Coderivative Document Recognition

Stijn Johannes de Reede

Coderivative Document Recognition

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

MEDIA AND KNOWLEDGE ENGINEERING

by Stijn Johannes de Reede
born in Lekkerkerk, the Netherlands

Man-Machine Interaction Group
Faculty EEMCS
Delft University of Technology
the Netherlands
www.ewi.tudelft.nl

ECM & UX Practice of Technology
Technology Services
Capgemini Nederland B.V.
the Netherlands
www.nl.capgemini.com

Author Details:

Candidate:	Stijn Johannes de Reede
Student number:	1017020
Email:	stijndereede@gmail.com

Thesis Committee:

Chair:	Dr. L.J.M. Rothkrantz
	Man-Machine Interaction Group, EEMCS, Delft University of Technology
Member:	Ir. P. Wiggers
	Man-Machine Interaction Group, EEMCS, Delft University of Technology
Member:	Ir. B.R. Sodoyer
	Information Systems Algorithms Group, EEMCS, Delft University of Technology
Supervisor:	Drs. O. Stegeman
	ECM & UX Practice of Technology, Capgemini Nederland B.V.

Preface

Beginning with this research project, the mountain before me seemed large, and the journey ahead long and unknown. Although slowly climbing at first, I noticed that once a critical mass had been reached, the incline became smaller everyday. Looking back, I cannot say where precisely the mountain was, or when I passed its peak. It seems the rocky mountain flattened out to a meadowy landscape, not so different from the surroundings where the Capgemini is located.

For the fact that my rock-climbing endeavour turned into a hike though some grasslands I have many people to thank. First of all of course my supervisor and chair of the thesis committee, Leon Rothkrantz. I always enjoyed our talks which for some reason always took longer than expected and ended up on a subject completely different from my thesis. Secondly, my supervisor from Capgemini, Onno Stegeman for the freedom and guidance he gave me during my thesis.

Stijn de Reede
Delft, June 10, 2008

Abstract

Knowledge management in large enterprises currently depends very much on the active participation of the employees. If an employee is unable or unwilling to share his knowledge, the knowledge management fails. To counter this situation, the documents of all employees can be collected automatically and stored in a knowledge management application. However, since every document is subject to change during its lifecycle, many versions of a document will be created, and thus collected and stored. A query put to a knowledge management application would produce a result list that is polluted by these many versions of all documents, which reduces the accessibility and usability of the documents in the knowledge management application.

We provide a solution to this problem by introducing the Cayman system. This software system consists of the implementation of a new algorithm that is able to recognize different versions of documents, or coderivative documents. Additionally, it consists of a prototype application that collects documents, and provides a way to fully integrate our algorithm with an existing well-known knowledge management application.

We compare our algorithm with six other well known algorithms using a real life dataset. The algorithms are evaluated with several useful graphical methods, and, most importantly, with one quantitative method. This enables us to make a solid comparison of their performance. Our experiment shows that the newly introduced algorithm surpasses every other algorithm, except for one. Surprisingly, this is the most simple baseline algorithm, which all algorithms should outperform.

Despite the fact that our algorithm's performance is not yet optimal, we can say that the coderivative document recognition performs well enough to be of practical use. The Cayman system is put to a practical test in a professional environment, and succeeds in collecting documents, recognizing coderivatives, and making them accessible and reusable for employees.

Contents

List of Tables

List of Figures

List of Listings

Chapter 1.

Introduction

With the Information Age well underway, the availability of information is becoming more widespread everyday. Not only the Internet is overflowing with user generated content through Web 2.0 technologies such as blogs and wikis, but also enterprises are overflowing with this tidal wave of information, due to the same approach of collaboration through digital channels.

As the mountain of information on the Internet and within enterprises gets larger and larger, the need for an efficient way to retrieve relevant information arises. On the Internet many search engines flourish, resulting in multibillion dollar companies such as Google and Yahoo. These companies are continuously introducing innovations to enhance their index and search techniques that aid the user in finding his way through the maze of available information. One of these innovations is the introduction of the *Similar pages* functionality. This provides the user, once he has stumbled upon an interesting piece of information, with an option to instruct the search engine to find pages similar to the one that has caught his interest. Enterprises are also embracing digital technology to disclose information to their employees. Documents in digital form are collected and stored in a central location, which provides search facilities for users to be able to retrieve information of interest. Advanced search engine techniques such as Similar pages have found their way from the Internet to these information management systems in enterprises. Google even offers specialized search engine solutions for business environments.

In the modern business environment information is usually spread through digital channels such as email. The ease with which this communication takes place leads to information being distributed among many people in an enterprise. Most of the information that is spread in an enterprise takes the form of textual documents. Since collaboration is an everyday practice in these companies, different people will work on the distributed documents, possibly creating many versions of them. When these documents are collected to be made accessible for employees, a new problem surfaces. The many different versions of the documents will flood the central storage location and clutter up the search results, resulting in a lower usability of the central information system and search facilities.

To clarify things, we use the following analogy. Suppose the box in Figure 1.1 represents the central location where documents are stored, and the various shapes in the box each represent an individual document. Querying the central storage location for information that is of interest to us is equivalent to finding a document with certain properties. This translates in the analogy to finding shapes with certain properties in the box. For example, consider the following query:

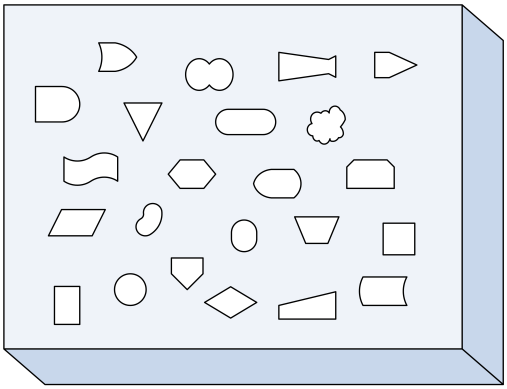

Figure 1.1: *Documents represented as shapes in a central storage location*

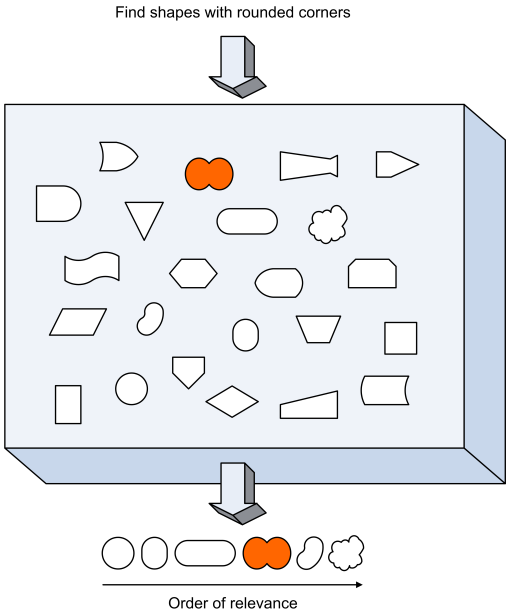

Figure 1.2: *Finding shapes with rounded corners*

"Find shapes with rounded corners", see Figure 1.2. The actual piece of information we are interested in is represented by the orange colored shape with the two joined circles, coming in fourth place in the search result that is ordered by relevance. This is an acceptable outcome, and we are able to quickly spot the interesting piece of information in the search result.

Now consider what happens if this system is placed in a collaborative environment, where many people will create many versions of the documents. A new version of an original document is represented by an identical shape with a different outline color. If the same query is put to the storage location, the search result is cluttered with these versions, and the document of interest only comes in eleventh place, much more difficult to spot, see Figure 1.3. If we are able to automatically group these versions together in the central storage location, the result for the query "Find shapes with rounded corners" becomes orderly again, and our document of interest is back in fourth place, see Figure 1.4.

Although this analogy might seem oversimplified, it is a direct translation of the problem that surfaces with automatic document collection in enterprises. In such an environment the documents are widespread and subject to continuous change by many users. If these documents are gathered periodically and put in a central storage location, measures have to be taken to prevent flooding of the system with different versions of the created documents. Since information and knowledge are becoming key assets of enterprises, proper management of these documents

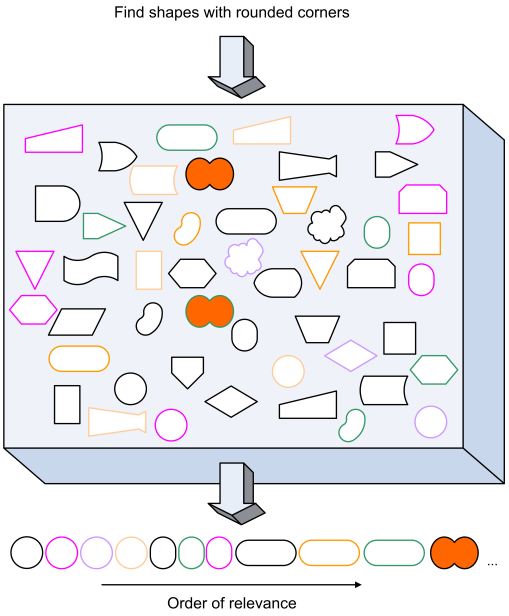

Figure 1.3: *Finding shapes with rounded corners, cluttered by multiple versions*

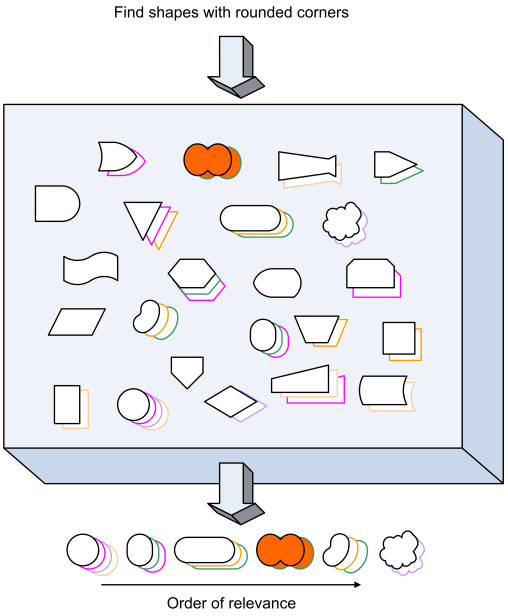

Figure 1.4: *Finding shapes with rounded corners, with grouped versions*

is crucial. This situation is what motivated us to conduct a thorough research into the field of version recognition.

1.1. Preliminaries

1.1.1. What is knowledge?

Prior to discussing (the problems of) knowledge management, we need to define knowledge itself. The Oxford English dictionary defines knowledge as:

> "(i) expertise, and skills acquired by a person through experience or education; the theoretical or practical understanding of a subject,
> (ii) what is known in a particular field or in total; facts and information,
> (iii) awareness or familiarity gained by experience of a fact or situation."

This broad definition proves to be of little use for us, and is it clear that we need something more specific to our field of research. However, as Bechina and Bommen (2006) state, there is no commonly accepted definition of knowledge. The first of the definitions given by the

Oxford English dictionary comes close to what we have in mind, but instead we have adopted the definition of Ayer (1956):

Knowledge "Knowledge is the justified true belief that increases an individuals capacity to take action."

The phrase "justified true belief" originates from the start of the philosophical debates on what knowledge is, and was coined by Plato in his dialog Theaetetus. According to this phrase, in order for a statement to be knowledge, three criteria have to be met. It has to be justified, which is to say that it has to have an explanation why the belief is true, or how one knows it is true. Secondly, it has to be true. Many theories on what truth is exist in philosophy and we adopt the correspondence theory of truth: "true beliefs and true statements correspond to the actual state of affairs" (Prior, 1967). Or, in other words, a true belief or statement has to be consistent with the world. Thirdly, the statement must be believed by someone, i. e. an individual must hold the statement to be true. The second part of the definition given by Ayer refers to the fact that the statement must be useful to someone. It has to provide insight and contribute to his decision making process.

We divide knowledge into two types: tacit and explicit knowledge. The scientist and philosopher Polanyi (1958) introduced the concept of tacit knowing. Polanyi's main idea was that knowledge was inevitably hidden and subconscious. The notion that knowledge could be made explicit and put into an information system was contradictory to everything he believed in. We relax his concept of tacit knowing somewhat and utilize the nowadays more common characterization of tacit knowledge. Tacit knowledge is intangible and it relates to the subconscious, intuition, beliefs and experiences. It is personal and cannot be easily captured and codified in order to communicate to many people. The only way tacit knowledge can be shared between people is through discussion and personal contact. Even then, the exchange of tacit knowledge is limited, as one of Polanyi's famous aphorisms states: "We know more than we can tell".

Opposite to tacit knowledge is explicit knowledge. This is knowledge that can easily be captured, codified and communicated. Note that explicit knowledge does not need to be explicit in form, it can very well reside in the mind of an individual. Contrary to tacit knowledge however, it can be made explicit in form without compromising the underlying knowledge itself. Many enterprises impose a set of rules and procedures on their employees in the form of design methodologies, document formats, etc. to guide the manifestation of knowledge into a rigid framework. Although not all knowledge is suitable to be subjected to this kind of standardization, it certainly helps to make the knowledge explicit in form.

When discussing knowledge throughout the rest of this thesis, we refer only to explicit knowledge, as we consider tacit knowledge to be out of the scope of this work.

1.1.2. What is knowledge management?

As with knowledge itself, we also need a definition of knowledge management. O'Leary (1998) uses the following phrase to define knowledge management, which we adopt:

Knowledge management "Enterprise knowledge management entails formally managing knowledge resources in order to facilitate access and reuse of knowledge, typically by using advanced information technology."

Informally knowledge management has always existed in forms such as on-the-job peer discussions, corporate libraries and yellow pages, discussion forums and training and mentoring programs. Modern forms of unstructured knowledge management, often facilitated through the Internet, include blogs, wikis, social network websites and specialized communities. With the rise of information technology, the possibility for a more organized form of knowledge management has become apparent. Nowadays, is not uncommon for companies to have the post Chief Knowledge Officer (CKO), a senior executive, responsible for creating an infrastructure and cultural environment for knowledge sharing.

In today's fast paced global economy, competition between companies is fierce, and any method to gain an advantage over the competition is appreciated. Enterprises must share information quickly, within the departments and throughout the company. A view held by many enterprises is that having the right type of knowledge management results in higher organizational effectiveness, with improved or faster learning and new knowledge creation. In today's dynamic global business environment, knowledge is one of the most competitive resources. Within a consultancy such as Capgemini it is even more important to focus on knowledge management, since it is the single most important asset. Therefor, it is crucial that this knowledge is managed properly. As the definition states, this requires that the knowledge is accessible and can be reused. Employees should transfer their knowledge to an explicit form, in order for it to be accessible and reusable. Capturing knowledge is often difficult for a consultancy, due to several factors: employees deal with high workloads and strict deadlines leaving no time for active knowledge sharing, high employee turnover rates result in the loss of knowledge and a dynamic work location reduces communication among employees. In addition to these difficulties, there is often a cultural barrier, and employees might be reluctant or even unwilling to share knowledge with colleagues (Husted and Michailova, 2002; Gupta and Michailova, 2004).

1.1.3. Problem definition and solution approach

The problem central to this thesis, is that within an enterprise the knowledge sharing is often lacking, although a knowledge management application is in place. We attribute this to the fact that employees fail to actively share knowledge. Whether this is due to time shortage or unwillingness, we consider to be of lesser importance. We advocate to increase knowledge sharing by reducing the action required by the employees. Firstly, we postulate that at least part of the explicit knowledge present in the minds of employees manifests itself through the documents they produce. Although this limits the range of knowledge available (as not all knowledge is represented by documents), we consider sharing the knowledge in documents as a good first step towards proper knowledge sharing.

Our approach is to improve the sharing of knowledge by the automatic collection of documents. Even though employees might not consistently participate in structured knowledge sharing, they do deliberately spread documents among colleagues to share part of their knowledge. However, this brings forth another problem which we will aim to solve in this thesis; the wide distribution of (identical) documents with no central management of knowledge resources. If, in addition to this, employees work simultaneously on a previously distributed document, keeping track of the document and all its revisions as a whole is becoming more and more difficult. To refrain from sharing a document until it has reached the final version is not a good strategy either, since intermediate versions can also contain knowledge that might be useful to someone.

By periodically gathering documents from employees, the knowledge management system would soon become flooded with (nearly) identical documents. Our aim is to prevent this by designing a system that automatically recognizes these documents and labels them appropriately in the knowledge system. To denote these documents, we adopt the definition, following Hoad and Zobel (2003):

Coderivative documents Documents that originate form the same source.

The word *source* in this definition does not refer to a physical source, such as a computer or even a person. Rather, it denotes a common document from which the documents are derived. Note that a document is always derived from itself, and thus forms a coderivative pair with an identical copy of itself. Another common occurrence of a coderivative document pair being formed is when an author plagiarizes other work. The created document then forms a coderivative pair with the original document. Our focus however is on versioned documents. Any revision of a document forms a coderivative pair with any other revision of that same document. Although the difference between an initial version and the final version of a document may be large, we still designate them a coderivative document pair. In practice, changes between consecutive versions are small if the documents are gathered periodically.

Throughout this thesis we make a distinction between coderivative documents and similar documents. The latter are documents that do not have a common ancestor, but are similar in topic. Thus we define similar documents as:

Similar documents Documents that discuss the same main topic, or at least share a certain amount of sub topics between them.

This is a somewhat vague and broad definition and we realize that the disparity of the two classes of documents is small. In fact, any coderivative document pair is also a similar document pair, while the opposite does not hold.

1.2. Research question

After having defined the problem and our solution approach, we can formulate our research question as follows:

Is it possible to automatically recognize coderivative documents?

Although the core question is if coderivative documents can be recognized, our perspective in answering the question is a bit broader. If a positive answer can be given to this question, it would be a substantial contribution to knowledge management and enable an automatic approach to it. By recognizing coderivative documents, it is prevented that the knowledge management system overflows with these documents. It would improve knowledge management since automatic document (and thus knowledge) gathering would be feasible without flooding the system with these documents. A coderivative recognition system is able to group coderivative documents together in a knowledge management system. If one would periodically gather documents of employees without such a system, the practical usability would become hugely compromised. A query for a certain piece of knowledge of interest to us will produce a list of documents that is cluttered with every revision of every document that matches the query, and the actual document that interests us might be obscured from view.

1.2.1. Research objectives

In order to provide an answer to the research question, we aim to fulfill the following objectives:

1. To select the best algorithm currently available as the basis for our new coderivative document recognition algorithm, based on research into related work
2. To select and enhance currently available methods to compare the algorithms
3. To design and implement a new coderivative document recognition algorithm
4. To validate the performance of the new algorithm against the currently available algorithms
5. To validate the practical use of the new approach to knowledge management by designing and implementing a software system that
    a) Automatically collects documents from employees
    b) Makes these documents accessible and reusable by storing them in an off-the-shelf knowledge management application
    c) Uses the newly designed algorithm to recognize coderivative documents

1.2.2. Research stakeholders

Capgemini

Within the ECM & UX technology practice of Capgemini the need for a custom knowledge management solution has become apparent. It is clear that the enterprise-wide applications fall

Capgemini is a global consulting company with over 83,000 employees. It is headquartered in Paris, France, and is present in more than 30 countries in the North American, European, and the Asian Pacific regions.
Its main activities are grouped into four business units, or disciplines: Consulting Services, Outsourcing Services, Technology Services and Local Professional Services.
The Technology Services discipline is divided into four market sectors: Financial Services, Products, Public and Telecom, Travel and Utilities (TTU). Within the TTU sector, there are several technology practices, one of which is the Enterprise Content Management & User Experience (ECM & UX) practice.
The core business of the ECM & UX technology practice is to provide a complete solution for their clients to be able to manage unstructured information. Often, this includes eliciting requirements and setting up a customized Enterprise Content Management (ECM) system. The client's employees need to be instructed (how) to use the ECM system, both to be able to store and retrieve information.

Delft University of Technology, founded in 1842, is the oldest, largest, and most comprehensive technical university in the Netherlands. With over 13,000 students and 2,100 scientists (including 200 professors), it is an establishment of both national importance and significant international standing.
The university comprises eight faculties: Aerospace Engineering, Applied Sciences, Architecture, Civil Engineering and Geosciences, Electrical Engineering Mathematics and Computer Science (EEMCS), Industrial Design Engineering Mechanical, Maritime and Materials Engineering, Technology Policy and Management.
The EEMCS faculty offers six different masters of two years, one of which is Media and Knowledge Engineering. The Man-Machine Interaction group in this master is focusing on intelligent multimodal systems with the view of optimal performance and satisfaction of the user. This specialisation educates students in user interface design, speech recognition,, user profiling and task management, context sensitivity and intelligent agents. Artificial intelligence techniques play a dominant role.

short in their facilities and usability. Firstly, DELIVER 2.0 is mainly designed for sharing business processes, training and best practices. This focus on the more formal documents ignores the documents that are created and updated on a day-to-day basis within a practice. Secondly, K!New is a repository system that facilitates the sharing of documents. Submitting documents to K!New is a tiresome process which involves mandatory selection of region, market, document title and abstract. Both these systems rely heavily on the employee to actively share his knowledge. Although this is of course the ideal situation, from a practical point of view this seems to be an utopia.

The objective for the ECM & UX technology practice is to have a knowledge management system in place that requires minimal interaction from its employees, while making their knowledge accessible and reusable. This would in turn lead to an increase in the main asset of Capgemini, knowledge, and to higher productivity. This last statement is based on the research done by Hoos (2007) within Capgemini, who analyzed the time spent by employees searching for knowledge that could have been contained in a knowledge mangement system.

Delft University of Technology

Delft University of Technology is always looking to support their students in developing new (and possibly ground-breaking) technology. It is important that a thesis has a solid theoretical foundation, but also that it does not loose sight of its practical application. Usefulness to the society and the business community matters greatly to Delft University of Technology. Additionaly,

instead of using the coderivative document recognition algorithm for knowledge management, Delft University of Technology could use it for fraud detection. As the availability of information increases everyday, it becomes easier for students to commit plagiarism, and harder for teachers to detect it. A software system able to recognize coderivative documents could help with such a task.

1.3. Thesis organization

This thesis consists of the following Chapters:

- **Chapter 1, Introduction.** This Chapter provides an introduction to the thesis and states the problem definition. Furthermore, the research question and the objectives of the thesis are explained
- **Chapter 2, Related Work.** This Chapter describes the technologies and algorithms related to coderivative document recognition
- **Chapter 3, Requirements, Architecture and Design of the Cayman System.** This Chapter provides the requirements for the software system that can facilitate knowledge management with coderivative document recognition. A design of the system as a whole is given, as well as detailed diagrams and descriptions of the components. An in-depth specification of the developed algorithm is dealt with in the next Chapter
- **Chapter 4, The C-CodeR Algorithm.** This Chapter describes the detailed design of the similarity algorithm developed by us and methods to evaluate it and compare it with other similarity algorithms
- **Chapter 5, Implementation of the Cayman System.** This Chapter follows up on the design and describes the implementation of the software system. The global implementation framework is given, followed by implementation details for each of the components
- **Chapter 6, Experiments.** In this Chapter we discuss the experiments we conducted and their results. The algorithm developed by us is compared with a baseline algorithm, as well as with some of the most promising algorithms we found while researching related work
- **Chapter 7, Conclusions and Recommendations.** The thesis closes with drawing conclusions on the developed algorithm and its application to the problem under consideration. We evaluate if the objectives set at the start of our thesis have been met

Chapter 2.

Related Work

In this Chapter we give an account of the work related to our research project. Before delving into designing a system and algorithm that can identify coderivative documents, we investigate related fields of research and their technologies. We found only one instance of an algorithm especially focused on coderivative document recognition, so we widened our search domain. We mainly investigated one other field that is closely related to coderivative recognition: (topically) similar document identification. Other fields we briefly looked into are revision control systems, duplicate document detection and compression techniques. We will use the term *similarity algorithm* throughout the thesis to refer to an algorithm that is (possibly) able to identify similar documents and might also be applied to coderivative document recognition. An algorithm designed especially for coderivative recognition will also be denoted by this common term, as coderivative documents are a subset of similar documents.

Similar document identification can be found in almost every search engine on the Internet, and in most search engines integrated into various applications. The basic idea of such a feature is that once you have found a document of interest, you are likely to be interested in other documents on the same topic, i. e. similar documents. A similar document identification algorithm analyzes the document under consideration, performs a search and tries to find related documents. This is a less stringent interpretation of similarity than we need for our coderivative recognition system as its objective is to retrieve documents on roughly the same topic, whereas we need to further distinguish between topically similar and coderivative documents.

A revision control system is used to keep track of all work on, and all changes in a set of files and thus allows its developers to collaborate, even if they are widely separated in space and/or time. Although developed for the purpose of managing source code of (open source) applications, these revision control systems are now also used to manage more textually orientated files, such as the possible coderivative documents we are trying to identify.

The purpose of duplicate document detection is to be able to identify (near) duplicates of documents that are created for example by faxing. Compression seems to be unrelated to coderivative document recognition, but delivers a completely non-semantic view on the subject.

This Chapter is divided into three Sections. Since most of the related work we found is focused on topically similar document identification, we devote the first two Sections of this Chapter to that area of research. The methods can roughly be divided into two families: techniques related to information retrieval which we deal with in Section 2.1, and fingerprinting techniques,

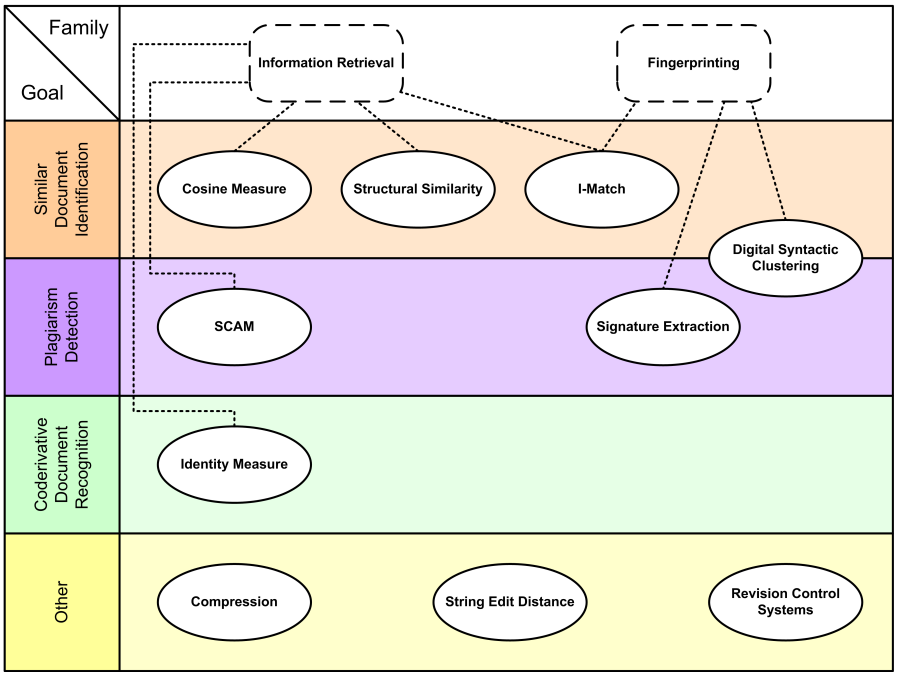

Figure 2.1: *Related work*

described in 2.2. Section 2.3 is a compilation of several other related fields and their algorithms. A graphical overview of the related work is given in Figure 2.1. Finally, in Section 2.4 we present a summary of the related work, and which algorithm we think is the most promising.

2.1. Information retrieval techniques

Information retrieval techniques analyze the textual content of documents and use this to calculate a similarity score. Underlying all these techniques is the Vector Space Model (VSM) introduced by Salton et al. (1975) where documents are represented by vectors in a multidimensional space. Each element of a document vector corresponds to a separate term (usually a word from the document). Attached to each term in the vector is a value. We use the following notions when discussing the VSM hereafter:

D A document,

N The number of documents in the collection,

n_d The number of terms in document D,

t A term in a document,

f_t The document frequency: number of documents in the collection containing term t,

$f_{d,t}$ The term frequency: the number of occurrences of term t in document D.

With these definitions, the following statistics can be calculated:

$$idf_t = \log\left(\frac{N}{f_t}\right) \tag{2.1}$$

$$w_{d,t} = f_{d,t} \times idf_t \tag{2.2}$$

, where idf_t is the inverse document frequency, and $w_{d,t}$ is the weight associated with term t in document D: the multiplication of the term frequency and inverse document frequency. In the classic vector space model a document D is represented by a vector $\mathbf{d} = [w_{d,1}, w_{d,2}, \ldots, w_{d,n}]$. In this term frequency-inverse document frequency model, each term in the vector is associated with a weight, as defined in Equation 2.2. Instead of combining these local and global parameters, a more simple model would only take the local parameter into account, setting the weight of each term equal to the number of occurrences of the term in the document.

2.1.1. Cosine measure

The cosine measure is one of the best known and most used similarity measures based on the VSM. The similarity of two documents in this model is the cosine of the angle between the two term vectors of the documents. Let **a** and **b** be the term vectors of two documents, then the cosine similarity measure is defined as:

$$\cos\theta = \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}|\times|\mathbf{b}|} = \frac{\sum_{t\in\mathbf{a}\cap\mathbf{b}} w_{a,t}\times w_{b,t}}{\sqrt{\sum_{t\in\mathbf{a}} w_{a,t}^2 \times \sum_{t\in\mathbf{b}} w_{b,t}^2}} \tag{2.3}$$

. If the document vectors are orthogonal, this results in a cosine similarity score of zero. For two documents with completely identical term vectors the similarity score is one.

2.1.2. Identity measure

The identity measures introduced by Hoad and Zobel (2003) are designed especially with the objective of identifying coderivative documents. In their paper, they develop five variations of which only the last one proves to be somewhat effective in identifying coderivative documents. This last variation of the identity measures is as follows:

$$\frac{1}{1+\log_e(1+|n_a-n_b|)} \cdot \sum_{t\in\mathbf{a}\cap\mathbf{b}} \frac{\frac{N}{f_t}}{1+|f_{a,t}-f_{b,t}|} \tag{2.4}$$

. Note that this measure does not return a similarity score between 0 and 1. Rather, a document compared with itself scores a certain maximum value, and the more similar another document is to it, the closer the score of the document pair comes to this maximum score.

2.1.3. SCAM

Shivakumar and García-Molina (1995) developed SCAM (Stanford Copy Analysis Mechanism) to detect overlap between documents. According to the authors, a shortcoming of the cosine similarity measure is that is does not work well when the magnitude of word frequencies differ significantly. To counter this, they introduce a new similarity measure that uses the relative frequency of terms and combine it with the cosine measure.

First, they define a set of terms that have similar frequency occurrences in the two documents that are under comparison. This closeness set $c(A,B)$ of two documents A and B is the set of terms for which

$$\varepsilon - \left(\frac{f_{a,t}}{f_{b,t}} - \frac{f_{b,t}}{f_{a,t}} \right) > 0$$

holds. Here, $\varepsilon = (2^+, \infty)$ is a parameter and $f_{a,t}$ is the frequency of term t in document A, as defined by the VSM. If the frequency of a term in a document is 0, then the value of the condition is undefined, and the term is not in the closeness set. The parameter ε determines the tolerance on the difference in frequency of a term and Shivakumar and García-Molina find that a value of 2.5 is appropriate in practice. Using the closeness set they are able to formulate the subset measure[1]:

$$subset(A,B) = \frac{\sum_{t \in c(A,B)} f_{a,t} \cdot f_{b,t}}{\sum_{t \in A} f_{a,t}^2}$$

. This measure is very similar to the cosine measure, but is it asymmetric. Finally, the similarity measure used by the SCAM system is defined as:

$$sim(A,B) = \max \{ subset(A,B), subset(B,A) \} \tag{2.5}$$

. The range of this measure is mapped onto $[0,1]$, so any value greater than 1 replaced by 1.

2.1.4. Structural similarity

Wan (2008) seems to be one of the few to really bring something new to the field of document comparison, incorporating the structure of a document into the equation. He acknowledges that the cosine similarity measure (among other similar measures) is a good method to detect topical similarity. The challenge, according to Wan, is to be able to further distinguish between topical identical documents. He believes that the structure of a document and the distribution of the terms over the document is a factor that would help with this. Two documents that not only share their common terms, but also their structure are more likely to be similar documents then those that have a completely different distribution of terms over the document.

While structural similarity has been studied thoroughly for semi-structured documents (e. g. HTML and XML documents) (Cruz et al., 1998; Nierman and Jagadish, 2002), the research

[1] They include term dependent parameter α_t^2 in both numerator and operator, but always set it equal to 1 in their experiments.

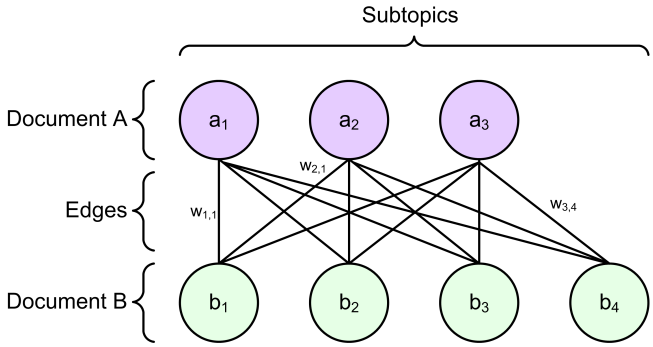

Figure 2.2: *Bipartite graph example*

done on structural similarity for ordinary documents is quite limited. Wan uses the TextTiling algorithm (Hearst, 1994) to derive the structure from a document. Hearst believes that a document "can be characterized as a sequence of subtopical discussions that occur in the context of a few main topic discussions", (Wan, 2008). The TextTiling algorithm splits the document into semantic passages, each corresponding with it's own subtopic.

To deduct the similarity between two documents Wan calculates the weighted sum of three factors: the optimal matching, text order and disturbing factor. Given two documents and their sequence of subtopics as inferred by the TextTiling algorithm, $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_m\}$, a complete bipartite graph $G = \{A, B, E\}$ can be constructed representing the structure of the two documents. The graph consists of two disjoint sets of vertices A and B, each made up of the subtopics of the corresponding documents. The set of edges E connects every vertex in the first set to every vertex in the second set.

The cosine similarity measure is evaluated for each subtopic pair between document A and B, and its value is used as weight w_{ij} for the edge e_{ij} between the subtopics in the graph. Note that this value w_{ij} is different from the value $w_{d,t}$ used in the VSM (see Section 2.1): here it refers to the result of the cosine similarity measure between a subtopic pair, whereas in the VSM it is the weight of an individual term in a document. To speed up subsequent computation, Wan uses a cutoff weight $\tau = 0.05$ and sets all weights below this value to 0. An example of a complete bipartite graph is given in Figure 2.2 (note that for readability not all the weight labels are shown).

A matching M is a subset of edges where no edge shares the same node. Intuitively, (most of the) the subtopics from the document with the fewest amount of subtopics are each matched with a different subtopic from the other document. There is a matching M that maximizes the sum of the weights of the edges contained in M. This optimal assignment of subtopics is the basis for the structural similarity measure developed by Wan (2008). Note that edges with a weight below τ are set to 0, and do not contribute to the maximum sum of the weights in the matching M. These edges are therefor never part of the matching subset of edges with maximal weight.

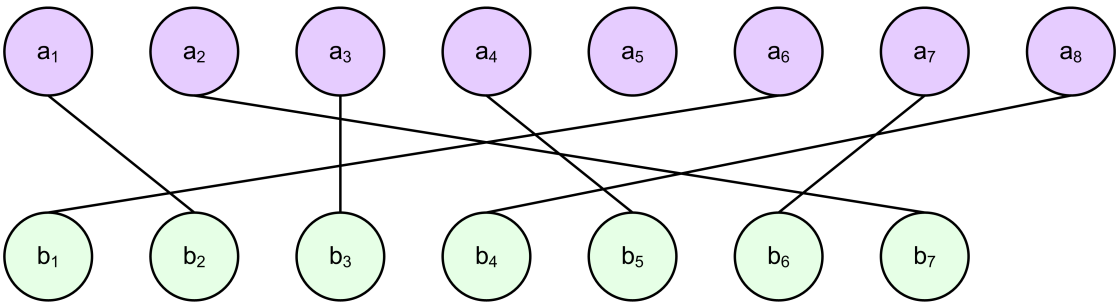

Figure 2.3: *Optimal matching example*

To be able to calculate the three factors used for the structural similarity algorithm the problem of finding the optimal matching (a matching with maximum weight) of subtopics must be solved. This problem translates to an assignment problem which can be solved by applying the Kuhn-Munkres algorithm (Kuhn, 1955; Munkres, 1957). Once the optimal matching (*OM*) of the bipartite graph has been found, the three factors can be calculated.

The optimal matching factor is the normalized sum of the weights:

$$sim_{OM}(A,B) = \frac{\sum_{w_{ij} \in OM} w_{ij}}{\min\{n,m\}} \tag{2.6}$$

. This is the main factor and its contribution to the actual similarity measure is large. It can be seen as the average cosine similarity value of the optimal matching pairs of subtopics.

The text order factor is based on the idea that for two structurally similar documents the order of the *OM* assignment of subtopics should not differ greatly from the original order of the subtopics in the documents. Wan formulates this as the longest common subsequence (LCS) problem, solvable by dynamic programming:

$$L(i,j) = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ L(i-1,j-1)+1 & i,j > 0,\ (a_i,b_j) \in OM \\ \max\{L(i,j-1), L(i-1,j)\} & i,j > 0,\ (a_i,b_j) \notin OM \end{cases}$$

, where *OM* is the optimal matching of the graph. Put more simply, $L(n,m)$ is the maximum number of non-crossing edges that form a subset of the edges in the bipartite assignment graph. For example, consider two documents $A = \{a_1, a_2, \ldots, a_8\}$ and $B = \{b_1, b_2, \ldots b_7\}$ and their optimal matching in Figure 2.3. By removing several edges we retain a subset of non-crossing edges from the graph. In Figure 2.4, the gray and interrupted lines are removed from the assignment, and the black lines remain. The number of remaining lines is 4, which is the value of the longest common subsequence. Wan (2008) uses the LCS to define the text order factor as:

$$sim_{TO}(A,B) = \frac{L(n,m)}{\min\{n,m\}} \tag{2.7}$$

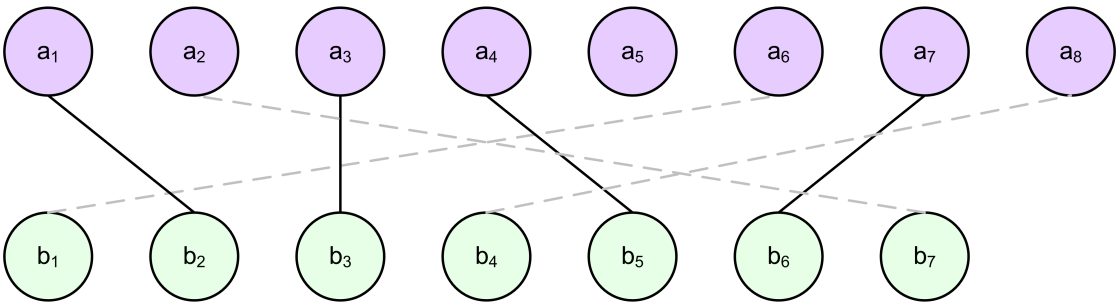

Figure 2.4: *Longest common subsequence example*

.

Finally, two (structurally) similar documents are likely to have approximately the same amount of subtopics which motivates Wan to introduce the disturbing factor. This factor translates a discrepancy in number of subtopics to a negative contribution to the similarity between documents. Wan defines it as follows:

$$sim_{DF}(A,B) = \frac{2 \cdot |OM|}{n+m} \tag{2.8}$$

, where $|OM|$ is the size (number of edges) of the optimal matching assignment. It is important to notice that the values of n and m correspond to to number of subtopics in the two documents. The size of the OM assignment of subtopics might be a lot smaller than the value of $\min\{m,n\}$, since edges with a weight of $w_{ij} < \tau$ are not in the optimal matching.

The overall structural similarity measure is a weighted combination of the three factors and is formulated as:

$$sim(A,B) = \alpha \cdot sim_{OM}(A,B) + \beta \cdot sim_{TO}(A,B) + \lambda \cdot sim_{DF}(A,B) \tag{2.9}$$

, where the sum of α, β and λ is equal to 1. Wan (2008) suggests values of 0.7, 0.2 and 0.1 respectively for the weight parameters, which are the result of tuning heuristically on a small dataset.

2.2. Fingerprinting techniques

The second family of techniques used in similar document identification is the family of fingerprinting methods. These methods are targeted more at comparing a document with a large set of documents, whereas the information retrieval techniques focus on comparing a single document pair. Because the number of documents involved is large, an efficient way to represent a document and to compare two of them is needed.

All fingerprinting techniques use a hash function to create a compact document description and use the hash values to compare documents. A hash function is a mathematical function for

transforming some kind of data into a relatively small integer, the hash value or simply hash. A hash function is typically a one-way function, meaning that it is not possible to reconstruct the input from the produced hash value. A good hash function must produce uniformly distributed hash values, that is, every hash value in the output range should be generated with roughly equal probability.

Manber (1994) was the first to explore these techniques to find similar files in a large file system. Subsequently, Brin et al. (1995) developed COPS (Copy Protection System) for detecting partial or complete overlap of documents. The fingerprinting techniques all share the same algorithmic steps. First, a document is partitioned into contiguous chunks of terms. These chunks are then filtered in a way to retain only a small portion of representative chunks. Each chuck is hashed into a relative short string of alphanumeric symbols. The set of hashes derived from a document is called a fingerprint or signature.

2.2.1. Digital Syntactic Clustering

Broder et al. (1997) developed the Digital Syntactic Clustering (DSC) method to discover whether two documents are "roughly the same". The focus of DSC is not only to detect plagiarism, but also to find similar documents in a large corpus of 30,000,000 HTML and text documents. They first lexically analyze each document, removing formatting, capitalization, etc. and this results in a sequence of terms. With each document d, they associate a set of subsequences of terms $S(d,w)$, where w is the length of the subsequences (they set w equal to 10). Broder et al. defines this contiguous subsequence of length w as a *shingle*. They then define the *w-shingling* of document d to be the set of all unique contiguous shingles of length w. These shingles are converted into fingerprints with a 40 bit hash function, based on Rabin fingerprints (Rabin, 1981). The set of retained shingles is determined by the modulus function: each fingerprint value f of each single for which $f \bmod m = 0$ holds, with m set to 25, is part of the filtered fingerprint set (essentially keeping every twenty fifth shingle). The similarity between two documents A and B is defined as:

$$sim(A,B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|} \tag{2.10}$$

, where $|A|$ is the size of the set A and $S(A)$ is the set of fingerprints of the shingles for document A. This results in a value between 0 and 1, with the former indicating a no similarity at all, and the latter a similarity of 100%.

To simplify similarity detection with the shingling technique and make it more efficient they generate *super shingles* (DSC-SS). A super single is constructed by sorting the shingles and then shingling the shingles. The fingerprint of a document is defined by the set of super shingles. Choosing the number of shingles to go into an super single is a non-trivial problem. Detecting similarity between two documents simplifies to finding a single common super shingle. They acknowledge that super shingles are not as flexible or accurate as the regular shingling technique, and that they do not work well with short documents.

Since a pairwise comparison of the 30,000,000 documents is infeasible (it requires $O\left(10^{15}\right)$ comparisons), they divide the dataset into clusters, compute the similarity per cluster and merge the results. Even with this approach it takes them almost 11 CPU days. The real performance of the DSC algorithm is unclear, since they lack a truth list containing which documents are similar and which aren't.

2.2.2. I-Match

I-Match, or ITT-Match in full, is introduced by Chowdhury et al. (2002) and is a combination between a fingerprinting technique and the collection statistics from the VSM. The author's main motivation for developing I-Match as an alternative to DSC, is that this algorithm has efficiency issues, and is ineffective when working with small documents. Although DSC tries to boost performance by keeping only every twenty-fifth fingerprint, the reported runtime for a collection of thirty million documents took over 10 CPU days to process (Broder et al., 1997). Furthermore, Abdur Chowdhury et al. state that this random pruning of fingerprints reduces the accuracy. Instead, they use the inverse document frequency idf_t as defined in Equation 2.1 to retain only representative terms. They filter a document by removing formatting, capitalization, etc. and retaining only terms with certain inverse document frequency values. They adopt two approaches, filtering with percentages based on inverse document frequency of the terms in the document itself, and filtering with a value threshold based on inverse document frequency normalized over the document collection to the $[0,1]$ interval. With the first method, they filter with percentages ranging from 10% to 90% with steps of 10%, and keep only the lower $X\%$ or the upper $X\%$. In addition to this, they keep or remove the middle 20%, 40%, 60% or 80%. The second method uses a threshold value for the normalized inverse document frequency ranging from 0.5 to 0.9, with steps of 0.1.

The filtered terms are sorted in ascending order, and of this ordered tree a unique fingerprint is computed with the SHA1 digest function. This fingerprint is used to compare two documents. If they have the same fingerprint, the documents are similar.

Chowdhury et al. (2002) conclude that I-Match performs better on large web collections and does not neglect small documents. I-Match runs five to six times faster than the supershingle variant of DSC. The filtering approach with normalized inverse document frequencies, keeping only terms with a inverse document frequency above 0.1, performs best at finding duplicate documents.

2.2.3. Signature Extraction

Signature extraction is again a variation on the fingerprinting technique, and it's main application is the detection of plagiarism. The main innovation of Finkel et al. (2002) is in the method of filtering the contiguous chunks. Instead of splitting a document into overlapping chunks of a fixed length, they adopt the hashed-breakpoint chunking method (Shivakumar and García-Molina, 1996). They compute a numeric value f for each term with a simple hash function and

test for which terms the equation $f \bmod c = 0$ holds, with c equal to 10. Any term for which the previous equation is valid, ends the current chunk. Provided that the hash function produces uniformly distributed numbers, the average chunk size will be c terms.

To reduce the number of chunks, Finkel et al. reason that a short chunk is not very representative of a document and that if two documents share a short chunk there will be no reason to suspect plagiarism. A long chunk would be very representative, but a clever plagiarizer would never copy a long piece of text word for word. They propose two methods of discarding the shortest and longest chunks:

- *Sqrt*: Retain $\lceil \sqrt{n} \rceil$ chunks whose lengths L are closest to m
- *Variance*: Retain those chunks such that $|L - m| \leqq b \cdot s$. If necessary, increase b until $\sqrt{n}$ chunks are selected. Start with $b = 0.1$

where L is the length of a chunk, n is the number of chunks, m is the median chunk size, s is the standard deviation of the chunk size and b is a constant. After the representative chunks are selected, the MD5 algorithm is used to convert them to a 128-bit numeric fingerprint. They authors argue that keeping only the leading 40 bits keeps the storage requirement low while it still ensures a small enough chance on a false positive (1 in $16^{10} \cong 10^{12}$).

They suggest three similarity measures: the asymmetric, symmetric and global similarity measure, but in their experiments they mainly use the symmetric similarity measure. For two documents A and B, this is defined as:

$$sim(A,B) = \frac{|d(A) \cap d(B)|}{|d(A)| + |d(B)|} \tag{2.11}$$

, where $d(A)$ is the set of fingerprints of document A. Note that although this looks similar to the similarity measure of the DSC method, defined in Equation 2.10, there is a subtle difference since in general $|A| + |B| \neq |A \cup B|$ (this is only valid when both sets are disjoint ($A \cap B = \varnothing$), in which case the similarity score is 0 because the numerator is 0). In fact, even if there is a maximum overlap with $A = B = A \cap B$, it follows from Equation 2.11 that the maximum value of the similarity score is 0.5. Since they mention that a similarity value of 1 indicates complete overlap, we believe the formula to be an error (at least in notation) of the authors.

In their experiments, they find that the Sqrt method of filtering the chunks does not store enough chunks for short files. They prefer the variance method and use it in their main experiment with a corpus of 2591 RFC[1] documents. Since they lack a truth list, evaluation of the performance of the signature extraction method is difficult. They are able to identify similar documents where one RFC is an update of the other. However, coderivatives or versioned documents are not contained in the corpus.

[1] `www.rfc-editor.org/rfc.html`

2.3. Other techniques

2.3.1. String edit distance

To focus of Lopresti (1999) is duplicate document detection, where the documents are photocopies, formatted in different ways, or faxed documents. He acknowledges that the document might have been changed by changes in font, minor alterations due to editing, crossing out sections or adding handwritten comments. He uses a commercial Optical Character Recognition (OCR) package to convert the documents into digital text, and proceeds with this text to try and detect duplicates. Lopresti (1999) considers a document image (scanned with OCR) to be a series of lines or a string of strings. He distinguishes between four different duplicate classes: full layout (same content, same layout), full content (same content, different layout), partial layout (shared content, same layout) and partial content (shared content, different layout). He computes the similarity between documents with four variants of the edit distance (Levenshtein, 1966; Wagner and Fischer, 1974).

In his experiments he uses a database consisting of 1000 news articles from Usenet joined with a set differently degraded versions of the same query document: faxing, excessively light or dark copying, third generation copying or handwritten annotations that completely obscure five of the lines on the page. These pages are scanned and converted to digital text with OCR. Each of the variants of the edit distance performs best with one of the duplicate classes, with one of them being the most general and applicable to all the classes. However, for special cases of duplicate detection, the other three might also be useful, according to Lopresti.

2.3.2. Compression

In addition to the signature extraction technique (2.2.3), Finkel et al. (2002) also introduces a compression technique to measure document similarity. They define the following similarity for two documents A and B:

$$sim(A,B) = 2 - \frac{2 \cdot |compress(A \boxplus B)|}{|compress(A)| + |compress(A)|} \tag{2.12}$$

, where $compress(A)$ is a compression algorithm, $A \boxplus B$ is document A concatenated with document B and $|compress(A)|$ is the size of the compressed document. A well performing compression algorithm will exploit the overlap between two documents, and not waste much extra space to compress this if the two documents are concatenated. An ideal compression algorithm will compress $A \boxplus A$ to about the same size as A, resulting in a similarity of 1. With two very dissimilar documents the compression algorithm will not be able to compress the concatenation of the two documents any more than to the sum of their individual compressed sizes, leading to a similarity of 0.

They experimented with several compression algorithms, including *gzip* (Deutsch, 1996), *bipz2* (Seward) and *ppm** (Cleary and Teahan, 1997). They conclude that gzip has problems

with large files and that ppm*, although the best compression algorithm, is very slow. The overall winner is the bzip2 compression algorithm. They note however, that this method is not suitable to compare a query document against a database of many documents, since this would be far too costly with respect to computation time.

2.3.3. Revision control systems

Revision control systems are mainly used by programmers to keep track of changes in their source code. This is especially useful when multiple people collaborate on a single software project. A revision control system acts as a central repository where the source code is stored. A user can check out a single file or folder, or even a whole project, and work on it locally. After he has finished editing the checked out files, he can submit his new version of the source code to the revision control system. This system then performs an analysis on the submitted version and the one in its repository and stores the differences between them. If another user also checked out the same source code from the repository and submits it after the work of the first user has been submitted, the revision control system is still able to recognize and store the differences between them. Two widely known revision control systems are CVS[1] and Subversion[2].

All revision control system share the same approach which has two major drawbacks making them less suitable for coderivative document recognition. Firstly, which file a user checks out is explicitly stored both on the server side and client side. Therefor, a revision control system does not focus on matching a submitted file with a possible coderivative, but rather on efficiently finding the differences between two files that are known to be coderivatives. Secondly, the difference finding algorithm of most revision control systems is line-based. This makes sense when maintaining source code, since these are typically short lines, which are not subject to change themselves, but are added or removed as a whole. With a normal textual document, a line in the source of the document does not correspond to a line on-screen or on a printed version of the document. A line is ended with a carriage return, i. e. pressing the enter key on the keyboard. This corresponds with a paragraph when the text is formatted on-screen or printed. Since revision control systems are not able to detect changes within lines, changing a single letter in a paragraph produces an unmatchable line and would greatly lower the similarity value produced by a derived similarity algorithm.

2.4. Overview and algorithm selection

Both the information retrieval and fingerprinting fields of research are populated with techniques that bare close resemblance to each other. The basis of the information retrieval field is the VSM and its cosine similarity measure. Virtually all methods in this field use the term vectors of the

[1] `ximbiot.com/cvs`
[2] `subversion.tigris.org`

VSM as input and differ for example in the weights of the terms, the precise set of terms to use, the mathematical formula calculating the similarity score, etc.

All fingerprinting techniques divide the text of a document into chunks and compute the hash value for each chunk. The more overlap there is between the sets of chunks of two documents, the higher the similarity score. The algorithms differ in the method of dividing a text into chunks, the selection of chunks to represent a document with, the mathematical formula used to calculate the similarity score, etc.

In our opinion the information retrieval techniques are a more intelligent approach and more suitable to apply to coderivative document recognition. These techniques try to incorporate the semantics of a document in their operation, instead of reducing it to plain hash values. The fingerprinting techniques purposely throw away much information that is contained in a document. We believe that to recognize coderivative documents, this line of approach is to coarse and will not be as successful as a technique from the information retrieval field.

The algorithm that strikes us as most promising is the one developed by Wan (2008). Not only taking into account the frequency of terms in a document, but also the order in which they appear, could provide to be a much more powerful instrument to recognize coderivative documents. Wan himself already states that he developed this algorithm with the objective to be able to further distinguish between topically similar documents. This comes close to our objective, since our interpretation of similarity is more stringent than plain topical similarity. We will follow in Wan's footsteps and develop our own structural similarity algorithm, which will be introduced in Chapter 4. Throughout the rest of this thesis we will refer to the original structural similarity algorithm developed by Wan (2008) as the TextTiling algorithm, and we will denote the set of algorithms that take the structure of a document into account by structural similarity algorithms.

Chapter 3.

Requirements, Architecture and Design of the Cayman System

This Chapter describes the requirements, architecture and design of a software system that is able to provide automatic document collection, knowledge management and coderivative document recognition. Our software system is named *Cayman*. The first part of this name, *cay*, is roughly derived from the phonetic representation of the letter *k*, which in turn stands for knowledge. The latter part the name, *man*, stands for management, resulting in capturing the whole of knowledge management in one (meaningful) word.

In the first parts of this Chapter the requirements and architecture for the software system are set out. This is followed by a detailed design of the system and its components.

3.1. Requirements

The requirements for the software system can be split into three groups: the project constraints, the functional requirements and the non-functional requirements. The functional requirements are separated into three parts. Each of these parts coincides with a function that the system must provide: automatic document collection, knowledge management and coderivative document recognition. The requirements were elicitated through several meetings with the self-formed focus group within the ECM & UX technology of practice of Capgemini. This group was formed to find a solution for the (lack of) knowledge management of the practice.

3.1.1. Constraints

This Section describes the restrictions placed on the project beforehand.

- The research project has to be completed in 8 months
- As it is experimental research, the time and money investment should be kept low. The use of open source applications is encouraged
- The Cayman system has to integrate with the existing computers and server at Capgemini

- The source code of our system has to be made publicly available (required by Delft University of Technology)

3.1.2. Functional requirements

Automatic document collection

The system must gather new and changed documents from the employees periodically and submit them to the knowledge management system.

Knowledge management

The software system must store all received documents and make them accessible to the employees. Apart from receiving the automatically gathered documents, users must be able to manually submit documents to the knowledge management system. The system must provide means to query the stored documents in order to facilitate the reuse of knowledge.

Coderivative document recognition

The system must be able to recognize coderivative documents from non-coderivative ones. It must be able to identify completely identical documents and not store these in the knowledge management system. If a document is received that is a coderivative of a document that is already in the knowledge management system, the system must group these documents together in the knowledge management system. The system must be able to recognize coderivative documents across different formats and renditions.

If the system cannot with reasonable certainty identify a coderivative document pair but has a strong suspicion, this must be made clear to the user for him to detect possible coderivatives.

3.1.3. Non-Functional requirements

Performance

The automatic gathering of documents must not hinder the employee in doing his work. The system must be active in the background and require minimal user interaction. In the case that the employee experiences the software system as having a negative impact on the computer's response time, it must be possible to temporarily suspend the system. The submission of documents to the knowledge management system must only occur when a connection with sufficient throughput is available.

The knowledge management system must be able to receive documents from multiple employees simultaneously. The primary function offered to the employees, providing a search interface through which the stored documents are accessible, must not suffer from the submission and processing of collected documents.

The recognition of coderivative documents must be performed quickly, so that the stream of collected documents can be stored in the knowledge management system without much delay.

Reliability

The automatic gathering of documents must be reliable and never compromise the actual documents of the employee. The knowledge management system must be a stable software application that is able to handle a large volume of documents and users.

Security

The privacy of the employees is part of the security of the system. Since the computers provided by Capgemini to its employees remain the property of Capgemini and may only be used for professional work, we assume that there are no private documents present that may be collected. However, a minimal document selection or exclusion functionality must be built into the software system to provide the employee with some choice.

The documents must only be sent over a secure connection to the knowledge management system in order to guarantee that these do not end up in the wrong hands. Similarly, the knowledge management system must only be available to employees of Capgemini.

Usability

The employee must have access to a simple graphical interface where he can see the status of the automatic collection of documents. Additionally, this interface provides the employee with the minimal interaction tools, for example to interrupt the document search or to change certain settings. The standard user interface design approach is user-centered design. The needs, wants and limitations of the end user of an interface are given extensive attention at each stage of the design process. As we are designing a completely new software system with no specific design assigment, the user and his needs, wants and limitations are unclear. Instead, we put ourselves into the position of the end user and verify the resulting design by having it reviewed by various employees within the ECM & UX practice.

The knowledge management system must provide a fully functional and complete graphical user interface with the features that are commonly used within the area of knowledge management systems. These include, but are not limited to: add documents to the system, add meta-data to the documents, provide advanced search options, retrieve and update documents. The (possible) customization of the design of this application is out of scope for this research project, as our focus lies more on the algorithmic background. The standard design of an off-the-shelf knowledge management system is used.

3.2. Architecture

The Cayman system can be viewed as consisting of three subsystems, each responsible for a core task. The automatic document collection function is fulfilled by a prototype application we design and implement. The knowledge management functionality is provided by an off-the-shelf enterprise content management (ECM) system. Finally, the third developed subsystem is responsible for the recognition of coderivative documents. These three subsystems and their interactions are displayed in Figure 3.1. This setup roughly coincides with the client-server architecture, where the documents are gathered by a client and send to a server which processes these documents. Other (web) clients may then query the server in order to retrieve documents from the knowledge management system.

3.2.1. Communication

Since the level of communication between the knowledge management subsystem and the coderivative recognition subsystem is high, we choose to position these on the same host. The communication between the clients and the server uses the HTTP 1.1[1] protocol as a carrier, chosen for its simplicity and ubiquity. Most programming languages have readily available HTTP libraries or complete clients, and most ECM systems provide access to their functionality by means of web services. The submission of documents from the client to the server is done by sending a simple HTTP POST request with the document file itself and its meta-data embodied in the request. If the knowledge management system accepts a HTTP POST request (with a web service for example), it can directly process the request. If not, the server might need an intermediate application that translates the request into a form understandable by the knowledge management system.

For retrieving documents the method of communication between the client and the server is undefined, as it depends on the implementation provided by the knowledge management system. However, virtually every ECM system provides access through a web client such as a browser.

3.2.2. Platform

Almost all computers provided by Capgemini to their employees are based on the Intel x86 platform and use either Microsoft Windows XP or Vista as operating system. Some more design oriented employees use an Apple computer with either Mac OS or Microsoft Windows. Although it is unlikely that the choice of platform or operating system will change radically, we aim to be as versatile as possible and try to develop a cross-platform document collection subsystem.

The platform and operating system of the server is restricted by the combinations supported by the implementation of the knowledge management system. We align our platform choice

[1] www.w3.org/Protocols/rfc2616/rfc2616.html

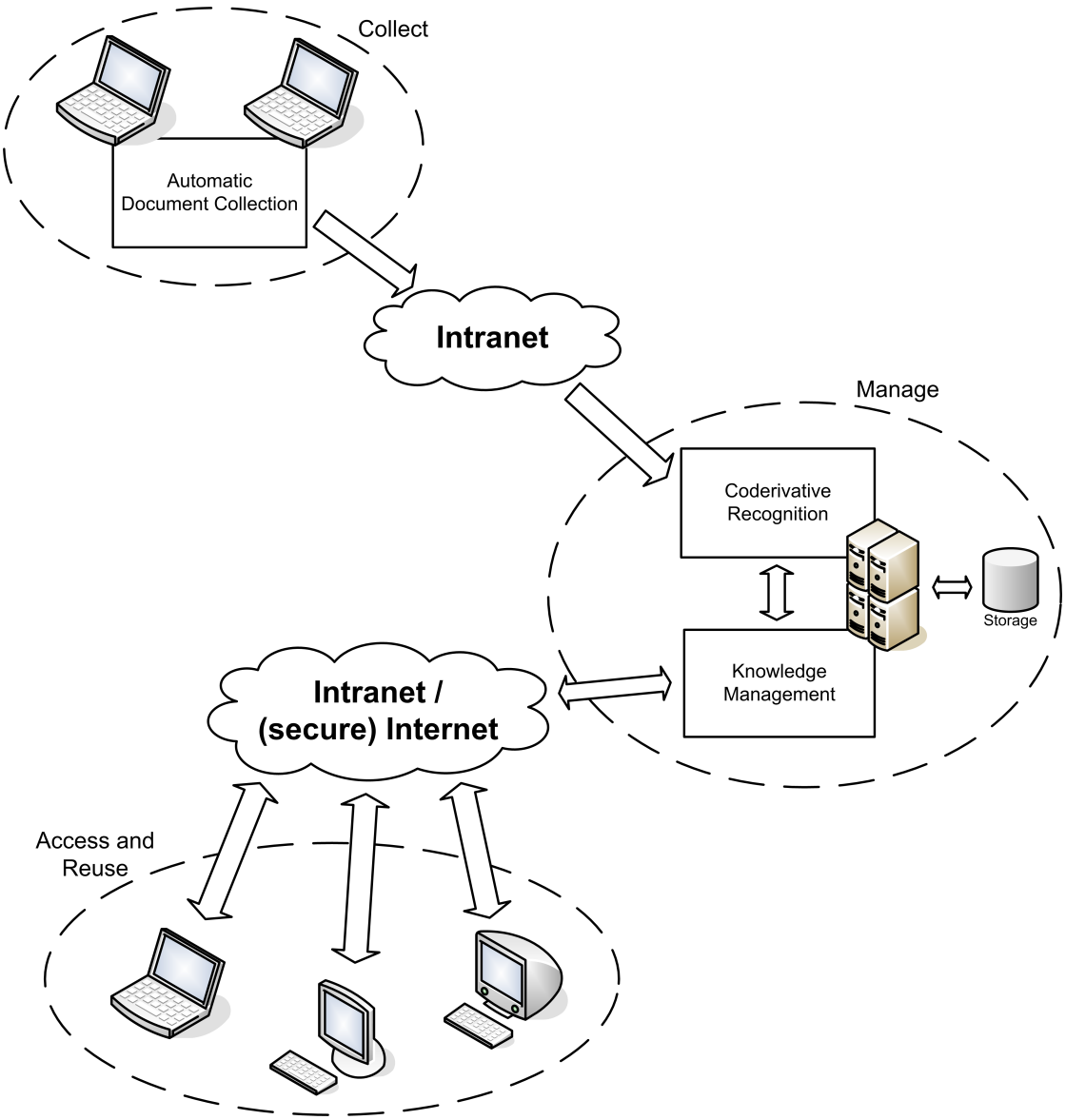

Figure 3.1: *Architecture of the Cayman system*

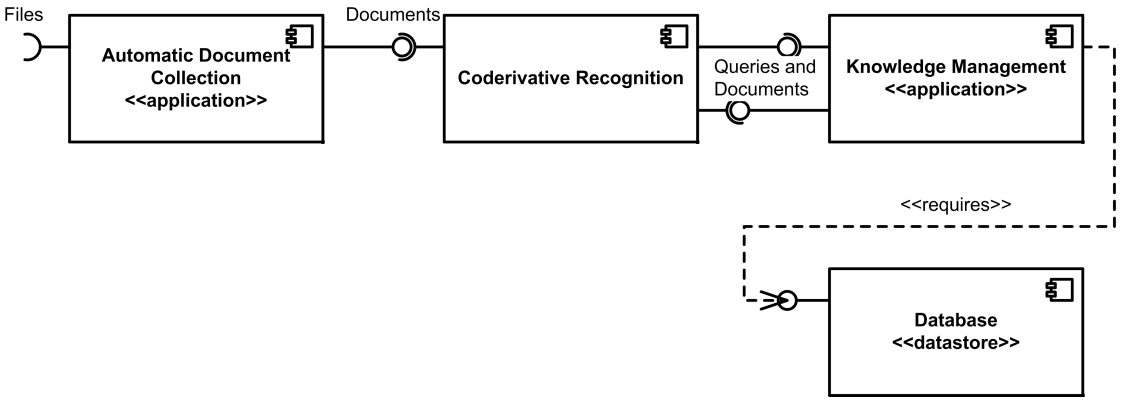

Figure 3.2: *Component diagram of the Cayman system*

for the coderivative recognition subsystem with the restrictions set by the ECM software and hardware.

3.3. Design

The design of the Cayman system can be split into four components, see Figure 3.2. In Section 3.3.1 we discuss the detailed design of the component responsible for automatically collecting the documents. The coderivative recognition and knowledge management components will both be described in Section 3.3.2. The fourth component is trivial and requires no explanation from us. Rather, it is incorporated in the implementation of the knowledge management system.

3.3.1. Automatic document collection

We choose the name *Cayman Collector* to apply to the prototype application fulfilling the function of automatic document collection. In order for the application to be as unobtrusive as possible, the graphical user interface (GUI) is optional and need not be present at all time. The program periodically searches the hard disk drive of the computer for new or changed documents. Detecting these documents is simply done by comparing the last modified attribute of the file to the last time the application performed a search. The Cayman Collector gathers documents of four file formats: Microsoft Word 2003, Microsoft Word 2007, Adobe PDF and plain text. These are file formats that are most commonly used to produce text based documents in Capgemini (as well as in most other companies). Since an employee might very well be located at a client's company for a week or longer (without a secure or fast Internet connection to the Capgemini network), the application only starts its periodic search if it detects the employee is connected again to the network of Capgemini. The application has a graphical user interface where the employee can view the state of the application, suspend the searching or sending of

Name	Type	Value
dir	String	The full path of the folder where the file was found
lastModified	long	The time this file was last modified, measured in milliseconds since 00:00:00 GMT, January 1, 1970
hostname	String	The hostname of the computer
username	String	The username of the person logged in on the computer
file	File	The filename and actual file

Table 3.1: *Fields of the HTTP POST request to send documents*

documents or change certain settings. These settings include the address of the server, what folder to search on the hard disk drive, and whether to exclude certain private files.

The documents are sent to the server with an HTTP POST request. An overview of the required fields of the request is given in Table 3.1. If an error occurs during sending the documents (the employee might be disconnected from the Capgemini network), the application caches the search result for a limited period of time. If the application is able to reconnect to the server within this time period, it continues to send documents where it left off. If the application cannot reconnect to the network in time the cached search result is cleared and a new search is started upon reconnection.

Zooming in on the technical design of the component under consideration, we present the composite structure diagram in Figure 3.3. Since this is a prototype application, following the exact guidelines of software design is not necessary. Instead, opting for a quicker development process, we use a combination of the Model-View-Controller and Observer-Observable design patterns. The view and controller are combined into one and the observer is allowed to interact with the observable by passing commands from the GUI classes. The central Cayman Collector class is responsible the main functionality, and for realizing the incoming and outgoing interfaces. This leaves the Document class, which is a container for a gathered document and its meta-data. This is depicted in more detail in the class diagram in Figure 3.4. Finally, a state diagram (see Figure 3.5) is useful to understand the internal workings of main class.

3.3.2. Knowledge management

Since we decided that the knowledge management and coderivative recognition subsystems reside on the same host, it makes sense to see how these two can be integrated. The advantage of embedding the coderivative recognition subsystem into an existing ECM application is that

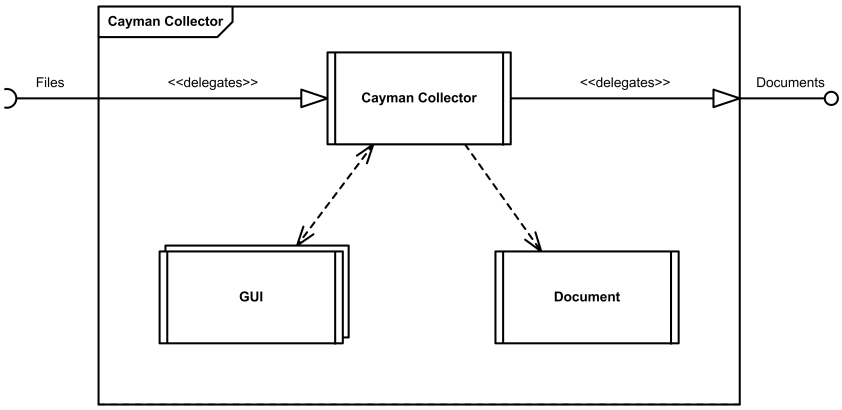

Figure 3.3: *Composite structure diagram of the Cayman Collector*

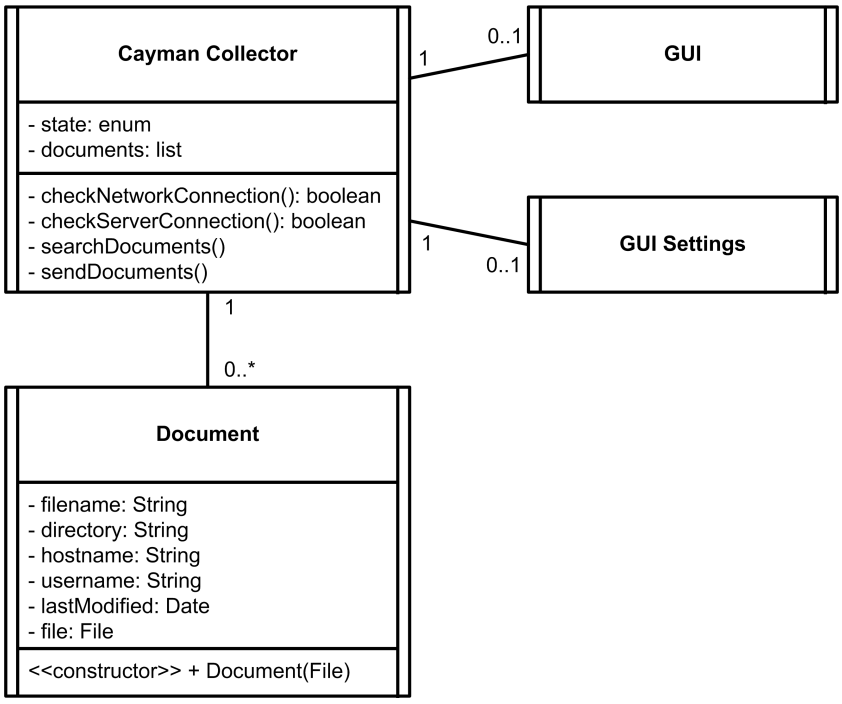

Figure 3.4: *Class diagram of the Cayman Collector*

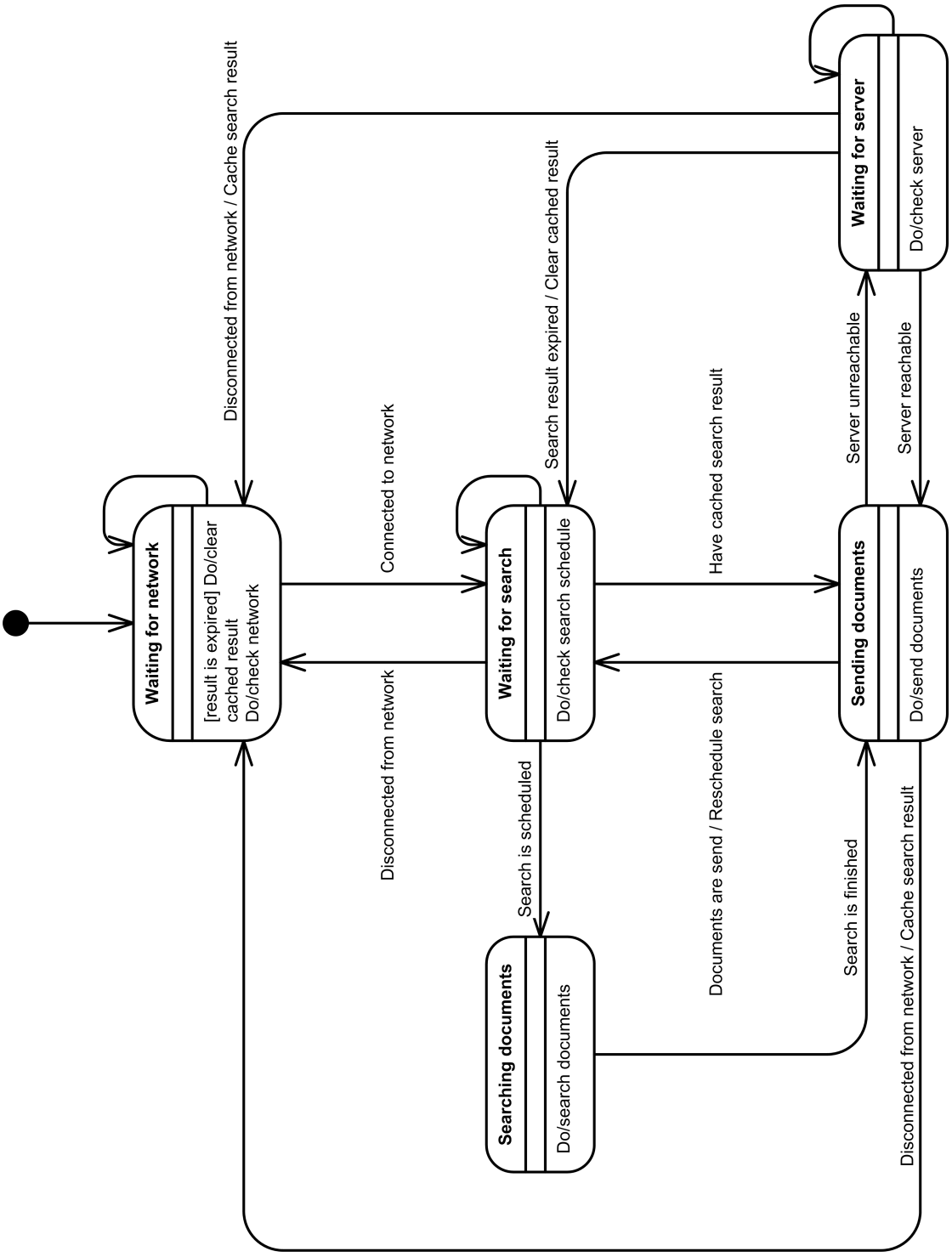

Figure 3.5: *State diagram of the Cayman Collector*

we are able to reuse core functionality of the application. This relieves us from setting up our own framework to support the recognition of coderivative documents. An example of the functionality we might be able to reuse is transforming documents from different formats into plain text.

We need to make a choice which readily available ECM application will realize the knowledge management functionality of our system. A preceding research project within the ECM & UX technology practice of Capgemini investigated the requirements for such an application. Hoos (2007) researched the functional specifications by holding interviews and workshops with the stakeholders. He compared three ECM applications: Alfresco, Documentum by EMC[2] and Microsoft Sharepoint. Matching features with requirements he concluded that Alfresco or Sharepoint are the best options.

Our choice for ECM application to use as knowledge management system is Alfresco. This software application meets all the requirements set by the ECM & UX technology of practice. Additionally, it is open source software, licensed under the GNU General Public License v.2.0[1]. The usage of open source software provides a number of advantages.

1. **The availability of the source code and the right to modify it.** This gives us full access to the application, and enables us to fully integrate the coderivative recognition subsystem with it

2. **The right to redistribute modifications and improvements to the code.** The right to redistribute our code ensures us of being able to test and possibly utilize the Cayman system in practice

3. **The low cost of ownership.** Since our project involves experimental research, funding is difficult to acquire. A low cost software application enables us to fully test and integrate our Cayman system with a widely used ECM application

4. **The active development community.** Our research involves developing custom extensions to the software application, and an active development community can aid us in doing so with its collective knowledge

Alfresco

Our subsystem is intertwined with Alfresco, which is why we proceed to simultaneously explain the basic elements of Alfresco and our extensions to them.

Firstly, there is the content model, which defines the entities within the Alfresco knowledge management system. The meta-model (the model for the content model) supports two primary constructs: the content type and the content aspect. Both provide the ability to describe a specific content structure including properties (meta-data) and associations to other content. Object-oriented constructs such as inheritance are also supported. The difference between a content

[1] `www.gnu.org/licenses/old-licenses/gpl-2.0.txt`

type and a content aspect is that content can only be of one content type. This describes the fundamental structure of the content: the properties and associations. The core content types of Alfresco are called `cm:folder` and `cm:content`. A document is stored within a content type of the type `cm:content`, and documents can be organized into a hierarchical structure with the `cm:folder` content type. Any number of content aspects may be applied to a single piece of content. Content aspects are used to add capabilities to a content type. Aspects can alleviate the issue where all capabilities are dumped onto a root content type.

A built-in aspect of Alfresco is the `cm:versionable` aspect. We use this aspect to be able to group coderivative documents together. The aspect adds a simple form of version management to a content type: both the properties and the associated content itself (i. e. the document) are subject to versioning and will remain accessible in the version history of the content type when it is updated. We realize that we cannot be sure that the gathered coderivative document is actually more recent than the document in stored in Alfresco and therefor cannot guarantee that the most recent version of a document is always at the top of the version history. Setting versions of a document in chronological order is not a trivial problem and if multiple authors edit a document creating a linear version history it might not even be possible.

We extend the content model of Alfresco with four aspects to suit our needs.

Aspect: `cay:filenameable`. This aspect is necessary for the most simple form of coderivative recognition, based solely on the file name. This aspect adds a property with the name `cay:filename`. In this property the original filename of the document is stored

Aspect: `cay:hashable`. The `cay:hashable` aspect adds one property: `cay:contenthash`. A unique hash value of the text contents of the document is stored in this field. This enables the system to quickly identify textual identical documents within the knowledge management system

Aspect: `cay:matchable`. This aspect adds an association to other content types. If a suspected coderivative document pair is identified by the system, but it cannot with reasonable certainty say that it is in fact one, an association between the two documents is made. This enables the user to see possible coderivative pairs and make adjustments if necessary

Aspect: `cay:gatherable`. This aspect adds two properties: `cay:dir` and `cay:hostname`. The aspect is applied to any document that is automatically gathered by the Cayman Collector subsystem. The original folder location on the hard disk drive and the hostname of the originating computer are saved in the properties

An overview of the extended Alfresco content model is given in Figure 3.6.

Secondly, Alfresco provides the functionality to apply an action to an instance of a content type. An action is a unit of work that can have parameters and conditions associated with it. Actions can be manually applied to a content item or scheduled to run (periodically) at a certain time. Closely related are rules that can be applied to folders and are executed when an instance

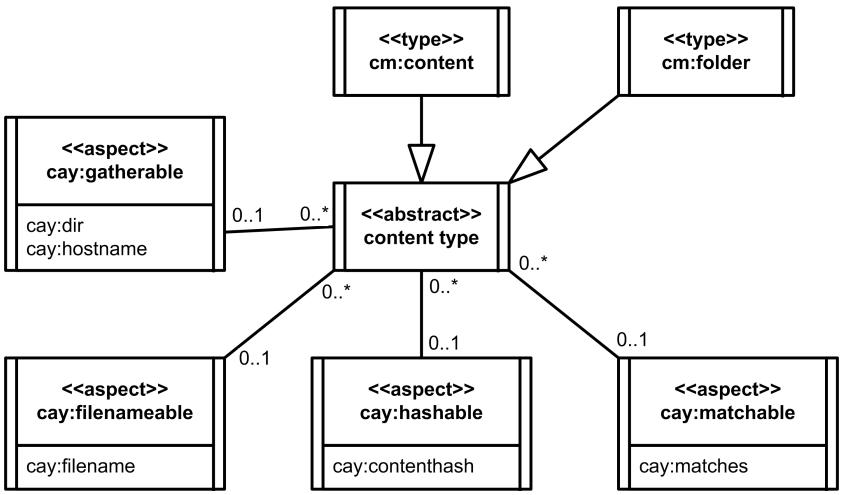

Figure 3.6: *Content model of the Alfresco extension*

of a content type enters, leaves or is updated in the folder. We add four actions to Alfresco, extending the functionality.

Action: **`ExtractLanguage.`** When running the action, Alfresco will determine the language of the contents of the document. It recognizes the major European languages. If the document does not have the `cm:localized` Alfresco aspect, it is added to it and the determined locale information is saved to the meta-data of the document

Action: **`GuessMimetype.`** This action enables Alfresco to determine the mimetype and encoding of a document and subsequently to save it to the properties of the content type

Action: **`CalculateHash.`** This action calculates a unique hash value of the text contents of the document and saves it to the properties of the content type

Action: **`MoveFailedRecognition.`** If an error occurs during the coderivative recognition process, then this action moves the document to a specific folder for manual inspection

Action: **`RecognizeCoderivatives.`** This is the core action of the Cayman system. When applied to a document, Alfresco goes through a number of steps to determine if the document under consideration has a coderivative document that is already present in the Alfresco knowledge management system. These steps are described in detail below.

1. If the content type has the `cay:hashable` aspect, the knowledge management system is queried for a document with the same hash value as the current document. If a document is found that has the same mimetype (in either the current version, or a previous version), the current document is discarded and not added to the knowledge

management system. If a document is found that does not have the same mimetype, then the current document is added to the knowledge management system as a new version of this document.
If no document is found or the document does not have the `cay:hashable` aspect, we proceed to the next step

2. The knowledge management system is queried for a document with the same filename (stored in the `cay:filename` property). For each returned document the similarity score between the text of the current document and the found document is determined with the similarity algorithm. If this score is above a certain threshold value the document is added to the list of coderivative documents. This list is then sorted by similarity score, and the current document is added as a new version of the best scoring document. A `cay:matchable` association is created between the top document on the list and the other documents on the list.
If no document is found from which the similarity score is above the threshold value we proceed to step 3

3. This step is similar to the previous step, with the exception that the knowledge management system is not queried for a document with exactly the same filename, but instead with a filtered filename. The rest of this step is identical to step 2, and if no suitable document is found we proceed to step 4. The filter applied to the original filename replaces several character sequences with a wildcard

   - Date values are recognized when in the following format: *YYYY-MM-DD*, *YY-MM-DD*, *DD-MM-YYYY*, with either a - or _ or nothing at all as separator between the day, month and year values
   - The characters *v*, *vs*, *versie* or *version* followed by a (decimal) number
   - The words *alpha*, *alfa*, *beta*, *final*, *old*, *new*, *oud*, *nieuw*
   - The words *copy of* and *kopie van*
   - A digit at the end of the filename
   - The extension of the filename

4. In the fourth we first do a basic keyword query. We select the top ten most frequent keywords from the current document (ignoring stop words, i. e. words like *a*, *the*, *are*, *I*), and query the knowledge management system for documents with these keywords. The underlying assumption is that a coderivative of the current document will at least share a large percentage of the most frequent words. For the first ten documents of the search results the similarity score is determined. Like in the two previous steps we add all the documents with a score above the threshold value to the coderivative list. This list is sorted, and the current document is added as a new version to the document with the highest similarity score. A `cay:matchable` as-

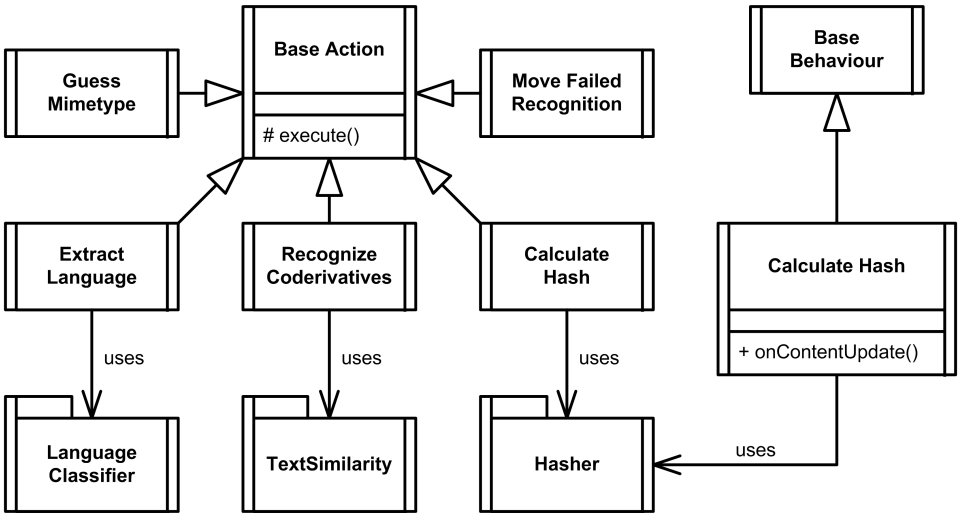

Figure 3.7: *Class diagram of the actions and behavior of the Alfresco extension*

sociation is created between the top document on the list and the other documents on the list

5. If all of the previous steps did not result in finding a coderivative of the current document, then the document is added as a new document to the knowledge management system

Thirdly, we extend Alfresco with one custom behavior. A behavior is a unit of work that can be attached to a certain content type and event. We add the `CalculateHash` behavior to automatically recalculate the unique hash value every time a document is updated. To achieve this, we attach the behavior to the content type `cm:content` and to the event `onContentUpdate`. Before actually calculating the hash value, we first verify that the content type has the `hashable` aspect, and only proceed if this is the case. A class diagram of the actions and the behavior designed by us extending Alfresco is depicted in Figure 3.7.

The last addition to Alfresco is a web script that is able to handle a HTTP POST request send by the Cayman Collector containing a document. This web script creates a new content type item in a specific folder in the Alfresco knowledge management system containing the received document. The meta-data values of the document described in Table 3.1 are stored in the properties of the content type.

Finally, we provide a composite structure diagram in Figure 3.8 to show how the various components are related to each other. The incoming HTTP POST request containing a document is handled by the web script and the document is placed in a folder in the knowledge management system. A scheduled action then proceeds to process the documents by calling four additional

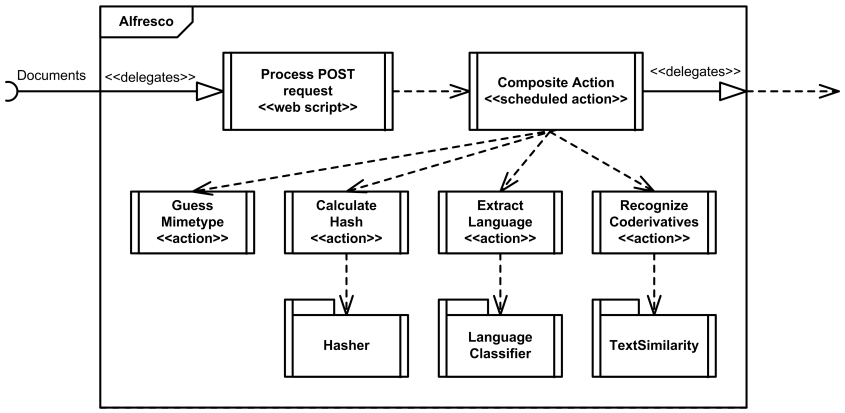

Figure 3.8: *Composite structure diagram of the Alfresco extension*

actions. Firstly, the mimetype of the document is determined and the unique hash value for the document is calculated. The language is extracted and finally the document is subjected to the coderivative recognition action. After the algorithm has finished, the document is either added as a new version to its coderivative match, or placed in the knowledge management system as a new document. If an unexpected error occurs during the coderivative document recognition, the document is moved to a specific folder for manual inspection.

The core of our research is of course the similarity algorithm that calculates a similarity score between two documents. We have placed this algorithm in its own package to be able to use it for testing and evaluation purposes without the burden of the Alfresco application. The content of this package is depicted by the class diagram in Figure 3.9. In addition to our own similarity algorithm (C-CodeR) we implement six of the most promising algorithms we came across during our research into related work. All similarity classes inherit from a common base class and interface. The two structural similarity variants differ mainly in their method of dividing the text into subtopics, both facilitated by a different implementation of the TextDivider interface. The TermExtractor class is responsible for most of the required functionalities described in the VSM (see 2.1 on page 30). We will describe our similarity algorithm in detail in Chapter 4.

Lucene

Alfresco makes use of another library to index and search the documents, namely the open source Apache Lucene[1] project. Before a document is stored in Alfresco, it is analyzed and meta-data and information on the contents are stored in an index to be able to quickly retrieve the document. The lexical analysis of a document consists of first splitting the text of the document into separate tokens, called tokenization. A token, also called a term in the VSM (see 2.1 on page 30), is a categorized block of text. Common categories in lexical tokenizers of textual information are words,

[1] `lucene.apache.org`

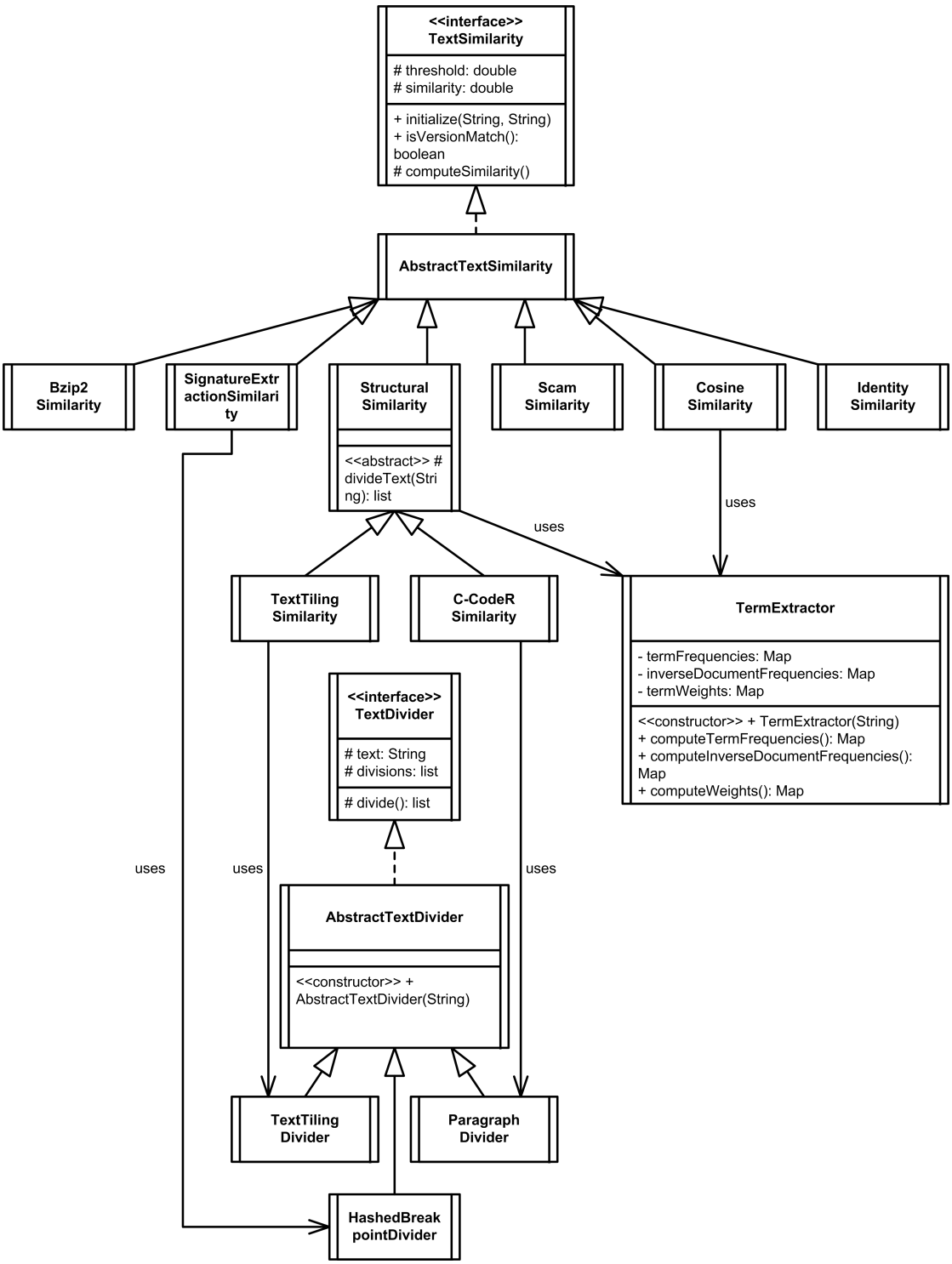

Figure 3.9: *Class diagram of the TextSimilarity package*

numeric sequences and acronyms. Alfresco uses the class `AlfrescoStandardAnalyzer` to analyze the text of a document in Lucene, which performs the following actions:

1. **Split the text into tokens.** Words are split at punctuation and at hyphens and email addresses and hostnames are recognized as one token
2. **Apply a standard filter to the tokens.** The filter removes *'s* at the ending of a word and removes periods from acronyms. Hostnames are split up into parts
3. **Change case.** Change every token to lower case
4. **Filter stop words.** If a token is a common word (e. g. *a*, *the*, *is*) it is removed from the stream of tokens
5. **Replace accents.** Accented letters (*é*, *á*, *ò*, etc.) are replaced by their non-accented equivalents

Each remaining token is then stored in an inverted index: the key of the index is the token, and the associated value is the set of documents in which the token occurs. Note that step 4 is language dependent, as each language has a different set of stopwords. Alfresco uses the English language as default, and cannot easily be configured to use multiple languages simultaneously, selecting a different analyzer depending on the language of the document. The search functionality within Alfresco is realized by passing the search query to Lucene and displaying its results.

Chapter 4.

The C-CodeR Algorithm

This Chapter describes the similarity algorithm we introduce for coderivative document recognition. In Section 4.1 we set out some theoretical groundwork on similarity algorithms. We continue in Section 4.2 to specify the detailed operation of our similarity algorithm. We conclude with Section 4.3 where we discuss various evaluation methods.

4.1. General theory

A similarity algorithm with the objective to identify coderivative (or topically similar) documents can be seen as a classifier. A classification problem is the problem of separating objects of a particular domain into separate classes. More specifically, coderivative document recognition is a binary classification problem, since there are only two classes involved: non-coderivative documents and coderivative documents. A classifier decides upon input of an object, to which class the object belongs. In general, a classifier has to be trained on a dataset before the actual classification can begin. The training is performed with a dataset of which the actual classification of objects is known (also called a *truth list*), and during training the parameters of the classifier are adjusted to reach optimal performance.

If we assume that a similarity algorithm's output is a similarity score $s \in [0,1]$ for a document pair, where 0 indicates no similarity at all and 1 corresponds with a perfect match, one of the parameters of the classifier is the threshold value $t \in [0,1]$. The classifier assigns document pairs with a similarity value equal to or greater than the threshold value to the coderivative document group, and classifies others as non-coderivative documents. Thus:

$$(A,B) \in \begin{cases} \text{non-coderivative} & sim(A,B) \equiv s < t \\ \text{coderivative} & sim(A,B) \equiv s \geq t \end{cases}$$

.

A perfect similarity algorithm will produce two non-overlapping ranges for the values of s corresponding to either non-coderivative documents or coderivatives. If this is the case, then a threshold value t can be any value in between these ranges. However, due to it being an automated process and due to the limited size of the training set, any similarity algorithm will make mistakes and assign to some non-coderivative documents higher similarity values than to

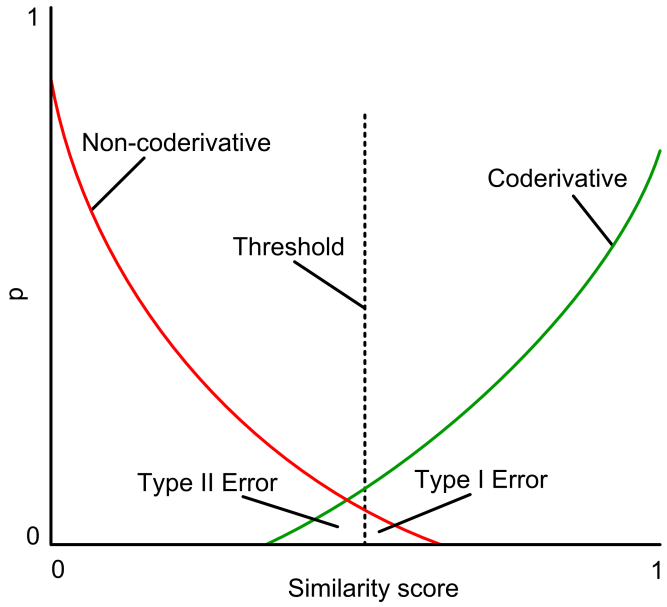

Figure 4.1: *Example of probability density functions of similarity values*

some coderivative ones, and vice versa. These classification errors are called Type I and Type II errors and under the null hypothesis of a document pair being non-coderivative are defined as:

Type I Error. The error of rejecting the null hypothesis given that it is actually true (false positive): classifying a document pair as coderivative while they actually are not

Type II Error. The error of failing to reject the null hypothesis given that the alternative hypothesis is actually true (false negative): classifying a document pair as non-coderivative while they actually are

To clarify this, we present an example graph of the two probability density functions of the similarity scores of documents classified as coderivative and non-coderivative, see Figure 4.1.

4.2. Algorithm description

Most of the document comparison algorithms aim to identify similar documents. Our aim is more specific, since we want to be able to identify coderivative documents. We found only one research paper investigating this objective, written by Hoad and Zobel (2003). However, we believe that the TextTiling algorithm introduced by Wan (2008) is the most promising one of the existing similarity algorithms. In a professional environment where many topically similar documents are likely to be created, the structure of a document would be a tell tale sign to decide

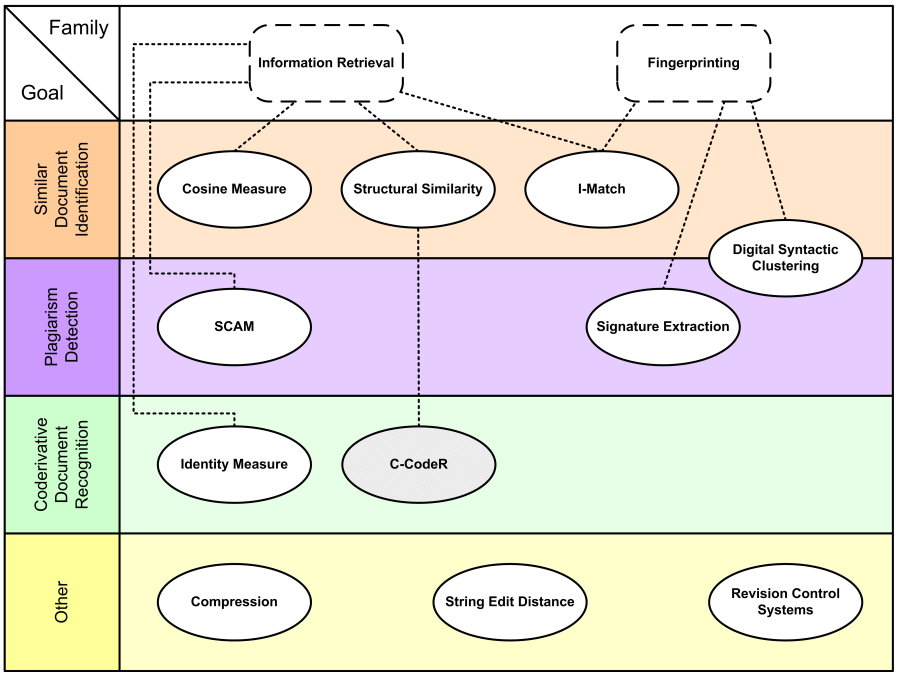

Figure 4.2: *C-CodeR compared to related work*

if the documents under consideration are just similar in topic, or are in fact coderivatives. We use the idea of Wan to extract the structure of two documents and compare the subtopics one by one. However, we extend his algorithm to be more suitable to detect coderivatives, creating our own algorithm: *C-CodeR* (Cayman-Coderivative Recognition). C-CodeR is a derivative of the TextTiling algorithm by Wan (2008) and relates to the other algorithms as described in Related Work (see Chapter 2 on page 29) as illustrated in Figure 4.2.

4.2.1. Subtopic separation

The similarity algorithm developed by Wan (2008) uses the TextTiling (Hearst, 1994) technique to separate a document into subtopics. This is an advanced method which tries to detect topic breaks within a document, using a sliding window of terms and calculating the cosine measure (see Section 2.1.1) at every position. In the experiments, Hearst (1994) evaluates her subtopic separation technique against a panel of human readers. Mostly, the location of the subtopic boundaries overlap with the human judgment, or are only slightly off. Although this suggests that the TextTiling algorithm is a good technique to separate a document into semantic passages, we adopt a different approach.

Because of the computation time involved when applying the TextTiling algorithm to a document, we prefer to use a simpler method to be able to perform quick coderivative document recognition. We simplify the subtopic separation step into splitting the document at paragraph boundaries. A paragraph boundary is considered to be an empty line in a document (i. e. two carriage returns). A minimal paragraph length is added to prevent short paragraphs such as headings being isolated as single subtopics. From manual inspection of collected documents, we conclude that a minimal paragraph length of 50 words is an appropriate value.

4.2.2. Subtopic matching

After we have split both documents into subtopics, we apply the Kuhn-Munkres algorithm (Kuhn, 1955; Munkres, 1957) to find the optimal matching of subtopics. A bipartite graph $G = \{A, B, E\}$ is constructed for the two documents $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_m\}$ (see Figure 2.2 on page 33 for an example). The cosine similarity value is calculated for every subtopic pair (a_i, b_j), $0 \leqq i \leqq n, 0 \leqq j \leqq m$, and set as weight w_{ij} for edge e_{ij} in the graph. We employ a high cutoff value $\tau = 0.25$ since subtopics with a low similarity score are not likely to be derived from the same source. The resulting optimal matching *OM* of subtopic assignments is used to calculate the similarity factors for the C-CodeR algorithm.

4.2.3. Similarity factors

The similarity factors and their weights as suggested by Wan (2008) strike us as leaving room for optimization, especially with respect to coderivative detection. We look into improving the existing factors and develop new ones.

The first similarity factor, the optimal matching factor as defined in Equation 2.6, is the average of the cosine similarity measure values of the *OM* of the subtopics between the two documents. A common change in a document is the addition and/or removal of one or a few paragraphs. If the paragraphs of the document and its coderivative are matched against each other, most of them will result in a high cosine similarity value. However, the few added or removed paragraphs will not match against anything, resulting in a very low cosine similarity value. This in turn will lower the average of the cosine similarity values of the optimal matching of subtopics.

To counter this unjustly low optimal matching factor, we propose to replace the average of the cosine similarity values by their median. The median is more robust against outliers, and will result in a high value, if most of the cosine similarity values are high. We therefor replace the optimal matching factor with the median weight factor, defined as:

$$sim_{ME}(A,B) = \text{median}(w_{00}, \ldots, w_{nm}) \tag{4.1}$$

, where w_{ij} is the weight of the edge e_{ij} in the set of edges in the *OM*. The median factor is also more robust against a few well matching subtopics (for example, standard legal paragraphs) among an *OM* of otherwise low scoring subtopic pairs.

We introduce a new similarity factor: the subtopic length factor. The cosine similarity value of two subtopics is completely based on the terms of the two subtopics and ignores all other information. Two subtopics that form a coderivative pair will not only share a large number of terms but will also be likely to be of approximately the same length. Two subtopics of a wrongly assigned subtopic pair can be of any length with equal probability. This leads us to introduce the subtopic length factor:

$$sim_{SL}(A,B) = \frac{|OM| - \sum_{e_{ij} \in OM} \frac{||a_i| - |b_j||}{\max\{|a_i|, |b_j|\}}}{|OM|} \tag{4.2}$$

, where $|OM|$ is the number of edges in the OM and $|a_i|$ is the length (in number of characters) of the subtopic a_i.

A more crude factor we introduce is the size difference factor. We argue that two coderivative documents are more likely to be of approximately equal length than two random chosen documents. The size difference factor is defined as:

$$sim_{SD}(A,B) = 1 - \frac{||A| - |B||}{\max\{|A|, |B|\}} \tag{4.3}$$

, where $|A|$ is the length (in number of character) of document A.

We leave it to our experiments to compare these new similarity factors to the ones defined by Wan (2008) and to select the most optimal combination of factors for the C-CodeR algorithm. For now, we combine them into a preliminary similarity measure as a weighted sum:

$$sim(A,B) = \zeta \cdot sim_{ME}(A,B) + \eta \cdot sim_{SL}(A,B) + \theta \cdot sim_{SD}(A,B) \tag{4.4}$$

.

4.2.4. Parameter optimization

The parameters of the many similarity algorithms are tuned heuristically on a small dataset, and Wan (2008) adopts the same approach for the factors of Equation 2.9 on page 35. We do not use heuristics to find the parameter values, but use a form of function optimization to ensure that the optimal values are found. Additionally, we use the k-fold cross-validation method to estimate the parameters and evaluate the algorithm. By using cross-validation, we prevent the overfitting of parameters and obtain a robust estimation of the performance of an algorithm. Section 5.4.4 contains more details on the employed cross-validation method and the function optimization.

4.3. Evaluation methods

Several methods exist to evaluate classification algorithms. To be able to perform a solid comparison of the C-CodeR algorithm against other algorithms, a quantitative measure is required.

The majority of the algorithms we found when researching related work focus on similar document identification. Since the topical similarity between two documents is subjective, it is impossible to attach a quantitative analysis of the performance to an algorithm identifying these. In their experiments, the authors of the related work we investigated are therefor rarely able to make a solid claim on the actual performance of their algorithm compared to others.

Our objective is to recognize coderivative documents, which is not a matter of subjectivity. A document is a new (or old) version of another document, or not (leaving aside the incidentally partly copied documents). In this Section we introduce four methods to evaluate the performance of a classification algorithm, one of which is a quantitative measure resulting in a single performance value. The first two are very common and widely known, while the second two are more rare, but useful to provide some insights into the performance of an algorithm over a range of parameter values or documents.

4.3.1. Confusion table

A confusion table can be considered as the summary of the outcome of classifying a number of instances. A classification of a pair of documents can fall into one of four categories. If the two documents are not coderivatives and the algorithm classifies them as such, then this is called a true negative (TN). If the algorithm unjustly classifies them as being coderivatives, the classification instance is a false positive (FP), or a Type I error. The same applies to two documents that are coderivatives, the correct classification is called a true positive (TP), and an incorrect one a false negative (FN), or a Type II error. These four numbers can be displayed in a so called confusion table, see Figure 4.3. A perfect classification algorithm will only have non-zero entries on the main diagonal and a FP and FN count of zero. For example, Figure 4.4 shows the confusion table of a perfect classification algorithm with K document pairs that are not coderivatives and L pairs that are.

4.3.2. F-measure

Although a confusion table is a good method to summarize the classification result, it lacks in quantifying the performance of the classification algorithm in a single number. Several performance measures exist that do have this property, and they use the confusion table as input to calculate a single performance value. Accuracy is one of the most used performance measures,

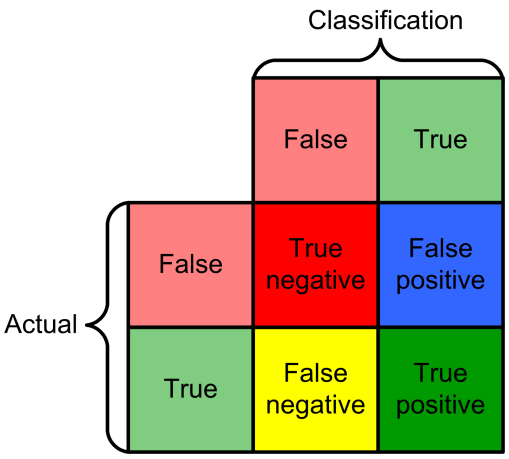

Figure 4.3: *General confusion table*

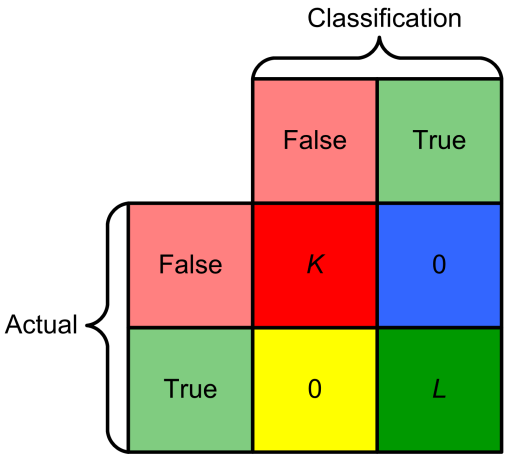

Figure 4.4: *Ideal confusion table*

and others are recall, precision, and specificity:

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.5}$$

$$recall = \frac{TP}{TP+FN} \tag{4.6}$$

$$precision = \frac{TP}{TP+FP} \tag{4.7}$$

$$specificity = \frac{TN}{TN+FP} \tag{4.8}$$

. On of the problems that all these performance measures have in common is that they are biased if the populations of actual true and false instances are not of (approximately) the same size (see also Provost et al. (2001)).

For example, consider 1000 document pairs consisting of 990 actual non-coderivatives and 10 actual coderivative documents. Now, an all-negative classifier would simple classify every document pair as being a non-coderivative pair, or an all-positive one would classify every pair as a coderivative pair. Clearly, both these classifiers have very poor performance, missing out on either all coderivatives, or all non-coderivatives. Their confusion tables are given in Figure 4.5 and Figure 4.6. Additionally, there might be a classifier that behaves very poorly, only classifying one coderivative instance as such, while classifying every other document pair as non-coderivative (see Figure 4.7 for the confusion table). It is obvious that these three classification algorithms perform very poorly, but in each case one of the performance measures suggests something different, resulting in a performance value of 1, or close to 1. For an overview of these performance values, see Table 4.1. The all-negative classifier achieves an accuracy value

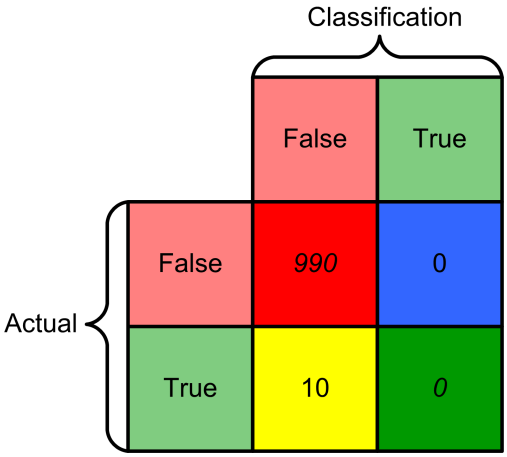

Figure 4.5: *All-negative confusion table*

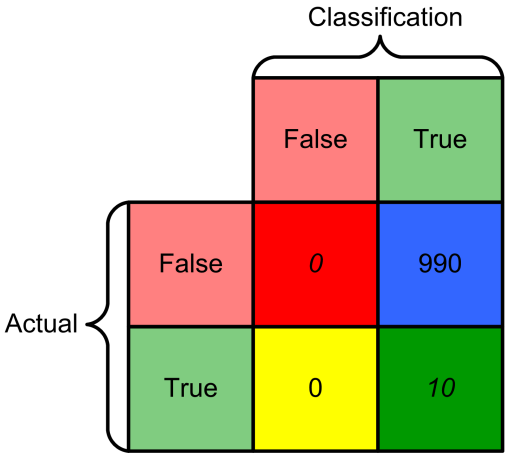

Figure 4.6: *All-positive confusion table*

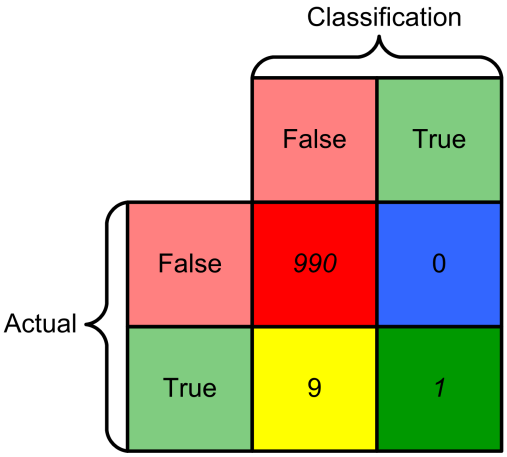

Figure 4.7: *One-positive confusion table*

	All-negative classifier	All-positive classifier	One-positive classifier
Accuracy	$\frac{990}{1000} = \mathbf{0.99}$	$\frac{10}{1000} = 0.01$	$\frac{991}{1000} = 0.99$
Recall	$\frac{0}{10} = 0$	$\frac{10}{10} = \mathbf{1}$	$\frac{1}{10} = 0.1$
Precision	$\frac{0}{0} = 0$	$\frac{10}{1000} = 0.01$	$\frac{1}{1} = \mathbf{1}$

Table 4.1: *Performance measure summary*

of 0.99, the all-positive classifier has a recall value of 1 and the precision performance measure of the one-positive classifier is equal to 1.

In the case of document comparison, the size difference between the classes is evident as it is clear that there are far more documents that are not coderivatives of each other, than there are documents that are. In our experiments, the population size could differ by as much as a factor of 800. Another problem is that in practical situations minimizing the false negatives (maximizing recall) and minimizing the false positives (maximizing precision) might not be of equal importance. We suspect that in the case of detecting coderivative documents in order to facilitate automatic knowledge management, minimizing the false positive count is far more important than minimizing the false negative count. Interviews with various people within the ECM practice of Capgemini confirm our suspicion. Missing out on a few coderivative document pairs is not that important, and can be corrected easily by hand. But wrongly classifying two different documents as being coderivatives could obscure the wrongly classified document from the view of the employees, rendering it irretrievable and therefor useless. For these two reasons, we need a different performance measure then the four mentioned above.

Rijsbergen (1979) derived the F_α-measure, which counters the problems above. It can be formulated as:

$$F_\alpha = \frac{(1+\alpha)\cdot(precision \cdot recall)}{\alpha \cdot precision + recall} \tag{4.9}$$

, where *precision* and *recall* are defined as in Equation 4.7 and Equation 4.6 and $\alpha \in (0,\infty)$ is a parameter which defines the relative importance of precision versus recall. The F_α-measure can be viewed as the weighted harmonic mean of precision and recall. Substituting a value of 1 for α leads to a performance measure which weighs precision and recall as equally important. A value of $\alpha > 1$ assigns more weight to maximizing recall, while a value of $\alpha < 1$ weighs maximizing precision (thus minimizing the false positives) as more important. We use $F_{0.5}$ as leading quantitative performance measure when evaluating the similarity algorithms.

By calculating the $F_{0.5}$ performance measure value for the complete range of threshold values, a threshold range graph can be constructed. This graph provides insight into the behavior of the algorithm for different threshold values, and demonstrates the dependence or independence of an algorithm on the threshold value.

4.3.3. Receiver Operator Characteristic

The receiver operator characteristic (ROC) curves were originally developed in the field of statistical decision theory and were used in World War II for signal detection on radar images (Lusted, 1971). The ROC curves were meant for radar operators to be able to distinguish between enemy targets, friendly ships and noise. ROC curves are currently widely used in the field of medicine to determine the efficacy of diagnostic tests (Swets, 1988). The aim of the ROC curve technique is twofold: on one hand, they enable one to compare multiple classifiers in their success in classifying instances in a single graph. Secondly, ROC curves can help to determine the desired threshold value.

Independent of what single-value performance measure is used, the difficulty of choosing the correct threshold value t remains. Ideally, the threshold value optimizes the chosen performance measure. The ROC curves help to visualize the behavior of the different classification algorithms over the complete range of possible threshold values.

The ROC curve is a graphical representation of the sensitivity (see Equation 4.6) versus ($1-$ specificity (see Equation 4.8)), or true positive rate versus false positive rate. The true positive rate is equal to the recall performance measure, and the false positive rate is defined as:

$$false\,positive\,rate = \frac{FP}{FP+TN} \tag{4.10}$$

. The values of these two performance measures depends on the chosen threshold value. Note that the ROC curves share the advantage of class size independence with the F_α performance measure.

Consider for example classifiers A, B, C and D. Their ROC curves are depicted in Figure 4.8. A perfect classifier would have a false positive rate of 0, and a true positive rate of 1, lying at point $(0,1)$ at the top-left part of the figure. A classifier with a threshold value of 0 simply identifies every document pair as coderivative and lies on point $(1,1)$, while a classifier with the opposite threshold value of 1 results in a classifier never distinguishing coderivatives from non-coderivative documents, lying on point $(0,0)$. The dashed line $y = x$ represents a classifier that randomly guesses the class the document belongs to. Any classifier that lies below this line should be immediately discarded, as it is worse than flipping a coin (note that inverting the classifier's classification does result in a classifier with above random performance).

Each point along a line of a classifier in the ROC graph represents a threshold value for that classifier. The further that point is to the top-left of the graph, the better the performance of the classifier. From the graph in Figure 4.8 it is immediately clear that classifier A is superior to all the other classifiers, independent of the choice for the threshold value. The ROC curve of classifier A continuously lies to the upper-left of all the other ROC curves. Following the same line of reasoning brings us to the conclusion that classifier D is the most inferior of all classifiers, independent of the value of the threshold. If minimizing the false positive rate is important, then classifier B is a better choice than C, since for the same low false positive rate, the true positive rate is higher. If, on the contrary, maximizing the true positive rate is crucial, then classifier C is the better choice. With these four specific classifiers however, classifier A is the best choice at all time.

Other performance measures based on the ROC curves include the area under the ROC curve (AUC) or the ROC convex hull method (Provost and Fawcett, 1997). Since these are complex methods and evaluating different performance measures is not within our scope, we do not incorporate them in our similarity algorithm evaluation.

4.3.4. Pixel rendering

Choi (2000) introduces a graphical representation of a head-to-head comparison of a group of documents which he calls a similarity matrix. To prevent confusion and to better suit the tech-

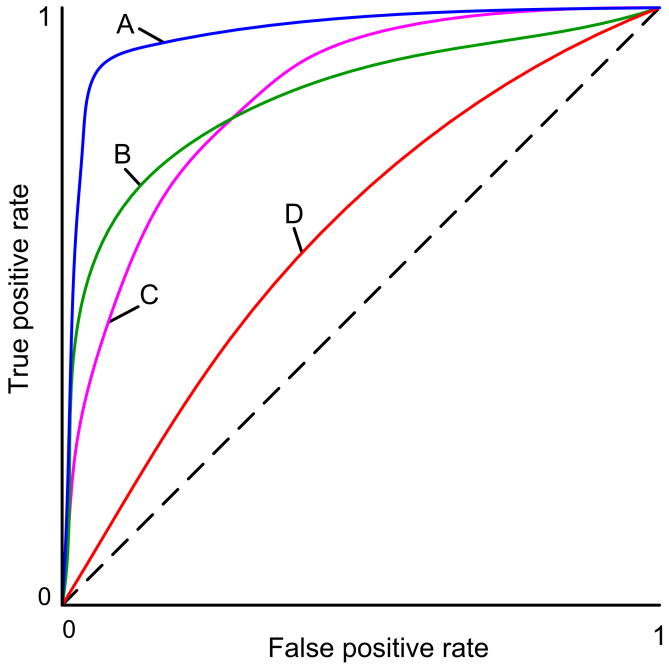

Figure 4.8: *ROC curves example*

nique, we use the name *pixel rendering*. In a pixel rendering the ordered set of documents are set alongside the two axes, both starting at the top-left. The color of each pixel corresponds with the similarity value, calculated by the similarity algorithm for the corresponding document pair. We assume that the similarity algorithm is symmetrical ($sim(A,B) = sim(B,A)$), resulting in a symmetric pixel rendering image. The main diagonal (from top-left to bottom-right) denotes the similarity value of each document compared with itself, and should therefor correspond with the highest possible value. An intuitive colormap for the $[0,1]$ interval of the possible similarity values is a grayscale colormap, with a value of 0 correspondent with the color black, and 1 correspondent to white. To improve readability and to save ink we invert this colormap, resulting in a black main diagonal and most pixels being white, corresponding with document pairs with low similarity values.

If a threshold for the similarity algorithm is chosen, the pixel rendering can be limited to two colors: either white for a non-coderivative pair, or black for a coderivative pair. This provides a threshold to the pixel rendering image: every pixel with a color lighter than the threshold gray value is set to white, and every pixel with a color darker than the threshold gray to black. If the documents are ordered in such a fashion that all the coderivative documents appear successively in the document set, then the perfect classification would result in having only white squares alongside the main diagonal. Any black pixel off the main diagonal provides a quick and clear signal that the classification of the similarity algorithm was not perfect: it corresponds with a false positive. False negatives are manifested as white pixels, often within black squares. To improve the visibility of the false positives and negatives, we adjust the color of the corresponding pixels to blue and yellow, matching the confusion table coloring. Examples of the pixel renderings are depicted in Figure 4.9, Figure 4.10 and Figure 4.11.

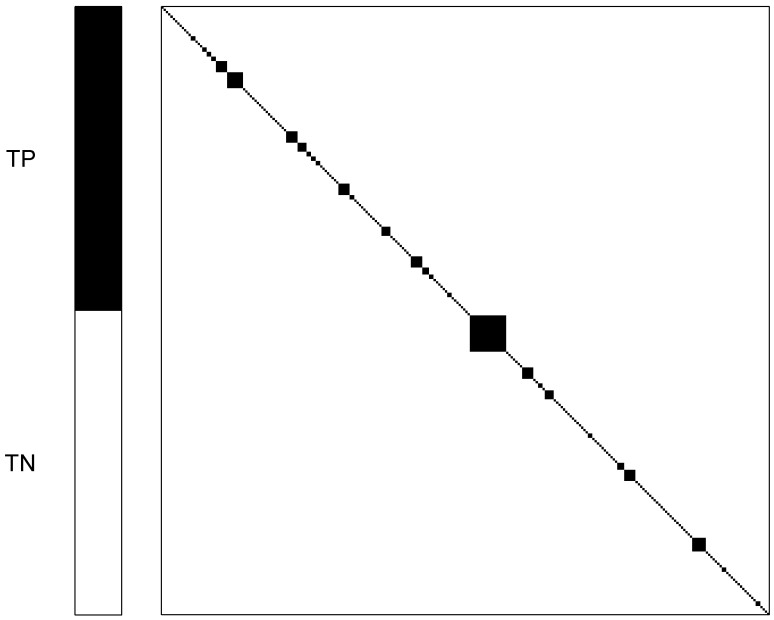

Figure 4.9: *Pixel rendering example, perfect classification*

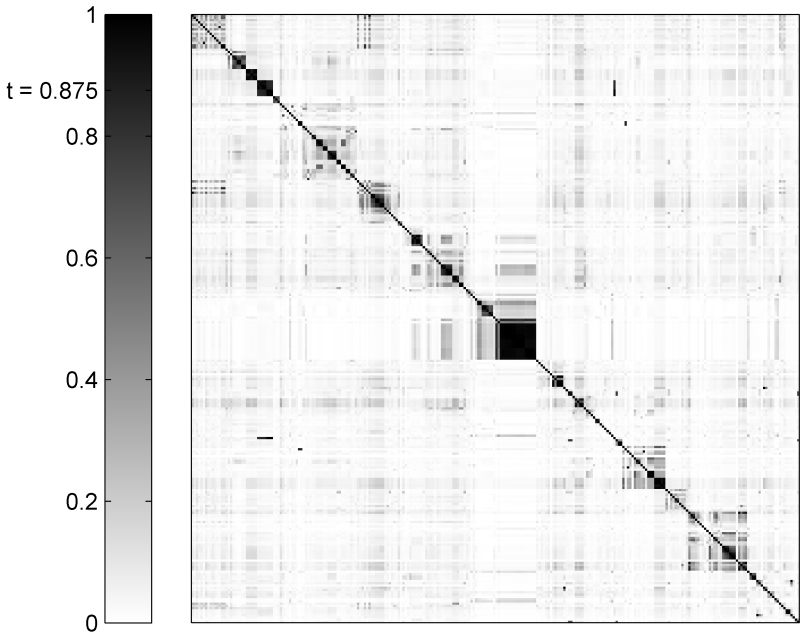

Figure 4.10: *Pixel rendering example*

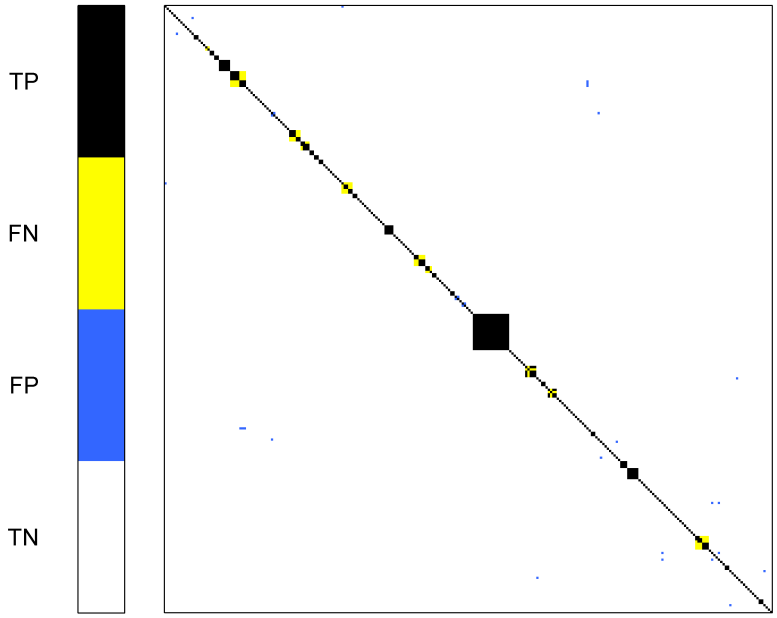

Figure 4.11: *Pixel rendering example, with threshold applied*

Chapter 5.

Implementation of the Cayman System

In this Chapter we describe the implementation of the Cayman application. We start with some general information on the development platform we use in Section 5.1. A description of the tools and programs we use during the implementation process can also be found in this Section. We then proceed to set out the implementation details of the various subsystems: the Cayman Collector is treated in Section 5.2, the Alfresco extension is discussed in Section 5.3 and Section 5.4 deals with the implementation of the similarity algorithm. Finally, we conclude the Chapter with some words on testing, in Section 5.5.

5.1. Development platform

5.1.1. Operating system

The Cayman system is developed on a notebook computer with the Microsoft Windows XP operating system. The system has an Intel Pentium M 1.5GHz processor and 1GB of RAM. During the course of our research, the possibility of upgrading the system with the Microsoft Windows Vista operating system and slightly more processing power was offered, which we declined. We estimated that the process of upgrading our system and transferring our applications would cost more time than could be gained with the slightly better software and hardware.

5.1.2. Programming language

We decide to use the Java Standard Edition platform[1] and the Java 6.0 programming language[2], implemented by Sun Microsystems, as our main development platform. The default Java compiler and Java Runtime Environment (JRE) are used. The code conventions[3] for the Java programming language are followed as much as possible.

By choosing the Java platform we can be sure that the application will run on a wide range of hardware and software, thus will be compatible with (almost) all the systems used within Capgemini. The Java platform offers two more advantages. Firstly, the high availability of

[1] `java.sun.com/javase`
[2] `java.sun.com/javase/6/docs/api`
[3] `java.sun.com/docs/codeconv`

reusable libraries and software tools, which prevents us from having to re-invent the wheel for every small required functionality. Secondly, the Alfresco knowledge management system is written in Java and since it is open source software, the Java source code is freely available. This simplifies the integration of our coderivative recognition subsystem with Alfresco and facilitates the reuse of part of its functionality.

Additionally, the MATLAB[1] language and environment are used for the more computational intensive parts of our research. This includes the parameter optimization, algorithm evaluation and graphing of the results. Apache Ant[2] is used as build tool for the Cayman Collector and the Alfresco extension. Since we are the only developers of the Cayman application and only use one system to write the source code, we do not feel the need to use a version control system such as Subversion. Backups are made regularly on an external hard disk drive to prevent data loss.

5.1.3. Software

Eclipse

The editor of choice is Eclipse 3.3[3] which offers an excellent integrated development environment for the Java programming language. Eclipse also supports developing and running Apache Ant build scripts.

MATLAB

The MATLAB 2007a environment is used to develop the scripts and functions written in the MATLAB language.

Netbeans

To create the graphical user interface of the Cayman Collector we use Netbeans 5.5.1[4]. The recently introduced GUI tools in the Netbeans application allow for fast development of a familiar Windows-like environment, without having to deal with the details of the various layout and GUI elements of the Java language.

5.1.4. Software development methodology

The standard software development methodology used in the ECM & UX technology of practice of Capgemini is the Rational Unified Process (RUP). As this process is more suited for large software development projects with many collaborating people, we decided (by mutual agreement) to deviate from the default software development process of the practice. It is unclear at

[1] www.mathworks.com/products/matlab
[2] ant.apache.org
[3] www.eclipse.org
[4] www.netbeans.org

```
org.cayman.collector
    Main
org.cayman.collector.model
    Collector
    Document
    DocumentFilter
org.cayman.collector.view
    Gui
    Settings
```

Listing 5.1: *Contents of the Cayman Collector package*

the start of our project if meeting the requirements is feasible, since the research and experiments have to show that a coderivative document recognition algorithm is possible. We therefor adhere to a more agile development approach, where each iteration of the software development is planned, designed, coded and tested. This incremental and prototyping methodology is used for development of Cayman Collector, the Alfresco integration and the C-CodeR algorithm.

We first develop the automatic document gathering application, the Cayman Collector. The development process is iterative: small pieces of functionality are added one at a time, and thorough debugging and testing takes place in between development. After the Cayman Collector is ready we look into the Alfresco application. We take time to familiarize ourselves with the various concepts, elements, services and frameworks found in this application. We also examine the application programming interface (API) and the methods of extending Alfresco. Since our research is experimental and we are not sure what the outcome will be, we start by implementing the basic features of the coderivative recognitions system, such as receiving the documents and storing them in the knowledge management system. After the groundwork has been laid out, we continue with the iterative design and implementation of the similarity algorithm.

5.2. Implementation of the Cayman Collector

As stated before, the Cayman Collector is implemented in the Java programming language. Two screenshots of the application's main and settings window are shown in Figure 5.1. An overview of the packages and classes is given in Listing 5.1.

5.2.1. Used libraries

Besides making use of the Java class library we use four other third-party libraries in this prototype application.

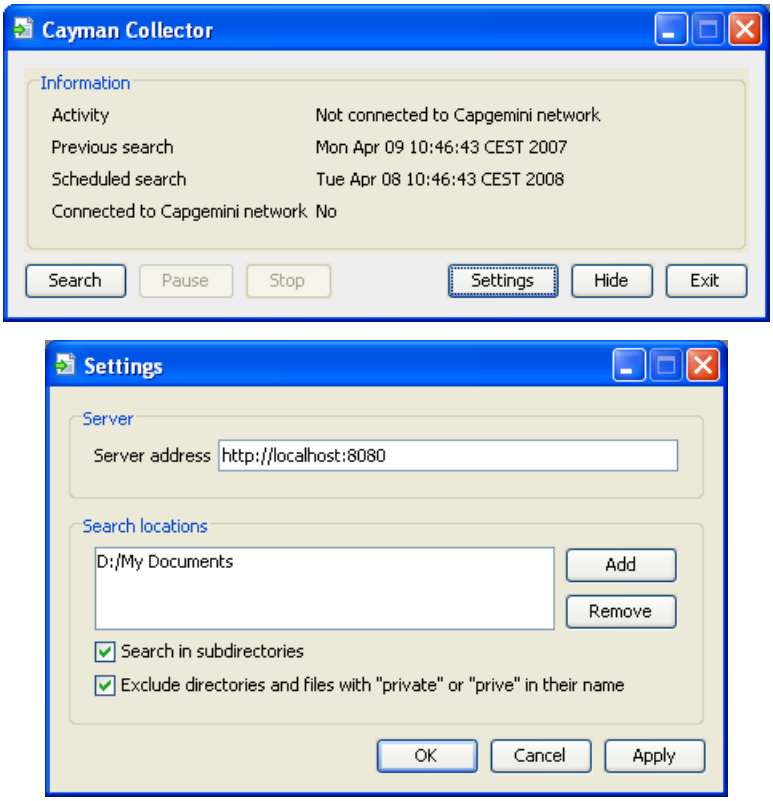

Figure 5.1: *Screenshots of the Cayman Collector*

- **MyJavaTools**[1]. Since sending a basic HTTP POST request is not incorporated in the Java class libraries, we are forced to either develop it ourselves, or find an alternative. Luckily, the Web package of MyJavaTools, available under the open source Apache License version 2.0[2], does exactly what we need

- **JDesktop Integration Components (JDIC)**[3]. To achieve seamless desktop integration across multiple platforms, we use JDIC to provide an icon in the system tray. This addition to the interface ensures that intrusion into the desktop workspace of the user is minimal, while having the key functionality near at hand. The JDIC code is available under the GNU Lesser General Public License[4] (LGPL)

- **log4j**[5]. The logging framework Apache log4j, also available under the Apache License version 2.0, is used to write log statements to a log file. The level of detail of logging can be set in a configuration file, as well as the log format and whether or not the log file should be changed daily

- **One-JAR**[6]. The final library we use is One-JAR. This library enables us to wrap the entire Cayman Collector application into one file. This makes integrating the required libraries and distributing the application much easier. It is available under its own open source license[7]

5.3. Implementation of the Alfresco extension

We use the Alfresco Community 2.9B release of the Alfresco distribution. This version can be downloaded and used free of charge, while it still contains (almost) all of the functionality of the Enterprise edition (which is only available as time-limited trial software). Our extensions to Alfresco are packaged as an Alfresco Module Package (AMP). This is a standard way of providing extensions to the core of the Alfresco distribution. It can easily be integrated into any existing Alfresco installation. An overview of the Java packages and classes is given in Listing 5.2.

The web script that is responsible for receiving the documents and placing them in a folder in the Alfresco knowledge management system is implemented in JavaScript. This is the standard programming language when adding a web script to Alfresco that responds to a HTTP request. We choose to use the Message-Digest algorithm 5[8] (MD5) cryptographic hash function to create

[1] `www.myjavatools.com`
[2] `www.opensource.org/licenses/apache2.0.php`
[3] `jdic.dev.java.net`
[4] `www.opensource.org/licenses/lgpl-2.1.php`
[5] `logging.apache.org/log4j`
[6] `one-jar.sourceforge.net`
[7] `one-jar.sourceforge.net/index.php?page=documents&file=license`
[8] `ftp.rfc-editor.org/in-notes/rfc1321.txt`

```
org.alfresco.module.cayman.alfresco.behaviour
    Contenthash
org.alfresco.module.cayman.alfresco.model
    CaymanModel
org.alfresco.module.cayman.alfresco.repo.action.executer
    CalculateContenthashExecuter
    ExtractLanguageExecuter
    GuessMimetypeExecuter
    RecognizeCoderivativesExecuter
    MoveFailedRecognitionExecuter
```

Listing 5.2: *Contents of the Alfresco module package*

the unique hash value for the text of a document. Although as a security algorithm the MD5 function is compromised, the probability that two texts generate the same 128 bit fingerprint is still negligible. One of the main advantages is that implementations of the MD5 algorithm are widely available.

We mainly make use of two services of Alfresco: mimetype guessing and content transformation. The first tries to make a guess of the mimetype of a document based on its file extension. If this fails, it reads the first few bytes of the file and tries to determine the mimetype from this. Our `GuessMimetype` action uses this service to extract the mimetype from a submitted document and stores it in the properties. The content transformation service is used extensively by our coderivative document recognition system, as the similarity algorithm is based on plain text. Additionally, the input for calculating the unique hash value of the document and extracting the language are also based on the plain text version of a document. Consequently, the other three actions we implement (`ExtractLanguage`, `CalculateHash` and `RecognizeCoderivatives`) use the transformation service.

5.3.1. Used libraries

We use two third-party libraries for the implementation of our Alfresco extension.

- **Fast MD5**[1]. The unique hash value of a document is calculated with help of the Fast MD5 package, available under the GNU LGPL. This package claims to calculate the MD5 hash value approximately twice as fast as the native Java method, or even faster when operating system native methods are enabled. As the MD5 hash of a document is recalculated every time the document is updated, a fast implementation is required

[1] `www.twmacinta.com/myjava/fast_md5.php`

- **TCatNG**[1]. The `ExtractLanguage` action uses the n-gram language classifier of the TCatNG toolkit to extract the language of a document (licensed with the BSD License[2])

5.4. Implementation of the similarity algorithms

The C-CodeR algorithm and the other similarity algorithms implemented for comparison are programmed in the Java language. Although the execution speed of this language might not be optimal, the ease of programming and availability of libraries shorten the implementation time significantly. The similarity algorithms and supporting classes are implemented as a stand-alone package with a small set of public methods. See Listing 5.3 for an overview of the `TextSimilarity` package. The public methods are described in Section 5.4.3.

In addition to our own algorithm, the newly developed C-CodeR algorithm, we implement five of the most promising similarity algorithms found in related work and a baseline algorithm which all similarity algorithms should outperform.

The baseline algorithm is the cosine measure, described in Section 2.1.1. The term vectors which are input for Equation 2.3 on page 31 are the terms of the documents, and their weights are calculated as the product of the term frequency and inverse document frequency as set out on page 30. The extraction of the terms from a document will be described in Section 5.4.2. This algorithm is the most simple one and forms the basis for almost all similarity algorithms in the area of information retrieval.

The identity measure (see Section 2.1.2) is the most obvious choice for an alternative similarity algorithm to implement, since it is specifically targeted on recognizing coderivative documents. We implement the fifth identity measure that Hoad and Zobel (2003) introduced and that is given in Equation 2.4. Because the algorithm does not return a similarity score in the $[0,1]$ range, we normalize the the score by dividing it to the score attained by the document compared with itself.

The third algorithm we add to our set is the TextTiling algorithm by Wan (2008) since it is the basis for the C-CodeR algorithm developed by us. We implement the new similarity factors and improve the algorithm by estimating their optimal values with a formal optimization method, as will be described in Section 5.4.4.

We implement two algorithms from the field of plagiarism detection: the SCAM and signature extraction algorithm. The fact that the SCAM algorithm filters the set of terms by taking into account the closeness of their frequency seems a novel method to us, and might be appropriate to recognize coderivative documents. The signature extraction algorithm comes from the family of fingerprinting techniques and uses the hashed-breakpoint method to divide the text into chunks. This strikes us as a suitable division method that is robust against the changes made in coderivative documents. We change the parameter c in the $f \bmod c = 0$ equation of the algorithm

[1] `tcatng.sourceforge.net`
[2] `www.opensource.org/licenses/bsd-license.php`

```
org.cayman.textsimilarity.algorithm
    TextSimilarity                      (Interface)
    AbstractTextSimilarity              (Abstract)
    StructuralSimilarity                (Abstract)
    Bzip2Similarity                     (Algorithm)
    CCodeRSimilarity                    (Algorithm)
    IdentitySimilarity                  (Algorithm)
    CosineSimilarity                    (Algorithm)
    ScamSimilarity                      (Algorithm)
    SignatureExtractionSimilarity       (Algorithm)
    TextTilingSimilarity                (Algorithm)
org.cayman.textsimilarity.evaluation
    CrossValidationSetup
    DocumentSimilarity
    SimilarityConstants
org.cayman.textsimilarity.util
    I18N
org.cayman.textsimilarity.util.text
    TextDivider                         (Interface)
    AbstractTextDivider                 (Abstract)
    HashedBreakpointDivider
    ParagraphDivider
    TermExtractor
    TextTilingDivider
    TextTools
org.cayman.textsimilarity.util.texttiling
    StopwordWrapper
```

Listing 5.3: *Contents of the* `TextSimilarity` *package*

to 50, to obtain average chunk length of 50 words, equal to the minimal parameter length of the C-CodeR algorithm.

As Finkel et al. (2002) did in their experiments, we too use the variance method to filter the chunks. Since we suspect that Equation 2.11 is not correct (reaching only a maximum similarity value of 0.5), we substitute it by:

$$sim(A,B) = \frac{2 \cdot |d(A) \cap d(B)|}{|d(A)| + |d(B)|} \tag{5.1}$$

The last algorithm we compare the C-CodeR algorithm with is based on the file compression technique described in Section 2.3.2. This out-of-the-box method seems to be a novel approach to similarity identification, and it will be interesting to compare it to other algorithms. As suggested by Finkel et al. (2002) we use an implementation of the bzip2 compression algorithm en calculate the similarity value as given in Equation 2.12.

5.4.1. Used libraries

We use three external libraries in our implementation of the similarity algorithms.

- **HungarianAlgorithm**[1]. We use an implementation of Gary Baker of the Kuhn-Munkres algorithm to solve the optimal matching problem of subtopics in the structural similarity algorithms
- **JTextTile**. We use the JTextTile Java implementation by Choi (1999) of the TextTiling algorithm (Hearst, 1994), which is free for research, academic and non-profit making purposes
- **Bzip2**. The bzip2 similarity algorithm uses the Windows implementation of the bzip2 compression algorithm by Seward

5.4.2. Document preprocessing

Every document is transformed to plain text by Alfresco before being passed to the `TextSimilarity` package. The similarity algorithms from the information retrieval family rely heavily on tokenization, and we use a tokenizer similar to the one described in 3.3.2, namely the Snowball analyzer[2], also supplied by Lucene. The only differences are (1) the localization of the stopwords, (2) the omission of the replacement of accents and (3) the addition of a stemmer. Stemming is the process of reducing inflicted words to their stem. For example, the words *manager*, *management* and *managing* would all be replaced by their root *manag*. The main advantage of using the Snowball analyzer is that equivalent implementations are provided for all the major European

[1] `www.thegarybakery.com/hungarian`
[2] `snowball.tartarus.org`

languages, such as French, German, Spanish, Portuguese and most important for our research: English and Dutch. Although Alfresco does supply lexical analyzers with stemming for various languages, they differ slightly in implementation, and therefor cannot be interchanged without influencing the result of the similarity algorithm.

5.4.3. Java methods

The knowledge management system needs to invoke three methods to determine whether or not two documents form a coderivative pair. Since all similarity algorithms share a common interface, the implementation of the similarity algorithm is completely transparent and can be replaced by an implementation of another algorithm without the need to change anything in the rest of the Cayman system. The `TextSimilarity` interface supplies five public methods.

`public void initialize(String aSourceText, String aTargetText)`. This method must be invoked prior to all other methods to initialize the similarity algorithm with the texts of the documents it has to compare. The implementation of the algorithm stores the texts for later comparison and may initialize some values needed for its computation

`public void computeSimilarity()`. This is the core method of the similarity algorithm and performs the necessary computations to determine if the supplied documents are a coderivative pair

`public boolean isCoderivative()`. This method returns `true` if the documents are considered to be coderivative, and `false` otherwise. Essentially, this method compares the computed similarity score with the threshold value

`public double getSimilarity()`. Although for basic operation calling this method is not required, a system using the `TextSimilarity` package might wish to know the similarity score $s \in [0,1]$ to determine the certainty of the algorithm

`public double getThreshold()`. This method can be used in conjunction with the method above to get an idea on how certain the algorithm is. It returns the threshold value used by the similarity algorithm

5.4.4. Parameter optimization

We found that many similarity algorithms use an estimate for the optimal value for threshold and other parameters of the algorithm. We wish to be more thorough and adopt a different approach, namely the approach of function optimization. The function to be optimized is the $F_{0.5}$ performance measure, defined as:

$$F_{0.5} = \frac{\frac{3}{2} \cdot precision \cdot recall}{\frac{1}{2} \cdot precision + recall} \tag{5.2}$$

. The outcome of the performance measure is dependent on the confusion table, which in turn is determined by the threshold value. If the desired precision for the optimal value of the threshold is 0.1, then the confusion table and $F_{0.5}$ value have to be calculated 101 times.

Suppose our dataset consists of $1,000$ documents and we want to do a head-to-head comparison of all the documents for performance evaluation. The number of distinct similarity values of all document pairs (given a symmetric similarity algorithm) is equal to $\frac{1000^2}{2} = 500,000$. It follows that we need to do $50,500,000$ comparisons of a similarity score to the threshold. While this still is feasible, consider what happens if the number of parameters increases from one to four (as in the structural similarity measure). To find the optimal values of these parameters with a precision of 0.1, the number of evaluations of the confusion table and the $F_{0.5}$ measure increase to $101^4 = 104,060,401$. With a dataset of $1,000$ documents, the number of comparisons of a similarity score to the threshold value adds up to $104,060,401 \cdot 500,000 = 52,030,200,500,000 = 5.2 \cdot 10^{13}$. With severe manual limitation of the ranges of the parameters we are able to evaluate one similarity algorithm in approximately 14 hours of CPU time, but is clear that a different method is needed.

To find the optimal values for the threshold and other parameters of the algorithms (for example the weight parameters of the structural similarity algorithm), we make use of the `patternsearch` function of the *Genetic Algorithm and Direct Search Toolbox* of MATLAB. This function uses a direct search method for solving optimization problems that does not require any information about the gradient of the objective function. In our case the objective function to be maximized is the $F_{0.5}$ performance measure, which, due to the discrete nature of the confusion table, is not differentiable. A direct search algorithm searches a set of points around the current point, looking for one where the value of the objective function is higher than the value at the current point. The upper and lower bounds for the threshold and other parameters are 0 and 1 and the linear equality constraint for the weight parameters of the structural similarity and C-CodeR algorithm is that their sum is equal to 1. We choose four starting points for the value of the objective function and parameters situated below and above the expected value to minimize the possibility of ending up in a local optimum. We run the `patternsearch` function with each set of starting points, and select the set of parameters associated with the highest value of the objective function after optimization.

5.4.5. Evaluation methods

The evaluation of the similarity algorithms is done without the burden of the Alfresco application. We add an extra class called `DocumentSimilarity`, calculating the similarity score between all documents contained in a specific folder in the file system, to the `TextSimilarity` Java package. This folder may contain many nested subfolders where the documents can also be placed. The actual classification of documents is derived from the folder structure. Every set of actual coderivative documents must be placed in its own subfolder, of which the name must start the string *match*. The class `DocumentSimilarity` writes the similarity scores (as well as the scores of the separate factors for the TextTiling and C-CodeR algorithms) to comma

separated value (CSV) files. Additionally, the actual similarity scores (0 or 1) and the file names of the documents are written to a file. The actual evaluation of the performance of the similarity algorithms is done in MATLAB and uses the generated CSV files as input. We use the $F_{0.5}$ performance measure as quantitative measure to assess the performance of the similarity algorithms.

Several MATLAB functions are written to assess the performance of the similarity algorithms. An overview of the main functions and scripts handling the similarity algorithms is given below.

`compute_simple_similarity`. This function computes the similarity scores for a set of documents for the simple similarity algorithms without additional parameters (besides the threshold). Essentially, this is equal to reading the CSV file where the score are stored

`compute_complex_similarity`. For more complex similarity algorithms where the similarity score also depends on one or more parameters, the score cannot directly be copied from the CSV file. This function is designed to compute them

`optimize_simple_similarity`. This function uses the `patternsearch` function to find the threshold value that maximizes the $F_{0.5}$ performance measure

`optimize_complex_similarity`. The more complex similarity algorithms require a different implementation to find the optimal values for parameters that maximize the performance measure. Again the `patternsearch` function is used

`display_roc`. Displays the ROC curves of multiple similarity algorithms

`display_pixel_rendering`. Displays the pixel rendering graphs of multiple similarity algorithms

`display_thresholdrange`. Displays the $F_{0.5}$ performance measure over the complete threshold range for multiple algorithms

`crossvalidate`. This function performs a cross-validation on multiple similarity algorithms

`compare_crossvalidation`. Compares the cross-validation outcome of multiple similarity algorithms with a statistical hypothesis test

The input parameters of the first two functions are: `(method, simobj)`, where the first is the name of the method under evaluation and the second is a MATLAB structure where information and computed values of the method are stored. Both optimization functions have one optional input parameter: `set_filter`. This parameter provides the possibility to filter the set of documents and only run the optimization on a subset, necessary for cross-validation. All four functions return a `simobj` instance. The remaining four are MATLAB scripts, operating not on one similarity algorithm, but on a set of algorithms, and make use of the first four functions. Additionally, there are several supporting functions.

`calculateConfusion(threshold, varTrue, varFalse)`. The calculateConfusion function calculates the confusion table upon input of the threshold value and the arrays of actual true and false similarity scores

`calculateSimilarityPerformanceMeasure(parameters, pm, varTrue, varFalse)`. This function is optimized in the `optimize_*_similarity` functions. Its input values are the parameters for the similarity algorithm, the name of the performance measure to be evaluated and the arrays of actual true and false similarity scores

`performanceMeasure_fdot5(confusion)`. Calculates the value for the $F_{0.5}$ performance measure upon input of the confusion table

`performanceMeasure_recall(confusion)`. Calculates the value for the recall performance measure upon input of the confusion table

`performanceMeasure_falsePositive(confusion)`. Calculates the value for the false positive rate performance measure upon input of the confusion table

5.5. Testing of the Cayman system

Although the Cayman system is designed to be a prototype, in order to guarantee a certain level of quality of the system a method of software testing is required. Our development process is agile and iterative, so we perform a software test in each cycle. Such a test consists of first performing thorough static testing by means of code inspection and matching code to the design. After this, we continue with dynamic testing using the integrated development environment. We perform line-by-line execution of the code with the stepping commands in the Debug perspective of Eclipse. Input files are carefully designed to reach 100% code coverage and perform boundary value analysis. This white box testing approach is followed by a black box test cycle in which we test the functionality of the system. When testing the implementation of the similarity algorithms the black box method is especially important, since the outcome of the evaluation and comparison depends solely on the correct functioning of these algorithms. Where possible, we verify the operation of an algorithm with manual calculation.

The software testing methods described above focus mainly on verification, i. e. checking that we build the software right. Additionally, there is validation which evaluates if we build the right software. Software validation in between development cycles is done by performing small usability tests and presenting the software to the users (the employees of Capgemini, mainly members of the focus group mentioned in Section 3.1 on page 43). Validation of the Cayman system as a whole is part of our experiments, and will be discussed in Section 6.2.

Chapter 6.

Experiments

In this Chapter we present the experiments we conducted with the Cayman system, as well as their results. As the performance of the system as a whole depends on the two main parts, we conduct two separate experiments. The first and most important part is the evaluation of our newly developed C-CodeR similarity algorithm. In Section 6.1 its performance is compared with our implementation of the other similarity algorithms, mentioned in Section 5.4.

The second part of our experiments focuses on *validation* of the Cayman system, as described in Section 5.5. We install the Cayman system in a practical environment and evaluate its performance in Section 6.2.

6.1. Similarity algorithms

Before entering into the details of the experiment, we first give a general overview of our setup. A flowchart of the experiment is depicted in Figure 6.1. In this flowchart yellow rectangles represent processes that are executed in the Java language, and processes displayed as orange rectangles take place in the MATLAB environment. The single green rectangle represents a macro written in Visual Basic for Applications.

The whole process starts with acquisition, preprocessing and creating a truth list of the dataset, which will be described in Section 6.1.1. Since manually creating a truth list is a difficult process which is susceptible to human error, we use the classification of a similarity algorithm to examine if our truth list is correct, visualized as the left part of Figure 6.1. In Section 6.1.2 we will introduce the cross-validation method which we will use to calculate the $F_{0.5}$ performance measure values.

The similarity factors of the C-CodeR algorithm introduced in Section 4.2.3 are preliminary and we need to select the most optimal combination of factors for this algorithm. This process is displayed in more detail in Figure 6.2 and will be described in Section 6.1.3.

After we have selected the most optimal factors for the C-CodeR algorithm, we can proceed to compare the seven algorithms with the cross-validation method. This process and the quantitative results are described in Section 6.1.4. These results consist of the average parameters and $F_{0.5}$ values, which are compared resulting in statistical probability values, or p values for short (displayed in the lower-right corner of Figure 6.1).

The results of several other evaluation methods are described in Section 6.1.5. Finally, in Section 6.1.6 we present an interpretation of the results described in the previous sections.

A description of the various data in the flowchart used as input for, or output of the processes in the experiment is given below.

Dataset. The dataset consists of all the documents converted to plain text

Dataset with truth list. If the coderivative documents in the dataset are grouped together in subfolders, we denote this by the dataset with truth list

Similarity scores. The similarity scores for all document pairs are placed in CSV files. Additionally, if the algorithm uses similarity factors, the similarity scores of these are contained in CSV files as well

Classified dataset. Applying a threshold to the similarity scores results in the classification, i. e. the number of true and false positives and negatives

Average $F_{0.5}$ and parameter values. The result of the cross-validation is the set of average $F_{0.5}$ performance measure and parameter values

Pixel rendering. This visualization method is a graphical representation of the similarity scores

Confusion table. The confusion table is a summary of the classified dataset

ROC curves graph. The ROC curves graph is a method to compare different algorithms

Threshold graph. The threshold graph displays the performance of different algorithms for the complete threshold range

P values. The p values are the probability values that the average $F_{0.5}$ value of an algorithm is statistically different from another algorithm

6.1.1. Dataset

Acquisition It is our objective to validate the performance of the similarity algorithms with a dataset of documents that resembles a real life environment. Most of the algorithms we found in our research into related work are not evaluated with such a dataset, but rather with a set of documents that are collected from the Internet. Examples of such sets are a collection of RFC documents, used by Finkel et al. (2002) for the signature extraction algorithm, a collection of random HTML documents, used by Broder et al. (1997) for the DSC algorihtm, and a set of articles from the LA Times, used by Chowdhury et al. (2002) for the I-Match algorithm. The results of these experiments are often not expressed in a quantitative measure, and even less often a solid comparison against existing algorithms is made.

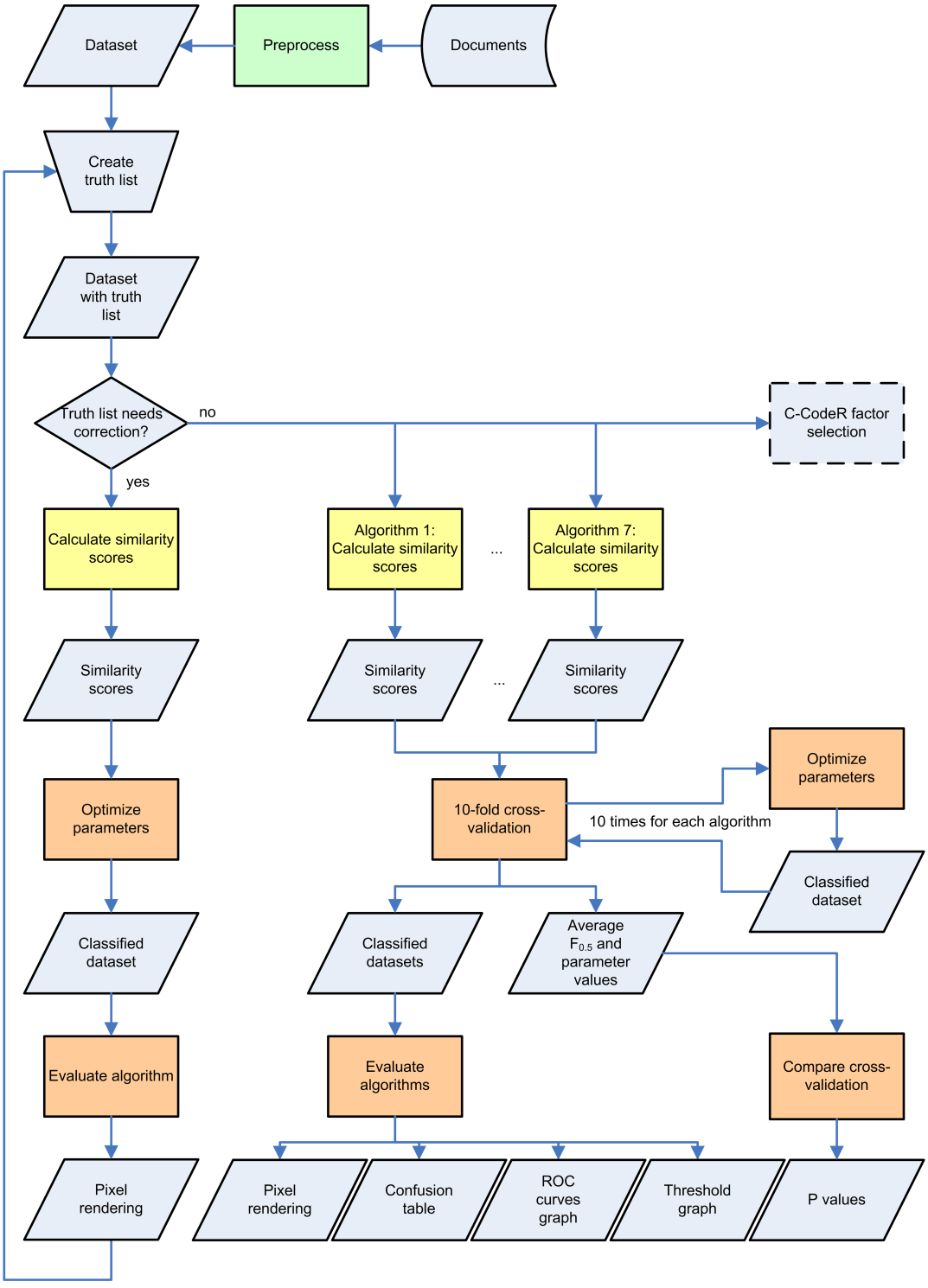

Figure 6.1: *Flowchart of the experiments*

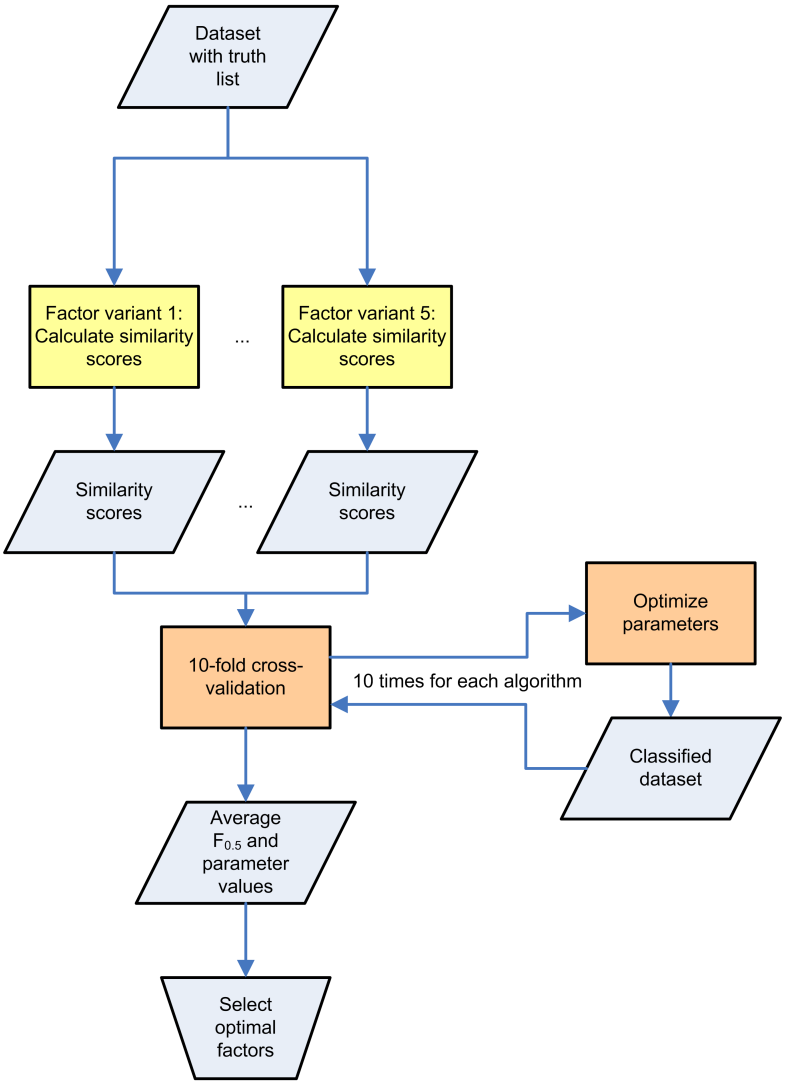

Figure 6.2: *Flowchart of the C-CodeR factor selection process*

Since the primary application of our algorithm lies within Capgemini, we strive to conduct our experiment with a representative dataset. To achieve this we collect documents used in the ECM & UX technology of practice of Capgemini. In concurrence with the self-formed knowledge management focus group of the ECM & UX practice, we ask seven project leaders within the practice to participate in the experiment.

Project leaders will acquire a large set of documents from all their team members during a project. As they collect documents from multiple sources and often need to retain old versions for continuity, the range of types of their documents is wide and their gathered documents will contain coderivative pairs. The document set collected from the project leaders is often the complete archive of current and recent projects. Note that since we collect all documents and not only coderivative pairs, the distribution over the classes is heavily skewed, favoring non-coderivative documents. Even if we collect only coderivative document pairs, the distribution would still be skewed, as the coderivative documents are grouped, and do not form a coderivative pair with every other document.

Preprocessing As this experiment is realized without the framework and services that Alfresco offers, we need to find a different way to transform the documents to plain text. To simplify this matter the documents are filtered on file type and only Microsoft Office 2003 and 2007 documents are selected for the dataset. A Visual Basic for Applications macro is written and executed in Microsoft Word 2003 to convert all the documents to the plain text format. Documents with a file size less of than 2 kilobytes (approximately half a page) or more than 200 kilobytes (approximately 150 pages) are discarded, as text based similarity algorithms perform poorly on small documents, and their computation time increases substantially with large documents. This results in a dataset consisting of 1807 documents.

We classify the language of the documents with the package mentioned in 5.3.1 on page 80 of which the result is displayed in Table 6.1. The Danish and German documents are probably mis-classifications, as these languages are rarely used and close to Dutch in letter frequencies. As the majority of the documents is written in the Dutch language we select the Dutch Snowball analyzer to tokenize our documents (see 5.4.2). Taking in account the substantial number of English documents and the fact that in technical design documents written in Dutch a large number of English terms will appear, we add the English stop words to the Dutch stop word set used in the lexical analyzer.

Truth list creation To create a truth list with the actual classification of the documents, manual inspection is required. We follow the method described in 5.4.5, placing all coderivative documents per coderivative group in a separate subfolder. The documents are first moved into subfolders by a method written in Java which performs a crude classification based solely on the filename of the document (for details see step 3 on page 55). We then proceed to manually verify the classification of the documents from each project leader separately. By splitting the total dataset up into small parts, we can keep a global structure of the documents per part in our

Language	Number of documents
Dutch	1256
English	542
Danish	5
German	3
French	1

Table 6.1: *Language statistics of the dataset*

Statistic	**Value**
Total number of documents	1807
Number of coderivatives	775
Number of coderivative groups	218
Average document file size	19.06 kb

Table 6.2: *General statistics of the dataset*

mind, and are able to correct errors made by the crude automatic classification. Unfortunately, it is impossible to keep over 1800 documents and their contents in mind, and thus we are bound to miss the coderivative document pairs spread over the document sets of different project leaders. Some general statistics of the dataset are given in Table 6.2 and Figure 6.3.

We use the pixel renderings evaluation method to be able to correct the manually created truth list of coderivative documents. First, the similarity scores are calculated and the parameters optimized. Since preliminary results already indicated that the Cosine similarity algorithm performs well, we chose to use this algorithm to correct the truth list. The similarity scores of all the documents pairs are visualized as a pixel rendering. We add a graphical user interface element in MATLAB which displays two documents and their similarity score as determined by the algorithm. The content of this element is updated every time the user clicks on a pixel. This provides us with the possibility to inspect the whole dataset on possible missed coderivative documents which are spread across the document sets of different project leaders. Any strip of false positives far from the main diagonal is a sign that these might be missed coderivative pairs. Manual inspection of the documents reveals if the created truth list is in error, or the if the document pair actually is a false positive and the algorithm is in error. Using the pixel renderings we were able to correct over 40 manually wrongly classified documents. This demonstrates that a coderivative recognition algorithm actually outperforms manual classification when applied to large volumes

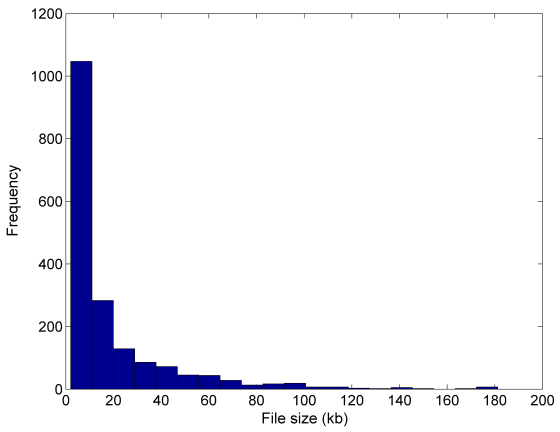

Figure 6.3: *Histogram of the file size of the dataset*

of data. The pixel renderings of some of the best performing algorithms are shown in Appendix E.

6.1.2. Cross-validation of the $F_{0.5}$ performance measure

To make a robust comparison of the $F_{0.5}$ performance measure value of each similarity algorithm, we use the k-fold cross-validation technique with k set to ten following popular choice. By using cross-validation we gain insight into the possible overfitting of the parameter values. Overfitting is the fact that the optimized values for the parameters are only optimal for that specific dataset, and that the algorithm's performance degrades severely when other data are presented to the algorithm

With 10-fold cross-validation, our dataset is first partitioned randomly into ten approximately even-sized subsamples. On average a subsample contains 181 documents. Of the ten subsamples a single sample is retained as the validation set, and the other nine samples are used as training data. In our case, we use the training data to calculate the optimal parameters values for the similarity algorithm. On average, this results in a dataset of $1807 - 181 = 1626$ documents, leading to $\frac{1626 \cdot 1625}{2} = 1,321,125$ document pairs evaluated (the similarity algorithm is symmetric and we do not compare a document with itself). The retained subsample is used to calculate the $F_{0.5}$ performance measure with the parameter values that are optimal for the other nine samples. Each of the on average 181 documents in the subsample is compared to all the 1807 documents in the dataset, which results in comparing $\frac{2 \cdot 181 \cdot 1807 - 181^2}{2} \approx 310,686$ document pairs. This process is repeated for every subsample, and the optimal parameter values and performance measure value are averaged over the ten samples.

The standard deviation of the set of ten parameter values and $F_{0.5}$ values is a measure for the robustness of the algorithm to different data. A small standard deviation indicates that the parameter estimates are reliable and that the algorithm should perform well on unseen data. A

large standard deviation tells us that the parameter values fluctuate and that their optimal value depends greatly on the supplied dataset.

Additionally, the set of ten performance measure values supplies us with a method to examine if the difference in performance is statistically significant. We use the Student's t-test to compare the means of the $F_{0.5}$ values of the different algorithms.

6.1.3. Similarity factor selection

To select the most optimal combination of similarity factors for the C-CodeR algorithm we run a 10-fold cross validation test. Five variants of both structural similarity algorithms are used: the C-CodeR and TextTiling algorithms with their own factors, both algorithms with the factors of each other, denoted by C-CodeR$^{\text{TT}}$ and TextTiling$^{\text{CC}}$ and the C-CodeR algorithm with all six factors: C-CodeR$^{\text{all}}$. The results for the average parameter values and the $F_{0.5}$ values are are shown in Table 6.3 (standard deviation in parenthesis).

The TextTiling algorithm performing better than the C-CodeR algorithm could be a consequence of the subtopic separation method, or of the selected similarity factors. Switching parameters gives the highest performance for the C-CodeR$^{\text{TT}}$ algorithm, which justifies our choice to continue with this subtopic separation method. However, it is clear that the set of similarity factors introduced by us are not optimal. The bottom row suggests that only the factors based on the weight of the subtopics (optimal matching and median weight) and the disturbing factor have significant prediction power. The table it makes clear that the β and η parameters and their corresponding factors, text order and subtopic length, have very little prediction power. The θ parameter only obtains a significant value when no other more powerful factor is available, thus we also do not select the size difference factor for the C-CodeR algorithm. The α and ζ parameters correspond to similar measures, based on the weight of the subtopics. Head to head testing shows that the median weight factor is more powerful than the optimal matching factor, which is why we select the median weight and disturbing factor and their parameters ζ and λ as similarity factors to use in the C-CodeR algorithm. We replace the similarity measure defined in Equation 4.4 with:

$$sim(A,B) = \zeta \cdot sim_{ME}(A,B) + \lambda \cdot sim_{DF}(A,B) \tag{6.1}$$

. In our further testing we use Equation 6.1 as similarity measure for the C-CodeR algorithm. The performance of this algorithm is shown in Table 6.4. We recognize the fact that the statistical evidence for our claims is thin given the standard deviation and sample size. However, as the core of our experiments is the comparison of the C-CodeR algorithm to the baseline algorithm and 5 competing algorithms, the statistical proof for the selection of the most optimal factors is not of great importance.

6.1.4. Comparison of the similarity algorithms

In addition to our own newly developed C-CodeR algorithm, we have re-implemented six other similarity algorithms to make a solid comparison. This is necessary because none of the pre-

Similarity algorithm	Average parameter values	Average $F_{0.5}$ value
TextTiling	$t = 0.74\ (0.007)$ $\alpha = 0.69\ (0.034)$ $\beta = 0.00\ (0.002)$ $\lambda = 0.31\ (0.035)$	0.84 (0.015)
C-CodeR	$t = 0.86\ (0.022)$ $\zeta = 0.77\ (0.030)$ $\eta = 0.02\ (0.016)$ $\theta = 0.21\ (0.025)$	0.82 (0.018)
TextTiling^{CC}	$t = 0.83\ (0.020)$ $\zeta = 0.73\ (0.033)$ $\eta = 0.01\ (0.014)$ $\theta = 0.26\ (0.031)$	0.82 (0.020)
C-CodeR^{TT}	$t = 0.77\ (0.012)$ $\alpha = 0.71\ (0.036)$ $\beta = 0.02\ (0.035)$ $\lambda = 0.27\ (0.051)$	0.85 (0.010)
C-CodeR^{all}	$t = 0.79\ (0.015)$ $\alpha = 0.50\ (0.189)$ $\beta = 0.04\ (0.041)$ $\lambda = 0.18\ (0.094)$ $\zeta = 0.23\ (0.192)$ $\eta = 0.02\ (0.020)$ $\theta = 0.03\ (0.041)$	0.85 (0.012)

Table 6.3: *Cross-validation of $F_{0.5}$ values for factor selection*

viously conducted experiments on the developed similarity algorithms share a common dataset and thus the results are not easily comparable. We solve this problem by re-implementing the algorithms, and by using a representative dataset collected from our application's target environment.

Performing a 10-fold cross-validation with the seven considered algorithms results in a set of ten $F_{0.5}$ performance measure values for each similarity algorithm. These are averaged to obtain a single quantitative measure, and Table 6.4 displays the results. The average values of the optimized parameters is given, as well as their standard deviation in parenthesis. The third column shows the average value for the $F_{0.5}$ performance measure, again with standard deviation in parenthesis. We use the Student's t-test to compare the mean of $F_{0.5}$ performance measure value of an algorithm with the mean of the $F_{0.5}$ value of the cosine similarity algorithm, which is considered the baseline algorithm, which all similarity algorithms should outperform. Since the Student's t-test assumes normality of data we test for this with the χ^2 goodness-of-fit test and create normal probability plots for all the $F_{0.5}$ distributions. The results are positive in that the data appears to be normally distributed. For details we refer the reader to Appendix C.

The comparison is performed with a paired Student's t-test to test whether the means of the two distributions of $F_{0.5}$ values are equal. More specifically, we test if the mean of the performance measure is larger than the mean of the performance measure of the cosine similarity measure, thus we perform a one-tailed Student's t-test with:

$$\begin{array}{lcll} \mathrm{H}_0 & : & \mu & = & \mu_{cosine} \\ \mathrm{H}_1 & : & \mu & > & \mu_{cosine} \\ \mathrm{H}_\mathrm{T} & : & \mu & \leqq & \mu_{cosine} \end{array}$$

, where H_T is the true null hypothesis. The fourth column shows the probability value that the (true) null hypothesis is true. In the last column we display the result of again a Student's t-test, but this time we test whether the C-CodeR similarity algorithm performs better than the algorithm under consideration. Thus the hypotheses are:

$$\begin{array}{lcll} \mathrm{H}_0 & : & \mu_{ccoder} & = & \mu \\ \mathrm{H}_1 & : & \mu_{ccoder} & > & \mu \\ \mathrm{H}_\mathrm{T} & : & \mu_{ccoder} & \leqq & \mu \end{array}$$

. The displayed probability value is the probability that the null hypothesis (the C-CodeR algorithm performing worse than the considered algorithm) is true.

It is wise to give some hints on what we expect to see in this table and how to interpret the results. We expect every algorithm to outperform the cosine similarity algorithm, and thus that the probability value in the fourth column is less then the confidence level of $\alpha = 0.05$. Furthermore, we hope to see that the C-CodeR algorithm developed by us is an improvement over every other algorithm, resulting in a rejection of the null hypothesis. Again, we expect to see a probability value of less than 0.05 in the last column.

Similarity algorithm	Average parameter values	Average $F_{0.5}$ value	p_{cosine}	p_{ccoder}
Cosine	$t = 0.84$ (0.002)	0.88 (0.019)	-	0.999
Identity	$t = 0.80$ (0.016)	0.80 (0.016)	1.000	0.000
Scam	$t = 0.99$ (0.000)	0.21 (0.016)	1.000	0.000
Signature extraction	$t = 0.27$ (0.020)	0.82 (0.018)	1.000	0.000
Bzip2	$t = 0.57$ (0.004)	0.82 (0.015)	1.000	0.000
TextTiling	$t = 0.74$ (0.007) $\alpha = 0.69$ (0.034) $\beta = 0.00$ (0.002) $\lambda = 0.31$ (0.035)	0.84 (0.015)	1.000	0.000
C-CodeR	$t = 0.80$ (0.012) $\zeta = 0.69$ (0.026) $\lambda = 0.31$ (0.026)	0.87 (0.016)	0.999	-

Table 6.4: *Cross-validation of $F_{0.5}$ values for algorithm comparison*

Similarity algorithm	$F_{0.5}$ value
Cosine	0.91
Identity	0.85
SCAM	0.26
Signature extraction	0.87
Bzip2	0.87
TextTiling	0.88
C-CodeR	0.90

Table 6.5: *$F_{0.5}$ performance measure value with average parameters*

6.1.5. Other evaluation methods

We use the average parameter values calculated with the 10-fold cross validation technique to evaluate the similarity algorithms with the remaining methods. To be complete we first recalculate the $F_{0.5}$ performance measure with the average optimized parameter values on the entire dataset, see Table 6.5. To clarify what the actual classification of the document pairs is, we construct the confusion tables for four of the best performing algorithms, depicted in Figure 6.4 through Figure 6.7 (the confusion table of the TextTiling algorithm is similar to the one of the C-CodeR algorithm).

To illustrate the behavior of the similarity algorithms over the complete threshold range, we calculate the $F_{0.5}$ performance measure with the average optimized parameter values for each threshold value (with intervals of 0.005). The result is depicted in Figure 6.8. The vertical lines are drawn at the average optimal threshold value (see Table 6.4, second column).

Another method to compare the similarity algorithms is to construct the ROC curves, see Section 4.3.3. Again we increase the threshold value from 0 to 1, with steps of 0.005. We calculate the false positive and true positive rate at every threshold value, resulting in the ROC curves as shown in Figure 6.9. Most of the lines lie close to each other in the top left part of the plot, enclosed by the gray dotted rectangle. We zoom in on this part in Figure 6.10.

6.1.6. Interpretation of the results

$F_{0.5}$ performance measure From Table 6.4 it is immediately clear that our assumption that every algorithm outperforms the baseline algorithm is wrong. The cosine similarity algorithm has its roots in the VSM and has been around for a long time. Almost all of the similarity algorithms from the field of information retrieval build on the cosine measure and claim to improve it. The fingerprinting algorithms are said to be a (better) alternative to the state-of-the-art information retrieval algorithms and therefor should also be an improvement over the

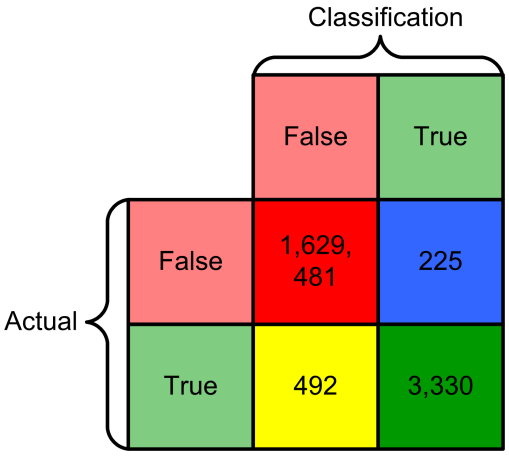

Figure 6.4: *Confusion table for the Cosine algorithm*

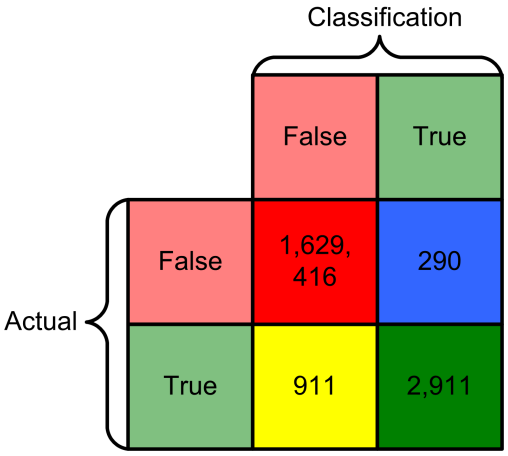

Figure 6.5: *Confusion table for the Identity algorithm*

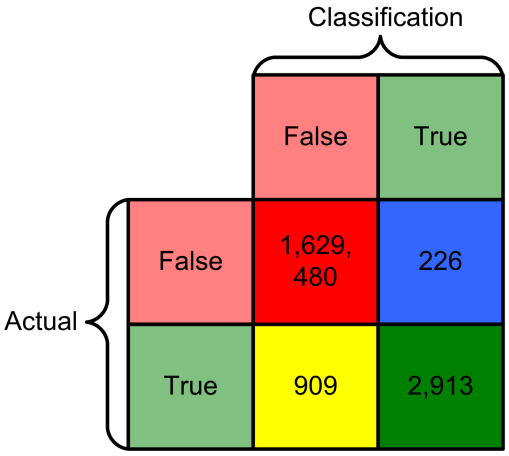

Figure 6.6: *Confusion table for the Signature extraction algorithm*

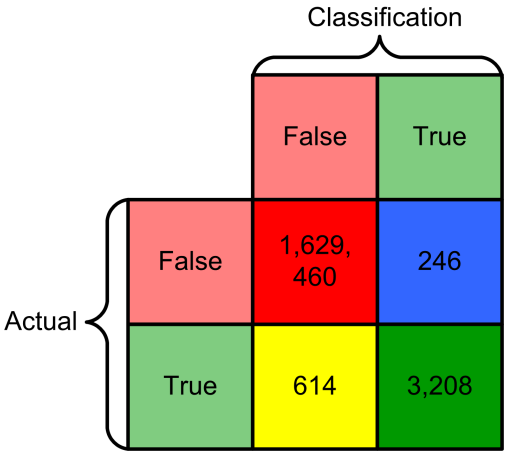

Figure 6.7: *Confusion table for the C-CodeR algorithm*

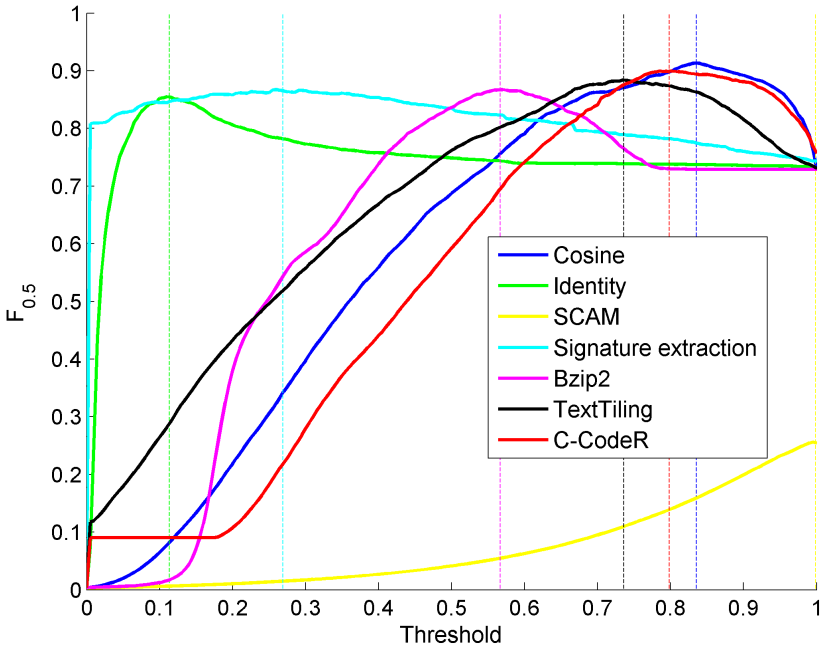

Figure 6.8: *$F_{0.5}$ value over the complete threshold range*

baseline case. However, when it comes to recognizing coderivative documents, they are all clearly outperformed by their ancestor, the cosine similarity algorithm. The fourth column of the table shows that the probability the cosine similarity algorithm performs better than the evaluated algorithm. As this value is always close to 1, there is absolutely no statistical evidence to reject the null hypothesis that any algorithm outperforms the baseline algorithm.

We are not only interested in comparing the algorithms against the baseline case, but also in the performance of our own similarity algorithm, the C-CodeR algorithm. The fifth column of Table 6.4 displays the probability that the C-CodeR algorithm performs worse than the evaluated algorithm. For every algorithm that has a p value in this column below the confidence level $\alpha = 0.05$ we can reject the null hypothesis and claim that the C-CodeR algorithm outperforms the algorithm under consideration. From this column it is clear that our algorithm, although still not outperforming the baseline one, is a useful addition to the field of coderivative document recognition. Its performance is better than any competing algorithm, and almost equal to the performance of the baseline cosine similarity algorithm (especially given the standard deviation). Both of the selected similarity factors makes a solid contribution to the overall calculated similarity, given the parameter values.

The performance of the bzip2 algorithm is remarkable. This crude and simple method of file compression rivals even the more advanced techniques employed by for example the identity and signature extraction algorithms. It should be noted that, although the method it simple, the

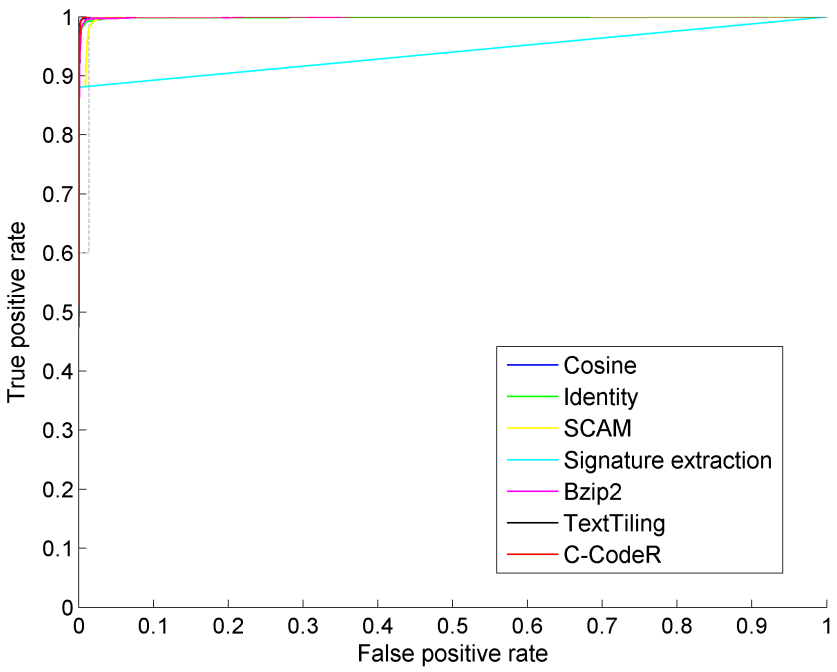

Figure 6.9: *ROC curves of the similarity algorithms*

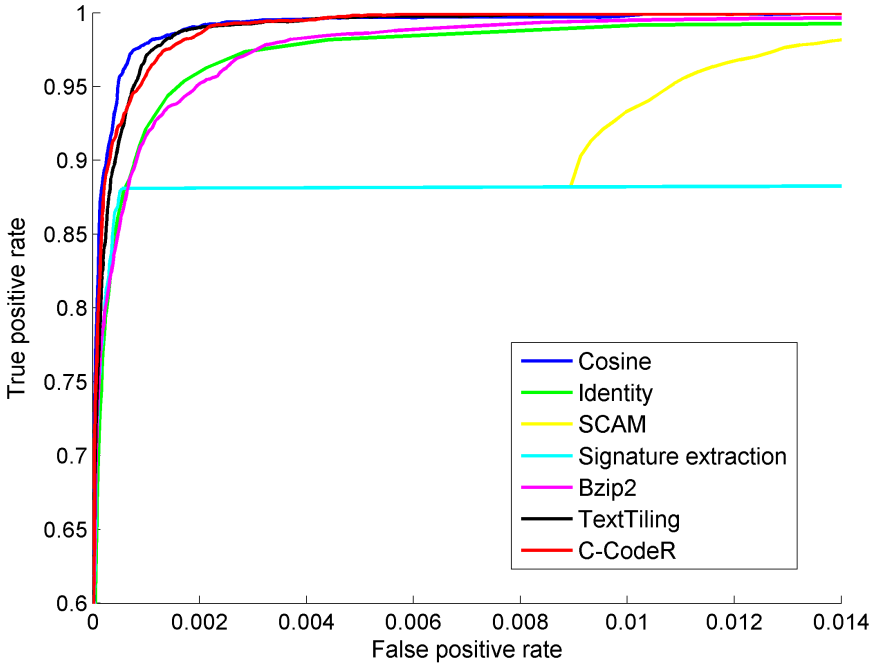

Figure 6.10: *ROC curves of the similarity algorithms, zoomed in*

time performance is the worst of all algorithms. File compression is a time consuming operation, requiring much CPU power.

One of the other similarity algorithms coming close to the cosine performance is the signature extraction algorithm. This algorithm actually is similar to the algorithms based on the notion of structural similarity (TextTiling and C-CodeR) in that the text of a document is also split into parts and a overlap measure is calculated for matching parts between documents. Instead of relying on semantic interpretation of the text, splitting at topic or paragraph breaks, the signature extraction algorithm divides the text into chunks based on the hash value of a word. The number of chunks that occur in both documents positively contributes to the similarity score.

Confusion tables From the confusion tables given in Figure 6.4 through Figure 6.7 it follows that the C-CodeR, identity and signature extraction algorithm are approximately equally successful in reducing the number of false positives. The cosine similarity algorithm is better able to reduce the number of false negatives for the same amount of false positives. The identity and signature extraction algorithms have a much higher false negative count, while the C-CodeR algorithm scores between these two algorithms and the cosine algorithm.

Graphical evaluation methods Figure 6.8 shows a peculiar behavior of the signature extraction and identity algorithms compared to the other algorithms. While most of the evaluated algorithms have a clear peak in performance at a certain threshold value, these two algorithms perform well over a broad range of values. This is the result of both algorithms assigning a similarity score of 0 to many document pairs. The pixel renderings in Appendix E further illustrate this behavior. Most of the pixel renderings are dominated by a wide range of gray values, indicating that the similarity scores across the document set differ greatly. The pixel renderings of the signature extraction and identity similarity algorithms are mainly white, which corresponds with a large number of similarity scores being equal to 0. This is set out in more detail in the probability staircase graphs in Appendix D. From these graphs it is clear that many non-coderivative document pairs have the minimal similarity score of 0, but at a high cost: a large number of actual coderivative document pairs also are assigned a similarity score of 0, resulting in many false negatives.

The reason for the line corresponding with the SCAM similarity algorithm not ending at the origin is the large number of false positives being assigned the value of 1. At the threshold value of 1, the false positive rate does not reach the value of 0. Many of the lines do not reach a true positive rate of 0 at the threshold value of 1 (i. e, end on the x axis), although this is only clearly visible for the SCAM similarity ROC curve in the zoomed in part of the graph, see Figure 6.10. The reason for this is that we have chosen to classify documents as coderivative with a similarity score larger than *or equal to* the threshold value. As a result the true positive rate does not obtain the value of 0 at a threshold value of 1.

The straight line segment of the ROC curve of the signature extraction similarity algorithm can be explained by the fact that this algorithm assigns a large number coderivative documents the

similarity score of 0. As the threshold is increasing slightly (in our case by 0.005) the true positive rate drops considerately. This classification error is again clearly visible in the probability staircase graph in Appendix D. From the ROC curves we can conclude that the cosine similarity algorithm is superior to all other algorithms for most of the threshold range. The curves of the C-CodeR and TextTiling algorithm approach the curve of the cosine similarity algorithm. The C-CodeR algorithm dominates the TextTiling algorithm when it comes to reducing the number of false positives, depicted at the left side of the curves, near the y axis.

Summary In conclusion we can say that our C-CodeR algorithm outperforms all other algorithms, except for one. This algorithm is the baseline consine algorithm, and is actually the most simple one. However, the C-CodeR algorithm's performance comes close to the cosine performance without having to compromise with a large number false negatives, such as the identity and signature extraction algorithms. Furthermore, the C-CodeR algorithm is superior to the TextTiling algorithm when it comes to reducing the number of false positives, which is deemed more important in our application of the algorithm.

6.2. Cayman system

In addition to verification of the performance of our similarity algorithm, we strive to validation of the Cayman system as a whole. Since the initial results of coderivative recognition are positive, Capgemini has decided to start a long term experiment with the prototype software developed by us. A virtual server machine is made available to us running Microsoft Windows Server 2003, operating with a Intel Pentium Xeon at 3.0GHz with 512MB of internal memory (which is a bit low, but it was all that was quickly available to us).

We proceed to install Alfresco Community 2.9B and all its required software as described in Appendix B. The Cayman module is loaded into the Alfresco distribution and Alfresco is registered as a service on the system. The next step is to install the Cayman Collector on notebook computers of employees of the ECM & UX technology practice at Capgemini. To be able to monitor the experiment closely we follow the so-called oil spot strategy to distribute the Cayman Collector. We start by selecting very few people to take part in the experiment. This keeps the initial phase of the experiment and the results surveyable and we are able to manually inspect the log files of both the Cayman Collector and the Alfresco module to verify the correct functioning of the system. After this initial phase we adopt a controlled spreading approach, by selecting more and more people to participate. Once the system has proven to be reliable we can inform the participants that they may further spread the Cayman Collector. The idea is that a casual demonstration of the usefulness of the system will incite other employees to install the application as well. By taking this guerrilla marketing approach instead of imposing a set of procedures to be followed we hope to overcome a possible cultural barrier of reluctance to participate and share knowledge.

As this is an on-going experiment, definite results are not yet available. After inspection and analysis of results of the first two weeks of the experiment, we can conclude that after some initial bug fixes the software appears to be functioning properly. Once the Cayman Collector has spread across the ECM & UX practice, the focus can be directed more to validation of the system. Interviews can be held with various people to examine if the Cayman Collector does not pose a burden on the performance of their notebook computer. Usability tests can be held to validate that the documents are properly indexed and that the search facility in Alfresco is adequate. Most importantly, employees can verify that their coderivative documents are recognized as they should be and that the search results of the Alfresco application are not cluttered with these documents.

As said previously, much of the work on this part of the experiment still lies in the future. The initial results are positive, but further testing and possibly improvement might be necessary.

Chapter 7.

Conclusions and Recommendations

7.1. Conclusions

Research question The results of our experiments have shown that although the performance of the C-CodeR algorithm is not yet optimal, the ability to recognize coderivative documents appears to be successful enough to be of practical use in a professional knowledge management environment. It is capable of preventing the knowledge management system from flooding with coderivative documents. By automating document collection, the time investment required of the employee is minimal and a possible cultural barrier is overcome. We can therefor supply a positive answer to our research question:

> **Yes, it is possible to automatically recognize coderivative documents**.

Objective 1: Select the best algorithm We have conducted a thorough research into the field of coderivative document recognition to select the best algorithm as the basis for our new coderivative document recognition algorithm. We found only one algorithm especially designment for coderivative document recognition, but selected another more promising algorithm based on the structure of a document. The results of the experiments confirm that our choice for the structural algorithm is justified, as our algorithm surpasses all others, except for one.

Objective 2: Select comparison methods An important contribution made by us is an extensive comparison of state-of-the-art algorithms that are applied to recognize coderivative documents. Unlike in much previously conducted research, we have evaluated them with a representative dataset acquired from a real life environment. By manually creating a truth list for this dataset, we were able to use a quantitative performance measure to evaluate the algorithms. Furthermore, we have studied several other useful graphical evaluation methods to gain insight into the behavior of a similarity algorithm.

Objective 3: Design and implement a new algorithm We have developed an effective algorithm that adds to the small number of coderivative document recognition options currently available: the C-CodeR algorithm. It is built on a structural similarity algorithm we studied in our research into related work. Although it shares the basis of taking the structure of a document

into account, we have modified it substantially and have shown that it outperforms the original algorithm.

Objective 4: Validate the performance of the new algorithm Combining the truth list with the robust performance measure made clear that almost all competing similarity algorithms fall short in recognizing coderivative documents. Even more surprising, the baseline algorithm still outperforms all others, even our own C-CodeR algorithm. However, it is our firm belief that the cosine algorithm will fail where our C-CodeR algorithm that takes the structure of a document into account will succeed. Although our dataset was quite substantial, it diminishes in comparison to the amount of documents in an actual knowledge management system placed in a professional environment. In such a system the sheer number of documents will lower the performance of the simple cosine similarity algorithm considerably, since it will contain many documents which, although similar in topic, are not coderivative. A more complex algorithm taking into account the structure of a document will benefit from this situation. We have shown that the C-CodeR algorithm ranks at the top and is superior to all other complex algorithms.

During the manual inspection of the dataset on coderivatives it became apparent that not all document types are equally suitable for coderivative recognition. Many document types, such as progress reports, meeting agenda and notes, function evaluations and legal contracts, adhere to such a rigid format that the resemblance among them is large and the difference between them small. They will undoubtedly always be marked as coderivative by a similarity algorithm, unless the algorithm is able to identify these special cases.

Although the method of comparing each document with every other document in the dataset to qualify the performance of an algorithm at first sight appears to be thorough and complete, looking back we must conclude that it may underestimate the true performance of a similarity algorithm. Many of the coderivative documents in the dataset are not pairs, but form complete sets of documents starting with the initial version developing to the final release of a document. It is clear that the difference between the first and last version is large, while the incremental difference between subsequent versions is small. It can not be expected from any similarity algorithm to recognize the complete set of documents as coderivative, nor is this required. Since the documents are gathered periodically, a newly submitted document will only contain small changes compared to its previous version stored in the knowledge management system.

Objective 5: To validate the practical use of the new approach to knowledge management The Cayman system which was developed during our research lives up to its expectations. We have successfully designed and implemented a prototype application that automatically collects documents, makes them accessible and their contained knowledge reusable.

If the approach of automatic document collection provides a solution for the knowledge management of the ECM & UX technology practice of Capgemini remains to be seen, but the fact that a six month trial period has been started with the prototype software suggests that Capgemini is confident that this is the right direction. The Cayman system meets the requirements set

by the focus group in that it collects documents, makes them accessible through an off-the-shelf knowledge management system and is able to recognize and group coderivative documents. The true added value of this system for Capgemini is difficult to calculate, as the financial benefit is not immediately clear. However, because this approach takes up little or no time of the users while it still discloses knowledge to them, the investment is small and the possible advantage of increased productivity through less time spent acquiring knowledge is bound to positively contribute to the enterprise.

The idea has even arisen within the focus group to investigate if this solution approach could also be interesting for clients of Capgemini. If the trial period proves to be a success, an investment might be made to develop a mature software system with the same functionality as the current prototype: document gathering, coderivative document recognition and integration with an existing knowledge management application. In the future, Capgemini might be able to sell this approach of knowledge management as a solution to their clients.

7.2. Recommendations

There are several recommendations we can make with regard to this research project. They are grouped into three sections, each corresponding with a part of the solution approach taken to answer the research question.

C-CodeR algorithm One of the largest drawbacks of the C-CodeR algorithm is the inability to match one subtopic to several others, i. e. perform an *n*-to-*n* matching of subtopics. The Kuhn-Munkres algorithm is limited to matching one subtopic to one other subtopic. A paragraph split into two or joined together will lower the similarity score considerately. This problem is solved by Wan and Peng (2005) by using the Earth Mover's distance algorithm, and by Wan and Yang (2006) by using the Proportional Transportation Distance algorithm.

It could be investigated if using other similarity measures instead of the cosine measure to calculate the similarity between subtopics leads to an improvement in the C-CodeR and TextTiling algorithm. Wan (2008) among others mentions the Jaccard measure, the Dice measure, and Overlap measure and others as alternatives.

Finding the optimal tokenizer that analyzes and preprocesses the text is another possible improvement. We used a very complete tokenizer which incorporates a stemming method to lexically analyze the text but this might not be the best option. When trying to find topical similarity matching the stem of a word might be adequate, but a coderivative match could require a more precise match.

Cayman system The Cayman Collector application, being a prototype, leaves some room for improvement. It could be rewritten in a different programming language, such as C++ of C#. Although this might compromise the platform independence, the performance gain will be substantial and it will burden the resources of the notebook computer to a lesser degree.

Functional improvements can be made in the area of giving feedback to the user. A popup message could show how many documents were sent to the knowledge management system. A more detailed view would give an overview of these documents with the possibility to recall some of them (because, for example, they were actually private files). If an error occurred during sending the documents or processing them by the knowledge management system, the Cayman Collector could resend them or at least notify the user.

The usability of Alfresco can be greatly improved by tailoring it to the specific needs for knowledge management of the ECM & UX practice. This is the field of expertise of an Alfresco specialist who has better understanding of the possibilities of the application and how it can be improved to better facilitate the accessibility and reusability of knowledge. The interface could be improved, as well as the search and index functionality (for example, custom tokenization for the originating folder could be added). Alfresco could be integrated with the authentication services of Capgemini, in order to provide single sign-on functionality and a home space for each employee. If a document is submitted from a notebook computer of an employee, Alfresco could match the user name submitted with the document to automatically place the document in the home space of the employee.

As the coderivative recognition relies heavily on proper transformation to plain text of a document, this service of Alfresco deserves close attention. It became clear during the initial phase of the trial period that for some PDF documents the transformation produced an unreadable sequence of characters. This prevents Alfresco from properly indexing the document, rendering it irretrievable, and additionally interferes with the coderivative recognition algorithm. Detailed attention should be given to the correct transformation of the carriage returns, as the paragraph division method of the C-CodeR algorithm depends on it.

A more substantial improvement could be the dynamic classification of documents in subject groups with semantic taxonomy extracted from the documents. Tags could automatically be added to the documents to improve document disclosure. Another radical addition could be to incorporate a push technology. When an employee creates a new document or opens an existing document the system could match it with existing documents in the knowledge management system and push possibly interesting documents to the employee.

Knowledge management A general recommendation we can make with regard to the field of knowledge management is that from our point of view it is certainly worth while experimenting with automatic document collection. A more thorough user test can be performed to analyze the practical use of such a system. This could include a broader employee participation coupled with a survey to verify that the coderivative documents are recognized properly, that the system makes the documents accessible and reusable and that the system increases productivity by saving time.

Bibliography

AYER, A. *The Problem of Knowledge. An Enquiry into the Main Philosophical Problems that Enter into the Theory of Knowledge*. Penguin, Harmonsworth, Middlesex, England, 1956.

BECHINA, A. and T. BOMMEN. "Knowledge Sharing Practices: Analysis of a Global Scandinavian Consulting Company". *The Electronic Journal of Knowledge Management*, volume 4 (2), pp. 109–116, 2006.

BRIN, S., J. DAVIS and H. GARCÍA-MOLINA. "Copy detection mechanisms for digital documents". In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pp. 398–409. ACM Press New York, NY, USA, 1995.

BRODER, A., S. GLASSMAN, M. MANASSE and G. ZWEIG. "Syntactic clustering of the web". *Computer Networks and ISDN Systems*, volume 29 (8-13), pp. 1157–1166, 1997.

CHOI, F. "JTextTile: A free platform independent text segmentation algorithm". 1999. Http://www.cs.man.ac.uk/ choif.

CHOI, F. "Advances in domain independent linear text segmentation". *ACM International Conference Proceeding Series*, volume 4, pp. 26–33, 2000.

CHOWDHURY, A., O. FRIEDER, D. GROSSMAN and M. C. MCCABE. "Collection statistics for fast duplicate document detection". *ACM Transactions on Information Systems*, volume 2, pp. 171–191, April 2002.

CLEARY, J. and W. TEAHAN. "Unbounded Length Contexts for PPM". *The Computer Journal*, volume 40 (2 and 3), pp. 67–75, 1997.

CRUZ, I., S. BORISOV, M. MARKS and T. WEBB. "Measuring Structural Similarity Among Web Documents: Preliminary Results". In *Proceedings of the 7th International Conference on Electronic Publishing, Held Jointly with the 4th International Conference on Raster Imaging and Digital Typography: Electronic Publishing, Artistic Imaging, and Digital Typography*, pp. 513–524. Springer-Verlag London, UK, 1998.

DEUTSCH, P. "RFC1952: GZIP file format specification version 4.3". *Internet RFCs*, 1996.

FINKEL, R., A. ZASLAVSKY, K. MONOSTORI and H. SCHMIDT. "Signature extraction for overlap detection in documents". In *Proceedings of the twenty-fifth Australasian conference*

on Computer science, volume 4, pp. 59–64. Australian Computer Society, Inc. Darlinghurst, Australia, Australia, 2002.

GUPTA, A. and S. MICHAILOVA. "Knowledge Sharing in Knowledge-Intensive Firms: Opportunities and Limitations of Knowledge Codification". Technical report, Working Paper, Centre for Knowledge Governance, NA, 2004.

HEARST, M. A. "Multi-paragraph segmentation of expository text". In *32nd. Annual Meeting of the Association for Computational Linguistics*, pp. 9–16. 1994.

HOAD, T. and J. ZOBEL. "Methods for identifying versioned and plagiarized documents". *Journal of the American Society for Information Science and Technology*, volume 54 (3), pp. 203–215, 2003.

HOOS, M. "Practice what you Preach", June 2007.

HUSTED, K. and S. MICHAILOVA. "Diagnosing and Fighting Knowledge-Sharing Hostility". *Organizational Dynamics*, volume 31 (1), pp. 60–73, 2002.

KUHN, H. W. "The Hungarian method for the assignment problem". *Naval Research Logistic Quarterly*, volume 2, pp. 83–97, 1955.

LEVENSHTEIN, V. "Binary Codes Capable of Correcting Deletions, Insertions and Reversals". *Soviet Physics Doklady*, volume 10, p. 707, 1966.

LOPRESTI, D. "Models and algorithms for duplicate document detection". In *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, pp. 297–300. 1999.

LUSTED, L. "Signal Detectability and Medical Decision-Making". *Science*, volume 171 (3977), pp. 1217–1219, 1971.

MANBER, U. "Finding similar files in a large file system". In *Proceedings of the USENIX Winter 1994 Technical Conference*, pp. 2–2. USENIX Association Berkeley, CA, USA, 1994.

MUNKRES, J. "Algorithms for the assignment and transportation problems". *Journal of the Society for Industrial and Applied Mathematics*, volume 5 (1), pp. 32–38, 1957.

NIERMAN, A. and H. JAGADISH. "Evaluating Structural Similarity in XML Documents". In *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002)*, pp. 61–66. 2002.

O'LEARY, D. E. "Enterprise knowledge management". *Computer*, volume 31 (3), pp. 54–61, 1998. ISSN 0018-9162. doi:http://dx.doi.org/10.1109/2.660190.

POLANYI, M. *Personal Knowledge: Towards a Post-Critical Philosophy*. Routledge, 1958.

PRIOR, A. "Correspondence Theory of Truth". *Encyclopedia of Philosophy*, volume 2, pp. 223–32, 1967.

PROVOST, F. and T. FAWCETT. "Analysis and visualization of classier performance: Comparison under imprecise class and cost distributions". In *Third International Conference on Knowledge Discovery and Data Mining*, pp. 43–48. 1997.

PROVOST, F., T. FAWCETT and R. KOHAVI. "The case against accuracy estimation for comparing induction algorithms". In *Machine Learning: Proceedings of the Fifteenth International Conference*, pp. 445–453. 2001.

RABIN, M. "Fingerprinting by random polynomials". Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

RIJSBERGEN, C. J. V. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 1979. ISBN 0408709294.

SALTON, G., A. WONG and C. YANG. "A vector space model for automatic indexing". *Communications of the ACM*, volume 18 (11), pp. 613–620, 1975.

SEWARD, J. http://www.bzip.org.

SHIVAKUMAR, N. and H. GARCÍA-MOLINA. "SCAM: A copy detection mechanism for digital documents". In *Proceedings of the Second Annual Conference on the Theory and Practice of Digital Libraries*. 1995.

SHIVAKUMAR, N. and H. GARCÍA-MOLINA. "Building a scalable and accurate copy detection mechanism". In *Proceedings of the first ACM international conference on Digital libraries*, pp. 160–168. ACM Press New York, NY, USA, 1996.

SWETS, J. "Measuring the accuracy of diagnostic systems". *Science*, volume 240 (4857), pp. 1285–1293, 1988.

WAGNER, R. and M. FISCHER. "The String-to-String Correction Problem". *Journal of the ACM*, volume 21 (1), pp. 168–173, 1974.

WAN, X. "Beyond topical similarity: a structural similarity measure for retrieving highly similar documents". *Knowledge and Information Systems*, volume 15 (1), pp. 55–73, 2008.

WAN, X. and Y. PENG. "The earth mover's distance as a semantic measure for document similarity". In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pp. 301–302. ACM Press New York, NY, USA, 2005.

WAN, X. and J. YANG. "Using Proportional Transportation Distances for Measuring Document Similarity". *Advances in Information Retrieval: 28th European Conference on IR Research, ECIR 2006, London, UK, April 10-12, 2006: Proceedings*, 2006.

Appendix A.

Glossary

Algorithm. An algorithm is a sequence of instructions by which a problem can be solved or a decision made

Coderivative documents. Documents that originate form the same source

Cross-validation. A method of estimating predictive error. Cross validation splits that dataset into k equal-sized parts called folds (typically ten)

Dataset. A dataset is any organised collection of data or information that has a common theme

Information Retrieval. Information retrieval is the searching of structured or unstructured data to retrieve information that is of interest. The textual content is analyzed and used for retrieval

Information Technology. Information Technology applies modern technologies to the creation, management and use of information

Keyword. A significant word or phrase in a text

Knowledge. Knowledge is the justified true belief that increases an individuals capacity to take action (Ayer, 1956)

Knowledge management. Enterprise knowledge management entails formally managing knowledge resources in order to facilitate access and reuse of knowledge, typically by using advanced information technology (O'Leary, 1998)

Null hypothesis. In statistics, a null hypothesis is a hypothesis set up to be nullified or refuted in order to support an alternative hypothesis. When used, the null hypothesis is presumed true until statistical evidence, in the form of a hypothesis test, indicates otherwise

Open Source Definition. The Open Source Definition is used to determine if a software license can be considered to be open source. The software license must meet ten conditions[1]

[1] `opensource.org/docs/osd`

Open Source Software. Open Source Software is computer software for which the human-readable source code is made available under a copyright license (or arrangement such as the public domain) that meets the Open Source Definition. This permits users to use, change, and improve the software, and to redistribute it in modified or unmodified form

Repository. A repository is a place where data are stored and maintained

Similar documents. Documents that discuss the same main topic, or at least share a certain amount of sub topics between them

Similarity measure. A similarity measure is a measure that expresses to what extend two objects are similar

Stemming. Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form

Term. A term often corresponds with a token, and can be used to analyze a text by attaching a weight value to it

Token. A token is a categorized block of text

Tokenization. In computer science, tokenization, or lexical analysis, is the process of converting a sequence of characters into a sequence of tokens

Appendix B.

Installation of the Cayman System

B.1. Client: the Cayman Collector

The Cayman Collector is an application that gathers documents from workstations and send them to a central server.

B.1.1. Requirements

- Java SE Runtime Environment (JRE) 1.5 or newer
- Windows XP, Windows Vista (untested), Linux (untested)
- Network interface (continuous network connection is not necessary)

B.1.2. Installation

Windows

The supplied file `caycoll.jar` should be placed in it's own folder on the workstation, for example in `C:\Program Files\Cayman`. To be able to start the application, create a shortcut to it by proceeding as follows.

- Right click on the file `caycoll.jar` and hold down the right mouse key
- Drag the file to an empty space within the same folder
- Release the right mouse key and select *Create Shortcuts Here*
- Rename the shortcut to `Cayman Collector`
- Right click on the created shortcut and select *Properties*
- Replace the target field by `javaw.exe -jar caycoll.jar`
- Click *OK*

Verify that the installation was successful by double clicking on the created shortcut. If the application fails to start, make sure that the `/bin` folder of the JRE is in the PATH variable.

To automatically start the application when Windows starts, create a shortcut in the Startup folder.

- Click *Start* > *Run*, enter `%userprofile%\Start Menu\Programs\Startup`, and click *OK*
- Right click on the created shortcut and hold down the right mouse key
- Drag the shortcut to the *Startup* folder and release the right mouse key
- Select *Create Shortcuts Here*

Unfortunately, Alfresco cannot transform Microsoft Word 2003 documents that are saved with the *Allow fast saves* option enabled to plain text. As the *Allow fast saves* option also allowed for a breach in security,[1] Microsoft disabled the feature as of Service Pack 3 (SP3). To guarantee optimal performance and security it is strongly advised that you disable the *Allow fast saves* option in Microsoft Word: *Tools* > *Options* > *Save* > uncheck *Allow fast saves*. Note that this only applies to Microsoft Office 2003, the 2007 edition should work properly.

Linux

To install the application on the Linux operating system, place the file *caycoll.jar* in it's own folder on the workstation and configure Linux to start the application when Linux starts up.

B.1.3. Configuration

Configuration of the Cayman Collector can be done in the Settings screen.

B.2. Server: the Alfresco Cayman module

The Cayman module for Alfresco is a module that enables Alfresco to receive documents send by the Cayman Collector. Alfresco can be downloaded from the Alfresco website[2]. The Cayman module has been tested with the Alfresco Community 2.9B release, with the Apache Tomcat server bundled.

[1] `support.microsoft.com/kb/938808`
[2] `wiki.alfresco.com/wiki/Download_Alfresco_Community_Network`

B.2.1. Requirements for Alfresco

- Java Development Kit 5.0/6.0[1]
- MySQL 5.0 Community Server[2]
- OpenOffice.org[3]

B.2.2. Requirements for the Cayman module

- Alfresco Community 2.9B or newer
- Alfresco Module Management Tool (MMT)

B.2.3. Installation

Configure the server in such a way that the Apache Tomcat server is started when the server starts up, i. e. configure it to run as a service[4]. To continue with the installation of the Cayman module, make sure that Alfresco is not running by temporarily stopping the service.

Locate the MMT (the file `alfresco-mmt-x.y.jar`) and copy the file `alfresco.module.cayman.amp` to this folder for your convenience. Open a console window, navigate to this folder and enter: `java -jar alfresco-mmt-x.y.jar install alfresco.module.cayman.amp /path/to/alfresco.war`. The file `alfresco.war` can usually be found at a location such as `C:\alfresco\tomcat\webapps`. After installation of the module into the Alfresco WAR file, it is necessary to delete the `alfresco` folder from the folder where the Alfresco WAR file is located. Restart the Alfresco service.

Alternatively, the Alfresco distribution might supply a simple batch file to install modules. If the file `apply_amps.bat` exists in the Alfresco installation folder, copy the file `alfresco.-module.cayman.amp` to the `amps/` subfolder in the Alfresco installation folder. Run the `apply_amps.bat` file to install the Cayman module and restart the Alfresco service.

[1] `java.sun.com/javase/downloads`
[2] `dev.mysql.com/downloads/mysql/5.0.html`
[3] `download.openoffice.org/index.html`
[4] `wiki.alfresco.com/wiki/Configuring_Alfresco_as_a_Windows_Service`

Appendix C.

Verifying Normality of the $F_{0.5}$ Performance Measure Values

The results of the χ^2 goodness-of-fit test for normality of the $F_{0.5}$ performance measure values calculated with 10-fold cross validation are given in Table C.1. The H_0 hypothesis is that the data are a random sample from a normal distribution with mean and variance estimated from the data, against the alternative that the data are not normally distributed with the estimated mean and variance. The second column displays if the H_0 hypothesis can be rejected at a 5% significance level.

Figure C.1 to Figure C.7 display the normal probability plots for the $F_{0.5}$ performance values calculated with 10-fold cross validation.

Similarity algorithm	Reject H_0
Cosine	False
Identity	False
Scam	False
Signature extraction	False
Bzip2	False
TextTiling	False
C-CodeR	False

Table C.1: χ^2 *goodness-of-fit test for normality*

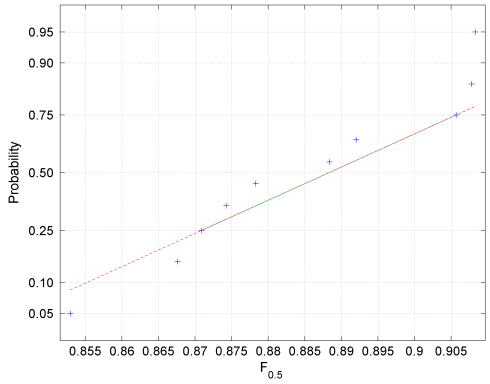

Figure C.1: *Normal probability plot for the Cosine $F_{0.5}$ values*

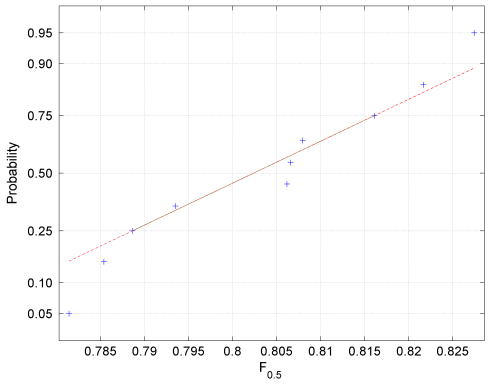

Figure C.2: *Normal probability plot for the Identity $F_{0.5}$ values*

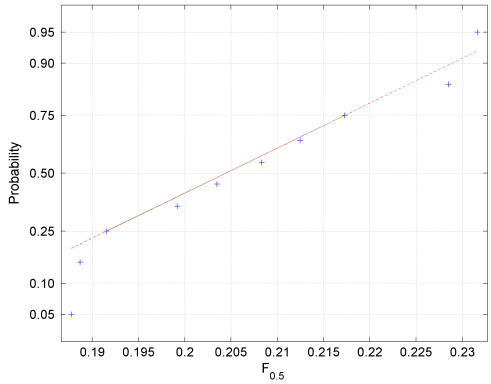

Figure C.3: *Normal probability plot for the SCAM $F_{0.5}$ values*

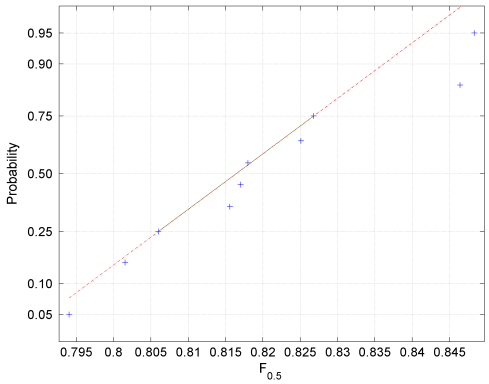

Figure C.4: *Normal probability plot for the Signature extraction $F_{0.5}$ values*

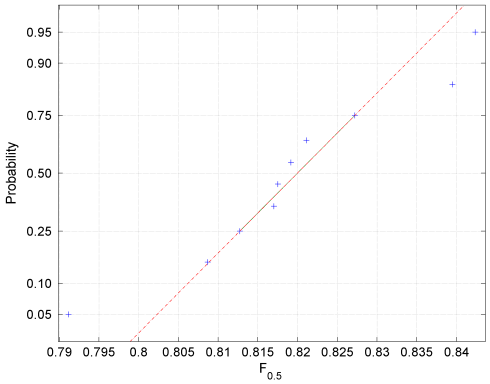

Figure C.5: *Normal probability plot for the Bzip2 $F_{0.5}$ values*

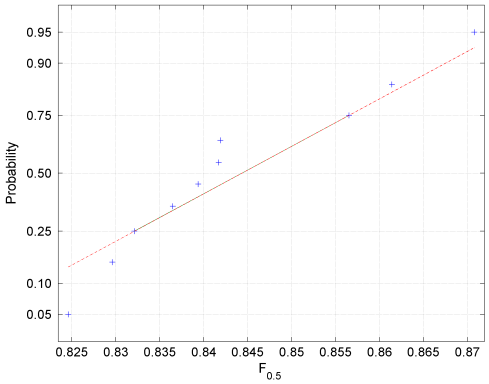

Figure C.6: *Normal probability plot for the TextTiling $F_{0.5}$ values*

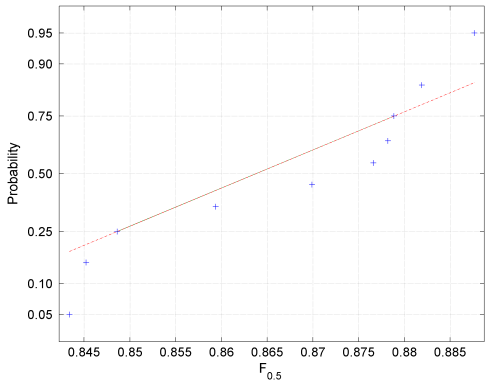

Figure C.7: *Normal probability plot for the C-CodeR $F_{0.5}$ values*

Appendix D.

Probability Staircase Graphs of the Similarity Scores

This Appendix displays the probability staircase graphs of the similarity algorithms. Such a graph is similar to the probability density functions shown as an example in Figure 4.1 on page 62. The main difference is that without a continuous function, the data in divided into bins, much like in a histogram. To counter the uneven class distribution, the values in the bins for the true and false similarity scores are divided by their class size, resulting in a percentage score. The average threshold value from Table 6.4 on page 99 is drawn as a vertical dashed line.

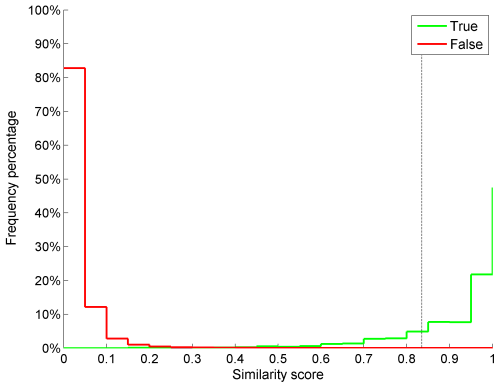

Figure D.1: *Probability staircase plot for the Cosine similarity scores*

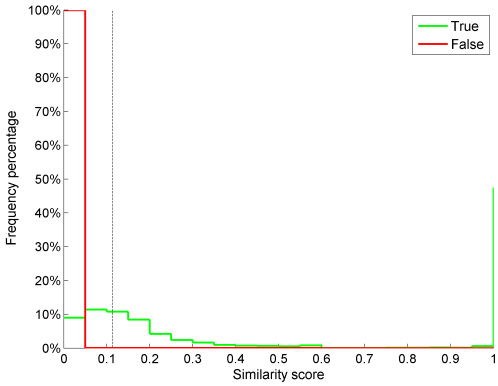

Figure D.2: *Probability staircase plot for the Identity similarity scores*

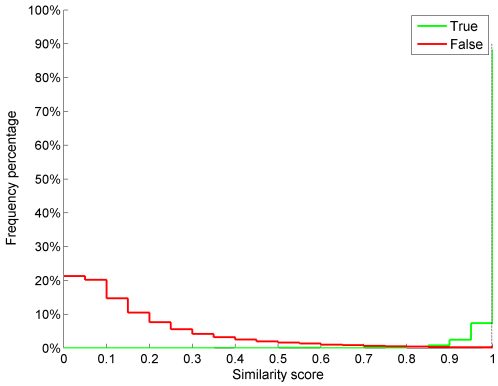

Figure D.3: *Probability staircase plot for the SCAM similarity scores*

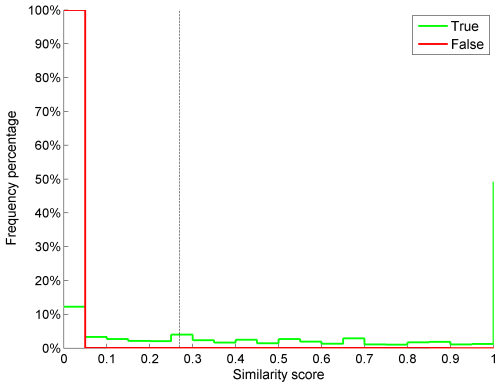

Figure D.4: *Probability staircase plot for the Signature extraction similarity scores*

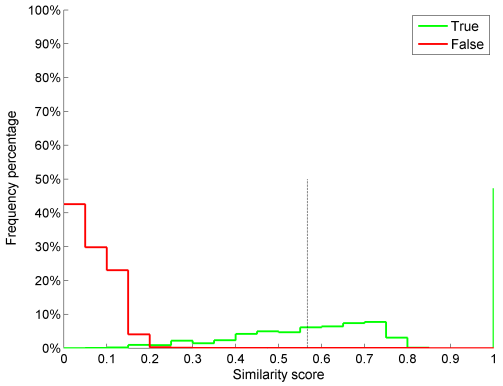

Figure D.5: *Probability staircase plot for the Bzip2 similarity scores*

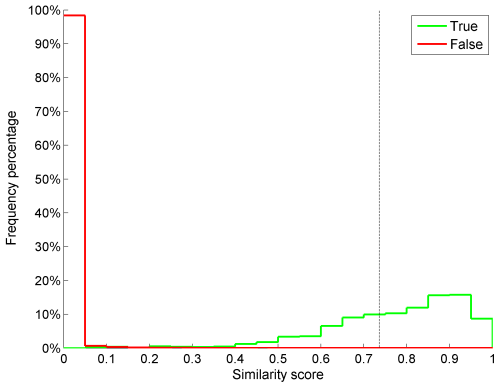

Figure D.6: *Probability staircase plot for the TextTiling similarity scores*

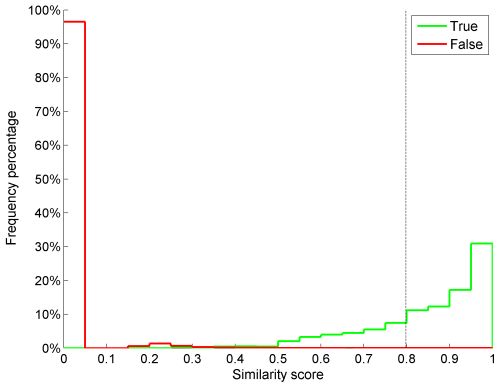

Figure D.7: *Probability staircase plot for the C-CodeR similarity scores*

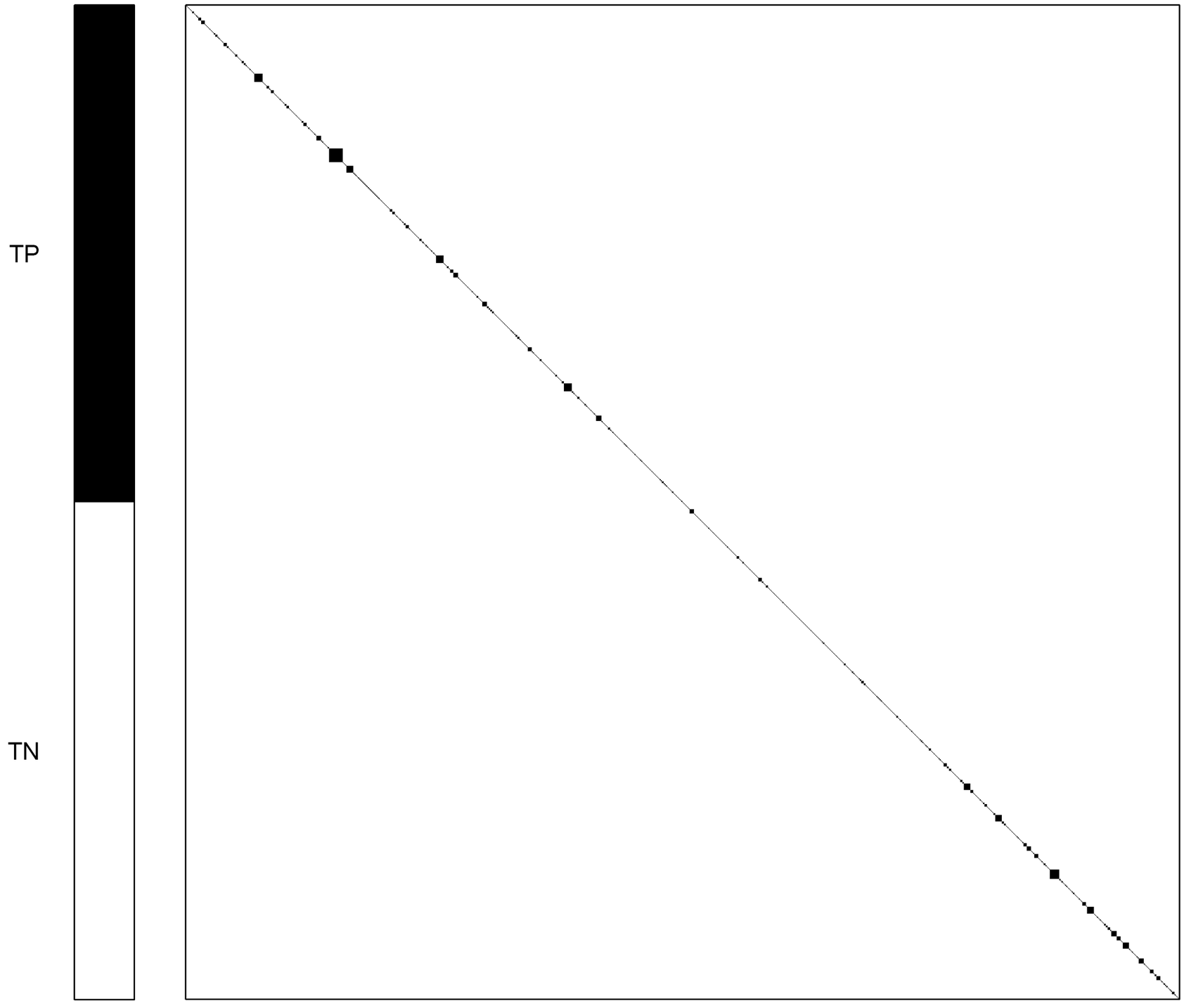

Figure E.1: *Pixel rendering, perfect classification*

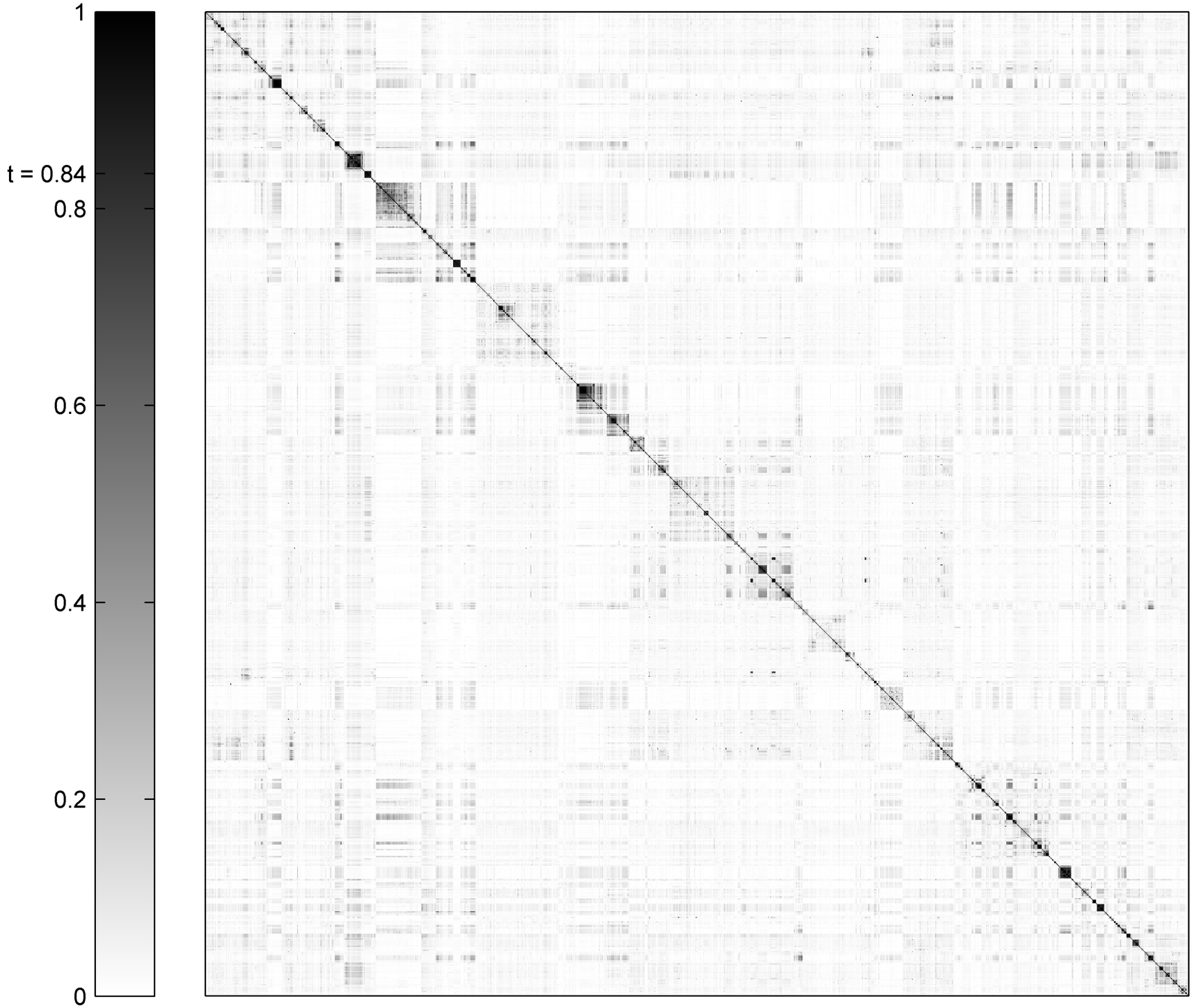

Figure E.2: *Pixel rendering for the Cosine algorithm*

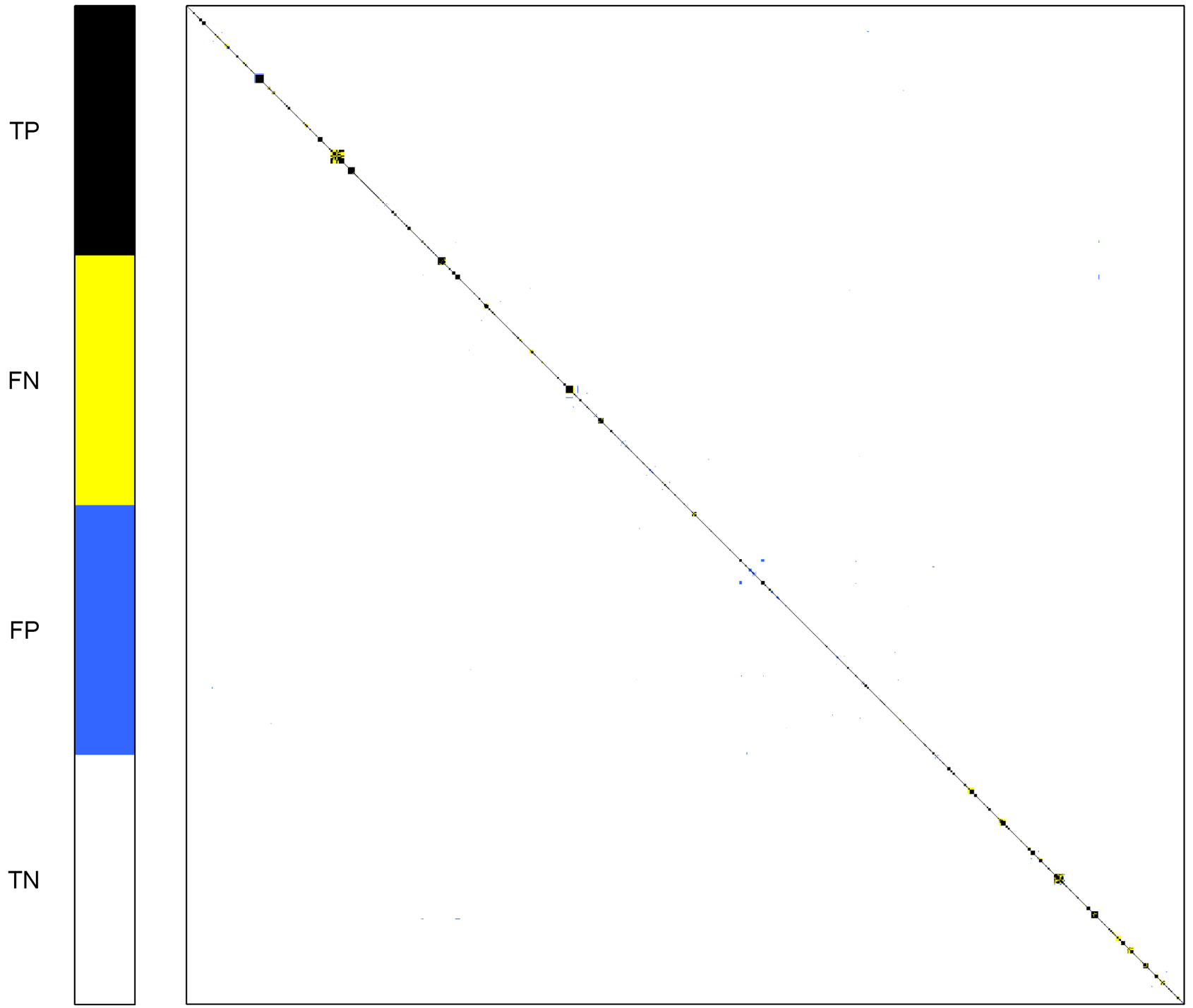

Figure E.3: *Pixel rendering for the Cosine algorithm, with threshold applied*

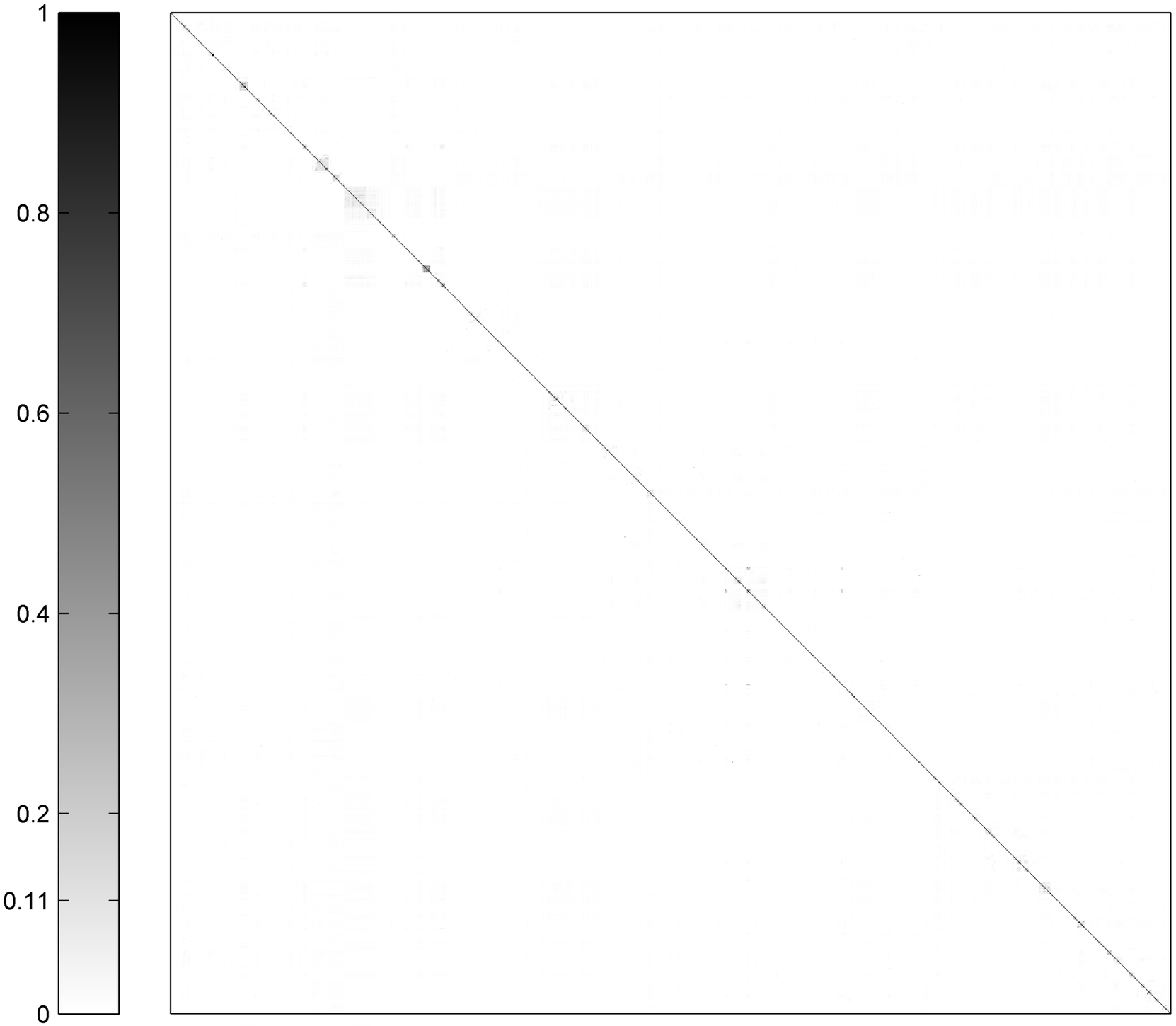

Figure E.4: *Pixel rendering for the Identity algorithm*

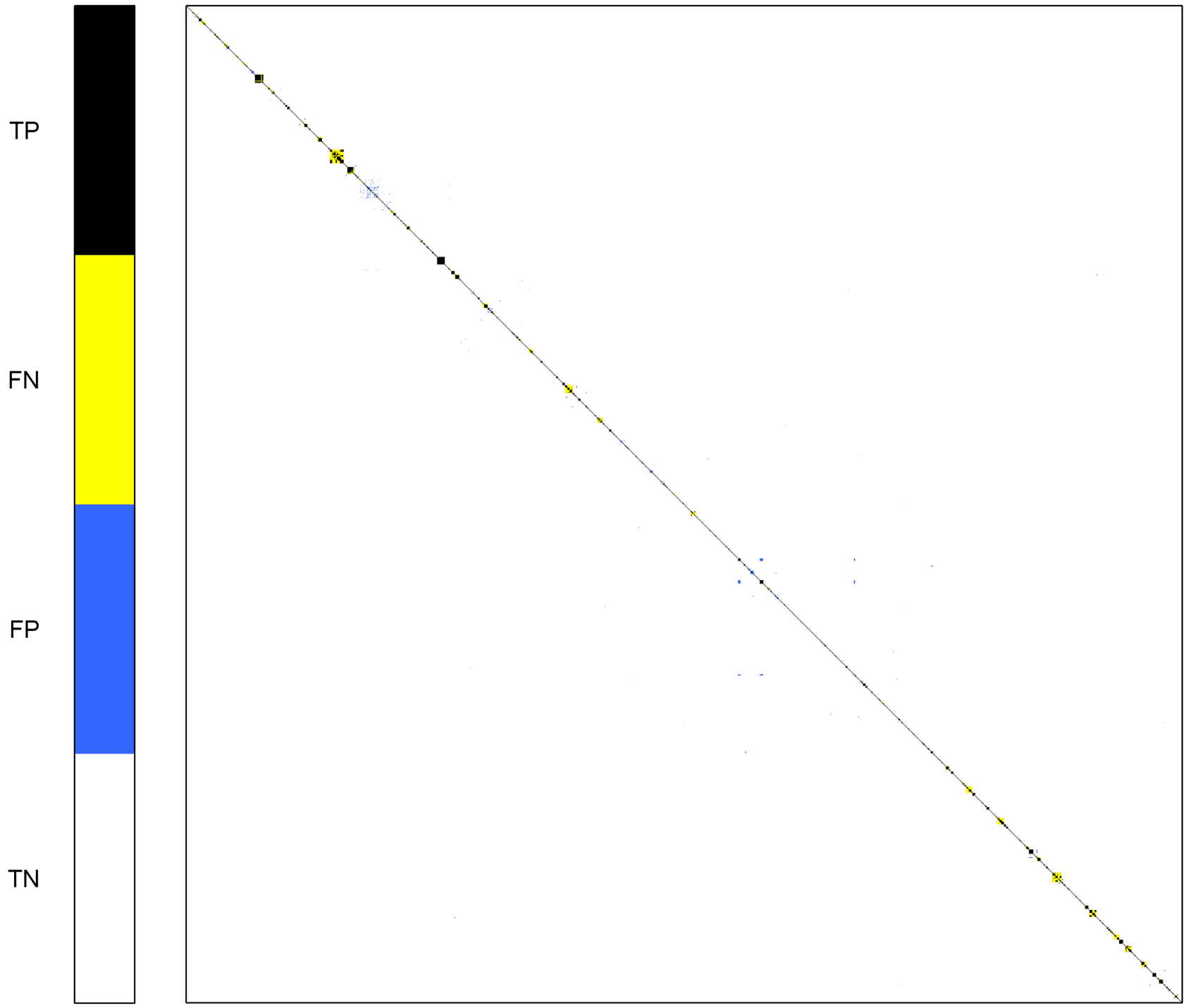

Figure E.5: *Pixel rendering for the Identity algorithm, with threshold applied*

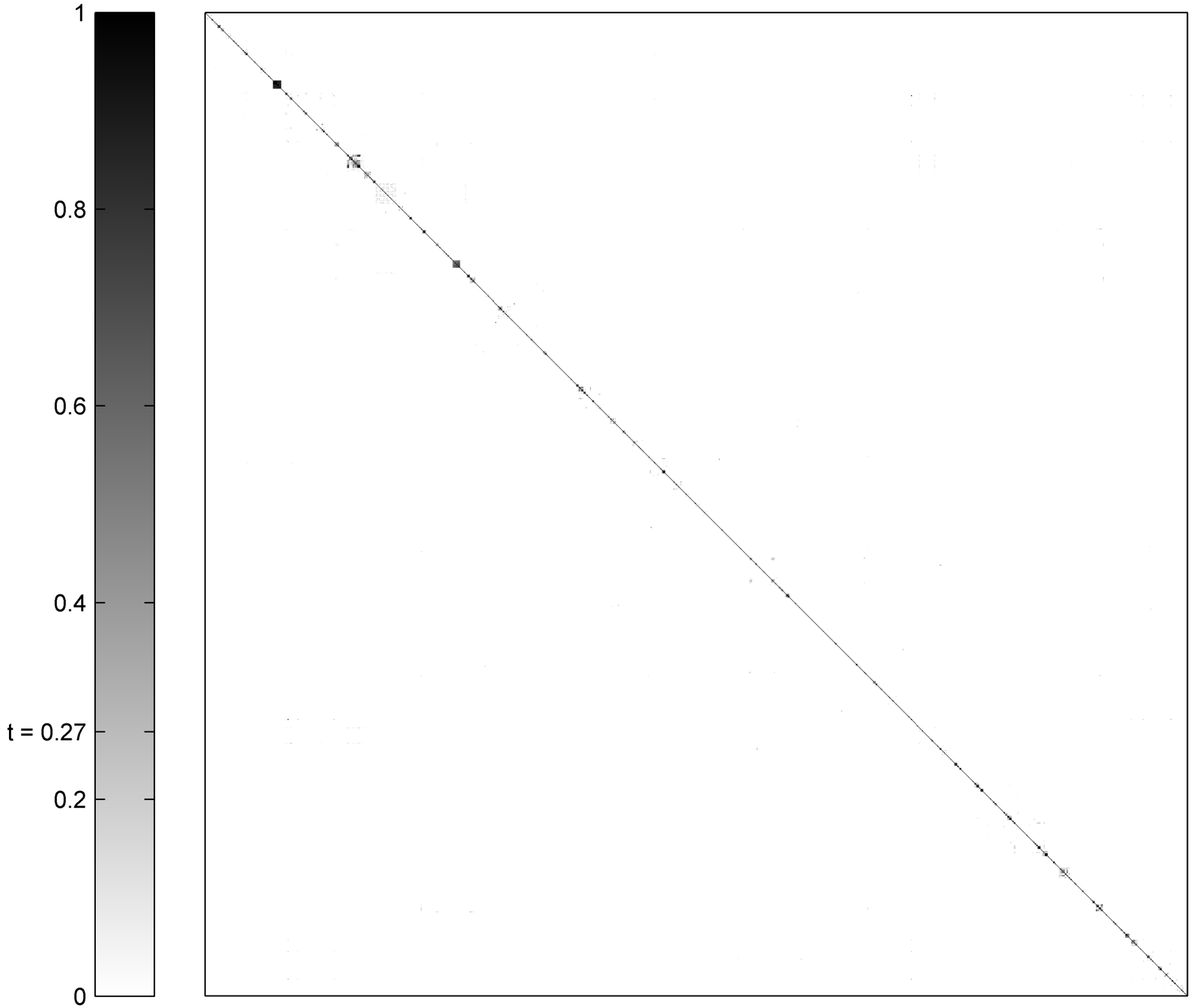

Figure E.6: *Pixel rendering for the Signature extraction algorithm*

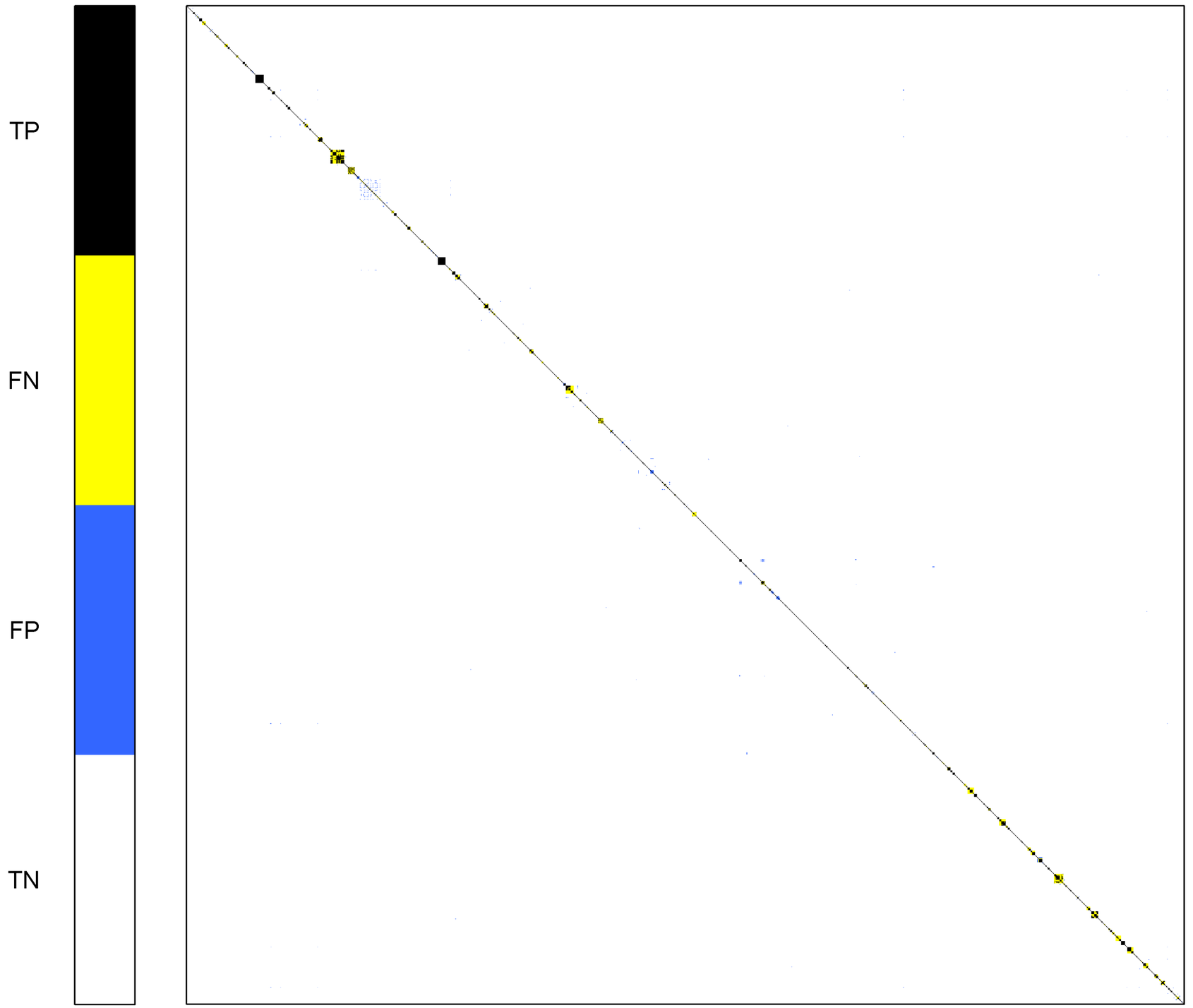

Figure E.7: *Pixel rendering for the Signature extraction algorithm, with threshold applied*

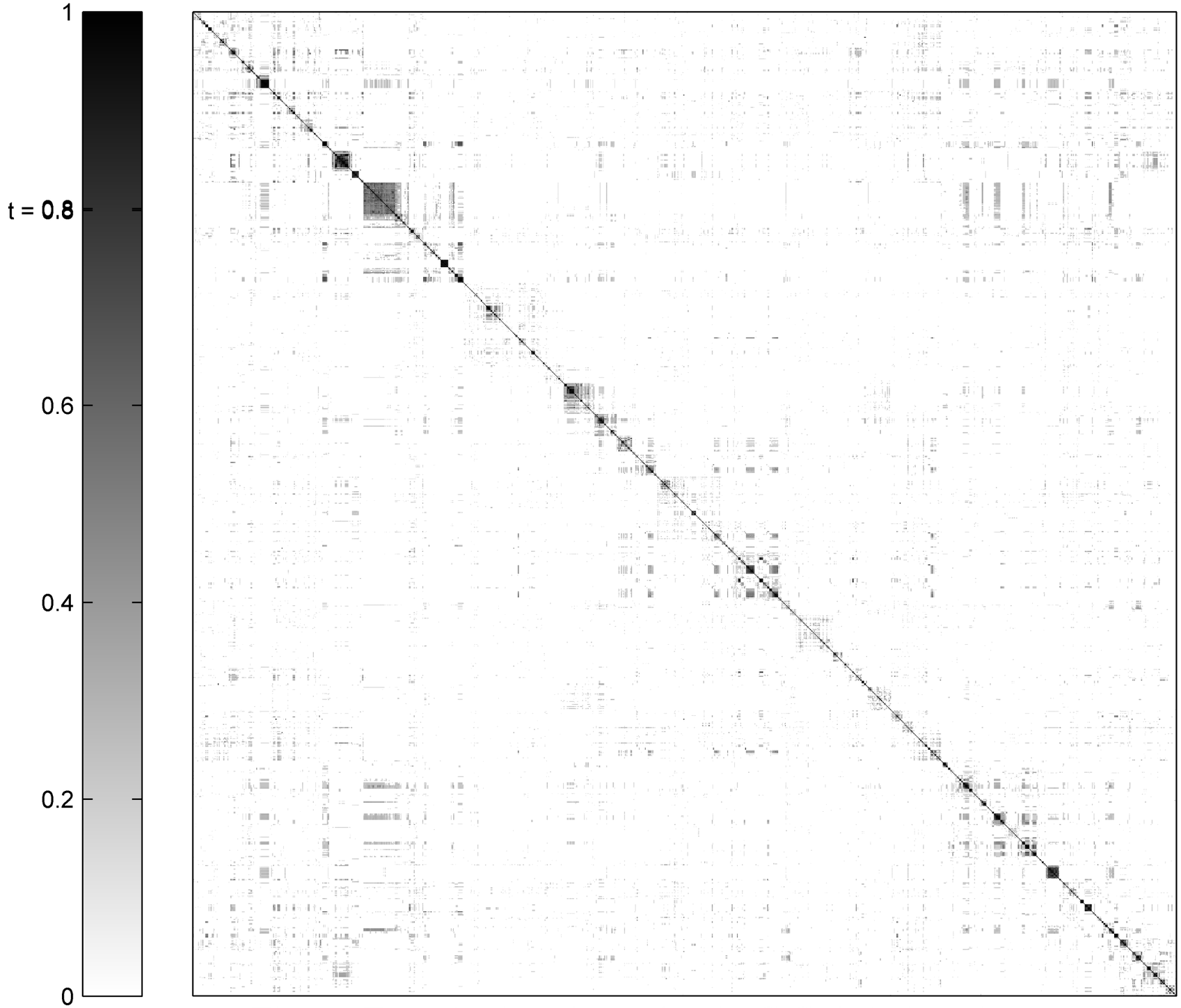

Figure E.8: *Pixel rendering for the C-CodeR algorithm*

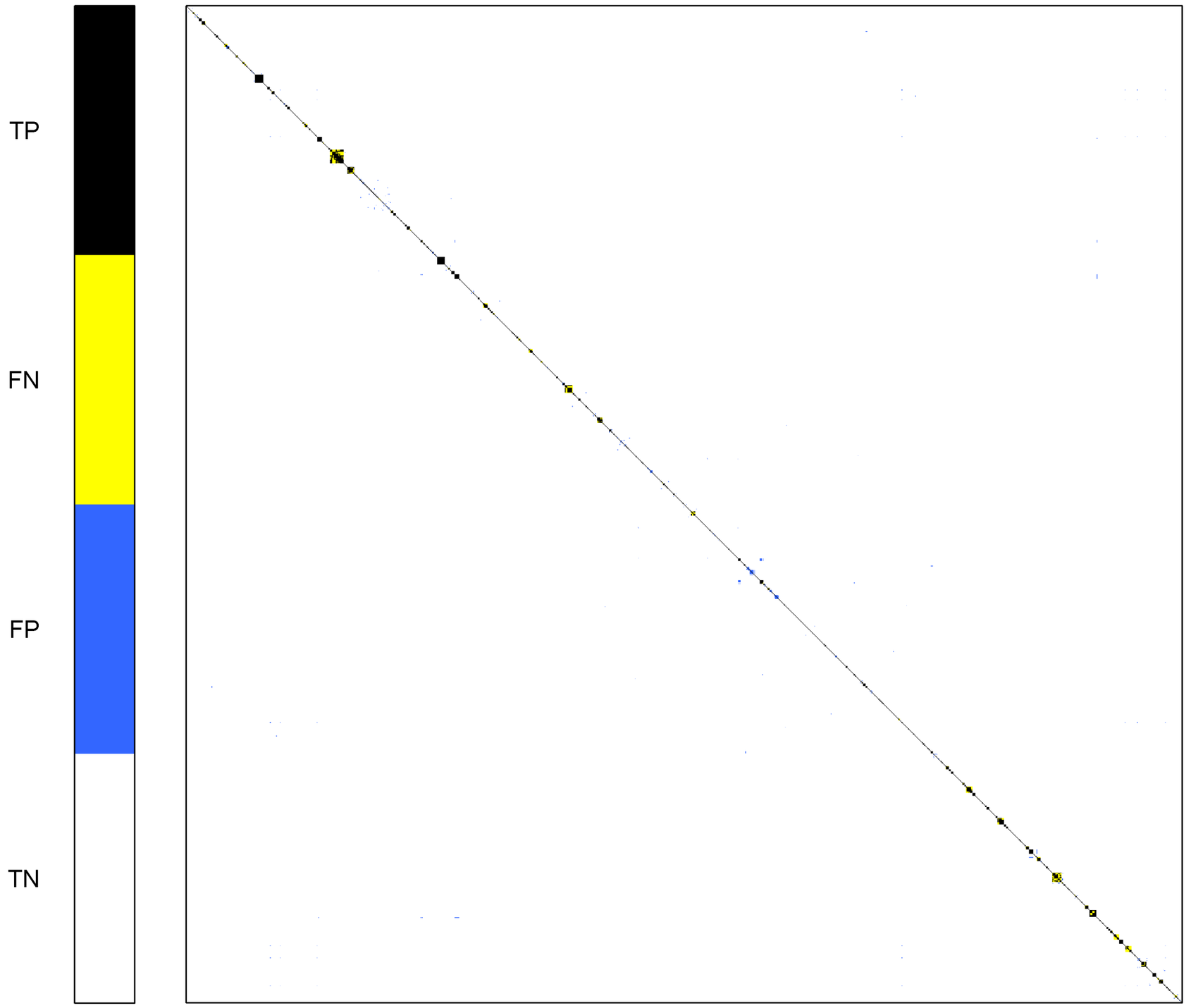

Figure E.9: *Pixel rendering for the C-CodeR algorithm, with threshold applied*

Colophon

This thesis was typeset by the author with the LaTeX 2ε Documentation System on an IBM notebook computer running Microsoft Windows XP. Text editing was done in LyX using the MiKTeX package. The illustrations and graphs were created with Microsoft Visio 2003 and MATLAB 2007a. The front cover illustration is an illustration by Lieke Langezaal. The fonts used for the body, chapter and section titles are: Times Roman and Computer Modern Sans. The monospace typeface used for the listings and program code is Adobe Courier.