# Clustering Full Text Documents

## Abstract

An index or topic hierarchy of full-text documents can organize a domain and speed information retrieval. The automatic generation of such an index is an ideal application for unsupervised learning, where the learner creates an integrated summary of a domain. Traditional indexes, like the Library of Congress system or Dewey Decimal system, are generated by hand, updated infrequently, and applied inconsistently. With machine learning, they can be generated automatically, updated as new documents arrive, and applied consistently. Despite the appeal of automatic indexing, organizing natural language documents is a difficult balance between what we want to do and what we can do. For optimal performance, the machine learner must know or acquire all that a human library patron knows about natural language. This will be beyond the capabilities of machine learning for many years to come. For the foreseeable future, we will have to apply approximate solutions to the problem and do whatever data engineering is necessary to yield good performance. This paper describes an application of clustering to full-text databases, presents a new clustering method, and discusses the data engineering necessary to use clustering for this application. In particular, the paper deals with engineering the feature set to permit learning and otherwise engineering the data to match assumptions underlying the learning algorithm.

## 1. Introduction

Few problems fit neatly into the parameter space of a given machine learning method. Often, the data must be engineered to match the assumptions underlying the learning method. This paper discusses the simplification of natural language data for probabilistic, unsupervised learning across a collection of full-text documents.

There is an ever-increasing number of on-line, full-text documents available and, as a result, access to just the right document is becoming much more difficult. As in a traditional library, this electronic maze of documents should be organized to make items easy to retrieve and browse. Ideally, this index would be generated automatically and would support hierarchical browsing of related documents.

For example, a patron may want documents on the population of the California county of El Dorado. The user of such a system would not need to enter a query of ?El ADJ Dorado AND (county OR california OR CA) AND population?. Instead, at each level of a hierarchical index, they would be given a few terms to choose among. Their choices would determine the next level of the hierarchy presented. Once they have found useful documents, they may wish to be notified of any other documents that arrive on that topic.

Generating a topic hierarchy is an ideal application for unsupervised learning, in which an integrated overview is created for a domain (cf. Rasmussen, 1992). In traditional libraries, hierarchical indexes, like the Library of Congress system or Dewey Decimal system, are generated by hand, updated infrequently, and applied haphazardly by individual librarians. With machine learning, they can be generated automatically, updated as new documents arrive, and applied consistently.

As alluring as this possibility is, there is a problem. Clustering natural language documents is a difficult balance between what we want to do and what we are able to do. To maximize performance, the learner must know or acquire all that a human library patron knows about natural language. Unfortunately, such profound understanding is many decades away. For the foreseeable future, some data engineering is necessary for any application of machine learning to the organization, indexing, and retrieval of full-text documents. Even if we wanted to design a more appropriate learning algorithm that appeals to the rich structure of natural language, it is not clear where to start or even whether we could achieve better performance (cf. Soderland & Lehnert,1994).

In the rest of the paper, we present the learning task, our learning method, and the data engineering difficulties encountered. Section 2 discusses the task of learning a hierarchy of documents. Section 3 presents a method for learning overlapping clusters. Section 4 briefly lists the data engineering problems we encountered. For

each problem described, we present our approach and our reasons for that approach.

## 2. Organizing a Document Hierarchy

There are many ways to search a complex collection of on-line information and many have direct analogs in the traditional library. Currently, we can perform content searches, browse hierarchically organized information, or ask a librarian to find books similar to an example. Once we have a book, we can scan its index or the table of contents.

With a subject index in the card catalog, we can find the card labelled ?El Dorado? to find which books are available. Then, once we reach the stacks of books armed with a Library of Congress call number, we can find our book. If we cannot find the book or it is not quite what we wanted, we can browse the nearby, related books, those that have a similar call number. Finally, if we want to be aware of any new related information, we can check that shelf every week or so.

A key organizing principle for the library is the use of the Dewey Decimal system or Library of Congress system for organizing related materials. This system allows patrons to browse books on the same or similar topics, looking for those they did not know about but might like to read anyway. The patrons do not need to have a fully specified query as long as they get to the right shelf of books. In addition, the organization scheme allows a librarian or patron to quickly check if there are new books on a particular topic.

Hierarchical indexes can play an important role for on-line documents. A large, growing source of on-line documents is the World Wide Web (Berners-Lee, 1994), a set of conventions for publishing information on the internet, and already many people have begun to publish hierarchical indexes to the information available on the World Wide Web. Indeed, one of the most popular WWW documents is a hand-built hierarchical index of the Web called ?YAHOO? (http:// www.yahoo.com/).

Our goal is to build one of these indexes automatically. It would serve several purposes. First, it could be used to browse on-line documents quickly. For instance, a patron might want to know the population of El Dorado, CA. They might start by typing ?El Dorado? and choosing from the following hierarchy:

The ?County, California? selection would then expand to include entries for, ?IRS ?, ?Population?, ?Maps?. Then by selecting ?Population?, the patron would discover one document containing census information: gopher://gopher.census.gov:70/00/Bureau/Population/Estimate/County/coues_CA.txt. Finally, they would find that as of July 1, 1994, 144,839 of the 31,430,697 people in California lived in El Dorado county. The system constructs the presented hierarchy from pieces of the hierarchical index it generated ahead of time.

Second, a hierarchical index could be used to organize an individual?s ?hotlist?, a list of favourite documents. The hierarchical structure of all documents can impose order on a subset of documents. Third, a hierarchical index could be used automatically to suggest extensions to a ?hotlist?. Essentially saying, ?Because you read pages related to El Dorado, CA, you might be interested in the following new page that was recently found: Los Rios College District (gopher://gopher.losrios.cc.ca.us:70/00/info)?.

## 3. Building a Hierarchical Index

To build an index of documents, the learning or clustering method must group related documents together (Rasmussen, 1992). In addition, because different people will wish to use the resulting index in many different ways, the learner ideally should build several overlapping hierarchies of the documents (Martin & Billman, 1994).

We will use an algorithm designed to learn several different divisions of the documents. Given a set of n documents, it will learn one or more ways to divide the documents into related subsets. Each different way to divide the documents will be called a factor of the documents. Each cluster of documents for a factor is given a name called a value of the factor. For example, a set of documents on El Dorado might be clustered by type of document: Postscript, HTML, GIF etc. Hence, one factor would be the type, with values corresponding to each of the types of document. Another factor might correspond to the content, with values related to different geographical locations. The method used is similar to that introduced by Martin and Billman (1994) and similar to a clustering algorithm used by Anderson and Matessa (1992).

The algorithm, FL (Factor Learner), is a learning method that constructs clusters from empirical data. It takes as input a set of variable-value vectors and produces one or more factors. A variable-value vector is simply a list of variable names, each paired with one possible value for that variable. FL searches for an appropriate set of factors. First, it uses classical tests of association to decide how

many sets of overlapping clusters to build, i.e., to decide which observable variables each factor should cover and how many factors to use. Second, it uses an approximation of Bayesian clustering to determine a highly probable number of values per factor and highly probable conditional dependencies between factors and observable variables.

Because we expect that we will have to engineer the data in order to use FL, we must be clear about what assumptions underlie it. FL is based on the following five assumptions:

1. FL assumes that the domain of objects to be clustered really are clustered in the world. For example, all the documents in one cluster were generated by members of the same research community or are otherwise related by some structure in the world. However, we do not have any direct access to the clusters from the world only to observable features.

2. In general, the clusters in FL are used by providing some variable values and requesting other variable values to be predicted. For example, if documents have been clustered by keywords, ideally some keywords would be given and FL would be used to predict what other keywords are relevant.

3. The objects to be clustered are represented as variable-value vectors such that all possible variables have an associated value. This means, for example, that if some relational information is relevant to a task, it will have to be changed into a zero-th order representation.

4. There is no mathematical reason for additional levels to the hierarchy. Whatever the probability distribution that represents the domain, it can be represented by one level of factors. The only rationale for multiple levels of a hierarchy is to improve retrieval time.

5. Once a factor value is known, no features relevant to that factor are inter-correlated.

## 3.1 Finding the factors

The topology of factors, i.e., the number of factors and which variables are relevant to the factor, is determined using classical tests of association between pairs of discrete variables.1 A structure of factors can be identified using the pattern of dependence and independence between pairs of observable variables.2 Two rules govern the structure:

1. If two variables are interdependent, one or more factors directly influence both.

1. FL currently uses Pearson?s test of association. However, there is no theoretical commitment to a particular test.
2. Higher-order conditional dependencies and independencies could provide converging evidence but are not strictly necessary.

2. If two variables are independent, no factor directly influences both.

FL constructs the factors by finding a set of cliques in an undirected graph of dependency relations. In this graph, a direct link between two variables means they are dependent, and no link means they are independent.

First, FL constructs the graph by introducing a link between two variables when the test of association indicates dependence between those two variables. FL labels each link with the probability (computed using classical statistics) that those two variables are dependent. Second, FL extracts cliques by iterating through the variables and adding a variable to an existing clique if the variable is dependent on all other variables in that clique. After doing this, if the variable has dependency links not already in a clique, FL starts another clique with two variables. Third, FL performs a greedy search for the cliques that jointly maximizes the sum of the labels represented in the set of cliques.

Each of the resulting cliques corresponds to a factor in the factor structure and the variables in the cliques are the variables that directly depend on that factor. When two or more cliques share a variable, then that variable has two or more factors on which its value depends.

## 3.2 Constructing the factors

Once the factors and related variables are chosen, FL must choose the number of values for each factor, that is cluster the objects according to a factor. It must also choose the conditional probabilities. FL attempts to do so very rapidly, only approximating the normative Bayesian method for learning hidden variables. The normative way to learn hidden variables is to consider all possible numbers of values and all possible assignments of hidden variable values to the instances in the dataset (Cooper & Herskovits, 1992; Cooper, 1994).

To be faster than the normative approach, a learner must have some way to find a highly probable assignment without considering all possible assignments of factor values to instances. FL uses an incremental, greedy search for a single most probable assignment of values. FL uses a variation of Anderson and Matessa?s (1992) single factor algorithm. It assigns factor-values to each instance one at a time. The assignment for the nth instance is the most probable given the current set of values for factors which is based on the first n-1 instances.

Also, FL does not consider all possible combinations of factor values when seeking the most probable assignment for an instance. Instead, it finds the most probable value for each factor serially, each time including the influence of previously chosen factor-values. For example, if there are three binary factors, there are eight possible combinations of factor values. FL does not examine all eight possibilities. Rather it greedily considers the values serially. To reduce the bias that serial assignment introduces, FL performs the assignment to factors in a random order.

In all cases where probabilities must be estimated from frequencies we used the following formula:

The quantities, , are parameters of a Dirichlet distribution (one for each value of the variable) and is the sum over those parameters. We set all the ?s to be 1.0 and never vary them. The probability of a new value is a constant and its magnitude is a parameter.

FL is very fast. Each of m instances is tested against each of r values for each of the factors, making for tests. Each test iterates over all observable variables, producing an overall cost of . If we assume a constant number of values, this becomes .

## 3.3 An example

Figure 1 shows an example factor structure. We took a random sample of 5000 instances based on the distribution shown in the Figure and gave those instances to FL. In the Figure, factors are represented by rectangular boxes and observable variables are represented by circles. Each of the values for each factor

has an associated prior probability that is given in each box. As well, there is a conditional probability of each observable variable value given any factor values.

Figure 1: The target factors from which the example data was generated.

Those conditional probabilities are represented in the matrices associated with the arrows. For example, the probability of V3 = 1 given that A has value 1 and B has value 0 is 0.2. Also, the probability of V4 =0 given that B has value 1 and C has value 1 is 0.1

Given this data, FL first computed the classical test of association for all pairs of variables. It returns the probability of the observed association between the variables given that the variables are independent.

Table 1 shows the results of the tests of association. FL concludes that there are dependencies between V1, V2, and V3, a dependency between V3 and V4, and dependency between V4 and V5. As a result, FL creates three factors: one that influences variables V1, V2, and V3, one that influences variables V3 and V4, and one that influences V4 and V5.

After learning, FL recovered the correct factor structure as shown in Figure 1. However, for two of the factors, it proposed additional values that had extremely low prior probabilities. These extra values are artifacts of the incremental hillclimbing search. FL should have a means of eliminating those values that are not adequately supported by the instances.

## 4. Applying FL for Full-Text Clustering

Although FL has many advantages over alternative document clustering approaches, it is still woefully inadequate for the task of automatic index generation. Several information retrieval researchers have applied statistical clustering methods to the organization of documents (Rasmussen, 1992). FL is a significant improvement over those approaches because it permits cross classification using overlapping hierarchies. The documents are not given one ?best? organization. Instead, FL permits several good organizations. In addition, FL, because it is incremental, naturally permits efficient cluster maintenance (Can & Ozkarahan, 1989). The structure of factors and clusters can be updated easily as new documents are added. Finally, unlike traditional clustering, FL?s applicability for particular tasks can be more easily evaluated because its underlying assumptions can be clearly stated. This is also true for such systems as Fisher?s (1987) COBWEB and Anderson and Matessa?s (1992) method.

However, we are not here to sing the praises of this method. Rather we will highlight the ways in which FL is ill-suited to our task. Because of its underlying assumptions, FL does not seem the right tool to build a retrieval hierarchy of natural language documents.

1. It does not build a hierarchy

2. It cannot accept natural language

3. Even if two documents share features, people may not find those documents similar for all purposes.

4. If words are binary variables (they either occur or do not) there are too few shared variable values.

5. The majority of shared words are not at all correlated with document similarity.

6. Too many documents are very similar because they share the absence of many words.

We discuss these problems with FL in terms of how we addressed them. Whenever there is a mismatch between a learning method and data, there are three general ways to reduce the difference:

1. Build a better learning method,

Table 1. Triangular matrix containing the probability that two variables are independent. Probabilities below 0.05 taken as evidence of association.

2. Simplify or restrict the task until it meets the learning assumptions

3. Transform the data to increase the probability of uncovering important structure.

## 4.1 Building a Better Learning Method

First, if there is enough time, the problem is general enough, and the task is well understood, we can design a specialized machine learning algorithm built just for the task. Clearly, we cannot do this for the whole task of natural language understanding. However, we can address some of the mismatch by modifying our algorithm. The FL algorithm does not learn hierarchies. We can simply pull a page out of COBWEB?s (Fisher, 1987) book and apply the clustering recursively. Once one level of factors has been formed, each cluster of documents can itself be further divided.

This will clearly lead to a hierarchical structure. However, it should be clear that this is an ad hoc solution to achieve the task we want. There is no clear definition of what each level of the hierarchy means. There is no way to argue that this hierarchy is better than some other hierarchy.

In fact, building a hierarchy in this way seems to violate one of the assumptions underlying FL. Once a set of factors has been learned, each cluster of documents should not have correlated variables. If there are no correlated variables, there is no reason to further subdivide the cluster. In general, however, subsequent levels of the hierarchy will cluster based on different variables. The first level of clustering will group based on a subset of the variables and result in clusters in which that subset of variables is no longer intercorrelated.

## 4.2 Making Index-building a Clustering Problem

Building an index of full-text documents and using that index for browsing is not like building a taxonomy of plants and animals and then using the hierarchy to make predictions about new plants or animals. Table 2 summarizes two ways that makes clustering a difficult match for index-building: FL cannot use structure found in the data and the task of retrieval is not the same task that was being optimized during learning.

Table 2. Meeting FL?s assumptions.

| Problem | Our Approach |
|---|---|
| Data represented without relations | Extract keywords and proximity features |
| Predicting relevance. | Adopt the cluster hypothesis |

### 4.2.1 No structure

On the one hand, it is not as easy to identify unary predicates concerning a text as it is for animals. This is possibly because we have more experience translating relational information about animals into simple features. Whatever the case, in order to use FL or another clustering approach, we must translate the rich structure of language into simple features.

There are many ways to do this. The simplest of which is to identify keywords in the text and remove all sequential or proximity information. At the other extreme is attempting to parse the entire text. Such parsing is very slow and current methods paradoxically lead to diminished performance (e.g., Soderland & Lehnert, 1994). Many information retrieval researchers have chosen some intermediate approach, including frequency counts , simple adjacency information and possibly rudimentary syntactic and semantic information (e.g., DeJong, 1979; Mauldin, 1991, Soderland & Lehnert, 1994).

For our application, we have chosen to identify keywords and maintain frequency and proximity information, such that two documents are more similar if they have certain keywords within N of each other. This gives us considerable flexibility in mimicking syntactic and semantic structure without requiring natural language capability. More specifically, for every cluster, FL maintains a description of the frequency of every keyword that appeared and a matrix of the expected distance between any two keywords.

### 4.2.2 Prediction and Retrieval

Even if this is an adequate simplification of natural language, library patrons want to use the factors to do something they were not designed to do. The factors and clusters are learned to recover the underlying factors that produced the documents. For example, it might cluster all the documents generated by one group of researchers. The result of FL?s learning would be ideal for predicting what other keywords are strongly correlated with a provided query of other keywords. However, a library patron simply wants documents relevant to their current real-world task. That task could be anything, and we cannot guarantee that FL will cluster together documents that the arbitrary user will find relevant.

To be clear about what FL can do, we will place a restriction on what types of retrieval the patron can expect to achieve. The user?s definition of what documents are relevant to their task should be a list of words, phrases, and partial phrases. The strings of words that are allowed depends further on how the keywords are extracted (see following subsection). Finally, during retrieval, the patron will find their document by repeatedly choosing the most relevant word, phrase, or partial phrase from a menu. This process will lead them to the documents mostly likely to

share all their words, phrases, and partial phrases.

In this view, relevance to the patron must be changed from relevance to a task to a list of ideas that themselves are possibly relevant to the task. The translation from the task to the list of ideas is the patron?s responsibility. This restriction is very similar to that made throughout the area of information retrieval and in the context of clustering is sometimes called the ?clustering hypothesis? (Rasmussen, 1992).

## 4.3 Engineering the Features

Assuming FL keeps track of keywords and keyword positions, can the underlying structure of documents be learned? If the authors of the documents chose their words carefully, then the keywords likely will be most highly related to the key ideas of those documents. However, this is rarely true. Blindly applying FL to a set of documents and using only the words that occur in the documents as possible keywords leads to very poor clustering performance. Related documents do not get clustered together.

### 4.3.1 Too few matches

There are three major reasons why FL does not work well using only the surface words (see Table 3). First, there are so many synonyms in English that it is hard to recognize two similar documents. They would have to use enough of the same words. A common solution to this problem is to use a dictionary or thesaurus to translate words into some canonical form and increase matching.

We decided to use WordNet (Miller et al., 1992) to add to (rather than replace) the keywords. In particular, we added synonyms, hypernyms, and contains words. A hypernym is a word that is more general than the given word. ?Vehicle? is a

hypernym of ?bicycle?. A contains word is a word that describes a system of objects, one of which is the given word. ?Automobile? is a contains word for ?radiator?. For efficiency, these words are only added when a document is being compared with another document or a summary of a cluster. For efficiency reasons, the only thesaurus words that are added are those that are shared by both the document and its comparison. We have found that adding synonyms is generally not useful. More often, the hypernyms fill that role adequately.

Beside using the thesaurus, we also augment the keywords with a limited list of keywords from hyperlinks or references that point to or refer to the document being indexed. For example, in the WWW, most documents have pointers or hyperlinks to other documents. If you click on one of the hyperlinks, you are shown another document. Each of the hyperlinks has a short description of what document will be found by clicking there. The words in the short description for a given document can be useful in describing the content or type of document it is. Again, keywords in this class are only added to a cluster summary if it is shared by some of the documents in the cluster.

### 4.3.2 Too many matches by missing keywords

Most or all the documents will be highly similar in that they do not share most of the dictionary of words. Each document contains only a fraction of the possible keywords. Two documents can be seen as very similar simply because they both omit 2000 important keywords. FL applied blindly would cluster unrelated documents solely because of the shared absence of words. This is a side effect of using binary variables to represent the presence or absence of words.

For practical reasons, when we can we treat the absence of words as a missing variable. If two documents both do not have a particular keyword, we assume that they could and we just have not been told. Similarly, if a document and a summary of a cluster both do not have a particular keyword, we assume that they could share that keyword or not, we just do not know.

However, if one document has a keyword and the other does not, we assume that the documents differ in that respect. We no longer assume that the document without the keyword has a missing variable.

This is a messy bit of practical engineering that prevents a clear description of what the resulting clusters mean. If one document omits the word ?retrieval?, FL in some cases treats that as meaning that the document could have ?retrieval? if we read between the lines or it might not have the word ?retrieval?. We just do not know. However, in other cases, FL treats the omission as meaning that the document definitely does not have the word ?retrieval?.

By adopting this policy, we greatly simplify the building of clusters by only focusing on clearly relevant keywords. As well, we eliminate the problem of strong similarity by omission. In the process, however, we interpret the data inconsistently.

### 4.3.3 Too many irrelevant matches

The thesaurus and references were used to increase the overlap between related documents. There is still a problem with high overlap between unrelated documents. There are many words that are shared by documents just by virtue of both documents being in English. As well, there are many documents that use different meanings of the same words.

The standard response to extraneous matching of keywords is to use a stop list of words that are too common. These are usually the 100 most common words. However, in a clustering setting, the stop list will be

different for different clusters and can grow to be very large deep in
the tree of clusters. Any word that appears in most or all the
clusters of a factor are essentially stop words, words so common they
can be ignored.

We implemented a stop list as a list of words and the probability that
the word will be in an arbitrary document. Any word more probable than
a threshold is ignored. At each level of the hierarchy, the stop list
is augmented by the cluster summary at the next level up. Essentially,
this implements a statistical measure of informativeness or
interestingness.

## 4.4 Preliminary Results

Preliminary results suggest that at least our decisions about
augmenting and filtering the words of documents helps organize
documents intuitively. We accepted the Yahoo organization of documents
as a gold standard and found a branch of that index with 54 documents
(http://www.yahoo.com/ Society_and_Culture/Holidays/). We applied FL
to build a hierarchical index for those documents and compared the
learned hierarchy with the provided one.

The best hierarchy generated by FL when only actual words were used as
features seems considerably different from the Yahoo directory,
sharing only a few headings. When we permitted the introduction of
synonyms, hypernyms, and partof terms, the best FL hierarchy shared
half of the headings with the Yahoo directory.

## 5. Discussion

When faced with a mismatch between a database and existing machine
learning method, we have three choices. Either the dataset and task
characteristics seem general enough to motivate the design of a
specialized machine learning algorithm,

the task can be redefined or simplified, or the data can be engineered
to fit an existing learning method. When applying to the problem of
automatic generation of a hierarchical index, we did all three. In the
best of all possible worlds, we would always generalize existing
learning methods to include our new task. However, for the next

decade, this will not be possible when dealing with natural language data. We must make do, simplifying our tasks and engineering our data.

## Acknowledgments

Many of the ideas presented in this paper concerning feature engineering arose in my frequent conversations with Peter Turney.

## References

Anderson, J. R. & Matessa, M. (1992). Explorations of an incremental, Bayesian algorithm for categorization. Machine Learning, 9, 275-308.

Berners-Lee, T. The World Wide Web. Communications of the ACM, 37.

Can, F. & Ozkarahan, E. A. (1989) Dynamic Cluster Maintenance. Information Processing and Management, 25, 275-291.

Cooper, G.F. (1994). A Bayesian method for learning belief networks that contain hidden variables. Journal of Intelligent Information Systems.

Cooper, G.F. & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. Machine Learning, 9, 309-347.

G. F. DeJong. (1979). Prediction and Substantiation: a New Approach to Natural Language Processing. Cognitive Science, 3.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. Machine Learning, 2, 139-172.

Martin, J. D. and Billman, D. O. (1994). The acquisition and use of overlapping concepts, Machine Learning.

Mauldin, M. L. (1991). Retrieval Performance in FERRET: A Conceptual Information Retrieval System, In the Proceedings of the 14th International Conference on Research and Development in Information Retrieval.

Rasmussen, E. (1992). Clustering Algorithms. In W. B. Frakes & R. Baeza-Yates (Eds), Information Retrieval: Data Structures and Algorithms. New Jersey: Prentice Hall.

Soderland, S. and Lehnert, W. (1994). Wrap-Up: a Trainable Discourse Module for Information Extraction, Journal of Artificial Intelligence Research, 2, 131- 158.