Investigation of Machine Learning Tools for Document Clustering and Classification

by

Lyudmila Borodavkina

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2000

© Lyudmila Borodavkina, MM. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author

Department of Electrical Engineering and Computer Science

Certified by

David R. Karger
Esther & Harold E Edgerton Associate Professor Of Electrical Engineering and Computer Science

Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Students
Investigation of Machine Learning Tools for Document Clustering and Classification

by

Lyudmila Borodavkina

Submitted to the Department of Electrical Engineering and Computer Science on September 1, 2000, in partial fulfillment of the requirements for the degree of Masters of Engineering in Computer Science and Engineering

Abstract

Data clustering is a problem of discovering the underlying data structure without any prior information about the data. The focus of this thesis is to evaluate a few of the modern clustering algorithms in order to determine their performance in adverse conditions. Synthetic Data Generation software is presented as a useful tool both for generating test data and for investigating results of the data clustering. Several theoretical models and their behavior are discussed, and, as the result of analysis of a large number of quantitative tests, we come up with a set of heuristics that describe the quality of clustering output in different adverse conditions.

Thesis Supervisor: David R. Karger
Title: Esther & Harold E Edgerton Associate Professor Of Electrical Engineering and Computer Science
Acknowledgments

With thanks to David Karger, Byron Dom and Shivakumar Vaithyanathan.
Contents

1 Introduction 8
  1.1 Motivation for Document Clustering 9
  1.2 Goals 10
  1.3 Background 11
    1.3.1 Model-based Clustering 13

2 Theoretical Basis 15
  2.1 Supervised vs. Unsupervised Learning 15
  2.2 Model 16
    2.2.1 Foreseeable Shortcomings 19
  2.3 Algorithms 20
    2.3.1 EM 20
    2.3.2 BIC 21

3 Document Corpus 23
  3.1 Golden Standard? 23
  3.2 Synthetic Data 25
  3.3 TREC 26
  3.4 Other Document Data Sets 26

4 Data Generation 27
  4.1 Model 27
  4.2 Implementation 28
List of Figures

4-1 Data Generation Output page ........................ 32
## List of Tables

5.1 Marginal likelihoods for synthetic and em-generated clusters for 25 feature documents ................................................................. 37
5.2 Confusion Matrix for 25 feature and 10000 documents .................. 38
5.3 KL Distances between synthetic clusters for 25 feature and 10000 documents ................................................................................. 39
5.4 Marginal likelihoods for synthetic and em-generated clusters for 50 feature documents ................................................................. 39
5.5 Confusion Matrix for 50 features and 10000 documents ............... 40
5.6 KL Distances between synthetic clusters for 50 features and 10000 documents ................................................................................. 40
5.7 Marginal likelihoods for synthetic and em-generated clusters for 1000 feature documents ................................................................. 41
5.8 Confusion Matrix for 1000 features and 100 documents ............... 41
5.9 KL Distances between synthetic clusters for 1000 features and 100 documents ................................................................................. 42
5.10 Confusion Matrix for 1000 features and 10000 documents .......... 42
5.11 KL Distances between synthetic clusters for 1000 features and 10000 documents ........................................................................... 43
Chapter 1

Introduction

This thesis is concerned with problems of data clustering and classification. **clustering** is a problem of determining an intrinsic structure of the data when no information other than the observed values is available. This problem is different from **classification**, in which known groupings of some observations are used to categorize others and infer the structure of the data as a whole [13].

Chapter two describes the theoretical basis for Bayesian model-based clustering and classification, associated evaluation methods and known optimizations and problems.

Chapter three describes the data used in research for this thesis — different document data employed, both real-world data and synthetically generated model documents.

Chapter four describes design of the data generation engine, as well as how the generation models discussed in section 2.2 were implemented. In addition, it discusses lessons learned when attempting to implement various models.

Chapter five describes the designed experimental setup and compiled results, including quantitative analysis and anecdotal evidence.

Chapter six contains a survey of alternative approaches to clustering, both for documents and other types of data, focusing on model-based work. It also presents a summary of the research and overview of directions for future work.
1.1 Motivation for Document Clustering

A past decade has seen a tremendous growth of online technologies and with them came an explosion in the amount of data that is available to people. Web pages, email, and other documents proliferate. Among them, there are well-authored informative documents, descriptive stories, personal and official communication, as well as badly-written web pages, snippets of communication logs, spam email, etc. This influx of information is unmanageable without additional tools. One possible way of presenting this information is through search engines and indexes, such as Altavista web search engine [8]. In that case, a user enters a search query and the results are displayed almost unfiltered, listing all the documents that include words from the query. The major drawback to that method is that the user must be very deft at creating a query or risk ending up with thousands of unsorted documents.

Alternative solution is to let the computer do work of determining relevance of each document. Since a machine cannot read minds to decide what’s relevant to the query and what’s not, it can instead assist humans by splitting all available documents into semantically similar groups, such that all documents on the same subject end up in the same group — in the same cluster. This process of clustering can be tremendously useful both when looking for new information, presenting existing document corpus, or attempting to make inferences from the available data.

There are some human-made clustered databases, but creation of those takes a lot of time, effort and money and is not possible for all available databases — overall, people create more documents than there are resources to make sense of them. Therefore, machine-made grouping is valuable.

At times, the number of documents in a corpus is such that even dividing it into thematic groups is not enough, because number of those groups will exceed easily-manageable one. For example, in a collection of news articles from the past decade, it is not enough to break them down into separate topics. For one, depending on granularity of those topics, they may number in the thousands.

One solution to this problem is organizing the clusters into hierarchies. That way
all groups containing documents about internal affairs will naturally fall together, just like all the documents about foreign events will end up together as well.

Once again, there are some human-generated hierarchies — taxonomies. Yahoo presents such a hierarchy for some subset of the web pages [25]. This tree hierarchy includes categories such as “sports,” “business,” “politics,” etc., with each parent category containing children nodes with yet more specific topics. A user can follow the topic tree down to as high level of granularity as needed. Another advantage of the hierarchy over the straight index is that it separates documents from homonyms, such as “salsa,” which is both the kind of food and the Latin ballroom dance. The clusters containing those documents would be markedly different [15].

Yahoo hierarchy is generated by hand, by people researching different topics and then placing the well-authored web pages in the appropriate classes. However, that approach is not scalable for all the hierarchies — for a lot of collections having a person do classification by hand is not only impractical, but also impossible. In those cases, having a machine-generated taxonomy would be a clear win. One possibility would be to have a human-generated hierarchy that is then automatically populated with documents. However, that requires the prior knowledge of topics in the document corpus. The most versatile solution is to have the taxonomy be both automatically generated and populated. In addition to being scalable, this approach has the advantage of being highly applicable to very diverse collection of documents or other data.

1.2 Goals

Neither clustering nor automatic taxonomy creation is new in Computer Science, and, specifically, in the area of Machine Learning. Rather than proposing a new solution, current work attempts to make a fair analysis of the most popular existing algorithms and investigate their performance when faced with various document data sets.

A number of the data clustering algorithms have been developed with data other than documents in mind and at times when collections numbering on the order of
thousands of data points were unusual, and, as such, it is important to understand how they work in the document clustering environment.

The goals of this thesis are twofold:

1. To do a quantitative analysis of generic model-based data clustering algorithms in the environment of document clustering.

2. To generate a number of heuristics that might be helpful in improving quality of the document clustering in particular type of conditions.

In order to achieve those goals, we design and implement a synthetic data generation engine that assists in creating, evaluating and improving different models and clustering methods.

In addition, this thesis strives to address issues of model evaluation and automatic taxonomy generation and will include a short survey of alternative approaches found in Machine Learning literature.

1.3 Background

Clustering methods range from those that are largely heuristic to more formal procedures based on statistical models. In many applications, actual methods and models are fine-tuned for the specific data, but it is important to strive towards a general approach that will work well in different circumstances and can be easily adapted to any domain.

There are two broad classes of clustering algorithms: so-called relocation algorithms and hierarchical clustering techniques.

**Relocation algorithms** are sometimes called “optimization partitioning” algorithms because they strive to optimize partitioning of data into clusters based on some particular distance measurement.

Optimization partitioning is typically done by the following steps:
1. Assigning data points to clusters (either randomly or based on appropriate heuristics).

2. Evaluating the resulting clustering according to the desired optimality criterion.

3. Relocating points in such a way that would maximize the optimality criterion.

4. Iterating steps 2-4 until convergence occurs or desired quality of clustering is achieved.

A most common optimality criterion is minimizing sums-of-squares distances; it is used, for example, in \textit{k-means} — the most common relocation clustering algorithm; but other criteria are not uncommon.

One of the disadvantages of relocation algorithms is that they require determining the target number of clusters before beginning of clustering\cite{3}. Since, commonly, this number is not known \textit{apriori}, it is usually picked from a well-educated guess, which might or might not be right for a specific data set in question. For clustering via mixture models, relocation techniques are usually based on the EM algorithm (see section 2.3.1).

\textbf{Hierarchical algorithms} partition the dataset in a hierarchical fashion, resulting in a tree structure where all data points that have a common parent are deemed to be similar by some appropriate metric\cite{14}. There are two types of hierarchical techniques: divisive and agglomerative\cite{13}.

1. Divisive algorithms function top-down by partitioning data into a small number of clusters and then iteratively partitioning those clusters into sub-clusters based on an optimality metric. A typical metric is minimizing sums-of-squares distances\cite{14}.

The disadvantage of divisive algorithms is that the order of possible divisions is proportional to square of the number of groups in the initial
partition [13], which can be computationally very expensive and hard to track.

2. Agglomerative algorithms function bottom-up by agglomerating initial clusters into larger parent clusters based on some optimality measure. While this approach does not suffer from the same rate of growth as the divisive approach, it requires pre-determining the number of leaf clusters in advance, which might be hard to do in view of unknown underlying structure of the data [18]. A typical starting clustering might place each data point in a cluster by itself. An example of agglomerative clustering algorithm is Scatter-Gather [22], [9], [10], [17].

In general, one disadvantage of the hierarchical algorithms is that it is not possible to get all kind of hierarchical structures. For example, while the underlying structure might be such that some nodes contain both a set of documents and a set of children nodes, the reality is that in most algorithms the set of documents would be separated into a separate leaf instead of being placed at the parent. That is, nodes are either leaves, and contain actual documents, or are parents to other nodes and contain no documents of their own.

In reality, neither divisive nor hierarchical clustering algorithms are used entirely without the other – often hierarchical algorithms are used to create a starting point for relocation algorithms. Alternatively, relocation algorithms can be used to create a hierarchical structure by iteratively clustering created clusters.

1.3.1 Model-based Clustering

In model-based clustering, it is assumed that the data is generated by a mixture of underlying probability distributions; each component of the mixture represents a different group or cluster.

Finite mixture models have traditionally been held as a good basis for cluster analysis. In this approach, the data is viewed as coming from a mixture of probability
distributions, such that each cluster has a different distribution that distinguishes it from other clusters. Methods of this type are used in various kinds of clustering – from work in identification of flaws in textiles from digitized images [5] to classification of data from astronomical observations [7].

One of the usually cited advantages of the model-based approach is that there are associated Bayesian criteria for evaluating the model. One such popular evaluation method is Bayesian Information Criterion (BIC) that has been successfully applied to the problem of determining the number of components in a model [11] and for deciding which among two or more proposed partitions most closely matches the data for a given model [21]. BIC will be further described in section 2.3.2. Another measurement criteria, which is heavily utilized in the present work, is Marginal Log-likelihood measurement [24]. Marginal log-likelihood is especially appropriate for clustering document data.
Chapter 2

Theoretical Basis

This chapter gives a brief overview of the theoretical basis for Bayesian data clustering, including the underlying model and our assumptions about document data as a particular clustering environment.

2.1 Supervised vs. Unsupervised Learning

Terms “supervised” and “unsupervised” learning generally refer to whether there is knowledge about inherent data structure prior to starting work with that data. Clustering, which is the main focus of this paper, is a classic example of unsupervised learning – when the process of clustering begins, typically, the model does not contain any details about a specific dataset. Instead, all the structure is inferred during the clustering itself[1].

In supervised learning, certain structure elements are known from the onset. For example in classification, clusters are known and the problem is one of placing a particular document in a cluster for which it is best suited. So it is supervised in a sense that there is something to compare the resulting clustering against and to make modification.

Of course, in reality, both supervised and unsupervised methods are often mixed to achieve the best results. For example, one might predetermine a cluster structure and then use machine learning mechanisms to populate it with documents. Such
A mix of existing and new modeling is useful when dealing with somewhat bounded text databases in order to speed up classification and allow efficient formations of well-clustered data. In fact, the underlying model for both type of machine learning is often the same. Distribution-based models are equally well adapted to supervised and unsupervised learning.

2.2 Model

Most of the work presented here was developed by Byron Dom and Shivakumar Vaithyanathan at IBM/Almaden Research Center (ARC). Please note that the real contribution of this thesis is in implementing a synthetic data generation engine and performing quantitative analysis of the performance of the document clustering algorithms under various adverse conditions, not in developing a theoretical background for it. Instead, we come up with a set of heuristics that might be useful in fine-tuning a particular clustering set.

The following excerpt is taken from a paper by Vaithyanathan and Dom[24].

Let $D$ be a data-set consisting of "patterns" $\{d_1, d_2, \ldots, d_n\}$. We will assume that the patterns are represented in some feature space $T$ that has a cardinality $M$. The particular unsupervised learning problem we address is that of clustering $D$ into groups (clusters) such that the likelihood of $D$ described by a certain probability model is maximized. In the most common formulation of the clustering problem this can be written as finding the $\hat{\Omega}$ and $\hat{\zeta}$ such that

$$ (\hat{\Omega}, \hat{\zeta}) = \arg \max_{\Omega, \zeta} P(D^T | \Omega, \zeta) $$

where:

\footnote{The term "patterns" here is used to refer to the objects to be clustered. For example, the patterns we cluster can be documents
$D^T$ indicates the representation of the data in feature space $T$. It will be implicitly assumed that $T$ consists of the complete set of features.

$\Omega$ is the structure of the model

$\zeta$ is the set of all parameter vectors

The model structure $\Omega$ consists of the number of clusters, the partitioning of the feature set - to be explained below - and (in the case of “hard” clustering) the assignment of patterns to clusters. Part of $\Omega$ such as the number of clusters, are usually fixed. If not, some regularization criterion is needed (more on this below.) The maximization of equation (2.1) constitutes a maximum-likelihood estimate of the model structure $\Omega$ and the parameter vector $\zeta$, which can be accomplished using several techniques - the most popular being the EM algorithm [12].

Assumption 1. The feature sets represented by $U$ and $N$ are (conditionally) independent

$$P(D^T|\Omega, \zeta) = P(D^N|\Omega, \zeta) P(D^U|\Omega, \zeta)$$

where:

$D^N$ indicates data represented in the noise feature space

$D^U$ indicates data represented in the useful feature space

Using assumption 1 we can rewrite equation (2.1) as below

$$(\hat{\Omega}, \hat{\zeta}) = \arg \max_{\Omega, \zeta} \left\{ P(D^N|\Omega, \zeta) P(D^U|\Omega, \zeta) \right\} \tag{2.2}$$

Assumption 2. Data \{\text{d}_1, \text{d}_2, \cdots, \text{d}_v\} is i.i.d (independent and identically distributed)

$$P(D^N|\Omega, \zeta) P(D^U|\Omega, \zeta) = \prod_{i=1}^{v} p(d^N_i | \zeta^N) p(d^U_i | \zeta^U_{k(i)})$$

where:
\( v \) is the number of patterns in \( D \)

\( \zeta_k^U \) is the parameter of vector for the cluster \( k \). It only applies to features in \( U \)

\( p(d_i^U) \) is the probability of pattern \( i \) given the parameter vector \( \zeta_k^U \) based on the assumption of some distribution - also called likelihood function

\( k(i) \) is the cluster to which pattern \( i \) is assigned

\( p_i^U \) is the probability of the pattern \( i \) given the parameter vector \( \zeta_N \)

Restricting our attention to hard clustering (i.e. every data-point is a member of one and only one cluster) and using assumption 2 we can write

\[
(\hat{\Omega}, \hat{\zeta}) = \arg \max \left\{ \prod_{i=1}^{v} p(d_i^N | \zeta_N) \cdot p(d_i^U | \xi_{k(i)}) \right\} 
\]

(2.3)

Note that while the explicit dependence on \( \Omega \) has been removed in this notation, it is implicit in \( k(i) \) and the partition of \( T \) into \( N \) and \( U \). Equation (2.3) can be rewritten as

\[
(\hat{\Omega}, \hat{\zeta}) = \arg \max \left\{ \prod_{i=1}^{v} p(d_i^N | \zeta_N) \cdot \prod_{k=1}^{K} \prod_{j \in D_k} p(d_j^U | \xi_{k}) \right\} 
\]

(2.4)

where: \( K \) denotes the number of clusters and the notation "\( j \in D_k \)" denotes that the document \( j \) is assigned to cluster \( k \).

Unfortunately, the objective function, represented in equation (2.4) is not regularized\(^3\) and attempts to optimize it directly may result in the set \( N \) becoming empty - resulting in overfitting. To overcome this problem we use the "marginal or integrated likelihood" \(^2\) whose negative logarithm is also called "stochastic complexity" \(^{23}\)

\(^3\)This lack of regularization essentially means that since there is no penalization for more complex models, resulting in the choice of the most complex model.

**Assumption 3.** All parameter vectors are independent.
\[ \pi(\zeta) = \pi(\zeta^N) \cdot \prod_{k=1}^{K} \pi(\zeta_k^U) \]

The marginal likelihood, using assumption 3, can be written as

\[ p(D^T|\Omega) = \int_{\mathcal{N}} \left[ \prod_{i=1}^{V} p(d_i^N|\zeta^N) \right] \pi(\zeta^N) d\zeta^N \times \prod_{k=1}^{K} \int_{\mathcal{U}} \left[ \prod_{i \in D_k} p(d_i^U|\zeta_k^U) \right] \pi(\zeta_k^U) d\zeta_k^U \]

where:

\[ \pi(\zeta^N) \] prior distribution over the noise-feature parameters

\[ \pi(\zeta^U) \] prior distribution over the useful-feature parameters

\[ \Xi^N, \Xi^U \] integral limits appropriate to the particular parameter spaces.

These will be omitted to simplify the notation.

Note that for a fixed number of clusters (K), as we increase the cardinality of the set U, the model complexity increases and similarly for a fixed U the model complexity increases with increasing K. The reader is referred to [4] for a lucid explanation of the use of marginal likelihoods for regularization.

\subsection{2.2.1 Foreseeable Shortcomings}

We do not presuppose that the model presented above correctly mirrors the underlying structure of any real-life document data. Instead, we hope that it gives a good basic structure which needs to be modified for a particular model in the process of document clustering.

Foreseeable shortcomings include the simplicity of some of the above assumptions. For example, it is fairly clear that in real life features are not all independent — “New York” would be two separate features: “New” and “York” and yet they are clearly interdependent — any time you see “York,” chances are, that it is preceded by “New.” At the same time, seeing “New” does not necessarily mean that “York” is to follow. Interdependence, and this kind of non-symmetric interdependence, is not something taken into account by a present model, but we research it later by generating syn-
thetic data sets that include inter-dependent terms and observing how the resulting clusterings are affected by this deviation from the underlying assumption.

Analogously, the model fails to mirror real-life text in a number of ways, but, perhaps, it is not as much of a shortcoming as a feature. After all, if we were to model exactly a particular dataset, we would end up overfitting to the data and our model would be unusable with any other dataset. By allowing certain simplifications, we preserve the generality of the model, encouraging its use in different situations.

2.3 Algorithms

2.3.1 EM

EM is the most popular of various relocation methods for clustering via mixture models. The EM algorithm is a general approach to maximum likelihood in the presence of missing data.

The EM algorithm iterates between an E-step in which values of cluster probabilities are computed from the data with the current parameter estimates, and an M-step in which the complete-data log-likelihood is maximized with respect to the parameters. Both E and M steps can be used in case of multivariate normal mixture models parameterized via the eigenvalue decomposition.

Under mild conditions, the iteration converges to a local maximum of the log-likelihood. The model and classification can be considered to be an accurate representation of the data if all of probability distributions are close to either 0 or 1 at a local maximum. Moreover, each observation has a measure of associated uncertainty in the associated classification [13].

The EM algorithm for clustering has a number of limitations. First, the asymptotic rate of convergence can be very slow. This does not appear to be a problem in practice for well-separated mixtures when started with reasonable values, but text does not always have well-separated mixtures and adequate starting values may be difficult to obtain. Second, the number of conditional probabilities associated with
each observation is equal to the number of components in the mixture, so that the EM algorithm for clustering may not be practical for models with very large numbers of components, and yet we use it in that context because we feel that it is the best available approach, and, through other fine-tuning, we can reduce running time and computational complexity to reasonable methods. Finally, EM breaks down when the covariance corresponding to one or more components becomes ill-conditioned (singular or nearly singular). In general, it cannot proceed if clusters contain only a few observations or if the observations they contain are very nearly collinear. If EM for a model having a certain number of components is applied to a mixture in which there are actually fewer groups, then it may fail due to ill-conditioning [13].

A number of variants of the EM algorithm for clustering presented above have been studied. The classification EM or CEM algorithm [6] converts the probability distributions from the E step to a discrete classification before performing the M-step. The standard k-means algorithm can be shown to be a version of the CEM algorithm corresponding to the uniform spherical Gaussian model[13].

2.3.2 BIC

One advantage of the mixture-model approach to clustering is that it allows the use of approximate Bayes factors to compare models. This is an approach that facilitates selection of not only the parameterization of the model and clustering method, but also the number of clusters. Fraley and Raftery [13] tout Bayes factors as a good systematic approach for determining both the intrinsic number of groups within a data set and the clustering method.

The Bayes factor is the posterior odds for one model against the other assuming neither is favored a priori. In their paper on model-based clustering, Banfield and Raftery [1] use heuristically derived approximation to twice the log Bayes factor called the 'AWE' to determine the number of clusters in hierarchical clustering based on the classification likelihood.

When EM is used to find the maximum mixture likelihood, another reliable approximation to twice the log Bayes factor called the Bayesian Information Criterion
or 'BIC' is applicable [16]. BIC is comparable to the Minimum-Description Length approach[24]. BIC is defined as follows:

\[
2 \log P(D^T | \Omega) + \text{const.} \approx 2 \arg \max_{\Omega, \zeta} P(D^T | \Omega, \zeta) - m_\zeta \log(n) \equiv BIC
\]  

(2.6)

where \( P(D^T | \Omega) \) is the log-likelihood of the data for the model \( \Omega \); \( \arg \max_{\Omega, \zeta} P(D^T | \Omega, \zeta) \) is the maximized mixture likelihood for the model and \( m_\zeta \) is the number of independent parameters to be estimated in the model. If each model is equally likely a priori, then \( P(D^T | \Omega) \) is proportional to the posterior probability that the data conform to the model \( \Omega \). Large values of BIC show strong evidence for the model. The BIC can be used to compare models with differing numbers of components or differing parameterizations. Although regularity conditions for BIC do not hold for mixture models, it has been used in that context with some success in Fraley and Raftery [13].

While we have not used BIC to calibrate results of clustering done in the present work, deferring instead to Mutual Information (MI) Index, and maximum log likelihood measurements because they naturally fit the environment; a number of other works discuss later rely on BIC as a "goodness" measurement tool [13], [19].
Chapter 3

Document Corpus

3.1 Golden Standard?

In general, Machine Learning proceeds by training a machine learning mechanism on a set of examples with a known “goodness” metric. For example, for image classification, one might have a set of images that are in a particular set and teach a program to recognize those images.

At first glance, the same would hold true for document clustering, but several problems arise:

1. The large number of documents in each corpus poses a problem. For training to be reasonable and comparable to actual working environment, the training dataset must have an order of thousands of documents. For those documents to be human-clustered would take hundreds of hours of work, and neither we nor do most of anyone else have resources for that.

2. Every different document corpus requires different degree of granularity in clustering. For example, a general news corpus would need breakdowns along fairly broad lines, such as “politics,” “sports,” etc., while, say, a database of medical records about a particular disease would need to be broken down to such detail as specific pathogens or medicines applied. Insofar as that is possible to do human-labeling of specific databases, it would need to be done by a specialist.
in that area who is also familiar with library science and general classification basics. Finding such a person or several people might be a complicated task, if such a person even exists.

3. Even if we were to find resources to have a human-clustered dataset, it is not clear what a “good” clustering is. There are documents that are, for example, clearly about sport or medicine, but where does one put a document about sport medicine? Library science is focused on solving problems like that, but even library scientists do not have absolute answers. Often, no two humans would agree on exact categorization of all documents in a data set. If there was an algorithmic standard for what a good clustering is, we would develop our algorithms to strive to achieve that standard, but no such standard exists.

4. Human-clustered databases might not have the underlying strict methods that can be inferred by machine-learning software. Instead, clustering might be done based on a set of heuristics rather than a specific model, which is not what we would want in a good learning example.

5. Situation is equally problematic for “flat” and “hierarchical” clusterings. In case of hierarchical clustering, the top few clusters might be undeniable, but the lower nodes might vary greatly depending on who you ask. Not only that, their very number is not fixed in stone and would be up for debate.

Despite those listed difficulties, there do exist human-labeled databases. One such classified hierarchy is the Yahoo web database [25]. Yahoo employs hundreds of people hand-clustering the web pages that their web crawler picks up on the web. While it is indisputably a useful resource, anecdotal evidence indicates that people are far from satisfied with the exact clustering rather, they are willing to use it because nothing better is available. To such end, we cannot use Yahoo database as a golden standard for machine learning experiments. ¹

¹While we have made attempts to use Yahoo hierarchy as our golden standard, after some review, we felt that it did not provide sufficiently clear basis to start from, and was not a good source for hard evidence.
In short, there is no golden standard for clustering against which we can compare machine-generated clustering. Is the situation then hopeless? We believe that it is not so. To facilitate the learning process, we use synthetically generated data that comes with the underlying structure and can be made to mirror different textual models. In addition to that, we work with the TREC database that is accepted in the industry as a good hand-clustered standard.

### 3.2 Synthetic Data

Generally, in clustering data, we try to model what we believe an underlying data structure to be. To that end, what could be better than having data that actually corresponds to that model, to see how our algorithm would perform in these, almost ideal, conditions?

In our experiments we rely heavily on what we call “synthetic data” — data generated by the data-generation software (see chapter 4). In our synthetic datasets, we find a wealth of available variations, corresponding to various possibilities within and outside the bounds of our models.

The data is generated according to desired parameters. As such, we have on order of hundreds of different datasets, each containing thousands of documents. The documents themselves are not actually composed of words, but of “synthetic words” — a.k.a. numbers that represent different words. Since, during the normal course of processing, even the regular text dataset gets converted to a mathematical representation for the purpose of clustering, it makes no difference whether words are words or numbers, as long as each number represents a unique word. We not only have all required statistics for synthetic datasets, but even individual documents themselves, just like we would with a regular dataset.

The documents themselves range in length and complexity. The model, cluster probabilities, feature probability distributions, length of the documents, number of documents, number of features, number of “noise” and “useful” features, and many more other options (see chapter 4) can be adjusted to create a variety of datasets that
could not be acquired from real text databases.

These synthetic datasets are all the more great in that they provide the missing structured example for learning — by varying parameters, we can ascertain that our clustering program performs well in various adverse conditions.

3.3 TREC

In addition to synthetic data, we test our algorithms on a real-world dataset. The dataset selected is a human-labeled document collection held to be a fairly good example of document clustering: a corpus of AP Reuters Newswire articles form the TREC-6 collection.

TREC corpus includes 8235 articles from the routing TREC competition track. The articles are spread out over 25 human-labeled clusters which we use as a target result in our experiment. While the TREC dataset includes documents that are assigned to multiple clusters, we ignore multiple assignments and assume that each document belongs to at most one cluster.

While the result section (ch. 6) does not include hard qualitative numbers from our experiments with TREC collection, a number of anecdotal results are discussed and analyzed.

3.4 Other Document Data Sets

During the course of our research for this work, we have considered using various other human-labeled databases, including Yahoo web database, a collection of Lotus Notes pages, and pre-labeled collections from various sources. While we have not used those datasets extensively, it might be a good direction for future work to test our ideas and algorithms on a number of real-world text databases, from general to very specific ones.
Chapter 4

Data Generation

Synthetic data generation is an excellent tool for training, testing and understanding clustering algorithms. By creating artificial text databases, we can vary parameters one by one, holding others constant, or in groups, and see how the resulting clusterings change — whether they are adversely affected by data effects or not. Such experiments give us a fair prediction of how a particular clustering heuristic might stand up in real life, where data will vary along different parameters.

In addition to training and testing, working with synthetic data is important in determining how a particular algorithms should be modified to better fit the particulars of some data set — that is, we use it to assist us in creating a set of guidelines for better clustering. For example, we create excessively large datasets, data sets with numbers of features ranging from single digits to thousands, and then check the resulting clusterings to determine how these conditions affect the outcome. By further varying the conditions and parameters of our clustering program, we can come up with a set of heuristics for dealing with adverse affects of real data that does not ideally correspond to our models.

4.1 Model

The basis of our data generation program is the model for the underlying real-text data structure. We take the basic model to be the same as discussed in chapter 2. In
the process of experimentation, we vary different parameters of the model (at times essentially converting it to other models) and note the results.

As a first pass, we take the model as is, with no modification, generate the data and check our clustering program against this synthetic data. You will note that, besides its research uses, a data generation program is an excellent debugging tool: if, when using the same model for both generating and clustering data, we get very poor results, then we know that there is a problem in our implementation that needs to be found and fixed. Otherwise, the results are almost impossible to check by hand — test results are not very human-readable or checkable.

Among the features of data generation software are creation of arbitrary data hierarchies, such that the probability distributions for both noise and useful features are kept inline with the hierarchical structure, creation of data from a given set of probability distributions, ability to simulate inter-dependent features (which is a major change from our Assumption 2 — that all features are independent) and ability to keep constant all factors except for those that are being specifically tested in a particular test.

In addition to data generation, our software includes models for collection of statistics and visual representation of results and such theoretical tools as modules for creating confusion matrices between the generated data and clustered data, computation of KL-Distances in generated clusterings, computation of marginal log-likelihood for both synthetic and training clusterings, cross-checking and recording probability distribution data and others.

### 4.2 Implementation

This section contains a description of variants of the data generation program we worked with in the course of this work. As stated earlier, the data generation program started out as a debugging tool and grew into an essential research tool that we feel is important to present in order to demonstrate its capabilities.
4.2.1 History

The initial data generation version was written in 1999 by Byron Dom and Shivakumar Vaithyanathan of IBM/Almaden Center and was used for testing their document clustering software. That first implementation contained only the most basic model and gave as output files needed to cross-check the clustering. The original version was written in Perl and did not use computationally efficient methods, so the running time was a major problem – at times it took hours to generate a comparatively modest dataset.

As such, it was decided that the data generation program had to be rewritten in C, taking advantage of more computationally scalable algorithms and data structures. I completed the conversion and took upon myself the work in further modifying the program to contain the additional features we required.

4.2.2 Design

The data generation program is based on a modular design that allows for easy addition and modification of modules. The following major functions are present:

Hierarchy creation functions take care of creating the desired hierarchy based on appropriate input files (or generating one on the fly). The hierarchy can be an arbitrarily-nested tree, which allows for almost unlimited flexibility in terms of modeling real-data taxonomies (the one notable exception is the current lack of multiple inheritance, which will be added in the future, simulating the documents which fall into multiple clusters).

Cluster probabilities creation functions generate apriori cluster probabilities, or read them in from the given file and create a cluster structure that will match those apriori probabilities.

Huffman tree data structures form a basis for data structures that hold information about our model — clusters, their apriori probabilities, and, in turn, probability distributions for different features within clusters. From our analysis,
Huffman trees are the most efficient data structure for searching and navigating the probabilistic network — following its basic principles, the most likely clusters end up closest to the root, with those that are least likely having a longer access time.

**Probability distribution generation** functions generate appropriate probability distributions for each feature in each clusters. It is perhaps here that lies the real complexity. We simulate continuous probability distributions through discrete coin tosses, probability of which correspond to our probability distributions. Additional care is needed when generating noise features for hierarchical structures. By definition, there should be a relation between probability distribution for some features for sibling clusters. We feel that this relationship is inherent in our definition of noise features — noise features at a particular level do not help us distinguish between different clusters, therefore probability distributions for noise features of sibling clusters should be the same.

A question can be asked as to whether it is better to generate probability distributions in top-down or bottom-up manner. We have tested both approaches and found inherent complexities in each, but we feel that bottom-up approach better reflects what we believe is the underlying structure of the real text to be, and therefore we have decided to go with that as our final version. Generation proceeds by first creating feature probability distributions for the leaves and then iteratively accumulating them for the parent nodes.

**Data generation** functions generate actual “documents” that serve as our dataset. The generation is done based on previously-created probability distribution by following the “bag of words” model where a particular word is selected by a coin toss with appropriate probabilities.

In addition to purely generative modules, there are also testing and data-checking modules and statistical functions:

**Confusion Matrix** functions create a confusion matrix of entries from a generated
clustering and clustering created by the clustering software that is being tested.

**KL-Distance** functions compute Kullback-Leibler (KL) Distances between clusters, that allow us to tell from a glance whether two clusters are close to each other, compared to distances to other clusters.

**Mutual Information Index** functions compute MI Index between the synthetic clustering and trained clustering.

**Marginal Log-likelihood** functions compute marginal log-likelihood probability for any given clustering.

All of these modules, taken together, create a powerful suit of software for generating synthetic data and testing and correcting clustering outputs.

### 4.2.3 Output and Navigation

Just as there are several input methods to the data generation program, there are several output modes. Most notable ones are the text and HTML formats. Text format is good for batch processing (there is a set of Perl scripts written designed to work with various output formats), while HTML is good for human viewing and analyzing.

For example, if one ran data generation software, EM clustering software, and output post-processing tools, all for 5 clusters, but for different number of combinations of features and total number of documents in the database, one would get back all relevant files for those batch runs, including properly formatted HTML pages. Main output is printed to a title page in a table that includes a lot of important information at a glance, as follows:

As you can see, the table includes links to more specific information about each run, such as KL-Distances, confusion matrices, cluster priors, and other relevant information. We feel that this User Interface provides both the advantages of a general view and allows the user to descend down to the level of a specific experiment following HTML links.
Run ID: 10
5 Clusters

<table>
<thead>
<tr>
<th>Run ID</th>
<th># of synthetic clusters</th>
<th># of features</th>
<th># of documents</th>
<th>KL Distances for synthetic clusters</th>
<th>Marginal likelihood for synthetic clustering</th>
<th># of em-generated clusters</th>
<th>Best ML found (by EM)</th>
<th>Confusion matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>25</td>
<td>100</td>
<td>KL Distances</td>
<td>-1926.466</td>
<td>5</td>
<td>-1654.809</td>
<td>Confusion Matrix</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>25</td>
<td>100</td>
<td>KL Distances</td>
<td>-18957.83</td>
<td>5</td>
<td>-14678.07</td>
<td>Confusion Matrix</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>25</td>
<td>100</td>
<td>KL Distances</td>
<td>-178033.1</td>
<td>5</td>
<td>-140941.2</td>
<td>Confusion Matrix</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>50</td>
<td>100</td>
<td>KL Distances</td>
<td>-1888.322</td>
<td>4</td>
<td>-1755.868</td>
<td>Confusion Matrix</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>50</td>
<td>100</td>
<td>KL Distances</td>
<td>-18093.38</td>
<td>4</td>
<td>-15036.39</td>
<td>Confusion Matrix</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>50</td>
<td>100</td>
<td>KL Distances</td>
<td>-176725.7</td>
<td>4</td>
<td>-143740.9</td>
<td>Confusion Matrix</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>100</td>
<td>100</td>
<td>KL Distances</td>
<td>-2112.572</td>
<td>5</td>
<td>-1883.724</td>
<td>Confusion Matrix</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>100</td>
<td>100</td>
<td>KL Distances</td>
<td>-18292.11</td>
<td>5</td>
<td>-12972.24</td>
<td>Confusion Matrix</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>100</td>
<td>10000</td>
<td>KL Distances</td>
<td>-178356.1</td>
<td>4</td>
<td>-123529.3</td>
<td>Confusion Matrix</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>1000</td>
<td>100</td>
<td>KL Distances</td>
<td>-3281.504</td>
<td>3</td>
<td>-3259.993</td>
<td>Confusion Matrix</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>1000</td>
<td>100</td>
<td>KL Distances</td>
<td>-20198.25</td>
<td>5</td>
<td>-17184.8</td>
<td>Confusion Matrix</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>1000</td>
<td>10000</td>
<td>KL Distances</td>
<td>-166044.8</td>
<td>5</td>
<td>-105899</td>
<td>Confusion Matrix</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>10000</td>
<td>100</td>
<td>KL Distances</td>
<td>1</td>
<td>2</td>
<td>-4908.612</td>
<td>Confusion Matrix</td>
</tr>
</tbody>
</table>
In the future, it is possible that additional output or User Interface formats will be added, following the need for them.

### 4.2.4 Running Analysis

Compared to the initial version, the data generation program has come a long way in terms of speed and usability. While it is hard to give exact numbers on its speed, we merely note that for most experiments the processing time is almost negligible (running on AMD K-6 600MHz processor with 96Mb of RAM). The largest batch process job took around 20 minutes to complete and generated what would have been gigabytes of data if it was expressed as a real-life database.

### 4.2.5 Platforms

The data generation and testing software currently runs on AIX (IBM Unix) and Linux platforms. However, all of it is programmed in ANSI C and does not use any platform-specific features, so porting to any other Operating System should not pause a problem and should be a matter of several hours.

### 4.2.6 Data Formats

The input parameters for the data generation software can be assigned through any number of ways, including a configuration file, command-line options and preprocessing commands within the program itself. Due to a large number of options and possibilities, we felt it would be optimal to provide different data formats and options to better suit the needs of a particular experiment. Configuration files are very human-readable (and maintainable), but command line options are more efficient in batch processing modes.

In addition to the actual programs, there are a number of scripts to facilitate running of the data generation software, both in single-job and batch job modes. Specifics about data formats can be found in Appendix A — config.h header file.
4.2.7 Directions for Future Work

While the data generation software is a fairly powerful tool as it is, there are many more additional modifications in the work. We plan to further extend dependent-features capability, to allow us to better model real-life data. To that effect, one possible addition is a data analysis tool that generates statistics from real data and then generates synthetic data based on that statistics. By comparing results of clusterings from the real data and synthetic data, we can further evaluate how close our models are to reality.

4.3 Discussion

There were a number of unexpected results and side-effects in developing and testing the data generation software. For example, thinking of a better set of data structures led us to better understand the underlying model. Likewise, implementing hierarchical data generation made us double-check our hierarchical model and to determine whether our proposed model was likely or possible in real world. After all, if the model was so complicated that we would not be able to implement it, it would give us an indication that it is unlikely to be the real structure behind any collection of documents.

One surprising difficulty arose in picking probability distribution for different features within each cluster. Our initial approach was to set the distribution for one feature, proceed to the next one, and so on, until probabilities for all features were determined, however, that led to features being improperly separated and severely stigmatized clustering results. After several different versions of probability generator, we arrived at the current version, which adds another degree of randomness in picking which probability distribution to determine next. This model seems to model better what happens in the real world with real data than our previous attempts.

Therefore, implementing the data generation software was a rewarding experience in itself, in that it gave me a much clearer understanding of our theoretical proposal and its real life ramifications, in addition to providing an excellent tool for testing,
training and analyzing our result. Looking back, this experience has been invaluable.
Chapter 5

Results

5.1 Experimental Set-up

In our experiments, we ran a large number of batch jobs, varying all possible parameters, both separately and in groups, to determine how they affect the resulting clustering. Presented below is the walk-through of our analysis on a few cases, followed by a summary of overall results, along with advice on how to fine-tune a particular model to better fit the dataset.

5.2 Expectations

In general we expected the following trends:

- Total number of clusters: as the number of clusters grows, the resulting clustering quality is reduced.

- Total number of features: as the total number of features grows, the resulting clustering quality is improved.

- Total number of documents: as the number of documents grows, the resulting clustering quality is improved.
- Minimum useful features at a node: as the number of minimum useful features at a node is reduced, the clustering quality goes down.

- Maximum document length: as the documents length grows, the clustering quality goes up.

The next sections look back on these predictions and explain the actual results.

5.3 Quantitative Results

To illustrate our experimental process, we will take a sample case of a batch testing done with the number of clusters being held constant at 5 and varying other various parameters.

5.3.1 5 Clusters, 25 features and variable number of documents

The overall results for 5 clusters, 25 features are as follows:

<table>
<thead>
<tr>
<th># of synthetic clusters</th>
<th># of features</th>
<th># of documents</th>
<th>ml for synthetic clustering</th>
<th># of EM-generated clusters</th>
<th>ml for EM clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>25</td>
<td>100</td>
<td>-1926.5</td>
<td>5</td>
<td>-1654.8</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>1000</td>
<td>-18957.8</td>
<td>5</td>
<td>-14678.0</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>10000</td>
<td>-178033.1</td>
<td>5</td>
<td>-140941.2</td>
</tr>
</tbody>
</table>

Table 5.1: Marginal likelihoods for synthetic and em-generated clusters for 25 feature documents

Through navigation tools, we can take a look at results for a particular run in more detail. Confusion matrices are very useful in getting immediate intuitive idea for what is going on in quantitative results. In addition to giving a breakdown of intersections between clusters, they give a good visual idea of where there is confusion between clusters, and whether it was one particular cluster that caused problems or are the results overall poor.
To investigate further, we descend to an example with 5 clusters, 25 features and 10000 documents; we get the confusion matrix in table 5.2.

<table>
<thead>
<tr>
<th>#</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1801</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2796</td>
<td>92</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>29</td>
<td>3168</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>703</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1411</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.2: Confusion Matrix for 25 feature and 10000 documents

As you can see, for most clusters there were no errors in locating the proper documents. In fact, for all but two clusters every single document was put in its proper place.

Even for clusters 2 and 3, between which there is some confusion, a number of improperly clustered documents is relatively small — order of thousands properly placed vs. order of tens that are not. But, in any case, we can investigate this confusion even further.

Next step in our investigation is the KL Distances table for original synthetic clusters. From this table, we can judge the complexity of differentiating between two clusters. If the distance is relatively large, then it should be easy to separate the documents, whereas if the distance is small, then some confusion may be warranted. See table 5.3.

It is obvious from this table that clusters 2 and 3 are a good deal closer together than other clusters are to each other, so it is understandable that there are misclassifications between the two. In fact, if the KL Distance was really small, it would suggest that the two clusters might need to be merged into one to provide for a better underlying structure. In this case, distance of 2.6 is noticeably smaller than average, but is not so much so as to make us question the validity of making them two separate clusters.
Total KL Distance: 138.3
Average KL Distance: 6.9

<table>
<thead>
<tr>
<th>From cluster #</th>
<th>To cluster #</th>
<th>KL Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>14.4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>15.0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>15.9</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>9.6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2.6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>16.8</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>14.5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>16.9</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>15.3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>17.3</td>
</tr>
</tbody>
</table>

Table 5.3: KL Distances between synthetic clusters for 25 feature and 10000 documents

5.3.2 5 Clusters, 50 features and variable number of documents

Once again, we first look at the overall results. The interesting thing in this table is that EM never finds the synthetic number of clusters. Instead, it chooses 4 clusters as the best structure.

<table>
<thead>
<tr>
<th># of synthetic clusters</th>
<th># of features</th>
<th># of documents</th>
<th>ml for synthetic clustering</th>
<th># of EM-generated clusters</th>
<th>ml for EM clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>50</td>
<td>100</td>
<td>-1888.3</td>
<td>4</td>
<td>-1755.9</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>1000</td>
<td>-18093.4</td>
<td>4</td>
<td>-15036.4</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>10000</td>
<td>-176725.7</td>
<td>4</td>
<td>-143740.9</td>
</tr>
</tbody>
</table>

Table 5.4: Marginal likelihoods for synthetic and em-generated clusters for 50 feature documents

So should we deem this experiment to be a failure? Not quite. The interesting column to look at is the marginal log-likelihood one. As you can see, the resulting ml-likelihood is actually better for the EM-generated clustering than for the synthetic clustering. And, indeed, in the way our data is generated, it is possible that the
structure that generated some dataset is in fact not the best fitting formation for that very dataset.

To investigate further, we descend to an example with 5 clusters, 50 features and 10000 documents; we get the confusion matrix in table 5.5.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>745</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3493</td>
<td>0</td>
<td>650</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>621</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4491</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.5: Confusion Matrix for 50 features and 10000 documents

As we can see from the confusion matrix, clusters 2 and 4 are merged together into one. If we look at the KL-Distances table, we discover that this is so because the distance between those two clusters is significantly smaller than the distances between other clusters. Therefore, it might be more optimal to unite them into one cluster.

<table>
<thead>
<tr>
<th>From cluster #</th>
<th>To cluster #</th>
<th>KL Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>16.3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>11.7</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>15.0</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>14.2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>16.2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3.2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>18.2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>15.1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>16.2</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>16.9</td>
</tr>
</tbody>
</table>

Table 5.6: KL Distances between synthetic clusters for 50 features and 10000 documents

Overall, it is hard to judge what is better — the original synthetic clustering or this EM-generated one. In this case, we feel that EM did a good job of separating
the clusters, even if it did not come up with the right answer.

5.3.3 5 Clusters, 1000 features and variable number of documents

We proceed now to the overall table for 5 clusters, 1000 features and variable number of documents:

<table>
<thead>
<tr>
<th># of synthetic clusters</th>
<th># of features</th>
<th># of documents</th>
<th>ml for synthetic clustering</th>
<th># of EM-generated clusters</th>
<th>ml for EM clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1000</td>
<td>100</td>
<td>-3281.5</td>
<td>3</td>
<td>-3260.0</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>1000</td>
<td>-20198.3</td>
<td>5</td>
<td>-17184.8</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>10000</td>
<td>-166044.8</td>
<td>5</td>
<td>-105899.0</td>
</tr>
</tbody>
</table>

Table 5.7: Marginal likelihoods for synthetic and em-generated clusters for 1000 feature documents

First let us look at the 100 document version of the 5 clusters, 1000 features run; we get the confusion matrix in table 5.8.

Table 5.8: Confusion Matrix for 1000 features and 100 documents

<table>
<thead>
<tr>
<th>#</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

As we can see, there are only three clusters as found by EM. To understand this better, we need to look at the KL-Distance table 5.9:

But even KL-Distances do not explain why clusters 1 and 4 and 2 and 5 were merged together. The explanation lies in the total number of documents being very small compared to the selected number of features. The separation is not enough to differentiate between clusters and so EM chooses a 3-cluster configuration.

Finally we come to the 10000 document version of the 5 clusters, 1000 features run; we get the confusion matrix in table 5.10.
Total KL Distance: 143.0  
Average KL Distance: 7.2

<table>
<thead>
<tr>
<th>From cluster #</th>
<th>To cluster #</th>
<th>KL Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1.62</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2.04</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1.17</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1.03</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1.99</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1.12</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0.99</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1.54</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1.40</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>0.53</td>
</tr>
</tbody>
</table>

Table 5.9: KL Distances between synthetic clusters for 1000 features and 100 documents

<table>
<thead>
<tr>
<th>#</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3509</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2070</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>927</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1019</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2474</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.10: Confusion Matrix for 1000 features and 10000 documents
Total KL Distance: 143.0  
Average KL Distance: 7.2  

<table>
<thead>
<tr>
<th>From cluster #</th>
<th>To cluster #</th>
<th>KL Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>16.9</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>18.7</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>16.4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>15.3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>18.3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>16.1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>15.0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>12.9</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>16.7</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>14.5</td>
</tr>
</tbody>
</table>

Table 5.11: KL Distances between synthetic clusters for 1000 features and 10000 documents

Unlike in the previous case, EM has picked the right number of clusters: 5. The explanation is that the number of documents is now appropriate for the job, and, although the rest of the parameters remain the same, additional documents provide better context to make a proper clustering.

Analogously to the above analysis, we conducted experiments with varying different parameters and sets of parameters. Although the illustrative example had only five clusters, most of the examples compiled into overall results were far more realistic in their sizes.

### 5.4 Inferred Heuristics

As predicted, certain trends emerged from the multitude of performed experiments; interestingly enough, they were not always as predicted, but, upon further examination, they fit well within the general context of underlying model structure. While the examples above use five clusters for simplicity, actual trends are more apparent from larger clusterings and larger data sets, which, unfortunately, are too large to quote here, but the conclusions should still be sufficient.
For each varied parameter, there was a slightly different kind of response, as explained below:

- Total Number of Clusters: we varied the number of clusters in synthetic data to see how it would affect results of EM clustering. Overall, the effects are not apparent without taking into account the number of features. If the number of clusters is much smaller than the number of features and features are well separated, then the resulting clustering is close to perfect or even perfect (re-capturing 100% of synthetic structure). If, however, the features are not well separated, the resulting clustering is far off from the proposed one. However, just as in the example with five clusters, the resulting clustering might actually be better than the synthetic one, demonstrating, once again, that creating data from a particular structure does not mean that it is an optimal structure for that data.

If the number of features was close to the number of clusters, results varied greatly based on the probability distributions among features. If the number of clusters was actually smaller than the number of features, EM could not come up with the desired number of clusters. While it was possible to modify EM to work with situations where there are more features than clusters, we felt the effort to be unnecessary, because in real life such situation will probably not arise — it is hard to imagine a document data set where the total number of words in all documents is less than the desired number of clusters.

- Total Number of Features: we varied the number of features, keeping the number of clusters and number of documents constant. Once again, the results needed to be viewed in context with other variables. While we expected the number of features in reality to be much greater than the number of clusters and even than the total number of documents, we did try situations with relatively small set of features — starting as low as 10. Depending on the way we generated our probability distributions for each cluster, we came up with results that either matched 100% our synthetic cluster structure or were better (as
evaluated by marginal likelihood measurement) than the synthetic structure.

- Total Number of Documents: as far as varying the total number of documents, we were expecting to find that a smaller dataset would give us worse results than a large dataset in all cases. It was surprising to discover that the reality is not always like that. Instead, the trend is more like a Gaussian curve — as the number of documents increases, the quality of clustering increases, but only up to a point. If the dataset is excessively large, the overall quality is actually worse than it would have been with fewer documents.

The cause for this behavior lies in the fact that not all clusters have the same apriori probability, which means that they end up with different numbers of documents in them. As the size of the dataset increases, this difference also increases, so much so that some clusters become several times larger in size than others, overshadowing the smaller ones and incorporating them, instead of being separated from them. For example, if one cluster has 50000 documents, and another — 5000, they may end up being merged into one, even if they would have clearly been two different groups if the sizes were more comparable.

In addition to observing this behavior, we came up with several ways of offsetting it: instead of using a whole dataset, the user should consider doing a random sample and clustering that. Not only would it provide a fair representation of the structure, it would also considerably speed-up overall clustering process. Once the initial structure is established, documents can be classified into those groups rather than moving all of them around at once. It is exactly this solution that is proposed in a couple of works surveyed in chapter six. At times, even sampling might not be a good way of splitting different clusters — what if in the process of sampling the sizes do not become comparable? One possibility is to re-introduce the non-predominant points as a means to counteract the size effect. Since the probability distributions for points will remain the same, introducing additional data points should not affect the overall clustering structure (up to a point).
- Minimum Useful Features at a Node: we varied the number of minimum useful features at a node from being equal to the total number of features (signifying no noise features) to almost zero (if it was zero, no clustering would be possible because, by definition, all noise features have the same probability distribution, and, therefore, all documents would end up in the same cluster). As expected, when the number of useful features was large, clustering was a bit better, because it allowed for greater separation in feature probability distributions. If the number of useful features was small, compared to the total number of features, then it was a toss-up, depending on how separated those features were. In effect, that was a situation akin to the one with a small number of features.

- Maximum Document Length: varying the document length had the effect of going from snippets of documents on one end to full-length "novels" on another. As predicted, clustering is tough with snippets of documents, but, with a sufficiently optimized model and with a large number of documents, the situation is far from hopeless, and we got correct results far more often than predicted. As far as the huge documents go, we did not detect much of an improvement over having a large database where all documents are of reasonable sizes. Once again, that validates our model, in which it is the probability distribution for each feature in each cluster that makes a difference, not its probability distribution for a particular document. That is, all documents in one cluster come from the same bag of words.

- Cluster Apriori Probabilities: varying cluster apriori probabilities has the effect of varying the total number of documents that end up in it in synthetic clustering (because the posterior probabilities are always very close to the apriori ones). As discussed earlier, unequal distribution of documents can lead to mis-clustering even if features are well separated, although the difference has to be large (at least an order of magnitude) before this effect becomes apparent.

- Inter-dependent Features: creating inter-dependent features violates our assumption about features being iid's, so, on the surface it seems that it would
significantly decrease the quality of the resulting clustering. That turns out to be not the case. If all the features have complex inter-dependencies, then yes, the effect is noticeable, but otherwise the model is flexible enough to absorb this violation and EM comes up with a fairly good clustering despite of this adversity.

- Hierarchical Structure: the main question was this: “how does flat clustering mechanism deal with data that inherently comes from a hierarchical structure?” And the answer is: it depends. If we are only attempting to recapture the leaf clustering, then the results are comparable – that is, if the hierarchical structure has ten leaves, and we attempt to do a flat clustering, we will get a structure very much like the leaves of the hierarchy. But if we try to recapture the whole hierarchy, the problems arise. One way to do that would be through a set of subdivisions, but each subdivision would pick an optimal number of sub-clusters, which might not correspond the total number of leaves in the original hierarchy. Overall, we are of the opinion that to recapture original hierarchical structure, special tools designed for it must be used – simply subdividing is not good enough.

The above list of variables provides a solid “ingredient set” for things to vary to fit a particular clustering methodology to the desired data set. If used in practice, they can assist researchers and users in creating better cluster and hierarchy structures.
Chapter 6

Conclusion

6.1 Survey of Alternative Approaches

The problem of data clustering is an important, but challenging one. There is a significant body of work on it, going as far back as the fifties and culminating in present times, when information explosion and accumulation of web data pose additional challenges to data mining and exploration[15].

While it is impossible to discuss the majority of work in this area, or even all of the milestones, I will attempt to give a brief overview of several recent papers in the field, restricting my attention to those that use Bayesian models for clustering, but not necessarily only document clustering.

6.1.1 Model-Based Cluster Analysis

Fraley and Raftery discuss model-based clustering in their technical report: “How Many Clusters? Which Clustering Method? Answers Via Model-Based Cluster Analysis” [13]. Just as in more recent work, they consider the problem of determining both the inherent number of clusters in the data and their composition. They use Gaussian components to obtain models with varying geometric properties \(^1\) through

---

\(^1\)They operate in \(N\) dimensional space, where \(N\) is not the total number of features, but a selected limited subset
Gaussian components with different parameterizations and cross-cluster constraints.

Noise is modeled by adding a Poisson component. Actual partitions are determined by the EM algorithm for maximum likelihood. Hierarchical Agglomerative clustering is used as a starting point for EM [13]. To compare models, an approximation to the Bayes factor based on the Bayesian Information Criterion (BIC) is used, which allows comparison of more than two models at the same time, and removes the restriction that the models compared be nested.

While this approach is shown to give better performance than standard procedures such as nearest-neighbor and k-means clustering, which often fail to identify groups that are either overlapping or of varying sizes and shapes, it is also much more computationally expensive and needs to be fine-tuned for the kind of data involved.

Fraley and Raftery consider various diverse domains, including medical data — diabetes analysis — and even minefield detection in the presence of noise [13]. However, they do not discuss document clustering, so it is hard to make direct comparisons between their and present approach due to differences in data scales. We deal with much larger database, where models are not obvious, and even most sophisticated modeling is only a rough approximation of the actual structure.

6.1.2 Clustering Massive Datasets

The problem of clustering massive datasets is addressed by Maitra in his paper on that topic [20]. In particular, the paper finds current clustering means inadequate for dealing with large datasets and attempts to devise a new, less computationally taxing procedure for those purposes. To that end, the paper develops a multi-stage sequential algorithm that works under Gaussian assumptions.

Overall, clustering in Maitra proceeds in stages. After clustering an initial chosen sample, observations that can be reasonably classified in the identified groups are filtered out using a likelihood ratio test. The remaining points are again sampled and clustered. This procedure is iterated until all cases have either been clustered or classified [20]. In a final post-processing step all empty groups are eliminated and checks are made to assure that each data point belongs to a particular group.
The author notes that, during the actual testing stages, no more than four stages of sampling were needed in any of these cases, suggesting that the procedure is practical to implement [20]. However, it is unclear how much advantage that is over a multi-staged EM iteration, where the number of clusters is predetermined for each iteration.

In addition, the author does not actually define what sort of dataset can be considered a massive dataset. The experimental results presented in the paper are based on the data set of two hundred thousand data points, which is comparable to what is used in the present work.

The advantages of this new algorithm lie in its ability to detect both small and large groups, thus avoiding a problem of smaller groups being swamped by larger ones due to sheer number of points. In addition, it raises a point of computational complexity and how much effort should go into decreasing it in real life situations.

6.1.3 BIRCH

Tian Zhang discusses data clustering for very large datasets plus applications in his doctoral thesis [26]. The proposed solution to the problem of large-scaled data mining is a structure called a CF-tree serving as a data distribution summarization. A system called BIRCH – Balanced Iterative Reducing and Clustering using Hierarchies is used to demonstrate performance of this new approach.

BIRCH is an interesting approach to a problem, and, according to the evidence presented, it does an adequate job of handling very large datasets. But it is doubtful that one summarization can fill the need in all situations.

6.2 Conclusion

This thesis presented an overview of existing document clustering techniques and did a quantitative analysis of their performance in view of adverse factors. In the analysis, a number of heuristics are presented to help cope with data clustering in real-life situations, both through adjusting the underlying models and through data manipulations that makes clustering more scalable, accurate, and less computationally intensive.
In addition, we feel that the data generation software is an invaluable tool in analyzing data clustering because it allows modeling of different datasets and access to parameters which cannot be varied (or, sometimes, even calculated) in the real data set.

6.3 Direction for Future Work

The search for better clustering techniques is by no means over. On the contrary, as the data overflow continues, more and more solutions are needed. Even techniques presented in this work need to be investigated further, not only to find possible modifications, but to better understand their performance in real-life conditions.

With respect to the data generation software, several modifications are needed to make it more flexible and add additional assessment tools that will aid us in further investigating results of synthetic clustering.

One important field of study is in hierarchical clustering — models need to be developed that take into account hierarchical structure of data, rather than relying on iterative operations to arrive at the tree structure.
Appendix A

Appendix A

#ifndef CONFIG_H
#define CONFIG_H

#include "tree.h"

#define TOTAL_NUMBER_OF_CLUSTERS 31
#define TOTAL_NUMBER_OF_FEATURES 10000
#define TOTAL_NUMBER_OF_DOCS 5000
#define MIN_USEFUL_FEATURES_AT_NODE 500
#define MAX_DOC_LENGTH 1000
#define VARIABLE_DOC_LENGTH 1
#define RELATIVE_NAMES 1
#define CLUST_PROB_OPTIONS 0

/* Modified for dependent features (LUCY) */
#define MEAN_ID 10
#define STDDEV_ID 3

typedef struct Config_info {
    int clust_probs_options;
}
int variable_doc_length;
int max_doc_length;
int total_number_of_docs;
int total_number_of_features;
int total_number_of_clusters;
int min_useful_features_at_node;
int total_number_of_generating_clusters;

char *dirname;
int relative_names;
char *hierarchy;
char *doc_data;
char *doc_stats;
char *doc_counts;
char *correct_result;

double *cluster_probabilities;
char *clust_probs_file;
int noise_list; /* QUESTION: what is this for? (LUCY)*/

/* modified for ID (LUCY) */
int mean_id;
int stddev_id;
int id_option;

/* this is to make sure
   that thetas are distinct
   for each cluster */
int global_i_start;
/* added to be able to keep thetas constant
   between runs */
int preserve_thetas;
char *given_thetas_file;
FILE *fp_given_thetas;
FILE *fp_thetas;

} config_info;

void create_default_config_file();
/* generates a file called default_config that contains the default
   configurations for all the settings in user-readable and modifiable
   format.

   The default config file looks like (where the second column variables
   are substituted with the actual values):

   TOTAL_NUMBER_OF_CLUSTERS	TOTAL_NUMBER_OF_CLUSTERS
TOTAL_NUMBER_OF_FEATURES	TOTAL_NUMBER_OF_FEATURES
TOTAL_NUMBER_OF_DOCUMENTS	TOTAL_NUMBER_OF_DOCS
MINIMUM_USEFUL_FEATURES_AT_NODE	MIN_USEFUL_FEATURES_AT_NODE
MAXIMUM_DOCUMENT_LENGTH	MAX_DOC_LENGTH
VARIABLE_DOCUMENT_LENGTH	VARIABLE_DOC_LENGTH

   RELATIVE_FILE_NAMES	RELATIVE_NAMES
OUTPUT_DIRECTORY_NAME	DIRNAME
HIERARCHY_CONFIGURATION_FILE	HIERARCHY
DOCUMENT_DATA_FILE	DOC_DATA
DOCUMENT_STATS_FILE	DOC_STATS
where CLUST_PROB_OPTIONS are as follows:

0 = create random apriori cluster probabilities (in this case apriori
cluster probabilities are randomly generated for all leaf nodes)

1 = create equal apriori cluster probabilities (in which case all leaf
nodes get apriori P = 1/(number of leaves) )

2 = read in the user-defined cluster probabilities, in which case the
last line of the config file should contain a filename which contains
list of cluster number followed by the apriority probability for that
cluster, with no separators other than whitespaces, eg:
1 .25 2 .5 3 .25
(in which case only the clusters listed will have apriori
probabilities, and the documents will only be assigned to them,
regardless of the actual hierarchy given by the hierarchy file) */

config_info *get_config(int argc, char **argv);
/* takes in the config file, and parses it, assigning the proper fields
to the config structure which it then returns. For the proper format
of the config file, see above. */

#endif
/* if we are providing arguments on the command line, 
then there should be altogether 11 of the arguments:
./find-thetas <total number of clusters>
<total number of features> <total number of docs>
<min useful features at a node> <max doc length>
<variable doc length?> <relative file names?>
<output dir name> <hierarchy file> <predefined probabilities?>
<cluster probabilities file>
*/
Bibliography


