# 20
# Text Mining

In this chapter we look at a particular type of classification task, where the objects are text documents such as articles in newspapers, scientific papers in journals or perhaps abstracts of papers, or even just their titles. The aim is to use a set of pre-classified documents to classify those that have not yet been seen. This is becoming an increasingly important practical problem as the volume of printed material in many fields keeps increasing and even in specialist fields it can be very difficult to locate relevant documents. Much of the terminology used reflects the origins of this work in librarianship and information science, long before data mining techniques became available.

In principle we can use any of the standard methods of classification (Naïve Bayes, Nearest Neighbour, decision trees etc.) for this task, but datasets of text documents have a number of specific features compared with the datasets we have seen so far, which require separate explanation. The special case where the documents are web pages will be covered in Section 20.9.

## 20.1 Multiple Classifications

An important issue that distinguishes text classification from the other classification tasks discussed in this book is the possibility of multiple classifications. Up to now we have assumed that there is a set of mutually exclusive categories and that each object must inevitably fit into one and only one of these.

Text classification is rather different. In general we may have $N$ categories such as Medicine, Business, Finance, Historical, Biographical, Management and Education and it is perfectly possible for a document to fit into several of these categories, possibly even all of them or possibly none.

Rather than broaden the definition of classification used up to now we prefer to think of the text classification task as $N$ separate binary classification tasks, e.g.

– Is the document about medicine? Yes/No

– Is the document about business? Yes/No

– Is the document about finance? Yes/No

and so on. The need to perform $N$ separate classification tasks adds considerably to the time involved for this form of classification, which even for a single classification is usually computationally expensive.

## 20.2 Representing Text Documents for Data Mining

For 'standard' data mining tasks the data is presented to the data mining system in the standard form described in Chapter 2, or something similar. There are a fixed number of attributes (or features) which were chosen before the data was collected. For text mining the dataset usually comprises the documents themselves and the features are extracted from the documents automatically based on their content before the classification algorithm is applied. There are generally a very large number of features, most of them only occurring rarely, with a high proportion of noisy and irrelevant features.

There are several ways in which the conversion of documents from plain text to instances with a fixed number of attributes in a training set can be carried out. For example we might count the number of times specified phrases occur, or perhaps any combination of two consecutive words, or we might count the occurrence of two or three character combinations (known as *bigrams* and *trigrams* respectively). For the purposes of this chapter we will assume that a simple word-based representation is used, known as a *bag-of-words representation.* With this representation a document is considered to be simply a collection of the words which occur in it at least once. The order of the words, the combinations in which they occur, paragraph structuring, punctuation and of course the meanings of the words are all ignored. A document is just a collection of words placed in some arbitrary order, say alphabetical, together with a count of how many times each one occurs, or some other measure of the importance of each word.

Assuming that we wish to store an ‘importance value’ for each word in a document as one instance in a training set, how should we do it? If a given document has say 106 different words, we cannot just use a representation with 106 attributes (ignoring classifications). Other documents in the dataset may use other words, probably overlapping with the 106 in the current instance, but not necessarily so. The unseen documents that we wish to classify may have words that are not used in any of the training documents. An obvious — but extremely bad — approach would be to allocate as many attributes as are needed to allow for all possible words that might be used in any possible unseen document. Unfortunately if the language of the documents is English, the number of possible words is approximately one million, which is a hopelessly impractical number of attributes to use.

A much better approach is to restrict the representation to the words that actually occur in the training documents. This can still be many thousands (or more) and we will look at ways of reducing this number in Sections 20.3 and 20.4 below. We place all the words used at least once in a ‘dictionary’ and allocate one attribute position in each row of our training set for each one. The order in which we do this is arbitrary, so we can think of it as alphabetical.

The bag-of-words representation is inherently a highly redundant one. It is likely that for any particular document most of the attributes/features (i.e. words) will not appear. For example the dictionary used may have 10,000 words, but a specific document may have just 200 different words. If so, its representation as an instance in the training set will have 9,800 out of 10,000 attributes with value zero, indicating no occurrences, i.e. unused.

If there are multiple classifications there are two possibilities for constructing the dictionary of words for a collection of training documents. Whichever one is used the dictionary is likely to be large.

The first is the *local dictionary* approach. We form a different dictionary for each category, using only those words that appear in documents classified as being in that category. This enables each dictionary to be relatively small at the cost of needing to construct $N$ of them, where there are $N$ categories.

The second approach is to construct a *global dictionary*, which includes all the words that occur at least once in any of the documents. This is then used for classification into each of the $N$ categories. Constructing a global dictionary will clearly be a lot faster than constructing $N$ local dictionaries, but at the cost of making an even more redundant representation to use for classifying into each of the categories. There is some evidence to suggest that using a local dictionary approach tends to give better performance than using a global dictionary.

## 20.3 Stop Words and Stemming

With the bag-of-words approach, it is possible to have tens of thousands of different words occurring in a fairly small set of documents. Many of them are not important for the learning task and their usage can substantially degrade performance. It is imperative to reduce the size of the *feature space* (i.e. the set of words included in the dictionary) as far as possible. This can be looked at as a variant of the methods of data preparation and data cleaning described in Chapter 2.

One widely used approach is to use a list of common words that are likely to be useless for classification, known as *stop words*, and remove all occurrences of these words before creating the bag-of-words representation. There is no definitive list of stop words that is universally used. The list would obviously vary from language to language, but in English some obvious choices would be ‘a’, ‘an’, ‘the’, ‘is’, ‘I’, ‘you’ and ‘of’. Studying the frequency and distribution of such words might be very useful for stylistic analysis, i.e. deciding which of a number of possible authors wrote a novel or a play etc., but for classifying a document into categories such as Medicine, Finance etc. they are clearly useless. The University of Glasgow has a list of 319 English stop words beginning with a, about, above, across, after, afterwards and ending with yet, you, your, yours, yourself, yourselves. Up to a point the longer the list of stop words the better, the only risk being the possible loss of useful classifying information if the list becomes excessive.

Another very important way to reduce the number of words in the representation is to use *stemming.*

This is based on the observation that words in documents often have many *morphological variants.* For example we may use the words computing, computer, computation, computes, computational, computable and computability all in the same document. These words clearly have the same linguistic root. Putting them together as if they were occurrences of a single word would probably give a strong indication of the content of the document whereas each word individually might not.

The aim of stemming is to recognise sets of words such as ‘computing’ and ‘computation’ or ‘applied’, ‘applying’, ‘applies’ and ‘apply’ that can be treated as equivalent. There are many *stemming algorithms* that have been developed to reduce a word to its *stem* or root form, by which it is then replaced. For example, ‘computing’ and ‘computation’ might both be stemmed to ‘comput’, and ‘applies’ etc. to ‘appli’.

The use of stemming can be a very effective way of reducing the number of words in a bag-of-words representation to a relatively manageable number. However, as for stop words, there is no standard stemming algorithm that is universally used and an over-zealous stemming algorithm can remove valuable words from consideration. For example the word 'appliqué' in a document may be an important guide to its classification, but might be reduced by stemming to 'appli', the same stem as if it were a much less significant word such as 'applies' (with which it is very unlikely to have any genuine linguistic connection).

## 20.4 Using Information Gain for Feature Reduction

Even after removing stop words from a document and replacing each remaining word by its stem, the number of words in a bag-of-words representation of a set of documents can be very large.

One way to reduce the number of words for a given category of documents $C_k$ is to construct a training set where each instance comprises the frequency of each word (or some similar measure) together with the value of the classification $C_k$ which must be a binary yes/no value.

The entropy of this training set can be calculated in the same way as in previous chapters. For example, if 10% of the training documents are in category $C_k$, the entropy is $-0.1 \times \log_2 0.1 - 0.9 \times \log_2 0.9 = 0.47$.

Using a method such as the frequency table technique described in Chapter 6, we can now calculate the information gain as far as classifying a document as belonging to category $C_k$ or otherwise is concerned that would result from knowing the value of each of the attributes in turn. Having done this we might choose to use only the features with the highest (say) 20, 50 or 100 values of information gain when classifying documents by whether or not they belong to category $C_k$.

## 20.5 Representing Text Documents: Constructing a Vector Space Model

We shall now assume that we have decided whether to use a local or a global dictionary and have chosen a representation which replaces each document by a number of features. For a bag-of-words representation each feature is a single word, but for a different representation it may be something else, e.g. a phrase. In the following we will assume that each feature is a *term* of some kind.

Once we have determined that the total number of features is $N$, we can represent the terms in the dictionary in some arbitrary order as $t_1, t_2, \ldots, t_N$.

We can then represent the $i$th document as an ordered set of $N$ values, which we will call *an N-dimensional vector* and write as $(X_{i1}, X_{i2}, \ldots, X_{iN})$. These values are just the attribute values in the standard training set format used elsewhere in this book, but with the classification(s) omitted. Writing the values as $N$-dimensional vectors (i.e. as $N$ values separated by commas and enclosed in parentheses) is simply a more conventional way of looking at the data in this branch of data mining. The complete set of vectors for all documents under consideration is called a *vector space model* or *VSM.*

Up to now we have assumed that the values stored for each feature (attribute) are the number of times each term occurs in the corresponding document. However that does not have to be the case. In general we can say that value $X_{ij}$ is a *weight* measuring the importance of the $j$th term $t_j$ in the $i$th document.

One common way of calculating the weights is to count the number of occurrences of each term in the given document (known as *term frequency*). Another possibility is to use a binary representation, where 1 indicates the presence and 0 indicates the absence of the term in the document.

A more complex way of calculating the weights is called TFIDF, which stands for Term Frequency Inverse Document Frequency. This combines term frequency with a measure of the rarity of a term in the complete set of documents. It has been reported as leading to improved performance over the other methods.

The TFIDF value of a weight $X_{ij}$ is calculated as the product of two values, which correspond to the term frequency and the inverse document frequency, respectively.

The first value is simply the frequency of the $j$th term, i.e. $t_j$, in document $i$. Using this value tends to make terms that are frequent in the given (single) document more important than others.

We measure the value of inverse document frequency by $\log_2(n/n_j)$ where $n_j$ is the number of documents containing term $t_j$ and $n$ is the total number of documents. Using this value tends to make terms that are rare across the collection of documents more important than others. If a term occurs in every document its inverse document frequency value is 1. If it occurs in only one document out of every 16, its inverse document frequency value is $\log_2 16 = 4$.

## 20.6 Normalising the Weights

Before using the set of $N$-dimensional vectors we first need to normalise the values of the weights, for reasons similar to the need to normalise the value of continuous attributes in Chapter 3.

We would like each value to be between 0 and 1 inclusive and for the values used not to be excessively influenced by the overall number of words in the original document.

We will take a much simplified example to illustrate the point. Suppose we have a dictionary with just 6 members and let us assume that the weights used are just the term frequency values. Then a typical vector would be $(0, 3, 0, 4, 0, 0)$. In the corresponding document the second term appeared 3 times, the fourth term occurred 4 times and the other four terms did not occur at all. Overall only 7 terms occurred in the document, after removal of stop words, stemming etc.

Suppose we now create another document by placing an exact duplicate of its content at the end of the first one. What if by some printing aberration there were other documents where the content of the original one was printed 10 times, or even a hundred times?

In these three cases the vectors would be $(0, 6, 0, 8, 0, 0)$, $(0, 30, 0, 40, 0, 0)$ and $(0, 300, 0, 400, 0, 0)$. These seem to have nothing in common with the original vector, which was $(0, 3, 0, 4, 0, 0)$. This is unsatisfactory. The four documents should obviously be classified in exactly the same way and the vector space representation should reflect this.

The method that is generally used to normalise vectors neatly solves this problem. We calculate the *length* of each vector, defined as the square root of the sum of the squares of its component values. To normalise the values of the weights we divide each value by the length. The resulting vector has the property that its length is always 1.

For the above example the length of $(0, 3, 0, 4, 0, 0)$ is $\sqrt{(3^2 + 4^2)} = 5$, so the normalised vector is $(0, 3/5, 0, 4/5, 0, 0)$, which has length 1. Note that the zero values play no part in the calculations.

The calculations for the other three vectors given are as follows.

$\underline{(0, 6, 0, 8, 0, 0)}$
The length is $\sqrt{(6^2 + 8^2)} = 10$, so the normalised vector is
$(0, 6/10, 0, 8/10, 0, 0) = (0, 3/5, 0, 4/5, 0, 0)$.

$\underline{(0, 30, 0, 40, 0, 0)}$
The length is $\sqrt{(30^2 + 40^2)} = 50$, so the normalised vector is
$(0, 30/50, 0, 40/50, 0, 0) = (0, 3/5, 0, 4/5, 0, 0)$.

$\underline{(0, 300, 0, 400, 0, 0)}$
The length is $\sqrt{(300^2 + 400^2)} = 500$, so the normalised vector is
$(0, 300/500, 0, 400/500, 0, 0) = (0, 3/5, 0, 4/5, 0, 0)$.

In normalised form all four vectors are the same, as they should be.

## 20.7 Measuring the Distance Between Two Vectors

One important check on the appropriateness of the normalised vector space model representation of documents described in the last two sections is whether we can make a sensible definition of the distance between two vectors. We would like the distance between two identical vectors to be zero, the distance between two vectors that are as dissimilar as possible to be 1 and the distance between any other two vectors to be somewhere in between.

The standard definition of the distance between two vectors of length one, known as *unit vectors*, meets these criteria.

We define the *dot product* of two unit vectors of the same dimension to be the sum of the products of the corresponding pairs of values.

For example, if we take the two unnormalised vectors $(6, 4, 0, 2, 1)$ and $(5, 7, 6, 0, 2)$, normalising them to unit length converts the values to
$(0.79, 0.53, 0, 0.26, 0.13)$ and $(0.47, 0.66, 0.56, 0, 0.19)$.

The dot product is now $0.79 \times 0.47 + 0.53 \times 0.66 + 0 \times 0.56 + 0.26 \times 0 + 0.13 \times 0.19 = 0.74$ approximately.

If we subtract this value from 1 we obtain a measure of the distance between the two values, which is $1 - 0.74 = 0.26$.

What happens if we calculate the distance between two identical unit vectors? The dot product gives the sum of the squares of the values, which must be 1 as the length of a unit vector is 1 by definition. Subtracting this value from 1 gives a distance of zero.

If we take two unit vectors with no values in common (corresponding to no terms in common in the original documents), say $(0.94, 0, 0, 0.31, 0.16)$ and $(0, 0.6, 0.8, 0, 0)$ the dot product is $0.94 \times 0 + 0 \times 0.6 + 0 \times 0.8 + 0.31 \times 0 + 0.16 \times 0 = 0$. Subtracting this value from 1 gives a distance measure of 1, which is the largest distance value achievable.

## 20.8 Measuring the Performance of a Text Classifier

Once we have converted the training documents into normalised vector form, we can construct a training set of the kind used in previous chapters for each category $C_k$ in turn. We can convert a set of test documents to a test set of instances for each category in the same way as the training documents and apply whatever classification algorithm we choose to the training data to classify the instances in the test set.

For each category $C_k$ we can construct a confusion matrix of the kind discussed in Chapter 7.

| | | Predicted class | |
|---|---|---|---|
| | | $C_k$ | not $C_k$ |
| Actual class | $C_k$ | $a$ | $c$ |
| | not $C_k$ | $b$ | $d$ |

**Figure 20.1** Confusion Matrix for Category $C_k$

In Figure 20.1 the values $a$, $b$, $c$ and $d$ are the number of true positive, false positive, false negative and true negative classifications, respectively. For a perfect classifier $b$ and $c$ would both be zero.

The value $(a+d)/(a+b+c+d)$ gives the predictive accuracy. However, as mentioned in Chapter 12, for information retrieval applications, which include text classification, it is more usual to use some other measures of classifier performance.

*Recall* is defined as $a/(a+c)$, i.e. the proportion of documents in category $C_k$ that are correctly predicted.

*Precision* is defined as $a/(a+b)$, i.e. the proportion of documents that are predicted as being in category $C_k$ that are actually in that category.

It is common practice to combine Recall and Precision into a single measure of performance called the *F1 Score*, which is defined by the formula $F1 = 2 \times \text{Precision} \times \text{Recall}/(\text{Precision} + \text{Recall})$. This is just the product of Precision and Recall divided by their average.

Having generated confusion matrices for each of the $N$ binary classification tasks we can combine them in several ways. One method is called *micro-averaging*. The $N$ confusion matrices are added together element by element to form a single matrix from which Recall, Precision, F1 and any other preferred measures can be computed.

## 20.9 Hypertext Categorisation

An important special case of text classification arises when the documents are web pages, i.e. HTML files. The automatic classification of web pages is usually known as *Hypertext Categorisation* (or *Hypertext Classification*).

Hypertext Categorisation is similar to classifying 'ordinary' text, e.g. articles in newspapers or journals, on the basis of their content, but as we shall see the former can often be considerably harder.

### 20.9.1 Classifying Web Pages

The most obvious question to ask is why should we bother to do hypertext categorisation, when there are powerful search engines such as Google available for locating web pages of interest.

It has been estimated that the World Wide Web comprises over 13 billion pages and is growing at a rate of several million pages a day. The size of the web will eventually overwhelm the conventional web search engine approach.

The present author lives in a small village in England. When he entered the village name (a unique one for England) into Google a year ago he was astonished to find it returned 87,200 entries — more than 50 times as many as the number of people who live there. This seemed a little excessive. Making the same query today we find that the number of entries has grown to 642,000. We can only speculate on what events have occurred in the village in the intervening year to warrant this much greater attention. For comparison the number of Google entries for Science a few years ago was 459,000,000. A year later it had reached 4,570,000,000.

In practice it is clear that many (probably most) Google users only ever look at the first screenful or two of the entries returned or try a more elaborate search. What else can they do? No one can possibly examine 4,570 million entries on anything. Unfortunately even highly specific queries can easily return many thousands of entries and this number can only grow as time goes by. Looking at only the first screenful or two of entries is placing a huge amount of reliance on the algorithm used by Google to rank the relevance of its entries — far more than can realistically be justified. This is in no way to criticise or denigrate a very successful company — just to point out that the standard approach used by web search engines will not keep working successfully for ever. We can be sure that the search engine companies are well aware of this. It is perhaps not surprising that there are studies that suggest that many users prefer to navigate through directories of pre-classified content and that this frequently enables them to find more relevant information in a shorter time.

When attempting to classify web pages we immediately run into the problem of finding any classified pages to use as training data. Web pages are uploaded by a very large number of individuals, operating in an environment where no widely agreed standard classification scheme exists. Fortunately there are ways of overcoming this problem, at least partially.

The search engine company, Yahoo, uses hundreds of professional classifiers to categorise new web pages into a (nearly) hierarchical structure, comprising 14 main categories, each with many sub-categories, sub-sub-categories etc. The complete structure can be found on the web at `http://dir.yahoo.com`. Users can search through the documents in the directory structure either using a search engine approach or by following links through the structure. For example we might follow the path from 'Science' to 'Computer Science' to 'Artificial Intelligence' to 'Machine Learning' to find a set of links to documents that human classifiers have placed in that category. The first of these (at the time of writing) is to the UCI Machine Learning Repository, which was discussed in Chapter 2.

The Yahoo system demonstrates the potential value of classifying web pages. However, only a very small proportion of the entire web could possibly be classified this way 'manually'. With 1.5 million new pages being added each day the volume of new material will defeat any conceivable team of human classifiers. An interesting area of investigation (which the present author and his research group are currently pursuing) is whether web pages can be classified automatically using the Yahoo classification scheme (or some other similar scheme) by supervised learning methods of the kind described in this book.

Unlike many other task areas for data mining there are few 'standard' datasets available on which experimenters can compare their results. One exception is the *BankSearch* dataset created by the University of Reading, which includes 11,000 web pages pre-classified (by people) into four main categories (Banking and Finance, Programming, Science, Sport) and 11 sub-categories, some quite distinct and some quite similar.

## 20.9.2 Hypertext Classification versus Text Classification

Classifying hypertext has some important differences from classifying 'standard' text. Only a small number of web pages (manually classified) are available for supervised learning and it is often the case that much of the content of each web page is irrelevant to the topic of the page (links to photographs of the creator's family, train timetables, advertisements etc.).

However one difference is fundamental and unavoidable. In text classification the words that the human reader sees are very similar to the data provided to the classification program. Figure 20.2 is a typical example.

Marley was dead: to begin with. There is no doubt whatever about that. The register of his burial was signed by the clergyman, the clerk, the undertaker, and the chief mourner. Scrooge signed it: and Scrooge's name was good upon 'Change, for anything he chose to put his hand to. Old Marley was as dead as a door-nail.

Mind! I don't mean to say that I know, of my own knowledge, what there is particularly dead about a door-nail. I might have been inclined, myself, to regard a coffin-nail as the deadest piece of ironmongery in the trade. But the wisdom of our ancestors is in the simile; and my unhallowed hands shall not disturb it, or the Country's done for. You will therefore permit me to repeat, emphatically, that Marley was as dead as a door-nail.

*Source: Charles Dickens. A Christmas Carol.*

**Figure 20.2** Text Classification: An Example

Automating the classification of a document based on its content is a hard task (for the example above we might perhaps decide on the categories 'death' and 'ironmongery'). However the problems pale into insignificance compared with classifying even a fairly short piece of hypertext.

Figure 20.3 shows the first few lines of the text form of a well-known web page. It is a small extract from the text that an automatic hypertext categorisation program would need to process. It contains precisely one word of useful information, which occurs twice. The rest is HTML markup and JavaScript that gives no clue to the correct classification of the page.

It is usually considerably easier (for humans) to classify web pages from the 'pictorial' form of the pages displayed by a web browser. In this case, the equivalent web page is a very familiar one (see Figure 20.4).

It is worth noting that most of the words on this page are of little or no use to human classifiers, for example 'images', 'groups', 'news', 'preferences' and 'We're Hiring'. There are only two clues to the correct classification of this page: the phrase 'Searching 8,058,044,651 web pages' and the name of the company. From these we can correctly deduce that it is the home page of a widely used search engine.

A program that attempts to classify this page automatically has to contend with not only the scarcity of useful information in the page, even for human classifiers, but the abundance of irrelevant information in the textual form that it is given.

```
<html><head><meta http-equiv="content-type"
content="text/html; charset=UTF-8">
<title>Google</title><style>
<!--
body,td,a,p,.h{font-family:arial,sans-serif;}
.h{font-size: 20px;}
.q{color:#0000cc;}
//-->
</style>
<script>
<!--
function sf(){document.f.q.focus();}
function clk(el,ct,cd) {if(document.images){(new Image()).src=
"/url?sa=T&ct="+es
cape(ct)+"&cd="+escape(cd)+"&url="
+escape(el.href)+"&ei=gpZNQpzEHaSgQYCUwKoM";}return true;}
// -->
</script>
</head><body bgcolor=#ffffff text=#000000 link=#0000cc vlink=
#551a8b alink=#ff00
00 onLoad=sf()><center><img src="/intl/en_uk/images/logo.gif"
width=276 height=1
10 alt="Google"><br><br>
```

**Figure 20.3** Hypertext Classification: An Example

We can deal with the second problem to some extent by removing HTML markup and JavaScript when we create a representation of a document such as a 'bag-of-words', but the scarcity of relevant information on most web pages remains a problem. We must be careful not to assume that HTML markup is always irrelevant noise — the only two useful words in Figure 20.3 (both 'Google') appear in the HTML markup.

Even compared with articles in newspapers, papers in scientific journals etc. web pages suffer from an extremely diverse authorship, with little consistency in style or vocabulary, and extremely diverse content. Ignoring HTML markup, JavaScript, irrelevant advertisements and the like, the content of a web page is often quite small. It is not surprising that classification systems that work well on standard text documents often struggle with hypertext. It is reported that in one experiment, classifiers that were 90% accurate on the widely used Reuters dataset (of standard text documents) scored only 32% on a sample of Yahoo classified pages.

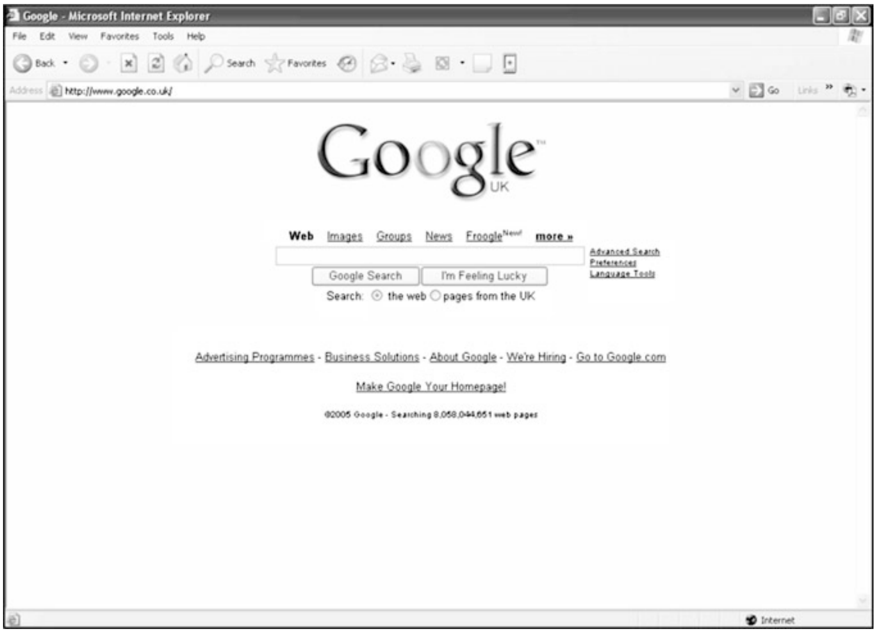
Google - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.google.co.uk/
Go Links
Google UK
Web Images Groups News Froogle New! more »
Advanced Search
Preferences
Language Tools
Google Search
I'm Feeling Lucky
Search: the web pages from the UK
Advertising Programmes - Business Solutions - About Google - We're Hiring - Go to Google.com
Make Google Your Homepage!
©2005 Google - Searching 8,058,044,651 web pages
Internet

**Figure 20.4** Web Page Corresponding to Figure 20.3

To counter the scarcity of textual information in the typical web page we need to try to take advantage of the information given in the tags, links etc. in the HTML markup (whilst of course removing the markup itself before converting the document to a bag-of-words representation or similar).

The information embedded in HTML markup can include:

– a title for the page

– 'metadata' (keywords and a description of the page)

– information about headers etc.

– words considered important enough to place in bold or italic

– the text associated with links to other pages.

How much of this information to include and how to do so is an open research question. We have to beware of 'game playing', where a page deliberately includes misleading information about its content with the aim of fooling internet search engines. Despite this, experience suggests that extracting important words from the markup (especially the 'metadata') and including them in the representation can significantly improve classification accuracy, especially if the words are given greater weighting (say, 3 times greater) than those extracted from the basic text content of the page.

To improve classification accuracy further we could look at the possibility of including some of the information in the 'linked neighbourhood' of each web page, i.e. the pages to which it points and the pages that point to it. However this is considerably beyond the scope of an introductory text.

## 20.10 Chapter Summary

This chapter looks at a particular type of classification task, where the objects are text documents. A method of processing the documents for use by the classification algorithms given earlier in this book using a *bag-of-words representation* is described.

An important special case of text classification arises when the documents are web pages. The automatic classification of web pages is known as hypertext categorisation. The differences between standard text classification and *hypertext categorisation* are illustrated and issues relating to the latter are discussed.

## 20.11 Self-assessment Exercises for Chapter 20

1. Given a document, drawn from a collection of 1,000 documents, in which the four terms given in the table below occur, calculate the TFIDF values for each one.

| Term | Frequency in current document | Number of documents containing term |
|---|---|---|
| dog | 2 | 800 |
| cat | 10 | 700 |
| man | 50 | 2 |
| woman | 6 | 30 |

2. Normalise the vectors $(20, 10, 8, 12, 56)$ and $(0, 15, 12, 8, 0)$.

   Calculate the distance between the two normalised vectors using the dot product formula.