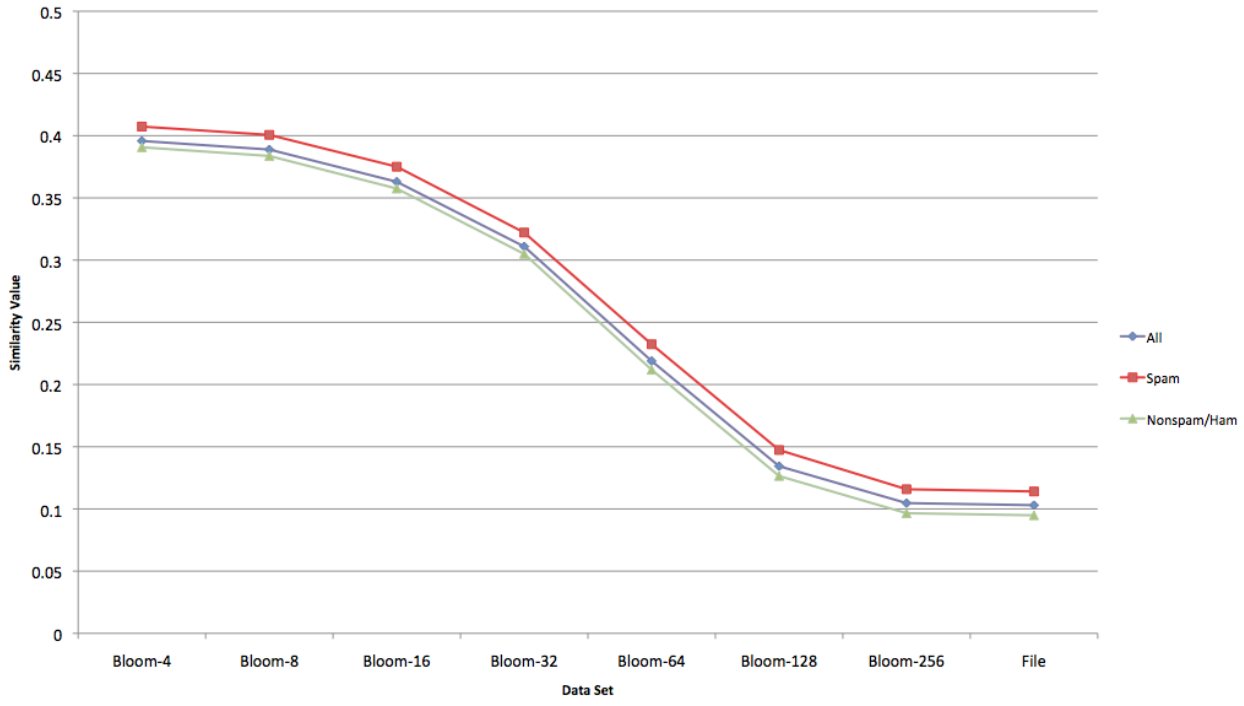


Figure 7.5: Average Value of Average Distance Similarity for All Data Sets With a **Spam Page** as Input

Chapter 8

Conclusions and Future Work

This chapter discusses overall conclusions drawn for the evaluation and improvements of the ranker. Areas of future work that aim to improve the overall application, by improving ranking methods as well as speed, are discussed in depth.

8.1 Conclusions

The Semantic Association Ranker is a ranking mechanism that uses semantic associations to rank pages based on their semantic association similarity. When used with Bloom filters, e.g. `Bloom-256`, the application is relatively fast, i.e. `.25` second required to compare two pages, and accurate, i.e. less than 1% false positive rate. Potential uses for this ranker include identification of candidates for spam pages from a list of URLs that are similar to a given spam page and identifying related pages for a given page given a point of interest. Spam page ranking was the main focus of analysis in this paper.

The ranker attempts to identify spam pages that are similar to a given spam page. Due to the variance in web pages and their content, the ranking mechanism was not able to clearly discern spam pages from nonspam pages. However, with further research, for certain applications the ranker could be used, like related page ranking as was used with the Bloom filter tests in Chapter 7 and compared pages to a set of pages including some that the Google API determined as related. However, in this case it was determined that an evaluation of effectiveness of related page determination is a subjective process and is beyond the scope of this work. The disk space required (256 GB for `Bloom-256`) and time required to compute the semantic relation discovery algorithm (60 days) could be a deterrent to its widespread

use, but as hard drive capacity continues to increase and because the set of associations could be generated offline, they are not severe limitations to its use. Also, since associations between entities do not change frequently, the associations would not have to be generated often. DBpedia has released ten different data sets, at the time of this writing, since 2007.

The simple proper noun identification, ranking and relation discovery methods are used as more of a proof-of-concept for the entire ranker and could be improved which might result in better rankings for pages. The proper noun tokenizer searches for sequences of capitalizes words to query against the DBpedia data set. The tokenizer would incorrectly combine sequences of capitalized words into one token search. For example, the tokenizer would find a string like "Suspected Russian"[1] and fail when querying DBpedia knowledge set; However "Russian" would be a valid entity in DBpedia's knowledge set. During pre-processing, the ranker also does not discern or rank types of associations and only focuses on shortest path relationships.

Based on this evaluation, I believe that the Semantic Association Ranker and Semantic Association related similarity have potential to be a valuable resource for the identification of related pages, but are not standalone methods that can effectively identify spam pages from a pool of spam/nonspam pages. It is possible however when used in conjunction with other classification methods, the methods mentioned may help identify spam pages. However a future analysis of the key components, including the proper noun tokenizer, the relation discovery mechanism and the similarity ranking mechanism, would be required to improve and refine each to improve overall classification and ranking. Also an analysis combining the methods using in this paper with other techniques would need to be overtaken. The time required to compare pages, 1 second per page comparison and ranking, is not fast enough for real-time classification, so an analysis of methods to improve ranking would also be beneficial.

[1]From CNN's Home Page (http://www.cnn.com). Accessed on July 4, 2010.

8.2 Future Work

Several aspects for future analysis and improvement of the Semantic Association Ranker have been identified throughout this paper. Research into the proper noun tokenizer, relation discovery and ranking based on value, and the similarity ranking methods are the key areas to focus on to improve the results of the ranker.

The proper noun tokenizer currently is a simple and naïve algorithm that extracts sequences of capitalized words that are used to query against the DBpedia knowledge base. There are many cases where the tokenizer will be unable to extract correct terms, e.g. "Russian" instead of "Suspected Russian" as discussed in 8.1. Some proper nouns do not start with capital letters, including facebook[2] and Apple's line of *i* products, iPod, iPhone, iPad, iMac, etc. A part of speech component could be added to the tokenizer, such as Wordnet[3] for English, that could identify remove adjectives from a sequence of capital letters or identify nouns that are not capitalized.

Ranking associations during the relation discovery component is probably the area that needs the most improvement. The current ranking mechanism treats all associations of the same distance as equal without regard for the type of property associating the entities and the end user's context. More advanced methods, such as the *inverse entity frequency*, or *ief* method introduced in [4], which ranks the value of an association based on how often the entity occurs, would need to be analyzed from inclusion into the ranker. The user's interest or context, as discussed in [8], should also be considered when ranking pages to return a related list to a user. The current ranker can rank pages based on 5 general categories, i.e. people, places, organizations, works and other, but a more robust set of categories is possible. Using an extended Bloom filter, as in [11], that returns a similarity value for an association instead of a true of false set existence could be used to rank associations between entities based on the *ief* or user's context.

[2]http://www.facebook.com
[3]http://wordnet.princeton.edu/

The simple similarity ranking methods, shortest and average distance, also have areas that need to be improved. The methods rank many pages based on one page as input without regard for the number entities found on the pages to be ranked. As shown in this paper, input pages that have a few, common entities will rank many pages that also have those entities high, ignoring the total number of entities each of these pages contains. For example, if a given input page has 1 entity and two test pages also have the same entity, both of their similarity scores will be `1.0` in the context of the input page even if one of the test pages has 100 entities and the other has 2. Also, similarity values are generated ignoring other similarity values. As another example, if a given page has 1 entity and of 2 test pages, $\texttt{P}_1$ also has 1 entity of distance `1` hop from the input. The other page, $\texttt{P}_2$, has 10 entities of `1` hop distance and 10 entities of `2` hop distance. Both of these methods will rank the $\texttt{P}_1$ higher than $\texttt{P}_2$ even though $\texttt{P}_3$ has more related entities at `1` hop distance. It could be argued however that since the $\texttt{P}_1$ does not have any entities at greater distances than `1`, then it should be ranked higher. Future work, possibly involving ranking with human subjects, could be undertaken to help decide tweaks to the ranking mechanisms to provide better results. Also, changes to the proper noun tokenizer and ranking associations during relation discovery, would mean that the ranking methods would have to be changed to incorporate weighted association values based on context or entity frequency.

Speed concerns must also be addressed that would allow this ranker to operate on a real-time manner. While this paper highlights the more than 9x speed improvement of using a Bloom filter over a binary search over files, the `.25` second duration to compare two pages is not fast enough to use this ranker in real-time over a large set of documents.

Bibliography

[1] Silverstein, Craig; Marais, Hannes; Henzinger, Monika; and Moricz Michael. Analysis of a very large web search engine query log. In *SIGIR Forum*, 33(1):612, 1999.

[2] Lehmann, Jens; Schüppel, Jörg; and Auer, Sören. Discovering Unknown Connections – the DBpedia Relationship Finder. In *Proceedings of the 1st SABRE Conference on Social Semantic Web (CSSW)*, 2007.

[3] Heim, Philipp; Hellmann, Sebastian; Lehmann, Jens; Lohmann, Steffen; and Stegemann, Timo. RelFinder: Revealing Relationships in RDF Knowledge Bases. In *Proceedings of the 4th International Conference on Semantic and Digital Media Technologies SAMT 2009*, Springer, 2009.

[4] Zaragoza, Hugo; Rode, Henning; Mika, Peter; Atserias, Jordi; Ciaramita, Massimiliano; and Attardi, Giuseppe. Ranking Very Many Typed Entities on Wikipedia. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, Lisbon, Portugal, November 06-10, 2007.

[5] Völkel, Max; Krötzsch, Markus; Vrandecic, Denny; Haller, Heiko; and Studer, Rudi. Semantic Wikipedia. In *15th World Wide Web Conference*, 2006.

[6] Sabou, Marta, d'Aquin, Mathieu, and Motta, Enrico. Relation Discovery from the Semantic Web. UK: ASWC/ISWC. 2008.

[7] Sabou, Marta; d'Aquin, Mathieu; and Motta, Enrico. SCARLET: SemantiC relAtion discoveRy by harvesting onLinE onTologies. In *Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, ESWC*, volume 5021 of Lecture Notes in Computer Science, pages 854–858. Springer, 2008.

[8] Aleman-Meza, Boanerges; Halaschek-Wiener, Christian; Arpinar, I. Budak; Ramakrishnan, Cartic; and Sheth, Amit P. Ranking Complex Relationships on the Semantic Web. In *IEEE Internet Computing*, (May-June 2005), pp. 37-44.

[9] Bloom, Burton H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM,* vol.13(7), July 1970.

[10] Chang, Fay; Dean, Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson C.; Wallach, Deborah A.; Burrows, Mike; Chandra, Tushar; Fikes, Andrew; Gruber, Robert E. Bigtable: A Distributed Storage System for Structured Data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI06)*. 2006.

[11] Li, Kang and Zhong, Zhenyu. Fast statistical spam filter by approximate classifications. In *Proceedings of 2006 ACM/SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (St. Malo, France, June 2006).

[12] Ntoulas, Alexandros; Manasse, Mark; Najork, Marc; and Fetterly, Dennis. Detecting Spam Web Pages through Content Analysis. In *Proceedings of the 15th International Conference on the World Wide Web*, Edinburgh, Scotland, May 2006.

[13] Gyöngyi, Zoltán; Garcia-Molina, Hector; and Pedersen, Jan. Combating Web Spam with TrustRank. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, 2004.

[14] Krishnan, Vijay and Raj, Rashmi. Web Spam Detection with Anti-Trust Rank. In *the 2nd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, August 2006.

[15] Westbrook, Andrew and Greene, Russell. Using Semantic Analysis to Classify Search Engine Spam. Technical report, Stanford University, 2002.

[16] Gyöngyi, Zoltán and Garcia-Molina, Hector. Web Spam Taxonomy. In *1st International Workshop on Adversarial Information Retrieval on the Web*, May 2005.

[17] Fetterly, Dennis; Manasse, Mark; Najork, Mark; and Wiener, Janet. A Large-Scale Study of the Evolution of Web Pages. In *Proceedings of the 12th International Conference on World Wide Web*, May 20-24, 2003, Budapest, Hungary.

[18] Fetterly, Dennis; Manasse, Mark; and Najork, Mark. Spam, Damn Spam, and Statistics: Using Statistical Analysis to Locate Spam Web Pages. In *Proceedings of WebDB*, pages 16, June 2004.

[19] Bizer, Christian; Lehmann, Jens; Kobilarov, Georgi; Auer, Sören; Becker, Christian; Cyganiak, Richard; and Hellmann, Sebastian. DBpedia – A Crystallization Point for the Web of Data. In *Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 7, No. 3*, (15 September 2009), pp. 154-165.

[20] Hepp, Martin; Bachlechner, Daniel; Siorpaes, Katharina. Harvesting Wiki Consensus – Using Wikipedia Entries as Ontology Elements. In *1st Workshop SemWiki2006 - From Wiki to Semantics at ESWC 2006*, Budva, Montenegro, 2006.

[21] Anyanwu, Kemafor; Maduko, Angela; Sheth, and Amit. SemRank: Ranking Complex Relationship Search Results on the Semantic Web, In *Proceedings of the 14th international conference on World Wide Web*, May 10-14, 2005, Chiba, Japan.

[22] Halaschek, Christian; Aleman-Meza, Boanerges; Arpinar, I. Budak; and Sheth, Amit P. Discovering and Ranking Semantic Associations onver a Large RDF Metabase, In *Proceedings of the 30th VLDB Conference*, (Toronto, Canada, 2004).

[23] Broder, Andrei and Mitzenmacher, Michael. Network Applications of Bloom Filters: A Survey, In *Proc. of Allerton Conference*, 2002.

[24] Liu, Obey. Relation Discovery on the DBpedia Semantic Web. Grenoble Institute of Technology - ENSIMAG. May 2009.

[25] Yahoo! Research: "Web Spam Collections". http://barcelona.research.yahoo.net/webspam/datasets/ Crawled by the Laboratory of Web Algorithmics, University of Milan, http://law.dsi.unimi.it/. URLs retrieved 06 2010.

[26] Castillo, Carlos; Donato, Debora; Becchetti, Luca; Boldi, Paolo; Leonardi, Stefano; Santini, Massimo; and Vigna, Sebastiano. A Reference Collection for Web Spam. In *ACM SIGIR Forum*, v.40 n.2, p.11-24, December 2006.

[27] Perkins, Alan. The Classification of Search Engine Spam. http://www.silverdisc.co.uk/articles/spam-classification/. Last accessed Jul 4, 2010.