Improving Performance of Biomedical Retrieval using Expectation Maximization

Alexander Borgerud Bjerkan

Master of Science in Informatics
Submission date: September 2011
Supervisor: Herindrasana Ramampiaro, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Information retrieval is a research area that there has been a lot focus on in the last decades. Little of this research has been tailored towards biomedical information retrieval which faces difficulties with it's specialized corpora. This thesis investigates the possibility of increasing the efficiency of biomedical retrieval systems by using statistical clustering to improve retrieval performance for biomedical searches. Our approach uses expectation maximization clustering with naive Bayes to cluster the search results in order to re rank the results.

We have tested our approach by developing a search engine prototype with this feature and evaluated it against a biomedical document collection. We conclude that while there is some increase in precision compared to a baseline search, the gain is too minimal to warrant the extra effort on the user side.

Contents

List of Figures

List of Tables

Acknowledgements

First I would like to thank my girlfriend, Ida, for continuous support throughout this process. Her inspiring words in the too early mornings and too late evenings have truly been a motivational boost for completing this work.

I also owe a great deal of gratitude towards my supervisor, Heri Ramampiaro. His counselling and guidance has been invaluable when hitting the roadblocks.

Finally I would like to thank NTNU, it has been a great five years.

Chapter 1

Introduction

The amount of information available is growing exponentially, as such there is a continuous quest in information retrieval (IR) for handling this ever growing amount of data gracefully [21]. In the biomedical domain especially it is critical for researchers to be able to precisely retrieve the relevant knowledge needed so to stay up to date on the most recent literature and findings in the field [18]. This thesis tries to bring more effiency to biomedical search using existing techniques such as expectation maximization clustering and naive Bayes together new ways. There has been very little research using these for biomedical retrieval.

1.1 The problem

Ramampiaro [18] describes several of the challenges faced when applying common information retrieval models to the biomedical field. In particular, one of the problems faced is that queries can be too broad, or that the retrieval systems are not effective enough, resulting in low precision and results that are not satisfactory enough in terms of answering the user's information need. This is because of the ambiguity and inconsistency within the field, where words can have several meanings and the mixing of natural English and with domain specific terms. For this reason, we are looking for ways to efficiently increase the performance of such search systems.

1.2 Scope

The authors suggests that a way to improve performance for biomedical retrieval is by including explicit user relevance feedback in the search process. Having the user provide binary relevance feedback for the top ranked documents in a search result serves as great input for constructing a traditional naive Bayes classifier known from machine learning. We believe that by re estimating this naive Bayes classifier using the statistical expectation maximization algorithm, the improved classifier's performance is suitable for partitioning the search results into a relevant and irrelevant set. We attempt to increase precision by returning only the documents from the relevant cluster. We have developed a prototype to implement this approach.

1.3 Thesis structure

This thesis is structured in the 7 following chapters:

Chapter 1

This chapter outlines the problem and motivation behind the work done in this thesis. It also describes the intents of our approach.

Chapter 2

This chapter explains the theoretical background and definitions needed to understand the concepts in this thesis. We describe general theory about information retrival and how information retrieval systems are evaluated.

Chapter 3

This chapter shows other relevant and related work to this thesis.

Chapter 4

This chapter explains the ideas and the theories behind our approach in detail.

Chapter 5

This chapter describes the implementation of our prototype.

Chapter 6

This chapter contains the evaluation and presents the testing and results gathered in this thesis.

Chapter 7

This chapter provides the final conclusion to our findings. We also show how further work could affect and enhance the results.

Chapter 2

Background research

This chapter will give a short introduction to information retrieval (IR) and other terms from the field. This lays the foundations for explaining our work understandably in the subsequent chapters.

2.1 Information retrieval

There are several definitions for what constitutes *information retrieval*, but in our context the following fits well: [11]

> Information Retrieval (IR) is finding the material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

Retrieval systems, or search engines facilitates this by having a user with an *information need* translating it into a *query*, a formal statement of the need, which is given as input to the system. The system then tries as best to answer the query with the documents it deems relevant from the collection. To prevent scanning the entire document collection for relevant documents for every query, a very time consuming process, the system employs an *index* [2]. An index is a data structure built over the collection to speed up searching it. The index is then searched instead of the collection. A simple example would be a list of every single word in the collection, and for each word a pointer to the document(s) containing that word. Even though an expensive initial scan over all the documents has to happen to construct the index at first, it is cheap to maintain the index at later stages when the collection is modified. The extra computer storage required for storing the index is outmatched by the benefit of measuring search times in milliseconds instead of hours.

When indexing, it is normal to strip off any punctuation and lowercase the text, in addition to performing a series of operations on it:

Stop word removal

Stop words are extremely common words that appear so frequently that they lose their usefulness as search terms [2]. Some examples are *the*, *is* and *a*. Stop words can be filtered out prior to indexing as they add little or no value for retrieval. They are insignificant as query keywords and they match too many documents.

Stemming

Stemming [2] is the process of reducing a word to it's root form, or *stem*. This is a countermeasure against the many different morphological variants of words. Most of the morphological variants have similar semantic interpretations, and for retrieval systems they can be considered as equivalent. For instance, the words "computer" and "computation" might be stemmed as "comput", as the stems themselves does not have to be valid words. With stemming, the words would be recognized as equivalent by the retrieval system, and the user's information need is more likely to be filled.

Stemming also has the positive side effect of reducing the index size, as the number of distinct terms representing the set of documents is diminished.

The same text preprocessing that is applied when building the index also has to be applied to the queries for the retrieval system to be able to match relevant documents.

2.1.1 Vector space model

The vector space model was first introduced in 1975 by Salton [19]. It is the most widely adopted model for information retrieval and therefore the one we present here. This is also the one adopted for our approach (See Chapters 4 and 5). Other common models include the Boolean model and the probabilistic model.

In the vector space model, each document in a collection of n documents, containing m separate terms, are represented as vectors. Each dimension of a vector corresponds to a weight for a particular term. In the vector $d_1 = (w_{j1}, \ldots, w_{jm})$, w_{ji} designates a weight for the term t_i, in the document d_j. There are several approaches to weighting the terms, but the most widely used is *term frequency - inverse document frequency* (tf-idf) and it is defined as:

$$w_{i,j} = tf \cdot idf = \frac{freq_{i,j}}{max_l freq_{l,j}} log \frac{N}{n_i} \tag{2.1}$$

where $freq_{i,j}$ is the number of occurrences of t_i in document d_j, maximum frequency is computed over all the terms in d_j, N is the total number of documents in the collection and n_i is the number of documents containing the term t_i. The denominator serves for normalizing different document lengths.

This statistical measurement seeks to calculate the importance of a term in a document in a document collection by balancing two factors: Terms that occur often in a document are more likely to be more important (*term frequency*), and terms that occur within many documents are likely to be less important when distinguishing between documents (*inverse document frequency*).

By calculating the distance between two vectors, we measure the similarity between them. The most common measure, cosine similarity, is defined as: [2]

$$sim\left(d_i, d_j\right) = \frac{\vec{d_i} \cdot \vec{d_j}}{\left|\vec{d_i}\right| \left|\vec{d_j}\right|} \tag{2.2}$$

By treating a query as a document vector, queries can be compared against documents, and if there is similarity a document belongs to the result set. The result set can then be ranked according to their degree of similarity. This inhibits the relevance degree of matching documents when searching.

2.2 Clustering

Clustering deals with finding a structure in a collection of *unlabelled* data. The goal is to organize similar or related objects into appropriate groups. While the content of a group, or *cluster*, should be as similar as possible, it should also be as dissimilar as possible from objects belonging to the other clusters [11] (see Figure 2.1). Today the intent of clustering in IR is to better represent the information retrieved and enhance the retrieval process [25], but initially it was used to increase precision and recall [24]. It is easy to see one of the applications of clustering. Imagine a user inputting the query “jaguar” in a retrieval system. The system will retrieve all documents found relevant, but

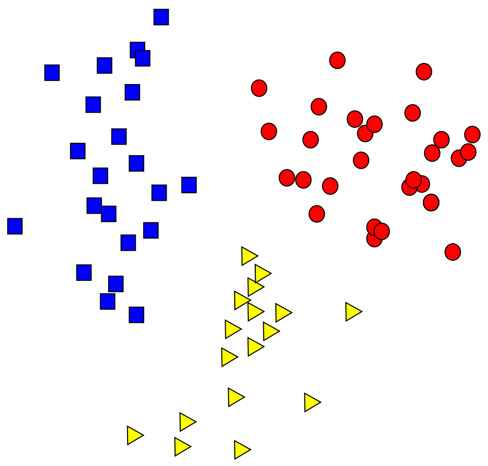


Figure 2.1: An example document set and the inherent cluster structure.

presumably, the results could be a mix of different documents about the big feline animal, the operating system by Apple Inc. and the luxury car brand. With clustering, the results would be ranked in their corresponding cluster. This structural overview would enable the user to quickly identify his wanted group, and the relevant documents within.

Many different clustering algorithms have been introduced in the literature, and possible classifications of the clustering methods can be according to whether they are *hard* or *soft*, *flat* or *hierarchical* and *local* or *global* [2][11].

Hard clustering means the data is partitioned into a specified number of mutually exclusive subsets. With hard clustering, each object is assigned to exactly one cluster.

Soft clustering, or *fuzzy clustering*, means object assignment is distributed over all clusters. An object can have partial membership in several clusters. As such, it is more natural than hard clustering. Objects on the boundaries between several clusters are not forced to fully belong to a single cluster, but rather assigned membership degrees indicating their partial membership.

Flat clustering is simply a set of clusters.

Hierarchical clustering is clusters within clusters, forming a tree structure.

Global clustering means the entire document collection is clustered beforehand.

Local clustering, also known as *online* or *query-time clustering* is the opposite, where the clustering is performed on subset of the collection, for instance the query results.

2.3 Text classification

Given a set of classes, *classification*, or *text categorization* is the process of determining which class a given object belongs to [11]. In general there are three different ways to classify: manually by humans, by computers after a set of rules crafted by humans (such as regular expressions[1] or by machine learning-based functions [11]. Manual classification is undesirable as it is time-consuming and therefore unsuitable for larger document collections and often it even requires trained professionals, such as librarians. Using a rule set is also undesirable as they are difficult to create and maintain.

As such there is focus on machine learning-based text classification. This is a supervised learning problem where a machine based classifier learns from training data [11], with the goal of replicating the categorical distinction that a human supervisor imposes on the data. More specifically, given a set of *labelled* training examples on the form $\{(x_1, y_1), \cdots, (x_n, y_n)\}$, a learning algorithm seeks to find the function

[1]A concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters.

$\gamma : X \rightarrow Y$ [13]. Labelled documents are documents that have already been manually classified, and are annotated to a class [11].

Even though text classification shares several characteristics with the field of clustering (Section 2.2), there is mainly one significant difference. Whereas classification tries to assign documents to existing groups, clustering tries to extract the groupings inherent from the document themselves. The essence of this is that clustering is a form of *unsupervised learning*, while classification is *supervised*. In our approach and prototype we use both unlabelled and labelled data (see Chapters 5-4) which falls between both categories and is known as *semi-supervised learning* [15].

2.4 Evaluation of information retrieval systems

In this section we describe ways in which we can measure and evaluate the performance of information retrieval systems.

2.4.1 Test collections

A test collection is a controlled collection of documents, queries and relevance judgements for the given queries [11]. These test collections are created for the purpose of evaluating and benchmarking retrieval systems.

Some of the test collections popular for IR evaluation are:

Cranfield

The Cranfield collection is considered the first test collection. It was assembled in the late 1950s in the United Kingdom. It consists of

1398 documents about aerodynamics and 225 different queries with relevance statistics.

Text REtrieval Conference (TREC)

TREC is a series of workshops run every year with the purpose of encouraging IR research. The initiative was started by the U.S. National Institute of Standards (NIST) in 1992, and it is now collaboration between NIST and Intelligence Advanced Research Projects Activity (IARPA). Each year there are one more *tracks*, or research areas. Each track is a specific challenge, complete with document collections, information needs and their relevance judgements.

The TREC collections is considered by many as a de facto standard for IR evaulation [14].

GOV2

GOV2, from the TREC Terabyte track, is another NIST initiative. It consists of 25 million different web pages and it is the largest test collection available for research purposes.

2.4.2 Recall

Recall is a measure of the fraction of the relevant documents retrieved for a query (see Figure 2.2). If we define $|Ra|$ as the number of relevant documents retrieved and $|R|$ as the total number of relevant documents for the query, then recall is defined as: [2]

$$Recall = \frac{|Ra|}{|R|} \tag{2.3}$$

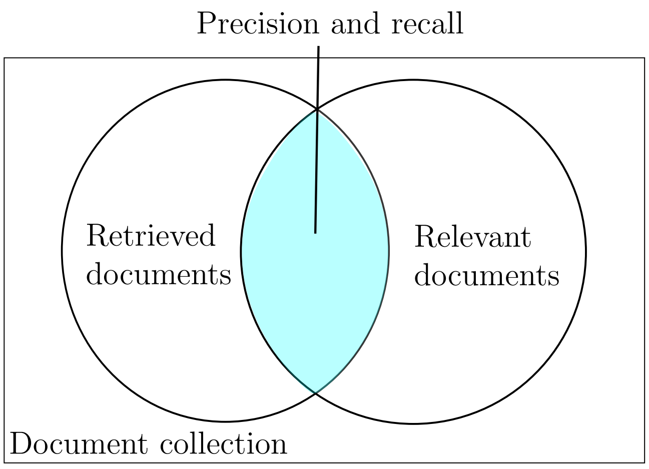


Figure 2.2: Precision and recall

2.4.3 Precision

Precision is a measure of the relevance of an answer set for a given query (see Figure 2.2). More specifically, precision is the fraction of retrieved documents which are considered relevant [2]. If we define $|Ra|$ as the number of relevant documents retrieved, and $|A|$ as the total number of documents in the answer set, then precision is defined as:

$$Precision = \frac{|Ra|}{|A|} \tag{2.4}$$

A high precision value means we are retrieving relevant documents for a query, while a low precision means our result has a high degree of irrelevant documents, or *false positives*.

2.4.4 Precision at n

Precision at n (P@n) measures precision where n is the given cut-off rank. Only the top n documents are considered when calculating precision. This is a useful measure because it reflects the searching habits of a user, who is only looking at the first n results or the first page, with n results [23][20]. However, this measure portrays poorly when the number of relevant documents is smaller than n [11].

2.4.5 R-Precision

R-precision is the precision after R ranked documents have been retrieved, where R also is the total number of relevant documents. This aims to fix the shortcoming of the precision at n measurement, as it accounts for the size of the set of relevant documents [11].

2.4.6 Mean Average Precision (MAP)

Precision and recall are set-based measures, which are inadequate when evaluating ranked retrieval search engines [11]. Moreover, P@n and R-Precision which measures ranked results are specific to a single query. Mean Average Precision (MAP) is a single value score for a set of queries. It is the average of the average of the precision values at the points which each relevant document is retrieved.

$$MAP\left(Q\right) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{m_j}\sum_{k=1}^{m_j} Precision\left(R_{jk}\right) \tag{2.5}$$

where the set of relevant documents $\{d_1 \ldots d_{mj}\}$ for $q_j \in Q$ and R_{jk} are the ranked documents from the top and down to document d_k [11].

Even though MAP is a single score value there are many factors included in it's calculation. Hence it is harder to interpret the value when compared to the other evaluation measures, since the same score can be obtained in several ways.

Chapter 3

Related work

The quest to improve biomedical information retrieval is not new. It has been, and is a, continually ongoing focus of research. This chapter describes some of these strategies and systems.

3.1 Increasing performance

3.1.1 Query expansion

Query expansion is the process of a information retrieval system adding terms to, or reformulating the user's initial query. This is to reduce the mismatch between queries and documents so to increase precision and/or recall [11]. This is to work out the problem of synonymity.

Almost all articles in the MEDLINE database are tagged with multiple MeSH (Medical Subject Headings) terms. MeSH is a controlled vocabulary maintained by the United States National Library of Medicine (NLM) for indexing purposes. Experiments have shown that query expansion on MeSH term improves performance [10]. Others have taken a more traditional approach of using terms from the query and the top ranked documents and also shown improvement for MEDLINE searches [1].

3.1.2 Structural documents with different weights

Another technique for improving performance is to take advantage of documents' inherent structure. For instance, most documents has a title, and the assumption could be made that the title is often chosen to best describe the contents of the document. If a term from a query is found in a document's title, it should contributes more to the likelihood that a document is relevant than if it is only found in it's contents. It may therefore be beneficial to weight the fields differently when searching or indexing, according to their degree of importance, as a way to improve performance. For MEDLINE citations, Ramampiaro shows that weighting the *title* field twice as much as the *abstract* fields yields better results [18].

3.2 Implemented systems

3.2.1 PubMed

PubMed is the official search engine for (mainly) the MEDLINE database[1]. It is developed by the National Center for Biotechnology Information (NCBI) for the United States National Library of Medicine (NML). When searching citations, PubMed automatically expands queries by combining boolean operators and relevant MeSH terms to increase performance. It also features an advanced search page which requires extensive knowledge of how MEDLINE is built and MeSH terms to use. PubMed does not use a document ranking model and results

[1]See `http://www.ncbi.nlm.nih.gov/pubmed/`

seems to be in chronological order [18]. As of 2011, PubMed indexed over 21 million records[2]

3.2.2 ClusterMed

ClusterMed is a meta-search engine developed by Vivísmo, Inc. that searches PubMed and then organizes the results into hierarchical categories[3]. It aims to improve search productivity and efficiency by clustering the results based on authors, affiliation, publication dates and MeSH terms etc.

As it's linguistic processing algorithms are proprietary, the actual categorization scheme is unknown, but it seems to utilize frequently occurring words and frequently occurring phrases in the *title* and *abstract* fields, spanning both each single result and all the results combined [8].

3.2.3 Anne O'Tate

Anne O'Tate[4] is another meta-search engine that relies on PubMed. Anne O'Tate searches PubMed then analyses the results and breaks it down into a number of categories, such as authors, journals and affiliations for easier navigation. It also has the ability break these sets down even further.

One of it's features is a soft clustering that cluster the results by their MeSH terms into no more than 18 clusters. Because of the simplicity of the cluster algorithm the authors claim they are able to cluster tens of thousands articles in real time [22].

[2]Query "all[sb]" in PubMed to see the current size of the database.

[3]See `http://demos.vivisimo.com/clustermed`

[4]See `http://arrowsmith.psych.uic.edu/arrowsmith_uic/index.html`

3.2.4 BioTracer

BioTracer is a proof-of-concept prototype search engine for the biomedical domain [18]. It combines a number of different techniques with the goal of improving performance by maximizing precision and recall. Among the methods used are an extended Okapi BM25 ranking model, boosting of specific document parts, user relevance feedback and support for wildcard queries. BioTracer has shown promising results against the TREC corpus.

Chapter 4

Approach

This chapter goes into depths explaining our approach in trying to increase the efficiency in biomedical searching. We present the three main theories that constitute our work, how we plan to combine them and our reasons for doing so.

4.1 The idea

The problem we are presented with is the issue of performance for biomedical retrieval systems. As described in the introduction (see Chapter 1), there is focus on maximizing the efficiency. Our take on reaching this goal is to combine several existing information retrieval techniques. Even though these are well known within the IR field, to our knowledge, they have never been utilized this way. This is especially for true for the biomedical domain, which is our area of focus.

What we propose, is that we extend a traditional search engine with a *user relevance feedback* process, where the user with the results after a query actively rates a subset of the results as either *relevant* or *irrelevant*. The user rated results are then used as input for training a naive Bayes *text classification* (see Section 2.3) function which we rebuild using other unlabelled data for training with the help of the *expectation maximization* (EM) algorithm. Using EM, the search results is then clustered into a relevant and a irrelevant cluster, and only the documents from the relevant cluster are returned, weeding out more irrelevant ones from the search results than the retrieval system could originally manage. As far as we know, this has never been tried before. The hypothesis is, that by including the presented method, the mean average precision (see Section 2.4.6) will increase, and thus improving the performance of the system.

We will know explain our idea in further detail, emphasizing on the methods, how we intend to use them and why it should work.

4.2 Relevance feedback

An important aspect of our idea is *relevance feedback*. This is an interaction cycle between the user and the system, where the user selects a small set of documents from the result set that appear to be relevant to the query [2]. The results are then reordered or re ranked. Traditionally, relevance feedback is used in conjunction with query expansion (see Section 3.1.1), but in our approach this technique is to manually classify documents as relevant or irrelevant, and use them as labeled training data for a classifier.

Relevance feedback is shown to improve retrieval relevance, as it exploits the fact that it can be hard to express a good query for an information need, but it is easy to assess the relevance of the documents returned [11][2]. We believe the same applies for our method, and that the relevance feedback will provide us with a better clustering than if we were to use random initialisation for the clusters.

4.3 Naive Bayes classifier

For classifying the search results into clusters of relevant and irrelevant documents we will use a *naive Bayes classifier*. This is a well known classifier from machine-learning which is based on estimating $P(A|B)$, the probability or probability density of features A given class B, and utilizing a property known as the Bayes' theorem [5]:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \tag{4.1}$$

In naive Bayes classification, documents are classified as the most probable, or *maximum a posteriori* (MAP[1]) class [11]:

$$c_{map} = \arg\max_{c}\ P'(c) \prod_{i=1}^{n} P'(t_k|c) \tag{4.2}$$

which means for each class, multiply together the conditional probability of each feature, given that class, and select the one with the largest posterior probability. The parameters are estimated using *maximum likelihood estimation* (MLE), which for naive Bayes is approximated using simple frequencies from the training set. For prior category distribution, the estimate is [11]:

$$P'(c) = \frac{N_c}{N} \tag{4.3}$$

where N_c is the number of times class c showed up in the training data, and N is the total number of training documents. The estimate for feature probability distributions is the probability of term t belonging

[1]Not to be confused with mean average precision (Section 2.4.6), which also has the acronym MAP. For consistency, we will refer to MAP as the IR evaluation measure.

to class c and is computed similarly [11]:

$$P'(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}{}'} \tag{4.4}$$

where T_{ct} is the number of times term t appeared in a document labelled c during training, with the denominator being the total number of occurrences of c for all terms.

This MLE suffers from the *zero probability problem.* If a particular feature, or term, is never seen together with a particular class during training, the MLE estimates zero probability. Just because a particular observation was not made in the training data does not mean we will not encounter it in the test data. To eliminate this issue, we use *additive* or *Laplace smoothing*, which simply adds one to each term count [11].

A naive Bayes classifier assumes that each feature of a class is conditionally independent of the other features that make up the class. Hence the term *naive.* This dramatically reduces the number of parameters that needs to be estimated and simplifies the training step [13]. It is easy to see why the class-conditional independence between features is not true. We know that when a term occurs once in a document, it is more likely to occur again. There is also a topical relation between the terms. A document with the terms "Bayes" and "classifier" is much more likely to also contain "naive" than any other random document. It has been shown that despite the simple assumption of class independence, even when violated, naive Bayes classifiers performs well [9].

Our choice of using naive Bayes is the appeal of it's simplicity because of the previously discussed assumption. This makes it blazingly fast computationally and only requires a small amount of training data as the assumed independence of features does not degrade it's predictive accuracy.

For supervised training data we will use the relevance judgements provided by the user as described in Section 4.2.

4.4 Expectation Maximization

Expectation Maximization (EM) is a statistical technique, or framework, for maximizing likelihood estimates with missing data [4]. It is not only useful for problems involving incomplete data, but also for problems that can be posed in a similar form, such as mixture model estimation [12], which for naive Bayes is multinomial or *a bag of words* [17].

EM is a form of *model-based clustering.* Model-based clustering assumes that the data were generated by a model and tries to recover the original model from the data. The model that we recover then defines assignment of documents to clusters [11].

The algorithm tries to find the parameters Θ that maximize the log-likelihood of generating the data D: [11]

$$\begin{aligned}\Theta &= \arg\max_{\Theta} L\left(D|\Theta\right)\\ &= \arg\max_{\Theta} \log \prod_{n=1}^{N} P\left(d_n|\Theta\right) \qquad (4.5)\\ &= \arg\max_{\Theta} \sum_{n=1}^{N} \log P\left(d_n|\Theta\right)\end{aligned}$$

where $L\left(D|\Theta\right)$ is the objective function that measures the goodness of the clustering. Given two clusterings with the same number of clusters, we prefer the one with higher $L\left(D|\Theta\right)$ [11].

The way EM does this is by iteratively computing two steps known as *expectation*, corresponding to reassignment, and *maximization* corresponding to recomputing the parameters of the model [16]. These steps also form the algorithm's name.

In a semi-supervised setting, with both labelled and unlabelled data, we still need to calculate the maximum a posteriori as in Section 4.3. The expectation step estimates the expectations of the missing values given the latest iteration of the model parameters [15]. In our setting, the E-step consists of using Equation 4.2 to classify each unlabelled document, but instead of selecting the class with the highest probability, we note the probability that each cluster generated the document. This set of assignment probabilities defines a soft clustering, were the documents have fractional membership in the two clusters "relevant" and "irrelevant".

The maximization step maximizes the likelihood of the model parameters using the previously-computed expectations of the missing values as if they were the true ones [15]. This means re estimating the maximum a posteriori estimate with the fractions computed in the E-step as true class labels.

In our case, the labeled data only, supplied by the user through relevance feedback, is used to build an initial naive Bayes classifier. The classifier is then iteratively re estimated with the unlabelled documents until it reaches N iterations or the classifier does not change from one iteration to the next. This is measured by a below-threshold change in the log probability of the parameters (Equation 4.6). For a full description of the algorithm, see Table 4.1.

In most cases, using EM yields significantly better classifiers then when just using labelled data alone [15][16], especially when there are only a few labelled documents. This is one of the reasons we are testing it in our approach. Given that the retrieval system performs reasonably well, it is doubtful that the user will provide enough relevance feedback for us to build an effective classifier using just naive Bayes alone. The unlabelled training data to help with estimation will be the subset of documents from the search results that the user has not provided relevance judgements for.

Finally it should be noted that there is nothing naive Bayes de-

- **Inputs:** Collections X_l of labelled documents and X_u of unlabelled documents.
- Build an initial naive Bayes classifier, $\hat{\theta}$, from the labelled documents, X_l, only. Use maximum a posteriori parameter estimation to find $\hat{\theta} = \arg\max_\theta P(X_l|\theta)P(\theta)$
- Loop while classifier parameters improve, as measure by the change in $l(\theta|X, Y)$ (the log probability of the labelled and unlabelled data, and the prior):
  - **(E-step)** Use the current classifier, $\hat{\theta}$, to estimate component membership of each unlabelled document, *i.e.*, the probability that each mixture component (and class) generated each document, $P(c_j|x_i; \hat{\theta})$
  - **(M-step)** Re-estimate the classifier, $\hat{\theta}$, given the estimated component membership of each document. Use maximum a posteriori parameter estimation to find $\hat{\theta} = \arg\max_\theta P(X, Y|\theta)P(\theta)$
- **Output:** A classifier, $\hat{\theta}$, that takes an unlabelled document and predicts a class label.

Table 4.1: The EM algorithm for semi-supervised learning of a naive Bayes text classifier [15].

pendent about expectation maximization itself, it can be applied to any type of probabilistic model that computes $P(A|B)$. Our reasons for using naive Bayes is presented in Section 4.3.

Chapter 5

Implementation

Previously we have described our approach to increasing the precision in biomedical search engines. This chapter explains the implementation specific details and the technology behind the prototype created for testing and evaluating our idea. The prototype is a vector space model search engine extended with expectation maximization clustering.

5.1 Technologies

This section presents the tools used to develop our prototype. We rely heavily on these as it allows us to focus on the problem at hand by building on the tried and testing work of others. As both the tools are written in the Java[1] programming language, it naturally became the language of choice for our prototype as well.

[1]Available from `http://java.com`

5.1.1 Apache Lucene

Apache Lucene[2] is an open-source search engine library written in Java. Despite it's simplicity it performs well and it is easy to index and search large document collections. Our prototype is based on Lucene, with our own expansions for expectation maximization clustering.

5.1.2 LingPipe

LingPipe[3] is a tool kit for computational linguistics text processing. It is written in Java and contains a vast array of features, several of them tailored for the biomedical field and the library integrates well with Lucene.

Our prototype uses version 4.0.1 of LingPipe.

5.2 Architectural Overview

The prototype has two main features, namely indexing MEDLINE abstracts and retrieving them. Even though they both operate on the same data, we think of them as two very different processes and they will be presented as such. First we will describe how we built the index, then how we query it. As the goal of this thesis is to improve performance through expectation maximization, the majority of this chapter will be devoted to the latter's (more advanced) implementation.

[2]Available from `http://lucene.apache.org/`

[3]Available from `http://alias-i.com/lingpipe/`

5.3 Parsing MEDLINE

The parsing of MEDLINE citations in XML (see Appendix B for a sample citation) is handled by the class MyMedlineHandler, which implements the MedlineHandler interface from the LingPipe project. This class receives callbacks with MedlineCitation objects as the sole argument, which we then pass on to our indexer through the addDocument↩
→() function. As we do not care for the later released updates to the dataset where citations may be removed, we have not implemented the required delete() method.

As this is an event driven model, this lets us bypass the limit of available memory, which would a case with a DOM (Document Object Model) parsing where the entire XML would be loaded into memory before processed. In practice that would be infeasible considering the size of the collection.

```java
public class MyMedlineHandler implements MedlineHandler {

  private MedlineIndexer mIndexWriter;

  public MyMedlineHandler(MedlineIndexer indexWriter){
    mIndexWriter = indexWriter;
  }

  @Override
  public void delete(String pmid) {
    throw new UnsupportedOperationException("not expecting ↩
        →deletes. got pmid= " + pmid);
  }

  @Override
  public void handle(MedlineCitation citation) {
    mIndexWriter.addDocument(citation);
  }
}
```

It is worth mentioning that the TREC 2004 Genomics Track collection is split into multiple several gigabyte XML files, which turned out to be a problem as the LingPipe library could not process the split collection. A small Python[4] script (see Appendix D) was written for the purpose of combining these into a single file suitable for handling by our prototype.

5.4 Indexing MEDLINE

The indexer in the prototype is initialized and run with the following code:

```java
mMedlineCodec = new SearchableMedlineCodec();
mIndexWriter = new IndexWriter(FSDirectory.getDirectory(↩
    →mIndex), mMedlineCodec.getAnalyzer(), new IndexWriter.↩
    →MaxFieldLength(IndexWriter.DEFAULT_MAX_FIELD_LENGTH));

MedlineParser parser = new MedlineParser(true);

MyMedlineHandler handler = new MyMedlineHandler(this);

parser.setHandler(handler);

System.out.println("Processing file: " + mXMLFilepath);

InputSource inputSource = new InputSource(mXMLFilepath);
parser.parse(inputSource);

System.out.println("Started optimizing index.");
mIndexWriter.optimize();
mIndexWriter.close();

System.out.println("Processing complete.");
```

and the `addDocument()` function called from our handler mentioned in Section 5.3 as such:

```java
void addDocument(MedlineCitation citation) {
    Document doc = mMedlineCodec.toDocument(citation);
    mIndexWriter.addDocument(doc);
}
```

[4] `http://www.python.org/`

The interesting bit here is the SearchableMedlineCodec. This class transforms MedlineCitation objects to Lucene Document objects which are indexable by Lucene. This class also contains a LuceneAnalyzer↩
→. An analyzer to Lucene is how it breaks a stream of characters into tokens, a policy for how to extract index terms from text. The analyzer used by our prototype is almost like the regular Lucene StandardAnalyzer which strips-off punctuation, lowercases and filters out common stop words. The main difference is that we allow digits in our tokens. This is prevent degrading the biomedical information, which would be the case for a term such as ferroportin1.

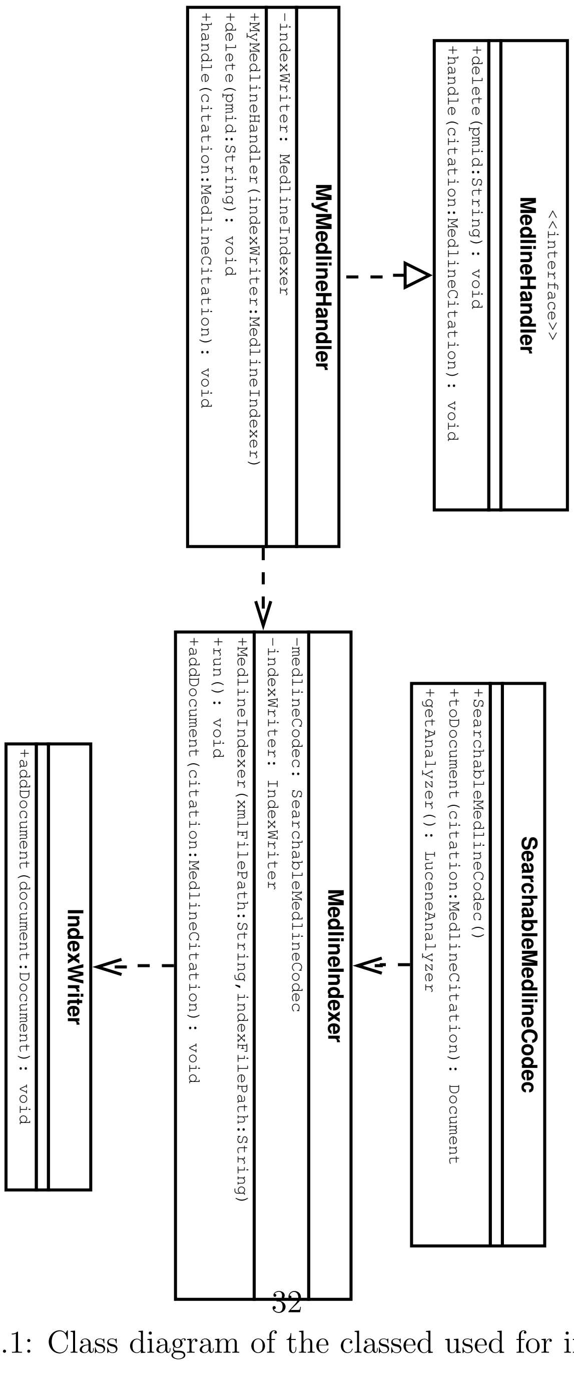
A MEDLINE citation contains a rich amount of structured data (see Appendix B for an example) such as title, abstract, journals and authors amongst other things. Documents created using SearchableMedlineCodec↩
→ allows us to search over all these elements in the resulting index, giving us a rich set of searchable fields which can be utilized during retrieval. Primarily in our prototype, we only make use of the title, abstract and PMID.

For building the index itself we simply use Lucene's standard IndexWriter↩
→. Once the index is built we call optimize() on it, priming it for the fastest available search.

See Figure 5.1 for a simplified class diagram of the classes used for indexing in our prototype.

5.5 Retrieval

For document retrieval we have a Search class. This contains an instance of LingPipe's MedlineSearcher through which we run a list of queries. More precisely there is a query tailored for each topic. The MedlineSearcher uses Lucene's standard IndexSearcher and SearchableMedlineCodec↩
→ to understand and query our index. This ensures that the analyzer used to parse the queries is the same as the one used to index doc-



<<interface>>
MedlineHandler
+delete(pmid:String): void
+handle(citation:MedlineCitation): void
MyMedlineHandler
-indexWriter: MedlineIndexer
+MyMedlineHandler(indexWriter:MedlineIndexer)
+delete(pmid:String): void
+handle(citation:MedlineCitation): void
SearchableMedlineCodec
+SearchableMedlineCodec()
+toDocument(citation:MedlineCitation): Document
+getAnalyzer(): LuceneAnalyzer
MedlineIndexer
-medlineCodec: SearchableMedlineCodec
-indexWriter: IndexWriter
+MedlineIndexer(xmlFilePath:String,indexFilePath:String)
+run(): void
+addDocument(citation:MedlineCitation): void
IndexWriter
+addDocument(document:Document): void

Figure 5.1: Class diagram of the classed used for indexing.

uments, a necessity for finding the indexed tokens when querying. All this is pretty straightforward, the interesting bit is where we filter out false positives based on the naive Bayes classifier built using expectation-maximization clustering (see Section 5.6 for the implementation details for EM):

```java
SearchResults<MedlineCitation> results = medlineSearcher.search(query);

TradNaiveBayesClassifier emClassifier = ExpectationMaximization.em(topicId, results, reporter);

for (int i = 0; i < (results.size() < 1000 ? results.size() : 1000); ++i) {
    MedlineCitation ml = results.getResult(i);

    if (emClassifier.classify(MedlineCodec.titleAbstract(ml)).bestCategory() == "Relevant") {
        (...)
    }
}
```

The prototype automatically saves the output from retrieval to plain text files compatible with the `trec_eval` tool. An excerpt from a text file produced by a run with our prototype yields:

```
1 Q0 10693807 1 4.039227962498965 tag1
1 Q0 11809412 2 3.717294454574585 tag1
1 Q0 12091366 3 3.575765371322632 tag1
1 Q0 12091367 4 2.615002155303955 tag1
```

where each column is defined as: [6]

- The first column is the topic number (1-50).
- The second column is the query number within that topic. This is currently unused and must always be Q0.
- The third column is the official PubMedID of the retrieved document.
- The fourth column is the rank the document is retrieved.

- The fifth column shows the score (integer or floating point) that generated the ranking.
- The sixth column is called the “run tag” and must be a unique identifier across all runs submitted to TREC.

The file is sorted by topic order and then subsorted by the generated ranking score as is required by the `trec_eval` tool. There are two files produced each run, one with the regular query results and one with the “relevant” results after clustering the answer set. This is so we are able to compare the results and evaluate the performance of our prototype.

See Figure 5.2 for a simplified class diagram of all the classes used for retrieval in our prototype, including the EM classes.

5.6 Expectation Maximization with Naive Bayes

For clustering we have a `ExpectationMaximization` class. It begins by declaring several constant declarations. These are inputs to both the EM algorithm itself and the naive Bayes classifier. By altering these we are able to affect the results produced by our prototype, as these correlate to many variables from Chapter 4.

```java
static final int MAX_ITER = 25;
static final double THRESHOLD = 0.0001;
static final double TOKEN_COUNT = 0.0001;
static final double DOC_LENGTH_NORMALIZER = 9.0;
static final double CATEGORY_PRIOR = 0.005;
static final double TOKEN_IN_CATEGORY_PRIOR = 0.001;
static final double INITIAL_TOKEN_IN_CATEGORY_PRIOR = 0.1;
```

The input to this class is a `ResultSet` of documents produced by queries (as presented in Section 5.5). The class then defines the corpora of the labelled and unlabelled data respectively, which will be

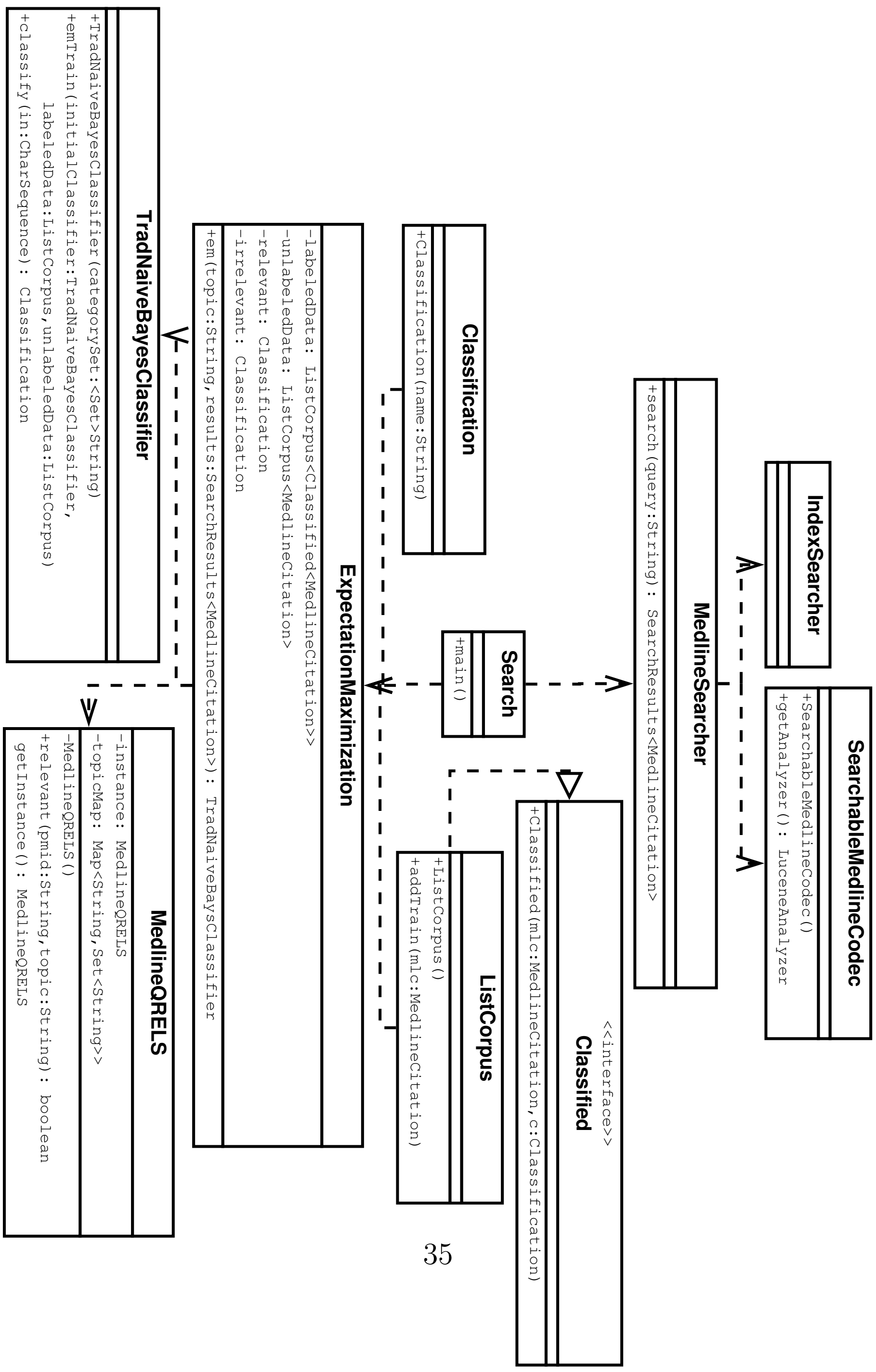

TradNaiveBayesClassifier
+TradNaiveBayesClassifier(categorySet:<Set>String)
+emTrain(initialClassifier:TradNaiveBayesClassifier,
labeledData:ListCorpus,unlabeledData:ListCorpus)
+classify(in:CharSequence): Classification
ExpectationMaximization
-labeledData: ListCorpus<Classified<MedlineCitation>>
-unlabeledData: ListCorpus<MedlineCitation>
-relevant: Classification
-irrelevant: Classification
+em(topic:String,results:SearchResults<MedlineCitation>): TradNaiveBaysClassifier
Classification
+Classification(name:String)
Search
+main()
MedlineSearcher
+search(query:String): SearchResults<MedlineCitation>
IndexSearcher
SearchableMedlineCodec
+SearchableMedlineCodec()
+getAnalyzer(): LuceneAnalyzer
MedlineQRELS
-instance: MedlineQRELS
-topicMap: Map<String,Set<String>>
-MedlineQRELS()
+relevant(pmid:String,topic:String): boolean
getInstance(): MedlineQRELS
ListCorpus
+ListCorpus()
+addTrain(mlc:MedlineCitation)
<<interface>>
Classified
+Classified(mlc:MedlineCitation,c:Classification)

Figure 5.2: Class diagram of the classed used for retrieval.

used to train the Naive Bayes classifier. The labelled data consists of `Classified` character sequences for the supervised training, whereas the unlabelled data for unsupervised training, is just plain character sequences. In our case a character sequence will be a MEDLINE citation, or more precisely a citation's title and abstract. The naive Bayes classifier is setup to extract tokens from these text sequences the exact same way as when the prototype builds it's indices (see Section 5.4) for no other reason than coherence.

As previously explained (Chapter 4), a document can be classified as either "relevant" or "irrelevant" by a user, which also defines our clusters.

```
ListCorpus corpus = new ListCorpus<Classified<String>>();
ListCorpus unlabeledCorpus = new ListCorpus<String>();

Classification relevant = new Classification("Relevant");
Classification irrelevant = new Classification("Irrelevant");
```

Training

As the prototype runs through numerous queries for evaluation purposes we have defined the class `MedlineQRELS` as a way to automatise the supervised training process. This lets the prototype itself take on the user role and provide relevance feedback for the top N search results, and spares us from a time consuming task when constantly testing.

The way we have implemented this is that `MedlineQRELS` parses the file with relevance judgements supplied with the TREC MEDLINE collection, and builds a mapping between all topics and their set of relevant documents. The file is on the format: [6]

$$<topic> <0> <PMID> <judgment>$$

where *judgement* = 1 (definitely relevant), 2 (possibly relevant), or 3 (not relevant). We followed the convention for the TREC official results which requires binary relevance judgements, so documents that are rated definitely relevant or possibly relevant were considered relevant [6].

Given a MEDLINE citation PMID and topic number we can then easily lookup that citation's relevance for that topic through the function `relevant()`:

```java
public boolean relevant(String pmid, String topic) {

    Set<String> set = mTopicMap.get(pmid);

    return set != null ? set.contains(topic) : false;
}
```

As the `MedlineQRELS` requires some time to load due to the parsing, we have decided to implement it using the singleton pattern[5] to keep the number of instances to one.

The following code demonstrates then how we set up the labelled and unlabelled training data. We argue that the user is not likely to bother classifying more than at most three documents to each category when giving relevance feedback, so the prototype exhibits this behaviour. All the documents that are not classified will be assigned as unlabelled training data to help with estimation.

```java
int trainedRelevant = 0;
int trainedIrrelevant = 0;

// set i to result set size or max 1000 to prevent ↩
    →indexoutofboundsexception
for (int i = 0; i < (results.size() < 1000 ? results.size() :↩
    → 1000); ++i) {
    MedlineCitation ml = results.getResult(i);
    boolean relevance = MedlineQRELS.getInstance().relevant(↩
        →ml.pmid(), topic.getID());
    if (relevance && trainedRelevant < 3) {
        Classified<String> classified;
        classified = new Classified<String>(MedlineCodec.↩
            →titleAbstract(ml), relevant);
```

[5]A design pattern that ensures that only one instance of a class is created.

```
        corpus.addTrain(classified);
        ++trainedRelevant;
    } else if (!relevance && trainedIrrelevant < 3) {
        Classified<String> classified;
        classified = new Classified<String>(MedlineCodec.↩
           →titleAbstract(ml), irrelevant);
        corpus.addTrain(classified);
        ++trainedIrrelevant;
    } else {
        unlabeledCorpus.addTrain(MedlineCodec.titleAbstract(↩
           →ml));
    }
}
```

Finally we create the `TradNaiveBayesClassifier` from the LingPipe project which is re estimated until it reaches below a certain threshold or after a certain number of iterations.

```
emClassifier = TradNaiveBayesClassifier.emTrain(↩
   →initialClassifier, corpus, unlabeledCorpus, MAX_ITER, ↩
   →THRESHOLD);
```

See Figure 5.2 for a simplified class diagram of all the classes used for retrieval, including the EM classes.

Chapter 6

Testing and results

This chaper explains how we tested our prototype, presents the results we gathered and their evaluation.

6.1 Test collection

To evaluate the performance of our prototype we have tested against the TREC 2004 Genomic Track test collection. We believe this collection to be best suited within the genomic domain with it's number of vastly different topics covered.

The TREC 2004 Genomics Track collection consists of 4,591,008 MEDLINE citations. A subset of these citations, namely 42,255, have been judged against 50 topics. Each of these topics consists of a *title*, *need* and a *context* field (see Appendix C for an example topic). The judged records have relevance judgements, being either *definitely relevant*, *possibly relevant* or *not relevant* to a given topic [6]. In our testing we do not differ between definitely relevant and possibly relevant and consider them both relevant. This is in standing with the official TREC protocol for results, which requires binary relevance judgements [6].

6.2 The queries

For queries there were several approaches to automatically generating queries tested. First, the *"title"* field from each topic was used as queries. Then a combination of the *"title"* and *"need"* fields and lastly, a combination of *"title"*, *"need"* and *"context"* was tested. However, all these approaches were deemed to give too low recall to provide any feasible results, even when weighting the *"title"* field twice as much when searching, as done in BioTracer [18]. Finally we decided using the same queries as Jervidalo [7], which with the previously described "title" weighting produced usable results.

The full set of queries for each topic can be seen in Appendix A.

6.3 Evaluation method

To evaluate the performance of our prototype we use measures for retrieval. As described in Chapter 5 the prototype outputs the results so they are compatible with Buckley's `trec_eval`[1] program (version 9.0), the standard tool used when evaluating TREC collections. Our focus is mainly on mean average precision, as it is the most standard measure among the TREC community [11]. However, we also measure the precision at 10 and 100, because we know that the user is likely to look at the top most results [20][23]. These are also the same measures used when evaluating BioTracer [18].

[1]See `http://trec.nist.gov/trec_eval/`

6.4 Test environment

The hardware and software specifications for the machine used during testing of the prototype is listed in Table 6.1. Most of these specifications are included for reference purposes only. There should be nothing system dependent about our prototype that affects the results and results and findings in this thesis. Note that the Java VM arguments were critical when our prototype was indexing the test collection, as it required more memory than the default settings to run. However, this was not necessary during retrieval and clustering.

CPU:	Intel Core 2 Duo CPU 2.40 GHz
Memory:	2 GB
Disk:	160 GB S-ATA 7200RPM
OS:	Ubuntu Natty Narwhal 11.4
Java version:	OpenJDK 6 (1.6.0_22)
Java VM arguments:	-Xms128m -Xmx512m

Table 6.1: Hardware and software environment used.

6.5 Results

In this section we present the results from running our prototype. The average measurements for the 50 queries from Appendix A are presented in Table 6.2. We can see that there is a slight increase in mean average precision (MAP), which is our measurement of focus, when applying EM to the regular search results. Figure 6.1 shows a graph comparison of map for baseline VSM and our VSM extended with EM approach.

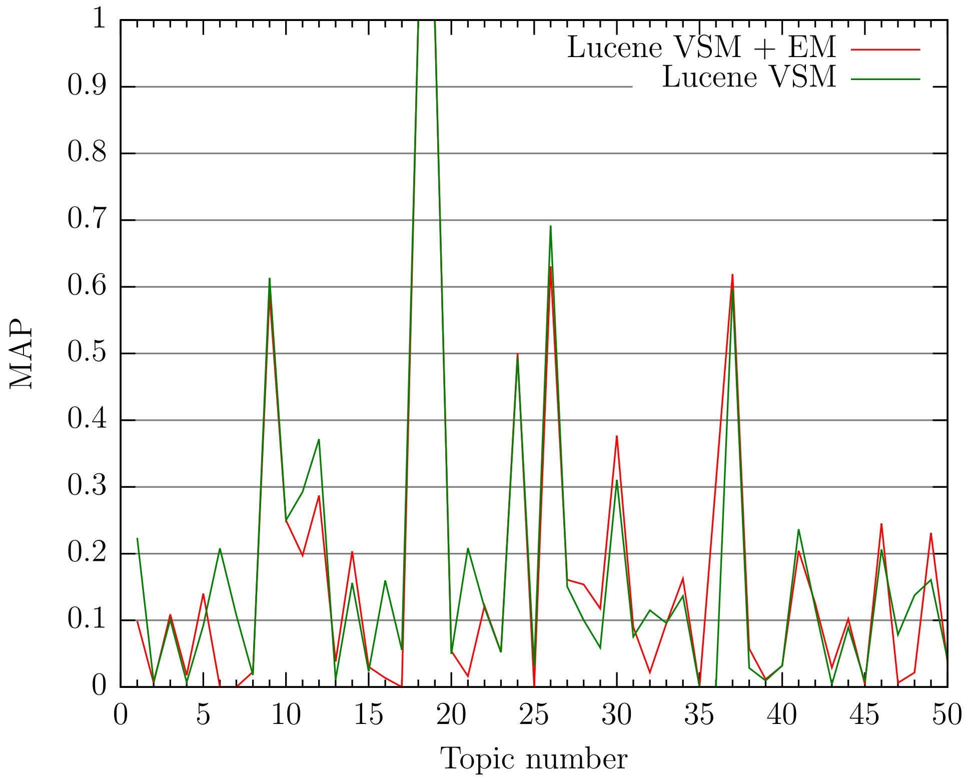


Figure 6.1: A comparison of MAP for VSM and VSM+EM

	MAP	R-precision	P@10	P@100
Lucene VSM	0.1690	0.2232	0.3918	0.1798
Lucene VSM + EM	0.1827	0.2506	0.4200	0.2058

Table 6.2: Evaluation results

6.6 Discussion

As seen in the previous section, there is some, but little overall increase in performance when extending our baseline vector space model with expectation maximization and naive Bayes. However, the graph in Figure 6.1 shows several fluctuations. Even though VSM+EM performs better on average, there are some topics where regular VSM significantly outperforms our approach. It looks like these are mainly the topics where there are few relevant documents and that the classifier gets better "irrelevant" training than "relevant", which affects the clustering negatively. Another possible reason for the fluctuations could be to the fact that a MEDLINE citation may only contain a title and no abstract [18], documents like this could be a bad source of training examples for the classifier.

Chapter 7

Conclusion

In this thesis we sought to improve the performance of biomedical search using the statistical expectation maximization technique. We have defined an approach where we use EM together with traditional naive Bayes classification and user relevance feedback to cluster the results into “relevant” and “irrelevant” documents. We proposed that this approach would increase precision as it would filter out irrelevant documents from the search results. To test our proposal, we have developed a retrieval system prototype that implements our idea. The prototype also featured “regular retrieval” as a means of producing results that were comparable with the extended retrieval that was our approach. The prototype has been evaluated against a biomedical test collection using well known evaluation measures.

Given the data collected in evaluation our prototype we have determined that using expectational maximization has little, but a positive effect on overall precision. However, in some special cases the performance would even degrade. The user relevance feedback cycle is an extra step, and with such little overall improvements, and in special cases worsened performance, we believe it is unlikely that the user will bother with the extra effort. Our testing shows that the initial search results the user is presented with are near close enough already to the results he is presented with after the clustering. Thus we conclude that our approach, as it currently stands, is unsuitable to increase performance in biomedical retrieval.

7.1 Further work

Our evaluation shows that there was some gain in performance, although relatively small. This could justify further research in using expectation maximization for biomedical search improvements. A suggestion by the authors is to look at other classification methods. Caruna shows that the classifiers *random forest* and *gradient tree boosters* both outperform naive Bayes [3]. At the time of this writing, there is also a not yet released paper called “Mixed-Effects Random Trees” by Ahlem Hajjem, from a newly held conference[1] on using mixed random forest together with EM. The abstract looks promising, and may also warrant further research when it becomes available.

[1]Statistics 2011 Canada / IMST 2011-FIM XX, July 2011

Bibliography

[1] Samir Abdou and Jacques Savoy. Searching in Medline: Query expansion and manual indexing evaluation. *Inf. Process. Manage.*, 44:781–789, March 2008.

[2] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 161–168, New York, NY, USA, 2006. ACM.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[5] Pedro Domingos and Michael Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss, 1997.

[6] William R. Hersh, Ravi Teja Bhuptiraju, Laura Ross, Phoebe Johnson, Aaron M. Cohen, and Dale F. Kraemer. TREC 2004 genomics track overview. In *In Proc. of the 13th Text REtrieval Conference*, 2004.

[7] Jørgen Jervidalo. Improving Performance of Biomedical Information Retrieval using Document-Level Field Boosting and BM25F Weighting. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, June 2010.

[8] Mika Käki and Anne Aula. Findex: improving search result use through automatic filtering categories. *Interacting with Computers*, 17(2):187 – 206, 2005.

[9] Kim Larsen. Generalized Naive Bayes Classifiers. *SIGKDD Explor. Newsl.*, 7:76–81, June 2005.

[10] Zhiyong Lu, Won Kim, and W. John Wilbur. Evaluation of query expansion using mesh in pubmed. *Inf. Retr.*, 12:69–80, February 2009.

[11] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA, 2008.

[12] Geoffrey J. Mclachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions.* Wiley-Interscience, 1 edition, November 1996.

[13] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, New York, NY, USA, 1997.

[14] Stefano Mizzaro. Hits hits TREC: exploring IR evaluation results with network analysis. In *In Proc. 30th Ann. Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 479–486, 2007.

[15] Kamal Nigam, Andrew McCallum, and Tom Mitchell. Semi-supervised Text Classification Using EM. In *Semi-Supervised Learning*, pages 33–56. MIT Press, Boston, 2006.

[16] Kamal Nigam, Andrew Kachites Mccallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using em. In *Machine Learning*, pages 103–134, 1999.

[17] Dmitry Pavlov, Ramnath Balasubramanyan, Byron Dom, Shyam Kapur, and Jignashu Parikh. Document preprocessing for naive Bayes classification and clustering with mixture of multinomials. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 829–834, New York, NY, USA, 2004. ACM.

[18] Heri Ramampiaro. Biomedical Information Retrieval: The BioTracer Approach. In Sami Khuri, Lenka Lhotská, and Nadia Pisanti, editors, *Information Technology in Bio- and Medical Informatics, ITBAM 2010*, volume 6266 of *Lecture Notes in Computer Science*, pages 143–157. Springer Berlin / Heidelberg, 2010.

[19] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18:613–620, November 1975.

[20] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33:6–12, September 1999.

[21] Amit Singhal. Modern information retrieval: a brief overview. *BULLETIN OF THE IEEE COMPUTER SOCIETY TECHNICAL COMMITTEE ON DATA ENGINEERING*, 24:2001, 2001.

[22] Neil Smalheiser, Wei Zhou, and Vetle Torvik. Anne O'Tate: A tool to support user-driven summarization, drill-down and browsing of PubMed search results. *Journal of Biomedical Discovery and Collaboration*, 3(1):2, 2008.

[23] Amanda Spink, Bernard J. Jansen, Dietmar Wolfram, and Tefko Saracevic. From E-Sex to E-Commerce: Web Search Changes. *Computer*, 35:107–109, March 2002.

[24] C. J. Van Rijsbergen. *Information retrieval.* Butterworths, London, 2nd edition, 1979.

[25] Oren Zamir, Oren Etzioni, Omid Madani, and Richard M. Karp. Fast and intuitive clustering of web documents. In *In Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 287–290, 1997.

Appendix A

Queries

This is a list of the queries used for evaluating our prototype. The queries are aquired from Jervidalo [7].

```
iron AND ( Ferroportin1 OR ( Ferroportin AND 1) OR SLC40A1 OR FPN1 OR HFE4 OR IREG1 OR ( Iron AND regulated AND gene AND 1) OR MTP1 OR SLC11A3)
(transgenic OR transgenesis OR ( copy AND gene ) ) AND (mice OR mouse OR murine)
(mouse kidney ) ( gene expression ) ( kidney development ) ( kidney )
kidney AND (( gene OR genes ) OR ( expression AND ( profile OR profiles ) ) ) AND (mice OR mouse OR murine)
( isolate OR isolating OR fractionation OR purify ) AND ( cell OR ( nucleus OR nuclei ) OR subcellular )
FancD2 OR ( Fanconi AND anemia) OR ( group D2) OR ( type AND 4 AND fanconi AND pancytopenia )
(( oxidative OR oxidation ) AND stress ) AND DNA AND repair
( oxidative OR oxidation ) OR ( cancer OR cancers OR carcinoma OR cancerous ) AND ( disease OR diseases OR carcinogenesis ) AND ( gene OR pathway) AND (DNA AND repair )
(muty OR hmyh) AND -myoglobin
(NEIL1)
hairless mice carcinogenesis skin OR UV
( gene OR genes ) AND smad4
(TGFB OR ( transforming growth factor beta ) OR (TGF) ) AND ( homeostasis OR angiogenesis )
```

```
(TGFB OR ( transforming growth factor beta ) ) AND (( head and neck squamous cell ) OR HNSCC)
(ATPase OR ATPases) AND ( apoptosis OR ( cell death ) )
(AAA proteins ) ( lipids ) (AAA protein family ) ( protein interactions )
(DO1 OR (p53 AND ( antibody OR anti ) ) ) AND binding
( Gis4 OR YML006C)
(GAL1 OR SUC1) AND ( repressors OR reprosessor OR activators OR activator ) AND SNF1
( covalent OR attachment OR covalence OR substrate ) AND ( ubiquitin OR ubiquitously OR ubiquitylation OR ubiquitination )
(p63 OR TP63) OR (TP73 OR p73) (( cell cycle arrest ) OR apoptosis ) DNA
p53 AND DNA AND ( respond OR responding OR response ) AND ( break OR damage OR (( single OR double ) AND stranded ) )
Saccharomyces OR cerevisiae ( protein OR proteins ) ( ubiquitin OR proteolytic OR pathway )
(PGRP) (mouse AND peptidoglycan AND recognition AND proteins )
( scleroderma OR (autoimmune AND disease ) ) AND (( genes OR gene OR genome) OR ( scan OR scans ) OR ( microarray OR ( micro AND array ) ) )
(BFA1) (BUB2)
( autophagy OR ( gene autophagic ) ) AND apoptosis
( autophagy OR ( gene autophagic ) ) AND apoptosis AND ( proteases OR morphological )
(gyrA OR (DNA gyrase ) ) AND (( phenotype OR phenotypes ) OR ( sequence OR sequences ) ) AND ( mutation OR mutations OR alteration ) AND ((E AND coli ) OR escherichia )
Nkx OR Sax
(TOR OR mTOR OR ( Target AND Of AND Rapamycin) OR FRAP1 OR (FK506 AND associated AND protein ) )
Xenograft AND ( tumorogenesis OR cancer OR cancers OR carcinoma )
( histoplasmosis OR ( histoplasma AND capsulatum ) OR ( blood borne pathogen ) ) AND (mice OR mouse OR murine)
Cryptococcus AND ( gene OR genes OR genome)
( histoplasmosis OR ( histoplasma AND capsulatum ) OR ( blood borne pathogen ) ) AND (mice OR mouse OR murine)
Cryptococcus AND ( gene OR genes OR genome)
PAM OR ( peptide AND amidating AND enzyme) OR ( peptidylglycine AND amidating )
```

```
38 ( stroke OR ( cerebrovascular AND accident ) OR CVA) AND (( ↩
     →genetic AND ( loci OR locus ) ) OR E4 OR ( factor AND V)↩
     → OR ( risk AND ( factor OR factors ) ) )
39 ( hypertension OR HTN OR ( high AND blood AND pressure ) ) ↩
     →AND (( ris k OR danger ) AND ( genes or gene ) )
40 ( antigen OR antigens ) AND ( epithelial OR epithelium ) ( ↩
     →lung OR pulmonary OR lungs )
41 ( mutation OR mutations OR altered ) AND (( Cystic AND ↩
     →Fibrosis ) OR CF OR mucovoidosis OR muscoviscidosis )
42 ((chromosome OR chromosomal) AND ( translocations OR ↩
     →translocation ) ) OR ((chromosome OR chromosomal) AND ( ↩
     →rearrangement OR rearrangements ) )
43 ( sleeping AND beauty ) OR ( Kleine AND Levin AND Syndrome) ↩
     →OR KLS
44 (( nerve AND growth AND factor ) OR NGF) AND ( protein OR ↩
     →proteins )
45 (MWH1 OR ( mental health wellness ) ) OR ( mental ( disorder ↩
     →OR disorders ) ( gene OR genes OR genetic ) )
46 RSK2 OR ( ribosomal protein kinase )
47 (BCL OR BCL2 OR (BCL AND 2) ) AND (( antagonists OR ↩
     →antagonist ) OR ( inhibitors OR inhibitor ))
48 (UNC OR ( homologues OR homolog) OR BGS) AND (( gene OR genes↩
     → ) OR ( c AND elegans ) OR ( Caenorhabditis AND elegans ↩
     →) )
49 ( glyphosate OR glycine ) AND ( tolerance OR tolerant OR ↩
     →immune)
50 ( temperature OR cold ) AND protein AND ((E AND coli ) OR ↩
     →escherichia )
```

Appendix B

Example MEDLINE citation in XML format

```xml
<!DOCTYPE MedlineCitationSet PUBLIC "-//NLM//DTD NLM Medline,↩
    → 1st November 2003//EN" "http://www.nlm.nih.gov/↩
    →databases/dtd/nlmmedline_031101.dtd">
<MedlineCitationSet>
<MedlineCitation Owner="NLM" Status="Completed">
    <PMID>10605436</PMID>
    <DateCreated>
        <Year>2000</Year>
        <Month>01</Month>
        <Day>07</Day>
    </DateCreated>
    <DateCompleted>
        <Year>2000</Year>
        <Month>01</Month>
        <Day>07</Day>
    </DateCompleted>
    <DateRevised>
        <Year>2003</Year>
        <Month>11</Month>
        <Day>14</Day>
    </DateRevised>
    <Article>
        <Journal>
            <ISSN>0021-9525</ISSN>
            <JournalIssue PrintYN="Y">
                <Volume>76</Volume>
                <Issue>2</Issue>
                <PubDate>
                    <Year>1978</Year>
                    <Month>Feb</Month>
                </PubDate>
            </JournalIssue>
        </Journal>
        <ArticleTitle>Concerning the localization of steroids in centrioles and
            basal bodies by immunofluorescence.
        </ArticleTitle>
        <Pagination>
            <MedlinePgn>255-60</MedlinePgn>
        </Pagination>
        <Abstract>
            <AbstractText>Specific steroid antibodies, by the
            immunofluorescence technique, regularly reveal fluorescent
            centrioles and cilia-bearing basal bodies in target and nontarget
            cells. Although the precise identity of the immunoreactive steroid
            substance has not yet been established, it seems noteworthy that
            exogenous steroids can be vitally concentrated by centrioles,
            perhaps by exchange with steroids already present at this level.
            This unexpected localization suggests that steroids may affect cell
            growth and differentiation in some way different from the two-step
            receptor mechanism.
            </AbstractText>
        </Abstract>
        <Affiliation>Istituto di Anatomia e Istologia Patologica, Universita di Ferrara, Italy.</Affiliation>
        <AuthorList CompleteYN="Y">
            <Author>
                <LastName>Nenci</LastName>
                <ForeName>I</ForeName>
                <Initials>I</Initials>
            </Author>
            <Author>
                <LastName>Marchetti</LastName>
                <ForeName>E</ForeName>
                <Initials>E</Initials>
            </Author>
        </AuthorList>
        <Language>eng</Language>
        <PublicationTypeList>
            <PublicationType>Journal Article</PublicationType↩
                ↪>
        </PublicationTypeList>
    </Article>
    <MedlineJournalInfo>
        <Country>UNITED STATES</Country>
        <MedlineTA>J Cell Biol</MedlineTA>
        <NlmUniqueID>0375356</NlmUniqueID>
    </MedlineJournalInfo>
    <ChemicalList>
        <Chemical>
            <RegistryNumber>0</RegistryNumber>
            <NameOfSubstance>Steroids</NameOfSubstance>
        </Chemical>
    </ChemicalList>
    <CitationSubset>IM</CitationSubset>
    <MeshHeadingList>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Animals</↩
                ↪DescriptorName>
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Centrioles</↩
                ↪DescriptorName>
            <QualifierName MajorTopicYN="Y">ultrastructure</↩
                ↪QualifierName>
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Cilia</↩
                ↪DescriptorName>
            <QualifierName MajorTopicYN="N">ultrastructure</↩
                ↪QualifierName>
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Female</↩
                ↪DescriptorName>
```

```xml
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Fluorescent Antibody Technique</DescriptorName>
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Human</DescriptorName>
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Lymphocytes</DescriptorName>
            <QualifierName MajorTopicYN="Y">cytology</QualifierName>
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Male</DescriptorName>
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Organelles</DescriptorName>
            <QualifierName MajorTopicYN="Y">ultrastructure</QualifierName>
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Rats</DescriptorName>
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Rats, Sprague-Dawley</DescriptorName>
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Respiratory Mucosa</DescriptorName>
            <QualifierName MajorTopicYN="N">cytology</QualifierName>
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Steroids</DescriptorName>
            <QualifierName MajorTopicYN="Y">analysis</QualifierName>
        </MeshHeading>
        <MeshHeading>
            <DescriptorName MajorTopicYN="N">Trachea</DescriptorName>
        </MeshHeading>
    </MeshHeadingList>
</MedlineCitation>
</MedlineCitationSet>
```

Appendix C

Example topic from the TREC 2004 Genomics Track

```
<?xml version="1.0" encoding="UTF-8"?>
<TOPIC>
    <ID>13</ID>
    <TITLE>Role of TGFB in angiogenesis in skin</TITLE>
    <NEED>Documents regarding the role of TGFB in ↩
       →angiogenesis in skin with respect to homeostasis and↩
       → development.</NEED>
    <CONTEXT>TGFB plays a crucial role in regulating ↩
       →angiogenesis, a biological process that occurs ↩
       →during development and homeostasis, as well as ↩
       →during inflammatory perturbation.
    </CONTEXT>
</TOPIC>
```

Appendix D

Python script

```
import os

MEDLINE_ROOT_DIR = "/home/alexanbj/MEDLINE/unpacked/"

xml_files = []

for subdir, dirs, files in os.walk(MEDLINE_ROOT_DIR):
    for file in files:
        xml_files.append(file)

xml_files.sort()
print xml_files

combined=open("/home/alexanbj/MEDLINE/combined", 'w')

for file in xml_files:
    print file
    for line in open(MEDLINE_ROOT_DIR+file, 'r'):
        combined.write(line)

combined.write("</MedlineCitationSet>")
combined.close()
```