The University of Sheffield

DEPARTMENT OF COMPUTER SCIENCE

Autumn Semester 2004-2005 **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. Provide short answers to the following questions.

a) Let $S = \{a, b, c\}$ (the sample space) and let P be the joint distribution on an ordered sequence of two events (i.e., on $S \times S$ ordered). If you know that:

$$\begin{aligned} P(a,a) &= 0.25 \\ P(a,b) &= 0.125 \\ P(b,c) &= 0.125 \\ P(c,a) &= 0.25 \\ P(c,c) &= 0.25 \end{aligned}$$

Can you estimate $P(b|c)$ (i.e., the probability of seeing b if we have already seen c)? If your answer to the question is yes, compute $P(b|c)$ using the relevant formula as part of your answer.

[10%]

b) Write down one example of a data sequence which faithfully follows the distribution given in the previous question. In other words write down a possible sequence of as, bs, and cs from which we could get the above bigram distribution using maximum likelihood estimation.

[10%]

c) What is Zipf's law? What does it tell us about the relationship between the frequency of a word f and its rank r? Give the relevant formula as part of your answer.

[10%]

d) What is the noisy channel model? Given an example of a text processing application that is based on the noisy channel model. As part of your answer draw a picture of the noisy channel model.

[20%]

e) What is the purpose of the following Unix commands? Explain what each successive command (separated by |) does.

```
cat genesis.txt | tr -sc 'A-Za-z' '\012' | tr '[A-Z]' '[a-z]' | grep
'^a.*e$' | sort | uniq -c | sort -nr |
awk '{x += $1} END {print x}'
```

[10%]

f) Specify a pipeline of Unix tools which takes as input a plain text file and produces as output a list of words that begin with two vowels and end with a non-vowel. Examples of such words are *eat* and *ought*. The vowels in English are *a, e, i, o, u, A, E, I, O, U.*

[10%]

QUESTION CONTINUES ON NEXT PAGE

g) What is printed out by each of the following Perl programs?

(i)
```
$_ = 'bananarama';
s/(an)*a/$1/;
print;
```

(ii)
```
$_ = 'uggawuggahuggamugga';
s/((a)(.)(u))/$2$4/g;
print;
```

[10%]

h) What is printed out by each of the following print statements?

```
@large_streets = qw(main broad high);
@small_streets = ("berry", "apple");
@streets = (@large_streets, @small_streets);
print "@streets\n";
print "@streets[1..3]\n";
$a = @streets;
print "$a\n";
print "$#streets\n";
$alley = "sparrow";
$street[8] = $alley;
print "@streets\n";
print "$#streets\n";
```

[20%]

2. a) Probabilistic language models are very popular in text processing. Using the notation of conditional probabilities, write down an expression for the exact probability of a sentence containing l words $w_1 w_2 \ldots w_l$. Now write down an approximation to this. The approximation should use only unigram and bigram probabilities (i.e., those having the form $P(w_k)$ where $k \in 1 \ldots l$ or $P(w_i|w_{i-1})$ where $i \in 2 \ldots l$). Briefly explain what information is used in the first expression but ignored in the second. Why might one choose to use the second form of the expression in preference to the first?

[10%]

b) Suppose you have a corpus that consists of the two documents below:

Document 1
I will eat a Chinese food lunch.

Document 2
I speak Chinese at lunch.

Create a unigram and a bigram language model from the above corpus.

[20%]

QUESTION CONTINUES ON NEXT PAGE

c) Calculate the probability of the following sentence according to the bigram and unigram language models you built from the above corpus:

I will speak Chinese.

If you encounter word combinations with zero frequency in the corpus, use Add-one smoothing (Laplace's law) to recreate the missing counts.

[30%]

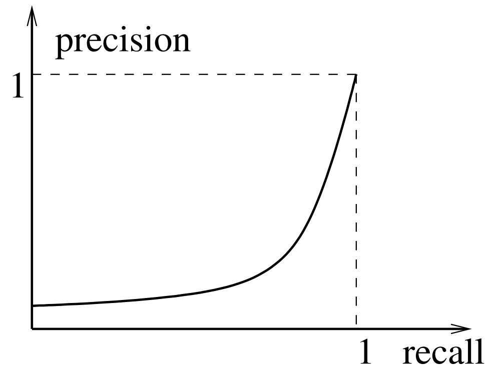
d) What is the per-word entropy for the above corpus? (Useful information: $\log_2 \frac{2}{12} = -2.6$ and $\log_2 \frac{1}{12} = -3.6$).

[10%]

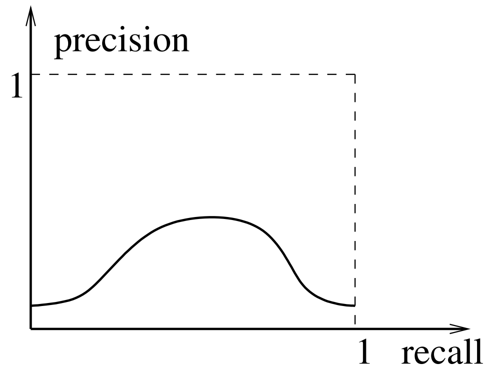
e) Create a document-by-word matrix for Documents 1 and 2 without stemming and without removing stopwords. Assume the matrix cells represent the number of times a word appears in Documents 1 and 2. Calculate the similarity of Documents 1 and 2 using the Euclidean measure.

[10%]

f) Define the precision and recall measures in IR. Is it possible for a precision/recall graph to look like (a) or (b). Explain why.



(a)



(b)

[20%]

3. a) When using the noisy channel model for machine translation, we estimate $\underset{E}{\operatorname{argmax}} P(E|F)$ by rewriting using Bayes rule to produce $\underset{E}{\operatorname{argmax}} P(E)P(F|E)$. Why do we do this? What would happen if we simply directly modeled $P(E|F)$ and did without the language model $P(E)$? Give one or more reasons why the rewritten model should produce better English translations.

[25%]

QUESTION CONTINUES ON NEXT PAGE

b) Consider a simple translation model in which we simply map words from one language to another, and allow words in any position to be equally likely to appear in any position in the translation. In effect, the translation model makes every ordering of the translated words equally likely. For example, the French sentence *Je t'aime* would produce equally likely sequences *I you love*, *I love you*, *you I love*, *you love I*, *love I you*, and *love you I*. Could we still produce reasonable translations using this translation model. Why? In what cases would it tend to fail? Give an example.

[25%]

c) Write a Perl program that takes two files as input that are translations of each other and have been aligned at the sentence level, and prints out all possible word alignments together with their frequencies. For example, if the input files were:

File 1 (English)
I eat chicken
Chicken is good

File 2 (French)
Je mange du poulet
Le poulet est bon

then the word *I* can be aligned to *je*, *mange*, *du*, and *poulet*. The word *eat* can be aligned to *je*, *mange*, *du*, and *poulet*, etc. Assume that each word in the English file can be aligned to exactly one word in the French file. The output of the program for Files 1 and 2 would be:

```
1       good    est
1       is      poulet
1       eat     du
1       i       poulet
1       is      le
1       good    bon
1       is      bon
1       chicken mange
1       good    le
1       eat     je
1       i       je
1       i       du
2       chicken poulet
1       chicken est
1       good    poulet
1       chicken bon
1       i       mange
1       chicken je
1       chicken du
1       chicken le
1       eat     mange
1       is      est
1       eat     poulet
```

QUESTION CONTINUES ON NEXT PAGE

Take a word to be any consecutive sequence of alphabetic characters separated by white space from any adjacent word. Assume that the input contains only alphabetic characters, spaces and line brakes, i.e., there is no punctuation or numeric data. Your program should normalize the words to lower case, as shown in the output example above.

[50%]

4. a) One of the most popular applications of text processing is automatic summarisation. Briefly describe why automatic summarisation is a challenging task. What information should an automatic summariser take into account so that it produces good summaries? What are the limitations of methods based solely on sentence extraction?

[30%]

b) Describe Kupiec et al.'s (1995) approach to document summarisation. Your answer should discuss: (a) what their training corpus consists of; some of the features used to detect important information in the texts; and how their system determines whether a sentence should be included in the summary or not.

[20%]

c) Write a Perl program that takes two arguments, a number and a document. Your program will summarize the document and the number stands for the compression rate, i.e., the number of sentences the user wants to be included in the output summary. Write a Perl summariser that extracts the *first N* sentences of the document, where *N* is the compression rate. Assume that the document has been preprocessed, i.e., it has been tokenised and every sentence corresponds to a separate line. So if the input to your program was:

Doc1.txt
Harry Potter and the Prisoner of Azkaban earned 34.9 million US dollars last week. The third Harry Potter movie has grossed 157.9 million dollars since its release. Disney's new movie The Chronicles of Riddick, earned 24.2 million dollars despite the bad reviews. Meanwhile, Shrek 2 earned 23.3 million dollars.

And the compression rate was 2, then you'd call your program as follows:

```
summarise.pl 2 Doc1.txt
```

And the output of the program would be the first two sentences:

Summary
Harry Potter and the Prisoner of Azkaban earned 34.9 million US dollars last week. The third Harry Potter movie has grossed 157.9 million dollars since its release.

[20%]

d) Modify the output of your summarisation program so that it outputs the *N longest* sentences in the corpus rather than the *N* first ones. Measure sentence length by the number of words in the sentence. Take a word to be any consecutive sequence of characters (including numbers, letters, and punctuation) separated by white space from any adjacent word.

QUESTION CONTINUES ON NEXT PAGE

So if you run your summariser using the same command as before (i.e., `summarise.pl 2 Doc1.txt`) the output now will be the following:

Doc1.txt
Disney’s new movie The Chronicles of Riddick, earned 24.2 million dollars despite the bad reviews. Harry Potter and the Prisoner of Azkaban earned 34.9 million US dollars last week.

Your program should sort (and output) the sentences according to their length. As you can see above, *Disney’s new movie The Chronicles of Riddick, earned 24.2 million dollars despite the bad reviews* is presented (i.e., ranked) before *Harry Potter and the Prisoner of Azkaban earned 34.9 million US dollars last week* since it is longer (15 vs. 14 words).

[30%]

END OF QUESTION PAPER

Solutions for COM3110 Exam 2004-2005
Setter: Mirella Lapata

1. a) The conditional probability $P(b|c)$ can be estimated since all the given probabilities ($P(a,a)$, $P(a,b)$, $P(b,c)$, $P(c,a)$, and $P(c,c)$) sum up to one and therefore P is fully defined. Here's how $P(b|c)$ can be derived:

$$P(b|c) = \frac{P(b,c)}{P(c)} = \frac{0.125}{0.375} = 0.33333333$$

$$P(c) = P(b,c) + P(c,c) = 0.375$$

   b) Examples of data sequences that generated the distribution in question (1a) are the following:

$$c\ a\ a\ b\ c\ c\ c\ a\ a$$

$$c\ a\ b\ c\ c\ c\ a\ a\ a$$

   c) Zipf observed that if we count how often each word (type) of a language occurs in a large corpus, and then list the words in the order of their frequency of occurrence, we can explore the relationship between the frequency of a word f and its position in the list, known as its *rank* r. Zipf's law says that:

$$f \propto \frac{1}{r}$$

   Or in other words that there is a constant k such that $f \times r = k$. Zipf's law is useful as a rough description of the frequency distribution of words in human languages. There are a few very common words, a middling number of medium frequency words, and many low frequency words.

   d) Using information theory, Shannon modeled the goal of communicating down a telephone line – or in general across any channel – in the following way. Imagine communication along a low-quality telephone line where the sender creates a message which is being transmitted over a channel with limited capacity; the receiver has to guess what the original message was. It is assumed that the output of the channel depends probabilistically on the input. The goal is to encode the message in such a way that it occupies minimal space while still containing enough redundancy to be able to detect and correct errors. On receipt, the message is decoded to give what was most likely the original message.

$$\begin{tikzcd} {} \arrow[r, "W"] & \boxed{\text{Encoder}} \arrow[r, "X"] & \boxed{\begin{array}{c}\text{Noisy Channel}\\ p(y|x)\end{array}} \arrow[r, "Y"] & \boxed{\text{Decoder}} \arrow[r, "\hat{W}"] & {} \end{tikzcd}$$

   A simplified version of the noisy channel model has been used to model the machine translation process. As an example, suppose we want to translate a text from English to French. The channel model for translation assumes that the true text is French, but that unfortunately when it was transmitted to us, it went through a noisy communication channel and came out as English. All we need to do in order to translate is to recover the original French, i.e., to decode the English to get the French.

e) The Unix commands tokenise the text, translate all alphabetic characters to lower case, print each word in a separate line, grep all words starting with an `a` and ending with an `e` (an example of such a word would be *are* or *age*), sort them alphabetically, squeeze and count repetitions, sort numerically in descending order, and count the total number of occurrences of these words (i.e., words with an `a` and ending with an `e`) in the corpus.

f)
```
cat genesis.txt | tr -sc 'A-Za-z' '\012' |
grep '^[aeiouAEIOU][aeiouAEIOU].*[^aeiouAEIOU]$'
```

g) (i) Prints `banrama`

(ii) Prints `uggauggauggaugga`

h)

`print "@streets\n";`	Prints `main broad high berry apple`
`print "@streets[1..3]\n";`	Prints `broad high berry`
`print "$a\n";`	Prints `5`
`print "$#streets\n";`	Prints `4`
`print "@stree ts\n";`	Prints `main broad high berry apple     sparrow` spaces indicate that the array will have 2 empty elements
`print "$#streets\n";`	Prints `8`

2. a) Here's the exact expression for the probability of a sentence containing l words:.

$$P(w_1 \dots w_l) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1,w_2) \dots P(w_l|w_1 \dots w_{l-1})$$

The above can be approximated as follows:

$$P(w_1 \dots w_l) \approx P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdots P(w_l|w_{l-i})$$

Contextual information is ignored in the second expression. We might chose the second expression over the first one, because the second one will be easier to estimate (i.e., it has less parameters).

b)

Unigram Model	
I	2
will	1
a	1
eat	1
Chinese	2
food	1
at	1
lunch	2
speak	1

Bigram Model									
	I	will	a	eat	Chinese	food	at	lunch	speak
I	0	0	0	0	0	0	0	0	0
will	1	0	0	0	0	0	0	0	0
a	0	0	0	1	0	0	0	0	0
eat	0	1	0	0	0	0	0	0	0
Chinese	0	0	1	0	0	0	0	0	1
food	0	0	0	0	1	0	0	0	0
at	0	0	0	0	1	0	0	0	0
lunch	0	0	0	0	0	1	1	0	0
speak	1	0	0	0	0	0	0	0	0

c) The size of the corpus is $N = 12$.

$$\begin{aligned} P(I\ will\ speak\ Chinese) &= P(I) \cdot P(will) \cdot P(speak) \cdot P(Chinese) \\ &= \frac{f(I)}{N} \cdot \frac{f(will)}{N} \cdot \frac{f(speak)}{N} \cdot \frac{f(Chinese)}{N} \\ &= \frac{2}{12} \cdot \frac{1}{12} \cdot \frac{1}{12} \cdot \frac{2}{12} \\ &= 0.0001929012 \end{aligned}$$

$$\begin{aligned}
P(I\ will\ speak\ Chinese) &= P(I)\cdot P(will|I)\cdot P(speak|will)\cdot P(Chinese|speak)\\
&= \tfrac{f(I)}{N}\cdot\tfrac{f(I,will)}{f(I)}\cdot\tfrac{f(will,speak)}{f(will)}\cdot\tfrac{f(speak,Chinese)}{f(speak)}\\
&= \tfrac{2+1}{12+9}\cdot\tfrac{1+1}{2+9}\cdot\tfrac{0+1}{1+9}\cdot\tfrac{1+1}{1+9}\\
&= 0.14\cdot 0.18\cdot 0.1\cdot 0.18\\
&= 0.0004536
\end{aligned}$$

The formula for Add-one smoothing is given below:

$$P(w_2|w_1) = \frac{C(w_1,w_2)+1}{C(w_1)+V}$$

where $C(w_1,w_2)$ and $C(w_1)$ is the number of times w_1,w_2 and w_1 appear in the corpus and V the size of the vocabulary. In our case V is 9. We can estimate $P(speak|will)$ as follows:

$$P(speak|will) = \tfrac{f(will,speak)+1}{f(speak)+V} = \tfrac{0+1}{1+9} = 0.1$$

Notice that smoothing applies to all word combinations in the corpus. So, it is not enough to smooth only the unseen bigrams; the frequencies of the seen bigrams must also be updated.

d)

$$\begin{aligned}
H(C) &= -\sum_{i\in\{I,will,eat,a,Chinese,food,lunch,,speak,at\}} p(i)\log_2 p(i)\\
&= -[3\cdot\tfrac{2}{12}\log_2\tfrac{2}{12} + 6\cdot\tfrac{1}{12}\log_2\tfrac{1}{12}]\\
&= -[-1.3-1.8]\\
&= 3.1 \text{ bits}
\end{aligned}$$

e) This is the document-by-word matrix for Documents 1 and 2.

This is the document-by-word matrix for Documents 1 and 2.

	I	will	eat	a	Chinese	food	lunch	speak	at
Document 1	1	1	1	1	1	1	1	0	0
Document 2	1	0	0	0	1	0	1	1	1

The *Euclidean* distance of two vectors $\vec{x}$ and $\vec{y}$ is:

$$\begin{aligned}
|\vec{x}-\vec{y}| &= \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}\\
|\vec{Doc}\ 1 - \vec{Doc}\ 2| &= \sqrt{2}\\
&= 1.4142135623731
\end{aligned}$$

f) Assuming that: RET is the set of all documents the system has retrieved for a specific query; REL is the set of relevant documents for a specific query; $RETREL$ is the set of the retrieved relevant documents, i.e., $RETREL = RET \cap REL$. The the precision is defined as follows: precision = $|RETREL|/|RET|$ and recall is defined as $|RETREL|/|REL|$.

It is not possible for a graph to look like (a). It is impossible for the curve to touch the (1,1) point. At (1,1), it means that the both the recall and the precision is one, meaning that all of the documents retrieved are relevant. However, this contradicts the fact that precision is less than zero when recall is low, which means that some irrelevant documents are already retrieved.

It is possible for a graph to look like (b). The curve means that there are relatively few relevant documents at the beginning and end of the retrieved documents but some relevant documents are concentrated in the middle of the retrieved documents.

3. a) Machine translation can be cast in terms of the noisy-channel model. The approach was pioneered by IBM in the early 90's. If we want to translate French into English, we shall assume that someone has E in his head but by the time he writes it down it's *corrupted* and becomes F. To recover the original E, we need to reason about: (a) the kinds of things people say in English and (b) how English gets turned into French. Given a French sentence F, the approach searches for English E that maximises $P(E|F)$.

$$\begin{aligned}\underset{E}{\operatorname{argmax}} P(E|F) &= \underset{E}{\operatorname{argmax}} \tfrac{P(E)\cdot P(F|E)}{P(F)} \\ &= \underset{E}{\operatorname{argmax}} P(E)\cdot P(F|E)\end{aligned}$$

We choose not to model $P(E|F)$ directly. Instead, we break things apart to get good translations even if the probability numbers are not that accurate. $P(F|E)$ will ensure that a good E will have words that generally translate to words in F. $P(E)$ will help save us from bad English and $P(F|E)$ only needs to say whether a bag of English words corresponds to a bag of French words. $P(E)$ and $P(F|E)$ can be trained independently.

b) Such a model would produce several ungrammatical translations. In fact, there would be a large number of English translations corresponding to a single French sentence. The model would need to be complemented with a mechanism that picks one translation from the set of possible translations. Since every ordering is equally likely, the selection process cannot be done probabilistically. One solution would be to select a random order. This of course entails that we are not guaranteed to have a good or even grammatical translation. The model will fail to capture the fact that there is no one-to-one mapping from one language to another. Several words in the source language may be rendered by a single word in the target language. Furthermore, even in cases where there is a direct mapping, the order of words may differ from one language to the other. The naive model fails on both accounts.

c)
```
#!/usr/local/bin/perl

$english = shift(@ARGV);
$french = shift(@ARGV);

$position = 0;
open(EN,$english);
while ($e = <EN>) {
    ++$position;
```

```
        chop $e;
        $e =~ tr/A-Z/a-z/;
        $hash_english{$position} = $e;
    }
    close(EN);

    $position = 0;
    open(FR,$french);

    while ($f = <FR>) {
        ++$position;
        chop $f;
        $f =~ tr/A-Z/a-z/;
        $hash_french{$position} = $f;
    }
    close(FR);

    foreach $member (keys %hash_english) {
        @english = split(/\s+/,$hash_english{$member});
        @french = split(/\s+/,$hash_french{$member});

        foreach $m1 (@english) {
            foreach $m2 (@french) {
                $alignment{$m1."\t".$m2} += 1;
            }
        }
    }

    foreach $a (keys %alignment) {
        print "$alignment{$a}\t$a\n";
    }
```

20% for using two hashes for storing the sentences and their positions in the document, 20% for using a double foreach loop for enumerating and storing the alignments, 10% for printing the right output. Full credit will be given for general solutions (using the hash data structure) as opposed to simpler alternatives which are inadequate, i.e., they don't solve the problem in all instances.

4. a) Summarization – the art of abstracting key content from one or more information sources – has become an integral part of everyday life. People keep abreast of world affairs by listening to news bites. They base investment decisions on stock market updates. They even go to movies largely on the basis of reviews they've seen. With summaries, they can make effective decisions in less time. Although some summarizing tools are already available, with the increasing volume of online information, it is becoming harder to

generate meaningful and timely summaries. Tools such as Microsoft s AutoSummarize option in Office 97, are useful, but their application is limited to *extraction* selecting original pieces from the source document and concatenating them to yield a shorter text. *Abstraction*, in contrast, paraphrases in more general terms what the text is about.

The concatenation approach to extraction does little to ensure that the summary is coherent, which can make the text hard to read. Moreover, the source may not always have text for example, a sports event on videotape or tables displaying economic data and current tools cannot summarize non-textual media. Finally, these tools do not currently handle multiple sources. For example, there may be many stories on the Web about a particular news event, and it would be useful if the summarizer could capture common and new information. To address these limitations, researchers are looking at a variety of approaches, which roughly fall into two categories. Knowledge-poor approaches rely on not having to add new rules for each new application domain or language. Knowledge-rich approaches assume that if you grasp the meaning of the text, you can reduce it more effectively, thus yielding a better summary. They rely on a sizeable knowledge base of rules, which must be acquired, maintained, and then adapted to new applications and languages. The two classes of methods are not necessarily mutually exclusive. In fact, some approaches use a hybrid.

In both methods, the main constraint is the compression requirement. Extracts of single documents usually aim to be five to 30 percent of the source length. However, compression targets in summarizing multiple sources or in providing summaries for handheld devices are much smaller. These high reduction rates pose a challenge because they are hard to attain without a reasonable amount of background knowledge.

Another challenge is how to evaluate summarizers. If you are to trust that the summary is indeed a reliable substitute for the source, you must be confident that it does in fact reflect what is relevant in that source. Hence, methods for creating and evaluating summaries must complement each other.

10% for discussing the limitations of extractive summarisation, 10% for discussing the output of summarisers and the fact that it is not particularly coherent or informative, and 10% for mentioning problems with respect to evaluation.

b) The system presented by Kupiec et al (1995) is a modified Naive Bayes classifier. Their training corpus consists of document-summary pairs from a number of scientific journals. Before training, they run a sentence matching algorithm to determine which sentences in the document correspond to sentences in the summary. Instead of doing sub-sentence information extraction, the system concentrates on sentence selection, so sentences in the training data are marked as being part of the summary or not. The system examines a set of features for each sentence, all of which are discrete:

Sentence Length Cut-off Feature: Short sentences tend to not be included in summaries. Sentences longer than a certain threshold have a value of true, and the others are false.

Fixed-Phrase Feature: Sentences containing or following a number of key phrases tend to be included in summaries. Kupiec et al. use 26 indicator phrases. Sentences containing those phrases, or ones following section heads with key words, have a value of true. All other sentences have a value of false.

Paragraph Feature: Sentences are categorized as being the initial sentence in a paragraph, a medial sentence, or a final sentence.

Thematic Word Feature: The most frequent content words are designated theme words. Sentences are ranked based on the frequency of theme words. This feature is binary, depending on whether a sentence is among the highest ranked sentences.

Uppercase Word Feature: Proper names and acronyms tend to be important. This feature is similar to the previous one, with sentences ranked according to the frequency of uppercase words that appear more than once in the document.

For each sentence s, the system determines the probability that it will be included in a summary S, given features $F_1, \ldots, F_k$ using a Bayes decision rule:

$$P(s \in S | F_1, F_2, \ldots Fk) = P(F_1, F_2 \ldots F_k | s \in S) \times \frac{P(s \in S)}{P(F_1, F_2, \ldots F_k)}$$

Assuming that the features are statistically independent, the above becomes:

$$\frac{P(s \in S) \prod_{j=1}^{k} P(F_j | s \in S)}{\prod_{j=1}^{k} P(F_j)}$$

Thus, the system is essentially a naive Bayes classifier. $P(s \in S)$ is constant, and $P(F_j | s \in S)$ and $P(F_j)$ are estimated from the training set by counting occurrences.

5% for giving the details of the corpus, 10% for mentioning some of the features Kupiec et al. are using (full credit will be given if 3 features are mentioned), and 5% for describing the summary model (full credit will be given if the Naive Bayes classifier is described informally).

c)
```
#!/usr/local/bin/perl -w

$compression = shift(@ARGV);
$document = shift(@ARGV);

$counter = 0;

open(DOC,$document);
while ($line = <DOC>) {
    chop $line;
    ++$counter;
    if ($counter <= $compression) {
        print "$line\n";
    }
}

close(DOC);
```

10% for passing the arguments correctly and 10% for implementing the inequality test right

d)

```
#!/usr/local/bin/perl -w

$compression = shift(@ARGV);
$document = shift(@ARGV);

open(DOC,$document);
while ($line = <DOC>) {
    chop $line;
    @words = split(/\s+/,$line);
    $sentence{$line} = ($#words + 1);

}

$counter = 0;

foreach $key (sort hashValueDescendingNum (keys(%sentence))) {
    ++$counter;
    if ($counter <= $compression) {
        print "$key $sentence{$key}\n";
    }
}

sub hashValueDescendingNum {
   $sentence{$b} <=> $sentence{$a};
}
```

10% for the sorting function, 10% for counting the sentence length, and 10% for implementing the compression cutoff