MODEL SOLUTIONS COM3110/COM6150

SETTER(S): Mark Hepple
Mark Stevenson

THE UNIVERSITY OF SHEFFIELD

DEPARTMENT OF COMPUTER SCIENCE

Autumn Semester 2005-06 2 hours

TEXT PROCESSING

Answer THREE questions.

All questions carry equal weight. Figures in square brackets indicate the percentage of available marks allocated to each part of a question.

1. a) Explain what is meant by the "bag of words" model which is used for various text processing tasks. [10%]

answer:
The bag of words model assumes that text can be represented as a simple list of the words it contains and, possibly, their frequencies in the text. Information about the relations between these words and their position in the text is ignored.

b) Explain how the bag of words model can be used in Information Retrieval and word sense disambiguation. [15%]

answer:
The standard approach to IR models each document as a bag of words, and identifies relevant documents by comparing queries to documents using similarity measures that work over such bag-of-words representations.
There are various applications in word sense disambiguation including naive Bayesian classifiers which treat the context of an ambiguous word as a bag of words and the Lesk algorithm for disambiguation using dictionary definition overlap treats each definition as a bag of words.

c) Write down Bayes Rule and explain how it is used in the naive Bayesian classifier for word sense disambiguation. [20%]

answer:
Bayes rule is: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$
The naive Bayesian classifier aims to estimate the probability of a sense given the context. i.e $P(s|c)$, where s is a particular sense and c is the context used by the algorithm. However, this cannot be estimated directly so Bayes rule is applied and the formula re-written as: : $\frac{P(c|s)P(s)}{P(c)}$. $P(c)$ is constant for all senses and can be ignored. We approximate $P(c|s)$ as $P(a_1|s) \times \ldots \times P(a_n|s)$, where $a_1 \ldots a_n$ are the multiple features that make up 'context', i.e. making the naive bayesian assumption that these features are conditionally independent. The probabilities $P(s)$ and $P(a_i|s)$ can be estimated from training text,

d) What are the similarities between part of speech tagging and word sense disambiguation? Why are techniques for part of speech tagging not suitable for word sense disambiguation? [15%]

answer:
Both part of speech tagging and WSD aim to add extra information to each token in a text (or possibly just each content word). Part of speech tagging adds syntactic information while WSD selects amongst word meanings.
Part of speech tagging approaches generally examine a narrow context around the word being annotated, for example the previous one or two words. WSD requires a wider context.

e) Write a short Perl program that will read from a file on `STDIN` and that does nothing with the line that is read *until* it encounters a line containing the string `<BEGIN>`. Thereafter, each subsequent line will be printed (to `STDOUT`) *until* it encounters a line containing the string `<END>`. This line and any that follow are not printed. You may assume that the `<BEGIN>` and `<END>` strings will occur only once in a file. [20%]

answer:
Could be done in more than one way:

e.g. as:

```
$print=0;
while (<>) {
    if (/<BEGIN>/) {
        $print=1;
    } elsif (/<END>/) {
        $print=0;
    } elsif ($print) {
        print;
    }
}
```

or as:

```
while (<>) {
    last if /<BEGIN>/;
}
while (<>) {
    last if /<END>/;
    print;
}
```

f) Assume that, in reading text from an HTML file, we can identify a URL ('web address') expression *within* a line of text provided that the string meets the following requirements:

i. the string should fall between double quotes (")
ii. it should start with `http://`
iii. it should end with `.htm` or `.html`
iv. between this start and end, there may be any sequence of characters except that ", < and > may not appear.

Write a Perl regular expression that will match against strings that contain a URL expression, with the the URL string found being assigned to the variable $1. [10%]

answer:

```
/\"(http:[^\"<>]*\.htm(l)?)\"/
```

g) Specify what will be printed by each of the following pieces of Perl code:

i. [5%]

```
$_ = 'abcde';
s/(\w)(\w)/$2$1/g;
print "$_\n";
```

ii. [5%]

```
$_ = 'abcde';
while (s/(a)(\w)/$2$1/) { }
print "$_\n";
```

answer:

i. gives: `'badce'`

ii. gives: `'bcdea'` – iteratively transposes the a down to the end of the string.

2. a) Describe Lesk's algorithm for word sense disambiguation using dictionary definition overlap and explain its limitations. [25%]

answer:
Lesk's algorithm relies on the textual definitions of words in a dictionary to disambiguate words. The pairwise combinations of senses are considered and the number of words they share in common counted. The pair of senses with the largest number of words in common are chosen as the senses.
To work well the algorithm requires the definitions to be pre-processed in a number of ways. Empty heads and stop words are removed. (Empty heads are short expressions generally found at the start of dictionary definitions which indicate the hypernym of the definition, such as "a type of".) The remaining words are stemmed.
The limitations are:

1. The approach depends on the correct senses sharing words in their definitions. This means that the approach depends on the particular dictionary used and will prefer senses with longer definitions over shorter ones.
2. The approach may not be tractable for long sentences; the number of sense combinations which have to be checked could be prohibitive.

b) Describe two techniques which can be used to annotate text with word sense information automatically without the need for manual annotation. These techniques are used to provide data which can be used to train and/or test a word sense disambiguation system. Describe the disadvantages of each. [15%]

answer:
Two techniques are:

1. Make use of parallel corpora where the sense is defined as the translation of a word into another language.
2. Use pseudowords to automatically introduce ambiguity into text. (Pseudowords are created by choosing two or more words, e.g. car and bottle, and replacing each occurrence of each with their concatenation, car-bottle. The task of the disambiguation algorithm is to identify the original word.)

Their disadvantages are:

1. Parallel text may be hard to obtain.
2. Sense distinctions in pseudowords may not be appropriate for actual applications (e.g. translation). The sense distinctions are artificial and so an algorithm may learn to disambiguate between contexts in which each of the words is likely to occur rather than meanings of the words.

c) What assumption is made by a naive Bayes classifier when it is used for word sense disambiguation? [10%]

answer:
The naive Bayes classifier assumes that the probabilities of the words in the context of the ambiguous words are conditionally independent. That is, the presence, or otherwise, of a word in the context has no influence on other words.

d) Explain the differences between direct, transfer and interlingua approaches to Machine Translation. [30%]

answer:
The key difference between the three approaches is the level of analysis which is applied to the source text.
Direct approaches apply very little analysis to the the source text and rely on simple translation of each word in the source text. Statistical MT could be considered to be a direct approach.
Transfer approaches attempt to analyse the structure of the source text to produce an intermediate representation. The intermediate representation of a sentence from the input text is then used in generating the translated sentence in the target language. Transfer approach can employ syntactic or semantic representations.
Interlingua approaches rely on a representation of the meaning which is independent of both the source and target language. The source sentence goes through syntactic and semantic analysis to be translated into the interlingua. This representation is then used to generate a sentence in the target language. The difference between transfer approaches which use semantic representations and interlingua approaches rests on the independence of the system used to represent meaning; interlinguas are completely independent of the source and target languages while the representation used in semantic transfer simply aims to capture enough information to allow translation.

e) Describe the approach used by the BLEU system for evaluation of Machine Translation systems. [20%]

answer:
The BLEU system relies on multiple reference translations, each of which represents a possible way in which a text could be translated into a target language. The translation being evaluated (the candidate) is compared against the reference translations by counting the number of possible ngrams (strings of words) which occur in them. BLEU assumes that there are several possible ways to translate a text, each of which are equally valid, and uses multiple reference reference translations to provide these alternatives.

3. a) What are stop words, in the context of an Information Retrieval system? Why are they generally not included as index terms? [10%]

answer:
Stop words are generally words belonging to closed grammatical classes such as determiners, conjunctions and prepositions. They are not included as index terms as they occur so frequently in documents that they are not good discriminators between relevant and irrelevant documents.

b) Consider the following collection of three short documents:

Document 1: Tyger, tyger, burning bright
Document 2: Tyger sat on the mat
Document 3: The bright mat

Show how the documents would be represented in a vector space model for Information Retrieval, as vectors in which term weights correspond to term frequencies. Do not remove stop words, and do not use stemming in creating these representations. [10%]

answer:
The documents would be represented as follows:

	bright	burning	mat	on	sat	the	tyger
Document 1	1	1	0	0	0	0	2
Document 2	0	0	1	1	1	1	1
Document 3	1	0	1	0	0	1	0

c) The cosine coefficient can be used by Information retrieval systems to rank the relevance of documents in relation to a query. Compute the similarity that would be produced between Document 1 and the query "burning tyger" using this measure. [15%]

answer:
$cos(Document1, \text{“burning tyger”}) = \frac{1.0+1.1+0.0+0.0+0.0+0.0+2.1}{\sqrt{1^2+1^2+0^2+0^2+0^2+0^2+2^2}\sqrt{1^2+1^2}} = \frac{3}{\sqrt{6}\sqrt{2}} = 0.866$

d) Explain what is meant by term weighting in Information Retrieval systems and why it is used. [15%]

answer:
Term weighting is the process of deciding on the importance of each term and is normally carried out by assigning a numerical score to each term.
Term weighting is used in IR systems because not all terms are equally useful for retrieval; some occur in many of the documents in the collection (and are therefore bad discriminators) while others occur infrequently (and will return few documents). Term weighting aims to assign high scores to terms which are likely to discriminate between relevant and irrelevant documents. The term weights are taken into account by the ranking function with the aim of improving retrieval performance.

e) Explain how tf.idf term weighting is used to assign weights to terms in Information Retrieval. Include the formula for computing tf.idf. [15%]

answer:
tf.idf assigns weights to terms by taking into account two elements: the frequency of that term in a particular document and the proportion of documents in the corpus in which it occurs. The tf part prefers terms which occur frequently in a document and the idf part gives extra weight to terms which do not occur in many documents in the collection. The tf.idf weight is produced by computing the product of these two terms.
The formula for computing tf.idf is:
$tf.idf = tf_{ik} \times log_{10}\left(\frac{N}{n_k}\right)$
where tf_{ik} is the frequency of term k in document i, N is the total number of documents in the collection and n_k the number which contain the term k.

f) Show the weights which would be generated if tf.idf weighting was applied to Document 1 in the document collection of three documents shown above. [10%]

answer:
Using tf.idf the weights in document 1 would become:

	bright	burning	mat	on	sat	the	tyger
Document 1	0.176	0.477	0	0	0	0	0.352

g) Consider the following ranking of ten documents produced by an Information Retrieval system. The symbol ✓ indicates that a retrieved document is relevant and × that it is not.

Document	Ranking	Relevant
d6	1	✓
d1	2	×
d2	3	✓
d10	4	×
d9	5	✓
d3	6	✓
d5	7	×
d4	8	×
d7	9	✓
d8	10	×

Compute the (uninterpolated) average precision and interpolated average precision for this ranked set of documents. Explain your working. [25%]

answer:

Document	Ranking	Relevant	Recall	Precision	Interpolated Precision
d6	1	✓	0.2	1	1
d1	2	×	0.2	0.5	0.67
d2	3	✓	0.4	0.67	0.67
d10	4	×	0.4	0.5	0.67
d9	5	✓	0.6	0.6	0.67
d3	6	✓	0.8	0.67	0.67
d5	7	×	0.8	0.57	0.57
d4	8	×	0.8	0.5	0.56
d7	9	✓	1.0	0.56	0.56
d8	10	×	1.0	0.5	0.5

(Uninterpolated) average precision is computed by averaging the precision score each time a relevant document is found. So, (uninterploated) average precision is given by $\frac{1+0.67+0.6+0.67+0.56}{5} = 0.7$

Interpolated average precision is computed by taking the interpolated precision at 11 sample points (0%, 10%, 20%, ... 100% recall). So, interpolated average precision is given by: $\frac{(3\times1)+(6\times0.67)+(2\times0.56)}{11} = 0.74$

4. We want to create a Perl program that will identify the tokens appearing in two text files (ignoring stop words), and then use this information to compute a measure of the similarity of the two files, using a metric that is detailed below.

   a) Write a Perl subroutine `tokenise` which takes a single input parameter, which is a string (corresponding to a line of input from a text file), and returns a list of strings for the *tokens* appearing in the input. For example, given the input string `"Today, John found anti-matter.\n"`, the subroutine would return the list `("today", "john", "found", "anti", "matter")`. Assume that tokens are any consecutive sequence of alphabetic characters, with any non-alphabetic characters (including punctuation) serving to separate tokens, as the example illustrates. We are not interested in case distinctions, e.g. `John` vs. `john`, so the tokens should be mapped to lowercase. [25%]

answer:

A compact solution is:

```
sub tokenise {
    return split("[^a-z]+",(lc $_[0]));
}
```

A less terse, but still good, answer might be:

```
sub tokenise {
    my ($line) = @_;
    $line =~ tr/A-Z/a-z/;
    $line =~ s/[^a-z]/ /g;
    return split(' ',$line);
}
```

   b) Write a Perl subroutine `readStoplist` that takes as its single parameter the name of a file that contains a list of stopwords, and which reads in and stores the stopwords for subsequent use. As part of your solution, you will need to determine a good data structure for storing the words, i.e. one allowing them to be efficiently accessed. Define also a subroutine `isaStopword` which when given a string for a token, determines whether or not it is a stopword, i.e. so `isaStopword($w)` returns `1` if the value of `$w` is a stopword, and returns `0` otherwise. [25%]

answer:

A good solution for storing the stopwords is as the keys of a hash, where the associated value is just `1` - as this allows us to check if some word is stored here or not very efficiently. Solutions where the stopwords are stored in a list, which is traversed to check the status of some token, will receive lesser credit.

```
sub readStoplist {
    my ($file) = @_;
    open(STOP,"< $file")
        || die "ERR: problem opening file ($file)\n";
    while (<STOP>) {
        chop;
        $stops{$_} = 1;
    }
    close STOP;
}
```

Given the use of a hash, this definition has little work to do (mostly ensuring that `0` is returned rather than `undef`). If the stopwords have been stored in a list, then this subroutine will have more work to do.

```
sub isaStopword {
    my ($w) = @_;
    if ($stops{$w} > 0) {
        return 1;
    } else {
        return 0;
    }
}
```

c) Write a Perl subroutine `countTokens` that takes a file name as its single parameter, and which counts the occurrences of tokens in the file, ignoring stopwords, and returns its results as a hash, i.e. a hash in which each key is a token appearing in the file, for which the associated value is a positive integer indicating how many times the token appears. Your definition should use the subroutines specified in parts (a) and (b) of the question (irrespective of whether you were able to provide working definitions for them). [25%]

answer:

```
sub countTokens {
    my ($file) = @_;
    my ($w,@wds,%counts);
    open(IN,"<$file")
        || die "ERROR: problem opening file ($file)\n";
    while (<IN>) {
        @wds = tokenise($_);
        foreach $w (@wds) {
            unless (isaStopword($w)) {
                $counts{$w}++;
            }
        }
    }
    return %counts;
}
```

d) Write a Perl *program* which requires three arguments on the command line, of which the first is the name of the stopword file, and the others are the names of two files that are to be compared, i.e. which might be invoked as:

```
perl myprogram.pl stoplist.txt file1.txt file2.txt
```

The program should use the previously specified subroutines to load the stopwords, and to count the tokens in the other two files, so that this information can be used to compute a similarity score for the two files. The measure to be computed is the proportion of terms (i.e. distinct tokens) appearing in *either* file that appear in *both* files. Note that this particular measure makes no use of the specific *counts* that have been made of token occurrences, only whether or not a specific distinct token appears in a given file or not. If A is the *set* of terms (irrespective of counts) appearing in one file, and B the corresponding set for the other file, then this measure can be expressed as:

$$\frac{|A \cap B|}{|A \cup B|}$$

The calculation of this measure can be done either in a subroutine or in the main body of the code, as you prefer. Your program should print the similarity value of the two files to STDOUT before terminating. [25%]

answer:

```
@ARGV == 3
    || die "ERROR: must specify 3 input files\n";

($stopFile,$file1,$file2) = @ARGV;

readStoplist($stopFile);
%counts1 = countTokens($file1);
%counts2 = countTokens($file2);

foreach $w (keys %counts1) {
    $allTokens{$w}=1;
}
foreach $w (keys %counts2) {
    $allTokens{$w}=1;
}

$inBothCount=0;
foreach $w (keys %allTokens) {
    if ($counts1{$w} * $counts2{$w} > 0) {
        $inBothCount++;
    }
}

if (0 < (keys %allTokens)) {
    printf "Similarity: %.2f\n", $inBothCount/(keys %allTokens);
} else {
    print "Similarity: 0\n";
}
```