# Identifying Near Duplicates in Email Corpora for eDiscovery

John Brogan, B.Eng.

A thesis submitted to University College Dublin in part fulfilment of the requirements of the degree of

M.Sc. in Business Analytics

Michael Smurfit Graduate School of Business

*August, 2015*

Supervisor(s): Dr. James McDermott

Head of School: Professor Ciarán Ó hÓgartaigh

# Table of Contents

## Acknowledgements

The author would like thank Dr. James McDermott for all his support, ideas, insights and answering all my questions.

Thank you to Colman Morrissey, Colm Murphy and Stephen Bowes in Espion for supporting the project and giving it the go ahead.

Thank you to Damir Kahvedzic, Jie Yang and Conor Gavin in Espion for all their help, ideas and answers.

# List of Figures

# List of Tables

## Abstract

Most commercial and academic solutions identifying duplicate emails in the eDiscovery process focus on the text body field solely or equally across the most commonly used fields. These approaches tend to identify exact duplicates well but struggle to identify near duplicates. This research attempts to investigate if there is middle ground between these two approaches by exploring what features of emails are most useful to identifying exact and near email duplicates.

In order to find alternative solutions, one needs to better understand what constitutes a duplicate and how they arise. As a result of attaining a better understanding of duplication, the author proposes a new definition for email duplicates. Ultimately a novel decision tree classification based approach is proposed, which would seem to have potential to accurately identify a wide range of duplicates by intelligently utilising email features and thus improve the eDiscovery process by speeding it up and making it more accurate.

# Chapter 1 - Introduction

The core theme of this dissertation is to improve the understanding and process of identifying duplicate emails across datasets and ultimately improving the eDiscovery process. This chapter describes the context, motivation and objectives of the project.

## *1.1 Research Context: eDiscovery and Email Duplication*

The vast majority of evidence for modern civil litigation cases and corporate/governmental investigations is in electronic form. The process of discovering and preparing that electronic evidence for a trial or investigation is called Electronic Discovery or commonly just eDiscovery. Over the space of a decade, eDiscovery has become an industry of its own right, worth approximately $5.5 billion in 2013[1].

The eDiscovery process involves sifting through all the available ESI (electronically stored information) relevant to the case or investigation. In a litigation case, it is the defendant's and plaintiff's lawyers that need to examine the ESI to determine if it is evidence. While technology allows fast, efficient and accurate searching of ESI, search results can still produce hundreds, thousands or tens of thousands of documents that may be of interest. The lawyers then have to review the documents individually to ascertain if a document is evidence or not. Lawyers' time is expensive and case preparation time is not infinite, so it is imperative that only the most relevant documents are provided for review.

The multi-user and semi-organised nature of electronic data storage, especially in corporate environments, means that duplicates of documents are kept in several locations. Removing duplicates can reduce the volume of documents for review significantly, as over 50% of ESI being emails Conrad (2007) and duplicate documents being found to make up 30% of ESI [2] [3](i.e. removing duplicates, reduce the volume by 30%).

[1] http://www.complexdiscovery.com/info/2013/06/20/an-ediscovery-market-size-mashup-2013-2017-worldwide-software-and-services-overview/
[2] http://www.discovery-assistant.com/Download/NearDuplicates.pdf
[3] http://www.law.pitt.edu/DESI3_Workshop/Papers/DESI_III.Equivio.pdf

"*Emails are usually at the top of the list when it comes to potentially relevant electronically stored information (ESI) sources. They often capture critical business correspondence, agreements, business documents, internal company discussions etc.*"[4]

Most eDiscovery software identifies exact email duplicates by creating a digital fingerprint of each email, based on the content of the most important and common email fields, such as the text body, subject line, date-time sent, to, from, CC fields and others. If a single character is different between two emails, they are near duplicates but will have completely different digital fingerprints. Changes can occur in several ways, including those below[5]:

- Date-time can change due to delays in transmitting the email, nature of an email application or time zone differences.
- The subject line can change when forwarding or replying, if the sending or receiving email's application is set up in another language. For example 'fwd' can be translated to 'wg' in German.
- The main email text body can change because of different interpretations of formatting between email applications.
- The email addresses in the To, From, CC and BCC fields can vary due to the addition of 'friendly names' in some email clients if a name is saved with an email address. For example John Smith's email address might be john@smith.com, it appears like that in his email application but the recipient might have his name saved and appear in emails as 'John Smith <john@smith.com>'.
- The exact same email can be sent to the same person or different people at another time, also creating non-exact duplicates (It should be noted this would not be considered a duplicate by everyone).

It is these differences that can cause simple and fast deduplication tools to miss near duplicate emails. Ultimately if there is a high number of these types of emails in a corpus, lawyers may waste time reviewing the same emails.

[4] http://www.meridiandiscovery.com/articles/email-forgery-analysis-in-computer-forensics/
[5] For more information on the sources of NDDs see: https://www.ablebits.com/office-addins-blog/2013/10/10/remove-duplicate-emails-outlook/

### *1.2 The Gap: Business Problem*

Espion is a consultancy firm that assist law firms from a technological perspective with the eDiscovery process. In fact Espion carries out a large proportion of the pre-review process (due to its technical nature) on behalf of the law firm, including exact duplicate and near duplicate detection (NDD).

Espion works principally with two commercial eDiscovery technologies (from Symantec and Nuix) and both have de-duplication functionality built in. These products, like most eDiscovery platforms, identify exact duplicates well but Espion has found their near duplicate functionality to be hard to use, provide bad quality results and take an unreasonable amount of time. They have undertaken extensive searches and trials of other near duplication commercial tools but have found them ineffective and/or expensive.

Espion has developed their own simple, quick and cheap approach (to be referred to as the Espion approach from here on in) to identify near duplicate emails. The approach simply involves cleaning the data and then comparing certain metadata[6] fields of an email with the most recently sent email. If all the fields are the same, that email is identified as a duplicate (see Section 3.5.1 for a full description of the Espion approach).

The Espion approach is an adequate solution but the data cleaning step is generally the most time consuming aspect of the approach and it is worth investigating if there are approaches that provide at the least the same quality of results but do not require the data to be cleaned. The basic and limited comparison operation in the Espion approach leads to a wide variety of near-dupe scenarios that cannot be identified. For example, an email can only be compared with a single email, the one that was sent prior to that email. While this approach scales well with size increases in the corpus (i.e. n emails leads to n comparisons), it is worth looking for an adequate approach that produces better results. There would also be a business benefit if an approach could be developed into software and bring automation.

[6] Metadata refers to the fields found in the header of an email that provide information about the email for users and ensure the delivery of the email to recipients. Examples of Metadata fields include: From, To, CC, BCC, Subject, Attachment Names, Sent DateTime, MessageID.

### *1.3 The Gap: Academic Relevance*

As will become evident in the following Literature Review chapter, there is a significant dearth of academic work carried out in the area of NDD for emails. As a result there is lack of understanding about the best approaches to identify a wide range of near duplicate scenarios. In particular there is value to better understand the email features (e.g. 'Sent DateTime', 'Message ID', 'From', 'To' fields) that support NDD identification best. It would also be good to understand what particular established techniques, heuristics and/or algorithms apply well to this problem.

### *1.4 Project Objectives*

- This has been an academically neglected area, arriving at a better definition (e.g. what is a duplicate?) and understanding of the problem area is an over-arching project objective.
- Building on Espion's approach by exploring the possible improvements that feature selection, similarity analysis, existing heuristics and techniques can bring to the solution. Specifically looking to improve on the efficiency and accuracy of Espion's approach.
- It is possible to build de-duplication solutions to solve particular eDiscovery scenarios, e.g. for specific technologies, where data extraction & processing is carried out by the user of the solution, where the user receives data that has been extracted by multiple parties and is therefore in multiple formats. But the objective of this project is to develop a solution that is general use across all scenarios.

### *1.5 Methodology*

The project proposes a new definition for duplication with five different duplicate types: Exact, Near, Far, Content and Non duplicate. In order to assign pairs of emails to these five categories, decision trees are used to better understand the valuable email attributes for this type of classification and ultimately measure the usefulness of decision trees for identifying duplicates.

Email corpora were fabricated for the experimentation due to the lack of a labelled dataset with an adequate density and range of duplicates. Decision trees were built with a variation of tree building algorithms, string distance metrics and time distance representations. The

results from the decision tree experiments compared favourably with experiments with the Espion and commercial software (Symantec's Clearwell).

### *1.6 Project Contributions*

This research has made several business and academic contributions:

- A novel decision tree classification approach to email deduplication.
- A corpus of labelled duplicate emails, which also includes a wide range of duplicate scenarios that can be used for future research in the area.
- A novel, broader and more informative set of definitions of email duplication types.
- A better understanding of the accuracy of the Espion approach relative to standard eDiscovery software such as Clearwell.
- Extended understanding of the attributes and measures that benefit email deduplication.
- Software that automates the sorting, comparing and labelling in the Espion process.
- Foundations of software that uses decision tree classification to provide a more accurate and informative deduplication approach and that does not require spending time on normalising the data.
- A comprehensive compilation of email near duplication scenarios, to an extent which can't be found elsewhere

# Chapter 2 - Literature Review

This chapter reviews publications of previous work that relate to near duplicate email detection. By understanding what has been done in directly and indirectly related domain areas, it is possible to establish the current knowledge gap in the domain.

The following chapter illustrates the massive lack of published knowledge around the use of metadata for NDD of emails. However, there has been much work undertaken on NDD of emails (in general), webpages and many other types of digital documents. The experience and knowledge from these domains will hopefully be transferable to this project's topic.

## *2.1 Email Near Duplication Detection*

The specific study at an academic level, of NDD for email, is very sparse. This is especially the case for academic work on this topic, in the context of eDiscovery.

However one paper that does examine email near duplication for use in eDiscovery was published by IBM staff, Joshi, et al. (2011), attempts to solve the problem by grouping three types of near duplicate emails. They use standard text body analysis to group emails into syntactically similar themes, semantically similar themes and email threads. While the paper's objective is similar to that of this project, they attempt to solve the problem the same way you would a standard text document. This approach is limited when emails often contain very short amounts of text in their body.

A lot of relevant studies have been published on spam-filtering or document management. Seshasai (2009), Kumar and Santhi (2012) and Wang, et al. (2007) all studied email NDD to assist email spam filtering, from different perspectives. But again metadata does not play a role in their NDD processes.

While Yang and Callan (2006) did not look specifically at email NDD, their paper examining NDD of public comments to US regulators, raised an interesting point regarding the variations or levels of duplication in documents. They proposed the following subcategories of near duplication:

- Block Added: Add one or more paragraphs (<200 words) to a document
- Block Deleted: Remove one or more paragraphs (<200 words) from a document
- Key Block: Contains at least one paragraph from a document;
- Minor Change: A few words altered within a paragraph (<5% or 15 word change in a paragraph)
- Minor Change & Block Edit: A combination of minor change and block edit
- Block Reordering: Reorder the same set of paragraphs
- Repeated: Repeat the entire document several times in another document
- Bag-of-word similar: >80% word overlap (not in above categories)
- Exact: 100% word overlap.

It would be interesting to investigate if subcategories of duplication exist in the email domain and if they would benefit the eDiscovery process.

### *2.1.1 Email Near Duplication Detection Using Metadata*

Considering how relatively little has been published about email NDD in general, even less can be found describing work using or considering metadata as a channel for NDD. In fact the author was unable to find any academic papers that used metadata in email NDD examination.

The most significant source that referenced the use of metadata for email NDD is a Symantec white paper Symantec (2011) entitled 'Deduplication in Compliance Accelerator and Discovery Accelerator'. The white paper describes in general how the "Metadata Analysis" and "Content and Metadata Analysis" NDD methods within Symantec's Enterprise Vault product. At the time Symantec did not have an eDiscovery solution but bought an emerging eDiscovery company a few months later and subsequently moved this functionality over to the Clearwell eDiscovery platform[7] (used by Espion).

The paper states that the 'metadata analysis method' "is intended as a quick and quite effective pre-assessment mode". It lists the metadata attributes that are used in the process as:

[7] http://www.symantec.com/connect/forums/discovery-accelerator-discontinued

- Author Display Name
- CC Display Name
- Content (It truncates the content to 120 characters)
- Conversation ID
- Modified Date
- Number of attachments
- Number of CC
- Number of To
- Subject
- To Display Name
- Message Type

The paper also warns that the metadata only method can provide false positives because "it cannot evaluate the entire content of each item". On the flip side it also warns that documents can be "found to be not similar, but content-based deduplication may find the items to be duplicates". It is clear from the paper that Symantec clearly see this method as a "pre-assessment" process that is quick but nothing more, as they don't trusts its accuracy.

The only other reference that could be found that mentioned the use of email metadata for NDD is in a US patent application by IBM, Sachindra, Ng and Singh (2011). Within this patent there is only a single mention of metadata being used for similarity or duplication detection:

*"Similarity can also be determined in one or more embodiments of the invention based on communication group (for example, sender, receiver, bcc, etc.) via use of, for example, exact or near duplicate techniques. For other digital documents, one or more embodiments of the invention include using a similar strategy with the document content and meta-data"*

Research for other email NDD related content from IBM yielded nothing. It was not even possible to find an IBM product that claimed to have email NDD capabilities.

### *2.1.2 Commercial Near Duplication Detection*

Email NDD functionality is widely available in standalone packages or as an in built feature of larger eDiscovery products. Symantec Clearwell[8] (see previous section too), Wave Software[9], Equivio[10], Discovery Assistant[11] and Lexis Nexis[12] are just a few examples of companies that offer email NDD solutions.

Unfortunately they provide very non-specific high level details of how exactly their products work. Generally their NDD related content is written with senior management and sales in mind, so it tends to be more about what the functionality does, how it performs and benefits the user, rather than how it works.

For example Discovery Assistant provides a 14 page document for their NDD solution ImageMAKER Discovery Assistant[13], which identifies near duplicates for various types of files including email. But only six paragraphs are spent detailing the algorithm behind the product. Even then it provides very little technical insight, but it is clear that it performs a text body analysis.

*"Discovery Assistant first indexes all the text from the documents in the set, then programmatically searches the index database for matching phrases contained in the search document"*

Equivio published a case study, Equivio (2012), for their general NDD technology but again it is devoid of any technical information regarding the products process for NDD. Espion's eDiscovery technology partner Symantec, were unwilling to divulge useful information regarding their NDD functionality. Other than the Symantec (2011) paper covered in the previous section, Symantec's next most insightful openly available information on email NDD, is a short web page Symantec (2014) listing the email fields it uses "to determine whether two emails are considered duplicates:"

[8] http://www.symantec.com/content/en/us/enterprise/fact_sheets/b-clearwell-review-and-production-module_DS_21198619.en-us.pdf
[9] http://www.discoverthewave.com/products/107
[10] http://www.equivio.com/product.php?ID=38
[11] http://www.discovery-assistant.com/Download/NearDuplicates.pdf
[12] http://www.lexisnexis.com/en-us/about-us/media/press-release.page?id=1362090768583590
[13] http://www.discovery-assistant.com/Download/NearDuplicates.pdf

- Sender email address
- "To" list (normalized; in sorted order)
- "From" list (normalized; in sorted order)
- "cc" list (normalized; in sorted order)
- "Bcc" list (normalized; in sorted order)
- Subject (normalized; alphanumeric characters only)
- Sent time (normalized to UTC format, hours and minutes only to the milisecond)
- Full text of email content (alphanumeric characters only, as extracted by our text extractor)
- Count of enclosed emails
- Attachment file names
- Attachment sizes

This can be considered an updated list of the fields stated in Symantec (2011). Other vendors seem to have similar policies of keeping their NDD techniques close to their chest, as one can understand.

### *2.1.3 Summary*

Some interesting and related content can be found regarding email NDD. However, it is the relative shortage of investigative work carried out in this domain and specifically with respect to using metadata, which highlights an academic gap here. There is also a lack of evidence of metadata being used commercially (especially metadata only). This and Espion's experience of difficulty finding a suitable email NDD tool (lightweight and cheap), points to a strong business gap in this very specific domain.

## *2.2 Document Similarity*

Exact and near duplication detection is essentially just finding similar items. For this reason there is much to be learned from the wide range of well-established sub-topics under 'Document Similarity', such as distance measurement, feature weighting, similarity thresholds and short text similarity. Xiao, et al. (2011) in fact undertook a large examination of "efficient similarity joins" that could be used for NDD of web pages. Another study Hajishirzi, Yih and Kolcz (2010) developed "a novel near-duplicate document detection

method that can easily be tuned for a particular domain" via similarity learning, but specifically targeted the web news article and email domains in this paper.

For the most part, like duplication detection, document similarity uses the content of the document to measure the level of similarity. Sometimes metadata is used in conjunction with the content, depending if it is available, but very rarely on its own.

### *2.2.1 Using Metadata For Measuring Similarity*

There are a handful of studies that examined the use of email metadata or header, as they all referred to it. One study by Zhang, Zhu and Yao (2004) that evaluated statistical spam filtering techniques, used email metadata in one experiment to examine the benefits for spam-filtering of email metadata in general and also particular fields individually. They concluded that "the information from message headers is as important as the message body". In relation to using metadata fields, they found that "classifiers trained on message headers actually yielded comparable or better results than the message body solution".

Jake and Mee (2000) utilised metadata for email classification and found again that when metadata was used on its own, classification performance dropped. However they did find a benefit in the metadata only method, as it reduced "the cost of featurization", "the feature index, and hence the memory footprint of incremental learning methods" and the cost of "feature selection and projection". There seems to be a trade-off between performance and cost when utilising a metadata only strategy for email classification.

The impact of metadata analysis for patent classification has also been studied in (Richter and MacFarlane (2005). They confirm that "many document classification systems apply their classifications only on the basis of the document text". They cite the lack of useful metadata, the in-ability of commercial classifiers to mix linguistic or semantic analysis with analysis of structure data such as metadata and the human nature to primarily decide on classification based on the content, as the main reasons for under-use of metadata.

Spotting a gap the authors experimented by including metadata in their patent classifier. They found that "the use of metadata significantly and consistently improves automated

classifiers". This result would give hope that the use of metadata may be beneficial but might depend on the document type and quality of the metadata.

### *2.2.2 Short Text Similarity*

The topic of short text similarity is concerned with measuring the similarity between two short text strings or one short text string and a larger text string (i.e. a document). There isn't a universally agreed definition of what constitutes a short text segment but the range of its length is generally considered to be between one and twenty characters. It is a commercially important area, especially for search engines and their main revenue stream pay-per-click advertising.

Short text similarity is of great importance to this project due to the short nature of metadata text lengths. While there is much potentially to be learned from standard document similarity techniques, Yih (2009) and Donald, Dumais and Mee (2007) have both stated that standard similarity measures such as Jaccard similarity are not effective when the number of words are few such as they are in short text segments. Yih (2009) proposed a specific short text similarity measure called Tweak that focused on the nuances of term/feature weighting. Donald, Dumais and Mee (2007) took an overall view of possible techniques to improve short text similarity, including various text representation practices and also looked at appropriate similarity measures such as probabilistic and lexical matching. Yih and Meek (2007) developed a machine learning approach that learns from that data and improves the similarity measures.

While not specifically covering short text similarity 'A Survey of Text Similarity Approaches' by Patel and Rana (2014) provides an excellent overview of the similarity techniques most commonly used. The diagram below comes from that paper:

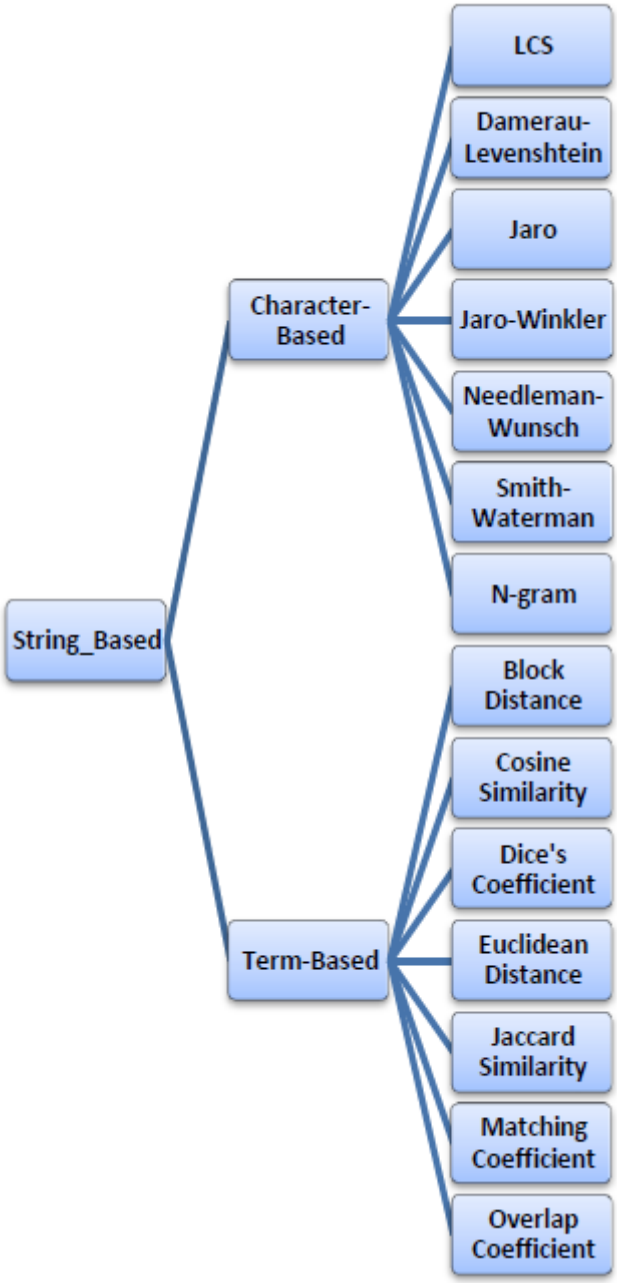


Figure 2a: String-Based Similarity Measures Patel and Rana (2014)

### *2.3 Email Classification*

As previously mentioned email classification is a well-covered subject area and its relevance to this project will be discussed in Section 3. A brief summary of publications exploring email classification indicates a trend of the approaches commonly used:

- Klimt and Yang (2004) Support Vector Machine, Naïve Bayes, Term Frequency–Inverse Document Frequency.
- Matwin and Kiritchenko (2011) Support Vector Machines, Naïve Bayes and their own "co-training" approach.
- Wenjuan, et al. (2014) Decision Tree, Support Vector Machines, Naïve Bayes and K-Nearest Neighbour.
- Shi, et al. (2012) Decision Tree.
- Meizhen, Zhitang and Sheng (2009) Fuzzy Decision Tree, Support Vector Machine, Neural Network, K-Nearest Neighbour

The above papers offer useful insights into what works and doesn't work when it comes to classifying emails. The question is then if they lend themselves to classification of duplicate emails.

# Chapter 3 - Methodology

This chapter details how the initial hands on experimentation led to the core ideas and objectives of this research and what was done to realise those objectives and test those ideas.

## *3.1 Exploring Data*

As well as studying publications and discussing NDD with Espion's eDiscovery consultants, exploring a real email corpus was a great way of better understanding what is a badly defined and under documented subject area. Espion provided a real dataset from a case they were involved in. Here are the main details on the corpus:

- Approximately 200,000 emails
- Approximately 20,000 of those, the Sent DateTime field could not be extracted
- Mixture of emails in several formats including .txt, .html, .pst, .tiff and email attachments in numerous formats such as .pdf, .exl, .txt

With such a big dataset, various scripts (in python) were built to make the analysis manageable. The scripts analysed the data in similar ways that Espion approach would, with a few small changes. These scripts aided the identification of exact and near duplicates and the different scenarios that lead to them. Exploring this dataset was an eye opener, as there were several aspects of it that weren't expected, for example:

- The subjective nature of what is and isn't a duplicate, for example:
    - The same out-of-office message sent to the same person multiple times or sending the same email to the same person more than once (possibly by mistake)
    - Separately sending the exact same email to multiple people
    - Meeting request response emails, as no usually adds content to them so only the From address is different
- The sparse examples of duplicates across the dataset. Approximately there was a near duplicate in every 300 emails. Although, it is possible that duplicates were not found

by the methods used to find them (i.e. false negatives) and this was not an example of a particularly duplicate dense dataset according to Espion

- The 'Number of Attachments' field was empty (This occurred again in creating a dataset in a particular email client, see Section 3.3)
- The majority of the content in the 'From', 'To', 'CC' fields were in username (e.g. JBloggs) format, rather than the email address (Again, this would happen in the fabricated email dataset, see Section3.3)
- Emails with minimal metadata (e.g. only 'Subject', 'To' , 'From') created a lot of false positives
- No examples of different versions of the same email with a time difference greater than 10 seconds (due to networking hiccups) [14]
- False positives due emailing different people separately by blind copying them
- Text body changing from one version to another because of different formatting in the recipients email client or provider
- No examples of the subject line abbreviation changing as had been mentioned by an Espion consultant and read about (e.g. Fwd: changing to the German form Wg:). Although there was no reason to expect that there would be in this particular dataset.

The exploration of this helped get a clearer understanding of what the problem of identifying for Espion really was. It also explained the nature of emails, duplication and eDiscovery. However, it also highlighted the unsuitability of this or any real dataset (such as the Enron dataset) to investigate (or create ideas for) new approaches, due to the sparse and narrow range of duplicate examples in any real email corpus.

### *3.1.1 Proposed New Duplicate Definitions*

As a result of the discovery of the subjectivity of defining a duplicate, the author proposed the new definitions (below) of five duplicate email types. The purpose of these definitions is to allow any approach or software to capture all the possible forms of an email duplicate and allow the user to decide which of those four types is of interest.

[14] For more information on the sources of NDDs see: https://www.ablebits.com/office-addins-blog/2013/10/10/remove-duplicate-emails-outlook/

- **Exact Duplicate** emails are versions of the same sent email that have all the same data (i.e. no changes)
- **Near Duplicate** emails are versions of the same sent email, which cannot be identified straightforwardly by a computer as duplicates due to changes in their content due to technology.
- **Far Duplicates** are emails (1) sent separately (2) to the same recipient(s) (3) but contain the same content (i.e. the same Subject, Text Body, Attachments)
- **Content Duplicates** are emails (1) sent separately (2) to different recipients (3) but contain the same content (i.e. the same Subject, Text Body, Attachments)
- **Non Duplicates** are emails that are not duplicates of any form

### *3.2 MessageID Field*

The majority of email providers create a MessageID for each sent email and add it to the email's header before it is sent. While it widely available, the Espion approach does not make use of it, even though it is a quick way of identifying exact and near duplicates (when it's is available in both emails in a pair). There is evidence that the MessageID is used in eDiscovery for threading and as an identifier of a message, but little evidence of its use in de-duplication.

However, it is not available for all emails or can be changed on the recipients end, depending on the source provider and email application[15][16][17] when creating an email corpora, there are examples of the MessageID field not being available and changing from sender to receiver. Depending on the technology and entity extracting metadata from emails, it is common that the MessageID field will not be available for any of the emails (see Section 3.3). This was the case for the Espion corpus. Finally, if you are looking to identify 'Far' and 'Content' duplicates, MessageID will obviously not be able to help you.

For these reasons the MessageID is the not the silver bullet to solve near duplicate problem. However, the author intends to use it where possible in the new approaches explored in this project.

[15] http://stackoverflow.com/questions/831183/is-the-message-id-email-header-unique-for-each-recipient
[16] http://www.pcreview.co.uk/threads/how-can-i-display-change-the-message-id-in-the-header-of-a-mail-sent-or-to-be-sent.3661672/
[17] https://social.msdn.microsoft.com/Forums/office/en-US/5afcc89d-c139-4c66-9b48-d2861b4ffe28/messageid-property-in-outlook-sent-items-folder?forum=outlookdev

## *3.3 Fabricating Email Corpora*

### *3.3.1 Why Create an Email Corpus?*

In order to discover how to improve the process of identifying near duplicate emails, new approaches need to be tested and a labelled corpus of duplicates and non-duplicates is required to benchmark new approaches. A corpus that contains a good representation and range of the common duplication scenarios is also required. There should also be examples of scenarios that could lead to false negatives and false positives, in order to fully examine approaches' performance.

Unfortunately, there are no existing corpora available (including the real case corpus made available by Espion or the openly available Enron dataset) that meet the requirements above. While it is not scientifically ideal to test one's theories on data one has created oneself, it was this felt that this was the best solution given the context and constraints. It was also believed that there would be something to be learned from attempting to recreate scenarios on demand. It was accepted that results and conclusions based on tests with this data would be somewhat limited by the artificiality, small size and source of the data.

Two corpora were fabricated; one from sending real emails to fake custodians[18] and another of fabricated metadata which allowed the reproduction of scenarios that either happen randomly or with technology not available to the author.

### *3.3.2 Method for Building the Fabricated Email Corpus*

This section deals specifically with construction of an email corpus called 'FabEmail', distinct from the construction a dataset of fabricated metadata, 'FabMeta', which is discussed in the next section. The first step was to research and compile the types of duplicate that can occur and how they occur. By speaking with experienced eDiscovery consultants in Espion, exploring Espion's sample client dataset, reading relevant published papers and other online content, a range of duplicate scenarios was compiled, see below:

[18] Custodian is a widely understood and used term to refer to the individuals in the scope of an evidence discovery process.

### Near Duplicates Scenarios

- By simply sending emails it was hoped that changes in the 'Sent DateTime' value between versions of the same email, as is regularly mentioned in published articles (and found in the Espion client data) as a source of near duplication.
- Setting up and sending emails from a non-English language email account in a non-English language browser. This was hoped to replicated published and anecdotal evidence that the subject line abbreviation (e.g. Fwd: and Re:) can be translated from sender to recipient if one email account/client is set-up in another language.[19]
- Multiple replies and forwards in an email thread, as some email accounts deal with mixtures and multiples of 'Re:'s and 'Fwd:'s in a thread in different. e.g. "Fwd: Fwd:" can be "Fwd:[2]" or just "Fwd". [20]
- Changing account setting of the subject abbreviation feature (above), in the middle of an email thread.
- Adding language character accents (e.g. Irish fada), symbols (e.g. € or %) and images to the subject line and text body, aiming to create near dupes due to formatting.
- Sending emails from different types of email clients (e.g. Microsoft Outlook, eMClient and Mozilla Thunderbird) to see if this creates near duplicates.
- Sending meeting requests, as email clients and providers deal with these differently.
- Forwarding and replying in plain text (note: forwarding in plain text leads to the attachment getting dropped).
- Saving custodians' email addresses in the 'address book' and giving them a nickname, which could lead to a different format of the email address.
- Using the StudentNumber@ucdconnect.ie address format instead of the Student.Name@ucdconnect.ie format.
- Sending emails with a time delay on the sending which could lead to a difference in the 'Sent DateTime' between versions.

### Far Duplicate Scenarios

- A custodian sending the exact same email to the exact same custodian(s) separately. Possibly as a mistake, auto generated or periodic email.

[19] Good overview of issues with subject line abbreviations: http://blog.freron.com/2013/re-the-complexity-of-a-simple-prefix/
[20] Good overview of issues with subject line abbreviations: http://blog.freron.com/2013/re-the-complexity-of-a-simple-prefix/

- Sending the same auto-reply (e.g. Out Of Office reply) to the same custodian multiple times.

**Content Duplicate Scenarios**

- Different custodians forwarding the same email without adding any content or editing anything.
- The same sender, sending the same meeting request to different people separately.
- Different custodians accepting or declining the same meeting request, without adding any content.

**False Negative Scenarios**

(i.e. near duplicate email that an approach may identify as non-dupe)

- Exact same email being sent from and to the exact same people (i.e. exact duplicate) but with another unrelated email in corpus sent in-between the exact dupes.

**False Positive Scenarios**

(i.e. Non duplicate emails that an approach may consider a duplicate)

- Sending emails with same (1) subject title and/or (2) text body and/or (3) attachments, but with something of the three different. Some within 60 seconds and/or in the same minute i.e. both 18:59.
- Replying to emails (and replying to replies) quickly and with minimal additional text, so fields are fairly similar.
- Sending reply emails from the same original email but from different custodians (sometimes with minimal metadata or content).
- Forwarding the same email to the same custodian but from different senders.
- Sending emails with minimal amount of metadata and text body.
- Sending very similar emails within a short space of time (i.e. a few seconds)

**Custodians & Email Addresses**

The next step is to pick and create email accounts to use in the fabrication process. Five email accounts were used to send emails, one account per fictional custodian. Two of the accounts were existing accounts and the other three were set up for this purpose:

- **Work** – real work account john.brogan@espiongroup.com (Microsoft Outlook /Exchange based)
- **College** – real college account john.brogan@ucdconnect.ie or 13205318@ucdconnect.ie (Gmail based)
- **English Personal** – Gmail account set up for this project englisghpersonaladdress@gmail.com
- **eM Client** – Hotmail account set up for this project emclient2015@hotmail.com with the intention of using the eM Client email application to access this account.
- **German Yahoo** – Yahoo account set up at the German version of Yahoo germanaddress@yahoo.de

For the purpose of replicating reality and creating certain scenarios, three accounts were set up to represent people that are not custodians in a case.

- noncustodian1@outlook.com
- noncustodian2@hotmail.com
- noncustodian3@gmail.com

**Email Clients & Providers**

In order to replicate reality and to have the best chance of creating the duplication scenarios desired, different email clients were used across the different accounts:

- Gmail, Yahoo, Hotmail and Outlook/Exchange via desktop web browser
- Outlook/Exchange via mobile web browser
- Yahoo, Gmail and Hotmail via their mobile android applications
- Outlook/Exchange via its desktop application
- Yahoo via the Mozilla Thunderbird desktop email client (set up with German language settings)
- Hotmail via the eM Client desktop email client

**Sending Emails & Extracting Metadata**

With all this set up, the next step was to send the emails. This was done gradually over the course of roughly a week. The native metadata export functionality from the browser and client based accounts varies, and unfortunately none export all the fields required. In order to extract the metadata, all the accounts were synced via IMAP and POP3 protocols, to

Microsoft Outlook's desktop client and extracted from there. There was a concern that the email metadata could be altered in the syncing process. This concern was alieved when:

- Sample comparisons of the email data within Gmail, outlook etc. and the data exported to CSV via Outlook showed no differences
- An Espion consultant explained that they'd never experienced the data changing
- An internet search produced no evidence of the data changing

The native export to CSV functionality in Outlook is limited so an add-on to Outlook was sourced (CodeTwo Exporter[21]), which exports all the fields with very little hassle. It should be noted that CodeTwo only extracts the first 256 characters from the text body. This is a similar method to other commercial eDiscovery tools, for example Symantec's eDiscovery software analyses the "first 120 characters of the content" (Symantec 2011). A java PST (see Appendix 2) parser built by Espion was used to get the MessageID field out of the emails, as the other tools couldn't extract the field.

### Labelling the Corpus

With the metadata extracted, the data was labelled. The labels were defined so that they could be applied to real eDiscovery situations i.e. the need to keep one version of each duplication and identify the remaining emails as duplicates, so they can be removed. For this reason the following logic was applied:

- The sent email in a group of EXACT or NEAR duplicates is not itself considered a duplicate. All the recipient emails are considered the duplicate of it. This relates to the reality that in a group of duplicates all but one email will be disregarded or deleted. These emails are labelled as '**Non Dupe (Sent Email, But Has Dupe)**'.
- In the case of FAR or CONTENT duplicates, the email sent first will be considered a non-duplicate and all other emails in the group are the duplicates. The label '**Non Dupe (First of Dupe Series)**' was used for these.

The label 'Non-Dupe' was used for emails that were not duplicates of any form of any other email in the corpus.

[21] http://www.codetwo.com/freeware/outlook-export/

### *3.3.3 Results of Building Process*

Table3a below is a sample of the email created in the FabEmail dataset.

| Custodian | From | To | CC | Subject | Text Body | Attachment # | Attachment Names | MessageID | Sent Date | Label |
|---|---|---|---|---|---|---|---|---|---|---|
| Work | /o=ESPION/ou=first | noncustodian1@outlook. | | at sollicitud | Morbi pretiu | 0 | | | 25/06/2015 17:24:46 | Non Dupe |
| Work | /o=ESPION/ou=first | john.brogan@ucdconnect | | commodo n | Quisque por | 1 | Mauris tincidun, neque.txt | | 25/06/2015 17:38:49 | Non Dupe (Se |
| College | john.brogan@espi | john.brogan@ucdconnect | | commodo n | Quisque por | 1 | Mauris tincidun, ne | 0F1003CE184 | 25/06/2015 17:39:52 | Near Dupe |
| Work | /o=ESPION/ou=first | englishpersc | john.brogan | Mauris a he | torquent pe | 2 | Sed egestas metus.txt;Aenean in. | | 25/06/2015 17:48:17 | Non Dupe (Se |
| English Gmai | john.brogan@espi | englishpersc | john.brogan | Mauris a he | torquent pe | 2 | Sed egestas metus.t | 0F1003CE184 | 25/06/2015 17:49:30 | Near Dupe |
| College | john.brogan@espi | englishpersc | john.brogan | Mauris a he | torquent pe | 2 | Sed egestas metus.t | 0F1003CE184 | 25/06/2015 17:49:30 | Exact Dupe |

Table 3a: Sample of data from the fabricated corpus of email (FabEmail)

**Interesting Notes on Duplication Results in Fabricated Email Corpus**

The creation of this corpus had the dual purpose of creating a useful labelled dataset for testing with, and to provide a better understanding email duplication. Below are interesting results from building the corpus:

- No examples of small changes between 'Sent DateTime' due to networking hiccups.
- No examples of subject line abbreviations changing due language settings differences between senders and receivers (e.g. English 'Fwd' to German 'Wg').
- There were many examples of near duplicates due to formatting differences, especially when special characters and images were in the text.
- There were many example of near duplicates due to differences in the email addresses. For example 'john.brogan@espiongroup.com' is written as '/o=ESPION/ou=first administrative group/cn=Recipients/cn=JBrogan' in the header of emails in Outlook/Exchange but as 'john.brogan@espiongroup.com' in all the other accounts.
- No examples of near duplicates due to differences in 'Subject' formatting.
- There were examples of near duplicates due to difference in the 'Attachment Number' field.
- The Message ID was available for sent emails from Outlook client.
- Sending emails with a send delay (e.g. one minute delay), leads to near duplicates due to differences on the 'Sent DateTime'.
- It was not possible to test if automatic replies created near duplications as automatic replies are not saved in the sent folder.

**High Level Stats**

- There are 153 emails in the dataset
- The dataset represents 67 sent emails and 86 received emails
- The sent date range over the corpus is 10 days
- Dataset includes the following breakdown of duplicate type:
    - 13 Non Dupes
    - 2 Non Dupes (First of Dupe Series)[22]
    - 47 Non Dupes (Sent Email, But Has Dupe)[26]
    - 51 Exact Dupes
    - 34 Near Dupes
    - 5 Far Dupes
    - 1 Content Dupes

**Summary of Fabrication Process**

The process of creating the 'FabEmail' corpus and the corpus itself was useful. However, it was not possible to recreate all the desired scenarios. This leads into the next section detailing the fabrication of another dataset called 'FabMetadata', comprised of metadata only.

## *3.4 Creating a 'Fabricated Metadata Dataset'*

### *3.4.1 Why Create a Metadata Dataset?*

Fabricating metadata is not the ideal approach scientifically but it is the best practical solution considering the project's constraints. The alternative would be to send thousands of emails in the hope that particular scenarios would occur or semi-manually weeding through tens or hundreds of thousands of real world emails. Creating metadata makes it possible to have a dense and wide range of duplications, including possible false positive scenarios, which is not possible by sending real emails.

### *3.4.2 Method for Building the Fabricated Metadata Dataset*

The process to building this dataset were similar in many ways to process of fabricating emails but with subtle differences. Similar custodian structure, labelling and scenarios were used. However, as there was no experimentation to see what would happen in certain

[22] The 'Non Dupes (First of Dupe Series)' and 'Non Dupes (Sent Email, But Has Dupe)' classification tags are used to provide an extra level of insight across the different types of non duplicates.

scenarios, the aim was to create metadata that covered all know duplication scenarios and types, as well as contriving metadata that could lead to false positive duplication identification.

**Scenarios**

Like creating FabEmail, the first step of creating FabMetadata was to identify the duplicate scenarios that were to be replicated, these included the scenarios from the FabEmail process plus a few new scenarios that came to light in that process:

Near Duplicate Scenarios

- Altering text body (different spacing and special characters changes).
- Different subject line abbreviations.
- Attachments not available for some email versions.
- Small time sent differences to mimic: (1) networking changes (2) sending delays.
- Using the friendly email name for one version and the standard address for the other versions.

Far Duplicate Scenarios

- A custodian sending the exact same email to the exact same custodian(s) separately. Possibly as a result of a mistake, auto generated or periodic email.
- Sending the same auto-reply (e.g. Out Of Office reply) to the same custodian multiple times.
- Sending the same content to the same recipients multiple times, possibly days apart, possibly mimicking sending the same report on a periodic basis.

Content Duplicate Scenarios

- Different custodians forwarding the same email without adding any content or editing anything.
- The same sender, sending the same content separately.
- Sending the same auto-reply to the same forwarded emails from different custodians.
- Forwarding the same email to different custodians separately.

False Negative Scenarios

(i.e. near duplicate email that an approach may identify as non-dupe)

- Exact same email being sent from and to the exact same people (i.e. exact duplicate) but with another unrelated email in corpus sent in-between the exact dupes.
- Sending duplicate emails with recipients in the BCC field only.
- Missing MessageID.
- Combing multiple scenarios (i.e. differences) from above in one dupe pair.

False Positive Scenarios

(i.e. Non duplicate emails that an approach may consider a duplicate)

- Sending emails with same (1) subject title and/or (2) text body and/or (3) attachments, but with something of the three different.
- Replying to emails (and replying to replies) quickly and with minimal additional text, so fields are fairly similar.
- Sending emails with minimal amount of metadata and text body (might have the same text body of subject).
- Sending very similar emails within a short space of time (i.e. a few seconds).
- Multiple custodians forwarding the same email to the same custodian at the same time (i.e. possible false near of far dupe).

**Custodians & Non-Custodians**

Five custodians were invented randomly:

- Joe - joe.smith3765@gmail.com
- Tim - timmurphy@work.com
- Sam - sam.smith@hotmail.com
- Liz - liz-gordon@outlook.com
- Rita - rita.reynolds@college.edu

Two fake non-custodians were also used.

- non-custodian@email.com
- non-custodian1@mail.org

**Corpus Building Process**

As there were no emails to actually send, it was a case of putting together the specifics around those scenarios (i.e. sender, recipients, subject etc.), recording and labelling them in a spreadsheet. To match the character length of the Text Body string in the fabricated email dataset, the Text Body was cut down to 256 characters, in emails where it had originally been longer. Unlike the previous exercise, the MessageID had to be fabricated. In reality these IDs are composed in a specific structure, which varies across email providers. It was felt that to mimic these structures closely would be time consuming and bring virtually no value, as these IDs are only checked for exact duplication so any random string would work the same. These randomly generated 10 character strings were produced at www.random.org.

**Labelling the Dataset**

The same duplication labels detailed in section 3.3.2 were used for the FabMetadata dataset.

### *3.4.3 Results of Building Process*

**High Level Stats**

- There are 141 metadata representations of emails in the dataset
- The dataset represents 67 sent emails and 74 received emails
- The sent date range over the corpus is 12 days
- Dataset includes the following breakdown of duplicate type:
    - 7 Non Dupes
    - 4 Non Dupes (First of Dupe Series)
    - 37 Non Dupes (Sent Email, But Has Dupe)
    - 39 Exact Dupes
    - 35 Near Dupes
    - 13 Far Dupes
    - 6 Content Dupes

## 3.5 Approaches

After significant literature research, discussions with Espion eDiscovery practitioners, exploration of the Espion datasets and building email and metadata corpora, the approaches detailed in this section were considered interesting to develop and test further.

### 3.5.1 Current Espion Approach

The current Espion approach to identify near duplicates is a manual process of cleaning and identifying some types of near duplicates. It is limited to identify only particular duplicate scenarios by its rigid comparison function and the fact that it can only compare an email with the previously sent email. The pre-processing required for the process to work is time consuming, to the degree that it can make the process not worthwhile. The process is explained in detail below:

Espion Approach Pseudocode:

1. Extract relevant data via a eDiscovery platform or directly from email files to a single database (csv, MS Access, other)
2. Pre-process the data:
    - Converting the Sent Date/Time to UTC format, if not already done. Then truncate it to leave only the date, hour and minute if not already done (i.e. remove seconds and milliseconds)
    - Converting subject line abbreviations to English
    - Normalise email addresses
3. Sort order of emails in database by ‘Sent DateTime’.
4. For each row (i.e. email) in the database, compare the From, To, CC, Subject, Attachment Names, Attachment # and Sent DateTime, with that of the previous row (i.e. previously sent email).
    - If all these fields are exactly the same, label as ‘Duplicate’
    - Else, label as ‘Non-Duplicate’
5. QA a sample of identified duplicates by manually comparing the identified duplicate email pair files
6. If QA process satisfactory, remaining identified near-dupes are tagged as such in the eDiscovery platform

### *3.5.2 Proposed Decision Tree Approaches*

Initially the idea of calculating a similarity or 'confidence of duplication' measure between emails in a corpus was considered as a possible method of improving on the Espion approach. One key new aspect of such an approach is the move away from only doing a comparison with the previous email in the corpus i.e. only O(n) comparisons instead of $O(n^2)$ . This is the shrewdest aspect of the Espion approach, as it allows it to do the analysis very quickly and it scales linearly with the size of the corpus.

However it is worth investigating if there is value in losing processing efficiency for the benefit of improved accuracy. By only comparing with the previous email there is the potential to miss Near duplicates (where the Sent DateTime has changed between versions and another unrelated email is in-between), Far duplicates and Content duplicates.

Without cleaning (normalising) the data, the Espion approach would be much more inaccurate. This proposed approach will attempt to maintain or improve accuracy but remove the time consuming normalisation step.

As well as comparing across the dataset, with the author's proposed new email duplicate types definitions (i.e. Exact, Near, Far, Content and Non duplicates), a multivariate classification approach is required. A similarity or 'confidence of duplication' measure approach does not lend itself well to identifying more than two classifications. This lead the author to consider if applying decision tree analysis to the duplicate identification problem could improve the process. Experimentation by building decision trees with FabEmail and FabMetadata will hopefully provide insight into the email attributes that are most useful to identifying duplicates and classifying the different types of duplicate.

Below is a high-level overview of the proposed decision tree approach:

1. Calculate the distances between the From, To, CC, Subject, Text Body, Attachment Number, Attachment Names, Message ID and Sent DateTime attributes of each pair of emails.
2. Use a well-established decision tree building algorithm to build a decision tree based on the distances from (1)
3. Prune the trees if it is of benefit
4. Use the final tree to classify the pairs of email based on the distances from (1)
5. Map the classified email pairs back to the individual emails and in turn identify individual duplicate emails

Experimentation will involve testing different decision tree variables to provide insight into those aspects that most useful to the problem. Variation will include:

- Measuring the distance or dissimilarity between emails.
- Building decision trees.
- Attribute selection from email i.e. the attributes that benefit duplication identification

**Distance Metrics**

There are no widely accepted metrics for calculating the distance or dissimilarity between emails. Term-based similarity/dissimilarity metrics are most commonly used in email deduplication, for example (Xiao, et al. 2011) and (Hajishirzi, Yih and Kolcz 2010) use Cosine and Jaccard Similarity for deduplication of all types of documents. Such term-based approaches compare the number of common k-sized terms between documents to measure their similarity. However, the significance of email metadata and the fact that the metadata and email content can be very short, mean that term-based metrics would not seem the most appropriate approach to measure email dissimilarity (although it would be worth investigating this in future research).

Character or lexical based similarity metrics would seem more suited to measuring the distances between text fields within emails. These metrics purely look at the difference between strings. Levenshtein distance or Edit distance as it's also known, is the most well

established and commonly used approach. It is measured by calculating the number of edits (insertions, deletions and substitutions) required to transform one string into another.

Jaro distance is another character based distance metric that "is based on the number and order of the common characters between two strings; it takes into account typical spelling deviations and mainly used in the area of record linkage". Both of these metrics are also suitable for emails because they can handle strings of different lengths (not every character metric can handle this). In order to take into account the lengths of the strings, the Levenshtein and Jaro ratio will be used in experimentation.

Two of the fields of interest are numerical, 'Sent DateTime' and 'Number of Attachment'. The latter is integer and can remain an integer. The 'Sent DateTime' distance will be calculated in seconds, although when it comes to using this attribute to build a decision tree, the square root or logarithm of the value may be more useful, so it is worth experimenting with this.

**Decision Tree Building Algorithms**

Two established algorithms will be used in the experimentation; C.5.0 which is an extension widely used C4.5 classification algorithm[23] and CART. An overview of both algorithms is provided below and comes from (Patel and Rana 2014).

| | C5.0 | CART |
|---|---|---|
| Type of data | Continuous and Categorical, dates, times, timestamps | Continuous and nominal attributes |
| Speed | Highest | Average |
| Pruning | Pre-pruning | Post Pruning |
| Boosting | Supported | Supported |
| Missing Values | Can deal with | Can deal with |
| Formula | Use split info and gain ratio | Use Gini diversity index |

Table 3b: Summary of C5.0 and CART classification algorithm (Patel and Rana 2014)

[23] More information: https://cran.r-project.org/web/packages/C50/C50.pdf

The following also come from (Patel and Rana 2014):

**CART**: "*CART is a non-parametric decision tree learning technique that produces either classification or regression trees, depending on whether the dependent variable is categorical or numeric, respectively. The word binary implies that a node in a decision tree can only be split into two groups. CART uses gini index as impurity measure for selecting attribute. The attribute with the largest reduction in impurity is used for splitting the node's records. CART accepts data with numerical or categorical values and also handles missing attribute values. It uses cost-complexity pruning and also generate regression trees.*"

**C5.0**: "*C5.0 algorithm is an extension of C4.5 algorithm which is also extension of ID3. It is the classification algorithm which applies in big data set. It is better than C4.5 on the speed, memory and the efficiency. C5.0 model works by splitting the sample based on the field that provides the maximum information gain. The C5.0 model can split samples on basis of the biggest information gain field. The sample subset that is get from the former split will be split afterward. The process will continue until the sample subset cannot be split and is usually according to another field. Finally, examine the lowest level split, those sample subsets that don't have remarkable contribution to the model will be rejected. C5.0 is easily handled the multi value attribute and missing attribute from data set*"

## 3.6 Experimentation

### 3.6.1 Current Espion Approach

Analysing the data with the Espion approach will provide the baseline set of results to measure the accuracy performance of other proposed approaches. The FabEmail and FabMeta corpora were analysed separately by the Espion approach, to support the comparison with proposed approaches (explained in Section 3.5), plus there would be change in the results by combining the corpora, as the process is based on sorting on date time and comparing with the previous email.

The first step was to clean and prepare the data, by normalising email addresses & subject line abbreviations and truncating the Sent DateTime by removing the seconds. A script (in python – Appendix 3) was built to automate the process of sorting, comparing and labelling process, which can be used by Espion going forward.

### 3.6.2 eDiscovery Software

In order to have a baseline to compare the Espion and proposed approaches against, the two datasets were processed by a market leading eDiscovery platform mentioned previously Symantec Clearwell. This software deduplicates emails only if they are deemed exactly similar. It considers a certain subset of email fields to compare for duplication. Clearwell will output a similarity score for all pairs but will only duplicate if this is 100%. It is expected that the Espion and proposed approached will provide more accurate results.

As FabMeta was not based on real sent emails, Espion provided a java script (see Appendix 5) for building .eml email files from csv based data. These email files were loaded into Clearwell for deduplication.

### 3.6.3 Proposed Decision Tree Approach

This section details the experimentation process to investigate the viability of using decision tree concepts to identify duplicate emails. Rather than proposing a specific approach, this experimentation is an exploration of various different ideas that will hopefully indicate a particular approach that is best suited. As previously mentioned, the experimentation with decision tree approaches also has another purpose of learning more about nature of email duplication and classification, so the experiments were designed with that in mind too.

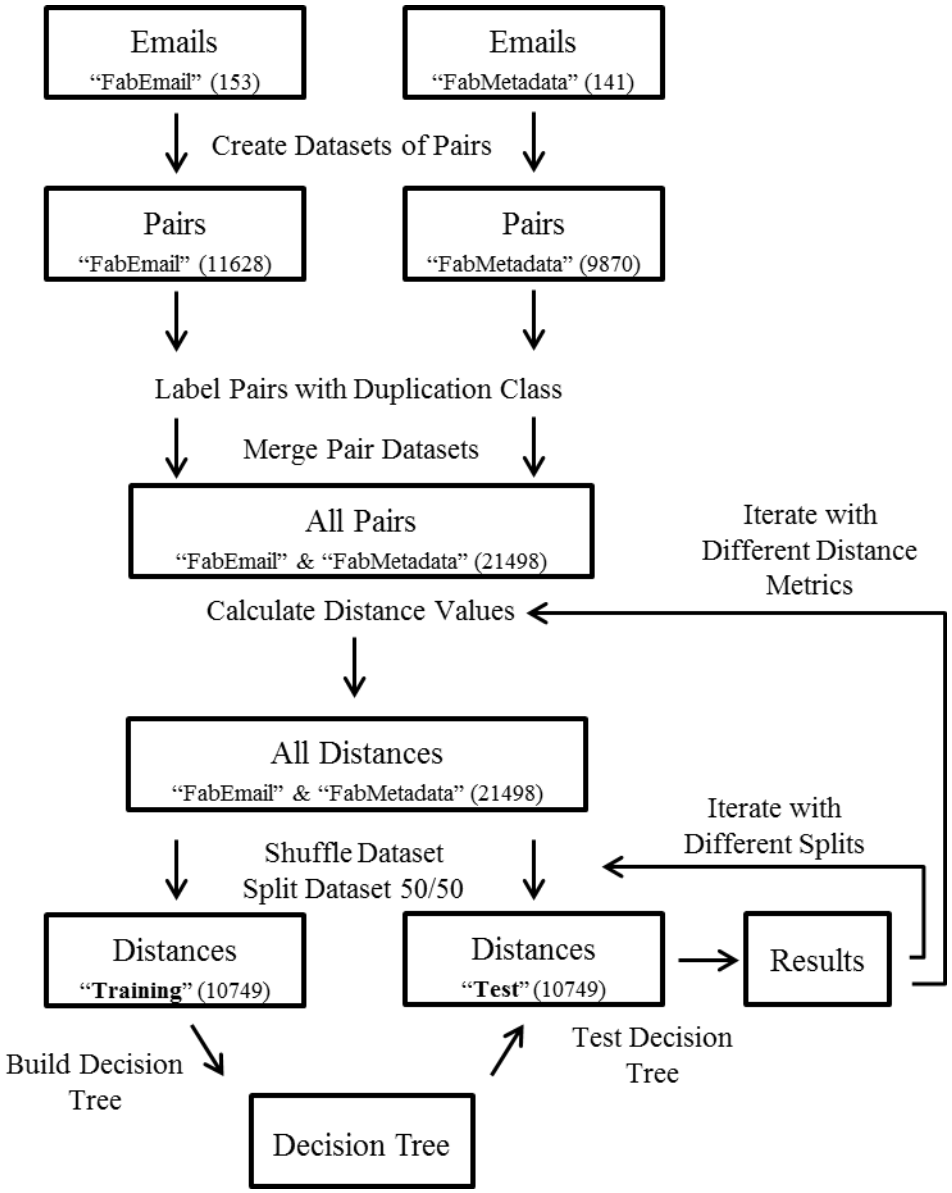


Figure3a: Map of experimentation process with decision trees

**Decision Tree Dataset Design**

With two datasets to test on, there were a number of methods of merging them so as to build a decision tree with them. The original email corpora have high duplicative density but the duplicative density significantly reduces for a dataset of all the pairs in the corpora. Merging the corpora before unique pairs were mapped would result in a significantly bigger number of data points (i.e. pairs) but with all those additional data points being non-duplicative pairs, hence the duplicative density would reduce even further. For this reason the datasets would be merged after pairs had been mapped. Thus the data, decision trees would be built with, would only contain distances between emails in the same corpus.

### Building the Dataset

The duplicates of each email were labelled manually. A python script was built and used to create the datasets of pairs. Then by using 'VLookUps' the pairs were labelled from the identified duplicates. Various manual and automated checks were carried out to ensure (as much as time allowed) that the dupe labels of each pair were correct. Below is a summary of the extent and type of duplication across the dataset:

| | Fab Email | FabMetadata | Total |
|---|---|---|---|
| Exact Dupe Pairs | 67 | 53 | 120 |
| Near Dupe Pairs | 74 | 48 | 122 |
| Far Dupe Pairs | 12 | 69 | 81 |
| Content Dupe Pairs | 1 | 21 | 22 |
| Total Dupe Pairs | 154 (1.32%) | 191 (1.94%) | 345 (1.60%) |
| Total Non-Dupe Pairs | 11,474 | 9,679 | 21,153 |
| Total Pairs | 11,628 | 9,870 | 21,498 |
| Total Emails | 153 | 141 | 294 |

Table 3c: Summary of email pairs datasets

### Calculating Distance Values

As mentioned in Section 3.5.2 Levenshtein ratio and Jaro ratio were used to calculate the distance between the presumed seven most important text fields. A Python package 'python-Levenshtein'[24] was used to help calculate both ratios (code in Appendix 4). Three copies of these two distance datasets were produced, each with one of seconds, square root or $Log_{10}$ values of the 'Sent DateTime' field, as it is presumed that this field could be important. The transformation of the Sent DateTime field (i.e. square root and logarithm) will help investigate if scaling this integer value differently will produce better results. As a result there were six dataset versions to test with:

- Levenshtein Ratio – Sent DateTime (Integer)
- Levenshtein Ratio – Sent DateTime (Square Root)
- Levenshtein Ratio – Sent DateTime ($Log_{10}$)
- Jaro Ratio – Sent DateTime (Integer)
- Jaro Ratio – Sent DateTime (Square Root)
- Jaro Ratio – Sent DateTime ($Log_{10}$)

[24] https://pypi.python.org/pypi/python-Levenshtein/0.12.0

### Building Decision Trees

As previously mentioned the CART and C5.0 decision tree building algorithms were picked to carry out this initial exploration of the usefulness of decision tree classification for email duplication identification. Below is an overview of the decision tree experimentation:

- Decision trees built using R programming. The 'data.tree'[25] and 'C50'[26] packages were utilised in the scripts.
- Three decision trees were built for each combination algorithm (2) and distance dataset (6), resulting in 36 decision trees.
- The datasets were shuffled and split into training and test sets, using three seeds. The same three seeds were used for all the datasets and algorithms.
- 50/50 split between training and test datasets.
- Decision trees built on the training dataset.
- Misclassification calculated on the test dataset.
- Cross-validation was carried out.
- Trees pruned based on the cross-validation data.
- Misclassification of pruned tree calculated.

### Mapping Decision Tree Results Back To Original Email Dataset

The fundamental comparison of a proposed new approach with the Espion approach is a comparison of their accuracy when identifying whether the individual emails in the corpora are duplicates of not (i.e. Espion and Clearwell approaches only label or remove the 2nd email of a duplicate pair). The identification of the duplicative nature of an email is the ultimate output of any approach when applied to an eDiscovery context.

The output from the decision tree experimentation can be seen as a set of 'If-Else' rules to classify pairs of emails. The twelve decision trees combinations (i.e. combinations of CART & C5.0; Levenshtein & Jaro; Seconds and the square root and logarithm of seconds) were applied to the data by manually building the rules with a python script (Appendix 7) and then mapping the outcome back to the original corpora. In order to maintain consistency across these results, a previously unused shuffling seed was used for all these trees, to split them into

[25] https://cran.r-project.org/web/packages/data.tree/data.tree.pdf
[26] https://cran.r-project.org/web/packages/C50/C50.pdf

the training and testing datasets. The final output is FabEmail and FabMeta labelled for duplicates, based on the decision tree rules, this is then compared with the real classifications.

Combinations of variables used to build trees for mapping back to original set:

- CART-Levenshtein-Seconds
- CART-Levenshtein-Square Root
- CART-Levenshtein-$Log_{10}$
- CART-Jaro-Seconds
- CART-Jaro-Square Root
- CART-Jaro-$Log_{10}$
- C5.0-Levenshtein-Seconds
- C5.0-Levenshtein-Square Root
- C5.0-Levenshtein-$Log_{10}$
- C5.0-Jaro-Seconds
- C5.0-Jaro-Square Root
- C5.0-Jaro-$Log_{10}$

As the decision tree approaches analyse each pair, each email in a corpus has a duplicate label for each other email in the corpus. The labels were given an order and those at the top of the order 'trump' the others below:

- Duplicate order:
    - Exact Duplicate = Most important
    - Near Duplicate = 2nd important
    - Far Duplicate = 3rd important
    - Content Duplicate = 4th important
    - Non Duplicate = Least important
- For example, if an email is labelled Exact, Near and Non through its pairing with other emails in the corpus. It is labelled Exact, as that trumps the rest.

**Decision Tree Pruning**

The decision trees were also pruned in order to investigate how that would affect performance. As stated in Section 3.5.2, the CART implements post-pruning and C5.0 pre-pruning. The in-built cross validation and pruning features in the two R packages were used to prune.

# Chapter 4 - Results

This chapter lists the high level, important and most interesting results from the experimentation described in Section 3.6. Supplementary results can be found in the appendices.

## *4.1 Current Espion Approach*

Below are the combined results of separately applying the Espion approach to FabEmail and FabMeta.

| | Result of Espion Approach Analysis | | | | | |
|---|---|---|---|---|---|---|
| | Labelled as 'Dupe' | | | Labelled as 'Non-Dupe' | | |
| **Real Classification** | FabEmail | FabMeta | Total | FabEmail | FabMeta | Total |
| Exact Dupe | 51 | 36 | 87 (97%) | 0 | 3 | 3 (3%) |
| Near Dupe | 21 | 22 | 43 (62%) | 13 | 13 | 26 (38%) |
| Far Dupe | 1 | 0 | 1 (6%) | 4 | 13 | 17 (94%) |
| Content Dupe | 0 | 0 | 0 (0%) | 1 | 6 | 7 (100%) |
| Non Dupe | 0 | 0 | 0 (0%) | 13 | 7 | 20 (100%) |
| Non Dupe (First of Dupe Series) | 0 | 0 | 0 (0%) | 2 | 4 | 6 (100%) |
| Non Dupe (Sent Email) | 1 | 0 | 1 (1%) | 46 | 37 | 83 (99%) |

Table 4a: Results of Espion Approach Analysis

The main desired end-result from an approach is that it identifies 100% of true duplicates within a corpus, while at the same time not identifying any false positive duplicates. The Espion approach is unable to provide the added insight and functionality of distinguishing between exact, near, far and content duplicates, however for the purposes of this analysis, we will consider the identification of a positive 'Dupe' as a satisfactory outcome (i.e. true positive), even if it can't tell us what type of duplicate. The following performance analysis process and definitions were taken from (Shi, et al. 2012):

| | | Definitions | | Result of Espion Approach | |
|---|---|---|---|---|---|
| | | **Dupe** | **Non-Dupe** | **Dupe** | **Non-Dupe** |
| **Real Classification** | **Exact Dupe** | TP | FN | 87 | 3 |
| | **Near Dupe** | TP | FN | 43 | 26 |
| | **Far Dupe** | TP | FN | 1 | 17 |
| | **Content Dupe** | TP | FN | 0 | 7 |
| | **Non Dupe** | FP | TN | 0 | 20 |
| | **Non Dupe (First of Dupe Series)** | FP | TN | 0 | 6 |
| | **Non Dupe (Sent Email)** | FP | TN | 1 | 83 |

Table 4b: True/False Positive/Negative results from FabEmail & FabMetadata

*TP (True Positive): the number of instances correctly classified to that class.*

$$TP = 87 + 43 + 1 + 0 = 131$$

*TN (True Negative): the number of instances correctly rejected from that class.*

$$TN = 20 + 6 + 83 = 109$$

*FP (False Positive): the number of instances incorrectly rejected from that class.*

$$FP = 0 + 0 + 1 = 1$$

*FN (False Negative): the number of instances incorrectly classified to that class.*

$$FN = 3 + 26 + 17 + 7 = 53$$

*"Accordingly, Accuracy, precision, recall and F1 measure can be defined as follows:"*

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = 0.816$$

$$Precision = \frac{TP}{TP + FP} = 0.992$$

$$Recall = \frac{TP}{TP + FN} = 0.712$$

$$F1\ measure = 2\ x\ \frac{Precision\ x\ Recall}{Precision + Recall} = 0.865$$

*"The Accuracy of the classifier is the proportion of instances which are correctly classified. F1 is the harmonic mean of precision and recall, where precision is the rate of instances classified correctly among the result of classifier and recall is the rate of correct classified instances among them to be classified correctly."*

### *4.2 eDiscovery Software*

The following are the results of duplication exercise using the eDiscovery software Symantec Clearwell. Table 4c summarises the classification accuracy of Clearwell. Like the Espion approach, Clearwell only provides a binary result 'Duplicate' or 'Non-Duplicate'.

| | **Result of Symantec Clearwell Analysis** | | | | | |
|---|---|---|---|---|---|---|
| | Labelled as 'Dupe' | | | Labelled as 'Non-Dupe' | | |
| **Real Classification** | FabEmail | FabMeta | Total | FabEmail | FabMeta | Total |
| Exact Dupe | 51 | 37 | 88 (98%) | 0 | 2 | 2 (2%) |
| Near Dupe | 0 | 6 | 6 (8%) | 34 | 29 | 63 (92%) |
| Far Dupe | 0 | 1 | 1 (6%) | 5 | 12 | 17 (94%) |
| Content Dupe | 0 | 0 | 0 (0%) | 1 | 6 | 7 (100%) |
| Non Dupe | 0 | 0 | 0 (0%) | 13 | 7 | 20 (100%) |
| Non Dupe (First of Dupe Series) | 0 | 0 | 0 (0%) | 2 | 37 | 39 (100%) |
| Non Dupe (Sent Email) | 0 | 0 | 0 (0%) | 47 | 4 | 51 (100%) |

Table 4c: Results of eDiscovery software (Symantec Clearwell) analysis

There are some unexpected results in the FabMeta results e.g. two exact dupes tagged as non-dupe, six near dupes tagged as dupes and one far dupe tagged as dupe. It was believed that Clearwell wouldn't be able pick up on the near and far dupes at all. The reason for these surprises seems to be:

- Different MessageIDs led to an exact dupe being labelled as a non-dupe. It was unknown if Clearwell would compare this field.
- Clearwell doesn't seem to have analysed the attachment number and name fields for some emails, which led to the near dupe being labelled as duplicates expectantly.
- Clearwell was able to normalise some email addresses e.g. It was able to identify 'liz-gordon@outlook.com' and 'Liz <liz-gordon@outlook.com>'as duplicates, presumably by parsing the actual email address from within 'Liz < >'.

Like the Espion approach, for the purpose of analysis of a Near dupe, Far dupe or Content dupe email is labelled 'Dupe' it will be considered a true positive. Table 4d summarises the true positive/negative and false positive/negative results.

| | | **Definitions** | | **Result of Espion Approach** | |
|---|---|---|---|---|---|
| | | **Dupe** | **Non-Dupe** | **Dupe** | **Non-Dupe** |
| **Real Classification** | **Exact Dupe** | TP | FN | 88 | 2 |
| | **Near Dupe** | TP | FN | 6 | 63 |
| | **Far Dupe** | TP | FN | 1 | 17 |
| | **Content Dupe** | TP | FN | 0 | 7 |
| | **Non Dupe** | FP | TN | 0 | 20 |
| | **Non Dupe (First of Dupe Series)** | FP | TN | 0 | 39 |
| | **Non Dupe (Sent Email)** | FP | TN | 0 | 51 |

Table 4d: True/False Positive/Negative definitions & results from FabEmail & FabMetadata

The calculations below are the same as section 4.1 and calculate the performance of the approach.

$$\textit{TP (True Positive)} = 88 + 6 + 1 + 0 = 95$$

$$\textit{TN (True Negative)} = 20 + 39 + 51 = 110$$

$$\textit{FP (False Positive)} = 0 + 0 + 0 = 0$$

$$\textit{FN (False Negative)} = 2 + 63 + 17 + 7 = 89$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = 0.697$$

$$Precision = \frac{TP}{TP + FP} = 1.0$$

$$Recall = \frac{TP}{TP + FN} = 0.516$$

$$F1\ measure = 2\ x\ \frac{Precision\ x\ Recall}{Precision + Recall} = 0.681$$

### *4.3 Decision Tree Approaches Experimentation Results*

The following sub section contains distilled and summarised results from experimentations with decision tree (DT) algorithms. In total, 36 trees were constructed with a combination of three variables, DT techniques (CART and C5.0), string dissimilarity measures (Levenshtein and Jaro ratios) and Sent DateTime difference representations (seconds, square-root and $\log_{10}$). Those 36 trees were also 'pruned' to create 36 slightly different trees. A high level summary of the results across the 72 decision trees is provided in Table 4e below. Similar summaries for pruned and unpruned trees can be found in Appendix 1.

| Pruned & Unpruned | Samples | Misclass. (Avg.) | Nodes (Avg.) |
|---|---|---|---|
| All Trees | 72 | 0.39% | 14.9 |
| All CART Algorithm | 36 | 0.52% | 8.7 |
| All C5.0 Algorithm | 36 | 0.25% | 21.1 |
| All Levenshtein Ratio | 36 | 0.38% | 15.0 |
| All Jaro Ratio | 36 | 0.39% | 14.8 |
| All Time Difference (Seconds) | 24 | 0.38% | 14.8 |
| All Square Root of Time Difference | 24 | 0.37% | 15.2 |
| All Log10 of Time Difference | 24 | 0.41% | 14.7 |
| Unpruned CART Trees | 18 | 0.51% | 9.2 |
| Unpruned C5.0 Trees | 18 | 0.25% | 21.7 |
| Pruned CART Trees | 18 | 0.54% | 8.2 |
| Pruned C5.0 Trees | 18 | 0.25% | 20.4 |
| CART & Levenshtein | 18 | 0.51% | 8.8 |
| C5.0 & Levenshtein | 18 | 0.25% | 21.2 |
| CART & Jaro | 18 | 0.54% | 8.6 |
| C5.0 & Jaro | 18 | 0.25% | 20.9 |
| CART & Seconds | 12 | 0.51% | 8.8 |
| CART & Square Root | 12 | 0.51% | 8.8 |
| CART & Log10 | 12 | 0.54% | 8.5 |
| C5.0 & Seconds | 12 | 0.25% | 20.8 |
| C5.0 & Square Root | 12 | 0.23% | 21.5 |
| C5.0 & Log10 | 12 | 0.27% | 20.9 |
| Levenshtein & Seconds | 12 | 0.40% | 14.1 |
| Levenshtein & Square Root | 12 | 0.33% | 16.5 |
| Levenshtein & Log10 | 12 | 0.41% | 14.5 |
| Jaro & Seconds | 12 | 0.36% | 15.6 |
| Jaro & Square Root | 12 | 0.42% | 13.8 |
| Jaro & Log10 | 12 | 0.40% | 14.9 |
| CART & Levenshtein & Seconds | 6 | 0.52% | 8.5 |
| CART & Levenshtein & Square Root | 6 | 0.48% | 9.5 |
| CART & Levenshtein & Log10 | 6 | 0.54% | 8.5 |
| CART & Jaro & Seconds | 6 | 0.51% | 9.2 |
| CART & Jaro & Square Root | 6 | 0.55% | 8.2 |
| CART & Jaro & Log10 | 6 | 0.55% | 8.5 |
| C5.0 & Levenshtein & Seconds | 6 | 0.29% | 19.7 |
| C5.0 & Levenshtein & Square Root | 6 | 0.18% | 23.5 |
| C5.0 & Levenshtein & Log10 | 6 | 0.29% | 20.5 |
| C5.0 & Jaro & Seconds | 6 | 0.22% | 22.0 |
| C5.0 & Jaro & Square Root | 6 | 0.29% | 19.5 |
| C5.0 & Jaro & Log10 | 6 | 0.25% | 21.3 |

Table 4e: High level summary of decision tree experimentation results

As can be seen in Table 4e (above) a combination of C5.0, Levenshtein Ratio and the square root of the sent time difference performed the best with a misclassification rate of 0.18%.

While this is from a small sample (two versions of three trees, pruned and unpruned), C5.0, Levenshtein Ratio and 'Square Root' marginally performed best (against their respective competing metrics) across all testing, as will be discussed in the following sections.

Figure 4a is an example of a tree built using the CART algorithm, Levenshtein ratio and seconds as the unit of time difference.

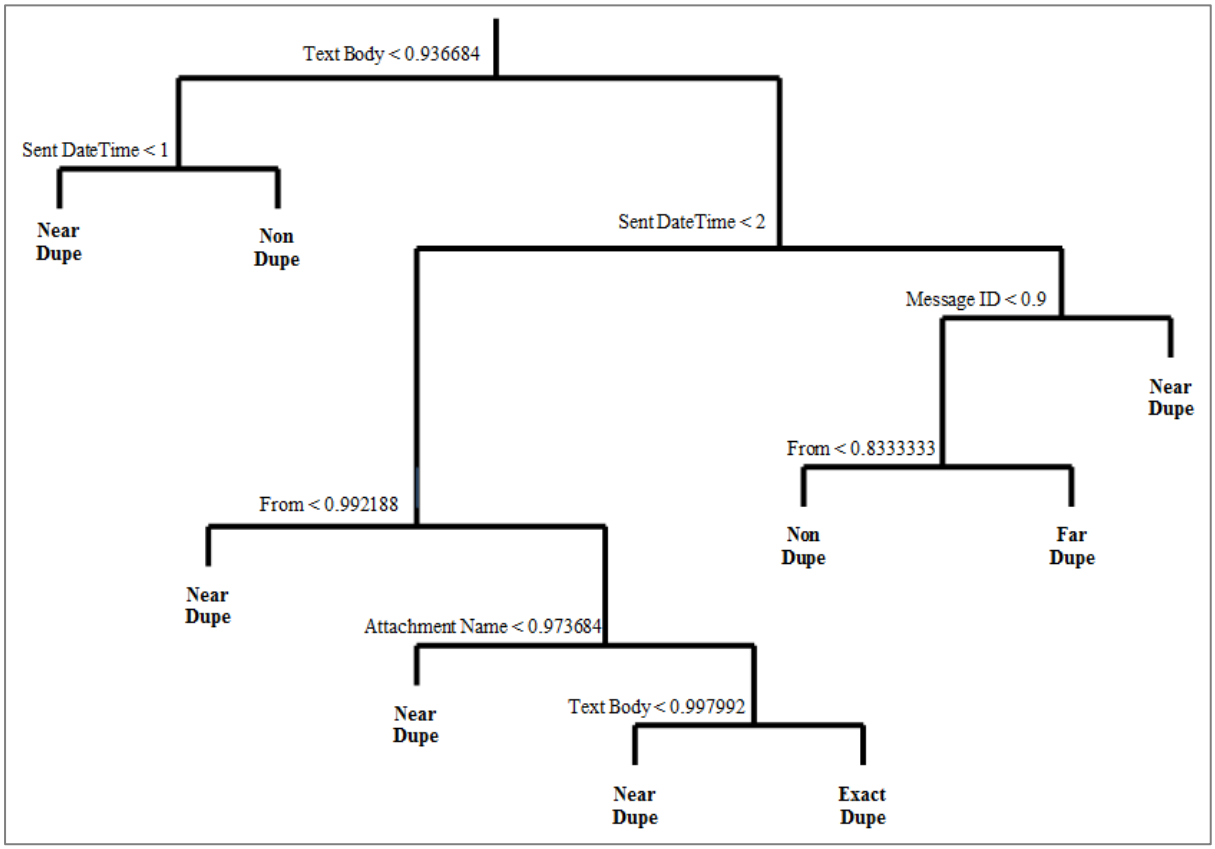


Figure 4a: Example of a CART decision tree, using Levenshtein ratio and Seconds

There is a significant correlation between the misclassification rate on the test dataset and the size of the decision trees, as Figure 4b illustrates. This trend applies across all combinations of variables used to build the trees.

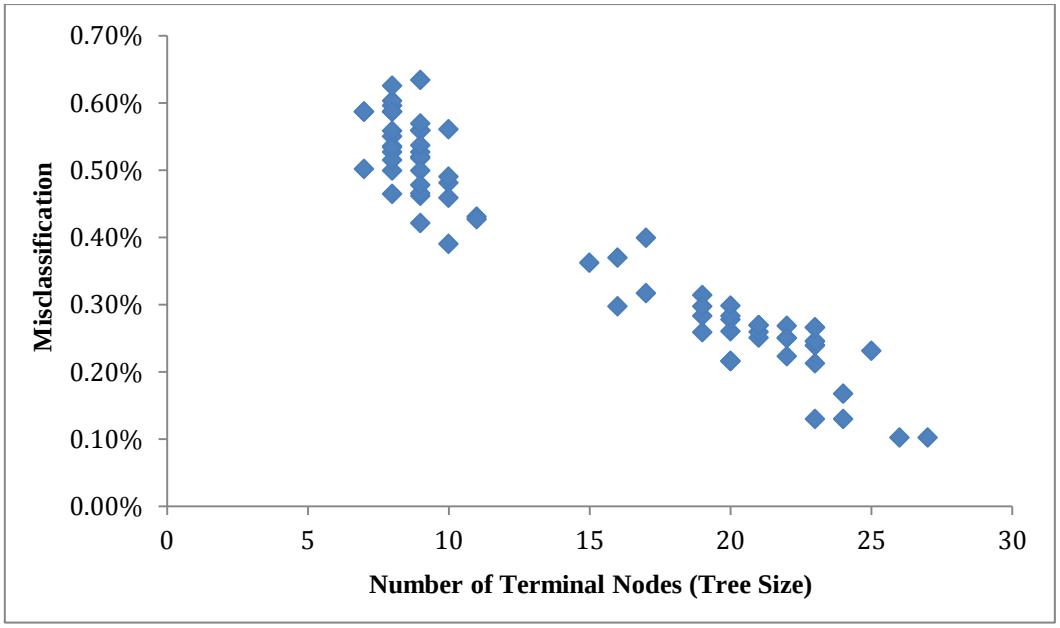


Figure 4b: Relationship between the misclassification rate and the size of the tree

The misclassification rates in Table 4e (above), of less than 1% seem to indicate good classification, however Non-Dupes make up 98.4% of the two corpora. The vast majority of Non-Dupe pairs are easy for humans and software to classify as non-duplicates.. The high proportion of Non-Dupes is skewing the overall misclassification and it does not give a real insight into the classification of the more 'difficult' to classify Near, Far and Content duplicate pairs. Figure 4c is a better indication of the variance of misclassification across the duplication types and with tree size.

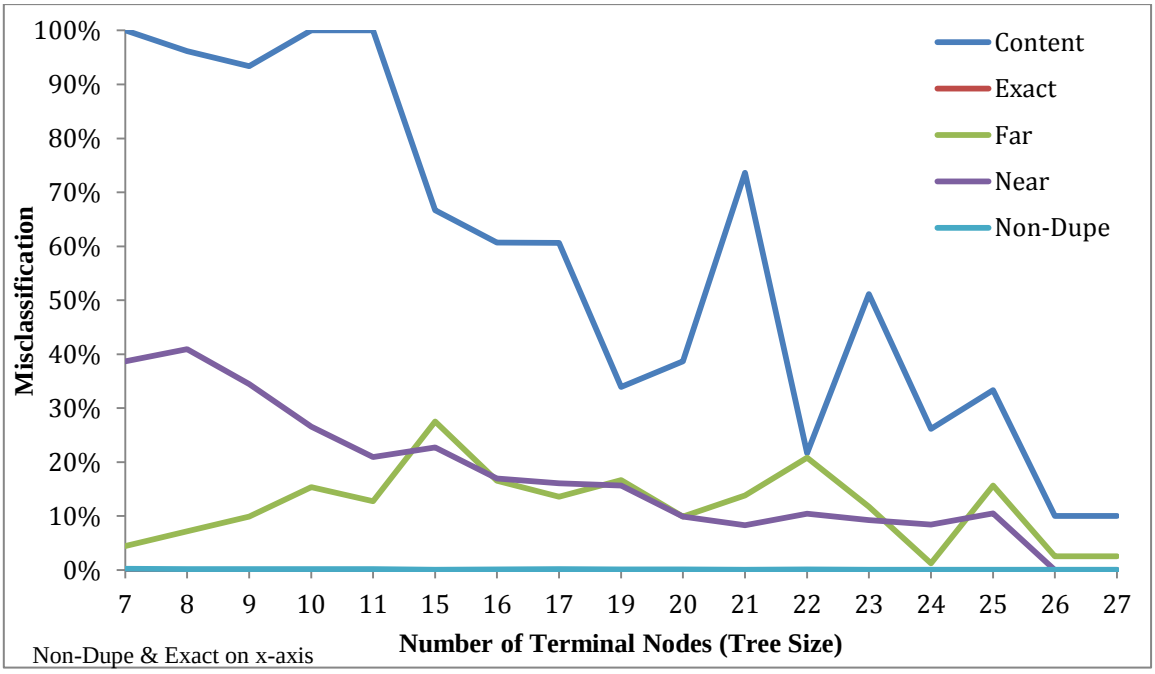


Figure 4c: Misclassification by duplication type and number of nodes (pruned & unpruned trees)

### *4.3.1 Decision Tree Building Techniques*

The most consistent difference between the results of the CART and C5.0 approaches was the number of leaves created by the trees, as can be seen in Figure 4d, which also illustrates the small reduction in leaves post pruning in both approaches.

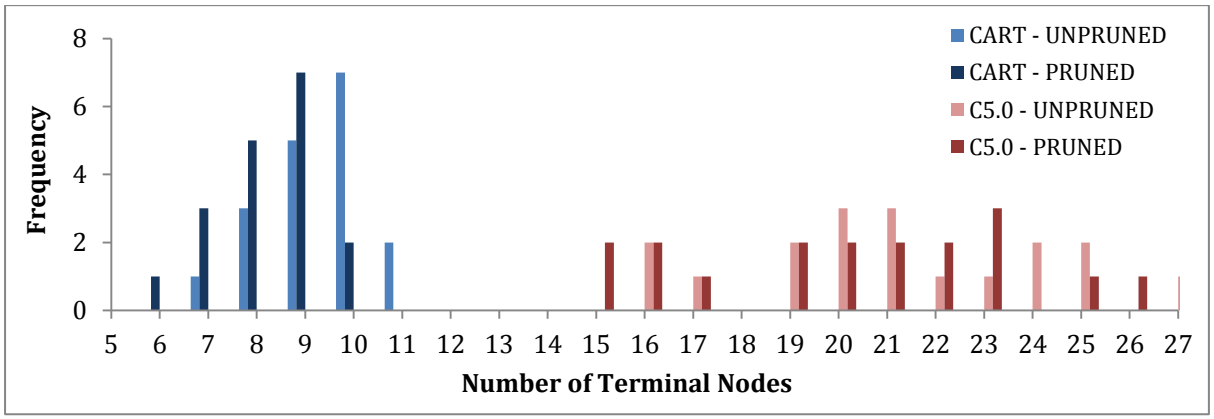


Figure 4d: Distribution of the number of leaves in the unpruned and pruned trees

As well as a consistent difference between the size of CART and C5.0 decision trees, there is a consistent and relatively significant difference in their misclassification results as shown in Figure 4e.

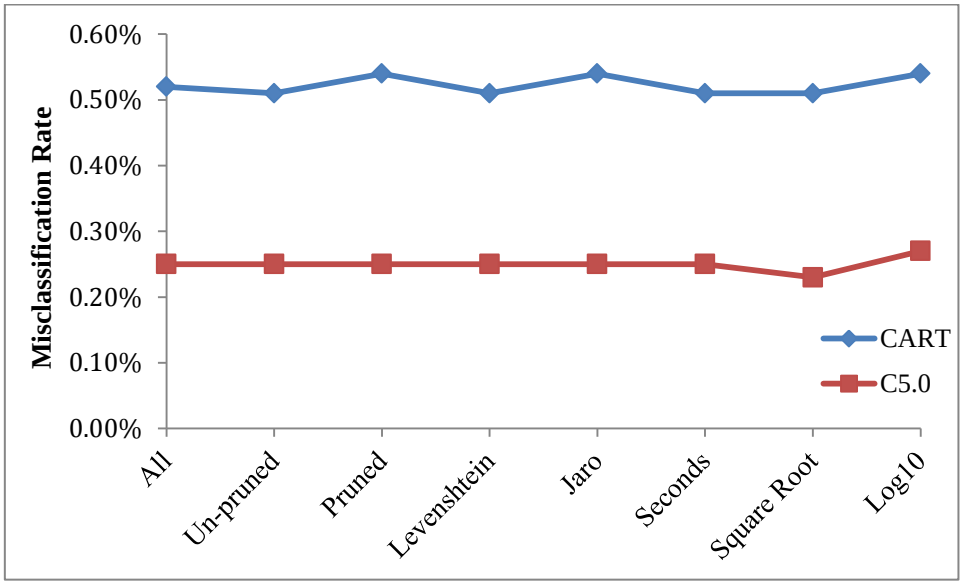


Figure 4e: Misclassification rates across all the variables

Figure 4f below, illustrates CART's poor performance classifying Content and Near duplicate emails, that accounts for its comparatively bad overall misclassification rate. In fact, many of

CART trees do not have any leaves that classify Content duplicates, so they have no chance of classifying such emails.

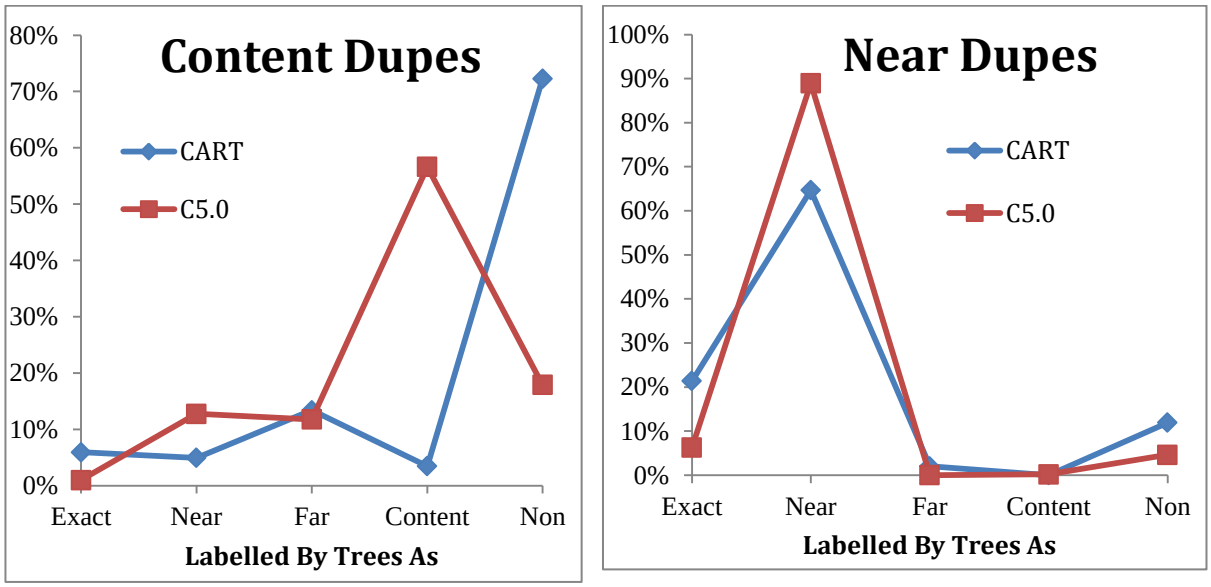


Figure 4f: CART classification of 'Content' and 'Near' duplicates in comparison to C5.0

The results of classifying the individual emails using decision trees (Tables 4k and 4l), highlight the same accuracy difference between C5.0's bigger trees and CART's smaller trees. CART is specifically unable to identify Content duplicates at all and is significantly worse at identifying Near duplicates.

| **C5.0 – FabEmail & FabMeta** | | **Decision Tree Classification** | | | | |
|---|---|---|---|---|---|---|
| | | **Exact** | **Near** | **Far** | **Content** | **Non-Dupe** |
| **Real Classification** | **Exact Dupe** | **97%** | 3% | 0% | 0% | 0% |
| | **Near Dupe** | 3% | **95%** | 0% | 0% | 2% |
| | **Far Dupe** | 0% | 0% | **94%** | 1% | 5% |
| | **Content Dupe** | 0% | 28% | 10% | **52%** | 10% |
| | **Non Dupe** | 0% | 2% | 2% | 3% | **93%** |
| | **Non Dupe (First of Dupe Series)** | 0% | 0% | 0% | 0% | **100%** |
| | **Non Dupe (Sent Email)** | 0% | 2% | 1% | 0% | **97%** |

Table 4k: Performance summary of C5.0 identifying duplicate types in FabEmail and FabMeta

| **CART – FabEmail & FabMeta** | | **Decision Tree Classification** | | | | |
|---|---|---|---|---|---|---|
| | | **Exact** | **Near** | **Far** | **Content** | **Non-Dupe** |
| **Real Classification** | **Exact Dupe** | **97%** | 3% | 0% | 0% | 0% |
| | **Near Dupe** | 22% | **71%** | 2% | 0% | 5% |
| | **Far Dupe** | 1% | 2% | **93%** | 3% | 1% |
| | **Content Dupe** | 12% | 26% | 17% | **10%** | 35% |
| | **Non Dupe** | 0% | 1% | 8% | 3% | **88%** |
| | **Non Dupe (First of Dupe Series)** | 2% | 2% | 8% | 2% | **86%** |
| | **Non Dupe (Sent Email)** | 0% | 4% | 3% | 1% | **92%** |

Table 4l: Performance summary of CART identifying duplicate types in FabEmail and FabMeta

A significant concern from the above results is the number of non duplicate emails classified as duplicates (enclosed by the bold box) in the above tables. Such misclassifcations in a real eDiscovery application may be deleted without being examined and thus vital evidence could be missed. There are 4.0% and 9.3% false positives in the C5.0 and CART results respectively, which is significant and another example of C5.0's better accuracy.

The above results are stricter than those of the Espion and Clearwell results in Sections 4.1 and 4.2, as the results in Tables 4k and 4l consider a classification to be wrong if for example, an Exact dupe is classified as a Near dupe. The results in Sections 4.1 and 4.2 accept a classification correct if for example an Exact dupe is classified as a Dupe (which can be Exact and some Near duplicates). Table 4m provides a summary of the strict & non-strict misclassifications and false positives of C5.0 and CART techniques.

| | **Strict Misclassification**[27] | **Non-Strict Misclassification**[28] | **False Positives** |
|---|---|---|---|
| CART | 13.8% | 5.6% | 10.3% |
| C5.0 | 4.8% | 2.3% | 4.0% |

Table 4m: Strict & Non-Strict misclassification of CART and C5.0 on individual emails

[27] Strict Misclassification: {Exact≠Exact},{Near≠Near},{Far≠Far},{Content≠Content},{NonDupe ≠NonDupe}
[28] Non-Strict Misclassification: {Exact, Near, Far, Content} = {NonDupe}

### *4.3.2 Decision Tree Pruning*

As stated in the previous section, pruning of both CART and C5.0 trees in every case provided either a very slightly higher or the same misclassification on the test dataset, even for small prunes as illustrated by Figure 4d (above).The figure below illustrates the misclassification of pruned and unpruned trees.

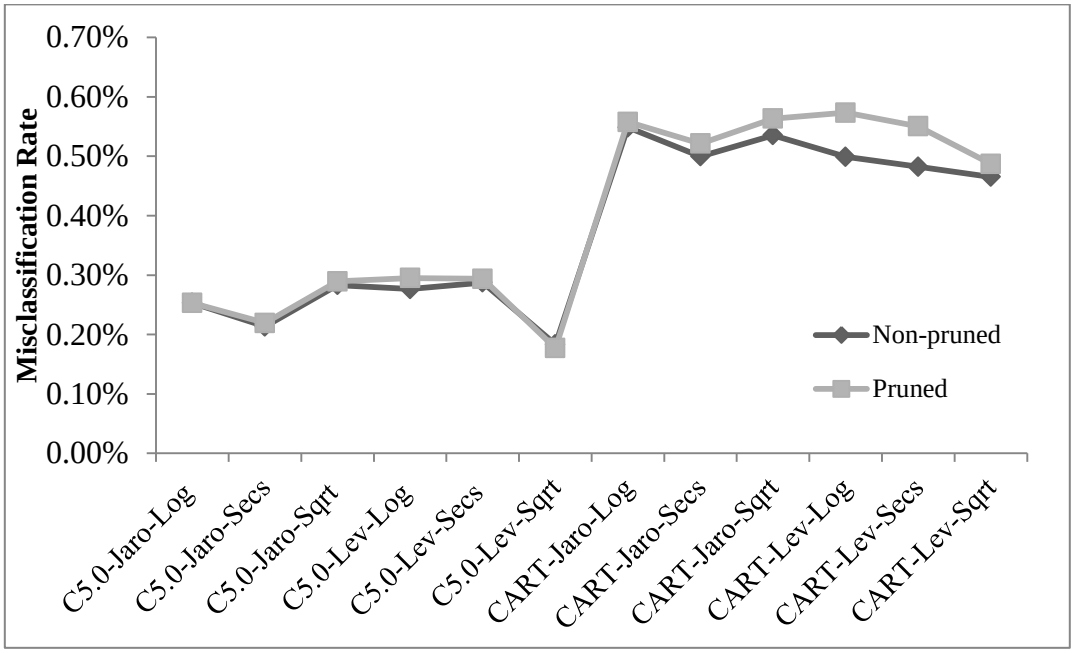


Figure 4g: Similarity between misclassification of pruned and unpruned trees

### *4.3.3 String Dissimilarity Metrics*

Table 4f summarises the very slightly smaller misclassification across 36 samples with the Levenshtein ratio used to measure the distance between string fields. The proportional difference between the misclassifications is similar to the difference in average number of nodes, with the Levenshtein trees being slightly bigger.

| | Samples | Misclassification Average | Nodes (Avg.) |
|---|---|---|---|
| All Trees | 72 | 0.39% | 14.9 |
| All CART Algorithm | 36 | 0.52% | 8.7 |
| All C5.0 Algorithm | 36 | 0.25% | 21.1 |
| All Levenshtein Ratio | 36 | 0.38% | 15.0 |
| All Jaro Ratio | 36 | 0.39% | 14.8 |

Table 4f: High level comparison of Levenshtein and Jaro Ratio based trees

Results from mapping back to the original corpora to identify duplicates, show Jaro performing slightly better than Levenshtein as shown in Table 4p below. Jaro provides slightly less false positives and is better on strict misclassification but overall the difference between is negligible.

| | Strict Misclassification[29] | Non-Strict Misclassification[30] | False Positives |
|---|---|---|---|
| Levenshtein | 9.5% | 4.0% | 7.3% |
| Jaro | 9.1% | 4.0% | 7.0% |

Table 4p: Strict & Non-Strict classification by Levenshtein and Jaro Ratios on individual emails

### *4.3.4 'Sent DateTime' Difference Representations*

As can be seen in Table 4h the difference in performance of the three different representations of the Sent DateTime distance only slight. However there does seem to be a distinct difference between the $Log_{10}$ performing worse than the other two (note: $Log_{10}(0)$ was assumed equal to 0 and $Log_{10}(1)$ as 1). Again, the correlation between a bigger tree and better performance is evident in these results.

| | Samples | Misclassification | Nodes # |
|---|---|---|---|
| Seconds | 24 | 0.38% | 14.8 |
| Square Root of Seconds | 24 | 0.37% | 15.2 |
| $Log_{10}$ of Seconds | 24 | 0.41% | 14.7 |

Table 4h: Performance and size of trees based on the different representations of the Sent DateTime distance

The performance of these three representations in classifying Near, Far and Content dis vary, as Table 4i shows.

| Misclassification (%) | Exact | Near | Far | Content | Non |
|---|---|---|---|---|---|
| Seconds | 0% | 21.80% | 11.81% | 72.10% | 0.14% |
| Square Root of Seconds | 0% | 23.96% | 10.40% | 65.25% | 0.13% |
| $Log_{10}$ of Seconds | 0% | 23.60% | 12.42% | 72.44% | 0.15% |

Table 4i: Classification accuracy of time distance representations across the five duplicate types

[29] Strict Misclassification: {Exact≠Exact},{Near≠Near},{Fart≠Far},{Content≠Content},{NonDupe ≠NonDupe}
[30] Non-Strict Misclassification: {Exact, Near, Far, Content} = {NonDupe}

The trends above are maintained when identifying the duplicate nature of individual emails in the original corpora as Table 4q indicates below.

| | Strict Misclassification[31] | Non-Strict Misclassification[32] | False Positives |
|---|---|---|---|
| Seconds | 9.9% | 4.1% | 6.8% |
| Square Root of Seconds | 8.9% | 3.3% | 6.5% |
| $Log_{10}$ of Seconds | 9.0% | 4.5% | 8.3% |

Table 4q: Strict & Non-Strict classification across time difference transformations

Table 4q show how C5.0 used the Sent DateTime field marginally less when the logarithm of that field was used.

| | C5.0 Rank of Sent DateTime | CART Usage of Sent DateTime |
|---|---|---|
| Seconds | 100% | 2 |
| Square Root of Seconds | 100% | 2 |
| $Log_{10}$ of Seconds | 99.9% | 2 |

Table 4q: Usage of Sent DateTime field across variations of the representations of that field

### *4.3.5 Attribute Usage in Tree Building*

The outputs from the C5.0 and CART packages summarising attribute usage is different, as Table 4j. C5.0 provides a percentage usage and CART ranks attribute usage (1 being most important). A composite rank of attribute usage was calculated across CART and C5.0 and can be seen in the last line of Table 4j.

| Averages | From | To | CC | Subject | Text | Att# | AttNames | MID | Time |
|---|---|---|---|---|---|---|---|---|---|
| CART (Average Rank) | 3.8 | Never Used | Never Used | 5.0 | 1.0 | 4.5 | 3.2 | 5.0 | 2.0 |
| C5.0 (% Usage) | 1.5% | 0.8% | 0.6% | 1.2% | 100% | 0.3% | 1.4% | 0.9% | 100% |
| | | | | | | | | | |
| CART (Overall Rank) | **4** | **-** | **-** | **6.5** | **1** | **5** | **3** | **6.5** | **2** |
| C5.0 (Overall rank) | **3** | **7** | **8** | **5** | **1.5** | **9** | **4** | **6** | **1.5** |
| Composite Rank | **3.5** | **8** | **9** | **5** | **1** | **7** | **3.5** | **6** | **2** |

Table 4j: Summary of attribute usage across C5.0 and CART trees

[31] Strict Misclassification: {Exact≠Exact},{Near≠Near},{Fart≠Far},{Content≠Content},{NonDupe ≠NonDupe}
[32] Non-Strict Misclassification: {Exact, Near, Far, Content} = {NonDupe}

The Text Body and Sent DateTime fields consistently ranked as the first and second most important attribute, respectively, in the CART trees. This coupled with the average usage in the C5.0 trees, indicates that the Text Body and Sent DateTime fields are by far the most important fields for duplication classification. The MessageID, From and Attachment Name being a group of the next important fields, depending on the algorithm to a degree. The To and CC fields are the most underused across the board.

### *4.4 Comparison of Results*

Table 4n gives an overview comparison of the accuracy of the Espion, Clearwell and proposed Decision Tree approaches. All three deal with Exact dupes very well but there are significant differences on the other duplicate types. Espion outperforms Clearwell when it comes to Near duplicates, as expected. The proposed DT based approach classifies Near, Far and Content dupes more accurately. The average accuracy of DTs on non-duplicates is high (as discussed in previous section) but the decision tree with best misclassification rate 'C5.0-Levenshtien-SquareRoot' deals relatively well with the non-duplicates.

| | **Experimental Classifications** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Labelled as 'Dupe'[33] | | | | Labelled as 'Non-Dupe' | | | |
| **Real Classification** | Espion | Clearwell | Best Tree[34] | Avg. Tree | Espion | Clearwell | Best Tree | Avg. Tree |
| Exact Dupe | 97% | 98% | 100% | 100% | 3% | 2% | 0% | 0% |
| Near Dupe | 61% | 8% | 100% | 96% | 38% | 92% | 0% | 4% |
| Far Dupe | 6% | 6% | 89% | 97% | 94% | 94% | 11% | 3% |
| Content Dupe | 0% | 0% | 100% | 77% | 100% | 100% | 0% | 23% |
| Non Dupe | 0% | 0% | 0% | 9% | 100% | 100% | 100% | 91% |
| Non Dupe (First of Dupe Series) | 0% | 0% | 0% | 8% | 100% | 100% | 100% | 92% |
| Non Dupe (Sent Email) | 1% | 0% | 2% | 6% | 99% | 100% | 98% | 94% |

Table 4n: Classifications of Espion, Clearwell and decision tree experimentation

Table 4o below and following calculations, calculate the different accuracy measure of the Dt approach so as to compare with two baseline approaches.

[33] Dupe = {Exact, Near, Far, Content Dupe}
[34] Best Tree = Decision tree with smallest misclassification

| | | Definitions | | Best Tree | | Avg. Tree | |
|---|---|---|---|---|---|---|---|
| | | **Dupe** | **Non-Dupe** | **Dupe** | **Non-Dupe** | **Dupe** | **Non-Dupe** |
| **Real Classification** | **Exact Dupe** | TP | FN | 90 | 0 | 90 | 0 |
| | **Near Dupe** | TP | FN | 69 | 0 | 66.3 | 2.4 |
| | **Far Dupe** | TP | FN | 16 | 2 | 17.8 | 0.5 |
| | **Content Dupe** | TP | FN | 7 | 0 | 5.4 | 1.6 |
| | **Non Dupe** | FP | TN | 0 | 20 | 1.8 | 18.3 |
| | **Non Dupe (First of Dupe Series)** | FP | TN | 0 | 6 | 0.5 | 5.6 |
| | **Non Dupe (Sent Email)** | FP | TN | 2 | 82 | 4.9 | 79.1 |

Table 4o: True/False Positive/Negative definitions & results for DT approach

$$TP(Best) = 90 + 69 + 16 + 7 = 182$$
$$TP(Avg.) = 90 + 66.3 + 17.8 + 5.4 = 179.5$$

$$TN(Best) = 20 + 6 + 82 = 108$$
$$TN(Avg.) = 18.3 + 5.6 + 79.1 = 103.0$$

$$FP\ (Best) = 0 + 0 + 2 = 2$$
$$FP\ (Avg.) = 1.7 + 0.5 + 4.5 = 6.7$$

$$FN\ (Best) = 0 + 0 + 2 + 0 = 2$$
$$FN\ (Avg.) = 0 + 2.4 + 0.5 + 1.6 = 4.5$$

$$Accuracy(Best) = \frac{TP + TN}{TP + TN + FP + FN} = 0.986$$
$$Accuracy(Avg.) = \frac{TP + TN}{TP + TN + FP + FN} = 0.961$$

$$Precision\ (Best) = \frac{TP}{TP + FP} = 0.989$$
$$Precision\ (Avg.) = \frac{TP}{TP + FP} = 0.964$$

$$Recall\ (Best) = \frac{TP}{TP + FN} = 0.989$$

$$Recall\ (Avg.) = \frac{TP}{TP + FN} = 0.976$$

$$F1\ measure\ (Best) = \ 2\ x\ \frac{Precision\ x\ Recall}{Precision + Recall} = 0.989$$

$$F1\ measure\ (Avg.) = \ 2\ x\ \frac{Precision\ x\ Recall}{Precision + Recall} = 0.970$$

Figure4i illustrates the performance of the three approaches against each other. The best and average DT accuracy, recall and F1Measure scores are better than those of the two baseline approaches. However, the precision, a measure of proportion of false positives, of the DT approaches is very slightly worse than the baseline approaches, which could be expensive in an eDiscovery context. A non-duplicate being classified as a duplicate could potentially lead to lost evidence, where a false negative may only lead to lost time of reviewing two versions of the same email.

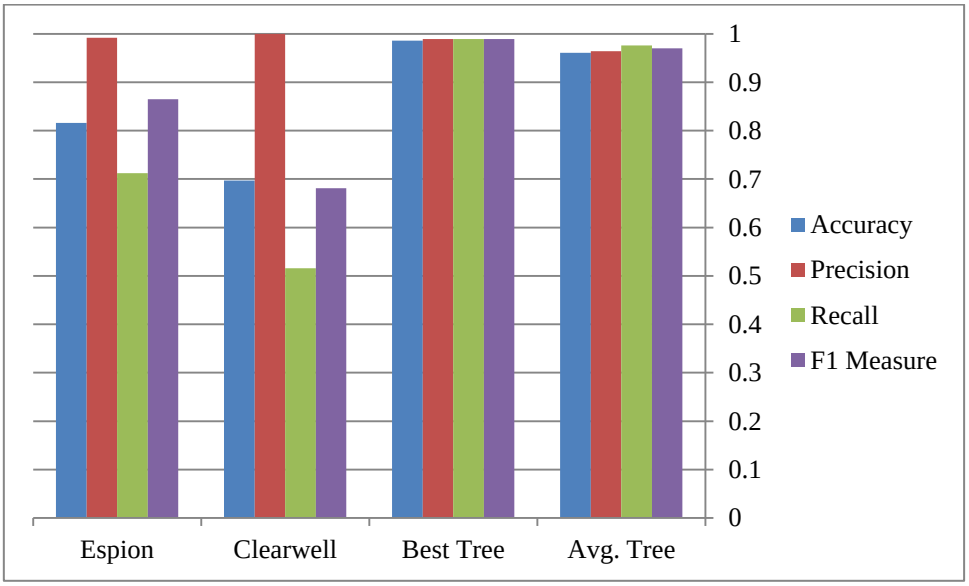


Figure 4i: Accuracy, precision, recall and F1 Measure of Espion, Clearwell and Tree approaches

# Chapter 5 - Analysis & Discussion

## *5.1 General*

The results of the experimentation with the datasets discussed in Chapter 4 indicate that there is potentially benefit in applying decision tree classification to identifying email duplicate types. As Figure4i illustrates, the DT approach has better accuracy, recall and F1 measure than the Espion and Clearwell approaches. Overall a combination of (and separately) the C5.0 DT building technique, the Levenshtein Ratio for string dissimilarity and the square root of the Sent DateTime distance in seconds gave the best results.

There are still question marks over applying this process to email deduplication, which the constraints of this project will not allow further exploration. Those question marks include false positives (labelling non-dupes as dupes) and the dataset.

The 'precision' (i.e. measure of false positives) of the DT approach is slightly higher than that of the Espion and Clearwell approaches (which are almost zero). In the context of eDiscovery, false positives are potentially very expensive. Further work should be done to find a DT approach that reduces the number of false positives to nearly zero. The results do indicate that C5.0 is significantly better at avoiding false positives than CART.

The corpora that this experimentation was based on are not ideal, as they are not a real world samples. In retrospect, the corpora that were created could have been improved too, as the low number of Content duplicates and the low density of duplicate pairs in the datasets probably hindered the performance of the DTs.

While the non-strict misclassification rate used to compare the DT approach with the Espion and Clearwell approaches was satisfactory, the stricter misclassification rate was higher, meaning that the DT approach struggled to accurately classify duplicates in their exact classification e.g. Exact as Exact and not Near, Far or Content. As mentioned above, a bigger sample of Content and Far dupes in particular, would probably have been ideal to bring this rate down.

### 5.2 Decision Building Algorithms

According to the results, the choice of decision tree building algorithm was the biggest influence on the classification accuracy. It would seem that the size of the trees impacted the accuracy and thus this led the C5.0 trees to perform better on all fronts, as like (Zhong 2007) and (Barros, de Carvalho and Freitas 2015) found, C5.0 consistently built trees bigger than the CART algorithm. Based on this data, the more leaves led to more accurate results on the test data. C5.0 specifically performed better on the Content and Near duplicates which was the main factor on its overall better accuracy.

### 5.3 Distance Metrics

The results indicate little difference in the classification results by using the Levenshtein or Jaro Ratio. Taking the square root of the number of seconds between the Sent DateTime fields, provide the best results but only marginally over unaltered seconds, with logarithm performing worst.

### 5.4 Email Attributes Usage

One of the objectives of the experimentation was to examine what attributes the DTs used most, to get an idea of what fields were of most use to email deduplication. The results corroborate most people's belief that the Text Body is the most useful field, as is evident from that fields used in all current deduplication processes. The Sent DateTime's usage was very close behind in second in the results. Again this is another widely used field and key to Espion's approach to finding near duplicates.

The MessageID as expected featured heavily but its occasional absence and changes between versions mean that it can't be used more. The From and Attachment Name fields also feature regularly. While the From and CC fields were used least frequently of the nine, it would seem because CART did not use them at all it struggled to classify Near and in particular content duplicates.

# Chapter 6 - Conclusions & Further Work

## *6.1 Summary*

This piece of research has explored a relatively unexplored academic territory of email deduplication beyond just text body or digital fingerprinting analysis. It has proposed a novel definition of duplicates beyond the standard binary definition of duplicate and non-duplicate, to accommodate different shades of duplication. It has also proposed and investigated a new approach to duplication identification using decision tree classification, which has shown promise of being a more informative and accurate successor to the more time consuming and basics approaches currently implemented. Question marks remain over the application of decision tree classification to the area which should be investigated with future research.

A by-product of the research is an improved understanding of the usefulness of various distance metrics and email attributes to the deduplication process. Also, two fully labelled and duplicate heavy datasets have been provided for future researches in the area.

## *6.2 Contributions*

- A novel decision tree classification approach to email deduplication.
- A corpus of labelled duplicate emails, which also includes a wide range of duplicate scenarios that can be used for future research in the area.
- A novel, broader and more informative set of definitions of email duplication types.
- A better understanding of the accuracy of the Espion approach relative to standard eDiscovery software such as Clearwell.
- Extended understanding of the attributes and distance measures that benefit email deduplication.
- Espion now has basic software that automates the sorting, comparing and labelling steps of their process.
- Espion now has the foundations of software that uses decision tree classification to provide a more accurate and informative deduplication approach and that does not require spending time on normalising the data.
- A comprehensive compilation of email near duplication scenarios, to an extent which can’t be found elsewhere.

### *6.3 Limitations & Future Work*

- Undertake end user validation regarding the non-binary proposed duplicate definition, to see if it meets real life requirements
- Further research to improve the strict and non-strict misclassification of Near, Content and Far duplicates (to a lesser extent)
- Testing the proposed approach on other datasets, if possible, a suitable real world example
- Explore DT approach with Exact-Near-Non and/or Dupe-NonDupe definitions
- Further research to reduce the number of false positives
- Instead of the multivariate definition of duplicates used in this project, investigate how to improve on Espion's approach and maintain the binary duplicative definition
- Investigate the benefits of other string and time distance measures
- Investigate the performance of other decision tree algorithms
- Investigate the benefits of similarity scoring to email deduplication
- Investigate if there are more intuitive If-Else based algorithms than the ones created by the decision tree building algorithms
- Looking at term-based similarity measures for email deduplication and/or comparing then against character/lexical approaches
- Experiment with a dataset that is more duplicate dense, in particular Content and Far duplicates

# Appendix 1 – Pruned & Unpruned Decision Tree Results

| Unpruned | Samples | Misclass. (Avg.) | Nodes (Avg.) |
|---|---|---|---|
| All Trees | 36 | 0.39% | 15.2 |
| All CART Algorithm | 18 | 0.51% | 9.3 |
| All C5.0 Algorithm | 18 | 0.28% | 21.1 |
| All Levenshtein Ratio | 18 | 0.36% | 15.7 |
| All Jaro Ratio | 18 | 0.43% | 14.8 |
| All Time Difference (Seconds) | 12 | 0.37% | 15.4 |
| All Square Root of Time Difference | 12 | 0.37% | 15.8 |
| All Log10 of Time Difference | 12 | 0.45% | 14.5 |
| Unpruned CART Trees | 18 | 0.51% | 9.3 |
| Unpruned C5.0 Trees | 18 | 0.28% | 21.1 |
| CART & Levenshtein | 9 | 0.49% | 9.2 |
| C5.0 & Levenshtein | 9 | 0.23% | 22.1 |
| CART & Jaro | 9 | 0.52% | 9.4 |
| C5.0 & Jaro | 9 | 0.34% | 20.1 |
| CART & Seconds | 6 | 0.49% | 9.3 |
| CART & Square Root | 6 | 0.50% | 9.3 |
| CART & Log10 | 6 | 0.53% | 9.3 |
| C5.0 & Seconds | 6 | 0.25% | 21.5 |
| C5.0 & Square Root | 6 | 0.23% | 22.2 |
| C5.0 & Log10 | 6 | 0.37% | 19.7 |
| Levenshtein & Seconds | 6 | 0.38% | 14.8 |
| Levenshtein & Square Root | 6 | 0.32% | 17.0 |
| Levenshtein & Log10 | 6 | 0.37% | 15.2 |
| Jaro & Seconds | 6 | 0.36% | 16.0 |
| Jaro & Square Root | 6 | 0.41% | 14.5 |
| Jaro & Log10 | 6 | 0.52% | 13.8 |
| CART & Levenshtein & Seconds | 3 | 0.48% | 9.0 |
| CART & Levenshtein & Square Root | 3 | 0.47% | 10.0 |
| CART & Levenshtein & Log10 | 3 | 0.53% | 8.7 |
| CART & Jaro & Seconds | 3 | 0.50% | 9.7 |
| CART & Jaro & Square Root | 3 | 0.54% | 8.7 |
| CART & Jaro & Log10 | 3 | 0.53% | 10.0 |
| C5.0 & Levenshtein & Seconds | 3 | 0.29% | 20.7 |
| C5.0 & Levenshtein & Square Root | 3 | 0.18% | 24.0 |
| C5.0 & Levenshtein & Log10 | 3 | 0.22% | 21.7 |
| C5.0 & Jaro & Seconds | 3 | 0.21% | 22.3 |
| C5.0 & Jaro & Square Root | 3 | 0.28% | 20.3 |
| C5.0 & Jaro & Log10 | 3 | 0.51% | 17.7 |

TableA1: High level summary of un-pruned decision tree experimentation results

| Pruned | Samples | Misclass. (Avg.) | Nodes (Avg.) |
|---|---|---|---|
| All Trees | 36 | 0.41% | 14.3 |
| All CART Algorithm | 18 | 0.54% | 8.3 |
| All C5.0 Algorithm | 18 | 0.29% | 20.2 |
| All Levenshtein Ratio | 18 | 0.38% | 14.7 |
| All Jaro Ratio | 18 | 0.44% | 13.8 |
| All Time Difference (Seconds) | 12 | 0.40% | 14.3 |
| All Square Root of Time Difference | 12 | 0.38% | 14.6 |
| All Log10 of Time Difference | 12 | 0.46% | 13.9 |
| Pruned CART Trees | 18 | 0.54% | 8.3 |
| Pruned C5.0 Trees | 18 | 0.29% | 20.2 |
| CART & Levenshtein | 9 | 0.54% | 8.2 |
| C5.0 & Levenshtein | 9 | 0.23% | 21.1 |
| CART & Jaro | 9 | 0.53% | 8.4 |
| C5.0 & Jaro | 9 | 0.34% | 19.2 |
| CART & Seconds | 6 | 0.54% | 8.3 |
| CART & Square Root | 6 | 0.53% | 8.3 |
| CART & Log10 | 6 | 0.55% | 8.3 |
| C5.0 & Seconds | 6 | 0.26% | 20.2 |
| C5.0 & Square Root | 6 | 0.23% | 20.8 |
| C5.0 & Log10 | 6 | 0.37% | 19.5 |
| Levenshtein & Seconds | 6 | 0.42% | 13.3 |
| Levenshtein & Square Root | 6 | 0.33% | 16.0 |
| Levenshtein & Log10 | 6 | 0.40% | 14.7 |
| Jaro & Seconds | 6 | 0.37% | 15.2 |
| Jaro & Square Root | 6 | 0.43% | 13.2 |
| Jaro & Log10 | 6 | 0.51% | 13.2 |
| CART & Levenshtein & Seconds | 3 | 0.55% | 8.0 |
| CART & Levenshtein & Square Root | 3 | 0.49% | 9.0 |
| CART & Levenshtein & Log10 | 3 | 0.58% | 7.7 |
| CART & Jaro & Seconds | 3 | 0.52% | 8.7 |
| CART & Jaro & Square Root | 3 | 0.56% | 7.7 |
| CART & Jaro & Log10 | 3 | 0.51% | 9.0 |
| C5.0 & Levenshtein & Seconds | 3 | 0.29% | 18.7 |
| C5.0 & Levenshtein & Square Root | 3 | 0.18% | 23.0 |
| C5.0 & Levenshtein & Log10 | 3 | 0.22% | 21.7 |
| C5.0 & Jaro & Seconds | 3 | 0.22% | 21.7 |
| C5.0 & Jaro & Square Root | 3 | 0.29% | 18.7 |
| C5.0 & Jaro & Log10 | 3 | 0.52% | 17.3 |

Table A2: High level summary of pruned decision tree experimentation results

# Appendix 2 – PST Parser

Provided by Espion.

```
import java.io.IOException;

public class PSTMain {

    public static void main(String[] args) {
        String pstFilePath = "C:/Users/jbrogan/workspace/PSTProcessor_Jie/Data/CollegeSent.pst";
        String csvFilePath = "C:/Users/jbrogan/workspace/PSTProcessor_Jie/Data/collegeSentPST.csv";
        new PSTMain(pstFilePath, csvFilePath);
    }

    public PSTMain(String filename, String csvFile) {
        try {
            PSTFile pstFile = new PSTFile(filename);
            System.out.println(pstFile.getMessageStore().getDisplayName());
            PrintWriter writer = new PrintWriter(csvFile, "UTF-8");
            processFolder(pstFile.getRootFolder(), writer);
            writer.close();
        } catch (Exception err) {
            err.printStackTrace();
        }
    }

    public void processFolder(PSTFolder folder, PrintWriter writer) throws PSTException, java.io.IOException {

        // go through the folders...
        if (folder.hasSubfolders()) {
            Vector<PSTFolder> childFolders = folder.getSubFolders();
            for (PSTFolder childFolder : childFolders) {
                processFolder(childFolder, writer);
            }
        }

        // and now the emails for this folder
        if (folder.getContentCount() > 0) {
            PSTMessage email = (PSTMessage) folder.getNextChild();
            while (email != null) {
                System.out.println(email.getSubject());
                writer.print("\"" + email.getSubject() + "\",");
                writer.print("\"" + email.getSenderEmailAddress() + "\",");
                writer.print("\"" + email.getDisplayTo() + "\",");
                writer.print("\"" + email.getOriginalDisplayCc() + "\",");
                writer.print("\"" + email.getOriginalDisplayBcc() + "\",");
                writer.print("\"" + email.getReceivedByAddress() + "\",");
                writer.print("\"" + email.getNumberOfRecipients() + "\",");
                writer.print("\"" + email.getClientSubmitTime() + "\",");
                writer.print("\"" + email.getNumberOfAttachments() + "\",");
                writer.print("\"" + email.getInternetMessageId() + "\",");
                writer.println("\"" + email.getInReplyToId() + "\",");
                email = (PSTMessage) folder.getNextChild();
            }
        }
    }

    private String readAttachment(PSTMessage email) throws PSTException, IOException {
        String fileNames = "";
        int numberOfAttachments = email.getNumberOfAttachments();
        for (int x = 0; x < numberOfAttachments; x++) {
            PSTAttachment attach = email.getAttachment(x);

            String filename = attach.getLongFilename();
            fileNames += filename + ",";

        }
        return fileNames;
    }

}
```

# Appendix 3 – Espion Approach Python Code

```
import csv   #used to import data from csv
from datetime import datetime   #used to extract datetime values from strings
startTime = datetime.now()  #to track processing time

data = open('C:\Users\John\Dropbox\MSc BA Materials\...\Espion Approach\FabMeta_Cleaned.csv')
csv_data = csv.reader(data)     #pulling the data in from the csv file
headers = csv_data.next()   #Creating a list of the headers so they can be added to output file
headers.extend(('Dupe_Test_Tag',)) #Adding these headers to the list of headers

#For sorting based on datetime, by extracting datetime properties from the string representations of the..
#...date and time pulled from the csv
sorted_data = sorted(csv_data, key=lambda row: datetime.strptime(row[9], "%d/%m/%Y %H:%M"))

#Next block compares sequentially sorted emails and labels them
prev = [None]   #used to record the values of the previous row
all_emails_with_comment = sorted_data   #list to append comments to emails when they are identified as dupes
i=0
for row in sorted_data:
#if DateTimeSent, From, To, CC, Subject, Attachment# are all exactly the same then mark as Exact Dupe & Sim=1.0
    if i > 0 and prev == [row[1],row[2],row[3],row[4],row[6],row[7],row[9]]:
        all_emails_with_comment[i].extend(('Dupe',))
    else:
        all_emails_with_comment[i].extend(('Non Dupe',))
    prev = [row[1],row[2],row[3],row[4],row[6],row[7],row[9]]
    i += 1

Output_Marked_Up = open("C:\Users\jbrogan\Desktop\Dissertation\Fabricated Sample Data_Marked Up.csv",'wb')
wr = csv.writer(Output_Marked_Up, quoting=csv.QUOTE_ALL)
wr.writerow(headers)
wr.writerows(all_emails_with_comment)
Output_Marked_Up.close()

data.close()
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 05 11:01:19 2015

@author: John
"""
```

# Appendix 4 – Example of Email Dissimilarity Code

This example is calculating the dissimilarity using Levenshtein ratio

```
import csv
from datetime import datetime
import Levenshtein   #python-Levenshtein 0.12.0

data = open('C:\Users\John\Dropbox\MSc BA Materials\...\DT Approaches\Merged_FabEmail&FabMetadata_Pairs.csv') #opening csv with metadata
csv_data = csv.reader(data)     #pulling the data in from the csv file
#creating headers
headers = ("Set","1stID","2ndID","From","To","CC","BCC","Subject","Text Body","AttNo.","Attachment Names","MessageID","InReplyToID","Sent Date")

MainOutput = []
next(csv_data)  #skipping first line
#rows in file contain data from both emails in pair
#for each row calculate the Jaro distances, difference between Attachment# and time difference
for row in csv_data:
    Output=[]
    FromDiff = Levenshtein.jaro(row[7],row[24])
    ToDiff = Levenshtein.jaro(row[8],row[25])
    CCDiff = Levenshtein.jaro(row[9],row[26])
    BCCDiff = Levenshtein.jaro(row[10],row[27])
    SubjectDiff = Levenshtein.jaro(row[11],row[28])
    TextDiff = Levenshtein.jaro(row[12],row[29])
    AttNoDiff = abs(float(row[13])-float(row[30]))
    AttDiff = Levenshtein.jaro(row[14],row[31])
    MIDDiff = Levenshtein.jaro(row[15],row[32])
    RIDDiff = Levenshtein.jaro(row[16],row[33])
    if datetime.strptime(row[17], "%d/%m/%Y %H:%M:%S") > datetime.strptime(row[34], "%d/%m/%Y %H:%M:%S"):
        TimeDiff = (datetime.strptime(row[17], "%d/%m/%Y %H:%M:%S")-datetime.strptime(row[34], "%d/%m/%Y %H:%M:%S")).total_seconds()
    else:
        TimeDiff = (datetime.strptime(row[34], "%d/%m/%Y %H:%M:%S")-datetime.strptime(row[17], "%d/%m/%Y %H:%M:%S")).total_seconds()
    Output.extend((row[0],row[3],row[20],FromDiff,ToDiff,CCDiff,BCCDiff,SubjectDiff,TextDiff,AttNoDiff,AttDiff,MIDDiff,RIDDiff,TimeDiff))
    MainOutput.append(Output)

MainOutput = sorted(MainOutput, key= lambda x: (x[0], [1])) #resorting output data

#creating the output file
MainOutputCSV = open("C:\Users\John\Dropbox\MSc BA Materials\...\DT Approaches\Jaro_Sqrt_Ints.csv",'wb')
wr = csv.writer(MainOutputCSV, quoting=csv.QUOTE_ALL)
wr.writerow(headers)
wr.writerows(MainOutput)
MainOutputCSV.close()

# -*- coding: utf-8 -*-
"""
Created on Thu Jul 16 12:14:06 2015

@author: John
"""
```

# Appendix 5 – Script for Creating EML Files

Provided by Espion.

**EmailRecord.java**

```
import java.util.Date;

public class EmailRecord {
    Long ID;
    Long SID;
    Long DID;
    String Custodian;
    String From;
    String To;
    String CC;
    String BCC;
    String Subject;
    String TextBody;
    Long AttachmentNu;
    String AttachmentNames;
    String MessageID;
    String InReplyToID;
    Date SentDate;
    String Note;
    String Label;

    public EmailRecord(Builder builder) {
        ID = builder.ID;
        SID = builder.SID;
        DID = builder.DID;
        Custodian = builder.Custodian;
        From = builder.From;
        To = builder.To;
        CC = builder.CC;
        BCC = builder.BCC;
        Subject = builder.Subject;
        TextBody = builder.TextBody;
        AttachmentNu = builder.AttachmentNu;
        AttachmentNames = builder.AttachmentNames;
        MessageID = builder.MessageID;
        InReplyToID = builder.InReplyToID;
        SentDate = builder.SentDate;
        Note = builder.Note;
        Label = builder.Label;
    }

public static class Builder {
    Long ID;
    Long SID;
    Long DID;
    String Custodian;
    String From;
    String To;
    String CC;
    String BCC;
    String Subject;
    String TextBody;
    Long AttachmentNu;
    String AttachmentNames;
    String MessageID;
    String InReplyToID;
    Date SentDate;
    String Note;
    String Label;

    public Builder() {

    }

    public Builder ID(String val) {
        if (val != null && val.length() != 0)
            ID = Double.valueOf(val).longValue();
        return this;
    }

    public Builder SID(String val) {
        if (val != null && val.length() != 0)
            SID = Double.valueOf(val).longValue();
        return this;
    }

    public Builder DID(String val) {
        if (val != null && val.length() != 0)
            DID = Double.valueOf(val).longValue();
        return this;
    }

    public Builder Custodian(String val) {
        Custodian = val;
        return this;
    }

    public Builder From(String val) {
        From = val;
        return this;
    }

    public Builder To(String val) {
        To = val;
        return this;
    }

    public Builder CC(String val) {
        CC = val;
        return this;
    }

    public Builder BCC(String val) {
        BCC = val;
        return this;
    }

    public Builder Subject(String val) {
        Subject = val;
        return this;
    }

    public Builder TextBody(String val) {
        TextBody = val;
        return this;
    };

    public Builder AttachmentNu(String val) {
        AttachmentNu = Long.valueOf(val);
        return this;
    };
```

## EmailRecord.java…continued

```
        public Builder AttachmentNames(String val) {
            AttachmentNames = val;
            return this;
        };

        public Builder MessageID(String val) {
            MessageID = val;
            return this;
        };

        public Builder InReplyToID(String val) {
            InReplyToID = val;
            return this;
        };

        public Builder SentDate(Date val) {
            SentDate = val;
            return this;
        };

        public Builder Note(String val) {
            Note = val;
            return this;
        };

        public Builder Label(String val) {
            Label = val;
            return this;
        };

        public EmailRecord build() {
            return new EmailRecord(this);
        }
```

## Main.java

```
import java.util.List;

public class Main {

    public static void main(String[] args) {
        try {
            List<EmailRecord> records = EXCELReader
                    .readEmailRecords("C:/Users/Desktop/john/data/FabEmail.xlsx");

            String fileOutputPath = "C:/Users/Desktop/john/output/FabEmail";
            EMLFileCreator creater = new EMLFileCreator(fileOutputPath);
            for (EmailRecord record : records) {
                System.out.println("create:" + record.ID);
                creater.createEMLInputStream(record);
            }

        } catch (Exception ioe) {
            ioe.printStackTrace();
        }
    }
}
```

## EMLFileCreator.java

```
import java.io.File;

public class EMLFileCreator {

    String outputFolderPath;

    public EMLFileCreator(String outputFolderPath) {
        this.outputFolderPath = outputFolderPath;
        File outputFolder = new File(this.outputFolderPath);
        outputFolder.mkdir();
    }

    public void createEMLInputStream(EmailRecord record)
            throws AddressException, MessagingException, FileNotFoundException, IOException {

        Message message = new MessageWithID(Session.getInstance(System.getProperties()), record.MessageID);
        if (record.From != null)
            message.setFrom(new InternetAddress(record.From));
        if (record.To != null)
            message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(record.To));
        if (record.CC != null)
            message.setRecipients(Message.RecipientType.CC, InternetAddress.parse(record.CC));
        if (record.BCC != null)
            message.setRecipients(Message.RecipientType.BCC, InternetAddress.parse(record.BCC));
        if (record.Subject != null)
            message.setSubject(record.Subject);
        if (record.SentDate != null)

         // create the message part
         MimeBodyPart content = new MimeBodyPart();
         if (record.TextBody != null)
             content.setText(record.TextBody);

         Multipart multipart = new MimeMultipart();
         multipart.addBodyPart(content);
         // add attachments
         if (record.AttachmentNames != null) {
             String[] attachmentNames = record.AttachmentNames.split(";");
             for (String attachmentName : attachmentNames) {
                 MimeBodyPart attachmentPart = new MimeBodyPart();
                 attachmentPart.setFileName(attachmentName);
                 attachmentPart.setDisposition(Part.ATTACHMENT);
                 attachmentPart.setText("");
                 multipart.addBodyPart(attachmentPart);
             }
         }

         // integration
         message.setContent(multipart);
         String folderPath = outputFolderPath + "/" + record.Custodian;
         File custodianFolder = new File(folderPath);
         custodianFolder.mkdir();

         File emlFile = new File(folderPath + "/" + record.ID + ".eml");
         FileOutputStream aFileOutputStream = new FileOutputStream(emlFile);
         message.writeTo(aFileOutputStream);
         aFileOutputStream.close();
     }

}
```

## EXCELReader.java

```
import java.io.File;

public class EXCELReader {

    public static List<EmailRecord> readEmailRecords(String xlsxFilePath)
            throws EncryptedDocumentException, InvalidFormatException, IOException {

        ArrayList<EmailRecord> records = new ArrayList<EmailRecord>();
        File file = new File(xlsxFilePath);
        InputStream inp = new FileInputStream(file);
        Workbook wb = WorkbookFactory.create(inp);
        Sheet sheet = wb.getSheetAt(0);
        int rowNu = sheet.getPhysicalNumberOfRows();
        Row row = null;
        String ID = null, SID = null, DID = null, Custodian = null, From = null, To = null, CC = null,
                BCC = null, Subject = null, TextBody = null, AttachmentNu = null, AttachmentNames = null,
                MessageID = null, InReplyToID = null, Note = null, Label = null;
        Date SentDate = null;
        EmailRecord record = null;
        for (int rowId = 1; rowId < rowNu; rowId++) {
            row = sheet.getRow(rowId);

            if (row.getCell(0) != null)
                ID = row.getCell(0).toString();
            if (row.getCell(1) != null)
                SID = row.getCell(1).toString();
            if (row.getCell(2) != null)
                DID = row.getCell(2).toString();
            if (row.getCell(3) != null)
                Custodian = row.getCell(3).toString();
            if (row.getCell(4) != null)
                From = row.getCell(4).toString();
            if (row.getCell(5) != null)
                To = row.getCell(5).toString();
            if (row.getCell(6) != null)
                CC = row.getCell(6).toString();
            if (row.getCell(7) != null)
                BCC = row.getCell(7).toString();
            if (row.getCell(8) != null)
                Subject = row.getCell(8).toString();
            if (row.getCell(9) != null)
                TextBody = row.getCell(9).toString();
            if (row.getCell(10) != null)
                AttachmentNu = row.getCell(10).toString();
            if (row.getCell(11) != null)
                AttachmentNames = row.getCell(11).toString();
            if (row.getCell(12) != null)
                MessageID = row.getCell(12).toString();
            if (row.getCell(13) != null)
                InReplyToID = row.getCell(13).toString();
            if (row.getCell(14) != null)
                SentDate = row.getCell(14).getDateCellValue();
            if (row.getCell(15) != null)
                Note = row.getCell(15).toString();
            if (row.getCell(16) != null)
                Label = row.getCell(16).toString();

            record = new EmailRecord.Builder().ID(ID).SID(SID).DID(DID).Custodian(Custodian).From(From).To(To)
                    .CC(CC).BCC(BCC).Subject(Subject).TextBody(TextBody).AttachmentNames(AttachmentNames)
                    .MessageID(MessageID).SentDate(SentDate).Note(Note).Label(Label).build();

            System.out.print("read:" + record.ID);

            records.add(record);
            System.out.println();
        }
        return records;
    }
}
```

## MessageWithID.java

```
import java.io.InputStream;

class MessageWithID extends MimeMessage {

    String messageID;

    public MessageWithID(Folder folder, InputStream is, int msgnum) throws MessagingException {
        super(folder, is, msgnum);
        // TODO Auto-generated constructor stub
    }
    public MessageWithID(Folder folder, int msgnum) {
        super(folder, msgnum);
        // TODO Auto-generated constructor stub
    }
    public MessageWithID(Folder folder, InternetHeaders headers, byte[] content, int msgnum)
            throws MessagingException {
        super(folder, headers, content, msgnum);
        // TODO Auto-generated constructor stub
    }
    public MessageWithID(MimeMessage arg0) throws MessagingException {
        super(arg0);
        // TODO Auto-generated constructor stub
    }
    public MessageWithID(Session session, InputStream is) throws MessagingException {
        super(session, is);
        // TODO Auto-generated constructor stub
    }
    public MessageWithID(Session session, String messageID) {
        super(session);
        this.messageID = messageID;
        // TODO Auto-generated constructor stub
    }
    protected void updateMessageID() throws MessagingException {
        setHeader("Message-ID", this.messageID);
    }
}
```

# Appendix 6 – Decision Tree Building Code From R

## Using the Tree package

```
'
<<The core of this code was taken from Professor Michael O'Neill's
lecture notes from the 'Data Mining & Applications MIS40970'>>

data = LevRatio_Log_Ints
set.seed = (1234)
ind = sample(2,nrow(data),replace=TRUE,prob=c(0.5,0.5))
train=data[ind==1,]
test=data[ind==2,]

library(tree)

tree.data=tree(train$Dupe.Type~., train)
summary(tree.data)

test.data=predict(tree.data, test, type = "class")
table(test.data,test$Dupe.Type)

cv.data=cv.tree(tree.data,FUN=prune.misclass)
cv.data

<<Pick the 'size' (i.e. terminal nodes) with reasonable deviance>>
<<This is the size of nodes for pruned tree>>

prune.data=prune.misclass(tree.data,best=??)
summary(prune.data)
test.data=predict(prune.data, test, type = "class")
table(test.data,test$Dupe.Type)

<<If misclassifcation higher than unpruned tree,
try the next size tree up,
until you reach the same size as the unpruned tree>>
```

## Using the C50 package

```
data = LevRatio_Log_Ints
set.seed = (1234)
ind = sample(2,nrow(data),replace=TRUE,prob=c(0.5,0.5))
train=data[ind==1,]
test=data[ind==2,]

library(C50)

tree.data=C5.0(train$Dupe.Type~., data=train, control = C5.0Control(noGlobalPruning = TRUE))
summary(tree.data)

test.data=predict(tree.data, test, type = "class")
table(test.data,test$Dupe.Type)

tree.data=C5.0(train$Dupe.Type~., data=train, control = C5.0Control(noGlobalPruning = FALSE))
summary(tree.data)

test.data=predict(tree.data, test, type = "class")
table(test.data,test$Dupe.Type)
```

# Appendix 7 – Example of Code Used to Map Pair-Based Data Back to Individual Emails

```
import csv
data = open('C:\Users\John\Dropbox\MSc BA Materials\Producing the Dissertation\My Dissertation Folder\
Experiments\Data\Datasets\DT Approaches\Jaro_Sqrt_Ints\FabEmail_Jaro_Sqrt_Ints.csv')
csv_data = csv.reader(data)     #pulling the data in from the csv file
headers = csv_data.next()   #Creating a list of the headers so they can be added to output file
headers.extend(('DT_Tag',)) #Adding these headers to the list of headers

i=0
#rowdata=()
output=[]
ID_labels=[]

#Example of a representation of a DT built in R
for row in csv_data:
    if float(row[8]) < 0.831655:
        if float(row[12]) < 0.707107:
            row.extend(('Near_Dupe','3'))
        else:
            row.extend(('Non_Dupe','0'))
    else:
        if float(row[12]) < 3.67945:
            if float(row[8]) < 0.998911:
                row.extend(('Near_Dupe','3'))
            else:
                if float(row[10]) < 0.983333:
                    row.extend(('Near_Dupe','3'))
                else:
                    if float(row[4]) < 0.994792:
                        row.extend(('Near_Dupe','3'))
                    else:
                        row.extend(('Exact_Dupe','4'))
        else:
            if float(row[4]) < 0.842262:
                    row.extend(('Non_Dupe','0'))
            else:
                if float(row[8]) < 0.963828:
                    row.extend(('Non_Dupe','0'))
                else:
                    row.extend(('Far_Dupe','2'))

    output.append(row)
    i += 1

    indiv_output_line=(row[3],row[13],row[2],row[14])
    ID_labels.append(indiv_output_line)
```

CODE CONTINUED ON NEXT PAGE

CODE CONTINUED FROM LAST PAGE

```
sorted_ID_labels = sorted(ID_labels, key=lambda x: int(x[0]))

#mapping pair-based results back to individual emails
j=2 #ID number
k=0 #Dupe type number
Dupeoutput=[]
for row in sorted_ID_labels:
    rowoutput=[]
    if int(row[0]) == j:
        if int(row[3]) > k:
            k = int(row[3])
    else:
        if k==0:
            rowoutput.extend((j,'Non_Dupe'))
        elif k==1:
            rowoutput.extend((j,'Content_Dupe'))
        elif k==2:
            rowoutput.extend((j,'Far_Dupe'))
        elif k==3:
            rowoutput.extend((j,'Near_Dupe'))
        elif k==4:
            rowoutput.extend((j,'Exact_Dupe'))
        j = int(row[0])
        k = int(row[3])
        Dupeoutput.append(rowoutput)


AllEmailsLabeledCSV = open("C:\Users\John\Dropbox\MSc BA Materials\Producing the Dissertation\My
wr = csv.writer(AllEmailsLabeledCSV, quoting=csv.QUOTE_ALL)
wr.writerow(headers)
wr.writerows(output)
AllEmailsLabeledCSV.close()

IDCSV = open("C:\Users\John\Dropbox\MSc BA Materials\Producing the Dissertation\My Dissertation
wr = csv.writer(IDCSV, quoting=csv.QUOTE_ALL)
ID_labels_headers=('2ndID','DT_Tag','1stID')
wr.writerow(ID_labels_headers)
wr.writerows(ID_labels)
IDCSV.close()

DupeLabels = open("C:\Users\John\Dropbox\MSc BA Materials\Producing the Dissertation\My Disserta
wr = csv.writer(DupeLabels, quoting=csv.QUOTE_ALL)
wr.writerows(Dupeoutput)
DupeLabels.close()

data.close()
```

# Glossary of terms

C5.0 = Decision tree building algorithm

CART = Classification And Regression Tree

DID = Duplicate ID

DT = Decision Tree

EML = Common email file format

ESI = Electronically Store Information

FabEmail = Corpus of email fabricated for this study

FabMetadata/FabMeta = Dataset of metadata fabricated for this study

FN = False Negative

FP = False Positive

MID = Message ID

NDD = Near Duplication Detection

PID = Pair ID

PST = Personal Storage Table (Microsoft file format for storing messages and events)

QA = Quality Assurance

SID = Send ID

TF-IDF = Term Frequency/Inverse Document Frequency

TN = True Negative

TP = True Positive

UTC = Coordinated Universal Time

## References

Barros, Rodrigo C., André C. P. L. F. de Carvalho, and Alex A. Freitas. *Automatic Design of Decision-Tree Induction Algorithms.* Springer, 2015.

Conrad, Jack G. "E-Discovery Revisited: A Broader Perspective for IR Researchers." *DESI: Workshop on Supporting Search and Sensemaking for Electronically Stored Informationin Discovery Proceedings.* 2007. 237-246.

Donald, Metzler, Susan Dumais, and Christopher Mee. "Similarity measures for short segments of text." In *Advances in Information Retrieval*, by Various, 16-27. Berlin: Springer Berlin Heidelberg, 2007.

Equivio. "Near-duplicate Detection in Electronic and OCR Collections." *http://www.equivio.com/.* 2012. http://www.equivio.com/files/files/Case%20Study%20-%20Near-duplicate%20Detection%20in%20Electronic%20and%20OCR%20Collections%20-%20Commonwealth%20Legal.pdf (accessed May 27, 2014).

Hajishirzi, Hannaneh, Wen-tau Yih, and Aleksander Kolcz. "Adaptive Near-Duplicate Detection via Similarity Learning." *SIGIR'10.* Geneva: ACM, 2010. 419-426.

Ioannis, Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. "Email mining: emerging techniques for email management." In *Web Data Management Practices: Emerging Techniques and Technologies*, by Athena Vakali, 219-240. IGI Globa, 2006.

Jake, Brutlag D., and Christopher Mee. "Challenges of the email domain for text classification." *International Conference on Machine Learning.* Stanford, 2000. 103-110.

Joshi, Sachindra, Prasad M Deshpande, Thomas Hampp, Kenney Ng, and Danish Contractor. "Auto-Grouping Emails For Faster E-Discovery." *The Proceedings of the VLDB Endowment*, 2011: 1284-1294.

Kennedy, Alistair, and Stan Szpakowicz. "A supervised method of feature weighting for measuring semantic relatedness." *Advances in Artificial Intelligence*, 2011: 222-233.

Klimt, Bryan, and Yiming Yang. "The Enron Corpus: A New Dataset for Email Classification Research." *ECML 2004, 15th European Conference on Machine Learning.* Pisa: Springer-Verlag Berlin Heidlberg, 2004. 217-226.

Kumar, Vijaya, and G. Santhi . "Optimized near Duplicate Matching scheme for E-mail Spam Detection." *International Journal of Scientific & Engineering Research*, 2012.

Matwin, Stan , and Svetlana Kiritchenko. "Email classification with co-training." *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research.* 2011. 301-312.

Meizhen, Wang , Li Zhitang, and Zhong Sheng. "A Method for Spam Behavior Recognition Based on Fuzzy Decision Tree." *IEEE Ninth International Conference on Computer and Information Technology.* Xiamen, 2009. 237-241.

Patel, Brijain R, and Kushik K Rana. "A Survey on Decision Tree Algorithm for Classification." *International Journal of Engineering Development and Research*, 2014: 1-5.

Rajaraman, Anand, Jure Leskovec, and Jeffrey D. Ullman. *Mining of Massive Datasets.* Cambridge University Press, 2011.

Richter, Georg, and Andrew MacFarlane. "The impact of metadata on the accuracy of automated patent classification." *World Patent Information*, 2005: 13-26.

Sachindra, Joshi, Kenney Ng, and Sandeep Singh. Dynamically Detecting Near-Duplicate Documents. United States of America Patent Application. 3 February 2011.

Seshasai, Shreyes. *Efficient Near Duplicate Document Detection for Specialized Corpora.* MEng Thesis, Massachusetts, 2009.

Shi, Lei, Qiang Wang, Xinming Ma, Mei Weng, and Hongbo Qiao. "Spam Email Classification Using Decision Tree Ensemble." *Journal of Computational Information Systems*, 2012: 949–956.

Symantec. "Deduplication in Compliance Accelerator and Discovery Accelerator." *www.symantec.com.* 10 February 2011. http://kbdownload.symantec.com/resources/sites/BUSINESS/content/live/DOCUMENTATION/3000/DOC3621/en_US/Accelerator_Deduplication.pdf?__gda__=1401227507_dfa64cc289c3b2c222ab60717da41f7b (accessed May 27, 2014).

—. "What is the Criteria for file deduplication." *www.symantec.com.* 9 January 2014. https://support.symantec.com/en_US/article.TECH197700.html (accessed June 13, 2015).

Tang, Guanting, Jian Pei, and Wo-Shun Luk. "Email mining: tasks, common techniques, and tools." *Knowledge and Information Systems*, 2013: 1-31.

Wang, Zhe , William Josephson, Qin Lv, and Moses Charika. *Filtering Image Spam with Near-Duplicate Detection.* Academic Paper, Princeton: Princeton University, 2007.

Wenjuan, Li, Meng Weizhi, Tan Zhiyuan, and Xia Yang. "Towards Designing An Email Classification System Using Multi-View Based." *2014 IEEE 13th International*

*Conference on Trust, Security and Privacy in Computing and Communications.* Beijing, 2014. 175-181.

Xiao, Chuan, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. "Efficient Similarity Joins for Near Duplicate Detection." *ACM Transactions on Database Systems*, 2011: 15.

Yang, Hui, and Jamie Callan. "Near-Duplicate Detection by Instance-level Constrained Clustering." *SIGIR '06 Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval.* New York: ACM, 2006. 421 - 428.

Yih, Wen-tau. "Learning Term-weighting Functions for Similarity Measures." *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing.* Singapore: ACL, 2009. 793-802.

Yih, Wen-tau, and Christopher Meek. "Improving Similarity Measures for Short Segments of Text." *Association for the Advancement of Artificial*, 2007: 1489-1494.

Zhang, Le, Jingbo Zhu, and Tianshun Yao. "An Evaluation of Statistical Spam Filtering Techniques." *ACM Transactions on Asian Language Information Processing*, December 2004: 243-269.

Zhong, Mingyu. *An Analysis of Misclassification Rates for Decision Trees.* Orlando, 2007.

# List of Contributors

Damir Kahvedzic - Espion

Jie Yang - Espion

Conor Gavin - Espion