Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
9220 Aalborg Ø
Denmark

Automated Assignment of Employees to Work Tasks

Master's Thesis
Aalborg University
June 2009

Lars K. Schunk | schunk@cs.aau.dk

Aalborg University
Faculties of Engineering, Science and Medicine

Department of Computer Science

Title:
Automated Assignment of Employees to Work Tasks

Project Period:
DAT6,
Feb. 2 – Jun. 10, 2009

Project Group:
d620b

Student:
Lars K. Schunk

Supervisor:
Gao Cong

Number of Pages: 76

Abstract:

In recent years, information retrieval techniques have been augmented in order to facilitate automated expert finding by searching document collections for both query topics and associated experts. Typical approaches presume that information about expert candidates exists within the document collections. I argue that such a presumption is not warranted in some enterprise settings. Instead, I turn to structured corporate data—transactions of working hours—to form document-candidate associations. Based on this idea, I design and implement a working expert finding system for a software company. I conclude that presuming the document collection to contain candidate information is not necessary in many enterprise settings because operational systems may contain structured data that is more reliable.

Preface

This Master's thesis documents my work on the DAT6 semester at Aalborg University. The DAT6 semester is the second of two semesters in which I have chosen to specialize in the field of database technology.

Within this broad field, I have chosen to focus on the challenges of utilizing historical data in organizations. In my experience, there often is a forward-looking tendency with a continuous focus on producing new things: new functionality, new business processes, and new data. In contrast, I would like to also look backward and increase the efforts of maximizing the use of latent information assets that already exist in organizations. Such efforts can transform *raw data* into *useful information* that can help improve future decision making and resource management, as well as increase the overall understanding of the implicit mechanisms underlying any organization and its surrounding environment.

In the previous semester, I participated in a project whose aim was to utilize historical numerical data using business intelligence technology such as data warehousing and online analytical processing (OLAP). For the present project, I take historical *textual* data into account in order to acquire a more "complete toolbox" for utilizing historical data.

For this purpose, I have been granted access to historical data from the Danish software company thy:data. I would like to thank them for their trust and for taking an interest in my work.

Lars K. Schunk
June 2009

Contents

Chapter 1

Introduction

During the past few decades, advances in storage and networking technology have resulted in the accumulation of huge amounts of information, which never needs to be deleted. The Web is the most obvious example of a data repository with virtually endless storage capabilities.

But the Web is far from the only place where we find an abundance of information. For companies, information is worth millions of dollars, and so—since storage is cheap—data is rarely deleted. Data is collected over the course of several years, but often it just sits there passively and is never used for anything.

In the last decade, business intelligence projects have been organizing and leveraging these huge amounts of historical data to create more tangible and immediately useful information in the form of pattern and trend identification, which is then used to improve business decision making. Typical examples of historical data that enters a business intelligence system include measures such as retail sales, customer satisfaction, order quantities, and inventory levels.

Common to these examples of historical data is the fact that this is *numerical* data, and it is usually structured in database rows and columns for easy access. This makes it ideal for all kinds of numerical analyses, including online analytical processing (OLAP) and data mining.

However, according to [Gue03] and [Kon06], it is often the case that as much as 80% of a company's knowledge base is lurking inside unstructured textual documents, which are not easily accessible, and may not be optimally used for analysis purposes, if used at all.

Text mining is the activity of processing unstructured textual data in order to "mine"—or uncover—nuggets of previously unknown information. It is a relatively new field of study, but it has great commercial value. A few examples of its use are in order:

- **Customer relationship management (CRM).** The objective of a CRM system is to build a 360-degree view of a company's customers in order to identify and anticipate current and future needs. With effective use of a CRM system a company can build and maintain long-term relationships with customers. Usually, this involves the integration of different facets about the customer, such as age, education, occupation, income, spending habits, marital status, and children. With text mining techniques we can add unstructured communications data (for example, in the form of emails or telephone transcripts) to get a true 360-degree view of the customer, and thus have information about the customer's recent state of mind. Has the customer emailed a complaint? Has the customer been especially pleased with his or her latest purchase? Which products do most customers complain about? What are the most frequently asked questions (FAQs) about a specific product? [IN08, Kon06]

- **Marketing research.** Successful marketing strategies are often dependent on the results of marketing research efforts, which are comprised of, among other things, questionnaire surveys, focus groups, and depth interviews. The data gathered from such activities form the basis of analysis. When processing large amounts of survey responses, marketing researchers typically prefer responses in the form of answers to multiple-choice questions because they can be easily tabulated and summarized for statistical analysis. However, using text mining techniques, we can analyze open-ended questions and process thousands of responses in natural language. As an example, sentiment analysis may categorize responses into positive, negative, or neutral categories, and in that way give a picture of public opinion. Text mining also provides better opportunities for analyzing text association questions in surveys. Besides analyzing survey responses, we can also extract, for example, vast amounts of product reviews directly from various Web sites [Kon06].

- **Knowledge management (KM).** Employees and their knowledge are the most valuable assets in many modern companies. It is not possible for any individual to know everything, and therefore, the ability to locate the appropriate experts for any given information need is essential. In small (and possibly some large) companies, locating an expert is a simple matter of asking around. However, in large companies with several specialized departments, which may even be geographically scattered, this approach becomes infeasible. An obvious solution is to maintain a database of employees and skills where each employee fills in his or her experience, skills, and fields of specialization. However, such manually maintained databases are subject to imprecision, partly due to employees' over- or underrating of their skills, and partly due to the fact that this kind of database often becomes outdated because employees are not motivated to update their entries. Using text mining, the generation of such "knowledge maps" can be automated by processing the information that is published within the organization, such as task descriptions, reports, emails, and memos [Kon06].

1.1 Routing Work Tasks to the Right Employees

Continuing along the path of knowledge management and the ability to locate experts, let us consider a common scenario. In many companies, some people write documents describing tasks that require the work of other people. It is the responsibility of managers to look at these task description documents, determine what skills are needed based on the contents of the documents, and then assign the appropriate people to the tasks.

This is not only a time-consuming process; it also requires extensive knowledge of *what* skills are available and *how* those skills are distributed in the organization. In large organizations, this process is prone to errors given the inherent lack of knowledge capacity of a human manager.

In recent years, *expert finding systems* have emerged to assist with this process. Such a system is provided with an enterprise corpus and a list of people—expert *candidates*, often employees. Given a query topic or document, the system will then estimate what candidates are most likely to be experts on the query topic.

An important aspect of this scenario is the connection between documents and candidates. Most expert finding systems presume that candidate information be available within the documents. For example, employees may be document authors who have written their names and/or email addresses in the documents, or they may in other ways be mentioned.

In this thesis, I investigate the problem of automating the process of locating experts in an enterprise setting. However, I will disregard the presumption mentioned above. Instead, I look for other ways to establish the connection between documents and candidates in the unstructured data that is maintained in operational management systems. Many companies maintain information about their employees, including how many hours they have spent working on different activities. If hours worked on activities can be allocated to documents, then this would seem a good starting point for establishing the necessary document-candidate associations.

To summarize so far: I design and implement an expert finding system assuming the following types of historical data are available:

- Documents describing tasks that require some kind of expertise.
- Employees who have been assigned to the documents.
- Measures that indicate how strongly the employees are associated with the documents, that is, indicators of their level of expertise.

As more data is accumulated over time, one would expect the accuracy of such a system to increase. This opposes the human variant, which is not likely to improve—may in fact become less accurate—as the wealth of data overwhelms the human managers.

1.2 thy:data

In this section, I introduce my industrial collaborator, thy:data, who has provided me with access to data that meet the requirements dicussed above. thy:data is a Danish software development and consulting company within the area of business solutions primarily based on Microsoft's enterprise resource planning (ERP) system Dynamics AX [ax]. They provide services for several mid-sized companies throughout Denmark.

Until 2008, the company had headquarters in Thisted and subsidiary companies in Aalborg, Copenhagen, and Kolding. In May 2008, the group initiated a major restructuring of the entire organization, which involved transforming the subsidiaries into departments (and the establishment of a new department in Århus), in order to streamline business processes and ensure the customers a sense of uniformity regardless of which local department they interact with.

Other major reasons for the restructuring were to better exploit the individual departments' specialized resources across the whole organization, as well as to optimize and centralize development of a common corporate image and employer branding strategies.

These objectives call for, among other things, effective knowledge management, an area in which an expert finding system fits in as a useful tool [thy].

1.3 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2, I look into thy:data's operational task management system in order to identify the required data, and I outline a basic model of the relationships between the data entities. This chapter sums up the thesis thus far with a concise problem statement that sets the focus of the project. In Chapter 3, I present and discuss the theoretical foundation upon which my project draws. Then, in Chapter 4, I briefly survey other people's approaches to similar problems, and I discuss how they can influence the solution to my problem statement. In Chapter 5, I formalize my approach before describing the implementation in Chapter 6. I conduct an evaluation of the system, discuss some evaluation issues, and reflect on how to refine the system in Chapter 7. Finally, in Chapter 8, I conclude on my work and point out some directions in which to take the project in the future.

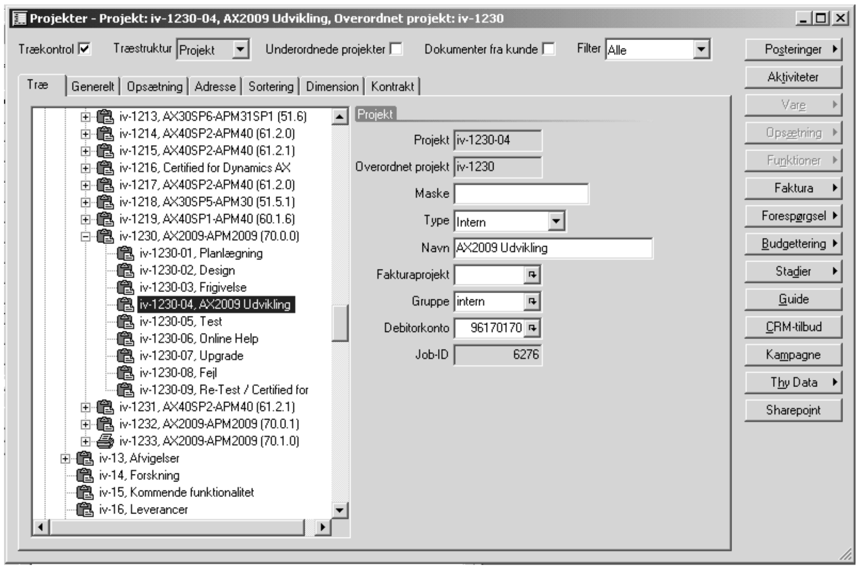
Chapter 2

Problem Analysis

In Section 1.1, I identified the types of data that were necessary in order to consider the problem of finding experts in an organization. In this chapter, I take a look at thy:data's operational management system and locate the necessary data. Furthermore, I explain relationships between the data, and present a basic model that can be used when working with the problem. Finally, I provide a succinct problem statement to summarize the introduction and analysis chapters.

2.1 The Task Management System

At thy:data, they store a large repository of *software development task descriptions* in an operational management system. These task description documents contain specifications for desired software functionality. Typically, a document describes one well-defined function of a larger system and represents a single unit of work that usually can be completed by one or two employees. The task descriptions are written by consultants who agree on the specifications with the customers. Afterwards, a software developer must be assigned to the task. The employees who work on a task register their work hours in the database system. This data is used for invoicing and payroll purposes. Thus, the operational management system contains both structured data (hours worked) and unstructured data (task description documents).



Projekter - Projekt: iv-1230-04, AX2009 Udvikling, Overordnet projekt: iv-1230

Trækontrol ☑ Træstruktur Projekt Underordnede projekter ☐ Dokumenter fra kunde ☐ Filter Alle

```
iv-1213, AX30SP6-APM31SP1 (51.6)
iv-1214, AX40SP2-APM40 (61.2.0)
iv-1215, AX40SP2-APM40 (61.2.1)
iv-1216, Certified for Dynamics AX
iv-1217, AX40SP2-APM40 (61.2.0)
iv-1218, AX30SP5-APM30 (51.5.1)
iv-1219, AX40SP1-APM40 (60.1.6)
iv-1230, AX2009-APM2009 (70.0.0)
    iv-1230-01, Planlægning
    iv-1230-02, Design
    iv-1230-03, Frigivelse
    iv-1230-04, AX2009 Udvikling
    iv-1230-05, Test
    iv-1230-06, Online Help
    iv-1230-07, Upgrade
    iv-1230-08, Fejl
    iv-1230-09, Re-Test / Certified for
iv-1231, AX40SP2-APM40 (61.2.1)
iv-1232, AX2009-APM2009 (70.0.1)
iv-1233, AX2009-APM2009 (70.1.0)
iv-13, Afvigelser
iv-14, Forskning
iv-15, Kommende funktionalitet
iv-16, Leverancer
```

Projekt

Projekt	iv-1230-04
Overordnet projekt	iv-1230
Maske	
Type	Intern
Navn	AX2009 Udvikling
Fakturaprojekt	
Gruppe	intern
Debitorkonto	96170170
Job-ID	6276

Figure 2.1: The main project form.

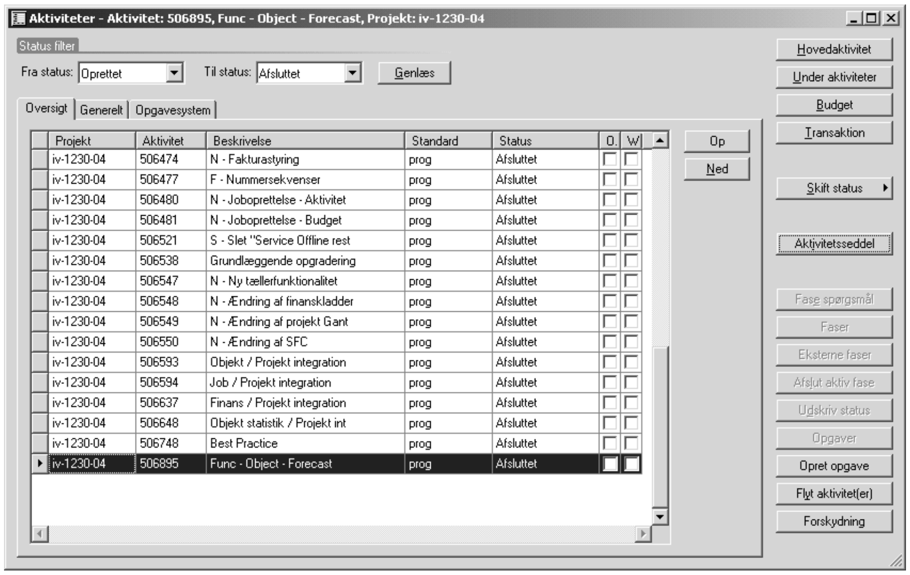
2.1.1 Project Hierarchies and Tasks

All of thy:data's activities are organized into a *project hierarchy*. There is a number of top-level projects, and each project contains a number of sub-projects, which in turn can contain sub-projects themselves, and so on indefinitely. The main project form is shown in Figure 2.1 where the project "iv-1230-04" is selected.

Each project can have associated with it a number of *activities*. An activity usually represents a certain well-defined task such as a software development task. The main activity form is shown in Figure 2.2 where the activity "506895" is selected.

The activities have various textual data associated with them. This includes descriptions of the task that the activity represents, as well as notes written by the people who have worked on the activity. The document form associated with activity "506895" is shown in Figure 2.3. In this case, three documents are associated with the activity. A file and two notes, as indicated by their `Type` fields. The note records have textual fields stored directly in the system's database. The file record holds a reference to a Microsoft Word file on the company's file system.

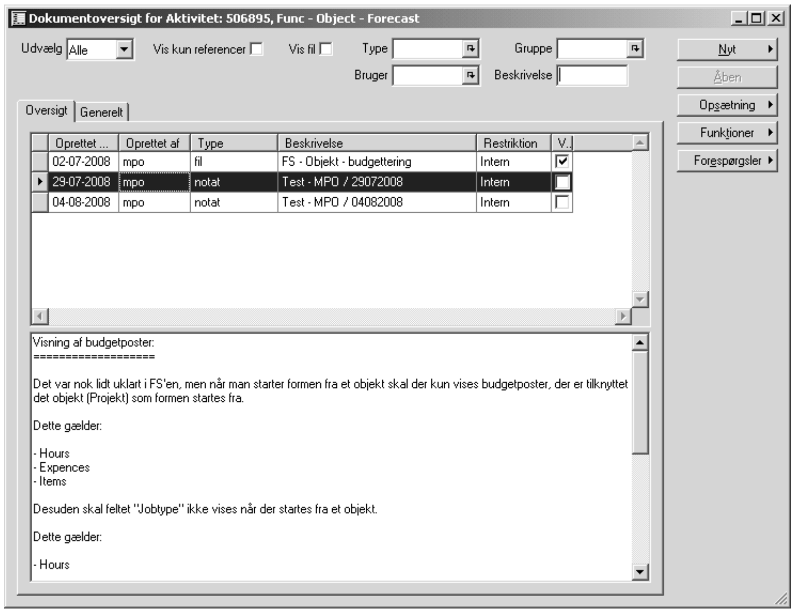
Furthermore, activities have transactions associated with them. Such a



Fra status: Opprettet
Til status: Afsluttet

Projekt	Aktivitet	Beskrivelse	Standard	Status	O.	W
iv-1230-04	506474	N - Fakturastyring	prog	Afsluttet	☐	☐
iv-1230-04	506477	F - Nummersekvenser	prog	Afsluttet	☐	☐
iv-1230-04	506480	N - Joboprettelse - Aktivitet	prog	Afsluttet	☐	☐
iv-1230-04	506481	N - Joboprettelse - Budget	prog	Afsluttet	☐	☐
iv-1230-04	506521	S - Slet "Service Offline rest	prog	Afsluttet	☐	☐
iv-1230-04	506538	Grundlæggende opgradering	prog	Afsluttet	☐	☐
iv-1230-04	506547	N - Ny tællerfunktionalitet	prog	Afsluttet	☐	☐
iv-1230-04	506548	N - Ændring af finanskladder	prog	Afsluttet	☐	☐
iv-1230-04	506549	N - Ændring af projekt Gant	prog	Afsluttet	☐	☐
iv-1230-04	506550	N - Ændring af SFC	prog	Afsluttet	☐	☐
iv-1230-04	506593	Objekt / Projekt integration	prog	Afsluttet	☐	☐
iv-1230-04	506594	Job / Projekt integration	prog	Afsluttet	☐	☐
iv-1230-04	506637	Finans / Projekt integration	prog	Afsluttet	☐	☐
iv-1230-04	506648	Objekt statistik / Projekt int	prog	Afsluttet	☐	☐
iv-1230-04	506748	Best Practice	prog	Afsluttet	☐	☐
iv-1230-04	506895	Func - Object - Forecast	prog	Afsluttet	☐	☐

Figure 2.2: The main activity form.

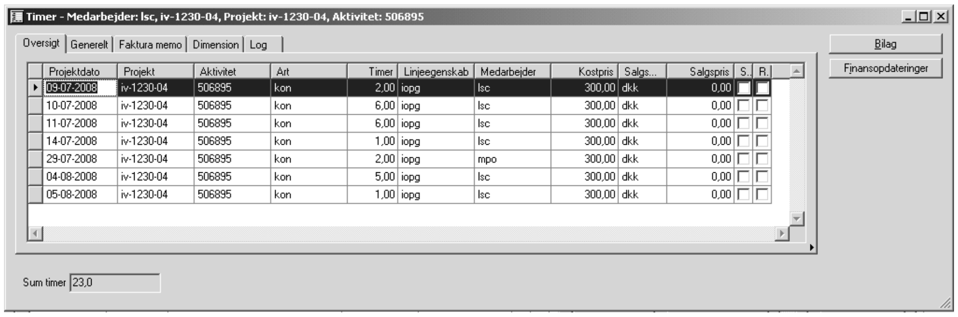

Dokumentoversigt for Aktivitet: 506895, Func - Object - Forecast
Oversigt
Generelt
Nyt
Åben
Opsætning
Funktioner
Forespørgsler

Udvælg: Alle
Vis kun referencer ☐
Vis fil ☐
Type
Gruppe
Bruger
Beskrivelse

Oprettet ...	Oprettet af	Type	Beskrivelse	Restriktion	V.
02-07-2008	mpo	fil	FS - Objekt - budgettering	Intern	☑
29-07-2008	mpo	notat	Test - MPO / 29072008	Intern	☐
04-08-2008	mpo	notat	Test - MPO / 04082008	Intern	☐

```
Visning af budgetposter:
====================

Det var nok lidt uklart i FS'en, men når man starter formen fra et objekt skal der kun vises budgetposter, der er tilknyttet
det objekt (Projekt) som formen startes fra.

Dette gælder:

- Hours
- Expences
- Items

Desuden skal feltet "Jobtype" ikke vises når der startes fra et objekt.

Dette gælder:

- Hours
```

Figure 2.3: The main document form.


Timer - Medarbejder: lsc, iv-1230-04, Projekt: iv-1230-04, Aktivitet: 506895

Projektdato	Projekt	Aktivitet	Art	Timer	Linjeegenskab	Medarbejder	Kostpris	Salgs...	Salgspris	S.	R.
09-07-2008	iv-1230-04	506895	kon	2,00	iopg	lsc	300,00	dkk	0,00	☐	☐
10-07-2008	iv-1230-04	506895	kon	6,00	iopg	lsc	300,00	dkk	0,00	☐	☐
11-07-2008	iv-1230-04	506895	kon	6,00	iopg	lsc	300,00	dkk	0,00	☐	☐
14-07-2008	iv-1230-04	506895	kon	1,00	iopg	lsc	300,00	dkk	0,00	☐	☐
29-07-2008	iv-1230-04	506895	kon	2,00	iopg	mpo	300,00	dkk	0,00	☐	☐
04-08-2008	iv-1230-04	506895	kon	5,00	iopg	lsc	300,00	dkk	0,00	☐	☐
05-08-2008	iv-1230-04	506895	kon	1,00	iopg	lsc	300,00	dkk	0,00	☐	☐

Sum timer 23,0

Figure 2.4: The main transaction form.

transaction represents a number of working hours that a certain employee has spent on the activity. Figure 2.4 shows the main transaction form associated with activity "506895." We can see that a total of 23 hours have been spent on this activity by the employees whose IDs are "lsc" and "mpo."

2.1.2 A Model of the Data

We can view the activities as constituting a central entity that ties together employees and documents, as illustrated in Figure 2.5. The employees are connected to the activities via transactions, and the activities are directly connected to the documents. We can view the *hours worked* measure on the transactions as an indicator of how strongly a given employee is associated with a given document.

These entities can be modeled as a weighted bipartite graph with two disjoint sets of vertices: a set of documents and a set of employees. Weighted edges between the two sets are derived from the activity and transaction entities. An example of this is shown in Figure 2.6. Here we see that document d_1 is associated with employee e_1 because e_1 has "worked" 60 hours on d_1, that is, e_1 has worked 60 hours on the activity to which d_1 is attached. Similarly, e_2 has worked 5 hours on d_1. Thus, d_1 is associated with both e_1 and e_2, but the association to e_1 is stronger than the association to e_2 because e_1 has worked more (60 hours) on d_1 than e_2 (who has worked 5 hours).

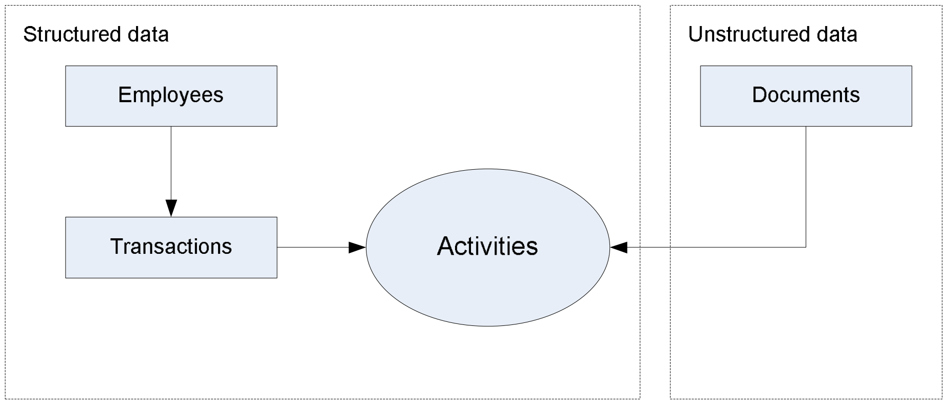


Figure 2.5: Activities as the central entity.

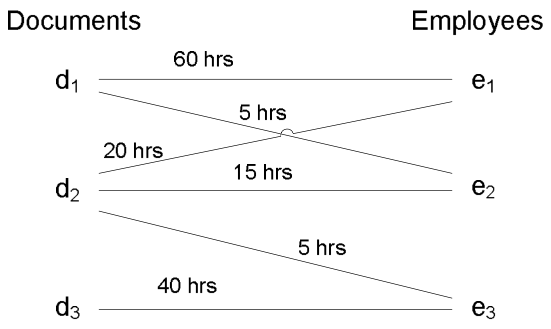


Figure 2.6: Modeling hours worked as a weighted bipartite graph that connects documents and employees.

2.2 Problem Statement

Given the large number of customers and their varying business needs and domain areas, the software developers and consultants often develop a knowledge of topics related to the tasks to which they are most often assigned. These "clusters" of knowledge within the organization are to a large extent implicit and geographically scattered across the company's departments, so task assignment often relies on the experience, organization knowledge, and intuition of the managers.

However, as is evident in this chapter, the data required to automate the process of assigning employees to tasks seems to be available, namely documents, employees, and an association measure, hours worked. It would be useful to design and implement such a system.

Therefore, in this thesis, I:

1. Present the fundamental techniques that apply to the field of processing and querying unstructured textual data.
2. Investigate approaches to expert finding problems and extend them to fit the problem presented in this thesis.
3. Design and implement *Thy Expert Finder*, an expert finding system that requires three types of data: an enterprise corpus, a list of employees, and a collection of hours worked transactions. A user will provide a query topic or task description document as input, and the system will output a list of employees ranked by how likely they are the appropriate person to handle the task.
4. Evaluate and reflect on the results.

With this problem statement I seek to address the organizational knowledge management objectives outlined in Section 1.1 on page 11.

Chapter 3

Theoretical Foundation

The purpose of this chapter is to provide a theoretical background and present the concepts upon which my design builds. I first elaborate on the distinction between structured and unstructured data, and discuss some characteristics of each. Afterwards, I focus on information retrieval and relevance ranking techniques.

3.1 Structured and Unstructured Data

Traditionally, computer-aided analysis has been applied to structured and transaction-based data, which can be neatly stored in rows and columns. The analytical tools developed for these purposes take advantage of the repetitive nature of structured data. The data is predictable in the sense that the tools know what types of data to expect and where they occur—the only thing that differs are the actual data values. An example of this predictability are schema-based database systems where tables and their fields are defined in detail. An analytical tool can easily add up or take averages across several thousands of data rows because of the repetitive structure of a database table.

In contrast to this is unstructured data, which has no repeatable patterns. Unstructured textual data is just sequences of characters typically typed in by humans with the intention to be later read by humans. Traditional analytical tools cannot do much with unstructured data due to its unpredictable nature, and therefore it is often ignored or stored "as is" as some kind of supplement to the structured data. Figure 3.1 shows a (contrived) example of how the

Team Members		
Name	Age	Nationality
James	37	English
Linda	32	Swedish
Pablo	45	Spanish

Okay, so we have three members of our team here. There is Pablo who is Spanish. Then we have 32-year old Linda; she is from Sweden. Finally, we have our Englishman, James, thirty-seven. Oh, by the way, Pablo is 45 years old.

Figure 3.1: Structured data vs. unstructured data.

exact same information can be exhibited in two completely different ways; a structured one that lends itself to automatic processing, and an unstructured one that certainly does not.

A *first generation* of textual analytical tools attempted to deal with unstructured data in isolation. Information retrieval systems—also called *search engines*—are the primary example of the early attempts to make better use of unstructured data by providing capabilities to search for relevant documents in large document collections.

Information retrieval systems are very useful. However, after retrieving documents from an unstructured knowledge base, they do not process the documents any further besides presenting them in a form readable to a human user. The *second generation* of textual analytical tools deal with unstructured information in a more intelligent manner. Three characteristics of second generation textual analytical tools can be identified [IN08]:

1. *Integration of unstructured data.* This involves "streamlining" text prior to analysis. For example, this can include resolution of synonyms[1] and homonyms,[2] as well as dealing with alternate spellings and common misspellings.

2. *Integration with structured data.* Some unstructured data can be naturally connected to the structured data. For example, unstructured customer emails can be connected to structured customer records through their common attribute, *customer name.* The same applies to company employees.

[1]Two different words are *synonymous* when they have the same meaning. The words "buy" and "purchase" are an example of synonyms.

[2]Two different words are *homonyms* if they are spelled the same way. An example is "right," which can mean "correct" or it can be the opposite of "left."

3. *Access to unstructured data in the structured environment.* When the unstructured data has been integrated and relations to structured data have been identified, it can be placed in a structured storage environment such as a relational database. Then it can be processed and analyzed by tools that typically deal with the structured environment.

In this project, I have identified a relation between employee data in the structured environment and the task description documents in the unstructured environment. Once integrated, these two types of data will be placed together in a structured environment in order to leverage the implicit connections that will be made explicit.

3.2 Information Retrieval and Relevance Ranking

Querying unstructured data is known as *information retrieval*, a field that has been the subject of research for many decades, but it has had a particularly great deal of attention since the advent of the World Wide Web. In an information retrieval system, a user wants to retrieve documents of interest from a large body of unstructured documents by submitting a set of keywords known as *terms*. The typical example of data is text documents where the query terms are matched against the actual terms in the documents, but information retrieval also applies to other types of data, such as video and audio, which may have content-describing keywords associated with them. For example, a query with terms "restaurant" and "Copenhagen" may retrieve documents describing restaurants in Copenhagen [SKS06].

3.2.1 The TF-IDF Approach

When querying very large bodies of unstructured documents, a query may retrieve hundreds or thousands of documents that match the query. Therefore, the relevance of the retrieved documents should be estimated and only the most relevant documents should be returned to the user. Thus, we need a measure of how relevant a document d is to a given term t. With such a measure we can rank documents by their relevance to a query term and return the top k documents for some k of our choice. Relevance ranking approaches and their effectiveness are a major research topic in the field of information retrieval.

One basic assumption is that the more frequently a term t occurs in a document d, the more relevant d is to t. We call this measure the *term frequency*, and denote it by $TF(d,t)$. A simple approach for calculating the term frequency would be to count the number of occurrences of t in d, that is, to let $TF(d,t) = n(d,t)$ where $n(d,t)$ is the number of occurrences of term t in document d. But a term is more likely to occur many times in long documents than in short ones, and long documents are not necessarily more relevant than short documents. Thus, a normalized measure would take the document's length into account. We can represent document length as $n(d)$, the total number of all terms in document d. Then we arrive at the following measure of term frequency [SKS06, FS07]:

$$TF(d,t) = \frac{n(d,t)}{n(d)} \tag{3.1}$$

Note that $\sum_{t \in d} TF(d,t) = 1$, so we can view $TF(d,t)$ as the probability of "drawing" term t from the "bag of terms," document d.

Now consider a query Q that consists of more than one keyword, for example, $Q = \{\mathit{mammal}, \mathit{giraffe}\}$. To determine the relevance of a document d to Q, denoted by $r(d,Q)$, we could simply add up the term frequency scores for each term:

$$r(d,Q) = \sum_{t \in Q} TF(d,t) \tag{3.2}$$

However, there is a problem if we use this measure as it stands in Equation 3.2 for estimating the relevance of d to Q because the two terms, "mammal" and "giraffe," are not equally important. Suppose that the document collection D in which we perform our search consists of articles describing all kinds of mammals. Then chances are that documents containing "mammal" but not "giraffe" will be less relevant than documents containing "giraffe" but not "mammal."

We assume that query terms that are rare across the entire document collection D are more relevant than query terms that are common in D. To take this assumption into account, we want to assign weights to each term that will downgrade common terms and upgrade rare terms when calculating relevance. For this purpose, we consider the *document frequency* of a term. The document frequency $DF(t) = |\{d \mid t \in d\}|$ is the total number of documents in which term t occurs. A term with a low document frequency is more specific than a term with a high document frequency, which is more general. The more specific a term is, the more valuable it is when matched in a query [Hie01].

We can now present the weight measure known as the *inverse document frequency* (IDF), which we define as follows:

$$IDF(t) = \log_2 \left(\frac{|D|}{DF(t)} \right) \tag{3.3}$$

Here, $|D|$ is the total number of documents in the document collection.

Applying the IDF measure to the relevance measure in Equation 3.2 we arrive at the following new relevance measure:

$$r(d, Q) = \sum_{t \in Q} TF(d, t) \cdot IDF(t) \tag{3.4}$$

This is known as the TF-IDF measure, and it is a popular approach for estimating the relevance of documents to queries [SKS06, FS07].

3.2.2 The Language Modeling Approach

An alternative to the classic TF-IDF approach for relevance ranking is the language modeling approach, which was first introduced in [PC98] and has garnered much attention in the past decade. Before being used for information retrieval, the concept of a *language model* was primarily used in speech recognition as a probability distribution that captured the statistical regularities and use patterns of a language. A language model would then assign probabilities to the possible next words to occur in a sequence of words, denoted by $P(w_1, w_2, \ldots, w_n)$ where each w_i is a word in the language.

By analogy, a query can be treated as if it were a sequence of words generated by a language model. In this case, the language model would be based on a document from the collection at hand. This document would thus act as a *language sample*. If we infer such a language model M_d for each document d in a document collection, we can rank documents by their probabilities of generating a given query $Q = t_1 t_2 \ldots t_n$ where t_i are query terms. To simplify, we say that the exact order of the terms is not important, so instead of generation of a *sequence* of terms, we opt to deal with the generation of a *set* of terms; then $Q = \{t_1, t_2, \ldots, t_n\}$. We speak of the probability of a query Q given a document d's language model, denoted by $P(Q|M_d)$, or more succintly, just $P(Q|d)$, the probability of a query Q given a document d. The document whose language model has the highest probability of generating the given query is deemed the most relevant document.

A basic building block for estimating these probabilities is the probability of generating a single term given a document, that is, $P(t|d)$. With the given document data at hand, we can obtain the maximum likelihood estimate $P_{mle}(t|d)$ of "drawing" term t from document d, or more precisely, generating term t given document d's language model. The maximum likelihood estimate is simply the term frequency, as defined in Equation 3.1. So $P(t|d)$ can be defined as follows:

$$P(t|d) = P_{mle}(t|d) = TF(d,t) = \frac{n(d,t)}{n(d)} \tag{3.5}$$

Furthermore, we can assume the occurrence of the different query terms to be independent, so we can now present the following ranking formula.

$$P(Q|d) = \prod_{t \in Q} P(t|d) \tag{3.6}$$

Now, what if some query term does not occur in a document? By Equation 3.6, such a document would be given a probability of zero, which is clearly inappropriate. With the documents being language *samples*, it is not unlikely that a query will contain terms not found in a document. However, intuitively, this does not mean that there is a zero probability of the document being relevant to the query. This is known as the problem of *data sparseness*, and the solution is to use a *smoothing method* that will assign non-zero probabilities to the terms that do not occur in the given document. In addition, it may improve the general accuracy of term probability estimation in various ways.

A common starting point for a smoothed model is to provide a "fallback" non-zero probability for non-occurring terms that would otherwise have a zero probability. If a query term does not exist in the document, we can use the probability of "drawing" the term from the entire document collection D instead:

$$P(t|D) = P_{mle}(t|D) = \frac{n(D,t)}{n(D)} \tag{3.7}$$

We can now present a general form of a smoothed model:

$$P(t|d) = \begin{cases} P_s(t|d) & \text{if } TF(d,t) > 0 \\ P_b(t|D) & \text{otherwise} \end{cases} \tag{3.8}$$

where $P_s(t|d)$ is a smoothed probability of t when it occurs in d, and $P_b(t|D)$ is the probability of t as assigned by a background model based on the document collection D when t does not occur in d.

The development and study of various smoothing methods are a major field of research in the information retrieval community [PC98, SC99, ZL01]. One specific smoothing method is the *Jelinek-Mercer method*. The idea is to interpolate the maximum likelihood estimate of $P(t|d)$ with the background collection model, thus taking both into account. It looks like this:

$$P_\lambda(t|d) = (1-\lambda)P_{mle}(t|d) + \lambda P(t|D) \tag{3.9}$$

where $0 \leq \lambda \leq 1$ controls the influence of each model. The Jelinek-Mercer method is used in one of the expert finding models that will be presented in Section 4.2 on page 30.

3.3 Summary

This chapter presented the theoretical background concepts of structured and unstructured data, as well as information retrieval and relevance ranking. It can be confusing as to how the field of information retrieval relates to text mining. A question is whether information retrieval is a text mining activity. In [Hea99], Marti A. Hearst notes that the activity of *data mining* is to discover or derive *new* previously unknown information from collections of existing data. By analogy, *text mining* activities should also provide new information. Hearst then goes on to make the following observation:

> The fact that an information retrieval system can return a document that contains the information a user requested implies that no new discovery is being made: the information had to have already been known to the author of the text; otherwise the author could not have written it down [Hea99].

Thus, according to this argument, text mining is not about making existing knowledge easier to find from a large knowledge base; rather, it is about extracting new, never-before encountered information from such a knowledge base. However, information retrieval can play an important role in a text mining system, and vice versa.

The project documented in this thesis makes use of information retrieval techniques, but its main purpose is not to locate and present information from the task description documents. The purpose is to discover who in the organization is likely the most capable to work on new tasks by analyzing the existing knowledge base. As such, this is new information and can be termed a text mining activity. However, in this project, I rely not only on unstructured textual task descriptions, but on a combination of unstructured textual data and structured numerical data.

Chapter 4

Approaches to Similar Problems

Others have worked on expert finding problems similar to the one discussed in this thesis. Before settling for a design for Thy Expert Finder, it is beneficial to survey some of this work. In this section, I take a look at some of the approaches that could influence my design choices.

4.1 P@NOPTIC Expert

P@NOPTIC Expert, described in [CHVW01], is a Web-based expert finding solution whose starting point has much in common with the problem statement presented in this thesis (Section 2.2 on page 20). The system works on a collection of intranet documents in an organization and a list of current employees. A user who needs to find, say, a Java programmer, enters "java" as a search query, and the system then presents a list of employees who are mentioned in documents that match the query. The top ranked employee is presented with detailed contact information and the relevant documents as evidence for that employee's expertise.

The relation between employees and documents is established by building an *expert index*, which consists of *employee documents*—one document is created for each employee. If there are, say, 150 employees, there will be 150 employee documents. This creation process is very simple. For example, the employee document representing Fred Nerk is simply the concatenated text from all intranet documents in which Fred Nerk's name appears. With the employee documents in place, the P@NOPTIC Expert can match queries against the expert index using any standard information retrieval technique (exactly which method they use is not specified), and retrieve in ranked order the employee documents that match. With the one-to-one correspondence between employee documents and employees, it is a simple matter to go from matching employee document to relevant employee.

4.1.1 Discussion

One could imagine a similar approach for Thy Expert Finder. However, employees are rarely mentioned in the task description documents, so another approach for linking employees with documents must be taken. A possible solution is to look at the number of hours h that employee e worked on the activity to which document d is attached. The text of d would then be appended ch times to the employee document d_e (for some constant factor c). Some drawbacks to this approach are the extra repository of the employee document index that must be maintained, and possibly the size of this index, as many employees may work many hours on many activities. The concatenation of documents will generate much redundancy of the textual data.

4.2 Document Language Models

Another approach to expert finding is presented in [BAdR06]. This is based on the language modeling approach described in Section 3.2.2 on page 25. Given a set of documents D, a set of candidates C, and a set of document-candidate associations $a(d, ca)$, we want to compute the probability $P(Q|ca)$, that is, the probability that candidate ca generates query Q. The candidate with the highest probability of generating Q is deemed the most relevant candidate.

In order to determine $P(Q|ca)$, we consider the following process (as stated in [BAdR06]):

1. Let a candidate ca be given.
2. Select a document d associated with ca with probability $P(d|ca)$. (How to determine such an association will be discussed shortly in Section 4.2.1.)

3. From document d, generate query Q with probability $P(Q|d)$.

By assuming conditional independence between $P(Q|d)$ and $P(d|ca)$, we obtain the joint probability by taking their product. Summing these joint probabilities for all documents $d \in D$ yields $P(Q|ca)$:

$$P(Q|ca) = \sum_{d \in D} P(Q|d)P(d|ca) \tag{4.1}$$

By plugging in Equation 3.6 from page 26 for $P(Q|d)$, we get:

$$P(Q|ca) = \sum_{d \in D} \left(\prod_{t \in Q} P(t|d) \right) P(d|ca) \tag{4.2}$$

Using the Jelinek-Mercer smoothing method, we now plug in Equation 3.9 from page 27 for $P(t|d)$:

$$P(Q|ca) = \sum_{d \in D} \left(\prod_{t \in Q} ((1 - \lambda)P_{mle}(t|d) + \lambda P(t|D)) \right) P(d|ca) \tag{4.3}$$

Computing this for *all* documents $d \in D$ is probably quite expensive. In practice, we want to constrain this to a subset of D by taking only documents that are relevant to the query at hand into account. We may also opt to further constrain this by considering only the top k most relevant documents for some appropriate choice of k.

Note that we still need to define how to compute $P(d|ca)$, the probability of document d given candidate ca. We deal with this in the next section.

4.2.1 Document-Candidate Associations

A central part of the document language model described above are the document-candidate associations, which provide a measure of how strongly a candidate is associated with a document. Given a collection of documents D and a collection of candidates C, to each pair (d, ca), where $d \in D$ and $ca \in C$, a non-negative association score $a(d, ca)$ must be assigned such that $a(d, ca_1) > a(d, ca_2)$ if candidate ca_1 is more strongly associated with document d than candidate ca_2.

With such a document-candidate association score, we can compute the probability that a document is associated with a candidate like this:

$$P(d|ca) = \frac{a(d, ca)}{\sum_{d' \in D} a(d', ca)} \tag{4.4}$$

Given a candidate ca, the document d with highest probability $P(d|ca)$ will be the document with which ca is most strongly associated.

We still need to find some way to calculate the actual document-association scores $a(d, ca)$. Balog et al. solve this in [BAdR06] by using four binary association methods $\mathtt{A}i$ $(i = 0, \ldots, 3)$ that return 0 or 1 depending on whether document d is associated with candidate ca according to each rule. They assume that each candidate is represented by a unique *person_id*, one or more *names*, and one or more *e-mail* addresses. The four rules are as follows (as stated in [BAdR06]).

- `A0: EXACT MATCH` returns 1 if the name appears in the document exactly as written.
- `A1: NAME MATCH` returns 1 if the last name and at least the initial of the first name appears in the document.
- `A2: LAST NAME MATCH` returns 1 if the last name appears in the document.
- `A3: EMAIL MATCH` returns 1 if the e-mail address appears in the document.

Association scores are then assigned as follows.

$$a(d, ca) := A_\pi(d, ca) = \sum_{i=0}^{k} \pi_i \mathtt{A}i(d, ca) \tag{4.5}$$

where $\pi = \{\pi_1, \ldots, \pi_k\}$, π_i is some weight on $\mathtt{A}i$, and $\sum_{i=0}^{k} \pi_i = 1$. Note that if `A0` returns 1, then `A1` also returns 1, and if `A1` returns 1, then `A2` also returns 1. The result of `A3` is independent from the others.

4.2.2 Discussion

I will need a similar document-candidate association measure for the Thy Expert Finder project. However, the rule-based approach with `A`i methods will probably not work well because, as also mentioned in the P@NOPTIC discussion in Section 4.1, employee names are only rarely mentioned directly in the task description documents. However, the hours worked measure that is available seems to be a good foundation on which to design an $a(d, ca)$ measure.

Furthermore, it seems likely that the document model proposed by Balog et al. for locating experts could be used for this project when their $a(d, ca)$ measure is replaced by a new one that considers hours worked instead of relying on named entity extraction.

4.3 Summary

This chapter discussed two approaches to solving expert finding problems. Both approaches presumed the existence of a collection of documents and a list of candidates. They also presumed that the textual contents of the documents would somehow contain candidate information, which would indicate that a candidate to some degree is associated with, or has contributed to, the corresponding document. Such assumptions seem unlikely to work for my project, because looking at a few of the task description documents reveals that employee names are only rarely mentioned. Therefore, another approach must be taken to establish document-candidate associations.

This is where the hours worked measure comes in. Intuitively, it seems likely that such a measure would provide a very accurate means of determining the degree to which a candidate is associated with a document. If a candidate has worked many hours on tasks described by a certain class of documents, it seems likely that the candidate has developed some degree of expertise within the topics of those documents. Apparently, though, association indicators as explicit as hours worked do not seem to have been the subject of much research in the expert finding community. Perhaps a reason for this is that such approaches will be relatively tightly coupled with the enterprise setting at hand, and consequently it may be difficult to generalize the results.

Chapter 5

Method Design

The fundamental theoretical concepts presented in Chapter 3 along with the expert finding approaches discussed in Chapter 4 provide the foundation for Thy Expert Finder. In this chapter, I take these building blocks and modify them in order to take advantage of the structured data that is available, namely hours worked. First, the central part of many expert finding solutions, the document-candidate associations, is modified. Then, in the following two sections, the necessary changes to the TF-IDF relevance ranking approach and the document language model approach are discussed. Finally, I briefly discuss some opportunities for extending the basic models.

5.1 Document-Candidate Associations

Let us formalize the model of the hours worked that was developed in an intuitive manner in Section 2.1.2 on page 18. Let D be the set of documents, and C the set of candidates. Let $G = (V, E)$ be a bipartite graph where $V = D \cup C$ is the set of vertices and $E = \{\{d, ca\} \mid d \in D \text{ and } ca \in C\}$ is the set of edges. To each edge $\{d, ca\} \in E$ we assign weight $w(d, ca) =$ total number of hours worked on d by ca.

Now we can introduce a simple document-candidate association measure $a(d, ca)$ in place of the one that was presented in Section 4.2.1 on page 31:

$$a(d, ca) = \begin{cases} w(d, ca) & \text{if } \{d, ca\} \in E \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

5.2 Modifying the TF-IDF Approach

In this section, I propose a solution to the candidate ranking problem by modifying the TF-IDF approach described in Section 3.2.1 on page 23. We want to establish a measure $r(ca, Q)$ of the relevance of a candidate ca to a query Q, much like the measure $r(d, Q)$ of the relevance of a document d to a query Q.

Suppose we have found the one and only document d that is relevant to the query Q. Then we can define the relevance of candidate ca to Q like this:

$$r(ca, Q) = a(d, ca) \tag{5.2}$$

The candidate who has worked the most hours on document d will be the top ranked candidate in terms of relevance to Q.

However, many documents may be relevant to Q, some more than others. We consider the following:

- If a candidate is strongly associated with a highly relevant document, then that candidate is likely to be highly relevant to the query.
- If a candidate is strongly associated with a moderately relevant document, then that candidate is likely to be relevant.
- If a candidate is moderately associated with a highly relevant document, then that candidate is likely to be relevant.
- If a candidate is moderately associated with a moderately relevant document, then that candidate is moderately likely to be relevant.
- We need only consider candidates who are associated with documents that are relevant to Q. If a document d is not relevant to Q, then no $a(d, ca)$ score for any ca will contribute to that ca's relevance score regarding Q.

With these considerations in mind, I propose the following definition of the relevance $r(ca, Q)$ of a candidate ca to a query Q:

$$r(ca, Q) = \sum_{d \in D_Q} a(d, ca) \cdot r(d, Q) \tag{5.3}$$

where $D_Q = \{d \mid r(d, Q) > 0\}$ is the set of documents that are deemed relevant to Q. This way we use the relevance of documents as weights on the document-candidate associations. In addition, the more relevant documents that a candidate has worked on, the more likely it is that he is a relevant candidate; we take this into account by summing over all relevant documents. Actually, we can sum over all documents, because the relevance scores for irrelevant documents are 0, thus eliminating document-candidate association scores for hours worked on irrelevant documents.

5.3 Modifying the Document Language Model Approach

In Section 4.2.2 on page 33, I argued that the expert finding approach presented in [BAdR06] was likely to work well for my project. The only requirement was that the document-candidate association measure $a(d, ca)$ was replaced by another one that relies on hours worked instead of the rule-based method using named entity extraction. Having provided such a substitute in Section 5.1 on page 35, it is straightforward to plug this into the document language model.

5.4 Possible Extensions of the Basic Models

Now that the basic models are in place, it may be appropriate to take a moment to consider possible future extensions. One could imagine some adjustments to the document-candidate association measure presented in Section 5.1. Consider the following scenario. Suppose that, given a query Q, the set D_Q are deemed relevant documents. Furthermore, candidates ca_1 and ca_2 are deemed relevant candidates. ca_1 has worked hundreds of hours on just one relevant document while ca_2 has worked moderate numbers of hours, say 10–20, on several relevant documents. Which candidate is more relevant? By Equation 5.3, ca_1 is likely to score higher because the hundreds of hours worked on one document will boost his score significantly. But the work on this *single* document may represent an exception. In contrast, if someone has worked moderate numbers of hours on *several* relevant documents, this may reflect the fact that he actually is an expert on this topic who completes his tasks quickly.

To take this into account, I introduce another measure, *document count*, denoted by $dc(ca, D_Q)$, which is the number of relevant documents in D_Q to which candidate ca is associated:

$$dc(ca, D_Q) = |\{d \mid d \in D_Q \text{ and } a(d, ca) > 0)\}| \tag{5.4}$$

We could extend Equation 5.3 with the document count measure:

$$r(ca, Q) = dc(ca, D_Q) \sum_{d \in D_Q} a(d, ca) \cdot r(d, Q) \tag{5.5}$$

Applying this to the example scenario of this section would boost ca_2's relevance score to reflect the fact that he has worked on several relevant tasks even if the total hours worked are less than those of ca_1.

The implementation of this and other possible extensions is left for future work.

5.5 Summary

In this chapter, I modified the TF-IDF relevance ranking approach to rank candidates instead of documents. I also modified the document language model approach by replacing the document-candidate association measure used in [BAdR06] by a new one.

Both of the modified approaches now take advantage of available structured data in order to better utilize the unstructured data. This is in contrast to other candidate ranking approaches where document-candidate associations are established by analyzing only the unstructured data.

For this project, I rejected the rule-based named entity extraction technique for establishing document-candidate associations. However, as possible future work, it could be interesting to combine both named entity extraction and hours worked into a single document-candidate association approach. Even though employee names and email addresses rarely occur in the documents, this does not mean that they never occur. For example, the hours worked measure could be integrated into a rule-based approach as one or more additional `A`*i* rules.

Chapter 6

Implementation

In this chapter, I describe the design and implementation of Thy Expert Finder, the application developed to solve the problem of suggesting employees given a query or another task description document. C# is the primary programming language used for developing the program. First, I give an architectural overview of the system and the environment from which it receives its basic data. This is followed by a description of the preprocessing tasks that transform the raw data into a form useful for analytical processing. The preprocessed data is placed in an SQL Server environment, which is then described. Then I present some statistical summaries of the transformed data to give an idea of what we are dealing with. Finally, I provide some details of the implementation of the relevance ranking procedures, followed by a demonstration of the running application.

6.1 Architectural Overview

Before delving into the design of the individual components, Figure 6.1 provides an architectural overview of all the components that constitute the entire system. The upper-left (framed) part of the figure shows the components of the existing system, that is, the task management system and the unstructured task description documents.

The Dynamics AX component is in fact a complete enterprise resource planning (ERP) system, but for the purpose of this project only the part concerning task management is relevant; hence the label "Task Management System." A typical Dynamics AX installation is itself a highly complicated

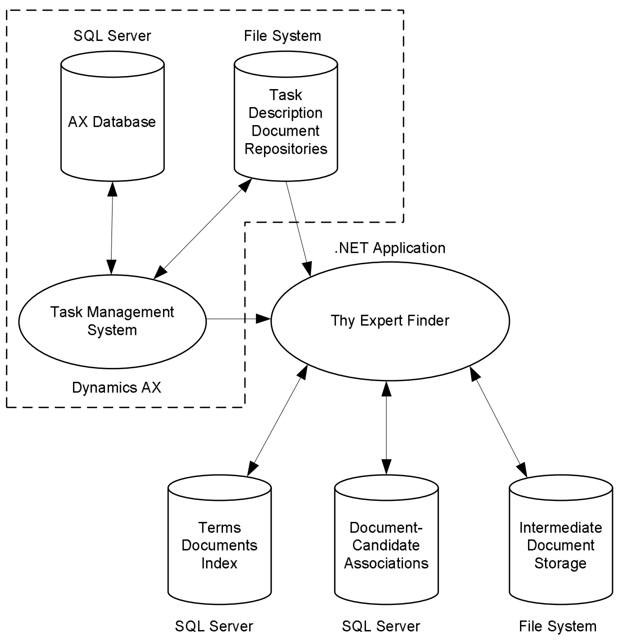


Figure 6.1: Architectural overview.

system with many layers of components. In this illustration, I have opted to just show the fact that the Dynamics AX system provides business logic to corporate data, which is stored in an underlying SQL Server database system. This is where the structured data is stored—hours worked are the structured data of our interest. The unstructured task description documents are stored separately in directories on the company file system. The Dynamics AX system provides links to the individual files.

The Thy Expert Finder component is a .NET application and the core component of the system developed for this project. Notice that it accesses the structured data through the Dynamics AX component, as indicated by an arrow, instead of accessing the SQL Server database directly. Dynamics AX provides development tools, business logic, and relational information to ensure the integrity of the data. The unstructured documents, however, are accessed directly via the file system, as there is no benefit in accessing them through Dynamics AX. The documents are simply copied to the Intermediate Document Storage data store (depicted in the lower-right corner of the figure) so that they can be transformed into forms appropriate for analysis. These transformation processes are discussed in detail in Section 6.2.

The Thy Expert Finder component maintains two SQL Server database tables: one containing a terms-documents index and one containing document-candidate associations. These are used for the relevance ranking of employees, and are described in Section 6.3 on page 48.

Finally, the arrows between system components and data stores in Figure 6.1 indicate data flow. A double arrow indicates that the system component both accesses and manipulates the data store. Notice how the arrows that cross the boundary between the existing system and Thy Expert Finder are unidirectional. This indicates the fact that Thy Expert Finder only accesses the existing data without manipulating it, so the application can be independently added to or removed from the environment.

6.2 Preprocessing Tasks

In order to effectively manage and analyze text data, we need to do some preprocessing on the documents to clean up the data and to represent them in a form appropriate for analysis. This involves converting Microsoft Word files to plain text, tokenizing the text, removing stopwords, stemming, and designing a data structure to represent the documents. In the following subsections I will describe the approaches taken for each preprocessing step.

Throughout, I illustrate the steps with a running example, `835.doc`, a Microsoft Word document retrieved from thy:data's task management system.

6.2.1 Conversion

The first preprocessing task is to convert the original file formats to a common plain text format. Most of the documents are stored directly in plain text fields in the database or as Microsoft Word files. In case of a Word file, it is converted to text by using functionality of the Microsoft Word COM component.

An extract from `835.txt` after being converted from `835.doc` is shown in Figure 6.2.

```
1 Objekt - budgettering

1.1 Objekt - budgettering
Det skal på et objekt være muligt at budgettere. Ved
budgettering sker dette direkte på det projekt som
objektet er tilknyttet.

Sti: Service og Vedligehold -> Objekter

Funktionalitet:
* Der tilføjes en ny knap mellem "Setup" og "Prices"
o Ledetekst  - Standard AX label
* Dansk Budget
* Engelsk Forecast
o Hjælpetekst
```

Figure 6.2: Contents of `835.txt` (extract).

6.2.2 Tokenization

The next preprocessing task is tokenization, which is the process of transforming a document, represented as a long sequence of characters, into a sequence of *tokens*. A token is a single unit of text. Generally, we say that the smallest unit of text is a word, and so tokens are separated by whitespace characters. This is a simplification, though, because one may argue that certain word sequences such as "bill of material" and "South America" are single units of text.

Consider the following sequence of characters:

> *The cat saw a rat with a hat. Then all the cats started chasing the rat's hat.*

Applying a simple tokenization process to this sequence would produce the token sequence shown in Figure 6.3. Here whitespace separates tokens, and punctuation characters are treated as separate tokens. Simple categories can be assigned to the tokens. Categorization could be as simple as distinguishing between "word" and "punctuation," or it could be more sophisticated by assigning the corresponding parts of speech to the tokens, such as "noun" and "verb." This is known as part-of-speech tagging [Kon06, IN08].

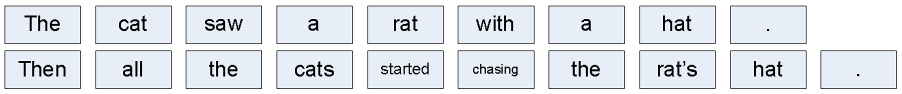


Figure 6.3: Token sequence.

6.2.3 Stopword Removal

Many of the tokens in Figure 6.3 represent extraneous words that are not useful for our analysis purposes. Words such as "the" and "a" are irrelevant and very frequent words that will distort our document representations. For our purpose, we are interested in the words that identify topics and distinguish documents from each other. In other words, we want to keep most *content words* and eliminate most *function words*. Content words are comprised of nouns, verbs, adjectives, and adverbs, whereas function words are comprised of prepositions, pronouns, conjunctions, and determiners. The unwanted words are known as *stopwords*, and we want to eliminate them from our token sequences. Similarly, we do not need punctuation characters, and we do not need to distinguish between upper and lower case [Kon06, IN08].

Continuing with the previous token example, after performing stopword and punctuation character removal, the token sequence now looks like this:

cat saw rat hat all cats started chasing rat's hat

Note how the number of occurrences of the word "the" has been reduced from three in the original sentence to zero. If we had kept the three occurrences of "the," then a typical text mining application might have considered "the" to be the most important word in the sentence, which would clearly be inappropriate.

In Thy Expert Finder, I have used the Lucene Information Retrieval Library [luc] to perform tokenization and stopword removal. For the stopwords, I have used a list of Danish stopwords provided by the Snowball project [sno], and added a few extra stopwords. In addition to the removal of stopwords during preprocessing, I have also opted to remove all tokens that begin with a number, as well as all tokens that consist of less than three characters.

6.2.4 Stemming

Even though we have removed stopwords and punctuation characters, there is still some clean-up to do. Consider the two noun words "cat" and "cats." As they stand, these will be treated as two separate words. However, they refer to the same basic term, namely the term "cat." *Stemming* is the process of replacing words with their *stems*. The word "cats" should be replaced with "cat," thus conflating the two words into their common stem. Similar principles apply to verbs. The words "started" and "chasing" should be stemmed to "start" and "chase," respectively.

After stemming our cat example, it looks as follows:

> *cat saw rat hat all cat start chase rat hat*

Note how the term "saw" has not been stemmed to "see," which would be linguistically appropriate in this case. However, taking all such irregular cases into account would require a fairly sophisticated stemmer. One could also argue that "saw" belongs in the list of stopwords. But then, one should keep in mind that "saw" need not be the past tense of "see," it could just as well be the noun "saw." POS tagging could be used to resolve such issues.

The Porter stemming algorithm described in [Por80] is the classic stemming algorithm used for stemming English text data for information retrieval tasks. Now consider a new (nonsensical) example:

> *I will connect all the connected connections connecting this connection.*

After stopword removal and stemming, this becomes:

> *connect all connect connect connect connect*

Here we see that this does not stem correctly in a strict linguistic sense, as these words belong to different parts of speech. We have a verb ("connect"), an adjective ("connected"), and a noun ("connection"). However, for text mining purposes, it is probably appropriate to conflate all these different words into one term, "connect," which in this way ties this term very tightly to the sentence [Kon06, IN08].

The Porter stemming algorithm was originally designed for the English language. Thy Expert Finder will work mostly with Danish text, which means it should use a Danish stemmer. Danish is grammatically complicated with many inflectional irregularities, which makes stemming difficult. To give an idea of some of the issues, consider the following English words with their corresponding Danish translations italicized:

a cat	the cat	some cats	all the cats
en kat	*katten*	*flere katte*	*alle kattene*
a duck	the duck	some ducks	all the ducks
en and	*anden*	*flere ænder*	*alle ænderne*

As can be seen, stemming Danish words is not just a simple matter of stripping suffixes, a method that works relatively well for English. Still, Danish stemmers are available, and for Thy Expert Finder, I have used the C++-based Oleander Stemming Library [ole], which provides Porter stemmers for several Western European languages.

After tokenization, stopword removal, and stemming, the original document `835.txt` shown in Figure 6.2 on page 42 now looks as shown in Figure 6.4. Note that the word "budget" from the original document has been erroneously stemmed to "budg." This is because the suffix "-et" (along with "-en") is the Danish postfixed definite article. For example, "huset" ("the house") would be correctly stemmed to "hus." Still, our preprocessing of this small example has captured the fact that the terms "budgettering" (forecasting) and "objekt" (object) are key terms in this piece of text data with three and five occurrences, respectively.

6.2.5 Document Representation

In Thy Expert Finder, a document is represented by a C# class, `Document`, which provides the intermediate interface for moving the documents from the unstructured environment on the file system to the structured SQL Server environment. Within the `Document` class, a document is transformed from

```
objek budgettering objek budgettering objek mul budget
ved budgettering sker direk projek objek tilknyt sti
servic vedligehold objek funktionalit tilføj knap
mellem setup pric ledetekst standard label dansk budg
engelsk forecast hjælpetekst
```

Figure 6.4: The document `835.txt` after tokenization, stopword removal, and stemming (extract).

its initial plain-text representation to a vector representation, which is implemented using a `Dictionary<string, int>` object. This is also known as a “bag-of-terms” representation.

In this subsection, I give a description of most of the methods that constitute the public interface of the `Document` class.

```
public Document(string docId, string docIdTxt)
```

This is the constructor. It takes two parameters, `docId`, which is the path to the original document, and `docIdTxt`, which is a path to the equivalent plain-text representation of the document. The system will work with the plain-text representation. However, for presentation purposes, it will use the original document.

```
public TokenStream Tokenize()
```

This method tokenizes the plain-text representation of the document. During this process, it also performs stopword removal and stemming. It returns the tokenized document as a `TokenStream` object. The `TokenStream` class is provided by the Lucene Information Retrieval Library [luc]. It also saves the tokens internally in the `Document` object as an array of bytes (`byte[]`) so that it can reconstruct a new `TokenStream` object at any time.

```
public TokenStream GetTokenStream()
```

This returns a `TokenStream` object from the internal array of bytes created by the `Tokenize()` method described above. It is necessary to recreate the token stream because the `TokenStream` class does not provide a method to reset a token stream.

```
public void SaveTokenStream()
```

This saves the token stream in plain-text format to disk in a separate directory. It is mostly intended for manual inspection and debugging purposes during development of the system.

`public void BuildVector()`

This method builds the vector representation of the token stream. The vector is a `Dictionary<string, int>` object, which consists of a key/value pair for each unique stemmed term in the token stream. The term is the key and the number of times the term occurs in the token stream is the value.

`public void InsertDocument()`

Inserts the document vector values into the underlying database system. Before insertion, this method also calculates the term frequency for each term. The database table schemas are discussed in Section 6.3.

`public int DeleteDocument()`

This deletes the document vector values from the underlying database system that correspond to the current document.

`public void BuildDocument()`

This is just a convenient shortcut method. It calls the following methods: `Tokenize()`, `BuildVector()`, and `InsertDocument()`.

`public string GetTopKTerms(int k)`

This returns a string containing the top k terms that are deemed "most important" for the document. This is used for generating queries from a document.

`public static void UpdateMeasuresForAllDocs()`

This is a static method that updates the corpus-wide term measures such as document frequency and inverse document frequency. This is done by calling a stored procedure to avoid a very large number of database calls. This method must be called after all documents have been indexed.

6.3 The Database

Two databases are maintained by the system. The terms-documents index maintains an index of all terms that occur in the corpus along with information about the documents in which they occur. This is called the *unstructured database*, and it is described in Subsection 6.3.1. The *structured database* maintains information about the employees and the number of hours worked on documents. This is described in Subsection 6.3.2.

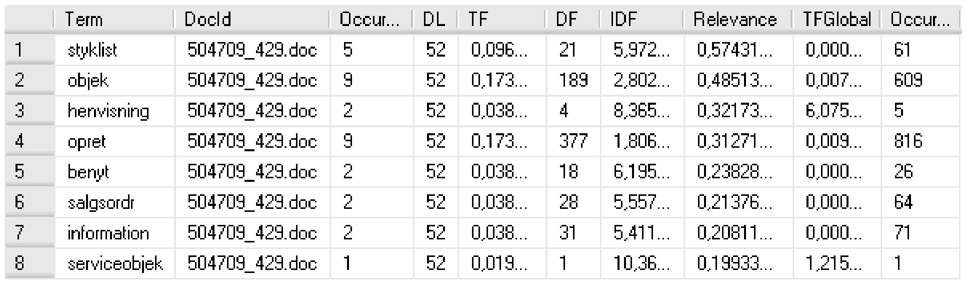
6.3.1 The Unstructured Database: The Terms-Documents Index

When we have preprocessed the documents and transformed them into vectors, we want to place them in a structured environment so that we can use traditional analytical tools, such as SQL, to analyze and query the unstructured data. "Unstructured database" is in a sense a self-contradicting term because a "database" is a collection of structured data, thereby making it "unstructured structured data" [IN08]. However, this is close to what we are going to do; we have structured the unstructured data, and now we will place it into a database, though maybe we should switch the terms and go for "structured unstructured data."

The document data is stored in a database table with the following schema:

> *TermsDocuments_schema =*
> *(<u>Term</u>, <u>DocId</u>, Occurrences, DL, TF, DF, IDF, Relevance,*
> *TFGlobal, OccurrencesGlobal)*

A record represents the fact that a term occurs at least once in a document, so *Term* is simply the (stemmed) text of the term, and *DocId* is a unique identifier of the document in which the term occurs; I simply use the filename of the document as the *DocId*. *Occurrences* is the number of times the term occurs in the document. *DL* is the document length, that is, the total number of terms in the document. *TF*, *DF*, and *IDF* are the term frequency, document frequency, and inverse document frequency, respectively, as described in Section 3.2.1 on page 23. *Relevance* is the measure of how relevant the document is to the term, and it is obtained by multiplying the term frequency by the inverse document frequency. *TFGlobal* and *OccurrencesGlobal*



	Term	DocId	Occur...	DL	TF	DF	IDF	Relevance	TFGlobal	Occur...
1	styklist	504709_429.doc	5	52	0,096...	21	5,972...	0,57431...	0,000...	61
2	objek	504709_429.doc	9	52	0,173...	189	2,802...	0,48513...	0,007...	609
3	henvisning	504709_429.doc	2	52	0,038...	4	8,365...	0,32173...	6,075...	5
4	opret	504709_429.doc	9	52	0,173...	377	1,806...	0,31271...	0,009...	816
5	benyt	504709_429.doc	2	52	0,038...	18	6,195...	0,23828...	0,000...	26
6	salgsordr	504709_429.doc	2	52	0,038...	28	5,557...	0,21376...	0,000...	64
7	information	504709_429.doc	2	52	0,038...	31	5,411...	0,20811...	0,000...	71
8	serviceobjek	504709_429.doc	1	52	0,019...	1	10,36...	0,19933...	1,215...	1

Figure 6.5: Data from the `TermsDocuments` table (extract).

are the corpus counterparts to *TF* and *Occurrences*. For example, *OccurrencesGlobal* indicates how many times a term occurs in the entire corpus, not just in one document.

An example of data in the table is shown in Figure 6.5, where the term "styklist" has the highest relevance score. This makes good sense because "stykliste" (bill of material) is a main topic of the document `504709_429.doc` with five occurrences, and the term appears in relatively few documents, namely 21, as indicated by the document frequency score.

Querying the Terms-Documents Index

With the terms-documents index in place, we can query the index with a set of query terms and retrieve a set of matching documents ranked by their relevance to the query. For this purpose, Thy Expert Finder has the `Query` class. I describe parts of its public interface in the following.

```
public string QueryTerms
```

A property used to get and set the basic terms to query.

```
public TokenStream Tokenize()
```

This method performs the same preprocessing tasks to the query string as are applied to the documents, that is, tokenization, stopword removal, and stemming. This ensures that the query terms and the document terms are comparable. The method returns the token stream, which is primarily intended to facilitate manual inspection during development.

`public TokenStream GetTokenStream()`

As was the case with the `Document` class, the `GetTokenStream()` method returns a new instance of the token stream created by the `Tokenize()` method. This is necessary because a token stream cannot be reset.

`public List<DocumentResult> RankDocuments()`

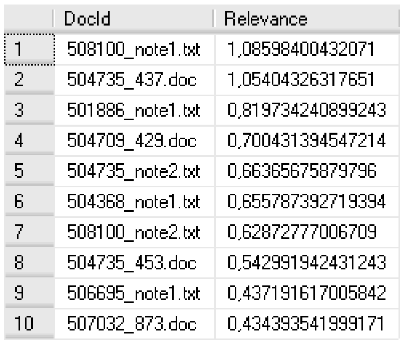
This method connects to the SQL Server and executes the stored procedure `procDocumentsByRelevance` shown in Listing 6.1. This procedure retrieves a set of documents ranked by their relevance to the query. The method returns this as a list of `DocumentResult` objects.

As an example, consider the query "styklister og underobjekter" ("bills of material and sub-objects"). After tokenization, stopword removal, and stemming, the string "styklist underobjek" is passed to the procedure. The results of this query are shown in Figure 6.6.

There will be more to say about the `Query` class later in this chapter.

```
create procedure procDocumentsByRelevance
  @QueryTerms nvarchar(max)
as
  select top 10 DocId, sum(Relevance) as Relevance from TermsDocuments
  where Term in (select * from stringList2Table(@QueryTerms))
  group by DocId
  order by Relevance desc
```

Listing 6.1: Stored procedure `procDocumentsByRelevance`.



	DocId	Relevance
1	508100_note1.txt	1,08598400432071
2	504735_437.doc	1,05404326317651
3	501886_note1.txt	0,819734240899243
4	504709_429.doc	0,700431394547214
5	504735_note2.txt	0,66365675879796
6	504368_note1.txt	0,655787392719394
7	508100_note2.txt	0,62872777006709
8	504735_453.doc	0,542991942431243
9	506695_note1.txt	0,437191617005842
10	507032_873.doc	0,434393541999171

Figure 6.6: Hits to the query "styklister og underobjekter."

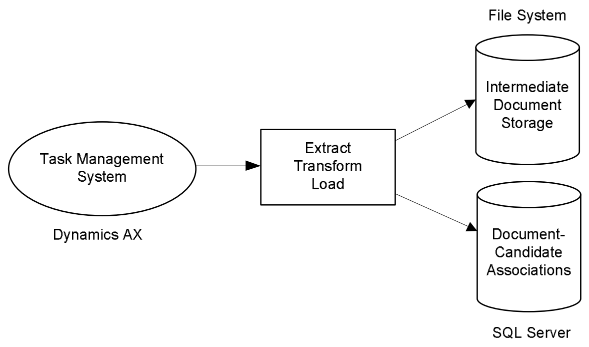


Figure 6.7: The extract-transform-load (ETL) process of document-candidate association data.

6.3.2 The Structured Database: The Document-Candidate Associations

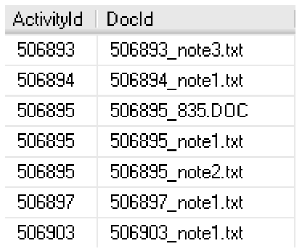
Now we turn our attention towards the structured database that is used for the document-candidate association measure. The data must be extracted from the Dynamics AX source system, transformed into a form appropriate for Thy Expert Finder, and finally loaded into data stores where Thy Expert Finder can easily get to it. This extract-transform-load (ETL) process is illustrated in Figure 6.7.

The extract and transform parts of the ETL process are developed from within the integrated Dynamics AX development environment using scripts written in the Dynamics X++ programming language. The load part of the process is developed using the C# programming language. The technical details of this process have been omitted here, but the result are two SQL Server tables with the following schemas:

$$Documents\_schema = (\underline{ActivityId}, \underline{DocId})$$

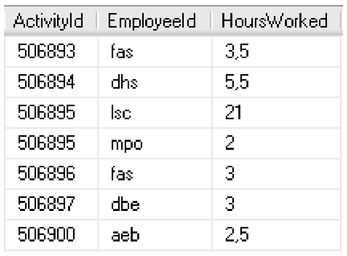
$$HoursWorked\_schema = (\underline{ActivityId}, \underline{EmployeeId}, HoursWorked)$$

An example of data from the `Documents` table is shown in Figure 6.8. Observe how activity 506895 has three records; one for each document with which it is associated: one Microsoft Word document and two notes. Compare this with the task management system's main document form that was shown for activity 506895 in Figure 2.3 on page 17. The Word document, whose original filename was `835.doc`, has been copied from its original data



ActivityId	DocId
506893	506893_note3.txt
506894	506894_note1.txt
506895	506895_835.DOC
506895	506895_note1.txt
506895	506895_note2.txt
506897	506897_note1.txt
506903	506903_note1.txt

Figure 6.8: Data from the `Documents` table (extract).



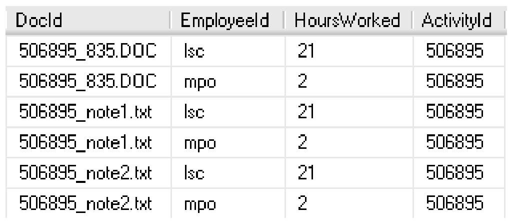
ActivityId	EmployeeId	HoursWorked
506893	fas	3,5
506894	dhs	5,5
506895	lsc	21
506895	mpo	2
506896	fas	3
506897	dbe	3
506900	aeb	2,5

Figure 6.9: Data from the `HoursWorked` table (extract).

store to the intermediate document storage, and its new filename has been prefixed with the activity ID. The textual contents of the two notes were stored directly in memo fields of the Dynamics AX database; they have been written to text files and stored in the intermediate document storage with filenames *activityId*`_note`*i*`.txt`.

An example of data from the `HoursWorked` table is shown in Figure 6.9. Here we see that two employees have been assigned to activity 506895 in the past. Compare this to Figure 2.4 on page 18, which shows the individual hours worked transactions. In the `HoursWorked` table, these transactions have been summed and grouped by activity and employee. Employee "lsc" has worked a total of 21 hours on activity 506895, and employee "mpo" has worked a total of 2 hours on the same activity.

These two tables, `Documents` and `HoursWorked`, provide us with links between activities and documents on the one hand, and links between activities and employees on the other. Thus, we can now see how activities tie the unstructured data with the structured data, as was argued in Section 2.1 and illustrated in Figure 2.5 on page 19. We can obtain document-candidate associations with an SQL view that joins the `Documents` table and the `HoursWorked` table on their common `ActivityId` attribute. This view



DocId	EmployeeId	HoursWorked	ActivityId
506895_835.DOC	lsc	21	506895
506895_835.DOC	mpo	2	506895
506895_note1.txt	lsc	21	506895
506895_note1.txt	mpo	2	506895
506895_note2.txt	lsc	21	506895
506895_note2.txt	mpo	2	506895

Figure 6.10: Querying the `DocumentsCandidates` view. Query: `SELECT * FROM DocumentsCandidates WHERE ActivityId = '506895'`

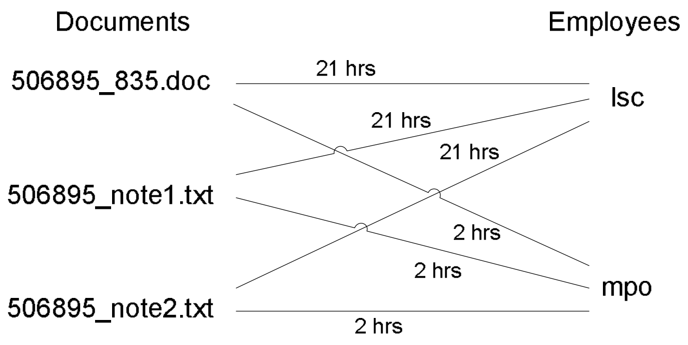


Figure 6.11: Modeling document-candidate associations for the documents attached to activity 506895.

is defined in Listing 6.2.

The result of using this view in a query is shown in Figure 6.10, which makes the relation between documents and employees explicit. We see that both “lsc” and “mpo” are related to each of the three documents that were assigned to activity 506895. In Figure 6.11, these relations are modeled using the approach taken in Figure 2.6 on page 19.

```
create view DocumentsCandidates as
  select Documents.DocId, HoursWorked.EmployeeId,
    HoursWorked.HoursWorked, Documents.ActivityId
  from Documents
  inner join HoursWorked
  on Documents.ActivityId = HoursWorked.ActivityId
```

Listing 6.2: The `DocumentsCandidates` view.

6.4 Data Exploration

Before we continue with the implementation details, it is worth taking a closer look at the data, which has now been indexed by the system. This section provides some statistical figures. Studying these figures is helpful when trying to develop an understanding of the data we are dealing with.

6.4.1 Document Statistics

For this project, I have only used data from the Aalborg department of thy:data. Documents with no associated employees are not indexed by the system. A total of 1319 documents have been indexed; of these are 257 Microsoft Word files, and 1062 are textual notes. Document length statistics are summarized in Table 6.1. Document length is the total number of tokens in a document after stopword removal and stemming.

The first row in Table 6.1 summarizes the lengths of all 1319 documents. However, a few extreme outliers (three documents of lengths 872, 2196, and 6827 tokens, respectively) distort the figures, so they have been removed from the summary in the second row.[1] Only 71 documents have more than 200 tokens, so in order to further reduce the skewness the third row removes these as well. Here we have 1248 documents with a mean length of 41 tokens. This distribution is shown graphically in Figure 6.12. This is still highly skewed to the right, and we can see that a vast majority of the document lengths are in the 1–50 range. The fourth row of Table 6.1 summarizes the 910 documents in this range, which amount to almost 70% of the whole corpus. Figure 6.13 shows this distribution graphically.

6.4.2 Term Statistics

A total of 7585 unique stemmed terms have been indexed by the system. Document frequency statistics are summarized in Table 6.2, whose bottom row summarizes all 7585 terms. The stemmed term with maximum document frequency is “opret” (“create”), which occurs in 377 documents. As indicated by the first row in the table, 4086 (54%) terms occur in only one document, and 1062 (14%) terms occur in two documents, as indicated by the

[1]Data is only removed from these statistical summaries in order to facilitate data representation and interpretation. The data is still kept in the system.

				Std.	Percentiles				See
#Docs	Min	Max	Mean	Dev.	25th	50th	75th	Skewness	Figure
1319	1	6827	62	210	13	29	61	26.23	n/a
1316	1	571	55	74	13	29	61	3.02	n/a
1248	1	200	41	40	12	27	54	1.67	6.12
910	1	50	21	13	10	19	30	0.52	6.13

Table 6.1: Document length statistics.

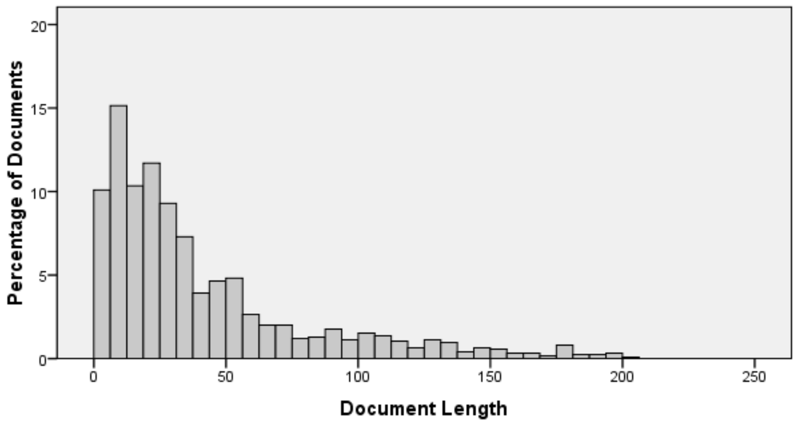


Figure 6.12: A relative frequency histogram of documents consisting of at most 200 tokens.

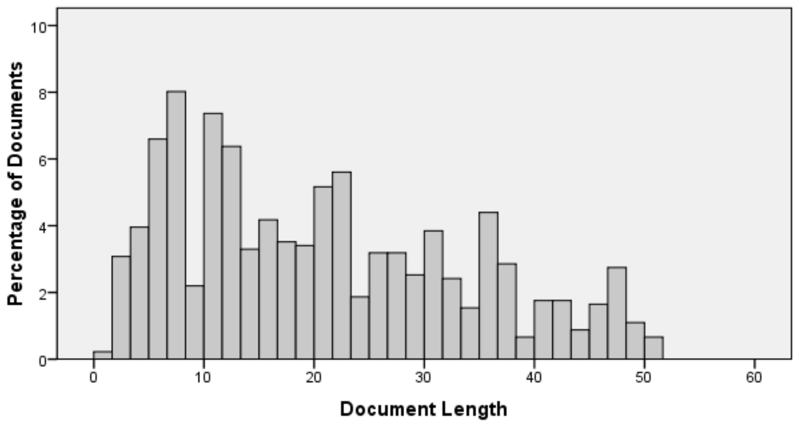


Figure 6.13: A relative frequency histogram of documents consisting of at most 50 tokens.

				Std.	Percentiles				See
#Terms	Min	Max	Mean	Dev.	25th	50th	75th	Skewness	Figure
4086	1	1	-	-	-	-	-	-	-
1062	2	2	-	-	-	-	-	-	-
2015	3	20	6.84	4.43	3	5	9	1.29	6.14
354	21	100	42.35	21.08	26	35	52	1.2	6.15
68	101	377	152.34	60.08	113	123	181	1.89	n/a
7585	1	377	5.98	18.09	1	1	3	8.55	n/a

Table 6.2: Document frequency statistics.

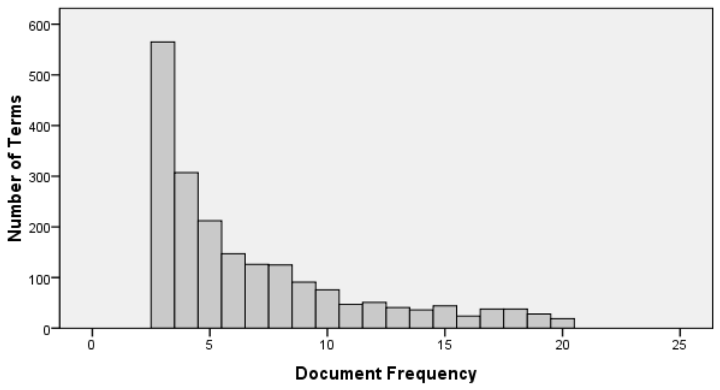


Figure 6.14: The distribution of document frequencies in the 3–20 range.

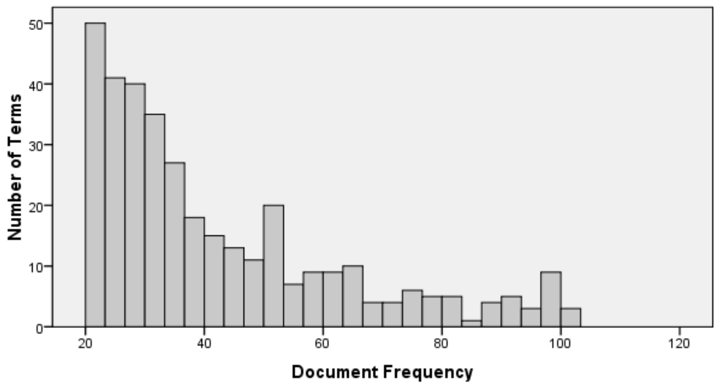


Figure 6.15: The distribution of document frequencies in the 21–100 range.

second row. At the opposite extreme, only 68 terms (0.9%) have document frequencies greater than 100, as indicated in the fifth row of the table.

This leaves us with 2369 terms in the range 3–100, whose distribution is highly skewed to the right, which is not really surprising for this kind of data. Because of the skewness, this has been split up into two groups: one in the 3–20 range, which is summarized in the third row of Table 6.2 and shown graphically in Figure 6.14; and one in the 21–100 range, which is summarized in the fourth row of the table and shown graphically in Figure 6.15.

It could be worthwhile to consider some of the terms with very high document frequencies and possibly include them in the list of stopwords.

6.4.3 Employee Statistics

There is a total of 28 employees who are associated with documents in the Aalborg department. They have worked a total of 22754 hours on activities to which there are associated documents. Hours worked on activities with no associated documents do not provide us with useful information and are not taken into account. No further details of employee statistics will be given here, as this could be considered sensitive information.

6.5 Candidate Ranking

At this point, we are able to locate and rank documents relevant to a given query. However, the documents are not our main concern, so now we take it a step further in order to locate and rank *candidates*, that is, employees who are most likely to be knowledgeable about the topic suggested by the query.

We pick up the `Query` class from where we left it in Section 6.3.1 to present some more methods.

```
public static float DocumentCandidateAssociationScore
(string docId, string employeeId)
```

An important part of candidate ranking is the document-candidate association scores $a(d, ca)$ designed in Section 5.1 on page 35. A simple way to implement $a(d, ca)$ is this:

$$a(d, ca) = \texttt{SELECT HoursWorked FROM DocumentsCandidates}$$
$$\texttt{WHERE DocId = } d \texttt{ AND EmployeeId = } ca$$

```
create procedure procRankCandidates
  @QueryTerms nvarchar(max),
  @Lambda float
as
  --- Get documents relevant to query terms
  declare @RelevantDocs table (DocId nvarchar(max), Relevance float)
  insert into @RelevantDocs
    exec procDocumentsByRelevance @QueryTerms

  --- Turn the list of query terms into a table with TFGlobal scores
  declare @QueryTermsTable QueryTermsTableType
  insert into @QueryTermsTable
    select * from stringList2QueryTermsTable(@QueryTerms)

  --- Rank employees who have worked on the relevant documents
  select EmployeeId, sum(dbo.ProbQGivenDProbDGivenCA(T.DocId,
      @QueryTermsTable, EmployeeId, @Lambda)) as Relevance
  from DocumentsCandidates
  inner join @RelevantDocs as T on DocumentsCandidates.DocId = T.DocId
  group by EmployeeId
  order by Relevance desc
```

Listing 6.3: Stored procedure `procRankCandidates`.

`public List<CandidateResult> RankCandidates()`

This method implements the modified TF-IDF approach for ranking candidates, as designed in Section 5.2 on page 36. The relevance score is computed for each candidate associated with at least one of the relevant documents. The method returns a list of `CandidateResult` objects.

`public List<CandidateResult> RankCandidatesLanguageModel`
`(double lambda)`

As an alternative to the modified TF-IDF approach above, this method implements the modified document language model approach as discussed in Section 5.3 on page 37. This approach is more complicated and most of it is implemented in SQL on the database server for efficiency. The method calls the stored procedure `procRankCandidates` (see Listing 6.3) to do most of the work. This makes use of the function `probQGivenDProbDGivenCA` (Listing 6.4) to calculate $P(Q|d)P(d|ca)$, which in turn makes use of the function `jelinekMercer` (Listing 6.5) to calculate $(1-\lambda)P_{mle}(t|d) + \lambda P(t|D)$. Refer back to Section 4.2 on page 30 to compare with the complete mathematical details.

```
create function probQGivenDProbDGivenCA
  (@DocId nvarchar(max), @QueryTerms QueryTermsTableType readonly,
   @EmployeeId nvarchar(10), @Lambda float)
  returns float
as
begin
  declare @JelMerProduct float
  declare @ProbDGivenCA float

  --- Create a table holding the DocId
  declare @DocTable table (DocId nvarchar(max))
  insert into @DocTable values (@DocId)

  --- Create a table of query terms, their TFGlobal scores, and the DocId
  declare @TermsDocs table (Term nvarchar(max), TFGlobal float, DocId
      nvarchar(max))
  insert into @TermsDocs select * from @QueryTerms, @DocTable

  --- Calculate the product of the Jelinek-Mercer scores, i.e., P(Q|d)
  select @JelMerProduct = exp(sum(log(dbo.jelinekMercer(TF, T.TFGlobal,
      @Lambda))))
  from @TermsDocs as T
  left outer join TermsDocuments
  on TermsDocuments.Term = T.Term and TermsDocuments.DocId = T.DocId

  --- Multiply by P(d|ca) and return result
  select @ProbDGivenCA = dbo.probDGivenCA(@DocId, @EmployeeId)
  return @JelMerProduct * @ProbDGivenCA
end
```

Listing 6.4: SQL function `probQGivenDProbDGivenCA`.

```
create function jelinekMercer
  (@TF float,
   @TFGlobal float,
   @Lambda float)
  returns float
begin
  declare @Result float

  if @TF is null
    select @TF = 0

  select @Result = (1-@Lambda) * @TF + @Lambda * @TFGlobal

  --- Ensure we don't return a zero probability. Otherwise,
  --- we can't simulate Product aggregate with exp(sum(log(i)))
  if @Result = 0
    select @Result = 0.0000000001

  return @Result
end
```

Listing 6.5: SQL function `jelinekMercer`.

6.6 Demonstrating the System

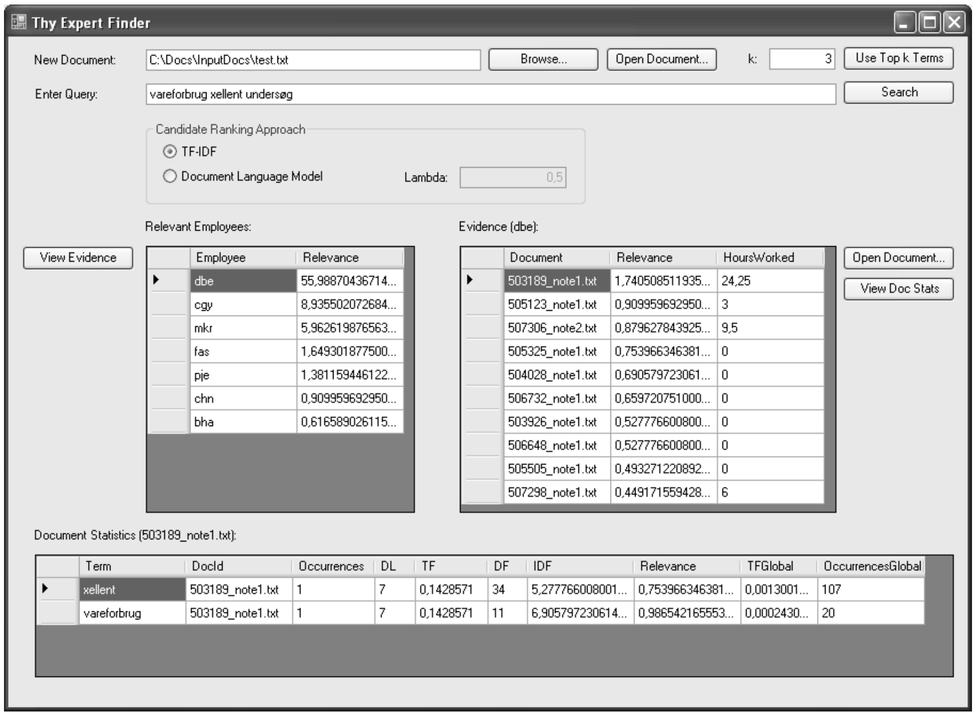
A screenshot of the running system is shown in Figure 6.16. In the upper part of the screen, you enter a query to search for, and you can specify which candidate ranking approach is to be used. If you have a new document at hand, you can have the system extract the top k terms that are deemed relevant to the document. This is done by entering the document path in the `New Document` text box and then clicking the `Use Top k Terms` button. These k terms will then show up in the query text box where you can edit them manually.

Click the `Search` button, and the employees deemed relevant to the query are displayed in the middle left part of the screen, ranked by their relevance with the top ranked employee at the top of the list. You can select an employee and click the `View Evidence` button to see how many hours that employee has worked on the documents that are deemed relevant to the query; this is displayed in the middle right part of the screen. The documents are ranked by their relevance.

You can then get more information about the relevant documents by selecting one and then clicking the appropriate buttons. If you want to verify the relevance of a document, clicking the `Open Document` button lets you view the original document by executing an appropriate document viewer. Clicking the `Doc Stats` button displays at the bottom of the screen the document figures that were used for computing the relevance score.

6.7 Summary

This chapter provided implementation details of Thy Expert Finder. First, it provided an important architectural overview. Second, preprocessing issues were discussed. Preprocessing tasks such as conversion, tokenization, stopword removal, and stemming are important in many text mining applications, and are thus of general interest. The preprocessing components may be refined and reused for the present project as well as future text mining projects. Third, the database stuctures were presented, which are more specific to this particular project. Fourth, the data indexed was examined to get an idea of what we are dealing with. Fifth, the implementation of two candidate ranking methods were presented, which included a few listings of SQL code for some of the central parts. Finally, the user interface of the



New Document: C:\Docs\InputDocs\test.txt

k: 3

Enter Query: vareforbrug xellent undersøg

Candidate Ranking Approach

[✓ TF-IDF]

Document Language Model

Lambda: 0,5

Relevant Employees:

Employee	Relevance
dbe	55,98870436714...
cgy	8,935502072684...
mkr	5,962619876563...
fas	1,649301877500...
pje	1,381159446122...
chn	0,909959692950...
bha	0,616589026115...

Evidence (dbe):

Document	Relevance	HoursWorked
503189_note1.txt	1,740508511935...	24,25
505123_note1.txt	0,909959692950...	3
507306_note2.txt	0,879627843925...	9,5
505325_note1.txt	0,753966346381...	0
504028_note1.txt	0,690579723061...	0
506732_note1.txt	0,659720751000...	0
503926_note1.txt	0,527776600800...	0
506648_note1.txt	0,527776600800...	0
505505_note1.txt	0,493271220892...	0
507298_note1.txt	0,449171559428...	6

Document Statistics (503189_note1.txt):

Term	DocId	Occurrences	DL	TF	DF	IDF	Relevance	TFGlobal	OccurrencesGlobal
xellent	503189_note1.txt	1	7	0,1428571	34	5,277766008001...	0,753966346381...	0,0013001...	107
vareforbrug	503189_note1.txt	1	7	0,1428571	11	6,905797230614...	0,986542165553...	0,0002430...	20

Figure 6.16: The main query form of Thy Expert Finder.

system was demonstrated.

As is often the case with software development, the most important part of implementation documentation is the design of the data structures and the representation of these. If the data structures are well designed and documented, the required procedural logic will be much easier to write and understand. This is also why I have used a considerable amount of space describing the data and data structures in this chapter, and less space describing the processes that work with the data structures. The implementation is, of course, subject to continuous efforts to make it more efficient, elegant, and maintainable.

Chapter 7

Evaluation

The previous chapter concluded with the presentation of a working expert finding system. Thy Expert Finder takes a document or a query as input and suggests employees that it deems relevant to the input. Some questions remain: Are the suggested employees indeed relevant to the query? Is one approach more effective than the other? In this chapter, I present some of the standard techniques used for answering questions such as these in information retrieval and expert finding research. I apply some of these to the present project and discuss the possibilities of conducting further evaluation.

7.1 Measuring Retrieval Effectiveness

The perfect expert finding system would suggest only candidates that are relevant to a query, and the highest ranked candidate would be *the* expert on the given topic. However, as with the perfect information retrieval system, the perfect expert finding system will never exist. This is because relevance judgments of documents and candidates are inherently subjective—they depend on the user posing the query. The exact same query could be posed by different users, yet documents or experts that one user deems relevant may seem irrelevant to another user [Hie01].

Nevertheless, some way of measuring the effectiveness of a retrieval system is desirable. A typical starting point is a test collection of documents D and a test collection of queries Q.[1] For each query $q \in Q$, the documents have

[1] Throughout this section, I speak of *documents* when presenting measures of retrieval effectiveness. The same principles apply when *candidates* are our main interest.

	Relevant	Not Relevant	Total
Retrieved	$A \cap B$	$\overline{A} \cap B$	B
Not Retrieved	$A \cap \overline{B}$	$\overline{A} \cap \overline{B}$	$\overline{B}$
Total	A	$\overline{A}$	D

Table 7.1: Partitioning the document collection for a query.

been manually tagged as relevant or irrelevant. Then we can partition D into two disjoint sets, the set A of documents that are relevant to q, and the complement of A, that is, the set $\overline{A}$ of documents that are not relevant. After having fed q to an information retrieval system and seeing the results, we can likewise partition D into the set B of documents retrieved, and the set $\overline{B}$ of documents not retrieved. These sets and their intersections are illustrated in Table 7.1.

From these sets we can define various measures of retrieval effectiveness. The *precision* of the results to a query is the percentage of the retrieved documents that are actually relevant. It is defined as follows:

$$\text{Precision} = \frac{|A \cap B|}{|B|} \tag{7.1}$$

The *recall* measure is the percentage of how many of all relevant documents are retrieved from the collection. This is defined as follows:

$$\text{Recall} = \frac{|A \cap B|}{|A|} \tag{7.2}$$

Note that any retrieval system can easily obtain a recall score of 100% simply by retrieving *all* documents from the collection, but this results in very low precision. Conversely, by retrieving only the number one top ranked document, there is a good chance for a precision of 100%, but then recall will suffer. In general, aiming for a high precision requires a sacrifice in recall, and vice versa. The challenge, of course, is to develop a system that provides high precision while minimizing the drop in recall (or vice versa) [vR79, SKS06].

Precision and recall take all retrieved documents into account; they do not distinguish between the rankings of the documents. However, it is important that relevant documents are ranked higher than irrelevant ones (false positives). To take this into account, we can calculate precision at different cutoff points. We call these measures *precision at N*, or P@N for short, where N is the cutoff point. This means that we only consider the top N ranked documents returned.

A prerequisite for effective evaluation is to have a good test collection. Furthermore, if different retrieval approaches for like scenarios are to be compared, they should be tested with the same collection. For this purpose, the Text REtrieval Conference (TREC) has provided test collections [tre] that researchers can use to assess different methods for information retrieval, expert finding, and other text mining tasks.

7.2 Evaluating Thy Expert Finder

For the present project, my test collection is the data from thy:data. In order to apply the evaluation principles from above, a set of test queries is required along with their relevant employees. Since I do not have the knowledge to annotate employee relevance by myself, I did a few interviews with some key employees at thy:data, asking them what keywords they think describe some aspects of their expertise. Based on these interviews I then posed five simple queries, fed them to Thy Expert Finder, and observed how well the self-described experts ranked at different cutoff points. The results are shown in Table 7.2 for P@1, P@3, and P@5. For the language model approach, I set $\lambda = 0.5$.

Generally, these results indicate that the system is fairly accurate with most results being above 60–70%. It may seem surprising that the TF-IDF approach performs slightly better than the language model approach because language modeling approaches are generally considered superior to TF-IDF. Of course, this evaluation is not very extensive. To reach more sound conclusions, a more extensive empirical study should be carried out, but such studies consume much time and resources, and would ideally involve the participation of thy:data people with intuitive knowledge of the data.

Approach	P@1	P@3	P@5
TF-IDF	0.8	0.867	0.64
Lang. Model	0.4	0.733	0.6

Table 7.2: P@1, P@3, and P@5 for TF-IDF and document language model approaches.

However, the primary objective of the project was to facilitate expert finding by utilizing structured and unstructured data. This objective has been fulfilled. The approaches taken are based on previous results that have performed well in empirical studies. This constitute the "subjective" aspect of the project. By augmenting these approaches to take hours worked into account, I have added an "objective" aspect. Given the assumption that hours worked are correlated with level of expertise, we can safely incorporate this measure when ranking the employees.

Nevertheless, I shall list some questions for which it might be useful to carry out empirical assessments in future work:

- How is retrieval effectiveness affected by using hours worked for establishing document-candidate associations compared to using named entity extraction?
- Would a combination of hours worked and named entity extraction improve results?
- What is the optimum value of λ for the language modeling approach?
- Would it be beneficial to remove very short documents from the index? What is the optimum cutoff document length?

7.3 Summary

This chapter introduced two of the most important measures for evaluating retrieval effectiveness, namely precision and recall, as well as precision at N for evaluating ranked retrieval. I conducted a simple evaluation of Thy Expert Finder, which indicated fairly accurate rankings. As indicated by the number of possible evaluation questions posed at the end of the previous section, evaluation could make for an entire follow-up project itself. Interestingly, the TF-IDF approach seems to perform slightly better than the language modeling approach. This is surprising because due to prior research results the language modeling approach is generally considered to perform as well as, or better than, the classic TF-IDF approach [PC98, Hie01]. Before jumping to a conclusion about this, a more extensive empirical study may be appropriate. However, the simplicity of the TF-IDF approach is attractive, which could make it the future model of choice for a project like this.

Chapter 8

Conclusion

In this chapter, I summarize and conclude on my results, and I point out some directions in which the project can be taken in future work.

8.1 Results

I have documented the design and development of an expert finding system. I introduced my industrial collaborator, thy:data, as well as their operational task management system, which they use to keep track of, among other things, employees, tasks, hours worked, task assignments, and unstructured documents that describe tasks. I analyzed this environment in order to determine relationships between structured data and unstructured data.

Furthermore, I analyzed previous approaches to expert finding in enterprise environments. These approaches typically consider only unstructured enterprise corpora as well as lists of expert candidates. However, it is often the case that companies maintain structured data that may indicate associations between documents and expert candidates. In this project, I utilized one such type of structured data, namely hours worked. This way, it is no longer necessary to rely on the assumption that candidate information exists *within* the unstructured documents, an assumption that may not be warranted in many situations.

My work has resulted in a fully functional stand-alone .NET application, which given a document or a query suggests employees that have worked on tasks described by similar documents. I conducted a simple evaluation, which indicates that employees are ranked fairly accurately. However, a more extensive evaluation would be desirable. In any case, while a top ranked employee for a query may not always be the one who will eventually be assigned to a new task, the system provides enough information for managers to explore historical data that is relevant to the query at hand. Thus, it is a useful tool to help managers make qualified decisions when faced with new tasks that need to be assigned to appropriate employees.

Currently, the system has only been fed with data from the Aalborg department of thy:data. As part of the restructuring process that was initiated about a year ago in order to unify business processes across geographically scattered departments, it would be interesting to feed data from all departments to the system. This way, managers in one department would become aware of potential experts in other departments, and possibly consider routing new tasks to other geographical locations. This would further specialization and help maximize the use of expertise across the entire organization.

At this point in time, all indexed documents have, obviously, been written without expert finders in mind. However, as people get used to the idea that their documents will be indexed by an expert finder, certain conventions may emerge. For example, it would be useful to include keyword sections in new documents, which would readily make their way into the system. Besides these possible changes in behavior, very little maintenance is required in order to keep the system useful. Currently, the only maintenance job is to periodically update the index, a process that could also be automated. This is a significant improvement over a manually maintained expertise database.

8.2 Future Work

There are several opportunities for improving the components of the system developed in this project. A list of evaluation questions was already presented in Section 7.2 on page 65, and other ideas were mentioned throughout the thesis. In this final section, I summarize ideas for future work, both previously mentioned ideas as well as new ones.

Modeling and evaluation considerations:

- Include named entity extraction into the document-candidate associations.

- Implement the document count measure (Section 5.4 on page 37).

- Conduct a more extensive evaluation. Ideally, this would involve knowledgeable key employees to assist with posing test queries and annotating the employees with relevance judgments.

Implementation considerations:

- Improve the Danish stemmer. Developing a good stemmer is a non-trivial task. Even though the stemmer is just a single small component of an overall system, the potential impact that the quality of the stemmer has on the higher-level components should not be underrated. Therefore, looking into algorithm designs specifically for Danish stemming would be interesting and useful for this project and others dealing with unstructured Danish content.

- Add more format filters such as PDF and Microsoft Excel files.

Organizational considerations:

- Currently, only the data from thy:data's Aalbog department is indexed by the system. As part of the organizational restructuring it would be interesting to include data from all departments. This would help coordinate human resources across the entire organization and route tasks to the appropriate departments.

- Consider the fact that even though an employee may be the top ranked expert to work on a given task, that employee may not have the time required to take on the assignment. This calls for a solution that takes into account not only the expertise level of employees, but also availability constraints.

User-interface considerations:

- Integrate Thy Expert Finder with the Dynamics AX task management system so that the managers can fill out the assigned employee fields of an activity by invoking Thy Expert Finder with a simple click on a button on the activity form. This could be done through Dynamics AX's .NET connectivity capabilities. It could even be taken a step further; it would be interesting to re-design the user interface of Thy Expert Finder entirely within thy:data's existing ERP system using the Dynamics AX development environment. This way, it would be easy to jump to basic information about the employees, departments, activities, and documents that are related to the results of a query.

- Consider the possibility of automatically feeding expertise information back into the operational system, for example, into the Human Resources module. One could imagine a knowledge map that for each employee presents his top fields of expertise.

Other opportunities are related to business intelligence and the integration of structured and unstructured data through the use of data warehousing, thereby linking topics to employees and number of hours worked. Such an effort could provide us with analytical capabilities to answer questions such as the following:

- On which topics do we spend most working hours?
- Given a topic, which department has the highest concentration of relevant expertise?
- Do we have any lost skills? For example, maybe all employees who are linked to a certain topic are no longer in the company.

Bibliography

[ax] *Microsoft Dynamics AX Homepage.* http://www.microsoft.com/dynamics/ax/default.mspx.

[BAdR06] K. Balog, L. Azzopardi, and M. de Rijke. Formal models for expert finding in enterprise corpora. In *Proceedings of ACM SIGIR*, pages 43–50, 2006.

[CHVW01] N. Craswell, D. Hawking, A.M. Vercoustre, and P. Wilkins. P@noptic expert: Searching for experts not just for documents. In *Poster Proceedings of AusWeb*, 2001.

[FS07] R. Feldman and J. Sanger. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data.* Cambridge University Press, 2007.

[Gue03] Lisa Guernsey. Digging for nuggets of wisdom. *The New York Times*, October 16, 2003.

[Hea99] Marti A. Hearst. Untangling text data mining. In *Proceedings of ACL*, 1999.

[Hie01] Djoerd Hiemstra. *Using Language Models for Information Retrieval.* PhD thesis, University of Twente, 2001.

[IN08] W.H. Inmon and A. Nesavich. *Tapping into Unstructured Data: Integrating Unstructured Data and Textual Analytics into Business Intelligence.* Prentice Hall, 2008.

[Kon06] Manu Konchady. *Text Mining Application Programming.* Charles River Media, 2006.

[luc] *Lucene Information Retrieval Library.* http://lucene.apache.org.

[ole] *Oleander Stemming Library.* http://www.oleandersolutions.com /stemming/stemming.html.

[PC98] J.M. Ponte and W.B. Croft. A language modeling approach to information retrieval. In *Proceedings of ACM SIGIR*, pages 275–281, 1998.

[Por80] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.

[SC99] F. Song and W.B. Croft. A general language model for information retrieval. In *Proceedings of ACM CIKM*, pages 316–321, 1999.

[SKS06] A. Silberschatz, H.F. Korth, and S. Sudarshan. *Database System Concepts, 5th Edition.* McGraw-Hill, 2006.

[sno] *Snowball Language for Stemming Algorithms.* http://snowball.tartarus.org.

[thy] *thy:data Company Homepage.* http://www.thydata.dk.

[tre] *TREC Data.* http://trec.nist.gov/data.html.

[vR79] C.J. van Rijsbergen. *Information Retrieval, 2nd Edition.* Butterworths, 1979. http://www.dcs.gla.ac.uk/Keith/Preface.html.

[ZL01] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of ACM SIGIR*, pages 334–342, 2001.

Summary

Because of advances in storage technology, it is not uncommon that companies store very large amounts of historical organizational information. This information consists of both structured data, such as repetitive numerical data, and unstructured data, such as textual documents written in an ad hoc manner.

In recent years, the accumulation of unstructured historical data has given rise to expert finding systems, which are based on the idea that unstructured data represents the knowledge of the people who are associated with it. Given a query topic, such a system suggests candidates ranked by how likely they are to be experts on the topic. Expert finding systems usually require a collection of documents and a list of candidates who are assumed to be associated with the documents. Explicit associations are then formed by analyzing the documents for occurrences of candidate information. For example, if a name of a candidate appears in a document, then there is an association between those two.

In this thesis, I document the design and implementation of such an expert finding system. However, unlike most systems, I disregard the assumption that associations can be derived from within a document collection itself. Instead, I turn to structured historical data for associations between documents and candidates. More specifically, I consider the historical transactions of working hours spent by employees on work tasks to which documents are attached.

I introduce thy:data, a Danish software development company whose work environment provides the three required types of data: documents, candidates, and association measures. Their existing operational task management system is analyzed, and the necessary data is identified. I also provide the theoretical foundation upon which my design is built. This includes information retrieval techniques where I focus on the classic TF-IDF approach and the more recent language modeling approach. Then I present and discuss how others have dealt with the problems of designing expert finding systems. Common to these existing approaches is the assumption that associations can be derived from within the document collections. Inspired by previous expert finding approaches, I design two candidate ranking approaches: one based on a modified TF-IDF ranking approach, and one based on a modified language modeling approach.

An architectural overview of the system is provided. The structured data resides in a Dynamics AX system, and the unstructured data resides partly in this system and partly in Microsoft Word documents. The expert finding system makes use of an extract-tranform-load (ETL) process to move the data to an SQL Server environment where the documents are indexed. Preprocessing tasks such as format conversion, tokenization, stopword removal, and stemming are performed, all of which are described in some detail.

The database tables that store the indexed data are divided into two main categories: the unstructured database, which stores a terms/documents index derived from the unstructured document collection; and a structured database, which stores information about which work tasks are attached to which documents, as well as how many working hours the employees have spent on the work tasks. From these two data sources, I derive document-candidate associations, which are used to rank employees. Then I present some statistical summaries of the indexed data in order to give an idea of its characteristics. After the data structures have been presented in detail, the implementation details of the candidate ranking processes are described briefly, and an instance of the running system is demonstrated.

This project is pragmatic in nature; the purpose is to find an alternative to the occasionally unwarranted assumption that association data exists within the unstructured documents. I perform an empirical assessment of the retrieval effectiveness of my approach. The results show that my basic approach is a promising alternative. Furthermore, I present some ideas of what kind of future refinements might be worth evaluating.

To conclude, I point out that it is not always necessary to rely on the assumption that document-candidate information exists within the unstructured document collection. In many organizations, there are advantages in turning to the structured environment to find document-candidate associations in the form of, for example, hours worked.

Resumé

Væsentlige forbedringer inden for lagringsteknologier har medført at virksomheder ofte opbevarer store mængder historisk information vedrørende organisatoriske aspekter. Denne information består af både strukturerede data, typisk i form af ensformige numeriske data, samt ustrukturerede data, som for eksempel tekstdokumenter skrevet til forskellige formål.

I de senere år har denne ophobning af ustrukturerede historiske data givet anledning til fremkomsten af ekspertlokaliseringssystemer. Sådanne systemer bygger på idéen om at ustrukturerede data repræsenterer den viden som mennesker tilknyttet dataene er i besiddelse af. Givet et emne vil et sådant system foreslå kandidater rangeret efter deres sandsynlighed for at de er eksperter inden for emnet. Ekspertlokaliseringssystemer forudsætter typisk en samling af dokumenter og en liste over kandidater som formodes at være tilknyttet dokumenterne. Herfra udledes eksplicitte forbindelser ved at lokalisere information om kandidaterne i dokumenterne. Der er for eksempel en forbindelse mellem et dokument og en kandidat hvis kandidatens navn optræder i dokumentet.

I denne afhandling dokumenterer jeg design og implementation af et sådant ekspertlokaliseringssystem, men i modsætning til de fleste andre systemer, ser jeg bort fra formodningen om at sammenhænge kan udledes udelukkende fra dokumentsamlingen. Jeg vender i stedet blikket mod ustrukturerede data som udtrykker sammenhænge mellem dokumenter og kandidater. Nærmere bestemt, så inddrager jeg tidligere registrerede timetransaktioner. Disse transaktioner angiver antallet af timer som medarbejdere har brugt på arbejdsopgaver der er tilknyttet dokumenter.

Jeg introducerer thy:data, et dansk softwareudviklingshus, hvis arbejdsgange producerer de tre nødvendige typer data: dokumenter og kandidater samt et mål for sammenhængen mellem dokumenter og kandidater. Jeg analyserer thy:datas eksisterende administrationssystem og identificerer de nødvendige data. Yderligere præsenteres det teoretiske grundlag som mit design bygger på. Dette inkluderer informationssøgningsteknikker hvor jeg fokuserer på den klassiske TF-IDF-metode samt den nyere metode der anvender sprogmodellering. Så præsenteres og diskuteres hvordan andre har grebet lignende problemstillinger an med udvikling af ekspertlokaliseringssystemer. Fælles for disse fremgangsmåder er formodningen om at sammenhænge kan findes udelukkende fra dokumentsamlinger. Med inspiration fra disse, designer jeg to metoder til rangering af ekspertkandidater: én baseret på TF-IDF og én baseret på sprogmodellering.

Beskrivelsen af implementationen indledes med et overblik over systemets arkitektur. De strukturerede data befinder sig i et Dynamics AX-system, mens de ustrukturerede data befinder sig dels i AX-systemet og dels i filsystemet. Ekspertlokaliseringssystemet benytter sig af en udlæs-transformér-indlæs-proces som overfører dataene til et SQL Server-miljø hvor dokumenterne indekseres. Der foretages klargøring af de tekstuelle data ved hjælp af formatkonvertering og opdeling i tegnenheder, stopord fjernes og resterende ord reduceres til deres stamme.

De indekserede data lagres i databasetabeller som inddeles i to kategorier: den ustrukturerede database opbevarer et ord/dokument-indeks; den strukturerede database gemmer information om tilknytning af arbejdsopgaver til dokumenter samt tilknytning af arbejdsopgaver til medarbejdere. Fra de to strukturerede datakilder udledes dokument-kandidat-sammenhænge som bruges til rangering af medarbejdere. Herefter præsenteres nogle statistiske sammendrag af de indekserede data for at give et indblik i dataenes sammensætning. Implementeringsbeskrivelsen afsluttes med en demonstration af det kørende system.

Projektet er overvejende pragmatisk. Formålet er at finde et alternativ til den formodning som ikke altid er berettiget, nemlig formodningen om at sammenhænge kan findes i de ustrukturerede dokumenter. Jeg udfører en empirisk vurdering af systemets effektivitet, og resultatet viser at alternativet er lovende.

Konklusionen er at man ikke nødvendigvis er afhængig af at dokument-kandidat-sammenhænge kan findes i den ustrukturerede dokumentsamling. I mange organisationer vil det ofte være muligt at finde brugbare alternativer i det strukturerede miljø, som for eksempel registreret arbejdstid.