

Language Technology Project 2007
Word of the Day

Sophie Arnoult Jelle C. Kastelein Mark Kroon
Stephan Schroevers Andrea Schuch Klara Weiand

Universiteit van Amsterdam

February 3, 2007

Contents

1	Introduction	1
2	Theoretical background	2
2.1	XPM	2
2.2	NLP	3
2.3	Categorisation	6
2.4	Identification	7
2.5	Clustering	8
2.6	Visualisation	11
3	Implementation	11
3.1	XPM	11
3.2	NLP	12
3.3	Categorisation	12
3.4	Identification	14
3.5	Clustering	14
3.6	Visualisation	14
4	Evaluation	15
4.1	Methodology	15
4.2	NLP	16
4.3	Categorisation	17
4.4	Identification	17
4.5	Clustering	18
4.6	Cluster Keywords	19
4.7	Visualisation	20
5	Discussion	21
5.1	XML preprocessing	21
5.2	NLP	21
5.3	Categorisation	22
5.4	Identification	22
5.5	Clustering	23
5.6	Visualisation	23
6	Conclusion	24

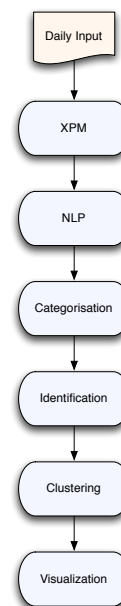


Figure 1: Process chain

1 Introduction

Our goal for this project was to produce a “Word of the day” application, similar to that described in Eiken et al. [2006]. Whereas the original application focused on Norwegian newswire, we chose English as our source language, and we considered two different source types: newspapers and blogs. Our newspaper input consists of the *New York Times* section of the *AQUAINT* corpus, while our blogs corpus is extracted from *LiveJournal*. Furthermore, we categorized our input data in order to make our application domain-specific. We did this using a hybrid approach, that combines static and dynamic categorization.

For practical reasons, we separated the implementation of our application into six modules, each with a distinct function. To speed up development, we chose to keep the modules as independent from each other as possible. To this end, all modules communicate using XML files that are formatted using two agreed-upon document type definitions (DTDs). The process chain is shown in Figure 1.

Input data first goes through the *XML Preprocessing* module (XPM), which processes the data to

simplify the processing tasks of the following module. The *NLP* module performs language analysis on the texts and identifies possibly interesting words and phrases. The *categorization* module uses this information to group the incoming texts into a fixed number of predefined categories. The *identification* module then selects a certain number of category-dependent keywords. The *clustering* module ends the selection process as it dynamically subdivides the existing categories and provides keywords for the resulting sub-categories. The *visualization* module finally displays our “words of the day” using two methods: *Tag Cloud* and *Click Through*.

Theoretical background for each module is given in section 2. Implementation issues are then presented in section 3, and evaluation methodology and results in section 4. Finally, the results of every module are discussed in section 5.

2 Theoretical background

In this section the theory behind each of the modules is described in turn.

2.1 XPM

To determine the words that are currently interesting, it is necessary to gather text publications from the web on — at least — a daily basis. This data may be in any format and will differ for each source.

In order to simplify the text processing by the other modules, the program includes a front end module that may handle various input formats and converts these to a strict, agreed upon output format. This module is named the XML preprocessing module, or XPM for short.

XPM must meet some specific requirements to simplify the language processing for the remaining modules. The module must also be able to support multiple input formats and be easily extensible. It should output strictly English texts. Furthermore some *normalisation* operation should be applied to the text, with the goal of removing HTML and other odd character sequences (smilies, for example). This operation is partially source specific. Normalised text allows for more successful stemming by the NLP module.

2.1.1 Extendability

Due to the variety of possible input formats, it is not possible to create a one-size-fits-all parser. Each new input format that will be supported will require adjustments to the code. By giving the module a modular structure this process of extending the code base can be simplified greatly. To this end XPM uses a single generic parser class that can and must be extended for each new input format that we may choose to support.

2.1.2 English text detection

Since the program was made for an English audience and some of the modules use methods that are strictly centred around the processing of English texts (the stemmer in the NLP module, the classification module), it is important to output as few non-English data as possible. For this purpose XPM uses an English stopword list, which is shipped with NLTK, the Natural Language Toolkit.

The “English score” S_{en} of a set of words (text) W is calculated using the stopword list L as follows:

$$S_{\text{en}} = \frac{\sum_{w \in L} c(W, w)}{|W|}$$

where $c(W, w)$ counts the number of occurrences of the word w in the text W .

After some testing a threshold of 0.30 appeared to give very reasonable results. This does not prevent some bilingual texts from being accepted. This may be explained by the assumption that a non-native English speaker uses relatively simple and common English words, thus increasing the value of S_{en} . In some cases this effect is not sufficiently cancelled by the the non-English contents of the text.

2.1.3 Text normalisation

XPM utilises several text normalisation methods. First of all there is the removal of redundant whitespace. Only spaces are preserved. Newlines are converted to a full stop followed by a space. This is because blog users often do not finish their sentence with a full stop when they start the next sentence on a new line.

Second there is the normalisation of punctuation. Exclamation and question marks are converted to

regular full stops. Several different quote characters are converted to the regular ascii single and double quote characters. Redundant full stops (occurrences of "...") and additional dots that are inserted in place of newlines) are removed.

Also removed are any HTML tags, simply by matching sequences between angle brackets. Obvious URLs and email addresses are not outputted either.

Optionally the module may remove any character that has a Unicode ordinal above a certain threshold. This allows XPM to strictly output e.g. ascii or latin-1 characters (by setting the threshold to 127 and 255 respectively). In the future, however, the module would be extended to support e.g. Korean texts, then specifying just an upper threshold is not enough. In such situation a *range* of Unicode ordinals would be required. This functionality is currently unimportant to our English-centred system and hence not implemented in XPM.

2.1.4 Date-time extraction

The word of the day is determined by comparing publications today with publications during some predetermined time span before today. This data is referred to as the reference corpus. The reference corpus must be updated on a daily basis, namely by purging the oldest text and adding yesterday's publications. For this purpose it is important that each document is timestamped. Different sources may supply timestamps in different formats. Because of this, XPM knows how to parse timestamps and converts these to UTC. This allows for easy date comparison by the other modules.

2.2 NLP

The goal of the Natural Language Processing (NLP) module is to function as a filter that preselects words (and phrases) which have the potential to become "Word of the Day". It extracts the types of words and phrases that can be expected to vary in frequency over time and thus reduces the amount of data that the following modules have to process. The use of the NLP module is thus based on the assumption that certain types of elements can be safely ignored by a "Word of the Day" system.

Additionally, the extracted elements are annotated with information that is needed by the con-

sequent modules.

Linguistics distinguishes between function words and content words. Function words (e.g. determiners, pronouns and auxiliary verbs) are those that mainly convey grammatical relationships within a sentence, while content words like nouns, verbs and adjectives refer to actions, concepts, objects and their properties. This distinction roughly coincides with the concept of closed (word classes that are typically fixed and do not easily add new elements) and open word classes (those that are constantly changing and growing).

Since function words are concerned with grammar rather than lexical meaning, we assume that their frequencies do not significantly differ between texts and over time.

The NLP module only extracts nouns, named entities, verbs, adjectives and noun phrases. The resulting elements are enhanced with information about frequency, part-of-speech tag, stem and document frequency ([Church and Gale \[1995\]](#)) and passed on to the following module. Figure 2 shows the different parts of the system which will be discussed in the following sections.

2.2.1 Tokenising

In order to prepare a text for further processing, it is first tokenised, that is, segmented into processing units. Words are separated from punctuation marks, while units that contain punctuation mark characters as a constituent, like prices or hyphenated words, should remain unchanged.

One of the biggest challenges in tokenising is the disambiguation between sentence-ending full stops and those that mark an abbreviation. A simple heuristic that is used in the NLP system is to consider a string of characters that ends in a full stop (and that does not contain any other full stops) an abbreviation if it is followed by further punctuation marks or a word starting in a lower case letter. The performance of this technique could be improved by keeping a list of known abbreviations that have been found so far using this method and checking each token in need of disambiguation against it. Alternatively, the list of abbreviations could also be created manually ([Gregory Grefenstette \[1994\]](#)).

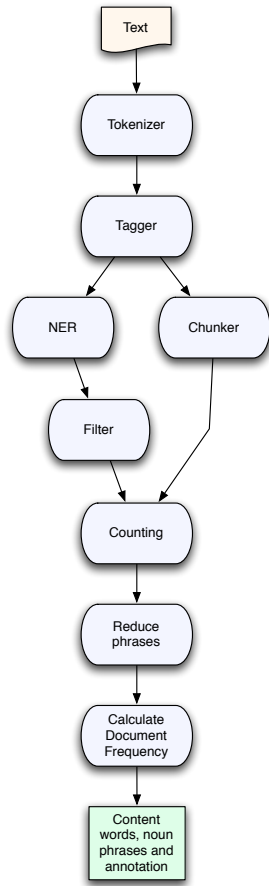


Figure 2: The NLP module

2.2.2 Part of Speech Tagging

In part-of-speech tagging, each token is assigned a tag that indicates its word class. All taggers discussed in the following were trained on the Brown corpus from 1961 which contains about 1 million tokens in 15 different categories. The Brown tagset contains 87 different word classes [Jurafsky and Martin \[2000\]](#).

Hidden-Markov-Model or n -gram taggers use supervised learning and find the best tag for a given word by choosing the tag for a word that is the most likely given the preceding $n - 1$ tags, or, more formally, the tag that maximises $P(\text{word}|\text{tag}) \cdot P(\text{tag}|\text{previous } n - 1 \text{ tags})$ ([Jurafsky and Martin \[2000\]](#)). Thus, for a Trigram tagger, the second term of the formula corresponds to the probability

that a candidate tag occurs given the two tags that precede the current word, while Unigram taggers do not take context into account, but simply assign the tag that occurred most often as the tag of the word in question in the training data.

Generally, a higher n yields better tagging results as the increasing context size allows for more fine-grained disambiguation. However, as the size of the context increases, sparse data problems arise, since the number of possible contexts for a word also increases.

Backoff schemes allow to combine fine-grained disambiguation with broad coverage by defining a backoff tagger for each tagger that is called when the first tagger is unable to find a tag for a given word. Since backoff taggers can have backoff taggers themselves, a cascade of taggers can be specified, allowing to assign each word the best possible tag given the available taggers.

Affix tagging, a special case of Unigram tagging ([Bird and Loper](#)), seeks to remedy another problem, namely that of previously unseen words, by only considering the final characters of a word.

The Brill tagger, unlike the class of taggers discussed previously, is not stochastic but rule-based. The rules are formed by comparing the tags of a text to those in the training data and inducing transformational rules from the differences between the two.

Table 1 shows the ratio of correctly tagged tokens for different taggers trained on subsection A (“Press: Reportage”) of the Brown corpus and tested on subsection C (“Press: Reviews”) of the same corpus¹. The upper half of the table contains the scores for single taggers, while the taggers in the lower half use all taggers above them as backoff taggers (in reverse order)^{2,3}.

The numbers illustrate both the sparse data problem brought about by increasing the context size –the accuracy of the n -gram taggers without backoff decreases as n grows–, as well as the unknown word problem – adding an Affix tagger to the Unigram tagger increases performance by al-

¹The accuracies were determined using tag accuracy function provided with NLTK.lite

²No backoff taggers can be specified for Brill taggers.

³At the lowest level, all taggers in the lower half fall back on a default tagger that assigns every token the tag “nn”. It is unclear why this leads to a decrease in accuracy for the Affix tagger.

Table 1: Tagger accuracies

Tagger	Accuracy
Affix	0.375
Unigram	0.727
Bigram	0.376
Trigram	0.218
Brill	0.79
Affix	0.332
Unigram	0.81
Bigram	0.817
Trigram	0.82

most ten percent. The best-performing tagger, namely the Trigram tagger that uses the Bigram, Unigram, Affix and default tagger as backoffs, was used in the module.

2.2.3 Named Entity Recognition

Named Entity Recognition is the task of extracting and classifying proper nouns from text. There are various approaches to this problem ranging from the use of manually created look-up lists (gazetteer) or heuristics to statistical models like Support Vector Machines and Hidden Markov Models (Sekine).

As the task at hand does not require the classification of named entities but only their identification –although the integration of a full named entity recognition system is conceivable–, a relatively simple approach using rules and two different lists is used. During a first pass, starting from the second word of the sentence, every token beginning with a capitalised letter –with the exception of “I” and and the contractions in which it occurs– is added to the list of proper nouns. A second run finds all lowercase instances of proper nouns identified in the previous run and adds those that cannot be found in a stoplist containing very frequent English words to the list of named entities.

The second pass was introduced to account for the fact that capitalisation in blog entries is often inconsistent within texts; however, especially when applied to the news domain, where standard, consistent capitalisation is used, this technique can introduce errors when a text contains proper nouns that are identical in spelling to other words that are not named entities.

2.2.4 Chunking

Chunking identifies syntactic constituents spanning multiple tokens and can serve as a preparation for full parsing (Ramshaw and Marcus [1995]). Typical chunk parsers which create a chunked representation of a whole sentence consisting of non-overlapping chunks.

However, this is unsuitable for the purposes of this module: On the one hand, we are only interested in extracting noun phrases and not in other kinds of phrases or the relation between phrases. On the other hand, noun phrases that consist of a series of nouns or named entities (“baseNPs” (Ramshaw and Marcus [1995])), can be composed of further compound nouns (as in “New York Times editors’ room”).

As it is not immediately possible to determine during chunking what subparts of a baseNP can be grouped into compounds, it is favourable to extract all possible sub-noun phrases and to remove those that do not correspond to actual compounds (e.g. “Times editors’”) later.

Chunking in this module consists of a simple matching of manually created regular expression templates defining sequences of tags and processes in four steps:

- First, “mixed” noun phrases consisting of a sequence of nouns, named entities and other tags are extracted
- The maximum expansions –the longest sequences consisting only of nouns and named entities– of the baseNPs are identified. They are added to a global list of maximum baseNPs found in all texts processed in one run.
- Templates specifying baseNPs of lengths two and three are separately matched on the maximum expansion baseNPs. Then, the first element of the maximum expansion is removed and the two templates are matched again. This step is repeated until the baseNP has been reduced to one element and no further matches are possible.
- When all documents in a batch have been processed and the frequencies of the three different types of noun phrases described above have been determined, every noun phrase extracted in the last step that has not been observed

appear as a maximum expansion at least once is discarded.

The final step thus uses the input itself to determine which sub-baseNPs constitute valid entities and which do not. Note that the number of sub-baseNPs discarded can depend on the size of the batch of documents processed: If not all sub-baseNPs also appear as maximal expansions in the text they were extracted from, the maximum baseNPs in the other texts in the batch determine which of the sub-baseNPs are kept.

If there are only few other texts, the chance of a valid sub-baseNP appearing as a maximum expansion is lower than when many texts are processed at once, and consequently analysing the same text two times can yield to different results depending on the context in which it is processed. This could be remedied of course by storing the list of known compounds.

2.2.5 Filtering and Annotation

From the tagged text, single content words that are over two characters in lengths and that are not found in a stoplist containing highly frequent English words, are extracted. Consequently, content words and noun-phrases are counted separately by part-of-speech tag.

The output of the module consists of content words and noun phrases annotated with their part-of-speech tag, non-normalised frequency, stem and non-normalised document frequency *df* (Church and Gale [1995]). The document frequency is calculated by counting the number of documents each word or phrase appears in.

2.3 Categorisation

The goal for the categorisation module is to classify documents in the corpus into predefined categories. In the overall system, document categorisation is just a means to an end, with the final purpose being the identification and categorisation of “interesting” words. From a categorised corpus, words with a “domain-specific” interestingness can be extracted. By this we mean words, which are interesting, because they occur more often than usual in text of a certain category. At the same time, a certain level of disambiguation can be achieved automatically, because homonyms (words with the same

form but a different meaning) seem more likely to occur with different meanings in the different categories. Consider, for example, the word “bank”, which has a different meaning when occurring in either the domain of finance or the domain of geography. In that case, document categorisation enables us to treat each of the homonyms as a different word. This Section presents the theoretical concepts underlying our approach. In Section 2.3.1 we introduce Naive Bayes, in Section 2.3.2 a method of feature selection and in Section 2.3.3 a semi-supervised learning algorithm.

2.3.1 Naive Bayes Classifier

We decided to use a Naive Bayes classifier for the text categorisation, as in Nigam et al. [2000] and Craven et al. [2000]. The aim is to calculate the conditional probability that the text belongs to category C given it contains the features F_1, \dots, F_n . According to Mitchell [1997] this learning algorithm works as follows:

$$P(C|F_1, \dots, F_n) = \frac{p(C)p(F_1, \dots, F_n|C)}{p(F_1, \dots, F_n)} \quad (1)$$

The algorithm is called *Naive* Bayes, because it assumes that all features are independent of each other (which is unlikely for most domains):

$$p(F_i|C, F_j) = p(F_i|C) \quad (2)$$

for every feature F_i and every other feature F_j with $j \neq i$. With this *independence assumption*, Formula 1 can be simplified:

$$P(C|F_1, \dots, F_n) = \frac{p(C)\prod_{i=1}^n p(F_i|C)}{p(F_1, \dots, F_n)} \quad (3)$$

The probability of the text features in the denominator is a scaling factor that can be ignored, since it does not change with the category.

2.3.2 Feature selection

We selected a set of n words with the highest *average mutual information* with respect to the categories C , as in Craven et al. [2000]. Here, $W_i = \{w_i, \neg w_i\}$ is a random variable determining whether or not a word w_i occurs in the document.

$$I(C; W_i) := H(C) - H(C|W_i) \quad (4)$$

Here, H means the entropy, which “characterises the (im)purity of an arbitrary collection of examples” Mitchell [1997]. The higher the average mutual information, the better the word characterises the document as belonging to one particular category.

$$H(C) = - \sum_{c \in C} Pr(c) \log(Pr(c)) \quad (5)$$

Using the definition of entropy from Formula 5, we can re-write Formula 4 as:

$$\begin{aligned} & - \sum_{c \in C} Pr(c) \log(Pr(c)) \\ & + \sum_{v_i \in \{w_i, \neg w_i\}} Pr(v_i) \sum_{c \in C} Pr(c|v_i) \log(Pr(c|v_i)) \end{aligned} \quad (6)$$

According to Craven et al. [2000], this can be reformulated as:

$$\sum_{v_i \in \{w_i, \neg w_i\}} \sum_{c \in C} Pr(c, v_i) \log \left(\frac{Pr(c, v_i)}{Pr(c)Pr(v_i)} \right) \quad (7)$$

This method provides a “domain-specific way” of feature extraction, which is more flexible than a general-purpose stop list Craven et al. [2000].

2.3.3 Semi-Supervised Learning

According to Nigam et al. [2000], semi-supervised learning is a method of using unlabelled data for the training of a supervised-learning classifier in order to be able to achieve a good result based on little labelled training data. In this paper, semi-supervised learning is used for text classification, where it is claimed that the method makes use of correlations between words in the unlabelled dataset. For example, if the word “homework” frequently co-occurs with the word “lecture”, and “homework” is an indicator for a certain class C , then “lecture” might be an indicator for C as well. Semi-supervised learning consists of the following steps: First, a classifier is built on the labelled data, then the unlabelled data is labelled by the classifier. Based on labelled and newly labelled (previously unlabelled) data, a new (improved) classifier is then built. The whole dataset is re-labelled and the result is compared to the previous labelling round. This procedure is repeated until there is no more change in the result. According to Nigam et al. [2000] it can be proved that any change in the result is an

actual improvement of the classifier. While semi-supervised learning does not lead to a better result if enough labelled training data is available, they claim it can still be used to achieve the same learning performance with less labelled training data.

2.4 Identification

The relevance of words in a given category is obtained by comparing their observed frequency with a reference frequency. To measure word salience, we are going to use the log-likelihood-ratio test, which has been proved by Dunning [1993] to perform better with corpora of unbalanced sizes and with low expected values than the previously popular Pearson’s χ^2 -test.

After a brief introduction to the log-likelihood test, we present the selection criteria we use in combination with this measure.

2.4.1 The log-likelihood-ratio test

The log-likelihood ratio λ is defined in Rayson and Garside [2000] as:

$$-2 \ln \lambda = 2 \sum_i O_i \ln \left(\frac{O_i}{E_i} \right) \quad (8)$$

with O_i the observed frequency, and E_i the expected frequency.

E_i is defined as:

$$E_i = \frac{N_i \sum_i O_i}{\sum_i N_i} \quad (9)$$

with N_i the total frequency.

For simplicity, the log-likelihood can be redefined as $LL = -2 \ln \lambda$. The log-likelihood can then be rewritten using equations (8) and (9) as:

$$LL = 2 \sum_i O_i \ln \left(\frac{O_i}{\sum_i O_i} \frac{\sum_i N_i}{N_i} \right) \quad (10)$$

Note that this is a simplification of the log-likelihood as defined by Dunning [1993], based on the assumption that observed frequencies are negligible with regard to total frequencies. This simplification is also referred to as G -test or G^2 -test in the literature.

As we are going to compare a test corpus to a reference corpus, we can rewrite equation (10) as

$$LL = 2 \cdot \left(O_1 \cdot \ln \left(\frac{O_1 N_1 + N_2}{N_1 O_1 + O_2} \right) + O_2 \cdot \ln \left(\frac{O_2 N_1 + N_2}{N_2 O_1 + O_2} \right) \right) \quad (11)$$

with O_1 and N_1 the observed frequency and the total frequency in the reference corpus, respectively, and O_2 and N_2 the observed and total frequencies in the test corpus.

All else being constant, the log-likelihood first decreases, then increases as the observed frequency increases. A minimum is reached, for a given word, as its relative frequency in the test corpus equals that of the reference corpus:

$$O_2 = O_1 \frac{N_2}{N_1} \quad (12)$$

2.4.2 Selection criteria

Only stems with an observed frequency higher than the threshold defined by equation (12) will be selected, to make sure that the stems with high log-likelihood correspond to stems that occur significantly more often in the test corpus than in the reference corpus, and not the other way round.

Stems should also have a minimum log-likelihood ratio to be selected. We set as a threshold the log-likelihood of a stem occurring once in the test corpus, but absent from the reference corpus:

$$LL_{min} = 2 \ln \left(\frac{N_1 + N_2}{N_2} \right) \quad (13)$$

Besides, stems should occur in more than one document to be selected. In fact, a word should not only have a high log-likelihood but should also be used by different sources to be accepted as a candidate keyword.

Finally, a limit is set on the number of keywords so as to keep the amount of “words of the day” reasonable: 30 keywords across all categories are handed over to the clustering module.

2.5 Clustering

A categorisation across a set of fixed labels may be satisfactory in some cases, but any word-of-the-day type system, by definition, should be very dynamic,

and should be able to handle many different topics, including those that are subject to discussion only on sporadic, or possibly even single, occasions. A fixed set of topics therefore has to be quite broad, producing only very general categories such as politics, or sports. The user, however, may be interested in much more specific subjects. The problem is now that these specific and highly dynamic topics can not possibly be efficiently captured by a fully supervised learning algorithm with a fixed set of categories, because they may have never come up before. Therefore, in order to produce a more nuanced view of the available topics, an unsupervised clustering algorithm was used.

There has been extensive research on clustering methods in the information retrieval community. For the purposes of this project in particular, it was considered desirable to use an algorithm that would be highly adaptable. That is, it should not rely on a specific fixed number of subcategories, and it should be able to handle broad articles and clusters, as well as specific ones. Finally, it should preferably produce a hierarchical output ordering in which a user can move down (or up) the hierarchy in search for more (or less) specific topics. An algorithm that satisfies these properties is the Hierarchical Agglomerative Clustering (HAC) algorithm (Rasmussen [1992], Chakrabarti [2002]), which was therefore used for the purposes of this project.

2.5.1 HAC

The Hierarchical Agglomerative Clustering algorithm is a well known traditional clustering algorithm that is used to group similar documents according to their inter-document similarity. The algorithm in itself works in a bottom-up fashion. It starts out with a set of clusters, each containing a single document. At each iteration, the algorithm then compares each cluster against the others, and groups the two most similar into a new supercluster. It removes the grouped clusters from the set, and adds the new supercluster in their place. The basic algorithm is described in figure 3. The final output of this algorithm is a binary tree of clusters called a *dendrogram*, in which the leaf nodes correspond to the single documents, and the top node to the entire document collection (and in our case, also to a single class).

```

1: Create initial set of clusters  $C$  by assigning each
   document to a single cluster
2: while  $|C| > 1$  do
3:   for all  $c_1$  in  $C$  do
4:      $i_1 = \text{index}(c_1)$ 
5:     for all  $c_2$  in  $C$  with  $\text{index}(c_2) > i_1$  do
6:        $s(c_1, c_2) = \text{similarity } c_1 \text{ with } c_2$ 
7:       if  $s(c_1, c_2) > s_{max}$  then
8:          $s_{max} = s(c_1, c_2)$ 
9:          $p_{max} = (c_1, c_2)$ 
10:      end if
11:    end for
12:  end for
13:  merge  $p_{max}$  into  $c_{1\&2}$ 
14:  remove  $c_1$  and  $c_2$  from  $C$ 
15:  insert  $c_{1\&2}$  into  $C$ 
16: end while

```

Figure 3: The basic layout of the HAC algorithm.

The biggest design concern is the similarity measure used. Any similarity measure (or the inverse of any distance measure) of choice can be used in principle, but some are more suitable than others. Another important design choice which is tied in with this first choice is that of efficiency. The HAC algorithm runs in quadratic time, and thus it is important that the similarity measure used be efficiently calculable. A total of five different similarity measures⁴ were compared superficially before final testing on the basis of their perceived estimated average cluster purity on a small test set (100 articles picked at random from the Aquaint corpus’ New York Times section), and on the balance of the generated tree structure⁵. The similarity measure with the best performance on this small test set was used to generate the final evaluation test set.

Contrary to our expectations, the relatively simple Jaccard test of similarity did the best by far on grouping similar articles close together. The Jaccard similarity for two clusters c_1 and c_2 is given

⁴The similarity measures tested were the Jaccard, Chi-squared-squared, Canberra, Self-Similarity, and Chord measures of similarity, and the inverse of the Euclidean distance between word vectors.

⁵A binary tree is said to be balanced if all of its child nodes contain an equal number of descendants, and its child nodes are also balanced, recursively.

by:

$$s_j(c_1, c_2) = \frac{|T_{c_1} \cap T_{c_2}|}{|T_{c_1} \cup T_{c_2}|} \quad (14)$$

with $|T_{c_i}|$ the number of terms in cluster i . In words, it is the ratio of the number of terms that two clusters share (the intersection of the sets of terms) to the total number of terms in the set of unique elements that spans both word vectors (their union). Note that this measure does not make use of the actual term frequencies, but uses a boolean approximation (where a boolean “True” value means the term is present and its frequency $f > 0$ in the cluster’s term frequency vector).

The usual way of calculating the similarity between clusters is to take the average pairwise similarity. For a newly created cluster, (c_1, c_2) , its average pairwise similarity, $s((c_1, c_2), c_3)$, to a cluster c_3 , is calculated by:

$$s((c_1, c_2), c_3) = \frac{(s(c_1, c_3)|c_1||c_3|) + (s(c_2, c_3)|c_2||c_3|)}{|c_1||c_2||c_3|} \quad (15)$$

with $|c_i|$ the number of documents in cluster i ⁶.

Alternatively, we may store an average word vector for each cluster and use that to calculate the similarities between clusters (and documents) explicitly. This is not unlike creating a hierarchical version of k -means clustering, with $k = 2$ (see also section 5.5). Superficial comparisons between these two approaches suggest that this latter approach performs better than the prior in terms of both resulting cluster purity and CPU time, and thus we chose to use this approach to generate our evaluation data.

2.5.2 Cluster Keywords

Since a word-of-the-day application should revolve around clustering words, not documents, clustering documents alone could be considered of little relevance to this particular project. A more interesting addition, however, which relies on a cluster-based approach, is to find additional keywords for each node in the resulting dendrogram to supplement those already provided for each class. These additional keywords should add more specific terms to the more general terms in the class keyword set, so

⁶Note that equation 15 gives the same result as calculating the average similarity between the document pairs explicitly.

that one can zoom in on smaller topics that are not sufficiently captured by the class set.

Since the keywords can be seen as a set of potential cluster labels, we can view this keyword retrieval problem as a cluster *labelling* problem. Recently, there has been an increasing interest in solving this type of problem, but automated procedures have some way to go before they can reach a same level of accuracy as a human being. Nevertheless, it is interesting to see if we can find useful keywords automatically. There are several alternatives to explore. For instance, Merkl and Rauber have devised methods to label Self Organising Maps, which is interesting in itself, but not useable for a hierarchical structure such as the one generated by HAC. Popescul and Ungar [2000] propose to use a χ^2 test to determine the best label candidates for hierarchical clusters. Treeratpituk & Callan present the DScore model in Treeratpituk and Callan [2006], specifically for hierarchical structures. However, they use trainable weights for their keyword extraction, which are optimised by performing linear regression on an annotated training corpus, which is something we do not have access to.

Instead, we propose a new method based on the well known $TF \times IDF$ scheme (Salton and McGill [1986]). The idea of the $TF \times IDF$ measure is that ordinary term frequency (TF) scores (where a higher frequency is an indication that this word is potentially important) will be weighed according to how many documents contain it (the Inverted Document Score, IDF), so that terms that are relatively rare across the entire corpus will be considered to carry more information. This has the desirable effect of rewarding multiple term occurrences in a document, while still ensuring that very common words that say nothing about a document’s topic (such as most adjectives) are not given overly high word scores due to their high frequency. More particularly, the $TF \times IDF$ score for term t at cluster c , is defined as

$$TFIDF_c(t) = tf_c(t) \cdot \frac{|D|}{df_D(t)} \quad (16)$$

with $tf_c(t)$ the term frequency of t across all documents in c , $|D|$ the total number of documents in the dataset D , and $df_D(t)$ the number of documents in D in which t occurs⁷.

⁷Note that while most of the similarity measures for HAC

Because the cluster structure is hierarchical, we would like to find keywords for clusters that are not too similar to those of their parent or sibling clusters. Therefore, we check the average $TF \times IDF$ vector of a node’s parent⁸ and use this to amend the score of each word at the current node.

Finally, certain types of words or phrases make more useful labels than others as a general rule. For instance, named entities and n -grams or chunked phrases often contain more information than most nouns (but are also usually much more specific), and verbs generally seem to contain less information than nouns, but more than adjectives (e.g., to legislate, to skate). In the current implementation, the best stem for each of the candidate terms is selected (i.e., plural nouns sound more general than singular ones, and gerund verbs sound more like categories than their 3rd person singular forms). This best stem gets assigned a score on the basis of its $TF \times IDF$ score, and this score is then weighed by the term’s POS type score.

Preferably, we would also like the phrases to be relatively short, as long phrases will usually not be to-the-point, and will therefore not make good keywords. On the other hand, we’d rather output “George W. Bush” than just “Bush”. To accomplish this, we modify the POS type score by giving additional credit to phrases of a length close to 3 words⁹.

Because we expect complex phrases to be somewhat more specific, and specific keywords are likely to be more appropriate for small, lower level clusters, the final amendment to this weighing scheme is made by adding the \log_2 of the depth (which is defined as the length of the shortest path to the top of the tree) at which the current node resides. Non-phrase terms are not weighed differently at different depths¹⁰.

can also make good use of the $TF \times IDF$ measure, the Jaccard similarity is insensitive to the actual magnitude of the scores of the terms, as it does not make use of them.

⁸Note that a parent’s average $TF \times IDF$ vector is always the average of the current node’s vector and its only sibling’s, and thus contains implicit information about the sibling’s word distribution.

⁹We do this by calculating the height of a Gaussian distribution with a mean of 3.0 and a standard deviation of 1.0 for a phrase’s length, and multiplying that height by the square of the phrase length.

¹⁰The weights are as follows. Phrases are given a weight of 5.0 plus the scores described above, nouns are weighed 10.0, verbs 5.0, and adjectives 0.001, because they tend to

The final “interestingness” score for a term t at a node s with a parent p is calculated as:

$$s_i(t) = s_s(t) - s_p(t) \quad (17)$$

with $s_s(t)$ the “self” score for t ,

$$s_s(t) = TFIDF(t) \cdot wsm_s(t) \quad (18)$$

and $s_p(t)$ the parent score of t ,

$$s_p(t) = TFIDF(t) \cdot wsm_p(t) \quad (19)$$

where $wsm_c(t)$ is the POS type based word score modifier for t at cluster c .

The m keywords with the highest score are this module’s final output.

Alternatives Before the final keyword selection method was designed, we explored and implemented several alternatives.

First of all, an attempt was made to use keywords by using NLTK.lite’s WordNet¹¹ interface to find common hypernyms or hyponyms, by searching the hypernym sets for a pair of words recursively for common elements. This strategy turned out to be both computationally infeasible (because the number of hypernyms increases exponentially with the depth at which we search) and impractical (because there were rarely any matches before any useable depth).

Secondly we implemented a variant of the DScore model that did not use trainable weights. Unfortunately, the training procedure turns out to be crucial for this procedure, and so this idea was also abandoned.

2.6 Visualisation

The presentation of the selected words is an important aspect of the system since the end-user is most likely unfamiliar with the information structure. We use a design guideline that is well-known in the field of information visualisation: the Visual Information Seeking Mantra, introduced by [Shneiderman](#). This guideline follows three basic steps: “Overview”, “Zoom and filter”, and “Details-on-demand”.

“Overview” refers to an overview of the entire

be very poor descriptors.

¹¹<http://wordnet.princeton.edu/>

collection of information, thus giving the user an abstract view of the available information. “Zoom and filter” refers to the step where the user switches from this abstract view to the more interesting parts, thus filtering out the less interesting parts. The last step, “Details-on-demand”, refers to a point of interaction where the user can ask for desired details.

Applying these steps to our information structure, we get the following requirements. The overview must at least capture the core of our output, that is, the collection of interesting words, without overwhelming the user with information about e.g. the context or relations with other words. Next, we want to bring to light the more interesting subset of words, the *interestingness* of which can be defined by a word’s meaning or a measure based on the word’s frequency. Finally, interaction should be possible, allowing the end user to get information about a word, for instance its context or its location in the cluster structure.

3 Implementation

In this section, the design choices for each of the modules are explained.

3.1 XPM

The XML preprocessing module, or XPM for short, is designed to handle various data formats in which the daily texts are supplied. Its purpose is to present the next module in line with as uniform output as possible. As previously mentioned, it must be extensible so that future data resources can easily be incorporated in the system.

The module is implemented in Python because this language allows for rapid development. It was tested with Python 2.5. It has an object oriented design that allows the inheritance general code. The module provides a general class *DataParser* which every parser should extend. Currently these are *AQUAINTParser* and *LJParser* for respectively the AQUAINT corpus and the LiveJournal RSS feed.

3.1.1 XML handling

The module does not directly parse any XML input. XML is not directly outputted either. Instead,

XPM relies on thoroughly on external libraries for this task, thus greatly reducing development time and chances of bugs in this part of the code.

The XML parsing is done using the SAX parser that is shipped with Python (in `xml.sax`). The major alternative XML parsing technique, namely DOM, was not chosen because of its heavy memory requirements. SAX implements an event-driven parser that signals an event for ever tag that is parsed and for chunks of CDATA inside the elements. This allows the module to serially parse any input data without requiring the entire input file to be in memory at any given time. SAX has the disadvantage that it is a bit slower than DOM, but only when the input files are small enough to fit into main memory.

XML output is handled by a third party package named 4Suite XML, version 1.0.2. (`Ft.Xml.Xslt.XmlWriter`). This package provides a simple interface which allows one to output the XML document while it is being built. Again, it is not necessary to keep all data in memory.

3.1.2 Language testing

If requested, the program may try to filter any documents that do not appear to be English. Perhaps worth noting is that the algorithm that does so is not directly reflected by the equation given in section 2.1.2. To repeat this formula,

$$S_{\text{en}} = \frac{\sum_{w \in L} c(W, w)}{|W|}$$

In words, this says that occurrence of each of the stopwords in the text is counted. However, this requires the program to iterate over all words in the text for each of the words in the stop word list. This turned out to result in a significant slowdown of the program, as can be explained by the complexity of this operation ($O(|L| \times |W|)$). This problem was solved by taking advantage of Python's internal implementation of the *dictionary* datastructure: a set of (key, value) tuples in which the keys are saved in a hash table. By storing the stopwords as keys in a dictionary and looping over the words in the text rather than over the stopwords, the problem was resolved. This also has another tiny advantage: the stopword list may now contain duplicates. For-

mula:

$$S_{\text{en}} = \frac{\sum_{w \in W} o(L, w)}{|W|}$$

where $o(L, w)$ returns 1 if the word w is found in the word list L , and 0 otherwise.

3.2 NLP

The NLP module was implemented in Python. Various modules of NLTK Lite [Loper and Bird \[2002\]](#), a natural language processing toolkit for Python, were used for tokenising, part-of-speech tagging, chunking and stemming. The tokeniser is based on the tokeniser included in NLTK Lite and uses manually defined regular expression templates to break up the input. It was augmented with a post-processing function in order to allow for the specification of context when matching a token. The part-of-speech taggers and backoff mechanism are part of NLTK Lite, the bigram and trigram modules were altered to allow for the storage and retrieval of trained taggers.

3.3 Categorisation

In this Section, we describe the implementation of the categorisation module. In Section 3.3.1, we introduce the categories used for the classification. In Section 3.3.2, we describe how we collected the training data. The implementation of the Naive Bayes classifier is described in Section 3.3.2.

3.3.1 The Categories

The target categories for newspaper text were defined easily: *politics*, *legal*, *economy*, *sports*, *entertainment* and *other*. (The entertainment category is supposed to cover articles on music, movies, theatre plays, celebrities, etc)

Blog texts are in general harder to categorise even for human readers, because they are usually badly structured and not focused on one particular topic. Due to the individual nature of the texts, a huge number of topics is covered, with great differences in topics and writing style between individual authors. For many of the authors, the blog seems very similar to a diary, where they put down their daily experiences and thoughts. For this reason, we were looking for something which would be

common in most people’s daily life, such as sleeping or eating. We considered “eating” an interesting category for the following reasons: First of all, we assumed that people like to talk about eating and the food they eat, because it can be a very (un-)pleasant experience. We also discovered that it was frequently mentioned in the blogs. Secondly, with regard to the system’s goal of extracting interesting words (nouns and named entities), we expected this category to be of practical value. Our hope was that the system would be able to extract what kind of food people are eating. We decided to only have two categories, namely *food* and *other* hoping that this would prevent an expected low level of classification performance.

3.3.2 Training Data

The next problem was to produce a gold standard for training, as not even the AQUAINT articles had categories attached to them. We considered the following options to circumvent our lack of training data:

1. Search for other text material in the web of which we know that they are from the respective domain.
2. Create a (small) hand-labelled corpus
3. Create a (bigger, but slightly erroneous) categorised corpus by labelling the texts on the basis of keywords.

The problem with the first approach is, that it turned out to be difficult to find such a corpus for each category. While there are domain-specific websites available on the web, these would have to be parsed in order to remove all web-site specific code and get out the content. Newsgroups on the other hand, did not offer enough material for all of the categories. Bearing in mind that the classification accuracy would also suffer from a possible difference in writing style between the training corpus and the test corpus, we rejected the idea as too work-intensive. Instead, we created a hand-labelled corpus with 2 texts of each categories, but this amount of training data turned out to be too small. Finally we decided to create the training corpus semi-automatically, based on the occurrence of hand-selected category-specific keywords.

For each of the categories, we selected the 10 documents with the highest number of occurrence of the domain-specific keywords. We did not apply this method to the “other” category, because it did not seem feasible for us to find corresponding keywords.

For the newspaper texts, we found that the accuracy of this gold-standard was actually quite high: In a small test, only two of the 50 documents were found to be labelled incorrectly. The gold-standard used during the experiments only contains one document that was completely misplaced (a question-and-answer text, which did not seem to be a newspaper article in the first place, was falsely assigned the category “politics” although it contained pieces of information about many domains). However, in the categories sports and entertainment, respectively, there was one text which was only related to the category but not a typical text of it. Furthermore, in the category “politics” there was one, and in the category “entertainment” there were even two documents which appeared twice. Unfortunately, the documents labelled with the category “food” in the gold-standard for the blog texts turned out to have only occurrences of “eating activities” where the food that is being eaten is not mentioned. Furthermore (as expected) in none of these texts, the topic of eating is the prevailing theme. Hence, the quality of the gold standard for the blog texts unfortunately is very low.

3.3.3 Implementation of the Naive Bayes Classifier

For the categorisation, a Naive Bayes classifier was implemented (see Section 2.3.1). The features of the classifier were pairs of word stem and frequency. However, since Naive Bayes is unable to establish a relation between the same word occurring in documents with different frequencies we grouped the frequencies into “one occurrence”, “two to four” and “more than four”.

For the feature selection, we used the mutual information measure described in Section 2.3.2. As in Craven et al. [2000] we implemented the feature selection based on words rather than features, thus extracting a vocabulary of 300 stems from which we allowed the algorithm to build features. This number was chosen empirically after several tests with different feature sizes on a preliminary test corpus.

Firstly, this relatively small number enables the algorithm to run at a reasonable speed. Secondly, we discovered that with this number the words that are selected for the classifiers are semantically related to the respective category.

We implemented a semi-supervised learning scheme as described in Section 2.3.3. However, a simple pre-test showed that running the semi-supervised learning for 100 rounds did not change the result. From this we conclude that the amount of training data we were able to produce (see Section 4.3) was sufficient to make semi-supervised learning superfluous. Hence, semi-supervised learning is not used in the final implementation.

3.4 Identification

The information pertaining to every category class is stored separately as input files are parsed,. Depending on a user-defined mode, one of the following operations is performed:

- **Extraction of a reference frequency distribution:** for each category, the frequency distribution of the stems, which is given by the word counts in the input file, is extracted and pickled.
- **Log-likelihood keyword selection:** candidate keywords are selected using the log-likelihood ratio test and the selection criteria defined in section 2.4.2. Subsequently, the keywords with the highest log-likelihood are selected from the list of keyword candidates. The number of selected keywords is 30 per default but can be defined by the user. Finally, these keywords are added to the input information and written to the output.
- **Random keyword selection:** The baseline system selects 30 random stems for each category, and gives them a random score functioning as log-likelihood.

3.5 Clustering

At each level in the cluster hive we keep an average term frequency vector for time efficiency. To implement the HAC algorithm in a somewhat more space-efficient way, sparse word vector representations are used (i.e., all terms that have a term fre-

quency of 0 for this cluster are left out). This requires us to create a lookup hash table that maps terms to vector indices, which comes at the expense of lookup overhead.

A separate inverted index is constructed for the full set of documents, and $TF \times IDF$ vectors are calculated and stored whenever they are first used (and unset again when the cluster in question is altered, to prevent inconsistencies). When comparing vectors, we always iterate over the vector with the fewest entries to save time.

The HAC algorithm itself first runs an entire comparison round through the entire document set, calculating and storing the similarity of each pair of documents in a large priority queue structure, which is kept ordered by a binary tree insertion procedure. The resulting queue contains a long list of all clusterable pairs, ordered by their similarity score. With this structure in place, it will perform iterations as described in algorithm 3, where at every iteration the pair to be clustered is simply the top entry in the priority queue. After each merge of two winning clusters (c_1 and c_2), all entries containing one of the winning clusters are removed from the priority queue. The similarity of each cluster in the unclustered set to the new cluster is then calculated, and resulting merging-candidates are again inserted into the queue on the basis of their similarity score.

In spite of these, and other optimisations, HAC has a theoretical $O(N^2)$ time complexity, and so it is not well suited for clustering very large numbers of documents. The fixed-set categorisation procedure therefore serves a second function, as a filter to keep clustering time and space demands manageable.

3.6 Visualisation

For having a natural environment to display the information we are more or less bound to create an online solution because thats also the origin of our input. From there, we chose a website for being a well-understood medium with numerous design possibilities. This gives us also the possibility to redirect the user to the actual source of the information on the internet.

The step from the internal clustered document structure with assigned words to HTML is non-



Figure 4: Tag-cloud representation of keywords.

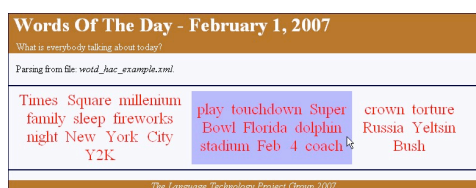


Figure 5: Click-through representation of keywords.

trivial in the sense that we cannot present all information at once without discarding the design guideline mentioned in section refThVis. The guideline told us to present an overview of the information first. A good method to accomplish this are *tag clouds* (see [Godwin-Jones](#) for a reference). Tag clouds are a relatively new but already widely-used (see *Flickr* and *del.icio.us*) method to display a collection of terms with different weights. A tag cloud representation of our system’s output is shown in Figure 4. The words are presented with a size reflecting their weight, which in our case is the measure of interestingness defined by the preceding identification modules. This brings us a step forward in the design guideline, as the importance of “Words of the day” is clearly presented to the end user.

A second method to present “Words of the day” is to exploit the knowledge we have about the word’s category. By grouping them around an explicitly called label we can focus the user to a group of semantically related words. This focus can also be applied more implicitly in a tag cloud as in figure reffig:visTagCloud, where words are grouped using colourisation. This *category view* and the tag cloud presentation form can off course be combined.

Finally, the user can get details-on-demand in two ways. The first is a presentation of a word’s context obtained by the user when she clicks on a word in a tag cloud or category view. The user can also get more detailed information by using the third view called *ClickThrough*. The ClickThrough view (as shown in Figure 5) allows the user to explore the cluster structure. It starts by displaying a per class word collection, but the user can now explore one of these categories by clicking on a group of words instead of an individual word. By clicking groups the user gets recursively new collections representing important words from the child clusters, that is, important words from all documents that are below that cluster in the tree structure. As the cluster-tree is binary, the user can always choose between two groups and will subsequently be redirected to a document or to a collection of two sub-clusters.

4 Evaluation

In the following sections, we will explain the methodology and present results for the evaluation of each module.

4.1 Methodology

The XPM module was not evaluated: in fact, the output files it produced proved for themselves that the module performed properly.

Each intermediary module produced two output files, one corresponding to its normal output and one to its baseline, for both the *LiveJournal* and the *AQUAINT* test corpora. In every case, the modules used the normal output of their predecessor. The baseline and the normal output files, which had to be indistinguishable from each other, were then provided to the person in charge with the next module for evaluation. Scores were attributed differently depending on each module. Sections 4.2 to 4.5 present module-dependent methods for evaluation, as well as the evaluation results.

By the nature of its output, the evaluation of the visualisation module was more delicate. Section 4.7 presents evaluation methods for this module.

4.2 NLP

In the following sections we will describe the methods and results for several individual components of the NLP preprocessing module.

4.2.1 Tokenising

35 sentences from each domain were randomly picked and their tokenised versions were manually evaluated. For the news data, a tokenising accuracy of 88.75% was found, while the result for the blog data was almost ten percent lower at 77.14%. The most common errors in tokenising the news data were abbreviations ending in a full stop and followed by either numeric or uppercase characters as in “Nov. 26” and “St. George”. In the blog data, all errors except for an instance of “sh*t” that was tokenised into “sh” and “t”, were caused by sentences beginning in lowercase characters, which lead to the merging of several tokens or even sentences into one token.

4.2.2 Tagging

The performance of the tagger was manually evaluated using a random sample of ten sentences each from the news and blog data.

Only correctly tokenised sentences were used. When it was unclear what the correct tag for a token was, the token was not counted. This was the case for only about two percent of the tokens.

The tagger used in the NLP module was evaluated together with a Bigram tagger (with Unigram, Affix and default tagger backoff), a Unigram tagger (with Affix and default tagger backoff), an Affix tagger (with a default tagger as backoff) and a Brill tagger.

Table 4.2.2 shows the accuracies found.

The numbers are relatively similar to the accuracies found for the test on the Brown corpus in section 2.2.2 with the Trigram tagger winning out only by a small percentage and the Affix tagger performing the worst. However, the accuracies lie about five percent over the scores found in the evaluation on the reviews section of the Brown corpus.

This could be due to a non-representative, atypical sample, errors in the manual evaluation or the fact that the taggers were previously evaluated on the “Reviews” section which contains a high number of proper nouns (e.g. artists’ names and titles

Table 2: Accuracy of taggers on news data

Tagger	Accuracy
Affix	0.354
Unigram	0.864
Bigram	0.877
Trigram	0.88
Brill	0.877

of their works) which tend to get tagged wrongly relatively often, while the texts in this experiment, just like the training data, were newspaper reports.

Table 4.2.2 shows errors that the Trigram tagger made, sorted by frequency in descending order. The column “Influence” indicates whether the error in question has any consequence for later processes in the NLP module, “-” indicates that this is not the case (e.g. when a capitalised word does not get tagged as a named entity, the named entity recogniser will correct this), “c” means that an error in chunking might be introduced (when the wrong label does not change whether the word is extracted but might lead to faulty chunks) and “e” means that the error leads to mistakes in the extraction¹².

The tagging accuracy results for the blog data are shown in table 4.2.2, which again are very similar to the results previously found. However, several sentences were discarded during the random selection process since they were incorrectly tokenised and the errors were found to be distributed less uniformly than in the news data with sentences often being either tagged completely without error or with only few correct tags.

4.2.3 Module

The performance of the module was tested and against a baseline system which tokenised and counted like the standard module but extracted only random words and blindly evaluated.

The standard system was correctly identified from the output both for the blog and news domain.

It was found that the baseline systems failed both in extracting the correct words, the content words

¹² “e” also implies “c”, since errors in extraction can always lead to errors in chunking.

Table 3: Errors found and their influence on the performance of the NLP module

Found	Correct	Example	Influence
nn	np	“Ernesto”	-
nn	nns	“guerillas”	-
np-tl	np	“Mexico”	-
nn	jj	“video-taped”	c
nn	cd	“1974”	e
vbn	jj	“armed”	c
rb	ql	“even”	-
nn	nn\$	“bug’s”	-
vbg	nn	“programming”	c
nn	fw-in+at	“del”	e

Table 4: Accuracy of taggers on blog data

Tagger	Accuracy
Affix	0.35
Unigram	0.864
Bigram	0.867
Trigram	0.867
Brill	0.853

that characterise the text and in not extracting unwanted words like determiners and conjunctions, while at least for the news domain, the standard system “only contains words that are related to the activities and more detailed circumstances of the activities”¹³.

However, the blog data for the standard system was found to contain “nonsense” phrases like “ages ago. announcement”. This ties in with the results found in the evaluation of the tokeniser and taggers, namely that tokenising does not perform very well on the blog data.

Every non-properly tokenised “nonsense” phrase is tagged as “nn” by the part-of-speech taggers, since no fitting tag can be found and the lowest-level tagger, the default tagger which always assigns the tag “nn” is called. Consequently, all of these phrases are added to the output.

Faulty tokenising that lead to “nonsense” phrases in the output was also observed in the news

¹³Quote from the evaluator.

Table 5: The results of the human-performed evaluation of the categorisation module.

Category	Precision (%)	Baseline (%)
Food	12.00	8.00
Politics	65.45	N/A
Sports	71.86	N/A
Entertainment	56.62	12.61
Economy	73.81	10.52
Legal	50.00	9.60

data of the standard system, but to a much smaller extent.

4.3 Categorisation

In this section we describe the evaluation of the categorization module.

4.3.1 Methods

The system was tested against a baseline of randomly assigned categories. The precision values were calculated based on a human evaluation of the categorisation result. For the blog corpus, the human evaluator examined the first 100 documents of each category. For the newspaper corpus, all documents classified as “economy”, “legal” or “entertainment” (a total of 169 documents after categorisation, 388 documents in the baseline) were examined. Due to time constraints, the baseline for the categories “sports” and “politics” were not evaluated in the baseline. However, since the assignment for the baseline was random, the achieved precision should be comparable to the other categories.

4.3.2 Results

The results for both corpora are presented in table 5. While the result of the newspaper text categorisation seems quite satisfactory, the result for the blogs is devastating. In Section 5.3, we will discuss the reasons and possible solutions.

4.4 Identification

In this section we describe the evaluation methods and results for the word identification module.

4.4.1 Method

Each of the 30 keywords in the four output files were attributed a score in the following way:

- keywords get a score of 0 if they do not fit in the category, or if they seem to be common in the domain specified by that category,
- keywords that fit the category and seem to be salient in this category get a score of 1

The precision for each category in each file was then computed using the cumulative scores.

4.4.2 Results

The evaluation results for the *LiveJournal* and the *AQUAINT* test corpora are given in Table 6. In the case of the *LiveJournal* data, the module

Table 6: *LiveJournal* and *AQUAINT* evaluation results

Category	Precision (%)	Baseline (%)
food	3	8
other	n.a.	11
politics	56	17
sports	50	33
entertainment	47	0
economy	33	30
legal	n.a.	17

found 30 keywords in the category ‘food’. Note that a considerable number of these keywords were first names. In the case of the *AQUAINT* data, the keywords were distributed more evenly, across four categories.

4.5 Clustering

The next two subsections describe the evaluation method and results for the clustering module.

4.5.1 Method

For evaluation purposes, cluster purity was examined at several different ‘cuts’ through the resulting cluster set. A cut simply consists of the set

Table 7: Measured average cluster purity at different depths D (BL stands for Baseline, AQ for *AQUAINT*, LJ for *LiveJournal*).

D	BL AQ	BL LJ	HAC AQ	HAC LJ
3	0.72	0.47	1.0	0.17
5	0.91	0.4	0.88	0.29
7	0.86	0.33	0.82	0.39

of all subcluster subtrees at a certain depth in the dendrogram. The results were averaged over a total number of 9 randomly picked clusters (3 per depth). We expect the bottom clusters to contain few interesting groupings because they are too small. The top of the cluster contains all documents and its purity is therefore the same as that of the class it is subject to. For these reasons we examined only clusters at depths 3, 5 and 7 (with 1 being the top supercluster, and 9 being the ideal tree height for a cluster of 500 articles). At each of these depths the perceived cluster topic was determined on the basis of the common topic of the majority of articles, and the ratio of the size of this majority to the total number of documents in the cluster was used as the cluster purity score. It was expected that the top cluster levels would contain relatively many impurities, since the merges of the least similar documents occur near the end of the clustering run.

This setup was then compared to a baseline system that merged clusters at random, in a blind test.

4.5.2 Results

Table 7 shows the average cluster purity for the HAC algorithm and the baseline system at different depths. Much to our surprise, the HAC algorithm did not breach the baseline by any significant amount. In fact, it seems to do worse than the baseline in all but a single case.

However, closer examination reveals that the clustering as performed by HAC is far from random. Consider the following partial example output from the largest cluster in the ‘politics’ class:

- 1 MIDEAST TALKS RESUME IN MOOD OF GUARDED OPTIMISM
- 2 SECOND ROUND OF ISRAELI-SYRIAN PEACE TALKS OPEN

3 (untitled): SHEPERDSTOWN, West Va.
President Clinton met (...) with Israeli and
Syrian negotiators (...)

4 THE HARD WORK BEGINS MONDAY IN
SYRIA-ISRAEL TALKS

So far, so good. But then, the cluster topic takes a
radical turn:

5 BRADLEY TO STAY THE COURSE OF
CAMPAIGN TRAIL IN N.H.

6 BRADLEY, GORE KEEP UP THE BEAT

7 BRADLEY AND GORE RENEW BATTLE
IN WARM-UP FOR DEBATES

The cluster then changes topics a number of times, ranging from Russian politics to the race between George W. Bush and Al Gore. Similarly, the first (and by far the largest) of eight clusters under the “sports” class (at depth 3 from the top of the hive) starts with approximately 40 articles about American football, and only then moves on to other sports (and some random articles that had not been correctly classified by the prior classification).

Upon visualising the clusters, it becomes clear that the algorithm has generated a large branch, underneath which the more accurate, smaller sub-clusters are hung. This is shown in figure 6. This is a recurring phenomenon which we believe to be due to the Jaccard similarity measure’s lack of resistance to noise. The clustering of the small sub-clusters is quite good, but as the clusters grow larger, their similarity becomes harder to determine (the denominator in equation 14 becomes smaller, while the size of the divisor increases), leading the algorithm to merge the largest clusters first if there are no clearly similar candidates left, and thereby creating a large single branch when merging the final few clusters. This also explains why the algorithm seemed to work well on the small dataset on which the similarity measures were tested: for a small dataset, the branching is not yet deep enough to notice the negative effect.

It should also be stressed again that the *Live-Journal* data was extremely difficult to categorise, even for a human being. In one instance, the clustering grouped blogs with pieces of text about “trouble sleeping”, “I’m sleepy”, and the movie “Sleepless in Seattle”, which gives an impression

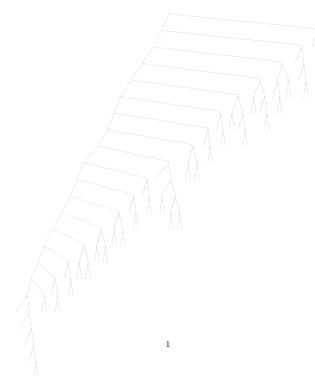


Figure 6: The cluster hive generated by HAC using the Jaccard similarity measure, for the “Politics” class.

of how difficult it is to find the right sub categorisation for these posts. The fact that all authors have their own writing style only increases the size of the problem.

Finally, it should be pointed out that the evaluation method is a very complicated procedure. Particularly for large clusters it is hard to identify the best approximate topic, and thus measuring the cluster purity accurately is very difficult.

4.6 Cluster Keywords

The next two subsections describe the evaluation method and results respectively for the cluster keyword module.

4.6.1 Method

For each cluster the m highest scoring keywords were returned (with m a cutoff point, here set to 20). For a total of 9 randomly picked clusters (3 per depth), it was then noted how many of these keywords seemed to be of interest. Although “interestingness” is a subjective evaluation, the system is, after all, an “interesting word application”, and should be evaluated on its ability to produce the desired output.

Table 8: Average keyword usefulness at different depths D for $m = 20$ (BL stands for Baseline, AQ for *AQUAINT*, LJ for *LiveJournal*).

D	BL AQ	BL LJ	HAC AQ	HAC LJ
3	0.23	0.1	0.79	0.35
5	0.3	0.2	0.64	0.3
7	0.27	0.23	0.78	0.33
AVG	0.27	0.18	0.74	0.33

Table 9: First five keywords found for the “politics” class of the *AQUAINT* corpus.

POS tag	Word form
NP	Putin
NP	Bradley
NN	campaign
NP	Russia
NP	Yeltsin

The ratings for the cluster keyword system were compared in a blind test to a baseline system, which produces output by randomly selecting words from a cluster’s $TF \times IDF$ vector.

4.6.2 Results

Looking at table 8, it is immediately apparent that our keyword algorithm outperforms the baseline system in all cases. An example of the *AQUAINT* data output is shown in table 9.

Although the results for the *LiveJournal* data are much lower than those for the *AQUAINT* corpus, this is actually also the case for the random baseline system, which suggests that there are simply fewer words in the blogs that may be considered of interest than there are in the newspapers. In many cases, the returned keywords concern first names of personal acquaintances of the blog poster, or movie stars or character names. These words are assigned a high word score by the POS tag weighing scheme. An example is shown below in table 10.

4.7 Visualisation

There are generally three methods to evaluate a presentation like ours. The two more interesting of them turned out to be too ambitious for a four week project, but for completeness I will discuss

Table 10: First five keywords found for a typical cluster the *LiveJournal* corpus.

POS tag	Word form
NP	Harry
NP\$	Author’s
NP	Tracy
NPC	Author’s Summary
NP	Elijah

them superficially.

One interesting method to test at least relative quality of the different presentations is empirical. With this method we should actually make the whole system with it’s different views publicly available. By logging usage data about which presentation is used more often we can find out what presentation method people are preferring. This is not an absolute measure of quality of the methods because different people have different information needs and the presentation methods offer different functionality.

A second evaluation method that does distinguish between different information needs is an experimental one. The experiment is set up to have a number of tasks, which range over the information needs that we are interested in, and a number of subjects. Each subject is asked to perform the tasks with a presentation method. By measuring reaction times we have information to identify the best presentation method for each type of task.

The final evaluation method, the one that we used, is a subjective explanation of why a particular presentation method is preferred above another. Results of these methods are off course hard to compare with others, but in some way the reviews should approximate the results of the method I described first.

People of our project group seemed to have a preference for the tag cloud based presentation because of the natural emphasising of important words by using different font sizes. The preference of the category view depended largely on the performance of the categorisation module, because if non-related words are still grouped the user gets confused. The same goes for the ClickThrough method, regarding the performance of the clustering. Also the limitation of the clustering module to generate only binary trees does not improve the

usability because the human perception bandwidth is large enough to take on more than two clusters at once. So we think the presentation method is not reaching its potential worth.

5 Discussion

In this section, the results found in the last section will be discussed.

5.1 XML preprocessing

The XML preprocessing module is far from perfect. For example, the language score assigned to a text is not always a good indicator. The earlier mentioned problem with bilingual texts is an indication of that. Solving this problem would require a different approach, for example looking at the structure of the words in the text to see if they are English-like. Non-English sentences could then be removed from bilingual text while non-English phrases would probably be preserved, which is important for an application such as Word of the Day.

5.2 NLP

Overall, the results show that the NLP module fulfils its goal of extracting the “potentially interesting” content words and noun phrases and performs noticeably better than a baseline system. However, certain parts of the module yield better results than others and the performance on the blog data is not as good as that on the news data.

The evaluation of the taggers showed that the tagger used in the system is not only the best when assessed on the Brown corpus data, but also performs best in practise.

The analysis of the errors observed in the news data showed that, while errors that negatively influence the output of the module in terms of what is extracted and counted do occur, this is not the case for the most frequent errors, some of which are corrected by the named entity recogniser.

While the tokenising step yielded good results overall, its performance on the blog data stayed behind that on the news data. This can be explained by the fact that tokenising relies heavily on capitalisation, e.g. when disambiguating between abbreviations and sentence endings. Many authors of

the blog data however do not use standard capitalisation and consequently tokenising becomes problematic.

As the capitalisation used in blogs is not only non-standard but also not consistent across (and even within) authors, the tokeniser cannot easily be adapted to fit the different capitalisation style and it is not clear how this problem could be solved.

Tokenising for news data yields far better results, but not all full stop ambiguities are resolved correctly and the tokenising could be improved for example by automatically constructing a list of known abbreviations during processing, or by employing a manually created list (Gregory Grefenstette [1994]).

The blog data also differs from the news data – which is closer to the kind of data that the tagger was trained on and that the methods used in this module are usually applied to – in other respects: The texts are less formal which means that the vocabulary is different from that used in standard written published text.

This is especially problematic because the Brown corpus which the taggers are trained on not only does not include comparable texts, but also was created more than 45 years ago and does not contain words that are frequent today but not common (or even existent) when the corpus was created. This of course also concerns the news data, but to a lesser degree.

Relatively frequent words and contractions like “imma”, “huck” , “she’s” for “she has” and “fuck” are tagged incorrectly for these reasons. The part-of-speech tagger employs smoothing in the form of the Affix tagger, which only slightly improves the tagging of unknown words.

Using a more recent corpus that included colloquial English is an obvious but seemingly infeasible solution to this problem.

Yet another thing that impairs the performance of the tagger on the blog data is that words are often written in all uppercase letters for stress and thus are not recognised as words that were in the training set (e.g. “IT/nn, WAS/nn, SO/nn, MUCH/nn, FUN/nn, ./.”). A possible solution would be to extend the tagger so, if no tag is found for an all-uppercase word using the n -gram taggers, it tries to tag the lowercase version of the word before falling back on the default tagger.

To summarise, the NLP module performs well

overall, but it could be improved in certain points. The tools available (and used in the module) are not well-suited to process text that uses current, colloquial vocabulary and inconsistent, non-standard capitalization.

5.3 Categorisation

While the result of the categorisation of newspaper texts is quite satisfactory, it could probably be improved if redundantly occurring articles and documents which are not newspaper articles in the first place would be filtered out. The latter seems feasible to be performed for example by Bayesian classification, as there seems to be a fixed number of types of these documents (such as the “question-answer-text documents” or article overviews). Having examined several such documents, we believe it should not be difficult to build a recogniser for them once the types have been identified, because they have a clear structure which differs from the usual newspaper articles. Furthermore, we realised a necessity of a sixth category (beside the ones introduced in Section 3.3.1) for a number of newspaper articles are dealing with science and technology.

The categorisation of the blog texts was a complete failure. This is not surprising when taking a closer look at these texts. The Bayesian classifier tries to categorise a text according to its word distribution. However, we found out that “food” is not the main theme of the texts. Therefore, a correlation between the words of documents in the category cannot be expected. We believe that blog texts require a different approach to categorisation. We propose the following two approaches:

The first option is to define different kinds of categories which actually cover the main theme of the texts. Due to the heterogeneous nature of blogs, this is difficult, and we believe it can only be done on a very rough level. One example would be a category which covers diary-like descriptions of the events of a day, which are not centred around a special event in particular. If a text is centred around a particular event, it usually contains an introductory sentence. That is because blog texts – in contrast to newspaper articles – often lack a title, and because of the immense heterogeneity of blog texts. Since discourse parsers have already been successfully used to evaluate the structure of essays in language tests [Burstein et al. \[2003\]](#), they could also be

useful to distinguish between “focused” and “unfocused” texts.

The second option deals with blog texts which are not centred around one specific topic. We believe it will be necessary to find a way of splitting the text according to the individual topics they cover. For the diary-like descriptions, this split could be attempted using a list of time expressions.

The lesson we learnt from the result of the evaluation is that the degree of difficulty of the task of text categorisation depends on both the text and the categories. If the text is well-written and well-structured and focused on a particular topic, it is easier to deal with. However, much depends also on the selection of categories, which have to be well-defined and particularly suited for the text. We believe that this explains the difference in the result of the classification on our two corpora.

5.4 Identification

The poor performance of the module with regard to the *LiveJournal* data reflects the performance of the categorisation module with blogs. In fact, a close look at the data clearly shows that blogs are much more often multi-topical or atypical than otherwise.

The following list presents the module’s output for the *AQUAINT* data. For each category, the keywords are given by descending log-likelihood.

- politics: ‘Y2K’, ‘power’, ‘torture’, ‘Yeltsin’, ‘Russia’, ‘calendar’, ‘camps’, ‘Friday’, ‘crown’
- sports: ‘Giants’, ‘play’
- entertainment: ‘millenium’, ‘Times Square’, ‘midnight’, ‘Times’, ‘Square’, ‘night’, ‘Y2K’, ‘family’, ‘Year’s’, ‘password’, ‘shows’, ‘Green’, ‘Maguire’, ‘New Year’s’, ‘sleep’, ‘New’
- economy: ‘merger’, ‘Sunday’, ‘options’

We could object to verbs and common nouns appearing as keywords in our output. We consider however that simply filtering them out would not be a correct solution. A word as ‘Sunday’ might indeed very well be salient in the domain of economy. A detailed verification of the module showed that these words have been selected because they actually *are* more salient in our test data than in our reference data. We conclude that the apparently

poor performance of the identification module is a result of the similarity between our reference and test corpora, for the *AQUAINT* data as well as for the *LiveJournal* data. The similarity between these sets is illustrated in the case of the category ‘politics’ by Figure 7¹⁴.

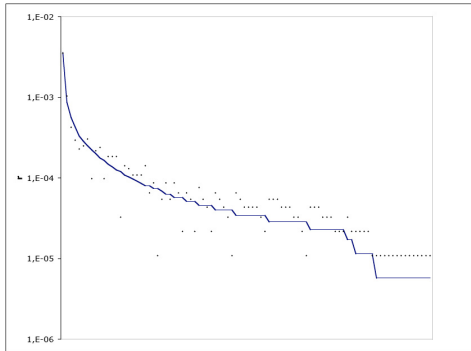


Figure 7: Extract in the test cor of the frequency distribution for the reference (line) and test (dots) corpora in the category ‘politics’

5.5 Clustering

Unfortunately the results mentioned in section 4.5.2 mean we have to discard the Jaccard measure as an appropriate measure of similarity for use with HAC. However, there are many other measures to choose from, and there are many hierarchical clustering algorithms besides HAC. The binary nature of this algorithm may not make it the best possible choice, because the subdivision within categories is rarely one in which a sensible fifty-fifty split can be made. The unbalanced tree structure in figure 6 also suggests that the binary nature of HAC is indeed problematic, especially when using a noise-sensitive similarity measure, so increasing the number of possible child nodes may make a world of difference.

A more appropriate algorithm might be a hierarchical k -Means clustering scheme (Chakrabarti [2002]). Such a scheme would start with a nor-

¹⁴considering the size of the data, we looked at one in 1000 pairs (reference frequency, observed frequency) for the 75% of the test-corpus stems with the lowest reference frequency, and at one in 100 pairs in the following 25%

mal k -Means clustering iteration¹⁵ over the dataset, outputting k clusters for a class. It would then proceed by subdividing each of these clusters in the same way, and continue this process until the number of subclusters under the second lowest level of clusters is lower than some threshold. The downside to such an algorithm is that we would have to find a best value for k .

5.5.1 Cluster keywords

Although the keyword finding algorithm breached the baseline by quite a bit, there are still some remarks that need to be made.

First of all, the keywords for child and parent nodes are not always that different. This is not really surprising, and perhaps even desirable if the child nodes are quite similar to their siblings (and thus also to their parents). Still, it would be nicer to return each keyword only once, in the most appropriate place, to avoid redundancy.

The same thing can be said for the relation between class and cluster keywords. We stated earlier that the goal should be to expand upon the class keywords, and make them more specific. Instead, many of the keywords overlap, because there is currently no relationship between the two modules.

Finally, the returned keywords are not always as complete as they could be. In the example in table 9, the names “Yeltsin” and “Putin” should have been returned as “Boris Yeltsin” and “Vladimir Putin” whenever available.

5.6 Visualisation

The current arsenal of presentation methods is thought to be enough for the capability of the current system. We also think the quality of the presentation methods will increase whenever the performance of the whole system increases.

As of new features and developments, it might be needed to introduce new presentation methods. An obvious example is the incorporation of time based

¹⁵The k -Means algorithm, in its simplest form, starts out by dividing a dataset of documents randomly over k clusters, and then performs a series of iterations, during which it proceeds by assigning each document in the dataset to the cluster with the closest mean. The cluster centroids are then updated for successive iterations, and the algorithm continues with this procedure until no further significant change in the cluster means was found during the last cycle.

information and correlation of word occurrences. This could be presented with a new method like an area graph that displays the changes of correlations between word occurrences over time to give information about the rise and fall of words.

6 Conclusion

We have presented a word of the day system which selects “interesting” words and presents them visually. We consider words to be “interesting” only if they are content words or noun phrases which occur in a given corpus more often than they usually do in texts from the same domain. This concept of “domain-dependent interestingness” distinguishes our system from Eiken et al. [2006]. Furthermore, we perform a hierarchical clustering of the text documents, which allows us to present the output in a more structured way. Finally, our system pursues a hybrid approach which combines pre-defined categories at the top-level with dynamic categories (sub-clustering) whereas Eiken et al. [2006] only pursue both approaches separately.

A human-performed evaluation of our system showed that it performs satisfactory on newspaper texts from the *AQUAINT* corpus but very poorly on *LiveJournal* blog texts. From this we conclude that the performance of our system is dependent on the input data to a higher degree than was initially expected. Blog texts are harder to categorise and show less coherence. Furthermore, the natural language processing tools we apply are not suited for the language used in blogs, as it differs considerably from the kind of text they have been trained on and rely on properties that the blog data do to possess. The difference in writing styles between different authors adds a further complication. The difference in performance of our system on the different text types is surprising considering that our method only relies on shallow (rather than deep) linguistic processing.

References

- Klein Ewan Bird, Steven and Edward Loper. Nltk tutorial: Introduction to natural language processing.
- J. Burstein, M. Chodorow, and C. Leacock. Criterion: Online essay evaluation: An application for automated evaluation of student essays. In *Proceedings of the Fifteenth Annual Conference on Innovative Applications of Artificial Intelligence*, Acapulco, Mexico, August 2003.
- Soumen Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*, pages 84–89. Morgan-Kaufman, 2002. ISBN ISBN 1-55860-754-4. URL <http://www.cse.iitb.ac.in/~soumen/mining-the-web/>.
- Kenneth Church and William A. Gale. Inverse document frequency (idf): A measure of deviations from poisson. In *Proceedings of the ACL Third Workshop on Very Large Corpora*, pages 121–130, 1995.
- M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to construct knowledge bases from the world wide web. *Artificial Intelligence*, 118(1–2):69–113, 2000.
- Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, Vol. 19(1):61–74, 1993.
- U. C. Eiken, A. T. Liseth, H. F. Witschel, M. Richter, and C. Biemann. Ord i dag: Mining norwegian daily newswire. In *Proceedings of the FinTAL*, Turku, Finland, 2006.
- R. Godwin-Jones. Tag clouds in the blogosphere: Electronic literacy and social networking. <http://llt.msu.edu/vol10num2/emerging/default.html>.
- Pasi Tapanainen Gregory Grefenstette. What is a word, what is a sentence? problems of tokenization. In *Proceedings of the 3rd Conference on Computational Lexicography and Text Research – COMPLEX’94*, pages 79–87, 1994.
- D. Jurafsky and J. Martin. *Speech and Language Processing*. Prentice-Hall, 2000.
- Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pages 62–69, 2002.

- C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*, chapter Text Categorization. MIT Press, 2000.
- D. Merkl and A. Rauber. Automatic labeling of self-organizing maps for information retrieval. URL citeseer.ist.psu.edu/265548.html.
- T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
- K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.
- A. Popescul and L. Ungar. Automatic labeling of document clusters, 2000. URL citeseer.ist.psu.edu/popescul00automatic.html.
- Lance Ramshaw and Mitchell Marcus. Text chunking using transformation-based learning. In *Proceedings of the ACL Third Workshop on Very Large Corpora*, pages 82–94, 1995.
- Eddie Rasmussen. Clustering algorithms. pages 419–442, 1992.
- Paul Rayson and Roger Garside. Comparing corpora using frequency profiling. In *Proceedings of the workshop on Comparing Corpora*, pages 1–6, 2000.
- Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986. ISBN 0070544840.
- Satoshi Sekine. Named entity: History and future.
- B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. Proceedings of the 1996 IEEE Symposium on Visual Languages.
- Pucktada Treeratpituk and Jamie Callan. Automatically labeling hierarchical clusters. In *dg.o '06: Proceedings of the 2006 international conference on Digital government research*, pages 167–176, New York, NY, USA, 2006. ACM Press. doi: <http://doi.acm.org/10.1145/1146598.1146650>.