Identification and on-line incremental clustering of spam campaigns

Charalampos Sarantopoulos

August 2015

Universiteit Utrecht

sentient information systems

Master thesis

Supervisors: Dr. A.J. Feelders
Drs. M.J. den Uyl
V. Hoekstra

Acknowledgments

For this journey, I would like foremost to express my deepest gratitude to my university supervisor, Dr. A.J. Feelders, not only for his support and guidance throughout my thesis project, but also for being a source of inspiration from the very beginning of my time in Utrecht University.

I owe my sincere gratitude to Marten den Uyl for offering me the opportunity to carry out my thesis project in an innovative and challenging environment. Special thanks to my daily supervisor at Sentient, Vincent Hoekstra, for his endless support, our interesting conversations, his critical view and for being always available and patient for questions. I would also like to thank George Solakidis for his valuable help, comments and suggestions. A big thank you, to all my colleagues from ParaBotS, Sentient and Vicar Vision, for making me feel comfortable from day one, having interesting conversations and playing darts during our breaks.

Finally, I would like to thank my family and my friends for supporting me spiritually throughout this project.

Contents

Abstract

The ever growing spread of spam emails, despite being adequately fought by spam filters, can be more effectively addressed by understanding how spammers act. Grouping spam emails into spam campaigns, provides valuable information on many aspects; how spammers obfuscate and correlation between seemingly different spam campaigns as well as many descriptive statistics. In this thesis, we focus on identifying spam campaigns from a 7.5 months period by clustering the web pages, which are referred to by the URLs inside the spam emails, based on their content. Following that, we apply Latent Dirichlet Allocation to assign a topic to every cluster and finally, we present a mechanism that incrementally clusters the incoming spam emails into spam campaigns in an automatic and on-line environment. We argue that our method for spam campaign identification is quick and efficient, able to represent the identified spam campaigns in a compact manner. On top of that it can assist towards better understanding of the domain and its applications.

1 Introduction

Spam messages are an efficient, cheap and common way of spreading unsolicited messages used for many purposes such as advertising, deceiving, testing the efficiency of spam detection systems, hacking and infecting computers etc. but most of the time, its main purpose is to make money. In this thesis, we narrow our interest to spam emails, also known as junk emails or unsolicited bulk emails (UBE), and we are particularly interested in identifying spam campaigns. In the context of this work, we will try to identify the spam campaigns, extract the topic for each spam campaign and finally create a mechanism that will be able to automatically cluster the spam emails into spam campaigns. Therefore, the work is divided in chapters as follows.

In chapter 1 we state the problem and present the related work in this field while we differentiate our work from the previous studies.

Chapter 2 refers to the database acquisition and management and describes the methods and the procedure we adopted to gather our data.

In chapter 3 we present the measures we used throughout the experiments we conducted. We give an overview of their functionality and explain their suitability for our cause.

Chapter 4 describes the process of identification of spam campaigns from spam emails that arrived during a 7.5 months period. We explain the procedure, the algorithm and the hypotheses we introduced for identifying the spam campaigns. Finally, we explain those results based on the evaluation we performed and our insight into the data.

In chapter 5, we try a topic modelling approach in order to label our clusters. Latent Dirichlet Analysis (LDA) is used so that we can get an idea of what the identified clusters are about and we also present some interesting descriptive statistics.

Chapter 6, describes an algorithm for automatic, on-line incremental clustering of spam emails. Based on the evaluation of the clusters, the best performing setting is chosen and applied.

Finally, in chapter 7, we conclude with a summary of the main results and open the path for future work and directions in this field.

1.1 Problem statement

According to [3], an electronic message is "spam" if (A) the recipient's personal identity and context are irrelevant because the message is equally applicable to many other potential recipients; AND (B) the recipient has not verifiably granted deliberate, explicit, and still-revocable permission for it to be sent. The volume of mass unsolicited electronic email has increased dramatically, being a threat not only to the internet but also to society. For instance, according to Symantec Intelligence Report [2], the global proportion of spam in email traffic had reached 71.9% during March and April 2013.

It is generally believed that spam campaigns are designed such that they serve numerous different purposes such as advertising, deceiving, testing the efficiency of spam detection systems, hacking and infecting computers etc. Most of the times though their aim is to increase revenue. People responsible for distributing spam emails are often those who try to spread these spam emails in a way such that they are not easily tracked by the spam filters, i.e. they use obfuscation techniques.

By looking at spam emails as a whole we can hardly ascertain any characteristics that may point to strategies being used [19], but by grouping those spam emails into spam campaigns, we can gain insight into spamming strategies and protect ourselves better. Although the way spammers obfuscate is not our subject of study, it could be better treated when spam emails are grouped so that each spam campaign can be examined separately for common patterns [10].

Eventually, by clustering the spam emails, we are able to extract insights that could assist in understanding how spammers obfuscate and disseminate their messages. Also, there is a vast amount of data, where analysing all these spam emails is infeasible and costly. By grouping the spam emails into clusters, we can focus on different set of spam emails that maintain their characteristics and speed up the procedure. On top of that, spam gangs are generally believed to be responsible for launching the spam campaigns. Different spam campaigns can be originated by the same spam gang [15]. As a result, identifying spam campaigns and studying their attributes and characteristics is an important step towards identifying spam gangs in terms of the way they operate.

Therefore, our main purpose is to find a way to identify the spam campaigns, which we will further use to build an incremental mechanism for clustering spam emails into spam campaigns, as described in chapter 6.

1.2 Spam campaign definition

There is no generally accepted definition for a spam campaign. According to [14], a spam campaign is commonly used with varying degrees of generality to mean anything from all spam of a certain type (e.g. pharmaceutical), to spam continuously generated from a single template. Also according to [14], three levels of abstractions are introduced to further define the term.

1. Classes: Intened purpose of the spam, e.g. phising, pharmaceutical offers, stock scam etc.

2. Types: Sets of spam campaigns that have messages that share content which can be matched directly, such as verbatim text.

3. Instances: Sets of spam campaigns that are identical but run across several time periods, delimited by 24 hour time slots.

Another more compact definition of spam campaign comes from [10], where they state that a spam campaign is defined as ”a set of spam messages that have the same goal, such as advertising a specific brand or product; and that also use the same obfuscation strategy among this set of messages”

Since we are not dealing with obfuscation techniques, our definition for a spam campaign is those spam messages that refer to the same entity (e.g. sets of spam emails that advertise a product).

1.3 Related work

Spam campaign identification has received a lot of attention and several studies have focused on this topic, although each one for different purposes.

A spam email has different attributes one can explore in order to identify the spam campaigns. Based on these attributes, we can categorize the related work as follows.

Li and Hsieh in [16], in order to cluster the behaviour of spammers, they try to cluster the spam emails based on the URLs of the spam email. Their group-based anti-spam framework can be used as a complementary tool to the anti spam filters for a more efficient block of spammers. In the same context, the authors in [21], have proposed the AutoRE framework, that can detect botnet hosts by signatures, which are generated from URLs that are embedded in the body of the emails. The two aforementioned studies focus on URL-based clustering. Despite exhibiting a very good performance, spammers can easily obfuscate the URLs, for instance by using dynamic source IP adresses or polymorphic URLs.

A different approach for identifying spam campaigns is to examine the content of the spam emails. Such an approach is presented in [22], where the goal of the authors is to map botnet memberships (group bots into botnets),for which they first need to identify the spam campaigns. This is performed by applying the shingling algorithm [9], as a text mining process.

A similar approach was conducted by [18], where they designed an online spam campaign detection tool called SpamCampaignAssassin (SCA). SCA involves a text mining framework that utilizes Latent Semantic Analysis (LSA) to identify campaigns. These two approaches though suffer from lack of scalability and as a result text mining becomes ineffective.

Wei et. al in [20] use the agglomerative hierarchical clustering algorithm together with the connected components with weighted edges in order to cluster the spam emails. Cailis, et al have tried to identify spam campaigns in order to characterize the spamming strategies. They have developed a technique based on a frequent pattern tree that explores many attributes of a spam email such as its language, layout, message type (HTML, text, image), URL and subject. Haider et al. in [12], in order to group spam emails into spam campaigns, have applied Bayesian hierarchical clustering [13]. A generative model is created so that binary vectors are clustered based on a transformation of the input vectors. Lems in [15], also tries to identify spam campaigns in order to examine to what extent spammers tend to share resources such as mail server, recipient list and web servers. To this end, he constructed a bag-of-words model to represent the text of the spam email (i.e. subject and body) and then he applied an agglomerative hierarchical clustering algorithm on the document vectors.

Finally, Anderson et al. in [4], have followed a different path towards identifying spam campaigns. Specifically, their model, spamscatter, follows the URLs lying inside the spam emails and cluster the spam emails into spam campaigns, using image shingling from the content of the referred web pages.

1.4 Spam campaign identification based on websites

In spite of the fact that the related works include novel ways for identification of spam campaigns, our work differentiates by exploiting an attribute of a spam email that has not attracted much attention so far. An attribute of a spam email is the attached link it contains, usually in the body of the spam email, that links to a web page. There might be more than one attached link possibly pointing to different web pages. Our work therefore focuses on studying the web pages that originate from the spam emails and trying to determine document similarity on the content of the referred web pages. We believe that by comparing the referred web pages, we can cluster the spam emails and identify the spam campaigns. Another significant reason behind studying the web page from the attached link, is that we believe it is not or hardly obfuscated. Although some typical obfuscation techniques are to replace words with synonyms, insert irrelevant words in between sentences, change the ordering of sentences etc. that usually apply in the subject, URLs or in the body of the email, we believe that the referred web page is less obfuscated and a better clue to take advantage of. To the best of our knowledge, only [4] has dealt with the referred web pages, but we focus on the text of the page instead of the images contained in it.

Looking at the web pages, the problem is then translated into comparison of text between documents. When it comes to document similarity, there is a vast literature devoted to that subject. However, due to the special nature of our dataset and the purpose we want to achieve, we explain the procedure we followed and our reasoning in chapter 3, where we describe the appropriate measures we applied for document similarity. First, we describe in the next chapter how our data are organized in the databases and the filters we applied in order to extract our dataset.

2 Data acquisition and management

The first challenge to encounter was the acquisition of the dataset. ParaBots (PAge RAting BOT Systems) has created a system that stores large volumes of spam emails that arrive daily.It processes the spam emails through the spam monitor software in order to analyse their behaviour and contribute to a better understanding of the domain. So, the daily flow of spam emails can reach up to 1 million which are then stored in the databases for further analysis.

2.1 Data organization

There are two databases for storing the data, a MongoDb and a MySQL database. The MongoDb database is responsible for storing all the information about the spam emails. It contains several tables, each one used for storing specific attributes of the spam emails. Every spam email has a unique number (id); a link between the tables. There are many tables in the MongoDb database but for illustrative purposes we refer to the two most important ones. The *spam* table contains attributes such as original-mail-from (sender's mail address), arrival date (date that the spam email was received), charset (UTF-8 or ISO-8859-1 format), PlainTxt (text of the body of the spam email), html (html code of the spam email) etc. The *url* table contains attributes such as charset (UTF-8 or ISO-8859-1 format), host (the domain of the URL), in (which spam emails contain a specific URL) etc.

These tables are characteristic examples of the organization of the MongoDb database; there are many more to fully describe and categorize the spam emails. What is more, the spam emails very often contain attached URLs that refer to an external web page. Spam emails can either contain more than one URL or none. If a spam email has attached links, it usually has only one. In case of more than one attached links, it is reasonable to believe that they refer to the same entity (URLs point to web pages with similar content). Whatever the case is, the web pages are uniquely numbered when stored in the MySQL database and we can always refer back to the MongoDb database to see in which spam email they are contained. The MySQL database is responsible for storing all the relevant attributes of the spam web pages with a structure similar to the MongoDb database. THe MySQL database is also organized in tables, each one containing relevant attributes of the web pages. Indicatively, we mention a couple of important tables so as to show the structure of the database. The table *pages* contains attributes such as title, summary, html code, charset, country code, plain text and download date. The table *language* contains the attribute language that the web page is written in.

2.2 Filtering English and url contained spam emails

From now on, we will only refer to the web pages as stored in the MySQL database as these are the subjects of study and our further analysis. Having clarified the way the databases are structured, we now describe the filters that are set in order to obtain the data. The first and apparent filter to be applied is the fact of whether a spam email contains an attached URL or not. Therefore, we filter out those spam emails that do not contain a URL. This is the filter with the highest impact on the reduction of spam emails.

The second filter is the language of the text of the website. There are many different languages that a web page can be written in. A language detector tool is applied in order to predict the language used for every web page and based on this tool the database is updated in the relevant table. We are interested only in the English language which is by far the most popular language (from 17/09/2014 until 29/04/2015 there are 44223 English web pages, 25048 German web pages, 854 Italian web pages, 706 French web pages, 564 Swedish web pages, 532 Dutch web pages and many more in decreasing order).

After these two filters applied, we ended up with an order of tens of web pages per day, where prior to the filters, it used to be up to one million. The MySQL database stores only the web pages, which means the spam emails have already been filtered on the URL attribute. So MySQL is our main database to work with, while we need to add the extra limitation on filtering on the language to retrieve only the English web pages.

3 Appropriate similarity measures

One of the basic and fundamental problems we faced was the choice of the measure that could identify the similarity between two web pages. Before describing the measures we applied, we need to report a couple of attributes that are related to the nature of the web pages.

From our experience and having manually inspected many web pages, we have the following remarks to present. Our first observation has to do with the type of the web pages in our dataset. In general, there are two categories of web pages. The first contains very similar web pages while the second contains those web pages that have some parts in common but are not considered near-duplicates. The first category usually contains web pages that differ only in a small percentage. The difference might lie in numbers, titles, names or different attached links. These web pages are often warning messages or fixed messages that are informative about malicious behaviour of sites, absence of a valid url etc. Sometimes, they might contain the actual content of a web page but whatever the case is, these web pages differ only in a small percentage.

On the other hand, the second category of web pages relates to those web pages that are similar in a different sense than what is described for type 1. It is a common phenomenon that web pages have different news in the context of their texts. A web site might be 200 words long, split in 6 paragraphs and each paragraph talking about a specific topic. Two such web pages might have one or more paragraphs in common, meaning that these paragraphs are exactly or almost similar. As a consequence, two web pages may share a common part of text but a third web page might share a different part of text with one of the two previous web pages. Nevertheless, as long as this is the case (identical parts shared among the web pages) the second category contains web pages that are partially similar among each others.

Next, we describe two measures that we tested and explain their suitability to our cause.

3.1 Simhash

Charikar's Simhash [11], is particularly useful for identifying near-duplicates in web documents belonging to a multi-billion page repository [17]. Initially, simhash was the fist measure to apply in order to obtain a similarity score between two web pages. Every document is represented as a number of bits with 64 and 128 to be the most common. Their binary representation allow for quick hamming distance calculation and eventually a score to indicate the number of different bits between two documents. A threshold is then defined to determine the similarity between documents. It was only after we got the first results, when we realized its unsuitability to our purpose. Simhash is designed such that it can detect near duplicate documents; documents that differ only in a small percentage, like those belonging in type 1 as described in 4. Simhash is very sensitive to minor changes in text and this is why the threshold for a 64 bit simhash value is really low. The threshold for a billion document database according to [17] with 64 bits is estimated to work properly for values 4 or 5. As a result, while it was able to identify near duplicate documents, it lacked in being able to distinguish between documents of type 2. Consequently, simhash was unable to capture the documents that constitute a spam campaign, because of their bigger than 5 difference in score.

3.2 Plagiarism detection

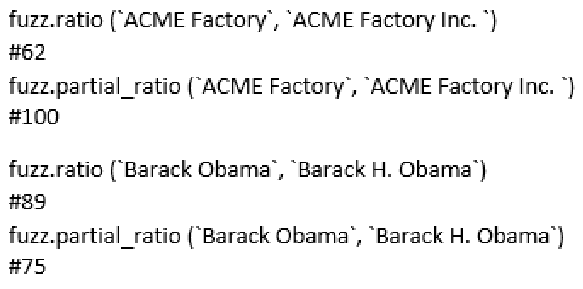
Having abandoned the idea of Simhash and figured out the cause of the problem, we tried to come up with a scheme that would fit to our needs. Plagiarism detection was the scheme to introduce as it is able to capture web pages that share common parts. It is reasonable to believe that two web pages that share a subtext of a few lines, each one taken from its corresponding text, originate from the same source. Plagiarism detection then could perform well in both categories of web pages. Consequently, for the web pages of the first category, the output score would be very high whereas for the second category would be lower.

In terms of plagiarism detection, the successful measure to be applied should be able to output a similarity score between two documents, indicating the degree of their similarity. The size of the document is a factor to take into account. On one hand it influences the percent of similarity between two documents but we believe that two documents that are considered to belong to the same spam campaign do not have substantial difference in their size. The ideal measure should be able to embed fuzzy logic. In other words, it should be able to detect that two documents refer to the same entity despite any possible obfuscation techniques or re-organization of the structure of the documents.

Fuzzywuzzy in Python [1], a library for Fuzzy String Matching or Approximate String Matching, was deployed to assist in that purpose. Fuzzy String Matching, is the process of finding strings that approximately match a given pattern. The closeness of a match is often measured in terms of edit distance, which is the number of primitive operations (insertion, deletion and substitution of a character) necessary to convert the string into an exact match.

Typical applications of Fuzzy String Matching involve spell-checking, text re-use detection (as of plagiarism detection), spam filtering as well as several applications in the bioinformatics domain, e.g. matching DNA sequences.

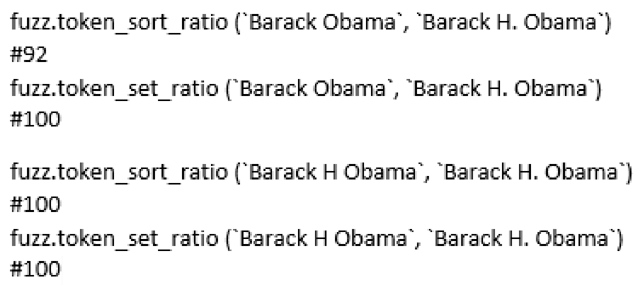
The fuzzywuzzy library contains several functions, each one designed for different applications of Fuzzy String Matching. The score ranges from 0 to 100 with 100 indicating exact similarity and 0 totally different. Next, we list those functions along with the description of their suitability and conclude to the one we chose to apply.



```
fuzz.ratio (`ACME Factory`, `ACME Factory Inc. `)
#62
fuzz.partial_ratio (`ACME Factory`, `ACME Factory Inc. `)
#100

fuzz.ratio (`Barack Obama`, `Barack H. Obama`)
#89
fuzz.partial_ratio (`Barack Obama`, `Barack H. Obama`)
#75
```

Figure 1: ratio vs partial.ratio for Fuzzy String Matching.



```
fuzz.token_sort_ratio (`Barack Obama`, `Barack H. Obama`)
#92
fuzz.token_set_ratio (`Barack Obama`, `Barack H. Obama`)
#100

fuzz.token_sort_ratio (`Barack H Obama`, `Barack H. Obama`)
#100
fuzz.token_set_ratio (`Barack H Obama`, `Barack H. Obama`)
#100
```

Figure 2: fuzz.token_sort_ratio vs fuzz.token_set_ratio for Fuzzy String Matching.

Consider the first example in figure 1. The fuzz.ratio() function is confused by the suffix ”Inc.”, which is used in company names, but essentially the two strings refer to the same entity.

Looking at the second example we observe the exact opposite behaviour. The reason is the term "H." in the middle of the string that produces a lower score for fuzz.partial_ratio(). In order to overcome that problem, we tried two other functions that tokenize the string and treat it as a set or a sequence of words as shown in figure 2.

The token_* functions split the string on white-spaces, turn all upper-cases into lower-cases and remove the non-alpha and non-numeric characters.

We chose the token_set_ratio as the function to apply for determining the similarity between two documents, because we believe it can represent an entity better. The score of the token_set_ratio is not influenced (decreased) by random words that might exist inside the strings. In that way, it avoids such cases of typical obfuscation techniques. Instead, it is searching for similar sets of terms that refer to the entity.

4 Clustering spam campaigns for identification

In this chapter, we are going to show how web pages can be clustered so that spam campaigns can be identified. Spam campaign identification as presented in this chapter, will serve as the baseline for the next chapters that deal with labelling of those clusters (chapter 5) and on-line, automatic incremental clustering of incoming spam emails (chapter 6). We open this chapter by referring to the importance and the role of the parameters we will employ, while examining the range of their values leading to a series of experiments (4.1). Next, we describe the idea behind the design of the algorithm along with some heuristics used to assist in performing the experiments as presented in 4.2. We continue in 4.3 by performing an evaluation of the clusters in order to select the best performing configuration of parameters and we conclude with 4.4 where we discuss the results.

4.1 Parameter selection

In this section, we focus on identifying spam campaigns. There are 44.223 web pages in total from a 7.5 months period (17/09/2014 - 29/04/2015). These web pages need to be clustered in such a way that one cluster represents a spam campaign. For that reason we employ the Fuzzy Wuzzy measure in order to get a score when the text of two web pages is compared. The algorithm that we describe next is based on two main functions. It is apparent that we cannot do pairwise comparisons in our whole dataset as it would require n^2 comparisons. Therefore, we needed to come up with a scheme in order to reduce the number of comparisons, while maintaining the trustworthiness of the results. Hence, we split our dataset into batches and for every batch separately, we identified the spam campaigns, The spam campaign identification for every batch is the outcome of the first function. The second function, has to do with the merge of those batches. As a result, prior to implementing the algorithm, we defined two parameters that are described next.

4.1.1 Length of batches

Regarding the first parameter, we split our dataset into batches and processed each batch separately. The length of the batch is a parameter to experiment with, but it is unavoidably limited within certain boundaries. The batch should not be small since it would seriously affect the results when the join-of-the-batches algorithm takes place. It should also not be too large because in that case the time complexity increases dramatically. In general, the bigger the batch, the more reliable the results would be. In other words, the more we increase the length of the batch the more it tends to reach the pairwise comparison state.

We found two cases that served our purpose adequate enough to experiment with. The batches contain either 1000 or 2000 web pages. Batches of 1000 web pages do not need a lot of time and it is considered big enough for obtaining reliable results. On the other hand, batches of 2000 web pages require more time to execute while the results theoretically should be closer to the actual results.

4.1.2 Determining the threshold

The second parameter to test is the threshold of the fuzzywuzzy measure, which determines whether two web pages should belong together under the same spam campaign or not. The score ranges from 0 to 100 with 100 indicating that the two web pages are exactly the same and 0 totally different. Applying such a measure without any previous knowledge on its appropriate threshold value, unavoidably leads to a big series of experiments in order to pick the most suitable threshold value. After some preliminary experiments with extreme values, we rejected threshold values ranging from 0 - 40 and 80 - 100 as in the first case the score is not adequate enough to distinguish between similar web pages whereas, in the latter case a high threshold categorizes web pages such that two "similar" web pages are not assigned to the same cluster as expected.

Finally, the threshold values to be examined range from 45 to 75 which combined with the two cases of batches constitute 14 different experiments. The following table shows these experiments.

Settings	1	2	3	4	5	6	7
Batch size	1000	1000	1000	1000	1000	1000	1000
Threshold	45	50	55	60	65	70	75

Settings	8	9	10	11	12	13	14
Batch size	2000	2000	2000	2000	2000	2000	2000
Threshold	45	50	55	60	65	70	75

Table 1: set of experiments with 14 parameter configurations.

4.2 Methodology - implementation

After having determined the possible values of the parameters we also need to describe the pipe line of the algorithm. The size of the batches and the threshold of the similarity measure are the two apparent and predefined parameters in the algorithm. Nevertheless, during the implementation the need for defining a couple of different parameters came up. These parameters, embedded in the core of the algorithm need also to be stated as they also (slightly) influence the outcome of the algorithm and are described next along with the explanation of the decisions made for the design of the algorithm.

After fetching the data from the MySQL database, we preprocessed the text in the standard way so as to reduce the feature space and make the comparison between web pages faster. The preprocessing step involves lowering the capital letters, removing the punctuation, removing the numbers, stripping the white space and removing the English stop words.

Starting with the first function and having set the threshold and the length of the batch, the algorithm works as follows: the first web page is assigned to the first cluster since the dictionary that stores the clusters is empty. The second web page to be clustered is compared with the first already clustered and if they are found to match (the score is above the predefined threshold) then it is assigned to the same cluster as the first web page. If the score is less than the threshold, the second web page is assigned a new cluster and thus starts a new spam campaign. Algorithm 1 below in pseudo-code describes this procedure.

It is worth mentioning that every page is assigned to the first cluster it matches. Theoretically, the ideally formed clusters should be different from each other. That means that the ordering of the clusters should not have any influence on the result. No matter the ordering of the clusters (line 3 of algorithm 1), the page is supposed to be assigned to the cluster if and only if its score with every page from the cluster-sample is above threshold. In reality, the clusters are not formed ideally, which leads to low intra- and high inter-cluster similarity scores. To check how sensitive the result is with respect to the processing order, would require the repetition of this experiment several times. Then, we would need to evaluate the clusters based on a different ordering of the available clusters at each step of the algorithm. All in all, different ordering of the clusters should not influence the result, but in our case, due to the sampling method, different ordering of clusters might give a better insight.

At this point, we introduce the implicit parameters that take place when

Algorithm 1 clustering the batches.

```
for each batch do
    for each new web page P in batch do
        for each already existing cluster C do
            determine size C
            Select random sample R from cluster C
            for each page Pr in R do
                if  score(P,Pr) >= threshold then
                    NrOfTrue increases by one
            if NrOfTrue == number of samples in R then
                assign page P to cluster C
                break
        if PageNotAssignedToExistingCluster then
            Create new cluster with page P
```

a web page to be clustered has to be compared with the already clustered web pages. Let us suppose that we are in the middle of the process and there have been 500 web pages already clustered. In that point, these 500 web pages have been clustered creating a number of clusters. Each cluster contains a number of web pages which can vary from 1 to 500. In order to assign a new web page into one of the already existing clusters, it needs to check the web pages inside each cluster and as soon as it matches with those web pages it is assigned to the corresponding cluster. For instance, a cluster may contain one hundred web pages and we simply can not afford one hundred comparisons in order to check if the new page belongs in that cluster. As a result, after every assignment of a new web page into some cluster, the updated clusters are split into 5 categories depending on their size. The new web page to be clustered is compared against a random selection of web pages from each cluster. The overview of how clusters are categorized together with the corresponding samples is presented in table 2.

Also, in order for a new web page to be assigned to a cluster, the score for each comparison must be above threshold. In other words, it must be considered "similar" with every randomly selected web page from a cluster. We refer to these parameters as implicit because they were not clearly stated prior to building the algorithm. In contrast to the two initially defined parameters based on which, we performed a series of experiments, we experimented with the implicit parameters by evolving from the early versions of the algorithm to the latests. A more detailed description on the implicit parameters is given at the end of section 4.2. All in all, we could

Category	Size of the clusters	Random sample per cluster
1	1 - 5	1
2	6 - 10	2
3	11 - 50	5
4	51 - 100	8
5	More than 100	10

Table 2: Size of the clusters with the number of random web pages retrieved per cluster.

say that the selection of the implicit parameters is mainly done because we want to keep the number of comparisons relatively low, so the choice of implicit parameters is not only influenced by the performance of the algorithm, but it is more of a trade off between cluster performance and speed.

This procedure takes place for all the batches until the last web page is clustered. Before continuing to the second function, which is the join of the batches, we introduce another scheme. It is a very common phenomenon that clusters contain only one web page. Theoretically, these clusters constitute separate spam campaigns. Although these single web pages are part of our dataset, we chose not to include them in the join-of-the-batches algorithm for two reasons. Firstly, even in the "bad" scenario with 1000 web pages per batch, it is reasonable to believe that a web page that exists alone inside a cluster, is not very likely to match with any other web page from clusters from different batches. It has already been tested against all the clusters from this batch and found not to match with any of these. The probability then for a one-web page cluster to merge with other clusters from different batches is small. Even in the case where this might occur, we expect it to be the minority of the one-web page clusters and therefore we do not lose in generality. In the case where the batch contains 2000 web pages, the likelihood of a one-web page cluster joining another cluster is even smaller. The second reason is that we need to reduce the number of web pages before the next step of joining the batches takes place. The number of one-web page clusters is relatively big, reaching 7604 different clusters. This essentially means that we have 7604 more spam campaigns and 7604 more clusters to be tested when the join-of-the-clusters function takes place. To sum up, clusters containing only one web page, are not considered relevant and are left out from the rest of the process.

Coming next to the second function of joining the clusters from different batches, two web pages from each cluster are randomly selected and are compared against each other. The procedure is similar to the first function and the first cluster contains its own initial pages. Two web pages from the second cluster are randomly selected and are compared pairwise against the two randomly selected web pages from the first cluster. If both pairs have a score above the predefined threshold then the two clusters are merged and the new-formed cluster contains the page-ids from both clusters. This procedure continues until the last cluster from the last batch is compared. The pseudo-code for the second function is shown in below in algorithm 2.

Algorithm 2 Merging the batches.

```
1: assign the first cluster Cinit to the final dictionary F
2: for every other cluster C do
3:     select two random web pages P1, P2 from cluster C
4:     for every cluster Cf existing in F do
5:         Select two random web pages from Cf
6:         compare P1, P2 from C with the two randomly
7:         selected web pages from Cf
8:         if both comparisons have a score above threshold then
9:             merge the two clusters
10:            break
11:    if ClusterNotMergedWithExistingClusters then
12:        create new cluster in F with P1 and P2
```

Finally, after the merging the batches, we have our spam campaigns with at least two web pages per spam campaign. These spam campaigns have been created for each parameter configuration along with the implicit parameters embedded in the algorithm. The implicit parameters for the second function e.g. join-of-the-batches, is the number of pairs created from two clusters in order to be compared. The choice of 4 randomly selected web pages that lead to 2 pairs for comparison, might seem small for joining the clusters, especially the big ones. However, big-sized clusters from algorithm 1 have been formed based on many comparisons and are expected to be very homogeneous. On top of that, putting execution time into the equation, the two pairs for comparisons seem a logical choice to make.

While clearly stating the experiments conducted with the different configurations of the explicit parameters, we have not done the same with the implicit ones, which may have an (important) influence on the outcome. Nevertheless, we managed to gain insight in their role by adjusting their values starting from the early versions of the algorithm until the final, as explained next.

In that point, we describe two different versions of the algorithm which were applied before we concluded to the final version as described above. Initially, and in order to tackle the problem of clustering the web pages and merging the clusters efficiently, without any ground truth in hand, we experimented with different implicit parameters that are described next. The first approach to follow, had as prerequisite condition only a fraction of comparisons to be above threshold. There were 4 categories in total and the sufficient matches are shown in table 3.

So, in the case of 10 randomly selected web pages from a cluster containing 50-100 web pages, it was sufficient only if 3 out of 5 comparisons were above the threshold. This parameter configuration was abandoned since the results were not really promising.

The second version of the algorithm included also 4 categories for the clusters but the limitation of achieving a minimum number of matches was quashed. In this second version, the web page to be clustered must match all the randomly selected web pages from a cluster. In the previous example, where 10 web pages were randomly selected from a cluster containing 50-100 web pages, the new web page must have a score above threshold with all the 5 randomly selected web pages. The overview of the implicit parameters for the second version of the algorithm is shown in table 3.

The second approach, despite exhibiting a decent performance especially in big-sized clusters, suffered from a pitfall. Its drawback occurred in the small-sized clusters mainly due to the fact that clusters with up to 10 web pages are formed based on only one comparison (1 random selected web page from clusters with 0 to 10 web pages).

Version of algorithm	1, 2	1	2
Clusters (Web pages)	random sample	Min. matches	Min. matches
1-10	1	1	1
10-50	3	1	3
50-100	5	3	5
More than 100	10	6	10

Table 3: Number of minimum matches per sample per category for the two versions.

In spite of the fact that our final experimentation set-up performs well and is able to identify the spam campaigns in a very satisfactory degree, there are better configurations that cluster the web pages with better accuracy. The more categories we split our clusters into, the more likely is to achieve better accuracy. Also, the more documents we randomly select from a cluster to compare with a new web page, the better the chances of assigning the correct cluster are. All in all, our model achieves a reliable performance and more elaboration on the parameters' configuration is left for future work.

4.3 Cluster evaluation

For each parameter configuration applied, we obtained a different distribution of clusters. Therefore, we need to select the best parameter configuration that points to the most suitable cluster formation.

Cluster evaluation in this form, is not a trivial procedure as there is no general accepted measure to perform evaluation on this kind of clusters. The special nature of our clusters require alternative measures that can depict the homogeneity of the clusters. Moreover, when talking about cluster evaluation we need to take into account both intra- and inter-cluster similarity. In the first case, a cluster should be as coherent as possible, so that it contains web pages that are meant to belong in the same cluster. In the latter case, the ideal scenario involves no two clusters containing web pages that are supposed to be one cluster. We need to avoid circumstances when different clusters should be actually one cluster, leading to high inter-cluster similarity.

Therefore, we came up with a measure designed in such a way that takes into account both intra- and inter-cluster similarity. It outputs a score for intra- and inter-cluster similarity where one needs to maximize the first and minimize the latter. The point in 2D plane with the best ratio is selected as the best performing parameter configuration.

To begin with, we need to sample both clusters and web pages, otherwise it would be infeasible to perform pairwise comparisons in all of our clusters. Therefore, we made a sample choice on the number of clusters for evaluation and from these randomly selected clusters, we performed a random choice on the number of web pages they contain. By doing so, we were able to make it timely feasible while maintaining the integrity and the notion of similarity of web pages.

More specifically and in order to make our results less skewed, we split our clusters based on their size (the number of web pages inside each cluster). In order to do so, we categorized all clusters as belonging to one of the five following categories: 1) clusters that contain from 2 to 10 web pages, 2) clusters that contain from 10 to 50 web pages, 3) clusters that contain from 50 to 100 web pages, 4) clusters that contain from 100-500 web pages and 5) clusters that contain more than 500 web pages.

Returning next to the way how intra- and inter-cluster similarity scores are calculated, the sampling method on the number of clusters and the number of web pages is a good approach for performing cluster evaluation. Starting with intra-cluster similarity and depending on the size of the cluster, we created random pairs of web pages for each (randomly) selected cluster. We calculated the score for each pair and we finally divided the summed score with the number of comparisons made. This is the average score for intra-cluster similarity and a similar procedure follows for the inter-cluster similarity. Using randomly selected web pages from the randomly selected clusters, we formed pairs so as the two web pages in every pair must come from two different clusters. By doing so, we are able to see the degree of similarity between web pages from different clusters. The results for every parameter configuration are summarized in table 4 as follows:

Settings	1	2	3	4	5	6	7
Intra_score	70.42	86.9	90.43	90.04	92.1	95.03	95.12
Inter_score	29.82	32.32	29.67	35.85	33.03	33.23	26.65
#Clusters	341	743	971	800	605	473	411

Settings	8	9	10	11	12	13	14
Intra_score	72.95	83.54	92.19	92.98	91.97	95.73	95.83
Inter_score	33.35	32.26	30.89	31.27	28.16	29.43	25.6
#Clusters	354	783	1021	843	662	526	461

Table 4: Intra- and inter-cluster similarity scores for different settings.

As mentioned earlier, we need to select the point in plane that performs best in the two axes. Specifically, the point of our choice should have a high score on y-axis and a low score in x-axis. Y-axis represents the intra-cluster similarity while x-axis the inter-cluster similarity. Therefore we need to select the point in 2D plane that combines high intra- and low inter-cluster similarity score.

In addition, we favoured the big-sized clusters during the process of intra-cluster similarity compared to the inter-cluster similarity where we favoured the small-sized clusters. This is explained because the intra-cluster similarity makes more sense and is of higher importance when a cluster is big. A cluster that contains for example 200 web pages is considered a big cluster and we need to be as certain as possible regarding its homogeneity. On the contrary, inter-cluster similarity makes more sense when there are many small-sized clusters and we need to check whether they should have merged or not. Small-sized clusters are more likely to merge than big-sized clusters due to

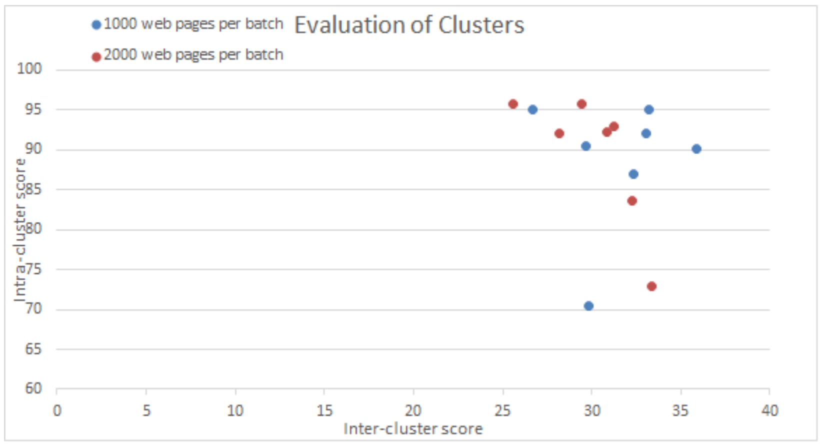


Figure 3: Intra- and inter-cluster similarity score.

the fact that a wrong assignment of a web page would have most likely occurred in the small-sized clusters. Big-sized clusters are considered more solid since more comparisons are made in order to be formed. Towards accomplishing that, during the intra-cluster similarity, we sampled more web pages from big clusters, so that the chance of two compared web pages belonging in a big cluster is higher that the chance of two web pages to originate from a small cluster. One the other hand, we favoured the inter-cluster similarity in such a way that more web pages are sampled from small-sized clusters and as a consequence more comparisons are potentially made between web pages from different small-sized clusters.

Finally, as a remark, the evaluation of the clusters is based on the sampling method, which means that sampling clusters and web pages is not a precise way to find the best parameter configuration. We can observe the difference when the parameters differ a lot, but when the settings are close regarding their parameters' values, the differences sometimes are not so obvious and the results might not depict the real situation. Because of sampling, when two settings exhibit similar scores we cannot be certain of the superiority of one over the other. The same applies when favouring big-sized and small-sized clusters for computing the intra- and inter-cluster similarity score respectively. The fact that we allow more pairs from big-sized clusters for instance, does not necessarily mean we actually selected more pairs from big-sized clusters. Favouring different sizes of clusters for the two cases is just a way to approach the idea behind the evaluation method. To the best of our knowledge we have not introduced any bias towards skewed results, neither are we aware of which clusters provided the pairs for comparison for each case.

4.3.1 Selecting the suitable setting

The graph with the settings represented as points in the 2D plane is shown in figure 3. As you can observe, the differences between the points are relatively small. However there is one point (Setting 14 with y=95.83 , x=25.6) that is slightly higher in the y-axis than the second (Setting 13 with y=95.73 , x=29.43). Also, Setting 14 has the lowest score in x-axis with x=25.6 while the second lowest score is from Setting 7 (y=95.12 , x =26.65). Although the differences in the two axes from the second best scores (setting 13 in y-axis and setting 7 in x-axis) are not substantially bigger to choose setting 14 without hesitation as the best, it is still superior than the rest of the points and it will be the point to choose as the best performing setting.

Looking how close the points in the plane are, it would be a good idea to see the impact of sampling on the settings' mark on the plane. For that purpose, we chose the two settings with the highest thresholds, namely settings 13 and 14, to observe their performance when different samples are taken to perform the evaluation of the clusters. The idea is to find the clusters for each setting and then perform evaluation by repeating the sampling method ten times per setting. In that way, we can see the variance of the points because of the uncertainty introduced by the sampling method. The table 5 below summarizes those results

From table 5, we observe that for each evaluation process, the fluctuation of the results ranges within a few percentage points. Specifically, for setting 13, the intra-cluster similarity score lies between 92.37 (5th iteration) and 95.83 (1st iteration) with a difference of 3.46. The inter-cluster similarity score lies between 24.32 (5th iteration) and 32.63 (8th iteration), having a difference of 8.31.

On the other hand for setting 14, the intra-cluster similarity scores ranges from 96.03 (3rd iteration) to 93.85 (4th iteration) leading to 2.18 difference, whereas its inter-cluster similarity score ranges from 25.59 (1st iteration) to 32.19 (8th iteration) with 6.6 difference.

Looking at figure 3, some points in the plane are very close to each other. Taking into account the variance of the settings 13 and 14, we can conclude that figure 3 eventually is not really helpful in drawing very safe conclusions.

The ideal scenario would be to repeat this ten-fold evaluation method for all settings. Averaging over all ten scores per setting per axis, we would get a more accurate estimate of the evaluation score for each setting. We

	Setting 13	Setting 14
Intra_cluster 1	**95.73**	**95.83**
Inter_cluster 1	**29.49**	**25.59**
Intra_cluster 2	92.94	94.88
Inter_cluster 2	31.37	30.54
Intra_cluster 3	**92.63**	**96.03**
Inter_cluster 3	**27.02**	**29.17**
Intra_cluster 4	92.74	93.85
Inter_cluster 4	28.13	30.53
Intra_cluster 5	**92.7**	**95.14**
Inter_cluster 5	**24.32**	**31.19**
Intra_cluster 6	92.37	94.36
Inter_cluster 6	29.15	32.09
Intra_cluster 7	**93.46**	**95.53**
Inter_cluster 7	**31.22**	**32.153**
Intra_cluster 8	93.54	95.38
Inter_cluster 8	32.63	32.19
Intra_cluster 9	**94.13**	**94.69**
Inter_cluster 9	**30.5**	**29.19**
Intra_cluster 10	93.04	95.53
Inter_cluster 10	28.98	30.88

Table 5: intra- and inter-cluster scores for 10 different samplings for settings 13 and 14.

chose to focus on settings 13 and 14 for two reasons. Firstly, these two settings have the highest thresholds and a high threshold implies better performance. Secondly, conducting 10 evaluations for both inter- and intra-cluster similarity for 14 settings would require a lot of time.

Nevertheless, the intra- and inter-cluster similarity score is not the only factor to base our choice. Although setting 14 performs better in the y-axis and worse on x-axis than setting 13, it has 461 total number of clusters, that is 65 less clusters than setting 13. The lower number of clusters of setting 14 in a sense, counterbalances the difference in the inter-cluster similarity score. Therefore, although it would be interesting to have a graph where its points are based on the average value over 10 evaluations from random samples procedures, we consider setting 14 as the best performing parameter configuration.

4.4 Discussion

After having performed the evaluation of the clusters as summarized in table 4, we can now discuss the findings and compare them with what we thought the case would be before conducting the experiments.

Before starting the experiments and after having set the different configurations of parameters, we tried to foresee how the experiments were going to perform. Specifically, the extreme values of the threshold, ranging from 0-40 were rejected in the first place for exhibiting poor results. As a consequence, threshold values close to the lower bound of 40 were expected not to perform well. Indeed, we can observe from table 4 that both intra- and inter-cluster similarity scores are poor, compared to those with higher threshold. On the other hand, the situation is more complicated when the threshold value reaches the upper bound of 80. We have also rejected the values ranging from 80 - 100 as not being able to distinguish the web pages as belonging to the same cluster. More analytically, when we increase the threshold we expect the intra-cluster similarity score to increase and the inter-cluster similarity score to decrease. It is clear that when we increase the threshold, the intra-cluster similarity score increases, making it a straightforward criterion to base our choice. The intriguing case concerns the inter-cluster similarity, where despite the fact that increasing the threshold the inter-cluster similarity score decreases (not always), there is a limit where beyond that, the assignment of web pages into clusters becomes problematic. The reason is that the clusters that are created, are malformed in the sense that these clusters contain very similar web pages which in turn, are unable to capture the spam campaign itself. Instead, the spam campaign is split in its web pages, with those web pages assigned into high homogeneous clusters and thus causes the inter-cluster similarity score to increase. The question is then, what this upper bound should be. After experimenting by setting the threshold value equal to 80, we found out that there are too many clusters generated, trying to capture one single spam campaign, than actually needed. As a result, we decided to exclude threshold values ranging from 80 onwards and experiment with values from 45 to 75 with step 5.

Except for the sampling idea behind the results, there is also the averaging procedure to influence the output. The averaging procedure has the property of influencing the results by pulling the score up when almost identical web pages are compared. As an example, consider a cluster with 100 near duplicate web pages. If we sample and create 5 pairs of web pages from this cluster, then we will end up with 5 scores close to 100. These 5 scores are added to the final score which will then be divided by the total number of comparisons. By doing so, the average intra-cluster similarity will be affected and unavoidably will be led to a high score. To moderate this effect, the same procedure applies to every evaluation process for every parameter configuration. The averaging process then is not essentially a pitfall since all evaluation methods work with the same principles, while they are mainly affected by the sampling method.

Another intuition we had prior to conducting the experiments, has to do with the superiority of the results concerning the set of experiments with 2000 web pages per batch. The parameter of the number of web pages per batch was introduced in order to avoid the pair wise comparisons and speed up the process. It was reasonable to believe that the bigger the batch is, the more it tends to reach the pairwise comparisons state and if possible to reduce the error introduced from the join-of-the-batches algorithm. Looking at table 4 more closely, we can see that for the majority of the thresholds, the intra- and inter-scores are higher when we set 2000 web pages per batch than 1000 per batch. Only settings 2 and 7 seem to perform better with small-sized batches but still the differences are small. Nevertheless, despite the small differences for each threshold, we can see that bigger batches tend to perform better.

A significant observation from table 4 concerns the number of clusters formed for every setting. The number of clusters play an important role especially when the settings have neighbouring values concerning their parameters. As an example, we examine settings 10 and 11. The intra-score for setting 11 is slightly higher than that of setting 10. On the contrary, the inter-score for setting 11 is worse (slightly higher) than setting 10. Taking only the numbers into account, it would be a tough decision to choose the best between these two settings. One is slightly better than the other in the two scores. If we look at the number of clusters for these two settings though, we observe a significant difference. Setting 11 has 178 less clusters than setting 10 that makes it a better choice.

The number of clusters is also an indication of how well are the clusters structured. Setting 11 has 178 less clusters than setting 10 meaning less clusters per spam campaign, which leads to better reflection of the spam campaigns. If we go one step further, better representation of a spam campaign means lower inter-cluster similarity score. By setting the threshold high, we guarantee the intra-cluster similarity, which combined with as few clusters as possible we increase the possibilities for a more accurate identification of spam campaigns. Finally, setting 14 has significantly less clusters than its neighbouring settings and that is another reason to support our choice.

5 Extraction of cluster descriptions with LDA

In the previous chapter, we went through the procedure of identifying spam campaigns. The spam campaigns in form of clusters contain the web pages that are derived from spam emails. Although we have managed to cluster the spam campaigns based on the content of their web pages, we have truly no insight in their content. It would be interesting if we could describe the clusters with a few indicative words that characterize these spam campaigns. From a 7.5 months period, we have identified 461 spam campaigns and we would like to see at least the domain these spam emails belong to. In this chapter, we demonstrate the effectiveness of Latent Dirichlet Allocation (LDA) on assigning a topic to every cluster such that with a few words to be able to have an idea of what a cluster is about. In section 6.1 we introduce LDA as a topic allocation model and explain its suitability in our case. Section 6.2 explains the procedure followed in order to obtain the topics for each cluster and finally in 6.3 we present some statistics and the main results from applying LDA in order to label the clusters.

5.1 LDA

Latent Dirichlet Allocation (LDA) as presented in [8], is a generative model used to extract topics from a set of documents. The idea behind LDA is to model documents as arising from multiple topics where each topic is a distribution over a fixed vocabulary of terms. It is reasonable to believe that a document contains a mixture of topics in different proportions since documents tend to be heterogeneous containing more than one idea or theme.

The main assumption of the LDA model is that the topics are defined before any data has been generated. Having that in mind, for each document in the collection, we generate the words in a two-stage process:

1. random choice of a distribution over topics
2. for each word in the document
   (a) random choice of a topic from the distribution over topics in step 1.
   (b) random choice of a word from the corresponding distribution over the vocabulary

The main principle is that there is a predefined number of topics, (distributions over words), that exist for the whole collection of documents (left

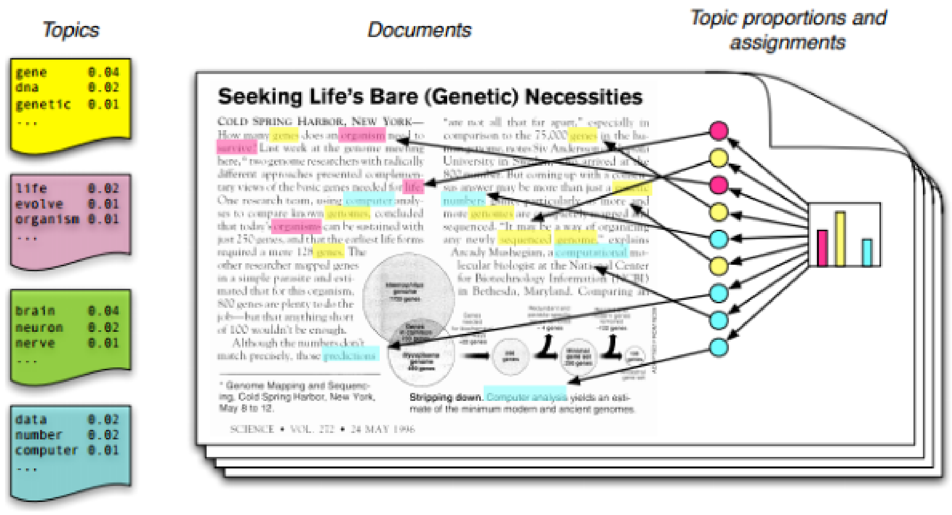


Figure 4: The intuitions behind latent Dirichlet allocation. We assume that some number of topics, which are distributions over words, exist for the whole collection (far left). Each document is assumed to be generated as follows. First choose a distribution over the topics (the histogram at right); then, for each word, choose a topic assignment (the colored coins) and choose the word from the corresponding topic. This figure is taken from [5].

in figure 4). Every document is then assumed to have originated as follows: Each document as shown at the right of figure 4, exhibits the topics in different proportions (step 1). Each word in each document is drawn from one of the topics (step 2b), where the selected topic is chosen from the per-document distribution over topics (step 2a).

The fact that there are no labels, tags, annotations and titles to describe the articles, means that we are dealing with an unsupervised method of labelling documents, where we have observed and hidden variables. The observed variables refer to the documents and the vocabulary acquired while the hidden variables pertain to the topics, the per-document topic distributions and the per-document per-word topic assignments. The main issue is to use the observed documents in order to reveal the hidden structure. This problem can be reformulated by "reversing" the generative process. The generative process defines a joint probability distribution over both hidden and observed variables. The model then, computes the conditional distribution of the hidden variables given the observed ones.

Having mentioned the principles based on which the data are generated and in order to see how the topics are actually constructed, we can divide the process of LDA in three major steps.

1. Firstly, we need to specify the number of topics.
2. Secondly, the algorithm assigns every word to a temporary topic (the two-stage process of word generation described previously.). This random assignment of the words (to be exact, they are assigned according to a Dirichlet distribution) is temporary as they will be updated in the third step. Notice that if a word has two occurrences, each word may be assigned to different topics.
3. Thirdly, the last iterative step will check and update topic assignments, while looping through each word in every document. For each word, two criteria define how the topic assignment is updated:
    (a) How prevalent is that word across topics?
    (b) How prevalent are topics in the document?

Weighing the conclusions from the two criteria, the algorithm assigns each word to a topic. This process takes place for all the words in the documents cycling through the entire collection of documents many times until there is a convergence and a roughly steady state.

The LDA model adopts three assumptions about the corpus where new research on this field tries to loosen these assumptions towards achieving different aims.

The first assumption to make is that it uses the "bag-of-words" model, in other words, LDA does not take the order of the words into account. This makes sense when one is interested in the semantic structure of the text but is not applicable for more sophisticated purposes such as language generation.

The second assumption is that the order of documents does not matter. In case that the archive contains documents that span over many years, one might be interested in knowing how the topics evolve over time. A potential solution to this problem would be the dynamic topic modelling, a model that respects the ordering of the documents and gives a richer posterior topical structure than LDA [7]

The third assumption is the fact that there is a fixed and predefined number of topics. In case where someone has no clue on what the number of topics should be, the Bayesian non-parametric topic model assists in that direction where the data are used to construct a tree like structure of topics moving from more abstract to more concrete topics [6].

The LDA model we applied does not incorporate the three aforementioned extensions. The standard model suffices to satisfy our needs.

5.2 Cluster topics

Before describing the application of the LDA model in the clusters, let us remember what the two types of clusters are, as described in chapter 3.

The first type of clusters contains web pages that are usually uninformative warning/fixed messages that differ only in a small percent (numbers, titles, names, dates etc.). On the other hand, the second type of clusters contain web pages, whose text is partially identical (two web pages have for instance a paragraph in common).

Based on those two types of clusters, it is apparent that topic selection must be applicable and adjusted to the type of each cluster. Therefore, we can discriminate between 3 possible scenarios for topic assignment in clusters. The first scenario involves clusters whose web pages are near duplicates. In this scenario, there is only one topic to describe all the web pages of that cluster. The second scenario is also about near-duplicate web pages inside a cluster but there are more than one topics. In that case, all web pages point to one main text which can contain more than one topics. While the first two scenarios have to do with clusters of type 1, the third scenario can be described as type 2 clusters, that contain documents that partially match each others.

As mentioned earlier, the purpose of applying LDA on the web pages of a cluster is to get an idea of what these clusters are about. We are not interested in a topic being fully descriptive, neither in finding all the possible topics for a cluster. What we seek is the ability to know what kind of web pages a cluster contains, in other words what is the subject of each cluster. Therefore, the application of LDA is described next in accordance with the purpose of topic modelling as described above.

The LDA model, prior to obtaining the results, asks from the user to define the number of topics. The number of topics is then taken as an input variable and one has to enter a value, most of the times without knowing how many topics there might be, especially when the dataset contains hundreds or thousands of different documents. In our case, and since we are not interested in identifying all the possible topics, we chose to get a maximum of three topics for each cluster and five words per topic. We applied LDA in every cluster and for each cluster we used the following scheme. After the topics were acquired, the first topic is automatically assigned as the first topic to represent the cluster. If there is a second topic, then we examine how many words it has in common with the first topic. If the two topics share three or more words out of the total five, then we do not include the second topic in the topic representation of the cluster. In the case where we have a third topic, we follow the same procedure and exclude it if it has three or more common words with topic one or two. Finally, the topics that represent a cluster have in the worst case two words in common. Our experience shows that clusters that contain near duplicate web pages tend to have only one topic in contrast to the clusters with distinguishable web pages that are represented with two or three topics.

5.3 Descriptive statistics

After having applied LDA to label our clusters, we present an overview of the most important findings.

A very interesting finding from topic modelling concerns the big-sized clusters. Specifically, we would like to know what the top 10 biggest clusters are about. Also, the results we acquired highlighted a problem which we were initially aware of, but not on its full magnitude. The pie chart in figure 5 shows the 10 biggest clusters along with the number of web pages written on every piece. Different colours indicate different spam campaigns and the topics are shown at the right of the figure.

The biggest cluster is coloured light blue, contains 8434 web pages and its topic is "tinyurl spam url use email". This cluster belongs to the type of clusters that contain near duplicate web pages and its text is simply a fixed, warning message. It is then very interesting to visualize the results since the top 10 biggest clusters contain 30940 out of the total 36619 web pages which is translated to 84.5% of the whole dataset. The dark blue piece does not have a topic since it is used to represent the web pages from the remaining clusters.

However, looking at figure 5 and manually inspecting the plain text of the web pages in these big clusters, we realized the big scale of the problem. In the beginning of this thesis, during the database manipulation and data extraction, we were aware of some web pages displaying a warning message and not the actual content behind these "temporary" web pages. The downloaded text of the web page happened to be a fixed warning message, where one should follow the link inside the web page, in order for the actual content to be revealed. Unfortunately, the web pages were parsed only by following the first step and that resulted in a big amount of non-informative

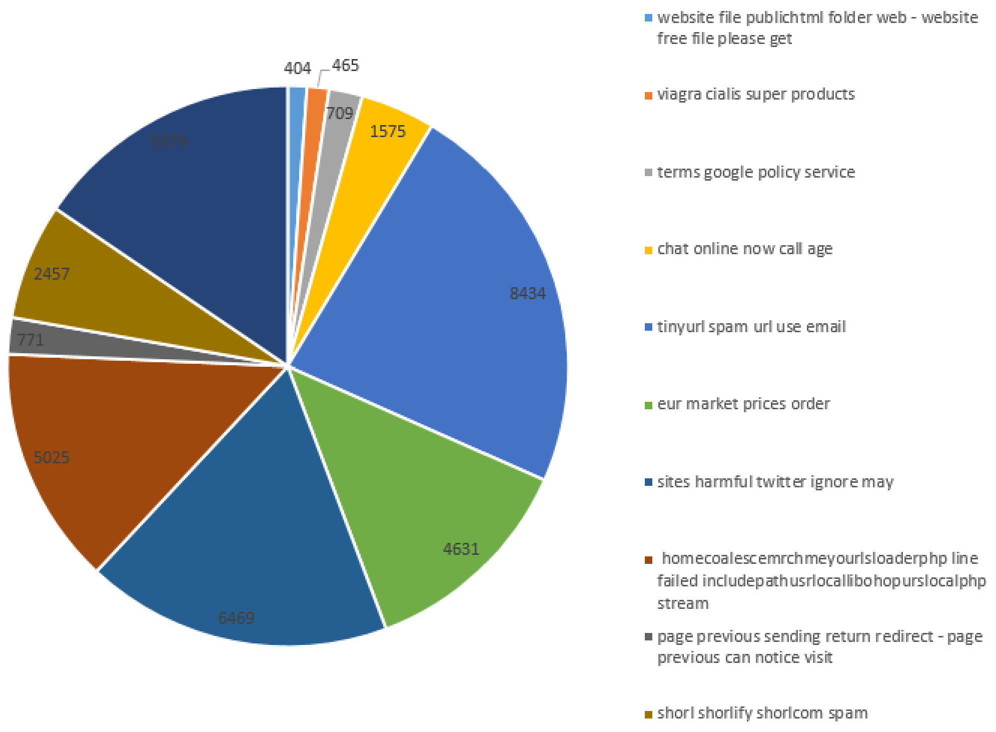


Figure 5: Topics for the 10 biggest clusters.

messages.

This phenomenon is depicted in figure 5 where many clusters contain fixed messages that are not intended by the creators of these spam campaigns. For example, the piece with 6469 is about a fixed message from twitter as this can also be seen from its topic (”sites”, ”harmful”, ”twitter”, ”ignore”, ”way”). In general, it is hard to say which clusters contain these fixed messages and which are actually designed to exhibit such messages. For example, the red cluster, which is considered very big with 5025 web pages, has a text of a few lines and its text cannot be said with certainty whether it is meant to be an uninformative fixed warning or not. The intriguing thing is that its topic contains 2 terms consisted of many syllabus and combined words. For instance, the term ”includepathusrlocallibohopurslocalphp” combines sub-terms such as ”include”, ”path”, ”local”, ”lib” and ”php”. This is a typical obfuscation technique that makes it difficult for spam filters to characterize it as spam. All in all, many clusters contain such fixed messages and are roughly estimated to be more than 50% of the whole dataset.

If we isolate those clusters, whose web pages contain beyond any doubt uninformed fixed messages, then we have the following pie chart.

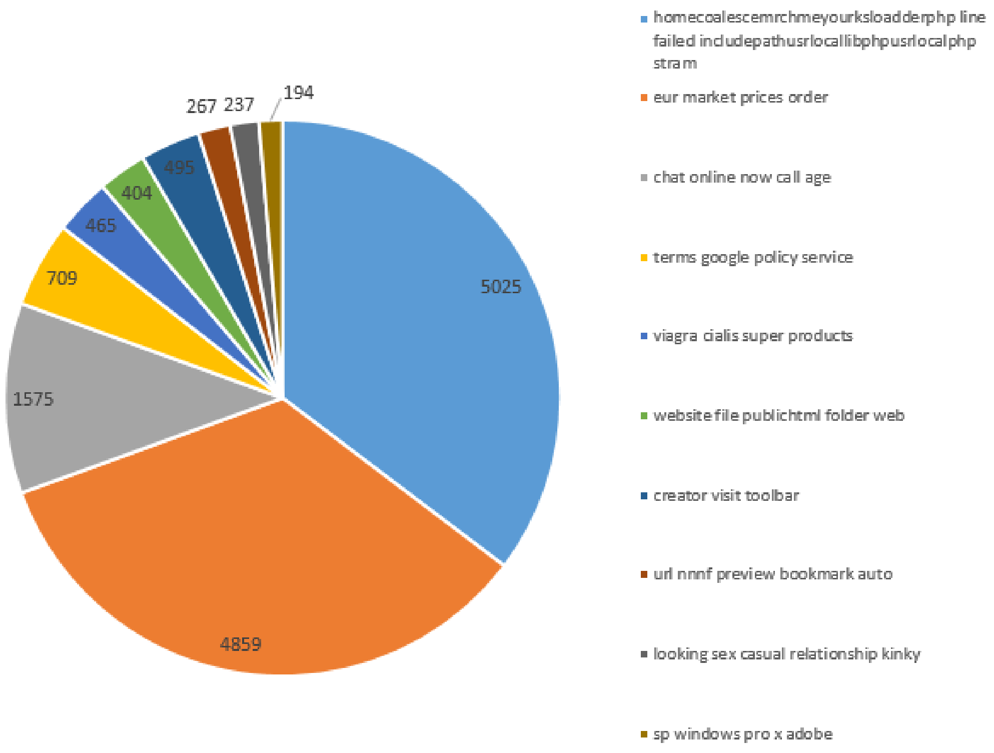


Figure 6: Topics for the 10 biggest clusters without the uninformative fixed messages.

In figure 6, we left out 18131 web pages coming from only 4 clusters, which is almost 50% of the whole dataset. The remaining web pages are 18488 and the majority of those belong to the top 10 biggest clusters (after the removal of the 4 uninformative clusters) that all together contain 14130 web pages or 76% of the dataset. Although we left out those clusters that we are certain not to contribute towards our aim, we are not entirely sure though about the rest of the clusters. Some of the remaining small clusters might also be uninformative or there might be different instances of the same spam campaign reflected by different clusters that all in all increase the percentage of more than 50%.

The size of the clusters and the knowledge of what they represent is very useful. For instance, it can serve as a guidance to see which are the most popular spam campaigns during a time period.

Beyond this scope, it would also be interesting to know what the top terms throughout our dataset are and in how many web pages they are contained. One way to achieve that, is to perform a frequency count on the whole dataset and select the top n most frequent terms. This approach though does not yield reliable results since it is highly skewed. The reason is that a big cluster for instance containing thousands of near-duplicate web pages, is certain to contain a number of words many times. As a result the majority of the most frequent terms would originate only from the big clusters and dominate the list of top n frequent terms. Although with this approach we can draw some conclusions even if the most frequent terms come from a few big clusters, we are more interested to see the frequency of terms that are evenly distributed across the documents.

The first approach to follow was to group the web pages by the cluster they belong to. In that way, we get a random web page for every cluster and perform a frequency count on the total vocabulary. Theoretically, every cluster is different than the others in the sense that it contains a unique set of homogeneous web pages. In the best case scenario, this is translated into 461 different web pages leading to one characteristic web page per cluster. The frequency count then is performed equally on all the clusters. However, due to the error margin introduced by the parameters, either the clusters are not that homogeneous or there are more than one clusters to represent a spam campaign. This, results in being the frequency count influenced by the aforementioned reasons. All in all, despite of some misclassification of web pages, frequency count performed on this type of corpus gives us an overview of the most important terms.

Specifically, to illustrate our claims, we show in table 6 the top 10 terms for the 461 web pages.

term	buy	gener	onlin	us	can	viagra	ciali	pill	mg	use
#.Of.occur.	1791	828	803	560	541	541	511	501	494	478

Table 6: top 10 most frequent terms from 461 web pages, one per cluster.

In order to perform the frequency count in the corpus of the 461 web pages, we also performed stemming. We decided to stem the words because the power of a stemmed word is assumed to be equal to the unstemmed, in terms of identifying the notion of this term. Stemming allowed us to avoid cases of retrieving different instances of one word (adjective, noun, verb etc.)

and thus allowing us to get a wider picture of the most frequent terms.

Looking at figure 7, we can obtain some useful observations. It offers a broader view of the dataset by stating the number of web pages that contain the top 10 most frequent terms. In general, 10 terms do not give a clear picture that spans all over the dataset but can be still informative. Many of these frequent terms come from certain clusters for two main reasons. If we look at the terms "viagra", "ciali", "pill" and "mg", we can understand that they are strongly correlated and most probably come from the same cluster. This is due to the fact that some popular clusters like this, appear more than once in the identified clusters. The second reason is the length of those web pages. If for example a web page selected to represent a cluster is long and on top of that contains many times some particular words, then the frequency of these terms is high and as a result they appear in our top list.

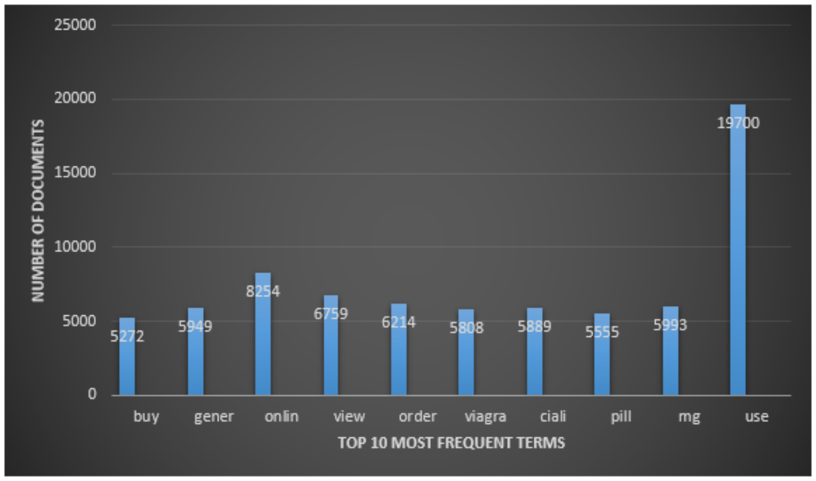


Figure 7: Number of web pages for the top 10 most frequent terms from table 6.

A different approach in order to find the top indicative and descriptive terms is to perform a frequency count on the topics acquired for each cluster. Theoretically, each topic should be different and not match with any other topic. Practically, this rule is violated by a couple of factors which are described next. Some topics have up to two words in common, which is the limit for not creating a new topic. Also, it might be the case where we have some misclassification of web pages into different clusters, leading to topic assignments that already exist. Lastly, due to the nature of the algorithm there might be some cases of more than one clusters defining a spam campaign and as a consequence similar topics to originate from different clusters.

As soon as we have acquired the topics for each cluster, we performed a frequency count on these topics, again using the stemming method. The number of topic ranges from 1 to 3 for each cluster, which means the number of terms ranges from 5 to 15 for each cluster. Finding the terms with high frequencies, gives us an idea of the vocabulary used. These top 10 most frequent terms are then extracted and for each term we compute the number of web pages that contain this term.

Before we generalize and show the number of web pages that contain each frequent term, let us see what these terms are and in how many topics they occur (see table 7).

term	line	redirect	email	web	fail	host	buy	error	websit	onlin
# Of occur.	40	29	24	24	22	22	22	21	21	21

Table 7: top 10 most frequent terms from the 461 cluster topics.

Figure 8 below presents these terms in the x-axis, while on y axis is the number of web pages. The information that we can extract from figure 4, concerns the nature of the most frequent terms. All 10 of them make great sense since these are the kind of words one would expect to see in spam emails. These terms are very general and do not include a name or a product or something more descriptive and concrete as we explain next.

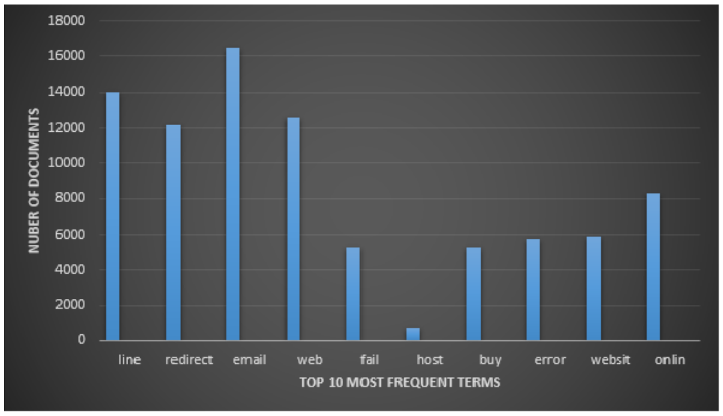


Figure 8: Number of web pages for the top 10 most frequent terms from table 7.

These two methods, used to present the most frequent terms in the whole dataset, yielded different but expected results. Trying to explain their differences, we concluded at the following. We can observe that the frequent terms from figure 8 are on average included in more web pages than those from figure 7. This is mainly explained because figure 8 contains the most frequent terms that come from the topics extracted for each cluster. Since a topic contains essentially the most characteristic terms of a cluster, it is reasonable to believe that the terms from the topics apply to a higher proportion of web pages. In contrast to the most frequent terms from figure 7, where these terms are based on the frequency count of one web page per cluster. Taking also into account the (small) error rate of the miss-assigned web pages into clusters, it supports the claim that topic terms are of general usage throughout our clusters' structure.

The second observation has to do with the nature of the terms in the two figures. It is a fact that figure 7 contains very specific terms such as names of drugs (”viagra”, ”ciali”) or even ”mg” that is an abbreviation for milligram. These kind of terms are very descriptive for the cluster they refer to, and they are contained in the list of the most frequent terms because of the frequency count on the web page itself. On the contrary, many clusters contain web pages whose text is not about a specific product or advertisement but contains more general text. The topics extracted for that kind of clusters contain more abstract terms like the ones we see in figure 8. Additionally, the number of such clusters that contain many web pages with abstract and vague topics, is the majority and as a consequence, the top terms in figure 8 originate from this type of clusters.

6 On-line and automatic incremental clustering

After the spam campaign identification (chapter 4) and the topic assignment of the clusters (chapter 5), we now focus on the 3rd main part of this thesis. In this chapter we describe the process of on-line and automatic clustering of continuously incoming spam emails. As mentioned earlier, the spam monitor receives a daily feed of spam emails that can reach up to 1 million. These spam emails are initially stored in the MongoDb database and only if they contain at least one link, they are transferred in the MySQL database that stores the attributes of the spam web pages. From now on, when we talk about incremental clustering of spam emails, we refer to those that contain a link and their language is English.

6.1 Defining "online" and "automatic" clustering

The spam emails arrive continuously and are stored in the databases for further analysis. The purpose is to build a mechanism that takes the spam emails the moment they arrive in the database, creates clusters and assigns each spam email to a cluster. In addition, on-line clustering means that it takes place almost in real time, after the spam emails are received. In our case, "almost" real time means that there is one-hour interval between every clustering of the spam emails. Every one hour, the spam emails that satisfy the conditions (pass the filters), are retrieved and processed in order to be assigned into clusters. The interval of one hour was chosen such that it is adequate to fit the time frame needed for performing the clustering procedure and small enough so that we can observe the fluctuation/formation of the formed clusters relatively often. On the other hand, the spam campaign identification in chapter 4 was conducted off-line since we had a fixed dataset of spam emails from a 7.5 months period to work with.

Automatic clustering of incoming spam emails means that we do not intervene in the process at all. The whole procedure of retrieving the dataset, executing the clustering algorithm and updating the database with the results, takes place automatically. Even in cases of malfunctions (e.g. failure of the server), it is designed in such a way that the mechanism waits until the problem is fixed, while notifying us about the cause of the problem. We explain those details later in section 6.4.

6.1.1 Threshold range

Similar to the off-line spam campaign identification, in the incremental clustering of the spam emails we need to select the parameters values of the algorithm. In this case, however, we have only the threshold as an explicit parameter. The threshold ranges from 45 to 75 with a step of 5. The size of the batches is no longer useful since the incremental clustering procedure handles only a small number of emails per hour (order of tens). This is the reason why we need to experiment again with different values of the threshold, i.e. we can not reuse the best one achieved in chapter 5. As a result, we need to test for which value of threshold do we obtain the best clusters. The experimentation set up then is consisted of 7 experiments for 7 different values of threshold.

We will use the same implicit parameters as in the spam campaign identification in section 4.2. The clusters are divided based on their size in the following categories and a corresponding sample of web pages is retrieved, as shown in table 8. Also, in order for an incoming web page to be assigned into a cluster, the score must be above the threshold for all the comparisons made.

Category	Size of the clusters	Sample size per cluster
1	1 - 5	1
2	6 - 10	2
3	11 - 50	5
4	51 - 100	8
5	More than 100	10

Table 8: Size of the clusters with the number of random web pages retrieved per cluster.

6.2 Methodology - implementation on training database

The algorithm for the incremental clustering of spam emails is based on approximately the same principles as the ones used for spam campaign identification. First of all, we needed a training dataset for two main reasons; in order to test the functionality of the algorithm and secondly to select suitable parameter values. Also, the advantage of incrementally clustering the web pages on a fixed dataset, is that we do not actually have to wait for one week. Instead, after the clustering algorithm for each batch (spam web pages that are retrieved during one hour) is finished, the clustering of the next batch occurs immediately. Therefore, our training dataset consists of those web pages that arrived during one week (17/09/2014 - 23/09/2014) with the total number of web pages during this period being 3100.

After the algorithm was tuned to the training dataset and the best parameter configuration for maximum performance is chosen, we were then in position to launch it on-line to automatically cluster the spam emails in (almost) real time.

The implementation of the incremental clustering algorithm involves several calls from Python to MySQL and vice versa. Initially, we created a table in the MySQL database that stores the clusters from the incremental procedure. Every hour, one call is made from Python to MySQL in order to retrieve the ids and the text of the web pages during the exact previous hour. Another call is made from Python, so that all clusters created so far, are fetched in order for their contained web pages to be tested with the newly arrived web pages. Moreover, we have also access to the table where the preprocessed text of the web pages (during the week of our training dataset) is stored. Next, we present the pseudo-code for the incremental clustering for the training dataset.

Algorithm 3 Incremental clustering of spam web pages for the train-database.

```
1: date = 17/09/2014 17:34:00
2: while date not equal to 23/09/2014, 24:00:00 do
3:     fetch the existing clusters from the database
4:     fetch the web pages arrived during the exact previous hour
5:     for each web page Ppr from the previous hour do
6:         for each already exisiting cluster C do
7:             determine size C
8:             select random sample R from C
9:             for each page Pr in R do
10:                if score (Ppr,Pr) >= threshold then
11:                    NrOfTrue increases by one
12:            if NrOfTrue == number of samples in R then
13:                assign page Ppr to cluster C
14:                break
15:        if PageNotAssignedToExistingCluster then
16:            create new cluster with page Ppr
17:    increase date by one hour
```

Every web paged arrived during the last hour, is tested against a sample of web pages from the existing clusters. If the web page matches (the score is above threshold) with all the web pages sampled from a cluster, it is assigned into that cluster. Then, the database is updated with the new entry and the next page is due to be clustered. Otherwise, it continues to the next cluster, where it is compared with a new sample of web pages. If there are no more clusters to compare, then a new cluster is created with the new page in it. This procedure finally stops when the condition is met.

6.3 Evaluation of clusters from training database

After experimenting with different values of threshold, we acquired the clusters for every configuration. Once again, we need to evaluate the clusters for all 7 settings and select the one with the highest performance. We use the same approach as the one used for evaluating the clusters during the spam campaign identification process. The results in terms of intra- and inter-cluster similarity scores are presented next in table 9 and the points in plane representing the settings in figure 9.

Settings	1	2	3	4	5	6	7
Intra_score	71.94	81.23	85.08	89.25	90.98	91.58	93.46
Inter_score	33.62	35.86	36.67	34.44	32.71	31.93	32.69
#Clusters	194	407	767	930	1029	1103	1166

Table 9: Intra- and inter-cluster similarity scores for different settings.

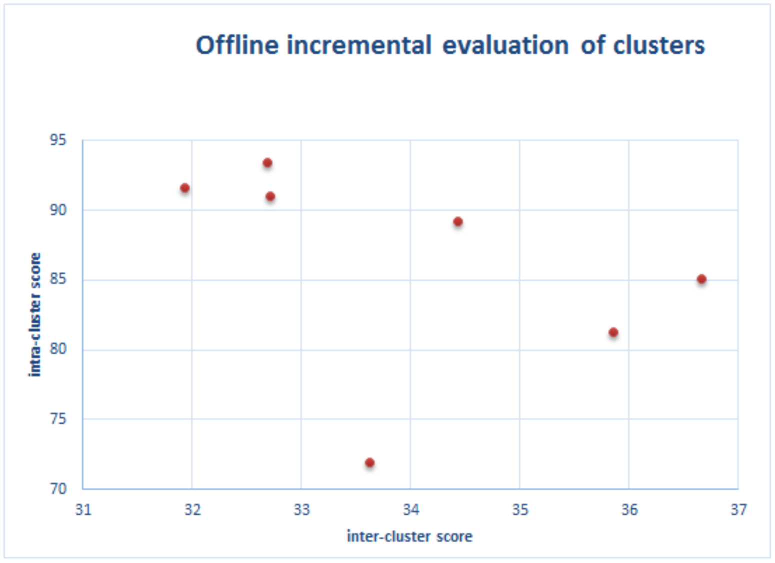


Figure 9: Intra- and inter-cluster similarity score.

Looking at figure 9, we need to select the point in the plane that combines high intra- and low inter-cluster similarity. In contrast to figure 3 in section 4.3 where we had one point both higher in y-axis and more to the left in x-axis, the situation now is more complicated. The problem of selecting the best point is a trade off between favouring intra- or inter-cluster similarity. If someone is more interested in highly homogeneous clusters, he would select the point lying higher in y-axis, otherwise he would pick the left most point in x-axis. The aforementioned criteria should be in accordance with the number of clusters for each setting. In our case, the best two performing settings are setting 6 (x-axis:31.93, y-axis:91.58) and setting 7(x-axis:32.69, y-axis:93.46). The difference in their intra-cluster score is 1.88 percentage points while their inter-cluster score difference is 0.76 percentage points. It is apparent that setting 6 performs better in inter-cluster structure but setting 7 has better intra-cluster score. From our experience, settings with high threshold, like settings 6 and 7, tend to be very homogeneous. Changing the threshold from 70 to 75 does not have such an impact on the homogeneity of the clusters as it has on their inter-cluster similarity. In other words, although the difference in y-axis is almost double than that of x-axis (1.88 vs 0.76), the difference in x axis, is more important for high threshold settings. On top of that, setting 6 has 63 less clusters than setting 7, which supports our claims for selecting setting 6 as the best performing parameter configuration.

We also performed a series of experiments for the top two threshold settings, in order to see their variance and if we could draw some safe conclusions from figure 9. Therefore, just like in section 4.3.1, for both settings 6 and 7 we evaluated their intra- and inter-cluster score by taking different samples of web pages and clusters for 10 times. The results are shown below in table 10.

Looking at table 10, we can see that both intra- and inter-cluster scores do not exhibit a big fluctuation over the 10 experiments. The intra-cluster similarity score for setting 6 for instance ranges between 91.49 (5th iteration) and 91.83 (10th iteration) producing a difference of 0.34. The inter-cluster similarity score ranges from 30.76 (2nd iteration) to 32.29 (3rd iteration) with 1.53 as a difference. Setting 7 also performs accordingly, with its intra-cluster‘s lowest score being 92.93 (2nd iteration) and highest 93.66 (10th iteration) with 0.73 difference. Finally, the inter-cluster score for setting 7 ranges from 30.88 (10th iteration) to 32.77 (7th iteration) leading to 1.89 difference.

It is apparent that in that case, the sampling method does not have a big impact and we can tell with higher certainty that setting 6 performs best

	Setting 6	Setting 7
Intra_cluster 1	**91.58**	**93.46**
Inter_cluster 1	**31.93**	**32.69**
Intra_cluster 2	91.53	92.93
Inter_cluster 2	30.76	32.12
Intra_cluster 3	**91.63**	**93.13**
Inter_cluster 3	**32.29**	**32.73**
Intra_cluster 4	91.6	93.62
Inter_cluster 4	31.37	32.63
Intra_cluster 5	**91.49**	**93.1**
Inter_cluster 5	**31.9**	**32**
Intra_cluster 6	91.58	93.45
Inter_cluster 6	31.73	32.36
Intra_cluster 7	**92.09**	**93.34**
Inter_cluster 7	**32.07**	**32.77**
Intra_cluster 8	91.6	92.98
Inter_cluster 8	32.25	32.05
Intra_cluster 9	**91.51**	**93.53**
Inter_cluster 9	**31.5**	**32.39**
Intra_cluster 10	91.83	93.66
Inter_cluster 10	32.18	30.88

Table 10: intra- and inter-cluster scores for 10 different samplings for settings 6 and 7.

among the rest based on our previous claims.

A reason for having less variation in the evaluation of clusters from the of-line incremental procedure is the size of the dataset. The training dataset contains the spam web pages that arrived during one week in contrast to the spam campaign identification dataset that spanned over 7.5 months.

6.4 Online application

The final step is then to apply the incremental clustering algorithm on-line. In order to do that, we need to make some adjustments to the algorithm used for incremental clustering in the training database.

Firstly, we describe how the task scheduler Windows tool was applied so that a call is made every one hour to the Python script. The task scheduler is designed in such a way that any program can be scheduled to run in specified points in time. In our case, the Python script was programmed to start on a daily basis and every one hour. Task scheduler offers also several other options. One of the most important ones we took advantage of, concerns the case where a problem might occur (e.g. server failure). In that case, the scheduled task that can not be executed is put in queue, until the previous one has successfully finished.

Secondly, in the incremental algorithm we added a limitation in one of the Python queries to MySQL. Specifically, for every hourly batch of spam emails to be clustered, we fetch from the database only those clusters that have been active during the last 2 months. Therefore, there are two main questions to answer. Why don't we retrieve all the clusters from the database and why did we set 2 months as a limit?

As an example, imagine that the incremental clustering algorithm is running for several months. At a specific time, when the next hourly batch of incoming spam web pages is fetched in order to be clustered, each web page must be compared against the clusters that exist in the database. The worst case scenario is to be compared with samples from all existing clusters and because of mismatches, to start a new spam campaign. Such a scenario would require hours and sometimes it is even unnecessary. In [22], the authors state that 50% of spam campaigns actually finish within 12 hours. After that the durations distributed rather evenly between 12 hours to 8 days, and about 20% of campaigns persist more than 8 days. The average duration of a spam campaign can then be used as a criterion to narrow down the number of spam campaigns needed as an index for comparisons. Ideally, we would like to identify what this average lasting period is, based on our own dataset. By averaging over the download dates of the spam web pages belonging into a cluster and then over the number of spam campaigns, we could have an estimate of the average lasting duration of a spam campaign. Unfortunately, for technical reasons that was not feasible, so the 2 months period is inserted as a time frame and assumed to have a generous error margin regarding the average time duration of a spam campaign.

Finally, we would like to know what happens in case of a malfunction. Therefore, after every hourly call from Python to MySQL in order to retrieve the web pages of the exact previous hour, 3 possible scenarios are identified:

1. At least one web page has been retrieved during the last hour.

2. No web pages have been retrieved during the last hour.

3. An error has occurred resulting in no web pages retrieved (e.g. server

failure).

For every aforementioned scenario, an indicative message appears to suggest the possible cause.

In this chapter, we have described the mechanism to incrementally cluster spam emails into spam campaigns on-line and automatically. A very interesting and important consequence is that our mechanism can be used to serve different purposes. Our intention is to use it in order to cluster the continuously incoming spam emails into spam campaigns, using a plagiarism detection measure. Someone can apply a different measure, for instance the simhash algorithm in order to detect near-duplicate documents. As long as there is a measure that can output a similarity score between two entities, our implementation can be used with some minor adjustments.

Finally, our incremental clustering mechanism is running consistently from the time it was created without any noticed malfunctions. Every one hour and for the first two months of its running mode (31/05/2015 - 30/07/2015) the Python script was being called and had already clustered 2,400 spam emails into spam campaigns.

7 Conclusions

In this chapter we conclude by giving an overview of our work and present some important directions for future work in this field.

We have indeed managed to identify spam campaigns based on the content of the web pages referred to by the spam emails. The results we presented showed that we can extract useful information by examining the content of the web pages. The referred web pages then can stand on its own as a criterion for identifying spam campaigns. Plagiarism detection based on Fuzzy String Matching was the scheme to introduce because of the special nature of our dataset. Our cluster evaluation method, despite not exhibiting very clear results in terms of selecting the best parameter configuration for an accurate spam campaign representation, in general assisted in capturing the idea and notion of how clusters are formed and helped us select the most suitable setting for a decent spam campaign identification.

After the identification of spam campaigns, we managed to label the clusters with a few but indicative words. We explained in principle, how Latent Dirichlet Allocation (LDA) works, and we were able to assign a topic to each cluster. Topic modelling in our case is important in order to see what the spam campaigns are about. Therefore, we presented some descriptive statistics and showed both the top 10 most common terms in our dataset and the top 10 biggest clusters in terms of their topic assignment.

Finally, we built an incremental mechanism that is able to cluster the incoming spam emails into spam campaigns in an automatic and on-line environment. The incremental algorithm was first tested in a training dataset. To this end, we did not use the parameter of the batches as we did for the spam campaign identification in chapter 4. In the evaluation process, the variation of the points (i.e. representing different settings) as showed in figure 9 was decreased. As a result, we are confident of having selected the best parameter configuration to be applied on-line.

7.1 Future work

Spam campaign identification has been studied a lot during the last decade, especially with the ever growing volume of spam messages spread every day on the web. Although very efficient mechanisms have been developed to filter spam emails, the need to understand how spammers work by summarizing their characteristics, is still very important. Spam filters can then be adjusted to the techniques used by spammers and become more effective.

Into that direction, the referred web page from the attached URL of the spam email can be combined with other attributes of a spam email. The more attributes of spam emails we explore, the more reliable the representation of the spam campaigns will become. Therefore, a study that could combine all those different attributes together (content of spam email, URLs, referred web pages, time stamps etc.) would lead to a more accurate spam campaign representation.

Also, our process of identification of spam campaigns used sampling. It would be interesting to see to what extent the increase in the size of samples influences the quality of the clusters. We conjectural think that the quality of the clusters, when increasing the sample size, increases to some extent, but is upper-bounded by the ability of the plagiarism detection tool to successfully identify similar web pages. A further study of the relation between sample size and cluster quality might lead to a better clustering of spam campaigns.

Another significant observation is the number of steps someone needs to follow in order for the intended text of the web page to be revealed. We encountered a serious anomaly with the text of the web pages, as the downloaded text was parsed by following the attached URL only once. That led to over 50% of the total texts to be uninformative. Further work on this subject would involve to follow the next link until the intended text to appear.

Furthermore, it would be very interesting if we could use the actual time stamp of the spam emails. Then, we could use the averaging lasting period of a spam campaign, based on our own dataset, to determine the search criteria that would maximize the performance, as explained in section 6.4. It would also give valuable insight when looking at the bursts of spam emails and associate them with major effects happening at the same time (e.g. elections of a country - bursts of spam campaigns related to this topic).

In addition, on-line and automatic incremental clustering of spam emails is performed for an indefinite time period, unless manually interrupted. That leads to some very nice opportunities we can explore in the future. The fact that we are able to store the spam campaigns and access them at any time, offers us some advantages. We can for example, search for the spam campaigns that appeared during August for two consecutive years and extract useful conclusions (e.g. size of the campaigns, duration, theme of the campaigns etc.).

Finally, the HTML code of the referred web pages could be studied in order to find out if it can be a criterion for spam campaign identification. We believe that a standard template is used to represent the texts of the web pages belonging to the same campaign. Even with some slight variations on the HTML code, we could examine its suitability in being able to identify spam campaigns. On top of that, different spam campaigns might share the same or similar HTML code layout. That could be an indication that different spam campaigns might be orchestrated by the same entity.

References

[1] Fuzzywuzzy 0.6.0 2015. `https://pypi.python.org/pypi/fuzzyquzzy/0.6.0`. Accessed: 2015-08-12.

[2] Symantec Intelligence Report may 2013. `http://www.symantec.com/content/en/us/enterprise/other\_resources/b-intelligence\_report\_05-2013.en-us.pdf`. Accessed: 2015-08-12.

[3] THE SPAMHAUS PROJECT. `https://www.spamhaus.org/consumer/definition/`. Accessed: 2015-08-27.

[4] David S Anderson, Chris Fleizach, Stefan Savage, and Geoffrey M Voelker. *Spamscatter: Characterizing internet scam hosting infrastructure*. PhD thesis, University of California, San Diego, 2007.

[5] David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

[6] David M Blei, Thomas L Griffiths, and Michael I Jordan. The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM (JACM)*, 57(2):7, 2010.

[7] David M Blei and John D Lafferty. Dynamic topic models. In *Proceedings of the 23rd international conference on Machine learning*, pages 113–120. ACM, 2006.

[8] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[9] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8):1157–1166, 1997.

[10] Pedro Calais, Douglas EV Pires, Dorgival Olavo Guedes Neto, Wagner Meira Jr, Cristine Hoepers, and Klaus Steding-Jessen. A campaign-based characterization of spamming strategies. In *CEAS*, 2008.

[11] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

[12] Peter Haider and Tobias Scheffer. Bayesian clustering for email campaign detection. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 385–392. ACM, 2009.

[13] Katherine A Heller and Zoubin Ghahramani. Bayesian hierarchical clustering. In *Proceedings of the 22nd international conference on Machine learning*, pages 297–304. ACM, 2005.

[14] Christian Kreibich, Chris Kanich, Kirill Levchenko, Brandon Enright, Geoffrey M Voelker, Vern Paxson, and Stefan Savage. Spamcraft: An inside look at spam campaign orchestration. *Proc. of 2nd USENIX LEET*, 2009.

[15] Marijn Lems. Identifying Spam Gangs Based on Shared Recourses. Master's thesis, VU University Amsterdam, March 2013.

[16] Fulu Li and Mo-Han Hsieh. An empirical study of clustering behavior of spammers and group-based anti-spam strategies. In *CEAS*, 2006.

[17] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*, pages 141–150. ACM, 2007.

[18] Feng Qian, Abhinav Pathak, Yu Charlie Hu, Zhuoqing Morley Mao, and Yinglian Xie. A case for unsupervised-learning-based spam filtering. In *ACM SIGMETRICS Performance Evaluation Review*, volume 38, pages 367–368. ACM, 2010.

[19] Mark David Spadafora. Discovering campaign based spamming strategies. 2009.

[20] Chun Wei, Alan Sprague, Gary Warner, and Anthony Skjellum. Mining spam email to identify common origins for forensic application. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1433–1437. ACM, 2008.

[21] Yinglian Xie, Fang Yu, Kannan Achan, Rina Panigrahy, Geoff Hulten, and Ivan Osipkov. Spamming botnets: signatures and characteristics. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 171–182. ACM, 2008.

[22] Li Zhuang, John Dunagan, Daniel R Simon, Helen J Wang, Ivan Osipkov, and J Doug Tygar. Characterizing botnets from email spam records. *LEET*, 8:1–9, 2008.