# Machine learning techniques for document selection

**3rd year dissertation**

**Department of Computer Science**

**University of Sheffield**

Author: Leon Derczynski (aca00lad@shef.ac.uk)

Supervisor: Dr Amanda Sharkey (A.Sharkey@dcs.shef.ac.uk)

Module: COM3021

25th April, 2006

This report is submitted in partial fulfilment of the requirement for the degree of Master of Computing with Honours in Computer Science by Leon Derczynski

All sentences or passages quoted in this dissertation from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations which are not the work of the author of this dissertation have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this dissertation and the degree examination as a whole.

Name:

Signature:

Date:

## Abstract

As humans use information retrieval systems, a wealth of data is generated. The problem of determining documents relevant to a query can be learned instead of developing a blind information retrieval system. Feedback on relevance can be used as training data for machine learning algorithms, with the end goal of creating a system reliant on human relevance judgements instead of conventional information retrieval methods.

This project will review approaches used for returning search results over a collection of independent documents, evaluation of information retrieval systems, and teaching machine learning algorithms to classify documents given a natural language query.

The performance of a set of machine learning algorithms at classifying relevant documents was examined. Some exploratory work on optimising problem representations is undertaken, with varying degrees of success. Other approaches for gathering data and classifying documents to aid humans in search are also discussed.

## Acknowledgements

I would like to give special thanks to my supervisor, Dr Amanda Sharkey, for pointing me in interesting directions, providing small tips that turn into useful resources, and constant tolerance of my timekeeping.

I would also like to thank Summit Media for their investment in my studies. The support and resources have been a great boon.

None of this would have been possible without the early information retrieval work of Gerard Salton and JJ Rocchio; and this project would never have been inspired had it not been for the success and open attitude of Larry Page and Sergey Brin with Google.

Finally, I would like to give my thanks to my family and friends for still talking to me despite having come off worse in decisions between spending time with them or in the library, and all their words of encouragement.

## Contents

i. Title page
ii. Signed declaration
iii. Abstract
iv. Acknowledgements
v. Contents
ix. List of tables, formulae and figures

1. Introduction 3

2. Literature Review 3

2.1 Information retrieval 3
2.1.1 IR Models 3
2.1.2 Stemming 4
2.1.3 Thesauri 5
2.1.4 Latent Semantic Analysis 6
2.1.5 Term Frequency 6
2.1.6 Stopwords 7
2.1.7 Indexing 8
2.1.8 Document formatting 10
2.1.9 Existing IR systems 10
2.1.10 N-gram analysis 11

2.2 IR system evaluation 11
2.2.1 Recall and Precision 11
2.2.2 F-Measure 12
2.2.3 Rank of selected document 12
2.2.4 Average precision 12
2.2.5 Kendall's tau 13

2.3 Human interaction with IR systems 14
2.3.1 Query refinement 14
2.3.2 Clickthrough analysis 14
2.3.3 Issues with implicit feedback 15
2.3.4 Ranking effect 16

2.4 Machine Learning 16
2.4.1 Decision trees 16
2.4.2 Neural nets 17
2.4.3 Bayesian learning 19
2.4.4 Instance based learning 19
2.4.5 Training and testing machine learning algorithms 19
2.4.6 Overfitting 20

3. Requirements and analysis 21

3.1 Aims and objectives 21

3.2 Which ML algorithms to use 21

3.3 Corpus usage 22
3.3.1 Collection notes 23

3.4 IR techniques used 23

3.5 Issues with using a classification approach 23

3.6 Multi-phrase queries 24

3.7 Evaluation of ML methods 24

4. Design

4.1 Processing reference collections 25
4.1.1 Document file 25
4.1.2 Query file 26
4.1.3 Query / relevant documents file 26

4.2 Word features 26

4.3 Describing positive training examples 27

4.4 Document and query based features 28

4.5 Use of non-text metrics 28

4.6 Base accuracy 28

4.7 Experiments run 29
4.7.1 Basic comparison 30
4.7.2 Boolean reduction 30
4.7.3 Adding metadata 30
4.7.4 Exclude empty documents 31
4.7.5 Vary hidden units 31
4.7.6 Vary learning rate and training time 31
4.7.7 Compare training set size with learning ease 31

5. Implementation and testing 32

5.1 Presenting the problem 32

5.2 Reference corpus processing 32

5.3 Negative examples 32

5.4 No-relationship flag 33

5.5 Notes 33

6. Results 34

6.1 Findings 34
6.1.1 Basic comparison 34
6.1.2 Boolean reduction 36
6.1.3 Adding metadata 40
6.1.4 Exclude empty documents 41
6.1.5 Vary hidden units 42
6.1.6 Vary learning rate and training time 44
6.1.7 Compare training set size with learning ease 45

6.2 Further work 46
6.2.1 Stemming 46
6.2.2 Practical study 46
6.2.3 Experiment ideas 46
6.2.4 Study practises 47

6.2.5 Handling prior knowledge 48

7. Conclusion 50

Bibliography 51

Appendices

Appendix A – Full results 53

Appendix B – Cranfield rankings 59

Appendix C – ARFF file format 60

Appendix D – Example entries from collections 66

## List of tables, formulae and figures

### Tables

Table 2.1 - Search term frequency and corresponding keyword density

Table 2.2 - Frequency of terms across a corpus, and the number of documents that use the term

Table 2.3 - Some stems and words derived from those stems

Table 2.4 - A thesaurus entry

Table 2.5 - A simple example reverse index

Table 2.6 - A reverse index allowing for tf.idf to be calculated

Table 2.7 - Uninterpolated average precision

Table 2.8 - Concordant and discordant pairs by Kendall's tau measure

Table 3.1 - Reference collections

Table 4.1 - Fields available in reference collections

Table 4.2 - Document features

Table 4.3 - Document:word tuple features

Table 6.1 - Initial corpus ARFF setup

Table 6.2 - Naïve Bayes initial results

Table 6.3 - C4.5 initial results

Table 6.4 - K* initial results

Table 6.5 - Neural net initial results

Table 6.6 - Average accuracy of trained algorithms, by corpus

Table 6.7 - Average accuracy of trained algorithms, by algorithm

Table 6.8 - Cranfield negative/positive split training sets

Table 6.9 - Cranfield Boolean reduction initial results

Table 6.10 - Cranfield Boolean reduction with equal negative and positive examples

Table 6.11 - Cranfield, equal positive and negative examples with only artificial negatives

Table 6.12 - Cranfield with limited training data

Table 6.13 - Cranfield with less limited training data

Table 6.14 - Performance of Naïve Bayes classifier using body text with titles added

Table 6.15 - Performance of C4.5 decision tree classifier using body text with titles added

Table 6.16 - Proportions of documents with titles

Table 6.17 - Performance with titles prepended to body text

Table 6.18 - Accuracy of a neural net while varying hidden layer size

Table 6.19 - Accuracy with increasing training time, using a learning rate of 0.2.

Table 6.20 - Accuracy with increasing training time, using a learning rate of 0.1

Table 6.21 - Corpus size vs. improvement offered in a trained system

Table 6.22 - Keyword/attribute associations for "cheap"

**Formulae**

Formula 2.1 - Term frequency x Inverse document frequency
Formula 2.2 - Recall in IR systems
Formula 2.3 - Precision in IR systems
Formula 2.4 - F measure
Formula 2.5 - F measure with precision and recall equally balanced
Formula 2.6 - Kendall's tau measure
Formula 2.7 - Perceptron weight adjustment
Formula 4.1 - Recall of trained algorithm when working with reference collection
Formula 4.2 - Precision of trained algorithm when working with reference collection

**Figures**

Figure 2.1 - A perceptron
Figure 2.2 - A neural network in WEKA with no hidden layers
Figure 2.3 - A neural network in WEKA one hidden layer
Figure 6.1 - Cranfield Boolean reduction with equal negative and positive examples
Figure 6.2 - Cranfield Negative examples vs. accuracy
Figure 6.3 - Cranfield, equal positive and negative examples with only artificial negatives
Figure 6.4 - Cranfield with limited training data
Figure 6.5 - Cranfield with less limited training data
Figure 6.6 - Accuracy at processing documents with additional titles vs. title usage
Figure 6.7 - Accuracy of a neural net while varying hidden layer size, with the CACM collection
Figure 6.8 - Accuracy of a neural net while varying hidden layer size, with the MED collection
Figure 6.9 - Corpus size vs. improvement offered in a trained system

# 1. Introduction

### Information retrieval

The ability to locate a relevant piece of information out of a huge mass of data is critically useful, especially as more and more data becomes available to search. The time saved by having an automated system search through information instead of manually looking up data is immense – for example, the indexing of books in a library makes locating them a feasible task; looking at thousands of titles one by one whilst trying to locate a single book is simply not a reasonable task.

Systems that provide the ability to discover potentially useful documents from a large collection are known Information Retrieval (IR) systems. The umbrella covered by this type of system is large – it includes the library system listed above, as well as the accompanying card index. In this document, we are only interested in automatic information retrieval systems that operate on a computer and work over digital representations of documents.

### Searching the web

The web is a huge and fairly unstructured collection of digital documents. It's large enough that locating data manually is a laborious, time consuming and often unrewarding task. This problem is one seemingly ideally suited to automatic information retrieval. As a result, internet search engines have appeared, and are one example of web-based IR systems that help people find documents and information useful to them.

These have evolved over the past years to a point where the most common means of initiating a search is to enter a few key words and perhaps specify a limited number of constraints. The search engine will then return results that its creators believe will be those most useful to the searcher (given the key words used), often with the most relevant document first and others in decreasing order of importance.

### Keeping results relevant

Systems that can provide this ability are significant time saving devices, and those that give the best results will help save the most time. A search engine that returns the perfect answer to what a user is looking for inside its top few results is a much more useful device than one that mixes irrelevant results in with useful documents, or even one that produces no strongly useful results at all. Users will still have to plough through suggested documents before discovering those containing useful information.

Writing an IR system that will work over a diverse set of documents is no mean feat in itself. For example, a direct approach would be to only return documents that contained the terms searched for; this would hopefully ensure that documents are relevant, as they contain requested words. However this risks returning a very large volume of documents, depending how large the corpus that's searched over is. There are a few methods of determining how well a document matches that a query will be examined and evaluated.

It's also possible to use embedded features in a document, including the formatting and position of words inside a document, as well as data about the document, otherwise known as metadata. This could include the date of the document's creation, or if the document relates to e.g. a person, then the location of that person. All these factors can be taken into consideration when trying to provide the best search results possible to end users.

One problem generated by the variety of data available is how to weight or prioritise certain measurements about a document. To this end, it's important to be able to evaluate how well (or poorly) an IR system is performing. A few mathematical measures are available, which we will visit. However most of the more accurate measures require human feedback; intuitively, it's hard to automatically grade whether an IR system is returning relevant documents – some comparison of actual rank and ideal rank would be needed, and determining the ideal rank automatically would solve the IR problem anyway.

### Monitoring performance

Collecting explicit feedback data on every set of results provided via questions (such as “are these documents suitable”, “which document is the best of the set”, “which documents seem irrelevant”) is an extremely time consuming process, both for users of the IR system and for the system itself. This makes the evaluation of an IR system at first seem like a non-trivial task. However, as users interact with a system, they leave certain pieces of data behind; the time taken to select a search result; if the results are ordered, the rank of the result clicked on; documents that they will have considered but not pursued further, and so on. Existing work on user behaviour and analysis of it will be considered, especially in the context of search results, where studies involving monitoring where users look on a page have been undertaken. These studies have resulted in conclusions that can be drawn about the performance of an IR system given user behaviour whilst interacting with it.

Such conclusions are valuable as they provide an inexpensive measure of how well an IR system is doing without requiring explicit feedback. This evaluation data could be used in a process of adjusting IR system internal weightings by trial and error, in an effort to tune the system; this would have to be a continuous process as data was accumulated and the collection of documents grew.

Working with trial and error is likely to produce good results over time. Another method of using the data to classify and sort documents would be to take all data from human interaction with the system – both implicit and explicit – and try to establish rules and conditions for determining if documents are going to be relevant or not.

A series of approaches could be considered for processing this data and making best use of it. Much study has been done in the field of Machine Learning (ML) algorithms that can classify or evaluate instances of data given a set of training data; these are the algorithms that we will pay most attention to as part of this project.

### Adapting to information retrieval

Machine learning algorithms are generally designed to find an optimal solution for processing previously unseen situations based on past experience. There is usually maths and procedure applied to processing the past experiences that allows the algorithm to learn and interpret new situations better than a system coded based on prior knowledge but with no past experience.

This ability to handle unforeseen circumstances and learn from past experiences is well suited to the task of processing data on relevance from humans and turning it into optimised adjustments and tunings of the IR system, in order to maximise performance. What we will attempt to do will be to first amass a quantity of training data, and then evaluate the accuracy of document selection by a learning algorithm. Hopefully there should be a significant rise in performance.

To train such a learning algorithm will require a significant amount of data. We will discuss methods of gathering this data. Hopefully, a wealth of training data will allow more accurate predictions from the learning algorithms, once it’s placed into a suitable format. Finally, we will discuss the performance of a selection of learning algorithms and the underlying IR system, and avenues worthy of further investigation.

# 2. Literature Review

## 2.1 Information retrieval

An information retrieval (IR) system is that provides references to useful information resources when queried. For example, a system that returns a set of film titles when asked for old horror movies could be considered an information retrieval system; the telephone directory enquiries system is also an information retrieval system.

### 2.1.1 IR Models

Models have grown to tackle the problem of retrieving documents based on a keyword query. Some are very simplistic and straightforward, such as the Boolean model, and often have an accordingly low performance level. Others are complex to implement but can achieve good levels of performance. We will look at some classic IR methods below.

#### Boolean model

The most basic version of the Boolean model decides which documents to return by simply adding any to the result set that contain the query term, using a single word as the query. Queries can be made more complex by adding joining more than one term with a standard Boolean operator, most typically AND or OR. For example,

`cat` would return all documents containing the word “cat”

`cat AND steamroller` would return all documents containing both the word “cat” and the word “steamroller”

`cat OR mog` would return all documents containing either the word “cat” or the word “mog”

Some aspects of the Boolean IR model are present in the most popular IR systems that we use – for example, Google [21] assumes use of the AND operator between all of the tokens in its search queries.

A big disadvantage with the Boolean model is that it doesn’t provide any ranking of results; they are simply returned as an unordered set. It is also prone to selecting irrelevant documents – while shorter queries may return many relevant documents, they’re also liable to returning many irrelevant ones. This can be described as having high recall but low precision (see 2.2.1).

A straightforward method to refine the result set and gain higher precision would be to add more query terms after examining the previous results. This process of query refinement (known as *iterative retrieval* [25]) has been studied significantly since the 1960s as part of the SMART retrieval system developed at Cornell university [26] right up to analysis of user behaviour on modern search engines [21]. The downside of adding to a Boolean query is that it can lead to very complex queries that are hard for users to keep track of, and is therefore a potentically unintuitive method of searching.

#### Vector model

The vector model attempts to rank documents by measuring how close they are to the query by representing the query and document as normalised vectors. The ranking algorithm presents documents in descending order of their proximity to the query vector. This allows the most relevant results to be presented first, thus saving time for the user, and making it easier to use results when compared to the Boolean model which returns an unordered set.

The vector has terms as its axes, and for each term, there is a weight, for each document and the query. Single term queries, when not using any kind of thesaurus (see 2.1.3), lead to a one-dimensional vector space. Vector comparisons in this linear space typically resemble a simple tf.idf ranking behaviour (see 2.1.5).

The proximity of the query to each document is typically measured as the cosine between the query vector and each document vector. Those documents closest to the query achieve values at or close to 1; those with weaker matches achieve a lower value down to a minimum of zero.

The vector model proves to be efficient at providing relevant documents in a sensible ordering, and has been used in research IR systems for over thirty years [26]. Its results are still considered acceptable [1], as it has a low computational overhead coupled with good performance.

Where the unaugmented vector model falls down is that while it may retrieve documents that contain the keywords searched for, is has no ability to return documents that are relevant to a query but don't contain the words. For example, a page about oak carving may rank well when searched for using the keywords "oak carving" or "carving in oak", but will probably not be selected by a search for "wood sculpting" or "arts and crafts".

### 2.1.2 Stemming

A problem with the vector space and Boolean models of IR is that they both fail to take into account similar words that are spelt differently. A common criticism of the vector space model is that is assumes term axes are orthogonal (e.g. that they have no relation) when this is in fact often not the case.

For example, a search for "exploring the amazon" using the vector space model would rank documents that featured the phrase "amazon exploration" much lower than those incorporating the search terms verbatim, and "amazonian exploration" ignored entirely. The weight of "exploration" as a term is assumed to be completely independent of "exploring", which is an incorrect assumption.

Many words do contain common roots, as in this example. Both "exploring" and "exploration" share the same first six letters, "explor". This root – or *stem* – is also present in some other words such as "explore" and "explorer", and all have related meanings.

This behaviour can be represented in IR by searching for matches using the stems of words instead of the literal terms. Such a search is likely to have a much broader match as it will incorporate a wider range of keywords and therefore more documents. It can be said to have a good chance of boosting the recall (see 2.2.1) value of the system. Further, as the extra documents returned will all contain terms that are at worst related to query terms, there should be little chance of precision falling despite a higher recall value.

| Stem | Potential words |
|---|---|
| explor | Explorable<br>Explore<br>Explorer<br>Explored<br>Exploring<br>Exploration |
| synchron | Synchronise<br>Synchronised<br>Synchronises<br>Synchronising<br>Synchronism<br>Synchronize<br>Synchronized<br>Synchronizer<br>Synchronizes<br>Synchronizing<br>Synchronous<br>Synchrony |

**Table 2.3** – some stems and words derived from those stems

As can be seen from the above examples, the ability to take a word back to its stem and search for matches based on stem greatly increases the number of keywords than can be considered equivalent. Also in this case, a trivial difference in UK and US spelling of worse ending –ise or –ize has been glossed over, as both “synchronise” and “synchronize” are considered equivalent. This allows users of either origin to search over the corpus in their native dialect and have relevant documents returned regardless of the author’s spelling preference.

A disadvantage of the stemming system is that it is sometimes hard to provide a stem for a word that does not introduce less relevant variations. For example, it is hard to stem the word “synchrotron” any further back than “synchrot” for fear of including all the “synchron” words above. Therefore it’s important to have an accurate stemming algorithm.

One popular and well developed stemmer for the English language is the Porter stemmer [22]. It is efficient and fairly accurate, and is used as part of the SMART retrieval system [12,26]. However some facets of the Porter stemming algorithm lack a linguistic base which leaves its results occasionally unintuitive or erroneous. For example, rather than incorporate a known set of common English prefixes (dis, un, re, anti, a), it will stem words containing these prefixes based on their ‘measure’, which is an approximation of syllable count based upon the ordering of vowels and consonants inside a word. It does not take into account typical English phonemes but instead uses an arbitrary list.

### 2.1.3 Thesauri

An alternative to stemming that provide alternative suggestions for words is the use of a thesaurus. A thesaurus is essentially a linked set of words, indexed by word. The links from each word point to other relevant words. For example, one entry may be:

| Word | Related terms |
|---|---|
| Recent | new<br>fresh<br>current<br>topical<br>hot<br>modern<br>up to date<br>latest<br>contemporary |

**Table 2.4** – A thesaurus entry

Looking up query terms in a thesaurus allows the set of terms that will be searched over to be expanded. Returning all documents that contain “recent” or any of the related terms listed above is likely to generate a larger result set that simply returning all those that contain just “recent”. This will probably increase the recall value (it definitely won’t decrease it as the result set will never decrease in size) whilst hopefully not adversely affecting precision. Certainly the precision of returning documents based on a thesaurus-expanded query should usually be greater than that of returning additional randomly selected documents.

Thesauri can be constructed manually by humans; many have been hand-compiled over the past centuries and are now available in machine readable format. The disadvantages with using of these are that, even though the compilation cost can be ignored, they are language-specific, and they may be missing associations. A general thesaurus will contain different links from, for example, a medical thesaurus. One way of capturing links between words would be to generate a thesaurus automatically. Methods include forming associations between query words and the term frequencies in selected documents (which can be enhanced after part-of-speech tagging is used to verify results), and semantic analysis.

### 2.1.4 Latent semantic analysis

A more recent method of determining how related words may be is latent semantic analysis (LSA) [8]. This works by mapping all documents in a corpus into *latent semantic space*, with each document represented as a vector, whose position is determined by the weights of its terms. Latent semantic space is usually high dimensional due to the large number of terms across a corpus. Then, a transformation is applied that reduces the number of dimension in this space, to help better approximate the proximity of documents. This should have the effect of placing documents that contain many similar words close to each other.

The effect that this has in grouping words with similar meaning comes from the way that similar documents are grouped, and the dimensional reduction. A document containing them phrase “desert weather” may end up being close to one on “Saharan meteorology” as they contain many common words with similar tf.idf weights, despite the actual lexical symbol (word) for the phrase being different. This should give LSA the ability to detect synonyms on its own, without need for external knowledge, such as a thesaurus, and provides an alternative to the orthogonality of axes suggested by the vector model. Further, the problem is reduced in size.

This grouping also allows for the mapping of groups of words as “concepts”; for example, there may be a set of documents that all contain significantly similar tf.idf factors for words such as “Gobi”, “desert”, “Sahara”, “sand”, “drought” and so forth. This relation between words could not be discovered via stemming, and would take extensive thesaurus analysis to discover otherwise.

A disadvantage of LSA is that is cannot distinguish homonyms; so, although documents containing “sand” and “desert” would have a degree of distinction from those containing “sand” and “woodwork”, some weight would still exist between the two uses of the word “sand”. Given its enormous automated gains over stemming or thesaurus based approaches to word expansion, this is not of huge immediate concern.

### 2.1.5 Term frequency

The frequency of a search term in a document can be an intuitively good measure of how relevant it would be. For example, a document containing the word “cat” 17 times may be more relevant to a search for “cat” than a document that only contained it say 5 times.

However such a direct measure fails to take into account the length of the document; it’s fine to have the word cat in 17 times instead of 5, but if the document with higher frequency is very long (or the one with lower frequency very short) then the situation becomes less clear. To remedy this, it is possible to take into account the document length and then calculate a keyword density.

| Document | Search term frequency | Document length (words) | Keyword density |
|---|---|---|---|
| **A** | 17 | 1000 | 1.7% |
| **B** | 5 | 80 | 6.3% |

**Table 2.1** Search term frequency and corresponding keyword density

This way, despite document A having a higher keyword frequency, document B would be the more relevant document. This keyword density metric is trivial to calculate as a document feature.

The importance of individual words in determining relevance is also a factor. Common words such as “try” and “the” will be found in many documents inside a corpus. When these words are used as part of a query, providing those documents that are most dense with them near the top of the result set may not be the best strategy. Other words used in the query could have much more value in determining relevant documents. For example, given a query such as “causes of cancer”, providing results dense with the word “of” with the same priority as those than contain “cancer” or “causes” is misleading.

To this end, a damping factor can be applied to the term frequency that is derived from how common the term is across all documents in the corpus. Because each document may have a

different word density, instead of counting all occurrences of the term across the entire corpus, only the number of documents containing it is counted.

For example:

| Term | Total corpus frequency | Number of documents containing term |
|---|---|---|
| **insurance** | 10440 | 3997 |
| **try** | 10422 | 8760 |

**Table 2.2** Frequency of terms across a corpus, and the number of documents that use the term [1]

In the above example, although both words are similarly frequent across the corpus, “insurance” occurs in many fewer documents. We can use this knowledge to give “insurance” greater weight in queries, as documents containing this are more likely to be relevant than the 84% that contain “try”.

We can derive a metric based on term frequency and inverse document frequency that ties together these two features. Formally this measure of *tf.idf* [27] (term frequency × inverse document frequency) for a single document and term pair can be defined as: [17]

$$w_{(i,\,j)} = (1 + \log(tf_{i,\,j})) \log \frac{N}{df_i}$$

**Formula 2.1** – Term frequency × Inverse document frequency

Where:

N is the total number of documents

$tf_{i,j}$ is the frequency of term j in document i

$df_i$ is the document frequency of term j

This measure can be directly used as a vector weight for the vector space model. The damping part of the inverse frequency measure scales the individual term frequency into a much more friendly form, and also gives words that do not occur frequently in the corpus a chance to gain extra weight.

A potential disadvantage of td.idf is that words that occur in the corpus very infrequently (e.g. once or twice) are given a very high idf factor. This means that although documents containing these words will rightly leap to the top of results, the weight of other words may be over diluted in subsequent listings.

### 2.1.6 Stopwords

While tf.idf goes a long way into making sure that the commonest of words (such as “and” or “the”) do not affect search results any more than absolutely necessary, there is still computational effort applied into computing these small weights, and some end effect derived from the minor words.

This can be completely eliminated by compiling a list of words that have no information relevant to the query or document’s content. These words will all be extremely common and have little or no informational content and certainly no contribution to selection of document by single keywords. They tend to be prepositions, pronouns, numbers, or conjunctions.

A list of stop words for a particular language can be taken from an external source, which is simple and tends to yield a comprehensive list. The disadvantage here is that some words which have unusual importance for a particular task may be excluded, and that it is time consuming to check the list over.

### 2.1.7 Indexing

As the size of a corpus grows, it becomes harder and harder to iteratively search over it and run search operations. It is particularly futile to examine documents for words that are not there every time a query is run, especially if the query consists of many key terms. Each term will involve the examination of each document to see if it contains the term; thus, the complexity of the task grows linearly with both the size of the corpus and the number of query terms.

#### Forward indexing

To reduce the amount of computational effort required in searching through documents, an index of the documents in the corpus can be built. This is usually a file sorted by document identifier. Each record in the file would contain the document identifier, and then a list of words in the document. The amount of data stored can be shrunk by removing stopwords and stemming the remainder; so, for example:

Document (excerpt taken from [3]):

```
“It certainly undermines still further the original pretense that the
police were firing in response to Panther gunshots, confused by
unfamiliar surroundings. The Chicago press has reported that the FBI
agent to whom O’Neal reported was the head of Chicago Cointelpro
directed against the Black Panthers and other black groups. Whether
or not this is true, there is direct evidence of FBI complicity in
the murders.
```

Corresponding record in forward index, with stopwords, case information and punctuation removed:

```
“undermines still original pretense police firing response panther
gunshots confused unfamiliar surroundings chicago press reported fbi
agent oneal reported head chicago cointelpro directed against black
panthers black groups true direct evidence fbi complicity murders”
```

To reduce the need for storage further (which makes sequential searching of the text quicker), multiple words can be removed, and instead replaced by an optional word count. Further, the text can be ordered alphabetically, to allowing rapid seeking, perhaps via binary partitioning. So, we could end up with:

```
“against agent black,2 chicago,2 cointelpro complicity confused
direct directed evidence fbi,2 firing groups gunshots head murders
oneal original panther panthers police press pretense reported,2
response still surrounding true undermines unfamiliar”
```

This text is much easier to search through for key terms. Stemming the words would further reduce the amount of data stored, in this case grouping “panther” with “panthers” and “direct” with “directed”.

However some data has been lost with reordering, stemming, and the removal of stopwords, punctuation and case information. This denies the option of studying some metrics associated with keywords, such as their position in the document, where they occur in a sentence (useful for part of speech tagging) and their formatting. Also, searches over this forward index are still relatively computationally intense especially given multiple key terms.

#### Reverse indexing

A reverse index (or inverted index), instead of being sorted by document ID, is sorted by word ID. This allows for the rapid lookup of data given a word – much more so than going through individual documents [17]. Reverse indices consist of a file ordered by word ID. Each word ID has a list of documents containing that word associated with it, as well as any optional data.

An example (very small) reverse index might look like:

| Word | Documents |
|---|---|
| Cat | 1,5,8 |
| Fish | 1,9 |
| Sahara | 2,7 |
| Sand | 2,3,4 |
| Turkey | 6,9 |
| Woodwork | 4 |

**Table 2.5** – A simple example reverse index

Significant performance increases can be made by representing the index in this way. Instead of examining each document to see if it is a match to the keywords used, only the documents listed by each keyword need be examined. The others can be discarded as irrelevant. Documents are indexed by deriving a list of words they contain, usually after stopword removal, and adding each word to the reverse index. This leads to the bulk part of computational work being done offline as documents are processed, which allows for greater flexibility than having load whilst online operations such as queries are performed.

Whilst stemming can be used to create a larger set of match candidates at search time, and then these can all be searched for, a much simpler approach would be to stem words found in the document at indexing time, and store stems instead of complete words in the reverse index. This reduces the number of lookups needed per query on the reverse index, and also reduces its size. In the worst case, stemming will be performed on a single word query, have no effect, and a single word's document list will be discovered. Performing stemming on the reverse index leaves flexibility as to whether or not to use stemming when considering returned documents for ranking, depending on if it is performed in the forward index, thus allowing a distinction between stemmed and complete matches.

Such a simple reverse index as described above omits much of the information about words found in documents, such as the density of the term or its position, and would have to be combined with a fairly rich – or even unaltered – forward index for this information to become available. Doing this would also involve either a complex forward index and a lookup to each document every time a search was conducted, or computation of extra metrics (e.g. term density) for every search. As documents tend to remain unaltered once indexed (depending on the setting), this seems inefficient.

One time-saving device could be to include some data about terms in the corpus and in each document as part of the reverse index. For example, tf.idf could be calculated more quickly by including the term frequency in each entry in the reverse index, so that the list is now of document ID and word count pairs.

| Word | Documents | Frequency |
|---|---|---|
| Cat | {1, 4}, {5, 34}, {8,12} | 50 |
| Fish | {1, 19}, {9, 7} | 26 |
| Sahara | {2, 22}, {7, 12} | 34 |
| Sand | {2, 19}, {3, 14}, {4, 6} | 39 |
| Turkey | {6, 4}, {9, 9} | 13 |
| Wood | {4, 8} | 8 |

**Table 2.6** – A reverse index allowing for tf.idf to be calculated

This would allow idf to be derived given the size of the corpus. The total number of occurrences of each term could easily be calculated, by summing its frequency in all documents it is found in simply by counting the record size. Alternatively, this total frequency

could be stored with the word. In the case of corpora where the total number of documents changes infrequency, idf for each word could even be stored with the word record.

### 2.1.8 Document formatting

With hypertext documents, typically presented in HTML, the presentation of individual terms is machine readable and can thus be taken into consideration when indexing and searching over documents. Two readily available metrics that can be simply implemented are:

- Font size and style

  Seven standard levels of heading are available in HTML, outside of normal text. These are denoted <h1> for highest priority headings, down to <h7> for minor headings. In addition, text not denoted as any kind of heading (body text) can be assigned a lowest priority. An example of a phrase tagged as a second-level priority heading in HTML is: <h2>Sahara Desert</h2>. Further, text can be designated as bold, italic, emphasised, and strong, using <b>, <i>,  and  tags respectively. Emphasised and italic text is considered semantically equivalent in current revisions of HTML, as are bold and strong text. Strong text will be given a higher priority than emphasised text, and both have higher than default priority. This leaves a total of 10 ordered font classes.

- Word case

  Case sensitivity is not straightforward to incorporate into the system. If we start by assigning a case value to a word based on the percentage of letters that are uppercase, then this value will quickly become unfairly skewed by word length. Categorising word case into class would overcome this; a set of three possible classes could be “all uppercase”, “sentence case” and “first character lowercase”. This however could be thrown off by author style; a document entirely in caps could incorrectly attain higher rankings. Therefore, to reduce this impact, the mean case value over a document would be calculated, and a case metric produced for each word based on how different it is from the document average.

  For example, if we arbitrarily assign a value of 8 to “all uppercase”, a value of 3 to “sentence case” and a value of 1 to “first character lowercase”, then the average value for a document all in capital letters will be 8. Therefore, the relative case value for capitalised words inside this document would be derived by determining the value for the word (8) and subtracting the document average (8) – e.g. 0. This system is simplistic and should yield satisfactory results; it will not cope well in the case where documents are in inverted case. Such behaviour could perhaps be seen as encouragement to authors make their documents more readable and conform to normal English grammar rules.

#### Deriving hit lists from hypertext documents

One problem with simple reverse indices is that they risk losing data about the instances of words inside listed documents. It is not directly easy to store data on capitalisation, formatting, and position of each individual word without using a more complex storage structure. Early iterations of Google overcame this by implementing the both the forward and reverse indices as files of records containing a document or word identifier (respectively) followed by a *hit list* [21]. As defined by Page:

“*A hit list corresponds to a list of occurrences of a particular word in a particular document including position, font, and capitalization information.”*

This allows for the lossless capture of any data required from a source document.

### 2.1.9 Existing IR systems

With the advent of the web as a collection of loosely related rich hypertext documents, there has been a proliferation of interest in and instances of information retrieval systems. For example, the Google search engine [21] is a mature and complex IR system working over billions of hypertext documents.

The SMART IR system [26], developed in the 1960s, was a leading development and research platform, replicated at many academic institutions. A good quantity of material based on studies using and experimenting with the system is available.

### 2.1.10 N-gram analysis

A document can be said to consist of unigrams – that is, words. These are already analysed by the tf.idf algorithm, which can on its own be used for ordering candidate documents for a single word query, and without too much effort, multiple word queries. Documents could also be seen as consisting of a set of both unigrams and bi-grams; that is, sequences of two words. For example, the following sentence:

`"Jack sat on the big fat mat"`

Has 7 unigrams, and 6 bigrams (`'Jack sat'`, `'sat on'`, `'on the'`, `'the big'`, `'big fat'`, `'fat mat'`). The tf.idf analysis could then be performed with the bigrams, to assist in multiple word searches. This will only help if words are ordered the same in both search query and document body. This can be extended for trigrams, 4-grams, or sequences of N words; hence "N-gram analysis".

Following modest n-gram generation, it would be possible to filter out statistically unimportant n-grams from a corpus. For example, N-grams appearing once are unlikely to be strong indicators of topic, and N-grams appearing over different classifications of document may be of little use unless they are particularly common in a small subset of classes. This would reduce the size of the problem to manageable proportions. The maximum value of N would depend on the point at which no more useful N-grams appear.

N-gram analysis could be used to generate training data for an ML algorithm. Feature reduction can be performed, by first removing stopwords and rare N-grams of frequency $f_{min}$ or less. Words occurring with a high frequency in one class can then be retained, using a measure such as $\chi^2$ [19] or similar. The top $k$ N-grams can then be retained, and their presence used as a Boolean feature. All documents in the corpus can then be represented as a list of the significant N-grams they contain. This is a simplified version of the approach used in [20].

## 2.2 IR system evaluation

To be able to judge how good a system is at retrieving information, some kind of objective metric is required. For these metrics, we will assume that results are provided as an ordered set of document abstracts and titles, truncated to show only the top $n$ most relevant items.

### 2.2.1 Recall and Precision

A perfect IR system would retrieve every single relevant document that was available, given a query, and no others. The proportion of these relevant documents recommended by the IR system is known as its *recall*. For example:

$$recall = \frac{number\ of\ relevant\ documents\ returned}{total\ number\ of\ relevant\ documents\ available}$$

**Formula 2.2** Recall in IR systems

Thus, recall is a measure of the volume of relevant documents returned from a search, and has a value in the range 0 – 1.

There is one problem with recall in that it is easy to maximise without actually providing useful results. For example, an IR system that returns the entire corpus as a search result will always provide maximum recall, as the set of relevant documents will always be a subset of the entire corpus.

To this end, the *precision* of a search is also calculable. Precision is a measure of how precise the result set is, taken from the proportion of the result set that is relevant to a query.

$$precision = \frac{number\ of\ \ relevant\ documents\ returned}{number\ of\ \ documents\ returned}$$

**Formula 2.3** Precision in IR systems

Precision, like recall, has a value in the range 0 – 1. Maximum precision can be achieved by returning a very small number of documents that are all relevant; it does not take into account how many unshown relevant documents there might be.

Both precision and recall are easy-to-calculate measures of the performance of an information retrieval system. They still demand some external estimation of whether a document is relevant to a query or not to be calculated.

### 2.2.2 F-measure

The factors of precision and recall are linked; selecting the entire corpus yields complete recall and low precision, and selecting only a very low amount of highly relevant documents will give a high precision accompanied by low recall. The two factors can be combined into a metric defining the tendency to return many documents with low relevance or few documents with higher relevance. This is known as the *F measure* [29] and defined as follows:

$$F_\alpha = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}}$$

**Formula 2.4** – F measure

Where $P$ and $R$ are precision and recall, respectively, and $\alpha$ is a factor of bias toward $P$ or $R$. A higher $\alpha$ leads to recall being preferred over precision, and a lower $\alpha$ favours precision over recall.

Often $\alpha$ = 0.5, where recall and precision are considered equally important. In this situation,

$$F_{0.5} = \frac{2PR}{P+R}$$

**Formula 2.5** – F measure with precision and recall equally balanced

### 2.2.3 Rank of selected document

The rank of the abstract chosen for further examination is a simplistic metric that is trivial to implement. Although it has noise and normalisation issues as described in 2.3.3, it's a very easy metric to obtain and has still some value, especially with large sample sizes.

### 2.2.4 Average precision

One way of assessing a set of results when the ideal ordering and IR system behaviour are known is to calculate the precision after each item in the result set, and takes the mean over all items. This is known as the *uninterpolated average precision* of the results [17,29].

| Produced ranking | Ideal ranking | Average precision |
|---|---|---|
| Relevant | Relevant | 1.0 |
| Relevant | Relevant | 1.0 |
| Irrelevant | Relevant | 0.67 |
| Irrelevant | Relevant | 0.5 |
| Relevant | Relevant | 0.6 |

**Table 2.7** – Uninterpolated average precision

As can be seen from the data above, this particular result set has a mean average precision of 0.6. This could be observed over many result sets to get an overall estimation of system performance.

### 2.2.5 Kendall's tau

Another metric for evaluating a ranked set of results against an ideal ranking is found in Kendall's $\tau$ [13,19]. The measure has a maximum value of 1 for perfect ranking matches, and a minimum of -1 for completely dissimilar rankings.

This metric involves counting the number of concordant and discordant pairs in a ranking. A pair of items is concordant if both the rankings choose to place both items in the same position.

| Ranking A | Ranking B | Status |
|---|---|---|
| Document 1 | Document 1 | Concordant |
| Document 2 | Document 2 | Concordant |
| Document 3 | Document 3 | Concordant |
| Document 4 | Document 5 | Discordant |
| Document 5 | Document 7 | Discordant |
| Document 6 | Document 6 | Concordant |
| Document 7 | Document 4 | Discordant |

**Table 2.8** – Concordant and discordant pairs by Kendall's $\tau$ measure

Quoting from [10],

> Kendall's $\tau$ can be defined based on the number *P* of concordant pairs and the number *Q* of discordant pairs.
>
> Kendall's $\tau$ can be defined as:

$$\tau(r_a, r_b) = \frac{P - Q}{P + Q}$$

**Formula 2.6** – Kendall's $\tau$ measure

So, the above example would have a $\tau$ of 1/7, or approximately 0.143.

This measure neglects cases where a large part of the ordering is correct, but slightly offset; for example, if an IR system produces a ranking ordered the same as the ideal ranking but for there being an extra document inserted at the first position, no concordant pairs will be found.

## 2.3 Human interaction with IR systems

The way that users behave whilst using IR systems can be indicative of the performance of the system and provide a useful insight into how humans go about locating information.

### 2.3.1 Query refinement

When users are looking at data, they typically have anof their requirements. Based on this, they will select a few keywords to use in a query, and submit this query to an information retrieval system. The system will then return a set of results, typically represented as abstracts and references to their parent documents, from which the user may select a particular line of investigation.

There is a significant risk, depending on the user and the specificity or scarcity of the information that they require, that there will not be a suitable document in the first set of results returned. One step to take to resolve this situation is to *refine* the query – especially in systems with elements derived from the Boolean search model – so as to alter the result set, in the hope of discovering more relevant documents. Another method for attempting to discover more documents in systems that sort results and initially return on the best matches is to seek further down the returned list of potentially relevant documents in the hope of discovering one that matches the requirements.

Lau & Horvitz [16] studied the logs from a busy public IR system and categorised query refinements into classes, including specialisation, generalisation, and reformulation. It was possible (to some extent) to gauge the type of query reformulation based on the time interval between requests for data, and also a measure of similarity between query phrases. This data could be extrapolated to better understand user intent.

There is a potential for information loss in *query formulation* – that is, between the user's initial desire for information and a query that they submit to an IR system. The specific data that the user wants will often have to be first translated into words as they attempt to build a query, and then biased by the user's former experiences of entering queries into IR systems and using the results returned. This step is a point of data loss that is uncontrollable as it occurs before initial interaction with the IR system.

### 2.3.2 Clickthrough analysis

As mentioned above, working out the performance of an IR system often requires some kind of external metric and subjective analysis. A direct approach to evaluation could be an explicit round of questions aimed at the user, asking them to subjectively gauge the experience. However this is time consuming to conduct and gather a large amount of data from, and subjects are often unwilling to donate their time and effort to provide such feedback.

A certain amount of logging can be taken from interactions with IR systems with minimal overhead. Generically, the most basic kind of data required would be;

- The query itself
- Some data about the result set returned
- A means of uniquely identifying users

In the case of web-based IR systems, some of this data is automatically available as part of many popular web servers' logging configuration. For example, the Apache web server, which includes time of request, the address of the requesting computer, and the name of the resource requested (its URL). Each log entry is time and date stamped to the latest second, which can serve as logging the time of query submission. The address of the requesting computer can be used as a unique identifier in most cases, with some exceptions. Further, the URL part of the log entry can contain the query used, as long at the HTTP GET method is used (form data is typically sent with the POST method, whose parameters are not explicitly get logged by Apache). Data on the result set is not logged as this is generated by application code on a much higher level than the Apache server operates; instead, it would have to be logged elsewhere by the IR system.

This data allows for the analysis of queries used, an approximation of the recall and precision value of the system (given extensive and possibly manual analysis of the corpus searched over in comparison with each query used), and analysis of the query refinement process. Whilst these are good metrics to have, they are either expensive to calculate, or do not produce much direct feedback on the performance of the system.

It is possible to increase the value of log analysis by logging more variables associated with the search process. For example, logging the position of the result clicked on can give an approximation as to how users see the result set (given that result sets are ordered). According to the *probability ranking principle* [29], documents that are more likely to be relevant to a query should be at the top of the result set (when ordered); thus, once users evaluate the result summary presented to them, selecting a higher-ranked document is a likely indicator that the IR system is working well, and selecting a low-ranked document could mean that relevant documents are being incorrectly assessed as irrelevant.

Logging the position of the selected document [2] is a simplistic and trivial approach, and adds some value to the process of feedback collection. It does have some issues in that the data is noisy and still subject to variations per individual user and query. If there are a large number of very similar and relevant documents, selecting the tenth or fiftieth document may not be an indicator of poor performance; similarly, if three documents are returned and the third is chosen, then this is possibly a likely indicator or bad performance. Also, human users are liable to misreading results or accidentally skipping over candidate documents, which would produce an untrue clickthrough position. Attention could also be paid to the frequency of certain searches, with the average position of more commonly sought after words being more heavily weighted when calculating average system clickthrough position. However despite its extreme simplicity and flaws, this metric does have some value once a sufficiently large amount of data can be collected.

Some in-depth analysis has been done on human-computer interaction using IR systems through a GUI, where results are returned in an ordered manner. This matches the setup of many web search engines and commercial systems and seems an appropriate setup to examine. Eye tracking studies [11] have shown that users tend to read through results in the order that they are presented. There is also a significant focus applied to evaluating the first one or two results before considering any of the remainder of the results. The WIMP environment and the use of the mouse to select a result lend the term *clickthrough analysis* to the practice of examining user actions in these systems.

Behaviour observed in these studies can be coupled with the rank of the selected document to provide information on which documents in a result set are relevant and also those that may have been considered but were not the selected item [23]. For example, if a user selects the third document in a set, it is assumed that this document is relevant to the query (which is positive feedback) and also that the two documents above are not relevant. Further, if the first result in a set is selected, not only can the evaluation of this document as relevant be reinforced, but also the second document in the set can be negatively assessed, as we know that the user has probably evaluated both of these abstracts in the process of their search. This provides both positive and negative feedback on some documents for every single query when submitted.

### 2.3.3 Issues with implicit feedback

Implicit feedback is gatherable, but as mentioned in [23], not accurate all of the time. The most confidence in user behaviour given an observed log pattern is only 84%; this drops to 62% for some actions. If this data is used, some consideration needs to be taken to ensure that this uncertainty is reflected. As both implicit log data and explicit user viewpoint will be gathered as part of a study (see 6.2.2 below), we can assess the accuracy of our inferred user opinions and see how accurate they are. This could possibly then be taken into account by requiring more instances of positive or negative feedback related to a document depending on how confidently user opinion is assessed. For example, if a particular log pattern represents a type of behaviour only 80% of the time, it may be possible to require 1.25 instances of this pattern before feedback is applied.

Also, in [16], relationships between the type of query just entered and the delay before subsequent queries (if any) from a single user is used to develop a statistical approach for

estimating a query type (see 2.3.1 query refinement above). This could in turn be used to weight the confidence of each implicitly observed behaviour.

### 2.3.4 Ranking effect

In [26], the *Ranking effect* is described, where queries are modified based on data from document references. This is akin to submitting a query to an IR system and then declaring a document in the result set as being more relevant to that query. Any relevant documents declared will then be ranked higher for this query. We expect to observe this affect after applying a machine learning algorithm to human behaviour gathered.

## 2.4 Machine Learning

To get best performance out of an information retrieval system based on feedback, a procedure of reacting optimally on the feedback is a great boon. It allows for efficient use of all the data available. An algorithm that can take an amount of historical data, possibly noisy, and learn rules from it would fit the bill. Machine learning provides algorithms that can adapt to and learn from a set of training data much faster and hopefully more optimally than a human user attempting to find correlations and trends. This kind of unsupervised approach is suitable for use in improving the performance of our IR systems.

### 2.4.1 Decision trees

When instances of a problem are presented as a set of attribute values, and the task is to classify unseen instances into one of a set of classes, the classification process could be tackled as a series of decision based on attribute values. For example, whilst discussing books, a crude piece of logic to distinguish between types of publication given the binding type and cover flexibility could be:

```
If COVER_FLEXIBILITY = hard
      TYPE = “hardback book”
ELSE
      IF BINDING_TYPE = spine
            TYPE = “paperback book”
      ELSE
            TYPE = “newspaper”
```

There are obvious omissions here, such as how to handle glossy magazines with spines. But for example’s sake, it is possible to translate this logic into tree form, with each node being a decision, where the decision process begins at the root, and terminates with classification at a leaf. This is known as a *decision tree*.

The process of choosing an attribute to make decisions upon can be governed in various ways. The ID3 algorithm [18] uses an entropy-based information gain metric, where it chooses an attribute that will contribute most to determining class at any one point in the tree for evaluation, and adds a node for it. This attribute is then removed from the set of attributes available for evaluation. The process continues recursively, with leaf nodes being left in cases where the only applicable instances are all in the same class, until no attributes remain.

The major issues with ID3 are that it is a direct hill climbing algorithm with no backtracking; thus, it is prone to finding locally optimal solutions instead of a globally optimal tree. Further, its information gain measure can be misdirected by attributes that have many potential values; as a result, ID3 decision trees are often very wide at the top and make decisions based on narrower sets of outcomes as depth increases. Also, ID3 is unable to cope with real-valued attributes.

C4.5 is a decision tree algorithm based on ID3 that implements backtracking, tree pruning, the ability to cope with real values, and due to its backtracking, it not as easily misled by information gain.

### 2.4.2 Neural nets

The human brain is composed of a large network of neurons, each with multiple inputs and an output that activates once certain levels have been reached at the inputs. An attempt to model this behaviour and thus copy human thinking ability has grown into a powerful machine learning algorithm in the form of artificial neural networks [18]. The networks typically used in machine learning have nowhere near the scale of interconnection found in the human brain, which consists of around $10^{11}$ neurons each connected to around $10^4$ others. For comparison, ALVINN [18] (a system that can steer a car down a highway) has around 1000 inputs, 4 internal neurons and 30 outputs.

A simplistic type of machine based neural net is comprised of units called perceptrons. These have a set of weighted inputs and a single output. The output is set to a value of either -1 or 1, depending on whether the sum of the input values received is over an internally set threshold. These units are connected to each other in networks of 1 or more layers; a 1-layer network typically has just one perceptron.

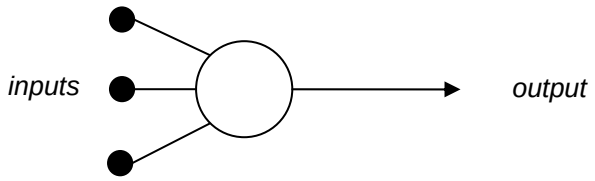


**Figure 2.1** – A perceptron

Perceptron input weights are derived using an algorithm that converges on a hypotheses matching the training data. Weights can be assigned random values initially, which are adjusted by a value derived from the target outputs at each step as follows:

$$\Delta w = \eta(t - o)x$$

**Formula 2.7** – Perceptron weight adjustment [18]

Where *t* is the desired output with the current training instance, *o* is the current output of the perceptron, *x* is the current input value, and $\eta$ is the *learning rate*. The learning rate sets the scale of changes made to the weights; a higher learning rate allows more mobility over potential weight values, which is good for a small training set, but can prevent weights from accurately converging on ideal values. A smaller learning rate is preferable, as long as there is enough training data available.

A neural net consists of perceptrons linked in a non-cyclic graph. They will have an input for each field of training data, and an output. The simplest net has one perceptron. This can be grown by adding more perceptrons in different layers. WEKA – a tool for experimenting with machin learning algorithms – builds nets by creating two output perceptrons, and connecting every input to each one.

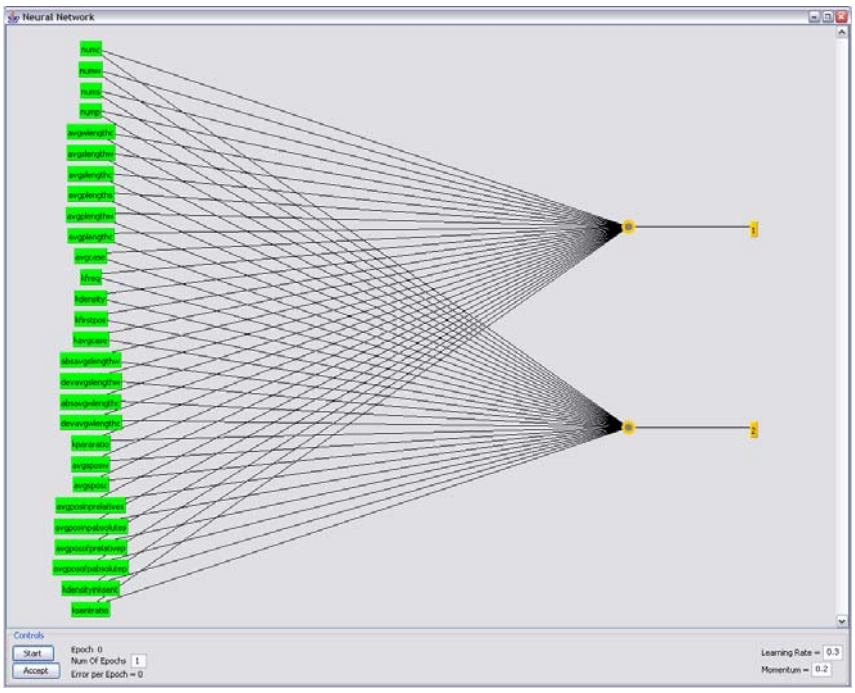


**Figure 2.2 –** A neural network in WEKA with no hidden layers

Hidden layers can be user configured. The default setup is to have one layer of n neurons, where n is the sum of the number of attributes available in the training data and the number of possible output classifications.

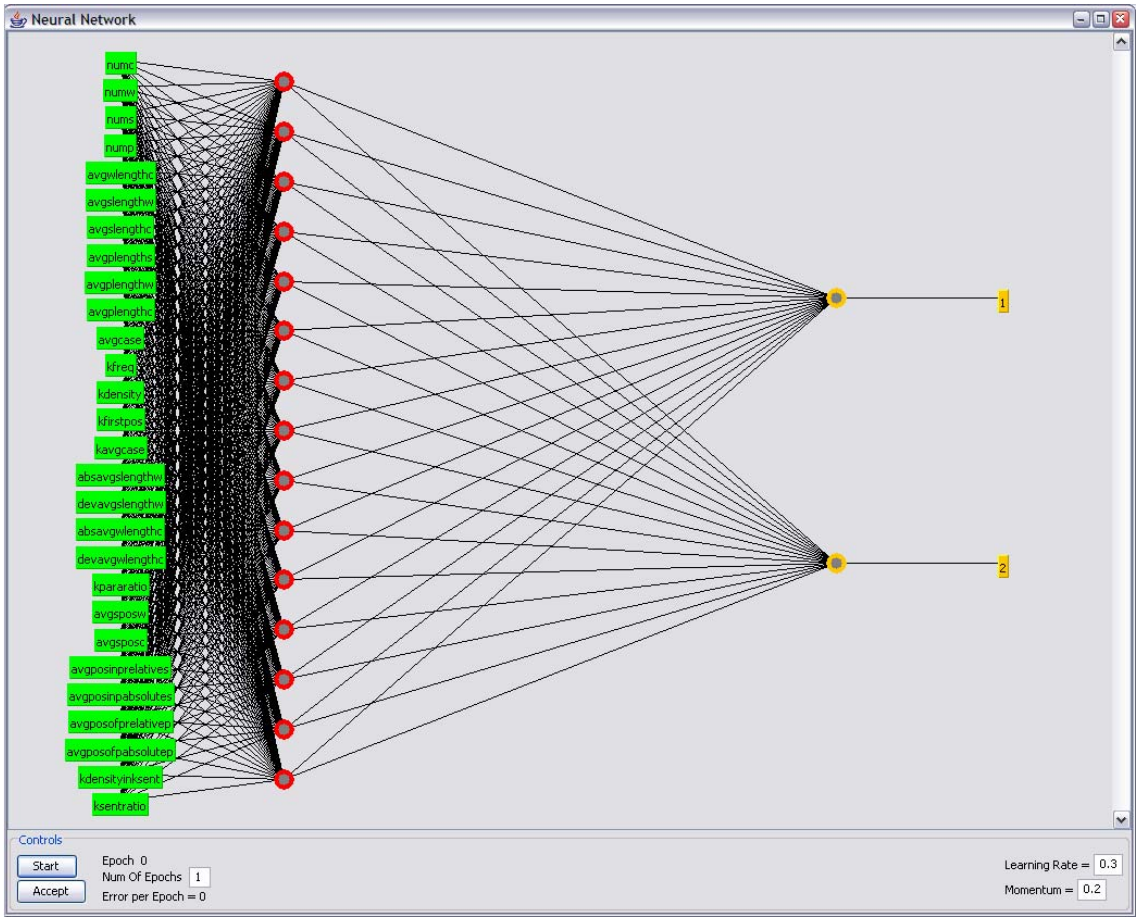


**Figure 2.3 –** A neural network in WEKA one hidden layer

As can be seen from figure 2.3 above, even one layer with a modest number of hidden layers can lead to a large number of relationships being created. Training such a net will involve a correspondingly large number of calculations.

### 2.4.3 Bayesian learning

The probabilistic theories of Bayes include methods for calculating the probability of events given certain situations. This can be mapped into machine learning by attempting to calculate the probability that an instance belongs in a certain class, based upon the statistical relationships and data found in a training set.

The naïve Bayes classifier is capable of learning the class of instances given a set of attributes associated with each one, and then estimating the class of future unseen instances. Instead of calculating the probability of an attribute set given a classification, the naïve Bayes classifier assumes that all attributes are independent (which is not necessarily true).

### 2.4.4 Instance based learning

Machine learning algorithms can be termed as lazy and non-lazy. Those that are non-lazy perform calculation when trained; lazy algorithms simply store the data. Instance based learning is a type of lazy learning algorithm, unlike the others discussed which are all non-lazy.

A straightforward instance based learning method is to store all the training data as points in *n*-dimensional Euclidean space, where *n* is the number of attributes per instance. Unseen examples can be classified by their proximity to existing stored points in this space. This makes both training and classification trivial tasks.

The *k*-Nearest Neighbour algorithm operates as described above, and classifies new examples by examining the *k* nearest points and designating the new instance as the most common value in this set of *k* examples. Issues with this method occur with noisy data, when some axes may have values that are not fair estimations of overall behaviour, and also when only a few of the attribute values actually have any bearing on the classification of a particular new instance. To this end, attributes / axes can be weighted differently, as described in 2.1.4.

The implementation of k-Nearest Neighbour used will be K* [4]. This uses an entropy-based distance measure to attempt to reduce the distorting impact of noisy data.

### 2.4.5 Training and testing machine learning algorithms

Machine learning algorithms need to be trained before use. This is typically achieved by sequentially passing training examples to an algorithm. Depending on the algorithm used, the training examples may be a vector of real numbers, or a group of nominal attributes. Usually some translation between various input types is possible. The final classification of each training example is supplied alongside the training data. This allows the algorithm to build associations between the training data and the final value however they see fit.

Once all the training data has been input, the algorithm is said to be “trained”. It can be evaluated by passing it the attribute values of further data – test data – and seeing whether its estimation of the final classification is accurate or not. Instead of manually supplying testing values, it can be easier to automate these tests. Often, if a fixed body of training and evaluation data is available (for example, a limited data capture), this can be split into training and test sets. The split can be varied to provide more training data or a more thorough test.

It’s good general practice to repeat any test before confirming the result, and ML algorithms are no exception. Each test should be repeated, perhaps using a different random seed. If a fixed data set is used, it’s recommended that the selection of examples in the training and test data be mixed around, even if the sizes stay the same. This will prevent any unusual examples skewing data.

### 2.4.6 Overfitting

When there is only a limited amount of training data available, (which is always the case when algorithms are not trained ad infinitum) it is possible to learn the training data too closely. This would occur by discovering nuances unique to the training set available and adapting to them. The adaptations typically detriment the ability to correctly classify the unseen data in the test set. Overfitting typically occurs when an algorithm is trained too much or in a very fine-grained manner.

Overfitting can by some extent be avoided by setting aside some data and measuring performance on this data after each training example has been processed. As soon as a significant reduction in accuracy is found, training can be stopped. This set aside data can be called a “validation set”.

# 3. Requirements and analysis

## 3.1 Aims and objectives

More often than not, no feedback is gathered on the performance of web-based search engines. This leaves the search systems and their operators utterly clueless as to whether or not the system is behaving as intended, without extensive explicit testing. A vast body of feedback information is available in the form of metrics than can be derived during human interaction with such systems. It seems reasonable to attempt to optimise the performance of such a search system by automatically adjusting its behaviour based on feedback generated by its users.

The aim of the project is to compare the performance of machine learning algorithms at the task of information retrieval. This will be achieved by considering the internal mechanics of a basic IR system and the capabilities of some machine learning algorithms, and attempting to construct an interface between the two. A basic IR system will then be built that allows for the storage and retrieval of hypertext documents. Decisions affecting which results to present in response to user queries, and the ordering of these results, will be made by a machine learning algorithm. The machine learning algorithm will have been trained to produce what are believed to be optimal results.

We should learn about information retrieval theory and practice, the strengths and weaknesses of a set of machine learning algorithms at assessing data associated with human behaviour, and human interaction with web-based information retrieval systems.

Further, it is important to represent the problem of teaching an algorithm to select documents. This will require a volume of training data and considerable effort in getting a high level of performance out of any learning algorithm. Various parameters will be experimented with when it comes to building the training sets and also the behaviour of the ML algorithms themselves.

We will compare a small set of machine learning algorithms at the task of information retrieval given a keyword-based query with a simple baseline IR system that does not make any use of feedback.

The primary focus here is on the IR performance of the system and the learning efficiency of various algorithms. System resource usage is not particularly relevant and certainly should not be a limiting factor. Thus, resource-oriented performance metrics are only given secondary consideration, if at all.

## 3.2 Which ML algorithms to use

In order to uncover further avenues of activity and get a diverse evaluation of machine learning algorithms whilst observing only a small set, some care has to be taken over the content of the set used. Despite there being a wide range of machine learning algorithms to choose from, only the simpler ones fall under the scope of this project due to time considerations. Even with these restrictions, it could be costly to experiment with a more complex algorithm when the performance of simpler ones remains unknown. Thus, representative algorithms from major families are used.

As decision trees form an easy to understand and powerful set of machine learning algorithms, it would be good to include a simple and well-examined decision tree based method. The C4.5 algorithm is well developed, relatively simplistic, and capable of handling the data presented. Its predecessor, ID3, is perhaps too simple for the tasks, and has some restrictions (such as the inability to cope with real numbered variables) that make it unsuitable for some problem representations.

Neural nets are of interest and again in their simpler forms, not too time consuming to understand and experiment effectively with. A simple perceptron based neural network could be used to classify documents based on training data provided. As neural nets cope well with noisy data, they seem well suited to the task. Further, the format of the training data should be straightforward to work with. We are also not too concerned with the internals of algorithm decisions, and much more so with the final results, so the likely unintuitive weights at individual perceptrons should not be a problem.

Bayesian learning is another simplistic and practical machine learning algorithm. Its non-exclusionary behaviour based on input suits the potential noise that may be found in data derived from human behaviour. Its weakness in potentially requiring large amounts of computational horsepower is not taken into consideration under the scope of this project and so not a problem. A naïve Bayes classifier ought to be suitable for categorising documents given a keyword based query. This particular Bayesian learning algorithm is preferable as the potential hypotheses space could be very large due to the high number of document / query features, and unlike other Bayesian algorithms, it does not attempt to search through hypotheses space.

Instance based learning is an example of machine learning algorithms that are classified as “lazy”. These are distinguished by the way that they react simply to stored training data. There is no dealing with anything outside of the known training set.

The k-Nearest Neighbour algorithm can be tolerant to noisy data (which is something we expect to encounter) and again, relatively simple. It should also be possible to gain some insight into how its decisions are made, if required. K-Nearest Neighbour is however susceptible to situations where there are a large number of attributes but only a small proportion of them are relevant to the problem at hand; to get over this, attributes can be weighted differently, thus reducing the influence of less important ones by reducing their axes.

All of these algorithms are easily accessible using the Weka software [15]. Data will need to be translated into `arff` format before loading it into this system, and experiments can then easily be conducted to test the performance of each system.

## 3.3 Corpus usage

A readily available corpus for inclusion in the project will be the product database of www.digital-cameras.com. Access has been granted to this data, as well as being able to alter and log the behaviour of the site search facility. Further, this corpus has been indexed by Google and other commercial web search engines, which may provide a basis for performance comparison.

Other standard corpora to be used could include, the complete set of RFC documents (from www.rfc.org), and a variety of standard IR reference corpora. The TREC corpora could provide a useful sample; we will examine them to see if there is a set of suitable size that provides useful results.

In the end, the following collections were used. These are included with the SMART information retrieval system, and available via anonymous FTP at ftp://ftp.cs.cornell.edu/smart/. A corpus is a set of documents; a collection is a corpus and some additional data.

| Name | Documents | Queries | Description |
|---|---|---|---|
| ADI | 82 | 35 | Very small |
| CACM | 1587 | 64 | Titles (and sometimes abstracts) from the CACM journal |
| CISI | 1460 | 112 | Titles and abstracts on information science, indexing and libraries |
| CRAN | 1400 | 225 | The ASLIB Cranfield II collection [5] |
| MED | 1033 | 30 | Abstracts of articles from a medical journal (MEDLINE) |
| TIME | 423 | 83 | Articles from Time magazine in 1963 |

**Table 3.1 –** Reference collections

These are mainly in a similar format and comprise:

- Documents – a file containing all the documents
- Queries – a set of reference queries, often quite verbose and in natural language, describing requests for information against the document set
- Query relevance judgements – a set of document and query tuples, describing which documents are deemed relevant to each query.

Some collections incorporate a degree of relevance to each query for each document/query tuple. This is particularly useful when testing a ranking problem; a Boolean "relevant or not relevant" judgement could lead to an overwhelming number of results being retrieved from queries on large document sets. However most collections simply declare this Boolean status, and rather than discard the bulk of testing data, we will attempt to train various algorithms to learn Boolean functions.

### 3.3.1 Collection notes

The only corpora that display any kind of variation in case are CISI and CACM. All the text in others is entirely lowercase, apart from Time, which is all in uppercase. Thus, the `avgcase` field for these corpora will be static across each document, and the case deviation features will always be zero.

The Time stopword file is comprehensive and has been used in the processing of all corpora. An alternative candidate would be Salton's [28]. Some other collections define their own stopword data, but one list of stopwords was used over all collections for consistency's sake. This may lead to problems comparing anything but the Time corpus with other studies. That said, the application of tf.idf to keyword-based weights should make the impact of any words that slip through the net very low. CACM was the only other collection with a supplied stopword list.

The CACM corpus contains only 1599 documents with words, out of a total 3213 documents (49.8%). The documents without words do contain titles.

Instead of simply stating which documents are relative to a query, the Cranfield II collection goes further and provides an assessment of graded relevance. This is set into one of four classes, ranging from 1 which signifies a complete answer, to 5 which says the document is irrelevant to the query. These were manually derived [5,6]. A complete guide to the gradings can be found in Appendix B.

The Time corpus has an unusual format, providing simply raw text for articles, which are entirely in upper case. There are no titles or abstracts. Queries are a simple string in upper case; only an identifying number is provided.

## 3.4 IR techniques used

An IR system that uses tf.idf, stopword removal, vector space calculations, overall case and position data, a forward and reverse index, and possibly also keyword weightings (see 2.1.5) will be implemented for testing ML performance after training data has been captured. This system will not implement hit lists as in [21], as there should be more than enough data to conduct an initial survey of learning algorithm behaviour without spending excess time coding this feature.

## 3.5 Issues with using a classification approach

To simplify the experiment, the ranking problem has been reduced to a classification one. There are some issues with taking this approach [24]. The inherent structure in the ranks of the classes is lost when data is sorted into seemingly uniform boxes (the classes). For example, if we are to rank films as "good", "mediocre", and "awful", a ranking algorithm would be aware that good is positioned above mediocre, and that mediocre supersedes awful. The knowledge that a very highly ranked mediocre film could potentially be classified as good is

completely lost when the problem is treated as a classification one, as no information about relationships between classes is presented.

Further issues with the classification approach occur when, for example, a large number of documents are returned in a single class for a search. In this case, some kind of internal ordering is required, to avoid the result set being an unmanageable mass of probably-relevant documents. This is more likely to be an issue as the corpus grows, and especially with broader queries (such as those containing a single phrase).

A primary classification boundary can be found in the top two results; these are those that users focus on most as mentioned in [7]. The class representing the most relevant documents should correspond to this esteemed position, and perhaps have fewer documents assigned to it.

## 3.6 Multi-phrase queries

The only similarity measure mentioned so far that directly attacks the problem of dealing with multi phrase queries is the vector space model. A query phrase can be defined as a word, or a sequence of words contained in quotes, as part of a query. For example, `cat` is a single-phrase query; `cat "black hair"` has two phrases; `black haired cat` has three phrases; and `"black cat hair"` has one. Any phrase containing more than one word should be search for as a whole string – this does inhibit the use of thesauri as they don't often index alternative for multiword phrases.

A straightforward way of tackling multi phrase queries would be to do a search for each phrase, and then weight the values found in each search according to the tf.idf value for that phrase, and sum the end result. This would take term rarity / frequency into account and should be a fairly simplistic measure to implement.

## 3.7 Evaluation of ML methods

The evaluation of learning algorithms should also be given some attention, as well as seeing how well the IR systems they will back perform. Metrics available include learning speed based on the number of examples required, final accuracy, and relative error. A high final accuracy is the main desired result. This could be achieved by adjusting the parameters the learning algorithm, and also by tuning the representation of the problem.

Learning algorithms can also be evaluated using some of the IR metrics described above, namely precision, recall, and therefore also F measure.

For Boolean classification problems, IR precision will be equivalent to the ML precision of the algorithm; that is, the proportion of the results correctly classified will be equal to those returned as relevant.

PAC learning suggests a number of training examples that a ML algorithm should probably learn and approximate hypothesis from. This concept could also be used to provide an objective benchmark for the volume of training examples required to reach certain performance levels. It may also be interesting to see how the explicit and implicitly gathered data differ in the speed and quality of learning and system performance.

# 4. Design

## 4.1 Processing reference collections

The reference collections mainly follow a set format. There are at least three files; one containing documents, one containing natural language queries, and another associating the queries with documents deemed relevant.

### 4.1.1 Document file

Each line contains either a metadata flag (if it begins with a full stop) or data. The metadata flags are used to signify what type of information is about to follow. The various fields available are:

| Field | Description |
|---|---|
| I | Document ID |
| T | Title |
| B | Book or journal that the data is from |
| A | Author |
| K | Keywords |
| N | Timestamp information relating to corpus creation |
| W | Words – the main body of the document |
| X | Encoded link and citation information, found in the CACM corpus |

**Table 4.1 –** Fields available in reference collections

Not all of these are present in all corpora, and some are missing across a corpus; for example, the Cranfield collection does not specify timestamp information, and CACM has many documents that contain no “words” section. Example entries from a few corpora can be seen in Appendix D.

A set of data (independent of any specific words) can be defined for any text document, describing features of that document. These are numeric and easy to automatically derive. The ones generated by the system and used in representations of the problem are as follows:

| Feature name | Description |
|---|---|
| `numc` | Number of characters in the document |
| `numw` | Number of words in the document |
| `nums` | Number of sentences in the document |
| `nump` | Number of paragraphs in the document |
| `avgwlengthc` | Average word length, in characters |
| `avgslengthw` | Average sentence length, in words |
| `avgslengthc` | Average sentence length, in characters |
| `avgplengths` | Average paragraph length, in sentences |
| `avgplengthw` | Average paragraph length, in words |
| `avgplengthc` | Average paragraph length, in characters |
| `avgcase` | The average case of the document |

**Table 4.2 –** Document features

Texts are converted into UTF8 Unicode before being processed, using the Unix line break convention of just a single newline (\n). Tabs are replace with whitespaces before processing, and any runs of multiple whitespace characters reduced to just a single space. For the purposes of the above table, definitions are as follows:

**Character** – a single byte

**Word** – a sequence of non-space non-punctuation characters

**Sentence** – a sequence of words, separated by spaces, and terminated by either a full stop or EOF (end of file)

**Paragraph** – a sequence of sentences or words, terminated by a newline followed by a space (this convention is adopted across all common-format corpora; the Time corpus is hardly formatted at all and does not include any paragraph breaks).

Average case is calculated by assigning a weight to both uppercase and lowercase characters, and multiplying these by the frequency of each type of character in the document, divided by the total number of characters (e.g. a simple weighted average). The defaults applied are 1 for lowercase and 3 for uppercase – intuitively, uppercase characters tend to have a lower frequency (there are only two in this paragraph) and so get a "boosting".

Other features that could be examined could include reading ease and further gradings of a discourse. These haven't been included in order to keep the problem simple, although they shouldn't get in the way of most machine learning algorithms. Only fields that are relevant and useful are ever used.

### 4.1.2 Query file

The query is in a similar format to the document file, and consists of query identifiers and the text of the query. They are in natural language and often contain many stopwords or extraneous data. Some ask multiple questions; for example, from the CISI set:

```
.I 36
.W
What are some of the theories and practices in computer translating
of texts from one national language to another?  How can machine
translating compete with traditional methods of translating in
comprehending nuances of meaning in languages of different
structures?
```

### 4.1.3 Query / relevant documents file

This file and contains a set of lines, each with a query ID, then a document ID, and possibly some relevance judgements. In the case where multiple documents match a query, there are multiple lines with that same query ID, and differing document IDs.

## 4.2 Word features

Each document contains a set of words (possibly an empty set in some cases). The important words from here can be described by a set of features, relating them to their document. The ones that have been chosen for this study are:

| Feature name | Description |
|---|---|
| kfreq | The frequency of the keyword |
| kdensity | The fraction of words in the document that are the keyword (kfreq / numw) |
| kfirstpos | Position of the first occurrence of the keyword, stored as a fraction of characters into the document |
| kavgcase | Average case of the keyword |
| absavgslengthw | Absolute average sentence length of sentences containing the keyword, in words |
| devavgslengthw | Deviation of absavgslengthw from average sentence length |
| absavgwlengthc | Absolute average sentence length of sentences containing the keyword, in characters |
| devavgwlengthc | Devavgslengthw in characters instead of words |
| kpararatio | Fraction of paragraphs containing the keyword |
| avgsposw | Average position in sentence of the keyword, in words |
| avgsposc | Average position in sentence of the keyword, in characters |
| avgposinprelatives | In paragraphs containing the keyword, the average ordinal of the sentence it occurs in |
| avgposinpabsolutes | In paragraphs containing the keyword, the average position of the sentence it occurs in, as a fraction of the paragraph |
| avgposofprelativep | The average ordinal of paragraphs containing the keyword |
| avgposofpabsolutep | The average position of paragraphs containing the keyword, as a fraction of the document |
| kdensityinksent | The density of the keyword in sentences containing it |
| ksentratio | The fraction of sentences over the document that contain the keyword |

**Table 4.3** – Document:word tuple features

Other features could include the vector proximity of the document to a query.

## 4.3 Describing positive training examples

An information retrieval system will have at its disposal a query (in its simplest for, a text string) and a set of documents. The goal is to return a set of these documents that should be relevant to the query. A reference collection provides a set of documents, a set of queries, and also associations between each query and documents deemed relevant. It could therefore be said that each relevant document returned for a query, coupled with that query, could be used as a positive training example.

One way would be to take the document features that are independent of any keyword and add them to the training examples (see table 4.1). If there are underlying document features that show a relevant document, this should help them get picked up, provided they are in the calculated set. If not, these features should be ignored; in the case of e.g. a decision tree, they will present low information gain and not be used as a major indicator of final classification. It's up to the ML algorithms to decide what to ignore.

Secondly, the keyword information of keywords used in the query and occurring in the document can be put to use. This would involve finding which words occur both in query and document, and then retrieving the pre-calculated vectors for document:keyword features.

As each training example has have the same number of fields, the set of vectors needs to be reduced to just one. This is trivial in the case of having just one matching keyword – simply add them verbatim to the example – but where there are two or more, they need to be combined. A statistical mean would be the most straightforward approach. One with more balance would be to weight the vectors by the tf.idf value for the keyword they represent; this would result in words that occur a lot over the corpus having a reduced weight, and rarer ones influencing the final training example more.

The final piece of data to add to each training example is the classification. As most of the corpora provide only links to relevant documents, when mixed with irrelevant ones, the classification problem is a Boolean one. However it is impossible to train a system to distinguish between two classes if only one class is provided in training examples. Thus, a set of negative examples is required. To provide a fair amount of data, the same number of negative and positive examples are used in most cases.

## 4.4 Document and query based features

Each document text in the index will have a certain number of inherently present measurable attributes. For example, we can measure the average word length, the average case (see 2.1.8), the average sentence length, the number of paragraphs, the total document size, and so forth. These can be recorded and stored in an index, and presented in examples to see if they are of any use.

Values can also be derived on a document given a single word. The frequency of the word in the document, the position of the first occurrence of the word, the proportion of words in the document that match, and the average case of the word in the document are all readily available. Queries containing multiple words can also have a useful vector proximity value associated with each document (see 2.1.1). Each of these metrics is also known as a feature.

## 4.5 Use of non-text metrics

The corpora available may also include metadata about each document – that is, information associated with the document other than the document content. For example, a product database will contain pricing and availability values, as well as perhaps size and weight information. As long as these can be translated into a format readable by the learning algorithms, they can be included. However, the primary focus is on classifying documents based on their words. A proposal for inferring relationships between queries and non-text document metrics is outlined in 6.2.5.

## 4.6 Base accuracy

Base accuracy, for the content of this document, is defined as the accuracy of an algorithm if it outputs the most common classification available for every example provided to it. That is, if training data consists of nine examples of class A and one of class B, the algorithm will always return “A” and thus have a base accuracy of 90% over the training data.

## 4.7 Experiments run

### Objectives

There are a number of goals:

1. Generate training data

2. Calculate basic performance of a system

3. Observe the query formulation process

4. Measure how accurate retrieved results are

5. Train a machine learning system to accurately classify documents

While a manual study (see 6.2.2) would be a good way of collecting this data, it is time consuming and difficult to coordinate, and may not be comparable with other related work in the field. Thus, reference collections have been used to fulfil 1 and 5, and also to help measure 4. Finding out how good humans are at building queries – goal 3 – is not heavily on the topic of machine learning and information retrieval, and while interesting, will not be covered. Goal 2 is effectively similar to goal 5 if a machine learning system is to power results and performance is measured automatically; if subjective human feedback were available, this could be used instead.

### ML Testing metholodogy

Each evaluation of an algorithm's performance is repeated three times, using a 66% train/test split. That is to say, approximately two thirds of the data available is used to train the algorithm. The remaining part is used to test the trained algorithm, by supplying the feature set from each example and seeing if the class was correctly estimated. Every iteration of an evaluation uses a different mix of examples in the training and test sets. This ensures that any rogue examples or combinations of training / test data are less likely to have impact on the final result.

The value we are interested in is the accuracy of the trained algorithm when classifying items in the test set. A high accuracy indicates a better learning of the problem. It is important to note that an accuracy of 100% indicates an ability to completely learn the set of examples provided, but does not show that the system is perfect. In the case where a collection is only of a limited size, if an accuracy of 100% is consistently achieved, it may be possible to show that the query relevance assessment method for that collection, but it is not right to say that other unseen documents will also be classified accurately (although if the training/test sample size is big enough and the relevance decision methodology remains the same, there may be a strong chance of correct classification).

It is possible to adapt so well to a set of training and validation data that unseen examples become wrongly classified. This is known as "overfitting" (see 2.4.6).

We can also measure some IR metrics of the trained system, namely precision and recall (see 2.2.1). These can be measured from the trained algorithms as follows:

$$recall = \frac{number\ of\ \ examples\ correctly\ classified\ \ as\ relevant}{number\ of\ \ relevant\ articles\ according\ to\ external\ \ assessments}$$

**Formula 4.1 –** Recall of trained algorithm when working with reference collection

$$precision = \frac{number\ of\ \ examples\ correctly\ classified\ \ as\ relevant}{total\ \ number\ of\ \ examples\ classified\ \ as\ relevant}$$

**Formula 4.2 –** Precision of trained algorithm when working with reference collection

### 4.7.1 Basic comparison

The machine learning algorithms selected can be directly compared using WEKA's default parameters, by presenting them all with the same classification problem and training data, and measuring accuracy. This can be performed for all 6 test collections. The results expected should show which of the collections' relevancy measures are easiest to learn, and which algorithms provide best overall performance.

### 4.7.2 Boolean reduction

The first task is to reduce all reference collections to a Boolean classification. Only the Cranfield collection isn't already in this state. Instead, it has 5 classes of relevance. Some adjustment of the boundary of what is considered relevant and what not can be performed, to see which is easy to learn. The different classifications are described in Appendix B.

### 4.7.3 Adding metadata

Excess data such as "Authors" are ignored by our basic system, yet this data is provided by both the corpora and queries. This could provide an explanation for some of the cases where documents match queries seemingly containing no common words.

For example, a document from the CACM corpus:

```
.I 3078
.T
Analysis of the Availability of Computer
Systems Using Computer- Aided Algebra
.W
Analytical results, related to the availability
of a computer system constructed of unreliable
processors, are presented in this paper.  These results
are obtained by using various computer-aided
algebraic manipulation techniques.  A major purpose of
this paper is to demonstrate that the difficulties
of obtaining analytical solutions to Markov processes
can be considerably reduced by the application
of symbol manipulation programs.  Since many physical
systems can be modeled by Markov and semi-Markov
processes, the potential range of application of these techniques
is much wider than the problem of availability
analyzed here.
.B
CACM July, 1978
.A
Chattergy, R.
Pooch, U.W.
```

Should intuitively match a query searching for papers by Pooch, W.:

```
.W
All papers by this author
.A
Pooch, W.
```

But the test system created will not, as A (author) data is ignored (only the W field is used to populate the document body), and the non-stopwords in the query don't occur anywhere in the document. This issue also occurs with document titles. Again looking at the CACM corpus, many documents contain no 'body' text:

```
.I 115
.T
Optimizers: Their Structure
.B
CACM December, 1960
```

```
.A
Wheeling, R. F.
.N
CA601201 JB March 20, 1978  6:46 PM
.X
115   5     115
115   5     115
115   5     115
```

While in the relevance assessments for the collection this document matches queries such as “Optimization of intermediate and machine code”, the system implemented will again be unable to find a match. It may therefore be worth trying to index additional fields of the documents to see if the problem of learning relevance assessments becomes any easier. This will be tried primarily with the CACM corpus as it has a significant number of document entries that, using the default indexing system, will have an empty body.

### 4.7.4 Exclude empty documents

Empty documents may also be impeding the learning problem. It could be interesting to see how excluding them from the test corpus, and excluding queries that reference them, affects performance.

### 4.7.5 Vary hidden units

The type of neural net used in the experiment can have a number of “hidden” units. Each takes a number of weighted inputs and outputs a value based on these inputs. Typically, there is a layer of input units, one for each value in the input, followed by one or more hidden layers, consisting of an arbitrary number of units; these then all provide values to a final output unit that delivers classification.

The number of hidden units available may affect the accuracy of the final classification. A net with many hidden units also takes longer to train and test, thus affecting performance (which is of secondary concern). It could be good to see if a point exists where adding additional units does not increase final accuracy. Coupling this with the use of a validation set should provide a clear cut-off point, as no excess training will be performed, reducing the chance of any overfitting.

### 4.7.6 Vary learning rate and training time

Altering learning rate should affect how well a neural net can classify a test set after a fixed amount of training time. This can be coupled with a validation set to reduce overfitting, which may be more of a problem if a low learning rate is used, as this would allow closer adaptation to nuances of the training data.

### 4.7.7 Compare training set size with learning ease

Some corpora may be easier to learn than others, and some algorithms may cope better with a restricted set. It could be interesting to look at the final accuracy of algorithms across various collections, bearing the size of the training data, the number of documents in the collection, and the number of queries available in mind.

# 5. Implementation and testing

## 5.1 Presenting the problem

The problem of describing relevant and non-relevant documents needs to be represented in a machine readable form, one that can be recognised by machine learning algorithms. The target representation is therefore a vector of real-valued or nominal attributes.

WEKA – a tool used to test machine learning algorithms and data sets – uses the arff format for loading in data. This format is described in Appendix C and takes a header, specifying the order and type for the fields of supplied data, and then a data section, containing the training information itself.

## 5.2 Reference corpus processing

As the reference collections provide not only a document set but also a set of predefined queries and the documents they relate to, we have a set of data that describes a “relevant” document. In its raw format, this consists of a document, a query (often natural language), and some coupling data describing the relationship.

To process a text collection / corpus, the document file is read into a database table. Then, a reverse index is built, by iterating through each document and performing the following steps:

- Remove special characters such as punctuation and unrecognised character codes
- Fold excess spaces into one
- Build a list of all words in the document
- Remove any duplicate entries – word counts and positions are of no concern for this process
- For each word:
    - Add the word to a word list if not there already
    - Add the current document into the collection’s reverse index, under this word

This results in a list of non-stopwords being created, as well as a simple reverse index.

The next operation performed with the document texts is to calculate document features, as described in 4.4. These are calculated for each document and stored in another database table.

After the documents have been read and indexed, a set of features can be derived for each word, as also described in 4.4. This is often the most time consuming part of preparing a corpus, as it involves a noticeable number of string manipulation, array processing and mathematical operations on every unique non-stopword in every document in the corpus. The Time collection has over 92,000 of these relations.

## 5.3 Negative examples

Two methods exist for generating negative examples.

Negative examples are generated by selecting a random document and two random relations and generating a vector for them. This is equivalent to white noise – there is a chance that a useful relationship exists there, but it is unlikely. This method, whilst risking introducing conflicting data in some places, may be harder to learn.

An alternative method for generating negative examples is to find a query, and locate at random a document that is not in the set deemed relative. Then, the words common to the document and the query (if any) are looked up, and their features added to the training

example, as per the positive ones. This may provide a much easier problem to learn, as the training example generated may more often than not contain a lot of zeros for frequency figures.

The former method is used. It is less computationally intense to execute, and distinguishing relevant documents from noise is a better comparison to real world situations than distinguishing data which likely has mainly zero values as its feature vector.

## 5.4 No–relationship flag

Not all document/query pairs could be used as training data. Occasionally, no common words will be found between the query and document. When this happens, there will be no data to generate features from; so, the relationship is skipped over. This error indicates that the system cannot find a relationship between query and document, though external assessors have seen one and placed it into the collection. A weakness or lack of complexity in the system may be the cause. Possible remedies could increase the chance of terms in documents being matches to those in queries; for example, allowing synonymous terms to match, or counting words with common stems as equivalent.

## 5.5 Notes

The CACM query file also ends in a spurious .I 0; this led to an error about there being a lack of data for a query, and a message reporting the last ID loaded to be 0, possibly indicating a problem (this was of course a false negative).

# 6. Results

## 6.1 Findings

Full results from all experiments can be found in Appendix A. Every test is executed with randomly shuffled representations of the problems, different random seeds, and different data on each side of the test/training split. All results are calculated three times, but in this section only the mean is shown. Unless otherwise stated, equal numbers of positive and negative training examples are included. Neural nets are built using a fixed training time of 200 iterations (unless otherwise stated).

### 6.1.1 Basic comparison

Once all collections had been loaded, the default ARFF file was generated for each one. This contained feature representations of the query/document pairs present in the collection. An experiment was then set up in WEKA with the following machine learning algorithms and every corpus.

| Corpus | Examples | Positives | Negatives | Base Accuracy |
|---|---|---|---|---|
| **ADI** | 248 | 124 | 124 | 50.00% |
| **CACM** | 1044 | 522 | 522 | 50.00% |
| **CISI** | 5090 | 2545 | 2545 | 50.00% |
| **CRAN** | 1136 | 568 | 568 | 50.00% |
| **MED** | 1146 | 573 | 573 | 50.00% |
| **TIME** | 232 | 116 | 116 | 50.00% |

**Table 6.1** – Initial corpus ARFF setup

Results and configurations are shown below. The improvement is the gain over base accuracy yielded.

Naïve Bayes classifier:

Parameters: NaiveBayes

| Corpus | Mean accuracy | Improvement |
|---|---|---|
| **ADI** | 81.97% | 63.94% |
| **CACM** | 83.57% | 67.14% |
| **CISI** | 78.72% | 57.43% |
| **CRAN** | 72.51% | 45.01% |
| **MED** | 75.62% | 51.25% |
| **TIME** | 59.98% | 19.95% |

**Table 6.2** – Naïve Bayes initial results

C4.5 Decision Tree:

Parameters: J48 –C 0.25 –M 2

Explanation: confidence factor 0.25, minimum 2 instances per leaf

| Corpus | Mean accuracy | Improvement |
|---|---|---|
| **ADI** | 81.18% | 62.35% |
| **CACM** | 86.76% | 73.52% |
| **CISI** | 88.98% | 77.97% |
| **CRAN** | 82.19% | 64.39% |
| **MED** | 81.01% | 62.01% |
| **TIME** | 70.20% | 40.41% |

**Table 6.3** – C4.5 initial results

K-Nearest Neighbour:

Parameters: KStar –B 20 –E –M a

Explanation: use entropy-based blending, average out missing values – though there are none here, global blend of 20

| Corpus | Mean accuracy | Improvement |
|---|---|---|
| **ADI** | 70.59% | 41.18% |
| **CACM** | 72.30% | 44.60% |
| **CISI** | 69.51% | 39.02% |
| **CRAN** | 58.79% | 17.58% |
| **MED** | 68.95% | 37.90% |
| **TIME** | 63.39% | 26.79% |

**Table 6.4** – K* initial results

Neural net:

Parameters: MultilayerPerceptron –L 0.3 –M 0.2 –N 200 –V 0 –H a

Explanation: learning rate 0.3, momentum 0.2, train for 200 iterations, no validation set, use one layer of hidden units, with a size equal to the number of attributes plus the number of classifications

| Corpus | Mean accuracy | Improvement |
|---|---|---|
| **ADI** | 82.60% | 65.20% |
| **CACM** | 90.81% | 81.63% |
| **CISI** | 91.31% | 82.63% |
| **CRAN** | 84.65% | 69.30% |
| **MED** | 93.76% | 87.51% |
| **TIME** | 61.29% | 22.58% |

**Table 6.5** – Neural net initial results

Overall average results showed the CACM collection to have the easiest relevance assessments to learn, and the Time one hardest:

| Corpus | Base accuracy | Average improvement | |
|---|---|---|---|
| **ADI** | 50.00% | 58.17% | |
| **CACM** | 50.00% | 66.72% | max |
| **CISI** | 50.00% | 64.26% | |
| **CRAN** | 50.00% | 49.07% | |
| **MED** | 50.00% | 59.67% | |
| **TIME** | 50.00% | 27.43% | min |

**Table 6.6 –** Average accuracy of trained algorithms, by corpus

Also, the neural net turned out to be best at learning the problem on average. Decision trees were almost as effective, and a lot less processor intensive to train. This meant that experiments using decision trees could be carried out a lot more quickly.

| Algorithm | Base accuracy | Average trained accuracy | Average improvement |
|---|---|---|---|
| **Naïve Bayes** | 50.00% | 75.39% | 50.79% |
| **C4.5 decision tree** | 50.00% | 81.72% | 63.44% |
| **K* lazy** | 50.00% | 67.26% | 34.51% |
| **N** | 50.00% | 84.07% | 68.14% |

**Table 6.7 –** Average accuracy of trained algorithms, by algorithm

One notable exception was that the Time data – although hard to learn with all algorithms – did particularly badly with neural nets, offering only a 22% improvement. Also, the Cranfield II data was unusually tough for K*, yielding only a 17% improvement, almost one and a half standard deviations (s.d. = 10.24) below the mean for this algorithm.

### 6.1.2 Boolean reduction

Cranfield needs to be reduced to a Boolean classification from its initial 5-state document classification. The meanings of the states can be found in Appendix B, with 1 being “completely relevant” and 5 being “not relevant at all”. The algorithm used to examine this was WEKA’s J48 implementation of the C4.5 decision tree algorithm; the time taken to create trees was very low compared to that of neural nets and K*, and it exhibited a more consistent improvement with the Cranfield corpus.

The first stage was to generate 4 sets of training data, with classifications ranging from (1,2,3,4) as positive and (5) as negative to just (1) as positive. There are no examples of 5 in the collection, as they are implied for every relationship not given a value of 1-4. Thus, 100 random negative examples were also added to each training set.

| Class | Set 1 | Set 2 | Set 3 | Set 4 |
|---|---|---|---|---|
| **1** | Positive | Positive | Positive | Positive |
| **2** | Positive | Positive | Positive | Negative |
| **3** | Positive | Positive | Negative | Negative |
| **4** | Positive | Negative | Negative | Negative |
| **5** | Negative | Negative | Negative | Negative |

**Table 6.8 –** Cranfield negative/positive split training sets

As there is a fixed number of training examples available with every collection, the distribution of positive and negative examples varied depending on where the negative/positive split was placed. Initial results found there to be little difference from base accuracy:

| Set | Positives | Negatives | Total | Base accuracy | C4.5 accuracy | Improvement |
|---|---|---|---|---|---|---|
| 1 | 54 | 627 | 681 | 92.07% | 92.08% | 0.00% |
| 2 | 198 | 483 | 681 | 70.93% | 70.89% | -0.03% |
| 3 | 444 | 237 | 681 | 65.20% | 68.54% | 3.34% |
| 4 | 581 | 100 | 681 | 85.32% | 91.08% | 5.77% |

**Table 6.9** – Cranfield Boolean reduction initial results

It is easiest to see improvements when base accuracy is 50%, and may not be “fair” to provide unequal numbers of positive and negative examples. Thus, we can revise this experiment so that each training set has an equal number of positive and negative examples; this can be done by suppressing or inserting negative examples.

| Set | Positives | Negatives | Total | Base accuracy | C4.5 accuracy | Improvment | Artificial negatives |
|---|---|---|---|---|---|---|---|
| 1 | 54 | 54 | 108 | 50.00% | 55.03% | 5.03% | 0 |
| 2 | 198 | 198 | 396 | 50.00% | 50.37% | 0.37% | 0 |
| 3 | 444 | 444 | 888 | 50.00% | 72.71% | 22.71% | 307 |
| 4 | 581 | 581 | 1162 | 50.00% | 87.79% | 37.79% | 481 |

**Table 6.10** – Cranfield Boolean reduction with equal negative and positive examples

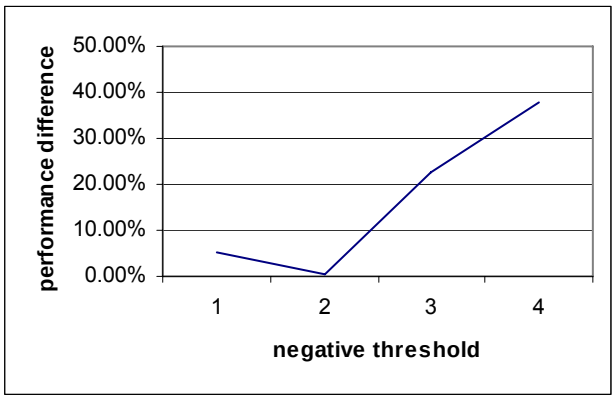


**Figure 6.1** - Cranfield Boolean reduction with equal negative and positive examples

A significant impact can be seen straight away, with the maximum difference reaching over 37%, compared to initial best results of under 6%. Providing equal numbers of positive and negative examples seems to have made the problem easier to learn overall, and there is a suggestion that less stringent requirements for positive classification may be better training examples.

However there is still not much improvement over base accuracy with sets 1 and 2. Set 1 has negative examples made entirely up of documents classified as being relevant to some degree (classifications 2, 3 and 4). These are ranked, and a document declared as having class 2 relevance to a query may have a very similar representation to one having class 1 relevance. That is, distinguishing class 1 from classes 2, 3 and 4 may be harder than distinguishing class 1 from a negative example (or class 5, in the case of Cranfield). Thus, the number of artificial

negatives can be counted for each of the above sets, and compared; there seems to be a relationship here with improvement.

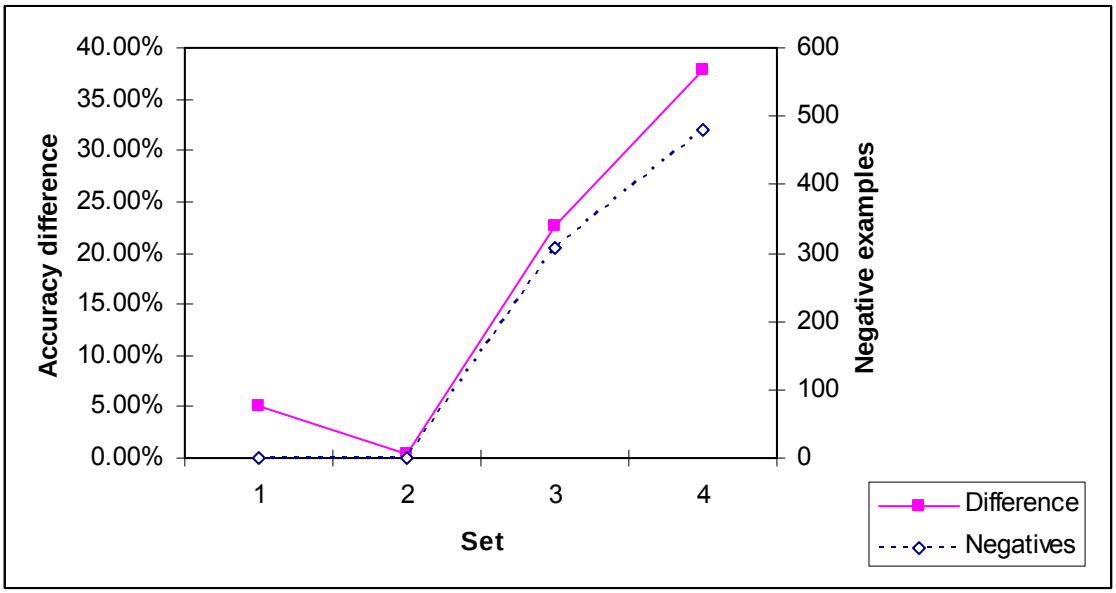


**Figure 6.2** – Cranfield Negative examples vs. accuracy

To test whether artificial negatives (e.g. class 5) are easier to learn than classes 2, 3, or 4 as negatives, the test was repeated, with equal numbers of negative and positive examples, and all negative examples being artificially generated. If this problem is easier, it may be an illustration of the ranking problem described in 3.5, where it is hard to declare the ordered relationship between ranked classes instead of their independence.

| Set | Positives | Negatives | Total | Base accuracy | C4.5 accuracy | Improvement |
|---|---|---|---|---|---|---|
| 1 | 54 | 54 | 108 | 50.00% | 84.43% | 34.43% |
| 2 | 198 | 198 | 396 | 50.00% | 85.85% | 35.85% |
| 3 | 444 | 444 | 888 | 50.00% | 87.40% | 37.40% |
| 4 | 581 | 581 | 1162 | 50.00% | 90.23% | 40.23% |

**Table 6.11 –** Cranfield, equal positive and negative examples with only artificial negatives

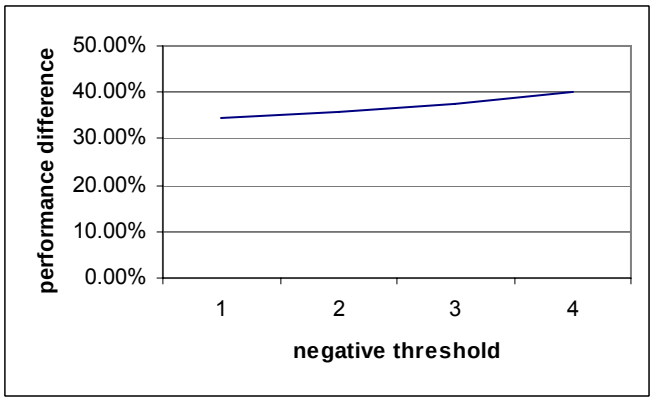


**Figure 6.3** - Cranfield, equal positive and negative examples with only artificial negatives

This problem was definitely easier to learn, with a marked improvement over base accuracy being seen in all sets. Accuracy gain went up as the threshold for negative examples moved

further away from 1. This may however have been due to the size of the training set increasing – Set 4 in this case is over ten times the size of Set 1. We can check this to some extent by limiting the amount of training data available.

| Set | Positives | Negatives | Total | Base accuracy | C4.5 accuracy | Improvement |
|---|---|---|---|---|---|---|
| 1 | 54 | 54 | 108 | 50.00% | 85.31% | 35.31% |
| 2 | 54 | 54 | 108 | 50.00% | 79.73% | 29.73% |
| 3 | 54 | 54 | 108 | 50.00% | 80.73% | 30.73% |
| 4 | 54 | 54 | 108 | 50.00% | 79.75% | 29.75% |

**Table 6.12** – Cranfield with limited training data

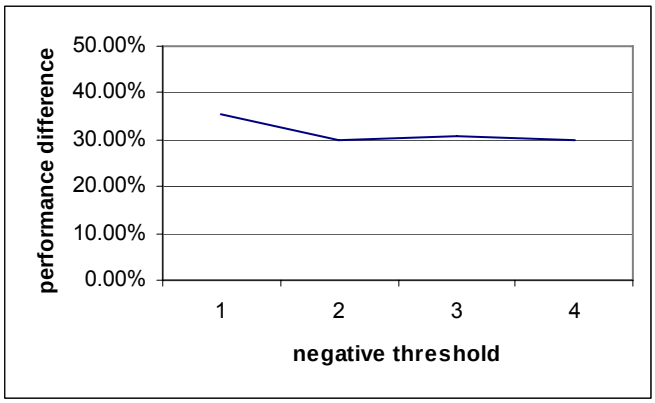


**Figure 6.4** – Cranfield with limited training data

Here, it seems that classifications of 1 vs. artificial negatives are easier to learn than anything else, though only slightly. This may be due to the strength of the highest relevance classification compared to the others. The experiment can be repeated, but limiting training data to the amount available for Set 2 (where possible) to verify this result – it seems weak as the amount of data is so low.

| Set | Positives | Negatives | Total | Base accuracy | C4.5 mean | Improvement |
|---|---|---|---|---|---|---|
| 1 | 54 | 54 | 108 | 50.00% | 88.11% | 38.11% |
| 2 | 198 | 198 | 396 | 50.00% | 84.87% | 34.87% |
| 3 | 198 | 198 | 396 | 50.00% | 87.10% | 37.10% |
| 4 | 198 | 198 | 396 | 50.00% | 81.15% | 31.15% |

**Table 6.13** - Cranfield with less limited training data

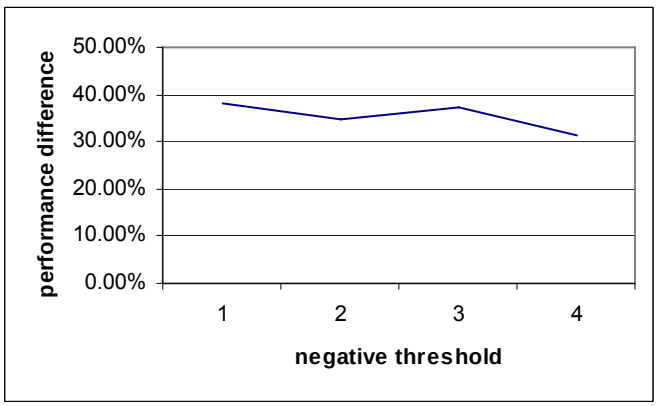


**Figure 6.5** – Cranfield with less limited training data

This reiterates the ease of classifying Set 1 previously found, especially considering it now has half the amount of training data available in other sets. Overall the result is not very strong, due to a lack of training data (a 66% split is used so only 36 examples are actually used to train in Set 1), and the standard deviation in results back up the weakness of the result – for Set 1 in the two last experiments, s.d. = 4.29 and s.d. = 4.00 respectively, which comes close to or encompasses results for the other sets. Thus, Set 1 may well be easiest to learn when compared to artificial data, but there's not sufficient evidence to strongly declare this.

### 6.1.3 Adding metadata

Documents exist with multiple optional fields aside from their main body. Queries can be created to match these optional fields, and don't even have to contain any text at all. As the simplified test system we will build will only cater for body texts, this could lead to queries being associated with empty documents, which may be hard to learn. It may be possible to measure any effect this has on the setup by allowing the system to also match extra metadata.

Titles are the most prevalent form of metadata present in documents. The CACM corpus in particular includes many documents that have no body (around half – see 3.3.1), Titles could be added to the system, perhaps prepended to the body text (with a separating space). Prepending titles would give different scores for keywords in the title on position-related metrics, as well as affecting overall keyword densities.

This would then allow the indexer to create entries for words in the title, which in turn could be matched against queries. Hopefully, more consistent training examples will be created from document/query matches involving some common text. Only the Naïve Bayes and C4.5 classifiers are used here, as they managed to perform reasonably without significant problems in the basic comparison, and are much quicker to run tests with in comparison to K* and neural net algorithms.

| Corpus | Accuracy | Without title | Difference |
|---|---|---|---|
| **ADI** | 68.29% | 81.97% | -13.68% |
| **CACM** | 72.22% | 83.57% | -11.35% |
| **CISI** | 70.27% | 78.72% | -8.45% |
| **CRAN** | 68.85% | 72.51% | -3.65% |
| **MED** | 75.69% | 75.62% | 0.07% |
| **TIME** | 59.98% | 59.98% | 0.00% |

**Table 6.14** – Performance of Naïve Bayes classifier using body text with titles added

| Corpus | Mean | Without title | Difference |
|---|---|---|---|
| **ADI** | 77.82% | 81.18% | -3.35% |
| **CACM** | 80.27% | 86.76% | -6.49% |
| **CISI** | 80.81% | 88.98% | -8.17% |
| **CRAN** | 83.01% | 82.19% | 0.82% |
| **MED** | 81.95% | 81.01% | 0.95% |
| **TIME** | 70.20% | 70.20% | 0.00% |

**Table 6.15** – Performance of C4.5 decision tree classifier using body text with titles added

No real performance increase can be seen here; if anything, there is a drop. Some, such as the Time corpus, have no title tags, and so performance remains static, as expected. The proportion of documents that have titles can be examined:

| Corpus | Naïve Bayes | C4.5 | Document count | Documents with titles | Proportion |
|---|---|---|---|---|---|
| **adi** | -13.68% | -3.35% | 82 | 82 | 100.00% |
| **cacm** | -11.35% | -6.49% | 1587 | 1586 | 99.94% |
| **cisi** | -8.45% | -8.17% | 1460 | 1460 | 100.00% |
| **cran** | -3.65% | 0.82% | 1400 | 1398 | 99.86% |
| **med** | 0.07% | 0.95% | 1033 | 0 | 0.00% |
| **time** | 0.00% | 0.00% | 423 | 0 | 0.00% |

**Table 6.16** – Proportions of documents with titles

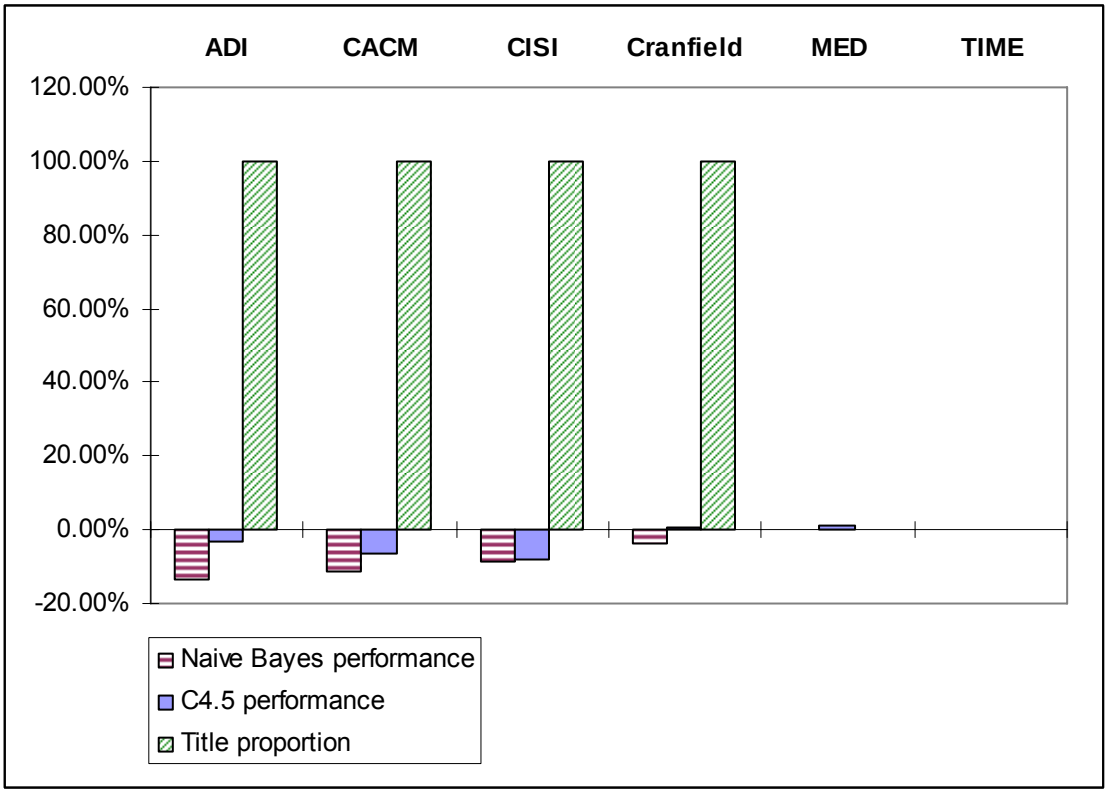


**Figure 6.6** – Accuracy at processing documents with additional titles vs. title usage

It may be worth experimenting with other machine learning algorithms in this setup. It could also be worthwhile assigning extra weight to the title text somehow. This could be crudely achieved by adding it in uppercase, under the current system (though this would have no impact on the entirely uppercase Time corpus, there are no titles there anyway). If a full HTML parsing system was written, the <title> tag content would be equivalent to the abstract titles given in reference corpora. There is a slight variance in MED's performance despite it having no titles, possibly due to differing sample selections.

### 6.1.4 Exclude empty documents

There are some documents that contain no body text. Notably the CACM corpus suffers from this notably, with around half of all document bodies empty. It may be easier to learn target classifications with links to empty documents removed. Any documents without bodies were not be indexed, and removed from query relevance lists. The CACM corpus was then re-indexed and its queries reloaded to form a new ARFF file. This was then run through each algorithm with default parameters.

| Algorithm | Accuracy | Unadjusted accuracy | Difference |
|---|---|---|---|
| Naïve Bayes | 68.70% | 83.57% | -14.87% |
| K* | 66.07% | 72.30% | -6.23% |
| Neural net | 90.63% | 90.81% | -0.19% |
| C4.5 tree | 72.54% | 86.76% | -14.22% |

**Table 6.17** – Performance with titles prepended to body text

A global drop in performance can be seen. This is significant with decision tree and Bayes classifier learners, which both drop their accuracy by over 14%. Thus, reducing the problem to exclude documents which intuitively do not contribute does not appear to be a good way of making the problem easier.

### 6.1.5 Vary hidden units

The number of hidden units in a neural net can affect its final performance. Having a high number of hidden units will also lead to very long real times for training and testing the net. It may be interesting to see where adding additional units has no further effect, by creating an experiment that uses neural nets with varying numbers of hidden units. The CACM and MED collections are learned well so far by neural nets, and are not enormous (and therefore time consuming to learn) when compared to others, e.g. CISI. The practice is to run these with a single hidden layer of between 1 and 20 units, for 200 iterations, with no validation set.

| Hidden layers | CACM accuracy % | MED accuracy % |
|---|---|---|
| 1 | 66.73 | 76.22 |
| 2 | 73.94 | 78.02 |
| 3 | 73.10 | 91.79 |
| 4 | 90.63 | 89.22 |
| 5 | 87.82 | 87.85 |
| 6 | 84.62 | 91.36 |
| 7 | 87.72 | 90.93 |
| 8 | 89.31 | 90.59 |
| 9 | 87.44 | 90.93 |
| 10 | 85.94 | 91.36 |
| 11 | 90.07 | 90.85 |
| 12 | 85.29 | 89.73 |
| 13 | 85.19 | 90.59 |
| 14 | 87.16 | 91.61 |
| 15 | 89.60 | 90.93 |
| 16 | 87.44 | 90.67 |
| 17 | 86.03 | 90.67 |
| 18 | 86.79 | 91.27 |
| 19 | 85.19 | 90.76 |
| 20 | 88.19 | 91.27 |

**Table 6.18** – Accuracy of a neural net while varying hidden layer size

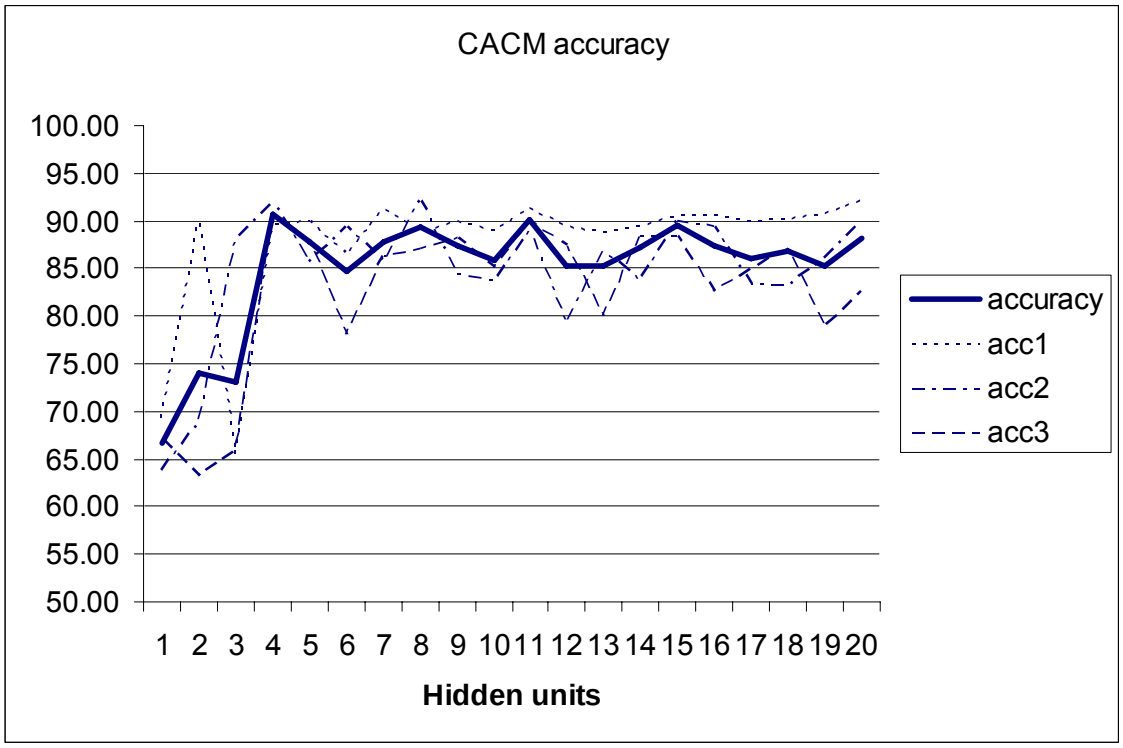


**Figure 6.7 -** Accuracy of a neural net while varying hidden layer size, with the CACM collection

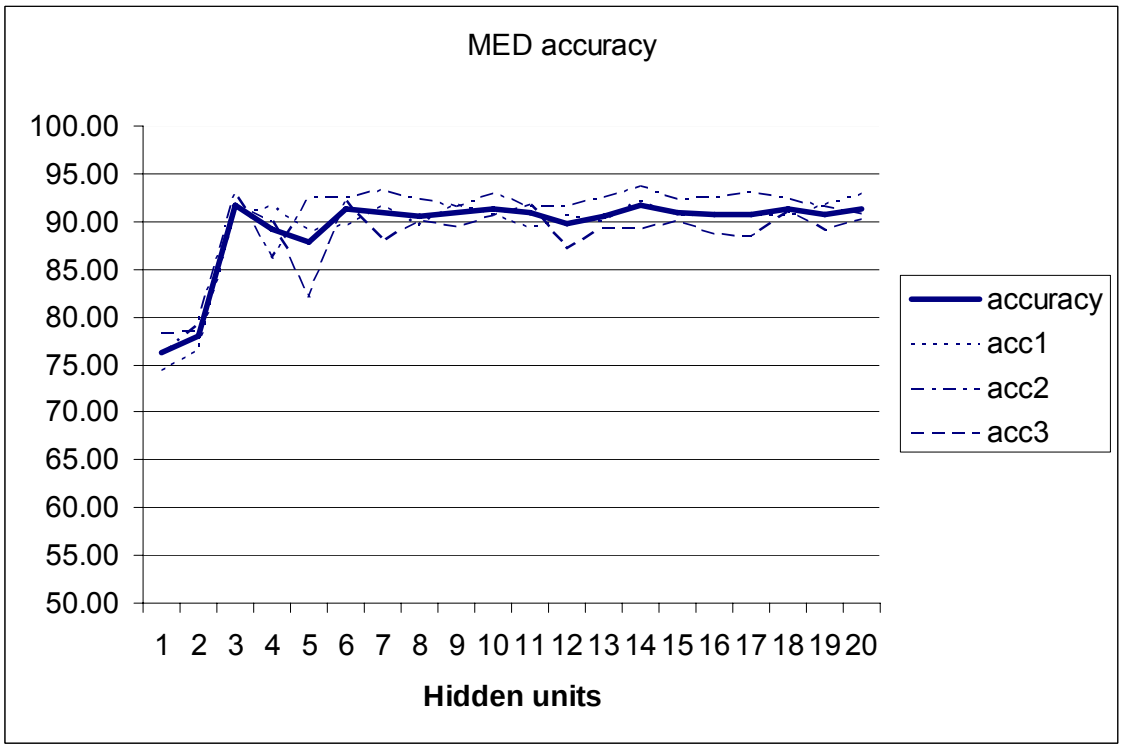


**Figure 6.8 -** Accuracy of a neural net while varying hidden layer size, with the MED collection

The accuracy with MED seems to peak after 3 units are added, and with CACM at 4 units, and doesn't get significantly higher. There are even some drops, perhaps as the algorithm overfits to the training data. Use of a validation set ought to remedy this. Running the experiment with

more than 3 iterations at each number of units may yield a smoother curve. MED's accuracy seems to be converging on a value around 91%.

### 6.1.6 Vary learning rate and training time

The learning rate affects the amount of change that can occur in a perceptron's weights per iteration. The training time determines how many iterations of backpropagation are run on the net before it is tested, in order to limit the amount of time taken to train. A lower training time should result in a less accurate net; a lower learning rate should require more training time to attain peak accuracy. We will use a validation set of 80 instances here to be able to better identify peak accuracy; training will stop when performance does not rise any more after 20 consecutive iterations. Hidden units were configured in a single layer of 10. Initially learning rate was set to 0.2. The MED corpus was the only one used here, as it has worked well with neural nets so far, and experimenting with every corpus would be extremely time consuming, with experiments taking days to run each time.

| Traintime | Accuracy |
|---|---|
| 5 | 61.68523 |
| 10 | 70.49217 |
| 20 | 72.03085 |
| 50 | 74.3381 |
| 100 | 77.41502 |
| 200 | 79.80819 |
| 300 | 79.2099 |
| 400 | 79.2099 |
| 500 | 79.2099 |
| 700 | 79.2099 |
| 1000 | 79.2099 |

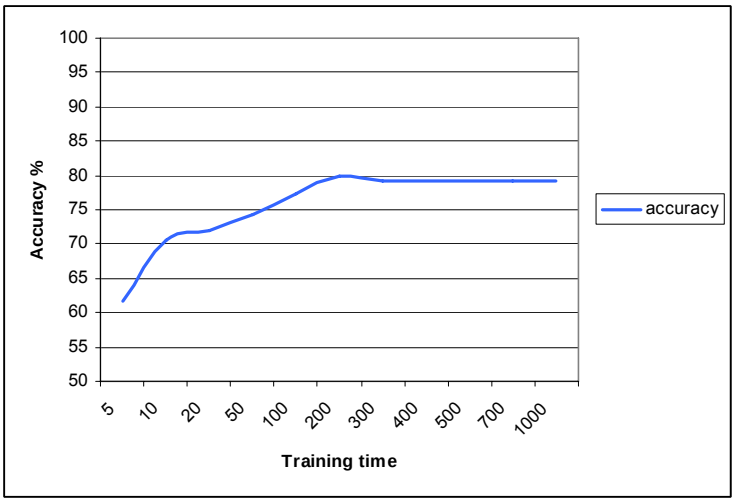


**Table 6.19 –** Accuracy with increasing training time, using a learning rate of 0.2.

Maximum consistent accuracy seems to have been reached by an epoch of 300; the validation set has kicked in here as the accuracy remains exactly the same for higher values. This is useful as we can now have evidence for restricting experiments to a low training time, thus saving real time.

| traintime | accuracy |
|---|---|
| 5 | 50.04285 |
| 10 | 54.66394 |
| 20 | 72.71417 |
| 50 | 72.88489 |
| 100 | 73.05561 |
| 200 | 74.42313 |
| 300 | 75.70518 |
| 400 | 75.70518 |
| 500 | 75.70518 |
| 700 | 75.70518 |
| 1000 | 75.70518 |

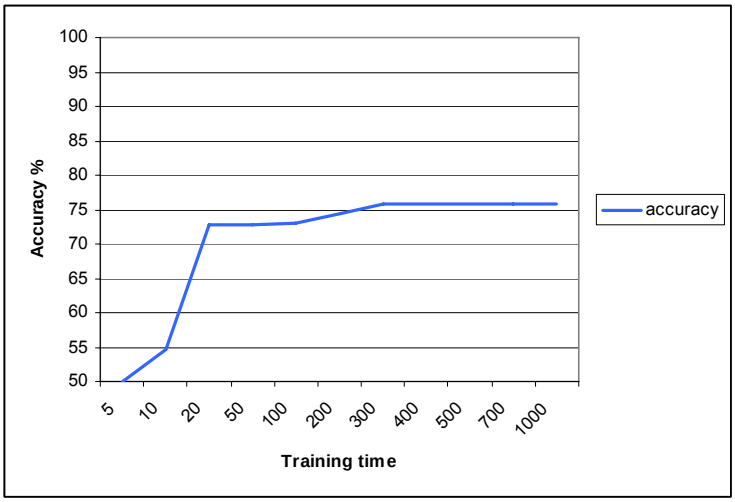


**Table 6.20 –** Accuracy with increasing training time, using a learning rate of 0.1

A decreased learning rate seems to impair the final accuracy, or at least, cause the current configuration of training set to stop training when accuracy is lower than that when the learning rate is 0.2. The point where the validation set kicks in is still 300 (or below) for this corpus. It may be worth extending the threshold for validation cut-off to more than 20 iterations, to se if a higher final accuracy can be reached.

### 6.1.7 Compare training set size with learning ease

The amount of training data available may affect the accuracy of the final system. We shall compare the accuracy improvement of all algorithms with the number of training examples available.

| Corpus | Examples | Naïve Bayes | C4.5 | K* | Neural net |
|---|---|---|---|---|---|
| **TIME** | 232 | 19.95% | 40.41% | 26.79% | 22.58% |
| **ADI** | 248 | 63.94% | 62.35% | 41.18% | 65.20% |
| **CACM** | 1044 | 67.14% | 73.52% | 44.60% | 81.63% |
| **CRAN** | 1136 | 45.01% | 64.39% | 17.58% | 69.30% |
| **MED** | 1146 | 51.25% | 62.01% | 37.90% | 87.51% |
| **CISI** | 5090 | 57.43% | 77.97% | 39.02% | 82.63% |

**Table 6.21 –** Corpus size vs. improvement offered in a trained system

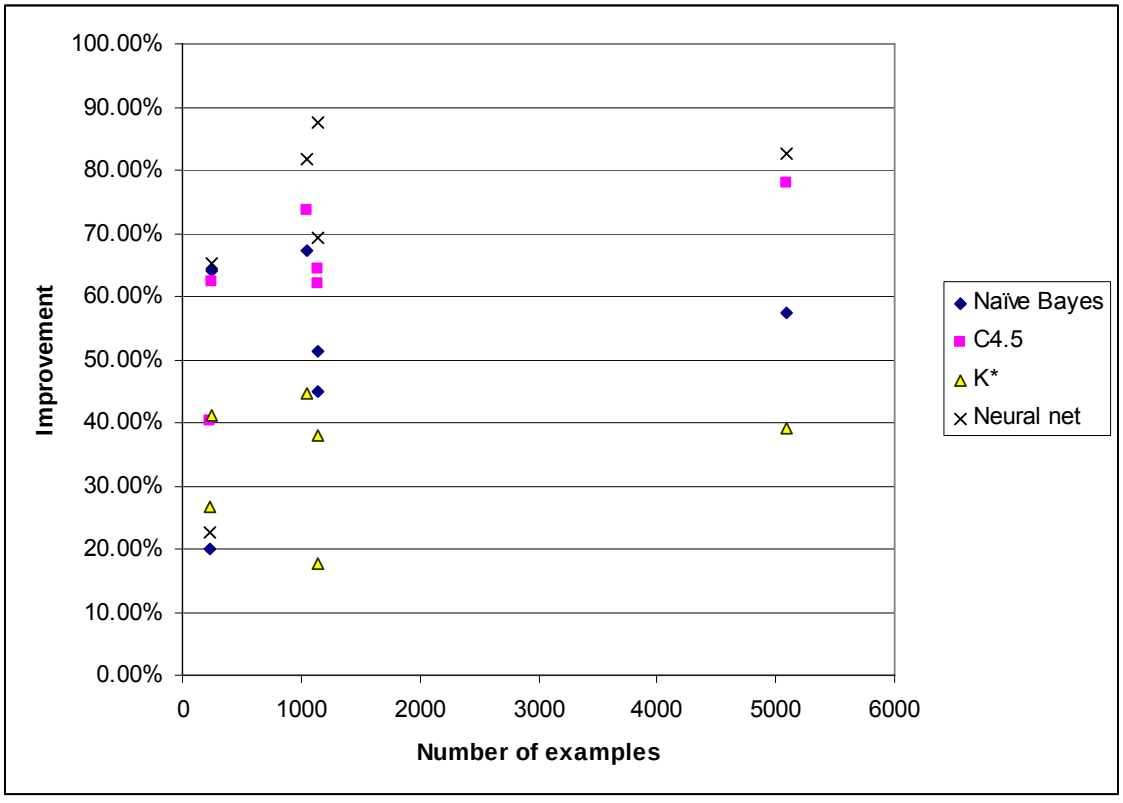


**Figure 6.9** – Corpus size vs. improvement offered in a trained system

There does not appear to be any strong relationship between the size of the training set and accuracy of the final system. While the minimum improvement with the largest sample is greater than that of other sized samples, the smallest improvement does not occur with the smallest corpora, and the greatest improvement does not occur with the largest ones.

Unfortunately, the sizes of datasets available seem to occur in three clusters; 230-250, 1000-1200, and CISI at 5090. To see a strong relationship here, results would have to be tightly grouped in each cluster, meaning all algorithms would have to perform equally. If we look at any one algorithm, there is not enough data to see a strong relationship. This study could be performed with a better populated range of training set sizes, and perhaps examine each algorithm individually.

## 6.2 Further work

### 6.2.1 Stemming

Applying a stemming algorithm to all queries and document bodies prior to indexing and ARFF file generation may result in a higher match rate, and reduced no-relationship messages (see 5.4).

### 6.2.2 Practical study

A substantial quantity of data is needed to thoroughly and effectively train and evaluate machine learning algorithms. Explicit data is expensive to gather, as mentioned above, as it involves user feedback. However implicit feedback is reasonably trivial to collect and can be used in place – or alongside – explicit human-originated data.

An issue with implicit data is that it is not 100% accurate; it suggests the opinion of the user but does not explicitly declare it. Thus, we need to know how accurate the implicit data is. Further, we need to be aware of any bias in the implicit feedback gathering mechanisms, and to be able to identify issues and test the setup. Also, some explicit – and therefore completely (or near-completely) accurate – data could be of great use when testing ML algorithms.

A study would involve testing some assumptions, measuring the performance of gathering implicit feedback, and should provide some authoritative data. It should also provide a good baseline for comparing against data gathered by other means.

### 6.2.3 Experiment ideas

**(i)** A basic task would be to ask participants to find a piece of data. They would search for it using an IR system logging the metrics described in [2], and would then be asked to classify each presented title and abstract into one of a limited number of classes. A potential set of classes for documents could be:

- Strongly relevant – similar to the top 2 documents, as they have special treatment
- Probably relevant – looks like it would contain helpful information
- Potentially useful – may be useful to the task, though the participant is unsure
- Irrelevant – definitely not useful to the task

The last class, “irrelevant”, indicates a document that should not have been recalled at all, and thus its presence in the result set shows a precision deficiency in the IR system.

**(ii)** An experiment that evaluates and provides feedback on how relevant a document is (as opposed to its abstract, as presented in results) could be useful. This would involve presentation of a results page, along with a keyword, and a classification task, whereby each abstract is judged for relevance; all documents should then (in any order desired) by opened and examined by the participant, perhaps in pop-up windows, and evaluated for their “real” relevance.

It may be possible – and tempting – for participants to approximate a judgement on the “real” document, or to misinterpret the task and not perform this second part correctly. To encourage following the correct message, the ability to enter feedback on the full document could only be

enabled after the document has been displayed and a small delay. Participants may also want to change their initial assessment of a document based on its abstract after examining it as a whole; this need to be prevented as the discrepancy is part of what the experiment measures. To this end, the element used to enter a classification for the abstract's relevance would be locked after the full document is opened.

**(iii)** A task where participants order results would also be good, and enable the use of some IR metrics. Participants would again be given an information-seeking task, and would then be presented with an initial ordering of candidate document abstracts. They would then enter their ideal ranking of these documents in descending order of relevance to the task. This would provide an IR system performance metric, to aid in evaluation, e.g. tau measure.

**(iv)** Participants could be given a task using an IR system, again given an information-seeking goal, where they enter a keyword, examine results, and then through interaction with the results page, navigate to the document they believe is best. This would be means of gathering comparative tracking data via common single click search behaviour. A "standard" set up, so the IR system will be web based, and the experiment conducted using common browsing software, screen resolutions and so forth. Participants are free to use the web in any manner they see fit, and they should just treat it as a normal browsing session, as it's important to capture data that is as natural as possible. We would be especially keen to capture events such as the use of the back button between results pages and documents, and query reformulation. To this end it may help for them to be unaware of the nature of the study, of what is logged, and even of being watched at all – some pretext for the experiment might help. Explicit feedback should be collected here, to validate the implicit data logged. This should help assess how accurate the training data mined later will be.

All these experiments only gather training data above a baseline untrained system; that is, they represent data on IR systems that have no prior knowledge of user preference or their own performance. Once a system is taught to provide results well, the trends in data gained from user behaviour on such a system may be different, as users are hopefully given more relevant results. Also, as the IR system provides better results, it should become easier to see any artefacts in the data that are independent of performance and perhaps more user related. For example, if a particular phrase is hard for users to read or accurately evaluate, this could hypothetically be interpreted as a performance deficiency for that phrase. As over performance improved, it may be discernable that there is in fact less of a problem with this phrase. Such deductions are however outside the scope of this project as the main concern is with the performance of learning algorithms compared with unintuitive IR, and not with identification of particular user behaviours in the gathered data.

### 6.2.4 Study practises

To ensure that the study isn't skewed badly by anomalous results, averages of at least 3 should be taken of any numeric measures derived from data collected. Noise and anomalies in the raw log data captured should not be considered a problem but rather a beneficial of what real-world data might look like and a test as to the resilience of the algorithms tested.

Participants are likely to get fatigued as the study progresses, and their behaviour could be susceptible to change. People are also likely to find it hard to concentrate on a repetitive task for a long period of time, which suggests that contiguous tasks should occur in shorter blocks. On the other hand, it will take some mental exertion to switch between tasks, so it's important to not make such blocks too short.

To help ensure an even level of concentration over the study as a whole, the experiments will be conducted one at a time, in random order. Inside each experiment, the series of individual tasks will be randomly ordered. This should help negate any bias caused by fatigue or particular questions / question orderings, as participants will approach experiments and tasks with evenly distributed levels of fatigue and bias from previous questions.

A good balance of classification possibilities in the training data gained from the study will help set the learning algorithms up to better classify documents. If only "relevant" and "very relevant" documents are returned, then examples of these two classes will be amassed, and the others will remain neglected. This results in leaving the system with impaired ability to classify items into other categories, and may lead to a lot of false classification into "relevant" and "very relevant".

To work around this, result sets used for classification oriented experiments within the study can be mixed with random and irrelevant abstracts. The results of the baseline IR system given a user query can be restricted to one part of the presented result set; a second part will be taken from randomly selected documents in the collection that contain the term used in the query (or, in the case of multiple word queries, any one of the words). A final part of the result set will then be taken from random documents that do not contain any part of the query, in the hope of getting some "irrelevant" examples.

These "mixed" results will be used in experiments (i), (ii), and (iv). Experiment (iii) does not attempt to classify documents but rather measure the performance of an IR system, and so should not be tampered with.

**Should users provide their own search phrases?**

All the experiments involve at least a query and a set of document abstracts. This set is often the result of submitting the query to an IR system as part of a larger task.

It would be possible to constrain the data by stipulating which search word is to be used. This could be done either with an arbitrarily selected word, or perhaps a query derived using an entropy related information gain metric (as in C4.5 / ID3, [18]) that attempts to use phrases most 'valuable' to the training set.

The initial inclination would be to present a task and let participants choose their own keyword, for the following reasons.

- An IR system should provide references to most useful document, not simply those closest matched to the terms provided in the query (the two may differ due to information loss during query formulation). The goal is to provide what a user wants, no how to simply match phrases.
- Participants may be frustrated by the inability to amend the search phrase user, especially if they consider it inappropriate.
- Unpredicted data and behaviour may be more likely to reveal unexpected effects and facts of the system / learning algorithms, especially when it comes from multiple people and not just the designer of the experiment.
- The experiment should be as close to a natural environment as possible. Noise in training data will help more thoroughly test ML algorithms. In the case of time-based metrics, the time taken to formulate and/or amend queries must be also included.

### 6.2.5 Handling prior knowledge

In some systems, intuitive decisions could be made by a human that are not automatically included in the ML implementation. An attribute labelled as "price" could have an effect on the ranking order if keywords such as "cheap", "bargain", or "premium" are found in the submitted query (given that the corpus and searcher are working in English). This kind of data could be hard coded into the system during implementation, but there are drawbacks to this approach. For one, a customised version of the system would be needed for each different corpus, and software maintenance will be required when the form of the corpus is altered. Secondly, hard coding this data reduces some of the ability of the learning algorithm to adjust results as it sees fit – the weighting of the association between attribute value and keyword used is not directly influenced by the learning algorithm and so harder to optimise. Also, the set of keywords that trigger the association may have to be manually controlled at implementation or, at best, by the IR system owners / maintainers.

Experimentally, to work around the disadvantages to implementing this kind of prior knowledge, some kind of mechanism that is capable of learning associations between keywords and attribute value ranges / tendencies. This would involve a mapping of keywords used in queries and the values of documents positively and negatively identified during the click session (see 2.3.2). One way of representing this mapping would be as a weighting on each feature found in the positive and negative documents. The average feature value across

the entire corpus would be computed. This would then be compared with the positively identified document and any differences stored with a keyword.

To avoid interfering too much with the IR function of the system too much by providing more general skewing, such as reinforcing that higher keyword density tends to provide favoured documents, the keyword/attribute weights should only be applied to metrics that are otherwise outside of the scope of the system. However when we encounter non-text attributes, some inference should be taken from these to draw as much value from the data available as possible. A field labelled “price” or “citations” would usually play no part in conventional text-based information retrieval; here we present an experimental approach for handling such fields.

The table below is based on document selection from the results of a query of “cheap shoe”.

For the term “cheap”:

| | **Price** | **Days since last update** |
|---|---|---|
| **Corpus mean** | 80.94 | 24 |
| **Selected document** | 19.99 | 22 |
| **Weights** | 0.947 | 0.217 |

**Table 6.22** – Keyword/attribute associations for “cheap”

Note also that a weight may accrued for the intuitively irrelevant attribute “days since last update”. To reduce the impact of such incidental weightings, the variance of the weight could be recorded, and then only those weights with sufficiently low variance would be applied when assessing documents for rankings. This ought to have the effect of eliminating any weight data gathered that is irrelevant, as those weights with high variance are likely to be of little use when biasing documents. Most ML algorithms should filter out less relevant attributes; for example, decision trees often use an entropy gain metric for this purpose

# 7. Conclusion

The field of information retrieval is mature and a lot of published work exists. It is relatively straightforward to build a simplistic IR system, through reverse indexing, and problems such as storage requirements and performance can be overcome with stemming, use of thesauri, and more efficient designs.

Work in information retrieval has provided reference collections which are ready made for the task of experimenting with IR. They can be compared to other works in the field and circumvent the need for collecting a set of documents. A collection also provides queries and relevance judgements, which again save time and effort.

These relevance judgements can be presented to machine learning algorithms, through numeric representations of the relationships between a query and documents. This representation set is a model of what kind of document is considered relevant. As our goal is to have a system learn to distinguish relevant documents from irrelevant ones, and many positive relations are identified in reference collections, all that is needed to complete a set of training data is inconsequential negative examples.

The basic machine learning algorithms tested are able to distinguish relevant documents from irrelevant ones using the initial feature representation. This shows that the representation carries across some values that change in a distinguishable way when a certain degree of relevance between document and query is reached.

Further, it has been shown that it is possible to increase the accuracy of machine learning algorithm when deciding relevancy. Adjusting the representation and the way that documents are indexed have an effect on the final quality of the trained system. Also, manipulating the parameters of the ML algorithm can affect both the time it takes to learn data and distinguish documents, and its ability to do so.

This approach is promising and much further work can be done in the field, especially around the interactions people have with IR systems. Much indirect data is generated and although some was captured by the study, not enough was available to provide directly significant results.

Initial results show that machine learning can provide an effective alternative to conventional IR systems, though many IR techniques can be used to help effectively represent the problem and increase final accuracy.

## Bibliography


1. Baeza-Yates R., Ribeiro-Neto B. (1999), Modern Information Retrieval, ACM Press Books, Addison-Wesley, Harlow, England.

2. Boyan, J., Freitag, D., Joachims, T. (1996), "A Machine Learning Architecture for Optimizing Web Search Engines", *Proceedings of the AAAI Workshop on Internet-Based Information Systems 1996, Portland, Oregon*.

3. Chomsky, N. (1990) – Language and Responsibility, in: Chomsky, N. *On Language*. The New Press, New York.

4. Cleary, J.G., Trigg, L.E. (1995), "K*: An Instance- based Learner Using an Entropic Distance Measure.", *Proceedings of the 12th International Conference on Machine learning*, vol. 12, 1995, pp. 108-114.

5. Cleverdon, C. (1960), *Aslib Cranfield research project: report on the first stage of an investigation into efficiency of indexing systems*, Cranfield, College of Aeronautics, Cranfield, England.

6. Cleverdon, C., Keen, M. (1966), *"Factors determining the performance of indexing systems. Vol.2: Test results"*, Cranfield, Royal Aeronautical College, Cranfield, England.

7. Granka, L. (2004), "Eye-Tracking Analysis of User Behaviour in WWW Search", *Proceedings of the 27th Conference on Research and Development in Information Retrieval (SIGIR) 2004*, vol. 27, 2004, pp. 478-479.

8. Hofmann, T. (1999), "Probabilistic Latent Semantic Analysis", *Proceedings of the 15th Conference on Uncertainty in AI (1999).,* vol. 15, 1999, pp. 289-296.

9. Ide, E. (1971), New Experiments in Relevance Feedback, in: Salton, G. (ed) *The SMART Retrieval System – Experiments in Automatoc Document Processing*, Prentice-Hall, Englewood Cliffs, NJ.

10. Joachims, T. (2002), "Optimizing Search Engines using Clickthrough Data", *Proceedings of the eighth ACM Conference on Knowledge Discovery and Data Mining (KDD).* vol. 8, 2002, pp. 133-142.

11. Joachims, T., Granka, L., Pan, B., Hembrooke, H., Gay, G. (2005), "Accurately interpreting clickthrough data as implicit feedback", *Proceedings of the 28th annual ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, vol. 28, 2005, pp. 154-161.

12. Kemp, C., Ramamohanarao, K., (2002), "Long-Term Learning for Web Search Engines", *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery,* vol. 6, August 19-23, 2002, pp. 263-274.

13. Kendall, M. (1955), *Rank Correlation Methods*, Hafner, New York.

14. Kirkby, R., Frank, E. (April 4th, 2006), "Attribute-Relation File Format", Waikzto, New Zealand, Available: http://www.cs.waikato.ac.nz/~ml/weka/arff.html (Accessed: January - May 2006)

15. Kirkby, R., Frank, E. (2006), "Weka 3: Data Mining Software in Java", Waikzto, New Zealand, Available: http://www.cs.waikato.ac.nz/ml/weka (Accessed: October 2005 - May 2006)

16. Lau, T., Horvitz, E. (1998), "Patterns of Search: Analyzing and Modeling Web Query Refinement", *Proceedings of the 7th International Conference on User Modeling*, vol. 7, June 1999, pp. 119-128.

17. Manning, C.D., Schűtze, H. (1999), *Foundations of Statistical Language Processing*, 6th edition, MIT Press, London.

18. Mitchell, T. (1997), *Machine Learning,* International edition, MIT Press, McGraw-Hill.

19. Mood, A., Graybill, F., Boes, D. (1974), *Introduction to the theory of statistics*, 3rd edition McGraw-Hill.

20. Oakes, M., Gaizauskas, R., Fowkes, H., Jonsson, A., Wan, V., Beaulieu, M. (2001), "Comparison Between a Method Based on the Chi-Square Test and a Support Vector Machine for Document Classification.", *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR01),* vol. 24, September 9-12, 2001, pp. 440-441.

21. Page, L., Brin, S. (1998), "The Anatomy of a Large-Scale Hypertextual Web Search Engine" *Proceedings of the Seventh International Web Conference*, vol. 7, April 1998, pp. 107-117.

22. Porter, M.F. (1980), "An algorithm for suffix stripping" *Program*, vol. 14, no. 3, 1980, pp. 130-137.

23. Radlinski, F., Joachims, T., (2005), "Query Chains: Learning to Rank from Implicit Feedback" *Proceedings of the 11th ACM SIGKDD International Conference On Knowledge Discovery in data mining*, vol. 11, 2005, pp 239-248.

24. Rajaram, S., Garg, A., Zhou, X.S., Huang, T.S. (2003), "Classification Approach towards Ranking and Sorting Problems", *Lecture Notes in Artificial Intelligence*, vol. 2837, 2003, pp. 301-312.

25. Rocchio, J.J., Salton, G. (1965), "Information Search Optimization and Interactive Retrieval Technique", *Proceedings AFIPS 1965 Fall Joint Computer Conference*, vol. 27, 1965, pp. 293-306.

26. Salton, G. (ed) (1971), *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice-Hall, Englewood Cliffs, NJ.

27. Salton, G. (1991), "Developments in Automatic Text Retrieval", *Science* vol. 253, 1991, pp 974-980.

28. Salton, G. (2002), "Stopword List 1", Provo, UT, Available: http://www.lextek.com/manuals/onix/stopwords1.html (Accessed: February - May 2006).

29. van Rijsbergen, C.J. (1979), *Information Retrieval*, Butterworths, London.

# Appendix A – Full Results

### Basic comparison

| corpus | examples | positives | negatives | base accuracy | avg improvement | |
|---|---|---|---|---|---|---|
| adi | 248 | 124 | 124 | 50.00% | 58.17% | |
| cacm | 1044 | 522 | 522 | 50.00% | 66.72% | |
| cisi | 5090 | 2545 | 2545 | 50.00% | 64.26% | |
| cran | 1136 | 568 | 568 | 50.00% | 49.07% | bool thresh = max |
| med | 1146 | 573 | 573 | 50.00% | 59.67% | |
| time | 232 | 116 | 116 | 50.00% | 27.43% | |

| naivebayes: | | | | 75.39% | 50.79% |
|---|---|---|---|---|---|
| corpus | acc1 | acc2 | acc3 | mean acc | improvement |
| adi | 77.67% | 75.29% | 92.94% | 81.97% | 63.94% |
| cacm | 85.92% | 81.13% | 83.66% | 83.57% | 67.14% |
| cisi | 79.20% | 78.73% | 78.22% | 78.72% | 57.43% |
| cran | 71.65% | 71.88% | 73.99% | 72.51% | 45.01% |
| med | 74.87% | 72.05% | 79.95% | 75.62% | 51.25% |
| time | 55.13% | 65.82% | 58.97% | 59.98% | 19.95% |

| c4.5 | | | | 81.72% | 63.44% |
|---|---|---|---|---|---|
| corpus | acc1 | acc2 | acc3 | mean acc | improvement |
| adi | 77.65% | 83.53% | 82.35% | 81.18% | 62.35% |
| cacm | 89.30% | 83.66% | 87.32% | 86.76% | 73.52% |
| cisi | 89.08% | 88.67% | 89.20% | 88.98% | 77.97% |
| cran | 81.70% | 81.25% | 83.63% | 82.19% | 64.39% |
| med | 82.82% | 82.56% | 77.63% | 81.01% | 62.01% |
| time | 64.10% | 72.15% | 74.36% | 70.20% | 40.41% |

| k* | (entropy blend on) | | | 67.26% | 34.51% |
|---|---|---|---|---|---|
| corpus | acc1 | acc2 | acc3 | mean acc | improvement |
| adi | 70.59% | 75.29% | 65.88% | 70.59% | 41.18% |
| cacm | 70.99% | 71.83% | 74.08% | 72.30% | 44.60% |
| cisi | 67.82% | 70.92% | 69.79% | 69.51% | 39.02% |
| cran | 60.71% | 58.71% | 56.95% | 58.79% | 17.58% |
| med | 68.46% | 68.46% | 69.92% | 68.95% | 37.90% |
| time | 56.41% | 65.82% | 67.95% | 63.39% | 26.79% |

| mlp | learntime = 200 | | | 84.07% | 68.14% |
|---|---|---|---|---|---|
| corpus | acc1 | acc2 | acc3 | mean acc | improvement |
| adi | 80.95% | 84.71% | 82.14% | 82.60% | 65.20% |
| cacm | 93.26% | 91.01% | 88.17% | 90.81% | 81.63% |
| cisi | 92.84% | 87.75% | 93.36% | 91.31% | 82.63% |
| cran | 85.71% | 83.26% | 84.98% | 84.65% | 69.30% |
| med | 91.28% | 94.36% | 95.63% | 93.76% | 87.51% |
| time | 62.82% | 58.23% | 62.82% | 61.29% | 22.58% |

**Cranfield negative threshold split**

all use j48 -c 0.25 -m 2, as it's quick to run, and picks up well

| negative threshold | positives | negatives | total | baseacc | c4.5 acc1 | c4.5 acc2 | c4.5 acc3 | c4.5 mean | difference |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 54 | 627 | 681 | 92.07% | 92.21% | 91.81% | 92.21% | 92.08% | 0.00% |
| 2 | 198 | 483 | 681 | 70.93% | 71.00% | 70.69% | 71.00% | 70.89% | -0.03% |
| 3 | 444 | 237 | 681 | 65.20% | 66.67% | 71.43% | 67.53% | 68.54% | 3.34% |
| 4 | 581 | 100 | 681 | 85.32% | 90.09% | 90.52% | 92.64% | 91.08% | 5.77% |

have trimmed or extended result sets to create 50/50 split of pos/neg examples

| negative threshold | positives | negatives | total | baseacc | c4.5 acc1 | c4.5 acc2 | c4.5 acc3 | c4.5 mean | difference | num artificial negs |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 54 | 54 | 108 | 50.00% | 63.89% | 56.76% | 44.44% | 55.03% | 5.03% | 0 |
| 2 | 198 | 198 | 396 | 50.00% | 49.63% | 50.00% | 51.49% | 50.37% | 0.37% | 0 |
| 3 | 444 | 444 | 888 | 50.00% | 71.19% | 74.17% | 72.76% | 72.71% | 22.71% | 307 |
| 4 | 581 | 581 | 1162 | 50.00% | 90.13% | 85.61% | 87.63% | 87.79% | 37.79% | 481 |

it may be harder to distinguish [1] vs [2,3,4,fail] compared to [1] vs [fail]. So, skip natural negs, and make them all artificial

| negative threshold | positives | negatives | total | baseacc | c4.5 acc1 | c4.5 acc2 | c4.5 acc3 | c4.5 mean | difference |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 54 | 54 | 108 | 50.00% | 86.11% | 81.08% | 86.11% | 84.43% | 34.43% |
| 2 | 198 | 198 | 396 | 50.00% | 88.89% | 88.81% | 79.85% | 85.85% | 35.85% |
| 3 | 444 | 444 | 888 | 50.00% | 85.10% | 88.41% | 88.70% | 87.40% | 37.40% |
| 4 | 581 | 581 | 1162 | 50.00% | 90.89% | 90.40% | 89.39% | 90.23% | 40.23% |

the reduced size of the training set may impede learning. Try at 54 for everything, and also 198.
write arff files in random order so that individual odd instances have reduced impact

| negative threshold | positives | negatives | total | baseacc | c4.5 acc1 | c4.5 acc2 | c4.5 acc3 | c4.5 mean | difference | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 54 | 54 | 108 | 50.00% | 80.56% | 86.49% | 88.89% | 85.31% | 35.31% | | 4.29% |
| 2 | 54 | 54 | 108 | 50.00% | 75.00% | 89.19% | 75.00% | 79.73% | 29.73% | | |
| 3 | 54 | 54 | 108 | 50.00% | 77.78% | 81.08% | 83.33% | 80.73% | 30.73% | | |
| 4 | 54 | 54 | 108 | 50.00% | 77.78% | 86.49% | 75.00% | 79.75% | 29.75% | | |

| negative threshold | positives | negatives | total | baseacc | c4.5 acc1 | c4.5 acc2 | c4.5 acc3 | c4.5 mean | difference | variance | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 54 | 54 | 108 | 50.00% | 88.89% | 83.78% | 91.67% | 88.11% | 38.11% | | 4.00% |
| 2 | 198 | 198 | 396 | 50.00% | 82.96% | 86.57% | 85.07% | 84.87% | 34.87% | | |
| 3 | 198 | 198 | 396 | 50.00% | 87.41% | 85.07% | 88.81% | 87.10% | 37.10% | | |
| 4 | 198 | 198 | 396 | 50.00% | 77.78% | 83.58% | 82.09% | 81.15% | 31.15% | | |

* = comment on the difference in these figures; should be the same, though files is randomised. Perhaps calculate CI

looks like we have insufficient data at relevance = 1.

**Inserted title**

nbayes

| corpus | acc1 | acc2 | acc3 | mean | without title | difference |
|---|---|---|---|---|---|---|
| adi | 68.42% | 63.83% | 72.63% | 68.29% | 81.97% | -13.68% |
| cacm | 74.40% | 72.22% | 70.05% | 72.22% | 83.57% | -11.35% |
| cisi | 69.26% | 70.61% | 70.93% | 70.27% | 78.72% | -8.45% |
| cran | 67.19% | 69.64% | 69.73% | 68.85% | 72.51% | -3.65% |
| med | 75.38% | 73.85% | 77.85% | 75.69% | 75.62% | 0.07% |
| time | 55.13% | 65.82% | 58.97% | 59.98% | 59.98% | 0.00% |

c4.5

| corpus | acc1 | acc2 | acc3 | mean | without title | difference |
|---|---|---|---|---|---|---|
| adi | 75.79% | 79.79% | 77.89% | 77.82% | 81.18% | -3.35% |
| cacm | 83.57% | 80.43% | 76.81% | 80.27% | 86.76% | -6.49% |
| cisi | 80.88% | 82.40% | 79.15% | 80.81% | 88.98% | -8.17% |
| cran | 84.15% | 81.25% | 83.63% | 83.01% | 82.19% | 0.82% |
| med | 80.00% | 80.26% | 85.60% | 81.95% | 81.01% | 0.95% |
| time | 64.10% | 72.15% | 74.36% | 70.20% | 70.20% | 0.00% |

| corpus | nb | c45 | totaldocs | docswithtitles | proportion |
|---|---|---|---|---|---|
| ADI | -13.68% | -3.35% | 82 | 82 | 100.00% |
| CACM | -11.35% | -6.49% | 1587 | 1586 | 99.94% |
| CISI | -8.45% | -8.17% | 1460 | 1460 | 100.00% |
| Cranfield | -3.65% | 0.82% | 1400 | 1398 | 99.86% |
| MED | 0.07% | 0.95% | 1033 | 0 | 0.00% |
| TIME | 0.00% | 0.00% | 423 | 0 | 0.00% |

**CACM – no empty**

| alg | acc1 | acc2 | acc3 | accuracy | unadjusted acc | diff |
|---|---|---|---|---|---|---|
| naivebayes | 71.63% | 66.57% | 67.89% | 68.70% | 83.57% | -14.87% |
| k* | 64.33% | 67.42% | 66.48% | 66.07% | 72.30% | -6.23% |
| mlp | 90.73% | 91.57% | 89.58% | 90.63% | 90.81% | -0.19% |
| c4.5 | 68.54% | 72.47% | 76.62% | 72.54% | 86.76% | -14.22% |

**NN – hidden layers**

chose medline and cacm as these worked best with nn in the basic comparison

| cacm | traintime = 200, | | | |
|---|---|---|---|---|
| hiddenlayers | acc1 | acc2 | acc3 | accuracy |
| 1 | 69.38202 | 63.76405 | 67.04225 | 66.73 |
| 2 | 89.88764 | 68.82023 | 63.09859 | 73.94 |
| 3 | 65.44944 | 87.92135 | 65.91549 | 73.10 |
| 4 | 89.60674 | 91.85393 | 90.42254 | 90.63 |
| 5 | 89.88764 | 85.67416 | 87.88732 | 87.82 |
| 6 | 86.51685 | 89.32584 | 78.02817 | 84.62 |
| 7 | 91.29214 | 85.67416 | 86.19718 | 87.72 |
| 8 | 88.76405 | 92.13483 | 87.04225 | 89.31 |
| 9 | 89.88764 | 84.26966 | 88.16901 | 87.44 |
| 10 | 89.04494 | 83.70787 | 85.07042 | 85.94 |
| 11 | 91.29214 | 89.04494 | 89.85916 | 90.07 |
| 12 | 89.32584 | 79.21348 | 87.32394 | 85.29 |
| 13 | 88.76405 | 86.79775 | 80 | 85.19 |
| 14 | 89.32584 | 83.70787 | 88.4507 | 87.16 |
| 15 | 90.44944 | 89.88764 | 88.4507 | 89.60 |
| 16 | 90.44944 | 89.32584 | 82.53521 | 87.44 |
| 17 | 89.88764 | 83.42697 | 84.78873 | 86.03 |
| 18 | 90.16854 | 83.14607 | 87.04225 | 86.79 |
| 19 | 90.73034 | 85.95506 | 78.87324 | 85.19 |
| 20 | 92.13483 | 89.88764 | 82.53521 | 88.19 |

| medline | tt = 200 | | | |
|---|---|---|---|---|
| hiddenlayers | acc1 | acc2 | acc3 | accuracy |
| 1 | 74.35897 | 76.15385 | 78.1491 | 76.22 |
| 2 | 76.41026 | 79.23077 | 78.40617 | 78.02 |
| 3 | 90.76923 | 93.07692 | 91.51671 | 91.79 |
| 4 | 91.53846 | 86.15385 | 89.97429 | 89.22 |
| 5 | 88.97436 | 92.5641 | 82.00514 | 87.85 |
| 6 | 89.48718 | 92.5641 | 92.03085 | 91.36 |
| 7 | 91.53846 | 93.33333 | 87.91774 | 90.93 |
| 8 | 89.48718 | 92.30769 | 89.97429 | 90.59 |
| 9 | 91.79487 | 91.53846 | 89.46015 | 90.93 |
| 10 | 90.76923 | 92.82051 | 90.48843 | 91.36 |
| 11 | 89.23077 | 91.53846 | 91.77378 | 90.85 |
| 12 | 90.51282 | 91.53846 | 87.14653 | 89.73 |
| 13 | 90 | 92.5641 | 89.20309 | 90.59 |
| 14 | 92.05128 | 93.58974 | 89.20309 | 91.61 |
| 15 | 90.51282 | 92.30769 | 89.97429 | 90.93 |
| 16 | 90.76923 | 92.5641 | 88.68895 | 90.67 |
| 17 | 90.51282 | 93.07692 | 88.43188 | 90.67 |
| 18 | 90.51282 | 92.30769 | 91.00257 | 91.27 |
| 19 | 91.79487 | 91.53846 | 88.94602 | 90.76 |
| 20 | 92.82051 | 90.76923 | 90.23136 | 91.27 |

**NN – learning rates**

params: learnrate 0.3, momentum 0.2, validation set 80, validation threshold 20, hidden units 10

| traintime | acc1 | acc2 | acc3 | accuracy | |
|---|---|---|---|---|---|
| 5 | 54.8718 | 58.46154 | 71.72237 | 61.68523 | |
| 10 | 63.33333 | 72.30769 | 75.83548 | 70.49217 | |
| 20 | 67.94872 | 72.05128 | 76.09255 | 72.03085 | |
| 50 | 70.25641 | 77.17949 | 75.57841 | 74.3381 | |
| 100 | 75.38462 | 81.28205 | 75.57841 | 77.41502 | |
| 200 | 82.30769 | 81.53846 | 75.57841 | 79.80819 | |
| 300 | 80.51282 | 81.53846 | 75.57841 | 79.2099 | - max reached here; probably due to use of validation set. |
| 400 | 80.51282 | 81.53846 | 75.57841 | 79.2099 | |
| 500 | 80.51282 | 81.53846 | 75.57841 | 79.2099 | |
| 700 | 80.51282 | 81.53846 | 75.57841 | 79.2099 | |
| 1000 | 80.51282 | 81.53846 | 75.57841 | 79.2099 | |

params: learnrate 0.1, momentum 0.2, validation set 80, validation threshold 20, hidden units 10

| traintime | acc1 | acc2 | acc3 | accuracy | |
|---|---|---|---|---|---|
| 5 | 50 | 50 | 50.12854 | 50.04285 | |
| 10 | 52.30769 | 54.8718 | 56.81234 | 54.66394 | |
| 20 | 70.25641 | 72.30769 | 75.57841 | 72.71417 | |
| 50 | 70 | 73.33333 | 75.32134 | 72.88489 | |
| 100 | 69.48718 | 74.61539 | 75.06427 | 73.05561 | |
| 200 | 69.48718 | 78.71795 | 75.06427 | 74.42313 | |
| 300 | 69.48718 | 82.5641 | 75.06427 | 75.70518 | - max reached here; probably due to use of validation set. |
| 400 | 69.48718 | 82.5641 | 75.06427 | 75.70518 | |
| 500 | 69.48718 | 82.5641 | 75.06427 | 75.70518 | |
| 700 | 69.48718 | 82.5641 | 75.06427 | 75.70518 | |
| 1000 | 69.48718 | 82.5641 | 75.06427 | 75.70518 | |

**Corpus sizes**

| Corpus | Examples | Naïve Bayes | C4.5 | K* | Neural net |
|---|---|---|---|---|---|
| time | 232 | 19.95% | 40.41% | 26.79% | 22.58% |
| adi | 248 | 63.94% | 62.35% | 41.18% | 65.20% |
| cacm | 1044 | 67.14% | 73.52% | 44.60% | 81.63% |
| cran | 1136 | 45.01% | 64.39% | 17.58% | 69.30% |
| med | 1146 | 51.25% | 62.01% | 37.90% | 87.51% |
| cisi | 5090 | 57.43% | 77.97% | 39.02% | 82.63% |

# Appendix B – Cranfield rankings

Taken verbatim from cranqrel.txt, available as part of the Cranfield collection via anonymous FTP at ftp.cs.cornell.edu/pub/smart/cran/

```
Here you will find two files containing relevance judgements.
CRAN.REL was taken from Ed Fox's Virginia Disc 1 CD-ROM
The other came courtesy of Donna Harman (who is a star).

Ed's file contains query-doc id pairs

Donna's file contains the same query doc-id pairs AND includes
degrees of relevance which are discussed below.

For some strange reason the files are identical for all but
three lines.  I leave it to you to figure out the difference.



I am attaching my copy of the qrels for cranfield 1400
(cranqrel.txt), including the codes for relevancy scale,
which were added here.  The qrels are in three columns:
the first is the query number, the second is the relevant
document number, and the third is the relevancy code.
The codes are defined by Cleverdon as follows:

     "1.  References which are a complete answer to the question.

      2.  References of a high degree of relevance, the lack of which
          either would have made the research impracticable or would
          have resulted in a considerable amount of extra work.

      3.  References which were useful, either as general background
          to the work or as suggesting methods of tackling certain
aspects
          of the work.

      4.  References of minimum interest, for example, those that
have been
          included from an historical viewpoint.

      5.  References of no interest."

Obviously no 5's are included in the qrels.
```

# Appendix C – ARFF format

This entire appendix taken verbatim from [14].

Attribute-Relation File Format (ARFF)

April 4th, 2006

This documentation is superceded by the WekaDoc Wiki. Version specific documentation is available there:

* 3.4.x
* 3.5.x

April 1st, 2002

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software. This document descibes the version of ARFF used with Weka versions 3.2 to 3.3; this is an extension of the ARFF format as described in the data mining book written by Ian H. Witten and Eibe Frank (the new additions are string attributes, date attributes, and sparse instances).

This explanation was cobbled together by Gordon Paynter (gordon.paynter at ucr.edu) from the Weka 2.1 ARFF description, email from Len Trigg (lenbok at myrealbox.com) and Eibe Frank (eibe at cs.waikato.ac.nz), and some datasets. It has been edited by Richard Kirkby (rkirkby at cs.waikato.ac.nz). Contact Len if you're interested in seeing the ARFF 3 proposal.

Overview

ARFF files have two distinct sections. The first section is the Header information, which is followed the Data information.

The Header of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. An example header on the standard IRIS dataset looks like this:

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
%
@RELATION iris
```

```
@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}
```

The Data of the ARFF file looks like the following:

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

Lines that begin with a % are comments. The @RELATION, @ATTRIBUTE and @DATA declarations are case insensitive.

Examples

Several well-known machine learning datasets are distributed with Weka in the $WEKAHOME/data directory as ARFF files.

The ARFF Header Section

The ARFF Header section of the file contains the relation declaration and attribute declarations.

The @relation Declaration

The relation name is defined as the first line in the ARFF file. The format is:

```
@relation <relation-name>
```

where <relation-name> is a string. The string must be quoted if the name includes spaces.

The @attribute Declarations

Attribute declarations take the form of an orderd sequence of @attribute statements. Each attribute in the data set has its own @attribute statement which uniquely defines the name of that attribute and it's data type. The order the attributes are declared indicates the column position in the data section of the file. For example, if an attribute is the third one declared then Weka expects that all that attributes values will be found in the third comma delimited column.

The format for the @attribute statement is:

@attribute <attribute-name> <datatype>

where the <attribute-name> must start with an alphabetic character. If spaces are to be included in the name then the entire name must be quoted.

The <datatype> can be any of the four types currently (version 3.2.1) supported by Weka:

* numeric
* <nominal-specification>
* string
* date [<date-format>]

where <nominal-specification> and <date-format> are defined below. The keywords numeric, string and date are case insensitive.

Numeric attributes

Numeric attributes can be real or integer numbers.

Nominal attributes

Nominal values are defined by providing an <nominal-specification> listing the possible values: {<nominal-name1>, <nominal-name2>, <nominal-name3>, ...}

For example, the class value of the Iris dataset can be defined as follows:

@ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}

Values that contain spaces must be quoted.

String attributes

String attributes allow us to create attributes containing arbitrary textual values. This is very useful in text-mining applications, as we can create datasets with string attributes, then write Weka Filters to manipulate strings (like StringToWordVectorFilter). String attributes are declared as follows:

```
@ATTRIBUTE LCC    string
```

Date attributes

Date attribute declarations take the form:

```
@attribute <name> date [<date-format>]
```

where <name> is the name for the attribute and <date-format> is an optional string specifying how date values should be parsed and printed (this is the same format used by SimpleDateFormat). The default format string accepts the ISO-8601 combined date and time format: "yyyy-MM-dd'T'HH:mm:ss".

Dates must be specified in the data section as the corresponding string representations of the date/time (see example below).

ARFF Data Section

The ARFF Data section of the file contains the data declaration line and the actual instance lines.

The @data Declaration

The @data declaration is a single line denoting the start of the data segment in the file. The format is:

```
@data
```

The instance data

Each instance is represented on a single line, with carriage returns denoting the end of the instance.

Attribute values for each instance are delimited by commas. They must appear in the order that they were declared in the header section (i.e. the data corresponding to the nth @attribute declaration is always the nth field of the attribute).

Missing values are represented by a single question mark, as in:

```
@data
4.4,?,1.5,?,Iris-setosa
```

Values of string and nominal attributes are case sensitive, and any that contain space must be quoted, as follows:

```
@relation LCCvsLCSH

@attribute LCC string
@attribute LCSH string

@data
AG5,   'Encyclopedias and dictionaries.;Twentieth century.'
AS262, 'Science -- Soviet Union -- History.'
AE5,   'Encyclopedias and dictionaries.'
AS281, 'Astronomy, Assyro-Babylonian.;Moon -- Phases.'
AS281, 'Astronomy, Assyro-Babylonian.;Moon -- Tables.'
```

Dates must be specified in the data section using the string representation specified in the attribute declaration. For example:

```
@RELATION Timestamps

@ATTRIBUTE timestamp DATE "yyyy-MM-dd HH:mm:ss"

@DATA
"2001-04-03 12:12:12"
"2001-05-03 12:59:55"
```

Sparse ARFF files

Sparse ARFF files are very similar to ARFF files, but data with value 0 are not be explicitly represented.

Sparse ARFF files have the same header (i.e @relation and @attribute tags) but the data section is different. Instead of representing each value in order, like this:

```
@data
0, X, 0, Y, "class A"
0, 0, W, 0, "class B"
```

the non-zero attributes are explicitly identified by attribute number and their value stated, like this:

```
@data
{1 X, 3 Y, 4 "class A"}
{2 W, 4 "class B"}
```

Each instance is surrounded by curly braces, and the format for each entry is: <index> <space> <value> where index is the attribute index (starting from 0).

Note that the omitted values in a sparse instance are 0, they are not "missing" values! If a value is unknown, you must explicitly represent it with a question mark (?).

Warning: There is a known problem saving SparseInstance objects from datasets that have string attributes. In Weka, string and nominal data values are stored as numbers; these numbers act as indexes into an array of possible attribute values (this is very efficient). However, the first string value is assigned index 0: this means that, internally, this value is stored as a 0. When a SparseInstance is written, string instances with internal value 0 are not output, so their string value is lost (and when the arff file is read again, the default value 0 is the index of a different string value, so the attribute value appears to change). To get around this problem, add a dummy string value at index 0 that is never used whenever you declare string attributes that are likely to be used in SparseInstance objects and saved as Sparse ARFF files.

# Appendix D – Examples entries from collections

**ADI**

```
.I 1
.T
the ibm dsd technical information center - a total systems approach
combining traditional library features
and mechanized computer processing
.A
H. S. WHITE
.W
the ibm data systems division technical
 information center (tic) provides an operating developmental
system for integrated and compatible mechanized
 processing of technical information received within the
organization.
  the system offers several advantages :
     1 . it is a sophisticated mechanized system for dissemination
and retrieval;
     2 . it is compatible with all library mechanized
  records produced under a standard processing format
  within ibm libraries, providing such traditional tools
  as 3 x 5 catalog cards, circulation records and overdue
notices;
     3 . it is reversible, so that discontinuation of machine
processing would not cause gaps in the library's
  manual records;
     4 . it is controlled, producing statistical evaluations
of its own program efficiency;
     5 . it is user-oriented, providing 24-hour copy access
and immediate microfilm access to its documents;
     6 . it is relatively simple, relying on the ibm 1401
  data processing system for all its processing and output.

  since the system has been operating for over a year, the
conclusions drawn are based on actual experience .
```

**CACM**

```
.I 74
.T
A High-Speed Sorting Procedure
.B
CACM July, 1959
.A
Shell, D. L.
.N
CA590704 JB March 22, 1978  6:20 PM
.X
1919  5     74
74    5     74
74    5     74
74    5     74
852   5     74
864   5     74
865   5     74
864   6     74
1175  6     74
232   6     74
232   6     74
308   6     74
309   6     74
309   6     74
74    6     74
```

```
74    6     74
74    6     74
74    6     74
3187  6     74
```

**CISI**

```
.I 2
.T
Use Made of Technical Libraries
.A
Slater, M.
.W
This report is an analysis of 6300 acts of use
in 104 technical libraries in the United Kingdom.
Library use is only one aspect of the wider pattern of
information use.  Information transfer in libraries is
restricted to the use of documents.  It takes no
account of documents used outside the library, still
less of information transferred orally from person
to person.  The library acts as a channel in only a
proportion of the situations in which information is
transferred.
Taking technical information transfer as a whole,
there is no doubt that this proportion is not the
major one.  There are users of technical information -
particularly in technology rather than science -
who visit libraries rarely if at all, relying on desk
collections of handbooks, current periodicals and personal
contact with their colleagues and with people in other
organizations.  Even regular library users also receive
information in other ways.
.X
2     5     2
32    1     2
76    1     2
132   1     2
137   1     2
139   1     2
152   2     2
155   1     2
158   1     2
183   1     2
195   1     2
203   1     2
204   1     2
210   1     2
243   1     2
371   1     2
475   1     2
552   1     2
760   1     2
770   1     2
771   1     2
774   1     2
775   1     2
776   1     2
788   1     2
789   1     2
801   1     2
815   1     2
839   1     2
977   1     2
1055  1     2
1056  1     2
1151  1     2
1361  1     2
1414  1     2
```

```
1451  1     2
1451  1     2
```

**Cranfield**

```
.I 12
.T
some structural and aerelastic considerations of high
speed flight .
.A
bisplinghoff,r.l.
.B
j. ae. scs. 23, 1956, 289.
.W
some structural and aerelastic considerations of high
speed flight .
  the dominating factors in structural design of high-speed
aircraft are thermal and aeroelastic in origin .  the subject
matter is concerned largely with a discussion of these factors and
their interrelation with one another .  a summary is presented
of some of the analytical and experimental tools available to
aeronautical engineers to meet the demands of high-speed flight
upon aircraft structures .  the state of the art with respect to
heat transfer from the boundary layer into the structure, modes
of failure under combined load as well as thermal inputs and
acrothermoelasticity is discussed .  methods of attacking and
alleviating structural and aeroelastic problems of high-speed
flight are summarized .  finally, some avenues of fundamental
research are suggested .
```

**MED**

```
.I 38
.W
studies of nickel carcinogenesis fractionations of nickel in
ultracentrifugal supernatants of lung and liver by means of dextran
gel
chromatography .
  chromatographic fractionations have been performed on the
ultracentrifugal supernatants of homogenates of rat lung and liver by
the use of columns of dextran gel (sephadex g-100) . a major
proportion
of nickel in these tissue supernatants has been demonstrated to be
firmly bound to macromolecular constituents . following acute and
chronic inhalation of carcinogenic levels of nickel carbonyl, the
predominant increases in the concentrations of nickel have been
observed
in the macromolecular fractions . these findings are consistent with
the
previous demonstration of nickel in purified preparations of
ribonucleic
acids (rna) from several rat tissues, and with the observation of
increased concentrations of nickel in high-molecular weight rna from
lung and liver following the inhalation of nickel carbonyl .
```

**TIME**

```
*TEXT 099 02/15/63 PAGE 038

CENTRAL AFRICA TROUBLE BREWING CUSTOMS AGENTS AND SPECIAL BRANCH

DETECTIVES CHARGED WITH SAFEGUARDING THE BORDERS OF THE SPRAWLING

RHODESIAN FEDERATION HAVE BEEN RUN RAGGED LATELY . IN THE NORTH,
THERE
```

IS A STEADY TRAFFIC OF WHITE MERCENARIES AND AFRICAN SOLDIERS FROM THE

ROUTED KATANGESE ARMY, WHO SLIP ACROSS THE CONGO LINE TO PEDDLE THEIR

WEAPONS TO EAGER WHITE AND BLACK RHODESIANS WHO MAY ONE DAY USE THEM ON

EACH OTHER . IN THE EAST, SMUGGLERS FROM THE PORTUGUESE COLONY OF

MOZAMBIQUE MAKE THEIR WAY THROUGH THE WILD, MOUNTAINOUS BUSH TO BRING

IN DAGGA WEED (MARIJUANA) AND TAKE OUT GOLD STOLEN BY WORKMEN IN

RHODESIAN MINES . LAST WEEK THE HARRIED BORDER GUARDS HAD A NEW CHORE

: TO PREVENT THE SMUGGLING OF HOPS INTO SOUTHERN RHODESIA . AT

BEITBRIDGE, ON THE LIMPOPO RIVER, A CUSTOMS OFFICER DUTIFULLY SEARCHED

THE LUGGAGE OF A VACATIONER RETURNING FROM SOUTH AFRICA, THEN

WHISPERED, " MAN, WHAT DOES A HOP LOOK LIKE ? NO ONE HERE HAS EVER

SEEN ONE ! " THE HOP CRISIS RESULTS FROM A $28 DUTY ON EVERY POUND OF

IMPORTED HOPS IMPOSED BY THE GOVERNMENT OF SIR ROY WELENSKY BECAUSE TAX

REVENUE FROM COMMERCIAL BEER HAS NOT BEEN UP TO EXPECTATIONS . " THIS

IS DUE TO THE SPREAD OF HOME BREWING, " COMPLAINED THE GOVERNMENT .

HOME BREWERS ARE GENERALLY RESPECTABLE CITIZENS, RANGING FROM RAILROAD

ENGINEERS AND CIVIL SERVANTS TO BANK CLERKS AND GARAGE MECHANICS MEN

WHO FIND COMMERCIAL BEER TOO EXTRAVAGANT FOR THEIR BUDGETS . THE NEW

DUTY WOULD MAKE HOME BREW TWICE AS COSTLY AS THE COMMERCIAL STUFF .

QUICKLY FORMING A PRESSURE GROUP GRANDLY NAMED THE AMATEUR BREWERS &

VINTNERS ASSOCIATION, SOME 300 DO-IT-YOURSELF BRAUMEISTERS FIRED OFF A

PROTEST TO WELENSKY, POINTING OUT THAT HOME BREWING " HAS TAKEN PLACE

IN THE UNITED KINGDOM FOR CENTURIES, AND AS THE BRITISH EMIGRATED TO

THE COLONIES, THIS TRADITION HAS BEEN ACCEPTED AS THE BIRTHRIGHT OF THE

ORDINARY MAN BY EVERY GOVERNMENT OF THE COMMONWEALTH .