Article Duplicates

Brian Lynnerup Pedersen

Kongens Lyngby 2014

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet, building 303B,
DK-2800 Kgs. Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

The goal of this thesis is create is to document my work with implementing a working prototype of using algorithms to find article duplicates in a large corpus of articles. I will describe how the article inflow is currently working and how Infomedia plans on implementing my work in this inflow.

I will look at various algorithms for text comparison, and look into the possibility of creating my own or tweak an existing algorithm to better suit the needs for this thesis.

Finally I will create a summary of the work done, problems I have come upon and what the future of the project will be.

Summary (Danish)

Målet for denne afhandling er at dokumentere mit arbejde med at finde artikel duplikater, i et større artikel corpus, ved brug af algoritmer. Jeg vil beskrive hvordan Infomedias artikel inflow virker nu, og hvordan Infomedia planlægger at bruge mit arbejde i fremtiden.

Jeg vil undersøge forskellige tekstsammenlignings algoritmer, og undersøge muligheden for at lave min egen algoritme, eller modificere en eksisterende algoritme til at bedre udføre det arbejde jeg laver i denne opgave.

Til sidst vil jeg gennemgå det arbejde jeg har lavet, hvilke problemer jeg løb ind i og hvad fremtiden for projektet vil være.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in fulfilment of the requirements for acquiring an B.Eng. in Informatics.
The project is equal to 20 ECTS points.

This thesis is protected by confidentiality. No information from this thesis can be handed off to any party without signed permission.

The thesis deals with the issue of finding articles that are duplicates in a large corpus of articles. This is done using various algorithms and is implemented in C#.

The thesis consists of ...

Lyngby, 09-June-2014

Not Real

Brian Lynnerup Pedersen

Acknowledgements

I would like to thank my supervisors from DTU, Inge Li Gørtz and Philip Bille, for the help they provided to my project. Also I would like to thank my company Infomedia, for letting me do my project with them, in particular my project leader Klaus Wenzel Jørgensen and Rene Madsen.

Contents

Chapter 1

Introduction

As mentioned above, I have done this thesis for the company I work for, Infomedia[1]. I have been working for them since my fourth semester at DTU (march 2012), while studying as an IT engineer (B.Eng.). Infomedia is in short a company that deals with news monitoring.

Infomedia is the result of a fusion between Berlingske Avisdata and Polinfo in 2002, which means that Infomedia is partly owned by JP/Politikens Hus[2] and Berlingske Media[3]. It is a company with around 150 employees, of which 107 is contract employees and the rest is student aides in the various departments. Infomedia has various departments, which includes an economy, sales, analysis and an IT department amongst others.
I am employed in the IT department (*PIT - Product Innovation and Technology)* as a student programmer (student aide).

Infomedia deals with news monitoring, which means that we have an inflow of articles[4] from various newspapers, news sites, television and radio media, which we then monitor for content that is of interest to our clients. This can be a client that wishes to know when their firm is mentioned in the press or a product they

[1] www.infomedia.dk

[2] www.jppol.dk

[3] www.berlingskemedia.dk

[4] Articles are sent to Infomedia daily, this can be more than 40,000 articles per day.

are using, if that is being mentioned. Currently (while writing this thesis) the upcoming EP[5] election is going on, and politicians are using the monitoring service from Infomedia to track how they are doing in the media Infomedia[6]. Infomedia had a free for all news monitoring of the EP election, meaning that all interested could sign up for a getting a daily (weekdays) news monitoring mail from Infomedia, containing the top stories about the EP election. Amongst other things a candidate visibility monitoring was created[7]. Infomedia have also begun monitoring social media. Infomedia sells various solutions to clients, so they can get the kind of media monitoring they want.[?]

One of the things that Infomedia tries to do, is that we want to present our clients with a fast overview of the articles in which terms[8], that trigger our news monitoring, appear. Many local newspapers are today owned by bigger media houses (like the owners of Infomedia) and as such, they will feature a lot of the articles that have also been printed in the "mother paper". This will make the same (or roughly the same[9]) article appear many times in news monitoring. In an effort to make the list of articles presented to the clients, easy to look at, and preventing a client having to read the "same" article many times, Infomedia has a wish to cluster article duplicates. Infomedia can then present the client with a list of articles and in that list have further sub lists that contains duplicates of the original aritcle[10]. This also have an economic factor as clients are charged per article read.

Another issue, is the issue of copyrights and when the same article will appear in different media, but without content given from the author of that article. An example that is often happening is that news telegrams from Reuters[11] or Ritzau[12] is published in a newspaper, but without the source indication. All news media are of course interested in knowing when their material is being published in competing media. This how ever can be tricky business, as official rules on the matter is incredible fuzzy.

I will in this thesis try and look into various ways of identifying article duplicates (or articles that have a lot of text in common) within a test corpus[13] of articles, by using algorithms. The long term goal for Infomedia is having this being

[5] EP - European Parliament.

[6] `http://www.altinget.dk/eu/artikel/messerschmidt-ja-partierne-taler-vaelgerne-efter-munden`

[7] `http://www.infomedia.dk/viden-indsigt/europaparlamentsvalg-og-patentdomstolsafstemning-kandidaternes-`

[8] A term is, in short, a word or a combination of words. For the rest of my thesis a term will however only be a single word.

[9] Articles can be slightly edited in order to make them fit into the layout of the various papers.

[10] Or the longest article rather, as this will tend to contain the most information.

[11] `www.reuters.com`

[12] `www.ritzau.dk`

[13] A days worth of articles from 10/31/2013 - totalling 22.787 articles.

implemented in the inflow of articles, and having a look back functionality so that we can group duplicates not just for one day, but for a longer period of time.

1.1 Limitation

Due to the time constraints, this will be done as prototype. I will show through testing how the algorithms works, and analyse their results. These results will then be evaluated and I will comment on these findings.

Chapter 2

General - Terms and Rules

I will in this chapter go into some of the expressions and terms used throughout this thesis.

2.1 Terms

As there is a lot of terms used in this thesis, a short introduction to the most used are in order.

- **Article:** For this thesis, a digital document containing the contents of a piece of news. Could originate from papers, magazines, TV or other forms of media. For this thesis a document corresponds to an article. Articles are in their electronic form stored at Infomedia as XML files, I will throughout this thesis only deal with the part of the XML files that contains data of value to me in this assignment. This being *Tags* (see below), *Headline*, *Sub headline* and *Article Text.*

- **Corpus:** From Latin meaning *body.* In this thesis that describes the test set of articles being used a test set throughout my thesis.

- **Monitoring:** In relation to the news monitoring (news surveillance) that Infomedia does, is the act of collecting news that holds information of value to our customers.
- **Tag:** Used in Ontology[1] to create words that describes the contents of an article.
- **Term:** Basically a word. A term will be something that can be searched for.

2.2 Matching

I will in this thesis talk about false and true positives and negatives. A match will mean that two articles to some extend have the same content.

- **False Positive:** When an algorithm wrongfully identifies two articles as a match.
- **False Negative:** When an algorithm wrongfully identifies two articles as not being a match, when in fact they are a match.
- **True Positive:** When an algorithm correctly identifies two articles as a match.
- **True Negative:** When an algorithm correctly identifies two articles as not being a match.

2.3 Duplicates

In this thesis I will often use the term duplicates about article comparisons. A duplication can be an article that has been taken directly from a news feed and posted in a newspaper. Many local newspapers is owned by larger newspapers, and they will often receive articles from their owning paper. They will then print this in their own paper. Sometimes they will only use parts of the article and this will also be considered a duplicate for this thesis. As such duplication in this thesis is a way of describing how similar two articles are, rather than saying different papers are doing conscious fraud. That is a matter for another debate.

[1] http://en.wikipedia.org/wiki/Ontology_(information_science)

2.3.1 Topic Matching

When looking at article matching, there is also the possibility of having articles score pretty well by the algorithms because they are dealing with the same topic. There are cases where the article have been heavily modified, and then there would be no basis to talk about duplications, then one could talk about topic matching. The article no longer contains the same phrases, but deals with the same topic. Of course two articles could describe the same topic, but never have been related to begin with. I will not try and dissect whether this is the case, only try and indicate when I find two articles that are dealing with the same topic.

Hvornår er en artikel et duplikat (korte artikler (breaking news), hvornår er to artikler "tilstrækkeligt" forskellige?) blah about copyright rules in Denmark...

Chapter 3

Algorithms in General

Before any sort of work can be done, one must consider various algorithm to work with. There are several text matching algorithms available for free on the Internet, and if one has the money for it, there are companies that can develop a specialized algorithm for you. As I do not have a lot of money (and paying for someone else, to do an algorithm for me, kind of defeats the purpose of this whole thesis) I have gone with the first option and found a free basic algorithm on the Internet, as well as contemplated to create my own algorithm from scratch.

3.1 Initial Considerations

Infomedia already have an algorithm implemented in the inflow to make a rough comparison of the articles coming in. How ever, the thought is that a combination of several algorithms would provide a better and more granular view of the articles as they are being compared. A new algorithm should be one that was specialized in text matching. It should also be an algorithm that would work in different manner than what Infomedia already have implemented[1], as having

[1] More on the algorithm already implemented in the next section.

two algorithms that work in more or less the same fashion would not produce results of much interest.

As the current implementation is rather fast, it could prove useful to have the algorithm that is already implemented, to do the initial split and then have a slower (but more thorough) algorithm look at the *interesting* article comparisons. Initially I have looked at two algorithms to fill this need, *Longest Common Substring* and *Semaphore Tag Matching* - an algorithm I would make from scratch.

3.2 Algorithms Used

Term Frequency - Inverse Document Frequency[2] **(Cosine)**, generates a vector from each document. **I will not cover this algorithm in detail, as it is not the focus for my thesis.** This is the algorithm already implemented in the inflow today (a version of it). Each word in every article is added to a *word map* which contains all the words of all articles in the corpus being checked (which also is used to create the document vector). The word map is used to generate a weight of each word (a word occurring in many articles will have less weight than a word that is only present in a few articles). Each word generates a bit of the articles total vector, a word that occurs in all documents will have a very short vector, a more rare word will have a longer (and therefore weigh heavier) in the article vector. Bla no[**?**]

Once the word map is created, the articles are then scored. The way that this is done, is that the algorithm compares two article's vectors with each other and then returns a score based on the cosine angle between the two article vectors. This is done one article comparison at a time (although done with parallel coding to speed up the process). As the word map is generated each time the algorithm is run, the word map can (and probably will) differ from each run (if the corpus of articles are being changed). Infomedia is therefore talking about implementing this bit differently, and building a constant word map, that only gets updated with each run, not overwritten. This will ensure that common words will always have a short vector.

The algorithm then returns a list of article comparisons (based on a threshold set by the user), with article ID[3] and scores. So each comparison has the ID of *"article 1"* and *"article 2"* and their score (the angle between the two vectors, in a multidimensional universe). The closer the score is to the value 1.0 the

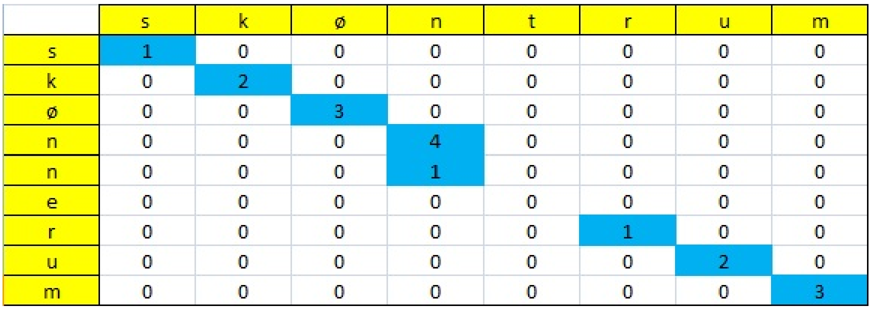
[2] `http://en.wikipedia.org/wiki/Tf-idf`

[3] Each article has their own unique ID.

more similar are two articles. A score of 0.0 indicates that two articles has nothing in common (according to this algorithm, I will discuss this point in the next paragraph), whereas a score of 1.0 indicates two perfectly identical articles (according to this algorithm). This algorithm and it has a good O-notation (O(log N)).

Problem: This string comparison is very sensitive to articles containing many of the same words, for instance "A man walks his dog in the park" and "A dog walks his man in the park" would result in returning a cosine value of 1.0, due to the way the algorithm works. The two strings are clearly different, but the words in each string is the same, therefore the Cosine algorithm will find them to be identical. This problem is very unlikely to yield false positives.

Longest Common Substring (LCS), compares documents in pairs. A general implementation would be to have a list of documents and then compare a document to every other document in the list. The algorithm will then return the length of the longest common substring. By default[4] the LCS algorithm checks the contents of a string character by character against another string. When initialized, the algorithm creates an double array and each time a match is found (when two identical characters are found ('a' and 'a' for instance)) the algorithm marks that in the array by adding a number. It then checks if this substring is longer than what has previous been found, if so, it discards the old substring and keeps the newly found.



	s	k	ø	n	t	r	u	m
s	1	0	0	0	0	0	0	0
k	0	2	0	0	0	0	0	0
ø	0	0	3	0	0	0	0	0
n	0	0	0	4	0	0	0	0
n	0	0	0	1	0	0	0	0
e	0	0	0	0	0	0	0	0
r	0	0	0	0	0	1	0	0
u	0	0	0	0	0	0	2	0
m	0	0	0	0	0	0	0	3

Figure 3.1: An example of how two words are compared in LCS. The fields in yellow are the two words broken into characters ('Skøntrum' and 'Skønnerum'). The fields in blue indicates when LCS finds a match, the number indicates the length of the substring. In this case, LCS finds three sub strings:'skøn', 'n' and 'rum', the longest of the three is the first, and this will be the result that LCS returns to the user.

[4] `http://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Longest_common_substring`

I will use the basic implementation of this algorithm in my thesis, I will then try and modify it to work better in context with finding article duplicates, instead of just the longest common substring.

Problem: This algorithm is very prone to fail in cases where there have been made alterations to the article in question. A word change in the middle of one of two otherwise identical articles will result in a 50 % match. If the article in question have been obfuscated with many changed words, the LCS will be extremely short. This is a high risk problem, as article duplication will often involve changing words. This algorithm is also substantially slower than the Cosine algorithm, having an O-notation of O(n*m). Ideally this algorithm is therefore best used on a selection of articles, rather than on the entire corpus.

3.3 Optimizing Performance

3.3.1 Stop Words

Stop word[5] removal would improve running time (performance) of both algorithms, and would pose little threat of causing either algorithm to fail (finding false positives). The exception to this could be very short articles (like breaking news articles[6]), that only contains common words, like "Man walks away". Depending on the stop word list, this article could end up being *null*[7]. We can safely (with respects to the previously addressed problem) remove stop words, as they don't provide any *semantic*[8] value to the text.

3.3.1.1 Cosine

For the cosine algorithm removal of stop words would improve performance, by reducing the size of the *Magnitude Vector*.

[5] http://en.wikipedia.org/wiki/Stop_words

[6] Breaking News articles is a thing of the 2000s. With the spread of media onto the Internet, an article is no longer a static printed piece of news in a paper. News can be published instantly on the web, and then updated as information are received. How ever, Breaking News articles can still be printed in the paper article.

[7] An article with no text data.

[8] http://en.wikipedia.org/wiki/Semantics

3.3.1.2 LCS

In regards to the LCS algorithm, the performance would also be improved, as the algorithm wouldn't need to traverse as many characters. This would reduce the length of the longest common substring, but it would be unlikely that it would affect the outcome of the algorithm, as stop words are rarely changed when duplicating articles.

3.3.2 Stemming

Stemming[9] Stemming is the act of conjugating a word to the base form (Danish: *'Grundform'*). It is done in order to reduce the amount of noise in a text. Words will then be conjugated and instead of having the same word represented several time in different conjugations they will all be noted as the same word. Stemming does not ruin the semantic content of the text.

3.3.2.1 Cosine

Stemming improves the performance of the cosine algorithm as this will reduce the number of words in the *vector space*. As words would be reduced to their base form (Danish: *Grundform*), words used several times, but with different endings would be counted as the same word. When using weighed evaluation this would actually improve performance to some extend. It can have a slight impact of the Cosine algorithm. This is because that without stemming the same word can occur several times in a text in different conjugations, and thus each conjugation would have a larger vector than when the conjugations are all counted as the same word. This impact will be minimal because the rare words are still occurring less often than normal words (stop words).

3.3.2.2 LCS

Stemming would not improve the performance of the LCS algorithm by much. As this algorithm matches characters one by one, it would make little difference if the words are stemmed or not. As the 'Grundform' is the shortest form of a word in Danish, it will make a slight difference, but this is hardly worth noting.

[9] `http://en.wikipedia.org/wiki/Stemming`

3.4 Semaphore Tagging

Another way of finding duplicates is by looking at each articles semaphore tags. When Infomedia receives articles in the inflow, these articles are enriched with *Semaphore Tags*[10]. Each article then gets a number of tags based on what terms are found in the article. A way of finding article duplicates could be through creating an algorithm that would check a pair of articles with their respective semaphore tags.

3.4.1 Cons to Semaphore

As these tags are more a general indicator to an articles contents than an actual text matching algorithm, it will provide very little value as a stand alone implementation. Each article will only contain tags for the terms worth of note. If 100 articles all contained various doings of a minister (picking up the children, going to meetings, being involved in a crisis) many of the same tags would be present in the 100 articles, this could be the ministers name, political party and other general tags that are linked to this minister. It would therefore be hard to decipher much information that could truly and uniquely link two articles together just by doing this. However this could be used to enhance another algorithm (for instance one of the two mentioned above). I will therefore not look any further into implementing this algorithm in this thesis.

3.5 Text Preparation

To reduce the chance of the algorithms failing in detecting duplicates, the text should be 'normalized'. As there are quite a few pitfalls in text analysis, one should try and take as many precautions as possible. A common source of error is common spelling errors, I will not check my article for spelling errors. These can have a rather big impact on the LCS algorithm. However as all text editors today have spell checking, this will be a tiny error source.

Another problem with text analysis is localized spelling. An example of this can be the Swedish town of Malmö. The issue here being the 'ö' letter which is in

[10] Semaphore tags are tags that describes the contents of the article, an article about financial fraud would contain the tag *Economic Crime*. These tags are created by the Infomedia Ontology (`http://en.wikipedia.org/wiki/Ontology_(information_science)`) team. They are creating rules for when a certain word (or words) appear in certain context, then an article will be tagged with a certain semaphore tag.

the Swedish alphabet. This town's name can be spelled in a few different ways.

- Malmö (Swedish spelling)
- Malmø (Danish and Norwegian spelling)
- Malmo (English spelling)
- Malmoe (Phonetic spelling of the 'ö' character)

The same article could be in different newspapers, but with different spelling of the word. All words, or rather words containing special characters (non English characters) should therefore be normalized. In this case, the character 'ö' could be normalized into the letter 'o'. The phonetic spelling of the 'ö' character is highly unlikely to occur, as the normalized way of spelling 'Malmö' in languages without the 'ö' vowel, would be the third option. I will in this thesis not normalize text, but accept that minor deviance in scores can occur due to this.

Stemming is another good way of normalizing text, I have already covered this topic in a previous section, and will not cover this more in detail here. I have not used stemming in this thesis.

Stop words will reduce the number of words in an article, and with less words, there are less error margin in terms of spelling errors. As stop words have little meaning when trying to figure out if two articles match, it is a good idea to remove these in order to improve performance. I will in this thesis remove stop words[11] to see if this have any effect.

[11] Danish - 912 most frequent words.txt - file included in the appendixes

Appendix A

Appendix

This appendix is full of stuff ... add more

Document from Infomedia about the general Vector Spaced Search.
Thesis document from Inge
Links to Wikipedia
The algorithm course book..
Example XML document (article) - to show what the XML documents look like.

Bibliography

[Oce06] Ocean. *Nobody*. 2006.