PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 :
G06F 17/27, 17/21, 15/00, 17/00

A1

(11) International Publication Number: **WO 00/41096**

(43) International Publication Date: 13 July 2000 (13.07.00)

(21) International Application Number: PCT/US00/00268

(22) International Filing Date: 6 January 2000 (06.01.00)

(30) Priority Data:

60/115,016	7 January 1999 (07.01.99)	US
09/351,952	12 July 1999 (12.07.99)	US

(71) Applicant: JUSTSYSTEM PITTSBURGH RESEARCH CENTER, INC. [US/US]; 4616 Henry Street, Pittsburgh, PA 15213 (US).

(72) Inventors: WITBROCK, Michael, J.; 294 Waverly Avenue, Newton, MA 02458 (US). MITTAL, Vibhu, O.; 4146 Murray Avenue, Pittsburgh, PA 15217 (US).

(74) Agents: BYRNE, Richard, L. et al.; Webb Ziesenheim Logsdon Orkin & Hanson, P.C., 700 Koppers Building, 436 Seventh Avenue, Pittsburgh, PA 15219–1818 (US).

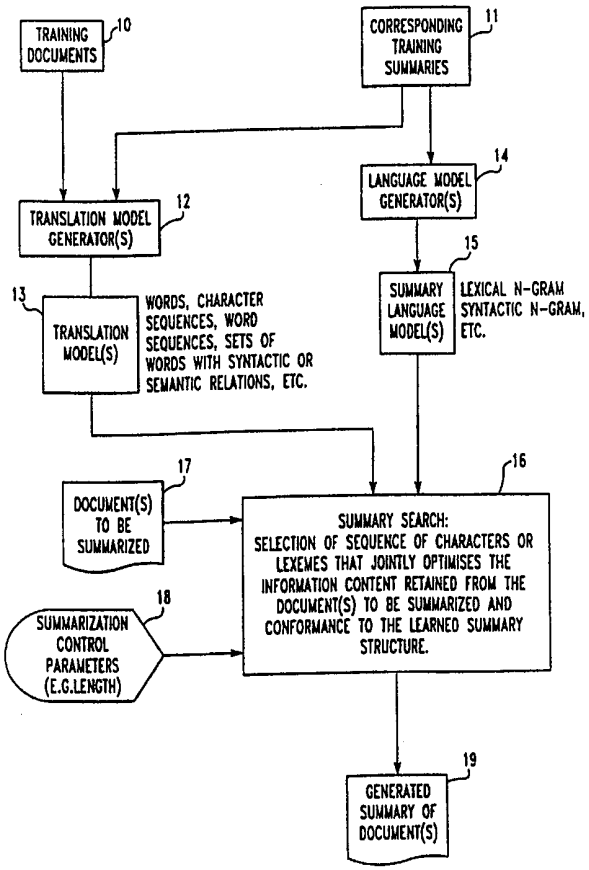
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: METHOD FOR PRODUCING SUMMARIES OF TEXT DOCUMENT

(57) Abstract

A computer method for preparing a summary string (19) from a source document of encoded text (17). The method comprises comparing a training set of encoded text documents (10) with manually generated summary strings (11) associated therewith to learn probabilities (13) that a given summary word or phrase will appear in summary strings (19) given a source word or phrase appears in encoded text documents (17) and constructing from the source document a summary string containing summary words or phrases (19) having the highest probabilities of appearing in a summary string (19) based on the learned probabilities established in the previous step.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD FOR PRODUCING SUMMARIES OF TEXT DOCUMENT

BACKGROUND OF THE INVENTION

Extractive summarization is the process of selecting and extracting text spans--usually whole sentences--from a source document. The extracts are then arranged in some order (usually the order as found in the source document) to form a summary. In this method, the quality of the summary is dependent on the scheme used to select the text spans from the source document. Most of the prior art uses a combination of lexical, frequency and syntactic cues to select whole sentences for inclusion in the summary. Consequently, the summaries cannot be shorter than the shortest text span selected and cannot combine concepts from different text spans in a simple phrase or statement. U.S. Patent No. 5,638,543 discloses selecting sentences for an extractive summary based on scoring sentences based on lexical items appearing in the sentences. U.S. Patent No. 5,077,668 discloses an alternative sentence scoring scheme based upon markers of relevance such as hint words like "important", "significant" and "crucial". U.S. Patent No. 5,491,760 works on bitmap images of a page to identify key sentences based on the visual appearance of hint words. U.S. Patent Nos. 5,384,703 and 5,778,397 disclose selecting sentences scored on the inclusion of the most frequently used non-stop words in the entire text.

In contrast to the large amount of work that has been undertaken in extractive summarization, there has been much less work on generative methods of summarization. A generative method of summarization selects words or phrases (not whole sentences) and generates a summary based upon the selected words or phrases. Early approaches to generative methods are discussed in the context of the FRUMP system. See DeJong, G.F., "An Overview of the FRUMP System", Strategies for Natural Language Processing, (Lawrence Erlbaum Associates, Hillsdale, NJ 1982). This system provides a set of templates for extracting information from news stories and presenting it in the form

of a summary. Neither the selection of content nor the generation of the summary is learned by the system. The selection templates are handcrafted for a particular application domain. Other generative systems are known. However, none of these systems can: (a) learn rules, procedures, or templates for content selection and/or generation from a training set or (b) generate summaries that may be as short as a single noun phrase.

The method disclosed herein relates somewhat to the prior art for statistically modeling of natural language applied to language translation. U.S. Patent No. 5,510,981 describes a system that uses a translation model describing correspondences between sets of words in a source language and sets of words in a target language to achieve natural language translation. This system proceeds linearly through a document producing a rendering in the target language of successive document text spans. It is not directed to operate on the entire document to produce a summary for the document.

SUMMARY OF THE INVENTION

As used herein, a "summary string" is a derivative representation of the source document which may, for example, comprise an abstract, key word summary, folder name, headline, file name or the like. Briefly, according to this invention, there is provided a computer method for generating a summary string from a source document of encoded text comprising the steps of:

a) comparing a training set of encoded text documents with manually generated summary strings associated therewith to learn probabilities that a given summary word or phrase will appear in summary strings given that a source word or phrase appears in an encoded text document; and

b) from the source document, generating a summary string containing a summary word, words, a phrase or phrases having the highest probabilities of appearing in a summary string based on the learned probabilities

established in the previous step. Preferably, the summary string contains the most probable summary word, words, phrase or phrases for a preselected number of words in the summary string.

In one embodiment, the training set of encoded manually generated summary strings is compared to learn the probability that a summary word or phrase appearing in a summary string will follow another summary word or phrase. Summary strings are generated containing the most probable sequence of words and/or phrases for a preselected number of words in the summary string.

In a preferred embodiment, the computer method, according to this invention, comprises comparing a training set of encoded text documents with manually generated summary strings associated therewith to learn the probabilities that a given summary word or phrase will appear in summary strings given a source word or phrase appears in the encoded text considering the context in which the source word or phrase appears in the encoded text documents. For example, the context in which the source words or phrases may be considered includes titles, headings, standard paragraphs, fonts, bolding, and/or italicizing.

In yet another preferred embodiment, the computer method, according to this invention, further comprises learning multiple probabilities that a summary word or phrase will appear in a summary string given a source word or phrase appears in the encoded text and considering the various usages of the word or phrase in the encoded text, for example, syntactic usages and semantic usages.

In a still further preferred embodiment, according to this invention, the step for comparing a training set of encoded manually generated summary strings takes into consideration external information in the form of queries, user models, past user interaction and other biases to optimize the form of the generated summary strings.

BRIEF DESCRIPTION OF THE DRAWING

Further features and other objects and advantages will become clear from the following detailed description made with reference to the drawing which is a schematic diagram illustrating the processing of text to produce summaries.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to the drawing, a collection of representative documents are assembled at 10 and corresponding manually generated summaries are assembled at 11. These comprise a training set. They are encoded for computer processing and stored in computer memory. They may be preprocessed to add syntactic and semantic tags.

The documents and summaries are processed in the translation model generator at 12 to build a translation model 13 which is a file containing the probabilities that a word found in a summary will be found in the document. The translation model generator constructs a statistical model describing the relationship between the text units or the annotated text units in documents and the text units or annotated text units used in the summaries of documents. The translation model is used to identify items in a source document 17 that can be used in summaries. These items may include words, parts of speech ascribed to words, semantic tags applied to words, phrases with syntactic tags, phrases with semantic tags, syntactic or semantic relationships established between words or phrases in the document, structural information obtained from the document, such as positions of words or phrases, mark-up information obtained from the document such as the existence of bold face or italics, or of headings or section numbers and so forth.

The summaries are processed by the language model generator 14 to produce a summary language model 15. The language model is a file containing the probabilities of each word or phrase found in the training set summaries following another word or phrase. The language model generator builds a statistical model describing the likely

order of appearance of text units or annotated text units in summaries. The headlines or summaries may be preprocessed to identify text items that can be used in determining the typical structure of summaries. These text items may include words, parts of speech ascribed to words, semantic tags applied to words, phrases, phrases with syntactic tags, syntactic or semantic relations established between words or phrases, structure information, such as positions of words or phrases in the summary, and so forth.

The translation model 13 and summary language model 15 along with a document 17 to be summarized and summarization control parameters 18 are supplied to the summary search engine 16 to select a sequence of items (characters or lexemes) that jointly optimize the information content extracted from the source document to be summarized. These are supplied to the summary generation engine 19 which generates the summary.

The following Table is an example document for explaining the practice of this invention:

Table 1

"The U.N. Security Council on Monday was to address a dispute between U.N. chief weapons inspector Richard Butler and Iraq over which disarmament documents Baghdad must hand over. Speaking in an interview with CNN on Sunday evening, Butler said that despite the latest dispute with Iraq, it was too soon to make a judgment that the Iraqis had broken last week's agreement to unconditionally resume cooperation with weapons inspector -- an agreement which narrowly averted air strikes by the United States and Britain."

Some possible headline/summaries for the document produced above are:

"Security Council to address Iraqi document dispute."

"Iraqi Weapons Inspections Dispute."

These summaries illustrate some of the reasoning required for summarization. The system must decide (1) what information to present in the summary, (2) how much detail to include in the summary or how long the summary can be, and (3) how best to phrase the information so that it seems coherent. The two summaries above illustrate some of the issues of length, content and emphasis.

The statistical models are produced by comparison of a variety of documents and summaries for those documents similar to those set forth above to learn for a variety of parameter settings, mechanisms for both (1) content selection for the most likely summaries of a particular length and (2) generating coherent English (or any other language) text to express the content. The learning for both content selection and summary generation may take place at a variety of conceptual levels ranging from characters, words, word sequences or n-grams, phrases, text spans and their associated syntactic and semantic tags. In this case, prior to the comparison, the texts in the training sets must be tagged.

Set forth in the following table is the text of Table 1 after being tagged with syntactic parts of speech using the LDC standard, e.g., DT: definite article, NNP: proper noun, JJ: adjective.

Table 2

The_DT U.N._NNP Security_NNP Council_NNP on_IN Monday_NNP was_VBD to_TO address_VB a_NN dispute_NN between_IN U.N._NNP chief_JJ weapons_NNS inspector_NN Richard_NNP Butler_NNP and_CC Iraq_NNP over_IN which_WDT disarmament_NN documents_NNS Baghdad_NNP must_NN hand_NN over._CD _NN _NN _NN Speaking_VBG in_IN an_DT interview_NN with_IN CNN_NNP on_IN Sunday_NNP evening,_NNP Butler_NNP said_VBD that_IN despite_IN the_DT latest_JJS dispute_NN with_IN Iraq,_NNP it_PRP was_VBD too_RB soon_RB to_VBP make_VB a_DT judgment_NN that_IN the_DT Iraqis_NNPS had_VBD broken_VBN last_JJ week's_NN agreement_NN to_TO unconditionally_RB resume_VB cooperation_NN with_NN weapons_NNS inspectors:_NNS an_DT agreement_NN which_WDT narrowly_RB averted_VBP airstrikes_NNS by_IN the_DT United_NNP States_NNPS and_CC Britain._NNP.

Set forth in the following table is the text of Table 1 after being tagged with semantic tags using the TIPSTER/MUC standards; NE: named entity, TE: temporal entity, LOC: location.

Table 3

The [U.N. Security Council]-NE on [Monday]-TE was to address a dispute between [U.N.]-NE chief weapons inspector [Richard Butler]-NE and [Iraq]-NE over which disarmament documents [Baghdad]-NE must hand over.

Speaking in an interview with [CNN]-NE on [Sunday]-TE evening, [Butler]-NE said that despite the latest dispute with [Iraq]-NE, it was too soon to make a judgment that the [Iraqis]-NE had broken last week's agreement to unconditionally resume cooperation with weapons inspectors -- an agreement which narrowly averted airstrikes by the [United States]-NE and [Britain]-NE.

The training set is used to model the relationship between the appearance of some features (text spans, labels, or other syntactic and semantic features of the document) in the document, and the appearance of features in the summary. This can be, in the simplest case, a mapping between the appearance of a word in the document and the likelihood of the same or another word appearing in the summary.

The applicants used a training set of over twenty-five thousand documents that had associated headlines or summaries. These documents were analyzed to ascertain the conditional probability of a word in a document given that the word appears in the headline. In the following table, the probabilities for words appearing in the text of Table 1 are set forth.

Table 4

Word	Conditional Probability
Iraqi	0.4500
Dispute	0.9977
Weapons	1.000
Inspection	0.3223
Butler	0.6641

The system making use of the translation model extracts words or phrases from the source text based upon the probability these or other words will appear in summaries.

The probability that certain subsets of words individually likely to appear in summaries will appear in combination can be calculated using Bayes theorem. Thus, the probability that the phrase "weapons inspection dispute", or any ordering thereof may be expressed simply:

Pr("weapons"| "weapons" in document)
*Pr("inspection"| "inspection" in document)
*Pr("dispute"| "dispute" in document).

Equivalently, this probability may be expressed:

Log(Pr("weapons"| "weapons" in document))
+ Log(Pr("inspection"| "inspection" in document))
+ Log(Pr("dispute"| "dispute" in document)).

More involved models can express the relationship among arbitrary subsets, including subsequences, of the words in the document and subsets of candidate words that may appear in the summary. The more involved models can express relationships among linguistic characterizations of subsets of terms in the document and summaries such as parts-of-speech tags, or parse trees.

The more involved models may express relationships among these sets of terms and meta-information related to the document or the summary, such as length, derived statistics over terms (such as proportion

of verbs or nouns in the document, average sentence length, etc.), typographical information, such as typeface, formatting information, such as centering, paragraph breaks and so forth, and meta-information, such as provenance (author, publisher, date of publication, Dewey or other classification) recipient, reader, news group, media through which presented (web, book, magazine, TV chiron or caption).

One of the advantages in learning a content selection model is that the system can learn relationships between summary terms that are not in the document and terms that are in the document, and apply those relationships to new documents thereby introducing new terms in the summary.

Once a content selection model has been trained on the training set, conditional probabilities for the features that have been seen in the summaries can be computed. The summary structure generator makes use of these conditional probabilities to compute the most likely summary candidates for particular parameters, such as length of summary. Since the probability of a word appearing in a summary can be considered to be independent of the structure of the summary, the overall probability of a particular candidate summary can be computed by multiplying the probabilities of the content in the summary with the probability of that content expressed using a particular summary structure (e.g., length and/or word order).

Since there is no limitation on the types of relationships that can be expressed in the content selection model, variations on this invention can use appropriate training sets to produce a cross-lingual or even cross-media summary. For example, a table expressing the conditional probability that an English word should appear in a summary of a Japanese document could be used to simultaneously translate and summarize Japanese documents.

An inventory of spoken word forms, together with a concatenative synthesis algorithm and a table of conditional probabilities that speech segments would be used in a spoken summary of a particular document, could be used to generate spoken summaries. Similarly, corresponding video or other media could be chosen to represent the content of documents.

Example

For use in generating summaries, the probability of finding particular words in a summary is learned from the training set. For certain words appearing in the text set forth in Table 1, the learned probabilities are listed in the following table:

Table 5

Word	Log probability of word in Reuters headlines
Iraqi	-3.0852
Dispute	-1.0651
Weapons	-2.7098
Inspection	-2.8417
Butler	-1.0038

Also, for generating summaries, the probability of finding pairs of words in sequence in the training set summaries is learned. For certain words appearing in the text set forth in Table 1, the learned probabilities are listed in the following table:

Table 6

Word pair (word 1, word 2)	Log probability of word 2 given word 1
Iraqi weapons	-0.7622
Weapons inspection	-0.6543
Inspection dispute	-1.4331

To calculate the desirability of a headline containing the sequence "Iraqi weapons inspection...", the system multiplies the likelihood of seeing the word "Iraqi" in a headline (see Table 5) by it being followed by "weapons" and that being followed by "inspection" (see Table 6). This may be expressed as follows:

Log(P("Iraqi"))+Log(P("weapons"| "Iraqi"))+Log(P("inspection"| "weapons")),

which, using the values in the tables, yields a log probability of -2.8496. Alternative sequences using the same words, such as "Iraqi dispute weapons", have probabilities that can be calculated similarly. In this case, the sequence "Iraqi dispute weapons" has not appeared in the training data, and is estimated using a back-off weight. A back-off weight is a very small but non-zero weight or assigned probability for words not appearing in the training set.

These calculations can be extended to take into account the likelihood of semantic and syntactic tags both at the word or phrase level, or can be carried out with respect to textual spans from characters on up. The calculations can also be generalized to use estimates of the desirability of sequences of more than two text spans (for example, tri-gram (three-word sequence) probabilities may be used).

Other measures of the desirability of word sequences can be used. For example, the output of a neural network trained to evaluate the desirability of a sequence containing certain words and tags could be substituted for the log probabilities used in the preceding explanation. Moreover, other combination functions for these measures could be used rather than multiplication of probabilities or addition of log probabilities.

In general, the summary generator comprises any function for combining any form of estimate of the desirability of the whole summary under consideration such

that this overall estimate can be used to make a comparison between a plurality of possible summaries.

Even though the search engine and summary generator have been presented as two separate processes, there is no reason for these to be separate.

In the case of the phrase discussed above, the overall weighting used in ranking can, as one possibility, be obtained as a weighted combination of the content and structure model log probabilities.

```
Alpha*(Log(Pr("Iraqi"| "Iraqi" in doc)) + Log(Pr("weapons"|
"weapons" in doc))+
Log(Pr("inspection"| "inspection" in doc)))+
Beta*(Log(Pr("Iraqi"| start_of_sentence))+Log(Pr((weapons"|
"Iraqi"))+Log(Pr("inspection"| "weapons"))).
```

Using a combination of content selection models, language models of user needs and preferences, and summary parameters, a plurality of possible summaries, together with estimates of their desirability, is generated. These summaries are ranked in order of estimated desirability, and the most highly ranked summary or summaries are produced as the output of the system.

Depending on the nature of the language, translation and other models, heuristic means may be employed to permit the generation and ranking of only a subset of the possible summary candidates in order to render the summarization process computationally tractable. In the first implementation of the system, Viterbi beam search was used to greatly limit the number of candidates produced. The beam search makes assumptions regarding the best possible word in at the front position of a summary and in consideration of the next position will not undo the assumption concerning the first position. Other search techniques, such as A* or IDA*, SMA*, may be employed to comply with particular algorithmic or resource limitations.

An example of the results of commanding the search to output the most highly ranked candidate for a

variety of values of the summary length control parameter is set forth in the following table.

Table 7

Number of Words	String
1	Iraq
2	United States
3	Iraq on Weapons
4	United States on Iraq
5	United States in latest week
6	United States in latest week on Iraq
7	United States on security cooperation in latest week

The following computer code appendix contains code in the Java language to implement this invention. The UltraSummarise class is the main function that makes a summarizer object, loads a story, creates a search object and uses the Vocabulary class and story to produce a summary. The ViteriSearch class defines the meat of the operation. It takes the LanguageModel class, the TranslationModel class and the story and searches for strings having the highest probability of being used in a summary for the story. The LanguageModel class reads in a file which is a model for summaries containing the probabilities of each word following another. The TranslationModel class reads in a file containing the probabilities that a word will appear in a summary given words in the story. The Story class reads in the story. The Vocabulary class reads in a file that turns words into numbers.

Those skilled in the computer programming arts could implement the invention described herein in a number of computer programming languages. It would not be necessary to use an object oriented programming language such as Java.

COMPUTER CODE APPENDIX

The following code in the Java language was written to implement the invention described above. The UltraSummarise class is the main function that makes a summarizer object, loads a story, creates a search object and uses the Vocabulary class and search object to produce a summary. The ViterbiSearch class defines the meat of the operation. It takes the LanguageModel class, the TranslationModel class and the story and searches for strings having the highest probability of being used in a summary for the story. The LanguageModel class reads in a file which is a model for summaries containing the probabilities of each word following another word in a summary. The TranslationModel class reads in a file containing the probabilities that a word will appear in a summary given words in the story. The Story class reads in the story. The Vocabulary class reads in a file that turns words into numbers.

```
import java.util.Date;
import LanguageModel;
import TranslationModel;
import Story;
import Vocabulary;
import ViterbiSearch;

final public class UltraSummarise
{
 final static int MAX_N_LEXEMES = 40000;
 final static int MAX_N_BIGRAMS = 400000;
 LanguageModel LM;
 TranslationModel TRM;
 Vocabulary   Vcb;
 boolean myboredom=true;
 String sty1,sty2;
 public UltraSummarise (String [] args) throws Exception
 {
   if (args.length >3) {       myboredom=true;     }
   Vcb=new Vocabulary(args[0],MAX_N_LEXEMES); //name,maxnlexemes
   LM=new LanguageModel(args[0], MAX_N_LEXEMES,MAX_N_BIGRAMS); // name,
maxnlexemes, maxnbigrams
   TRM=new TranslationModel(args[0], MAX_N_LEXEMES); // name, maxnlexemes
   sty1=args[1]; sty2=args[2];
 }
 public void Run()  throws Exception
 {
  Story       Sty;
  ViterbiSearch Search;
```

```
    Sty=new Story(sty1, Vcb,MAX_N_LEXEMES); // storyname , maxnleximes
    Search= new ViterbiSearch(myboredom,Vcb,LM,TRM);
    Search.produceStringSummary(Sty,15);
  }
  public static void main (String [] args) throws Exception
    {
     System.out.println(new Date());
     if (args.length < 2) {
        System.err.println("Usage java UltraSummarise corpusname story-file <bored>");
        System.exit(1);
     }
     UltraSummarise Ult=new UltraSummarise(args);
     Ult.Run();
     System.out.println(new Date());
     System.exit (0);

    }
}
```

```
import java.util.Hashtable;
import java.io.*;
import Vocabulary;

final public class LanguageModel {
  int MAX_N_LEXEMES;
  int MAX_LEX_BIGRAMS;
 // this is ugly, but it's not straightforward to avoid,
 // since the bigram file doesn't start with a count. Later, perhaps force it to.

 final static boolean verbose_debug=false;
 int lastlexicalbigram;
 int lastlexicalunigram;
 float [] lexicalunigramprobs;
 float [] lexicalunigrambackoffs;
 float [] lexicalbigramprobs;
 final float NOT_A_LOG_PROB = 5.0F; // n:log(n)==5 is not a probability

 String corpusname;
```

```
Hashtable bigram_hashtable;
int bigram_hashtable_last_element = 0; //used by bigram_index below

static char [] mycharacters = {
 'A','B','C','D','E','F','G','H','I','J',
 'K','L','M','N','O','P','Q','R','S','T',
 'U','V','W','X','Y','Z','a','b','c','d',
 'e','f','g','h','i','j','k','l','m','n',
 'o','p','q','r','s','t','u','v','w','x',
 'y','z'
};
static int range = mycharacters.length;
static StringBuffer keyspace = new StringBuffer("");

public LanguageModel (String setcorpusname, int MAX_N_LEXEMES,
                      int MAX_LEX_BIGRAMS) throws Exception
 {
  corpusname    = new String (setcorpusname);
  lexicalunigramprobs = new float [MAX_N_LEXEMES];
  lexicalunigrambackoffs = new float [MAX_N_LEXEMES];
  lexicalbigramprobs    = new float [MAX_LEX_BIGRAMS];
  bigram_hashtable = new Hashtable(MAX_LEX_BIGRAMS);
  System.out.println("Reading LM "+corpusname);
  readLM();
 }

public String getCorpusName()
 {
  return new String(corpusname);
 }

// convert two bigram index elements into strings and store them into a hash table,
// look them up later
int bigram_index (int word1, int word2, boolean create_p) throws Exception {
 String mytempstring = null;
 int index;
 keyspace.setLength(0);

 index = word1;
 while (index>0) {
  keyspace.append(mycharacters[index % range]);
  index /= range;
 }
```

```
  keyspace.append(' ');
  index = word2;
  while (index>0) {
   keyspace.append(mycharacters[index % range]);
   index /= range;
  }
  if (keyspace.length() >= 1024)
   throw new Exception("something wrong with indices to bigram_index");

  mytempstring=keyspace.toString();
  //     System.out.print(mytempstring+ ",");

  if (! (bigram_hashtable.containsKey(mytempstring))){
   if (create_p) {
       bigram_hashtable.put(mytempstring,
                            new Integer(bigram_hashtable_last_element++));
       if (verbose_debug) System.out.println("Put hash entry for ["+mytempstring+
                                             "] ="+word1+","+word2);
   }
   else {
       if (verbose_debug) System.out.println("no hash entry for ["+mytempstring+
                                             "] ="+word1+","+word2);
       return -1;
   }

  }
  return ((Integer)(bigram_hashtable.get(mytempstring))).intValue();
}

void readLM() throws Exception
 {
  // read bigrams
  try {
      StreamTokenizer BigramFile =
       new StreamTokenizer(new BufferedReader
                           (new InputStreamReader
                            (new FileInputStream(corpusname+".biprobs"))));
      // see http://charts.unicode.org/Unicode.charts/glyphless/U0000.html
      BigramFile.wordChars(0x0021,0x007e); // basically all the characters
      // that could concievably be in a word in English

      int last_bi= -1;
```

```
    while (BigramFile.nextToken() != BigramFile.TT_EOF){
     if (BigramFile.ttype != BigramFile.TT_NUMBER){
      System.out.println(" Non number ["+BigramFile.sval+"] in "
                         +corpusname+".biprobs");
      System.exit(1);
     }
     int w1=(int)BigramFile.nval;
     if (BigramFile.nextToken() == BigramFile.TT_EOF){
      System.out.println(" Number "+w1+" without second number in "
                         +corpusname+".biprobs");
      System.exit(1);
     }
     if (BigramFile.ttype != BigramFile.TT_NUMBER){
      System.out.println(" Non number where second number expected ["
                         +BigramFile.sval+"] in "+corpusname+".biprobs");
      System.exit(1);
     }
     int w2=(int)BigramFile.nval;
     if (BigramFile.nextToken() == BigramFile.TT_EOF){
      System.out.println(" Numbers "+w1+","+w2+" without probability in "
                         +corpusname+".biprobs");
      System.exit(1);
     }
     if (BigramFile.ttype != BigramFile.TT_NUMBER){
      System.out.println(" Non word in "+corpusname+".biprobs");
      System.exit(1);
     }
     float prob = (float)BigramFile.nval;
     if (verbose_debug)System.out.println((w1)+","+w2+" => "+prob);
     int bi = bigram_index(w1,w2,true);
     if (bi < last_bi){
      System.err.println("Got duplicate "+w1+","+w2+"both mappeed to "+bi);
     }
     last_bi=bi;
     lexicalbigramprobs[bi]=prob;
     lastlexicalbigram++;
    }
}
catch (java.io.FileNotFoundException e)
    {
     System.out.println(" Couldn't open "+corpusname+".biprobs");
     System.exit(1);
    }
```

```
catch (java.io.IOException e)
    {
     System.out.println(" Problem reading "+corpusname+".biprobs");
     System.exit(1);
    }
System.out.println(lastlexicalbigram+" bigrams read");

// read unigrams
try {
    lastlexicalunigram=0;
    StreamTokenizer UnigramFile =
     new StreamTokenizer(new BufferedReader
                         (new InputStreamReader
                          (new FileInputStream(corpusname+".uniprobs"))));
    // see http://charts.unicode.org/Unicode.charts/glyphless/U0000.html
    UnigramFile.wordChars(0x0021,0x007e); // basically all the characters
    // that could concievably be in a word in English

    while (UnigramFile.nextToken() != UnigramFile.TT_EOF){
     if (UnigramFile.ttype != UnigramFile.TT_NUMBER){
      System.out.println(" Non number ["+UnigramFile.sval+"] in "
                         +corpusname+".uniprobs");
      System.exit(1);
     }
     int w1=(int)UnigramFile.nval;
     if (UnigramFile.nextToken() == UnigramFile.TT_EOF){
      System.out.println(" Numbers "+w1+" without probability in "
                         +corpusname+".uniprobs");
      System.exit(1);
     }
     if (UnigramFile.ttype != UnigramFile.TT_NUMBER){
      System.out.println(" Non word in "+corpusname+".uniprobs");
      System.exit(1);
     }
     float prob = (float)UnigramFile.nval;
     if (verbose_debug) System.out.println((w1)+" => "+prob);

     lexicalunigramprobs[w1]=prob;
     lastlexicalunigram++;
    }
}
catch (java.io.FileNotFoundException e)
    {
```

```
      System.out.println(" Couldn't open "+corpusname+".uniprobs");
      System.exit(1);
    }
catch (java.io.IOException e)
    {
      System.out.println(" Problem reading "+corpusname+".uniprobs");
      System.exit(1);
    }
System.out.println(lastlexicalunigram+" unigrams read");

// read unigrambackoffs
try {
    int lastuniback=0;

    StreamTokenizer UnibackFile =
     new StreamTokenizer(new BufferedReader
                         (new InputStreamReader
                          (new FileInputStream(corpusname+".unibackoff"))));
    // see http://charts.unicode.org/Unicode.charts/glyphless/U0000.html
    UnibackFile.wordChars(0x0021,0x007e); // basically all the characters
    // that could concievably be in a word in English

    while (UnibackFile.nextToken() != UnibackFile.TT_EOF){
     if (UnibackFile.ttype != UnibackFile.TT_NUMBER){
      System.out.println(" Non number ["+UnibackFile.sval+"] in "
                         +corpusname+".unibackoff");
      System.exit(1);
     }
     int w1=(int)UnibackFile.nval;
     if (UnibackFile.nextToken() == UnibackFile.TT_EOF){
      System.out.println(" Number "+w1+" without probability in "
                         +corpusname+".unibackoff");
      System.exit(1);
     }
     if (UnibackFile.ttype != UnibackFile.TT_NUMBER){
      System.out.println(" Non word in "+corpusname+".unibackoff");
      System.exit(1);
     }
     float prob = (float)UnibackFile.nval;
     if (verbose_debug) System.out.println((w1)+" => "+prob);

     lexicalunigrambackoffs[w1]=prob;
     lastuniback++;
```

```
        }
        System.out.println(lastuniback+" unigrams backoffs read");
    }
    catch (java.io.FileNotFoundException e)
        {
         System.out.println(" Couldn't open "+corpusname+".unibackoff");
         System.exit(1);
        }
    catch (java.io.IOException e)
        {
         System.out.println(" Problem reading "+corpusname+".unibackoff");
         System.exit(1);
        }

  }
 float bigram_probability(int word1, int word2) throws Exception
 {
  int bi_index = bigram_index (word1, word2, false);

  if (bi_index == -1)
   return NOT_A_LOG_PROB;
  else return lexicalbigramprobs[bi_index];
 }

 float backoff_weight(int w) {return lexicalunigrambackoffs[w];}

 float unigram_probability(int w) {return lexicalunigrambackoffs[w];}

 public float probability(int w1, int w2) throws Exception
  {
   //p(wd2|wd1)= if(bigram exists) p_2(wd1,wd2)
   //        else          bo_wt_1(wd1)*p_1(wd2)
   float bigram_prob=bigram_probability(w1,w2);

   if (!(NOT_A_LOG_PROB == bigram_prob)){
       //System.out.println(w1+" "+w2+" bigram " +(bigram_prob));
       return bigram_prob;
   } else {
       //System.out.println(w1+" "+w2+" backoff "
+(backoff_weight(w1)+unigram_probability(w2)));
       return
        // -100000.0F;
        (backoff_weight(w1)+unigram_probability(w2))*3; // make backoff rather undesirable
```

```
    }
   }

}
```

```
import java.io.*;


final public class TranslationModel {
 float [] lexicaltranslationprobs;
 String corpusname;
 static boolean verbose_debug = false;

 public TranslationModel( String ThisCorpusName, int MaxNLexemes) {
  lexicaltranslationprobs      = new float [MaxNLexemes];
  corpusname=ThisCorpusName;
  System.out.println("Reading Translation Model "+corpusname+".numtrprobs");
  readTrModel();
 }

 public float probability(int w1)
  {
   return lexicaltranslationprobs[w1]; //log 1
  }

 void readTrModel() {
  try {
   int i=0;
   StreamTokenizer TransFile =
       new StreamTokenizer(new BufferedReader
                           (new InputStreamReader
                            (new FileInputStream(corpusname+".numtrprobs"))));
   // see http://charts.unicode.org/Unicode.charts/glyphless/U0000.html
   TransFile.wordChars(0x0021,0x007e); // basically all the characters
   // that could concievably be in a word in English

   while (TransFile.nextToken() != TransFile.TT_EOF){
       if (TransFile.ttype != TransFile.TT_NUMBER){
        System.out.println(" Non number ["+TransFile.sval+"] in
"+corpusname+".numtrprobs");
```

```
            System.exit(1);
          }
          int index=(int)TransFile.nval;
          if (TransFile.nextToken() == TransFile.TT_EOF){
            System.out.println(" Number "+index+" without string in "
                               +corpusname+".numtrprobs");
            System.exit(1);
          }
          if (TransFile.ttype != TransFile.TT_NUMBER){
            System.out.println(" Non float in "+corpusname+".numtrprobs");
            System.exit(1);
          }
          lexicaltranslationprobs[index]=(float)TransFile.nval;
          if (verbose_debug) System.out.println((index)+ " => "+( lexicaltranslationprobs[index]));
          i++;
      }

      System.out.println((i)+" translation probs read");
    }
    catch (java.io.FileNotFoundException e)
      {
          System.out.println(" Couldn't open "+corpusname+".numtrprobs");
          System.exit(1);
      }
    catch (java.io.IOException e)
      {
          System.out.println(" Problem reading "+corpusname+".numtrprobs");
          System.exit(1);
      }
  }

}
```

```
import java.util.Date;
import java.io.*;
import Vocabulary;

public class Story  {
  final static boolean verbose_debug=false;
```

```
static String storyvocabname;
private int n_unique_lexemes;
private int [] unique_lexemes;
private int [] lexeme_used;

public Story (String storyname, Vocabulary Vocab, int MAX_N_LEXEMES){
 storyvocabname=storyname;
 unique_lexemes = new int [MAX_N_LEXEMES];
 lexeme_used = new int [MAX_N_LEXEMES];
// use plain initVocab() if you have numeric stories
 initVocabFromTextFile(Vocab);
}

int termCount(){
 return n_unique_lexemes;
}
int term(int i){
 return unique_lexemes[i-1];
}


void initVocabFromTextFile(Vocabulary Vocab) {
 // Read the current story vocab
 try {
  n_unique_lexemes=0;
  StreamTokenizer VCBFile =
      new StreamTokenizer(new BufferedReader
                          (new InputStreamReader
                           (new FileInputStream(storyvocabname))));

  while (VCBFile.nextToken() != VCBFile.TT_EOF){
      String tok;
      if (VCBFile.ttype == VCBFile.TT_NUMBER){
       tok=(String.valueOf(VCBFile.nval)).toLowerCase();
      }else if (VCBFile.ttype == VCBFile.TT_WORD){
       tok=VCBFile.sval.toLowerCase().replace('.',' ').replace('"',' ').replace(',',' ').trim();
      }else {
       //       System.out.println("Story.java Found funny thing in "+storyvocabname+" type
"+VCBFile.ttype);
       tok="<unknown>";
       continue;
      }
      //       System.out.println(" "+tok+" "+Vocab.toIndex(tok));
```

```
    if (Vocab.toIndex(tok) >=0 ){
        if (0 == lexeme_used[Vocab.toIndex(tok)]){
          unique_lexemes[n_unique_lexemes++]=Vocab.toIndex(tok);
          lexeme_used[Vocab.toIndex(tok)]++;
        }
       }
       if (verbose_debug)
        System.out.println((n_unique_lexemes-1)+
                           " -> "+(unique_lexemes[n_unique_lexemes-1]));
   }
   //     n_unique_lexemes --;  // Undo extra increment
   System.out.println("Using vocabulary of "+(n_unique_lexemes)+" words");
  }
  catch (java.io.FileNotFoundException e)
   {
       System.out.println(" Couldn't open "+storyvocabname);
       System.exit(1);
   }
  catch (java.io.IOException e)
   {
       System.out.println(" Problem reading "+storyvocabname);
       System.exit(1);
   }
}

void initVocab() {
 // Read the current story vocab token indices
 try {
 n_unique_lexemes=0;
   StreamTokenizer VCBFile =
       new StreamTokenizer(new BufferedReader
                           (new InputStreamReader
                            (new FileInputStream(storyvocabname))));

   while (VCBFile.nextToken() != VCBFile.TT_EOF){
       if (VCBFile.ttype != VCBFile.TT_NUMBER){
        System.out.println(" Non number in "+storyvocabname);
        System.exit(1);
       }
       unique_lexemes[n_unique_lexemes++]=(int)VCBFile.nval;
       if (verbose_debug)
        System.out.println((n_unique_lexemes-1)+
                           " -> "+(unique_lexemes[n_unique_lexemes-1]));
```

```
    }
    //     n_unique_lexemes --;  // Undo extra increment
    System.out.println("Using vocabulary of "+(n_unique_lexemes+1)+" words");
   }
   catch (java.io.FileNotFoundException e)
    {
        System.out.println(" Couldn't open "+storyvocabname);
        System.exit(1);
    }
   catch (java.io.IOException e)
    {
        System.out.println(" Problem reading "+storyvocabname);
        System.exit(1);
    }
  }

  public static void main (String [] args) throws Exception {
   int MAX_N_LEXEMES = 100000;
   Vocabulary Vocab;
   Story CurrentStory;
   System.out.println(new Date());
   if (args.length < 1) {
    System.err.println("Usage Story corpusname textualstory");
    System.exit(1);
   }
   Vocab=new Vocabulary(args[0],MAX_N_LEXEMES); //name,maxnlexemes
   CurrentStory=new Story(args[1],Vocab, MAX_N_LEXEMES);
   System.out.println(new Date());
   System.exit (0);
  }


}

import java.util.Hashtable;
import java.io.*;
/* The vocabulary used by the search, It has a method
   It provides mehods that convert the lexical items used by the language model
   into strings for output*/

public class Vocabulary {
  final static boolean verbose_debug = false;
  String [] vocab_words;
```

```
Hashtable vocab_index;
int start_sent;
int end_sent;


public Vocabulary (String corpusname,int MAX_N_LEXEMES){
 vocab_words = new String [MAX_N_LEXEMES];
 vocab_index = new Hashtable(MAX_N_LEXEMES);
 // Read the current story vocab names
 try {
  StreamTokenizer VCBFile =
      new StreamTokenizer(new BufferedReader
                          (new InputStreamReader
                           (new FileInputStream(corpusname+".numvocab"))));
  // see http://charts.unicode.org/Unicode.charts/glyphless/U0000.html
  VCBFile.wordChars(0x0021,0x007e); // basically all the characters
  // that could concievably be in a word in English

  while (VCBFile.nextToken() != VCBFile.TT_EOF){
      if (VCBFile.ttype != VCBFile.TT_NUMBER){
       System.out.println(" Non number ["+VCBFile.sval+"] in "+corpusname+".numvocab
"+VCBFile.lineno());
       System.exit(1);
      }
      int index=(int)VCBFile.nval;
      if (VCBFile.nextToken() == VCBFile.TT_EOF){
       System.out.println(" Number "+index+" without string in "
+corpusname+".numvocab "+VCBFile.lineno());
       System.exit(1);
      }
      // read in string that has been quoted by the massage program,
      // and strip the quotes off (java kind of rules compared to c)
      vocab_words[index]=VCBFile.sval.replace('"',' ').trim().toLowerCase();
      if (vocab_index.containsKey(vocab_words[index])){
     System.out.println("Repeated Vocab term "+vocab_words[index]+" at index "+ index);
      } else {
       vocab_index.put(vocab_words[index],new Integer(index));
      }
      if (verbose_debug) System.out.println((index)+ " => "+( vocab_words[index]));
  }
 }
 catch (java.io.FileNotFoundException e)
  {
```

```
        System.out.println(" Couldn't open "+corpusname+".numvocab");
        System.exit(1);
    }
   catch (java.io.IOException e)
    {
        System.out.println(" Problem reading "+corpusname+".numvocab");
        System.exit(1);
    }
   // read sentence start and end markers
   try {
    StreamTokenizer StartStopFile =
        new StreamTokenizer(new BufferedReader
                            (new InputStreamReader
                             (new FileInputStream(corpusname+".startend"))));
    // see http://charts.unicode.org/Unicode.charts/glyphless/U0000.html
    StartStopFile.wordChars(0x0021,0x007e); // basically all the characters
    // that could concievably be in a word in English

    while (StartStopFile.nextToken() != StartStopFile.TT_EOF){
        if (StartStopFile.ttype != StartStopFile.TT_NUMBER){
         System.out.println(" Non number ["+StartStopFile.sval+"] in "
+corpusname+".startend");
         System.exit(1);
        }
        start_sent=(int)StartStopFile.nval;
        if (StartStopFile.nextToken() == StartStopFile.TT_EOF){
         System.out.println(" Number "+start_sent+" without second in "
+corpusname+".startend");
         System.exit(1);
        }
        if (StartStopFile.ttype != StartStopFile.TT_NUMBER){
         System.out.println(" Non word in "+corpusname+".startend");
         System.exit(1);
        }
        end_sent = (int)StartStopFile.nval;
    }
   }
   catch (java.io.FileNotFoundException e)
    {
        System.out.println(" Couldn't open "+corpusname+".startend");
        System.exit(1);
    }
   catch (java.io.IOException e)
```

```
    {
        System.out.println(" Problem reading "+corpusname+".startend");
        System.exit(1);
    }
   }

  public int startsenttoken (){
   return start_sent;
  }
  public int endsenttoken (){
   return end_sent;
  }

  public String toString(int i){
   return vocab_words[i];
  }

  public int toIndex(String s){
   if (vocab_index.containsKey(s)){
   return ((Integer)(vocab_index.get(s))).intValue();
   } else return(-1);
  }
}



/* Need to make this general, so that the probabilities for each
   transition come from an externally supplied method */

import java.util.Date;
import java.io.*;
import QSortAlgorithm;
import Vocabulary;

public class ViterbiSearch
{

  final static int MAX_N_SENSES = 40000;
  final static int MAX_SENT_LEN = 10;
  final public static float LOG_BOREDOM_DISCOUNT = -100.0F;
  final static float beam_width = 3.0F;
  final static int MIN_BEAM_SIZE = 20;
```

```
}

void setUpSearchStates (Story Sty){
 n_current_states=0;
 System.out.println ("Viterbi Search using story with "+(Sty.termCount())+" words");
 curr_start_sent=0;
 SetSearch_state(curr_start_sent,Vocab.startsenttoken());
 curr_end_sent=Sty.termCount()+1;
 SetSearch_state(curr_end_sent,Vocab.endsenttoken());
 int i;
 for (i=1; i<=Sty.termCount();i++){
  SetSearch_state(i,Sty.term(i));
 }
}

void SetSearch_state(int state, int value){
 search_states[state]=value;
 //   System.out.println("State:"+state+" word:"+value);
 state_to_lexeme[state]=value;
 n_current_states=java.lang.Math.max(n_current_states,state+1);
}
/**********************************
 * The following code section does the
 * actual viterbi search
 **********************************/


// change these later to dynamic size, since easy in java
int [][] backpointers
 = new int [MAX_SENT_LEN][MAX_N_SENSES];

// only actually need 2 slices, but this is clearer
float [][] scores
 = new float [MAX_SENT_LEN][MAX_N_SENSES];

int [] currentscoreindices = new int [MAX_N_SENSES];
float [] currentscores = new float [MAX_N_SENSES];



void backtrack(int pos_in_sent, int from_state)
 {
  if (pos_in_sent>=0)
```

```
        backtrack(pos_in_sent-1,backpointers[pos_in_sent][from_state]);

    System.out.print(Vocab.toString(search_states[from_state])+" ");//+"("+from_state+"
"+search_states[from_state]+") ");
   }

 // This backtracks through the current path to date,
 //  discouraging the selection of words already in the path
 //  -- multiple occurrances are multiply discouraging.
 // This is not a terribly good idea, since it can't undo decisions earlier on,
 // it may pick  a non optimal repetition disallowing path
 float discount_by_boredom_level(float bored_probability,
                                 int pos_in_sent, int from_word,

                                 int word_to_match){
  if (pos_in_sent>=0){
   if (from_word==word_to_match) {
       //       System.out.print("BORED!"+word_to_match);
       bored_probability += LOG_BOREDOM_DISCOUNT;
   }
   bored_probability=
       discount_by_boredom_level(bored_probability,
                                 pos_in_sent-1,
                                 backpointers[pos_in_sent][from_word],
                                 word_to_match);
  }
  return bored_probability;
 }


 void dump()
  {
   int word_in_sent;
   int this_vocab;
   for (word_in_sent=0;word_in_sent<MAX_SENT_LEN; word_in_sent++){
       System.out.println (word_in_sent);
       for (this_vocab = 0; this_vocab < n_current_states; this_vocab++){
        System.out.println(this_vocab+"<-"+backpointers[word_in_sent][this_vocab]+"
"+scores[word_in_sent][this_vocab]);
       }
       System.out.println("\n");
   }
  }
```

```
public void produceStringSummary (Story Sty, int length) throws Exception {
  setUpSearchStates(Sty);
  doSearch();
 }

 void doSearch() throws Exception {
   //   System.out.println("doSearch ncurrentstates is "+n_current_states+"\n curr_start_Sent=
"+Vocab.toString(curr_start_sent)+" ("+curr_start_sent+")"+"curr_end_Sent=
"+Vocab.toString(curr_end_sent)+" ("+curr_end_sent+")");
   // first state probabilities are transitions out of <s>
   for (int this_state = 0; this_state < n_current_states; this_state++){
    backpointers[0][this_state]=curr_start_sent;
    scores[0][this_state]
        =LMod.probability(search_states[curr_start_sent],
                          search_states[this_state])
        +TrMod.probability(search_states[this_state]);
    // Beam search
    currentscores[this_state]=scores[0][this_state];
    currentscoreindices[this_state]=this_state;
   }
   // Beam search
   quicksort.sort(currentscores, currentscoreindices);
   for (int word_in_sent=1;word_in_sent<MAX_SENT_LEN; word_in_sent++){
    int best_state=curr_start_sent;
    float best_score=-100000.0F;

    // Beam search - don't consider states with log probs (beam_width) times smaller
    // than best score
    int current_beam = 0;
    for (int from_state_i = 0; from_state_i < n_current_states; from_state_i++){
        //        System.out.println("From state i:"+from_state_i+" n_current_states:"+
                   n_current_states+" Current
Scores["+((n_current_states-1)-from_state_i)+"]:"+(currentscores[(n_current_states-1)-from_stat
e_i])+" Current Scores
indices["+((n_current_states-1)-from_state_i)+"]:"+(currentscoreindices[(n_current_states-1)-fro
m_state_i]));
        if ((from_state_i >=MIN_BEAM_SIZE) &&
           (currentscores[(n_current_states-1)-from_state_i]
           < (currentscores[n_current_states-1] * beam_width)))
         break;
        current_beam=from_state_i;
    }
```

```
for (int this_state = 0; this_state < n_current_states; this_state++){
    float max_score= -1000000.0F;
    int current_back=curr_start_sent;
    for (int from_state_i = 0; from_state_i <= current_beam; from_state_i++){
     float test_score;
     int from_state = currentscoreindices[(n_current_states-1)-from_state_i];

     //        System.out.println("state "+this_state+" from i:"+from_state_i+" from:"+
                        from_state);

     // Never repeat a state immediately, or make a transition out of EOS
     if ((from_state == this_state)
        || (from_state == curr_end_sent))
      continue;
     if ((word_in_sent > 1)  && (from_state == curr_start_sent))
      continue;

     test_score=scores[word_in_sent-1][from_state]
      +LMod.probability(search_states[from_state],search_states[this_state]);
     if (test_score >max_score){
      current_back=from_state;
      max_score=test_score;
     }
    }
    float bored_probability=TrMod.probability(search_states[this_state]);
    if (boredom)
     bored_probability=
      discount_by_boredom_level(bored_probability,
                                word_in_sent-1,current_back,this_state);
    scores[word_in_sent][this_state]=max_score+bored_probability;


    // Save scores for sorting, so can do beam search
    currentscores[this_state]=max_score+bored_probability;
    currentscoreindices[this_state]=this_state;

    backpointers[word_in_sent][this_state]=current_back;
    // we need to check if the best state now is end of sent, and stop if so
    if (scores[word_in_sent][this_state] > best_score){
     best_score = scores[word_in_sent][this_state];
     best_state = this_state;
    }
```

```
    }
    // Beam Search
    quicksort.sort(currentscores, currentscoreindices);

    System.out.print(word_in_sent+":");
    if (best_state == curr_end_sent) System.out.print ("* ");
    backtrack(word_in_sent, curr_end_sent);
    System.out.println(" "+scores[word_in_sent][curr_end_sent]+" Beam "+(current_beam+1));

   }
   // dump();
  }
}
```

As used in the following claims, a "summary string" is a derivative representation of the source document which may, for example, comprise an abstract, key word summary, folder name, headline, file name or the like.

Having thus defined our invention in the detail and particularity required by the Patent Laws, what is desired to be protected by Letters Patent is set forth in the following claims.

WHAT IS CLAIMED IS:

1. A computer method for preparing a summary string from a source document of encoded text, the method comprising the steps of:

a) comparing a training set of encoded text documents with manually generated summary strings associated therewith to learn probabilities that a given summary word or phrase will appear in summary strings given a source word or phrase appears in an encoded text document; and

b) constructing from the source document a summary string containing summary words or phrases having the highest probabilities of appearing in a summary string based on the learned probabilities established in the previous step.

2. The computer method according to claim 1, comprising constructing a summary string containing the most probable summary word, words, phrase or phrases for a preselected number of words or phrases in the summary string.

3. The computer method according to claim 2, comprising comparing the training set of encoded text documents with manually generated summary strings to learn the probability that a summary word or phrase appearing in a summary string will follow another summary word or phrase and constructing a summary string containing the most probable word or sequence of words and/or phrases for a preselected number of words in the summary string.

4. The computer method according to claim 1, comprising comparing a corpus of encoded text documents with manually generated summary strings associated therewith to learn the probabilities that a given summary word or phrase will appear in summary strings given a source word or phrase appears in the encoded text considering the context in which the source word or phrase appears in the encoded text documents.

5. The computer method according to claim 4, wherein the contexts in which the source words or phrases are considered include titles, headings and standard paragraphs.

6. The computer method according to claim 4, wherein the contexts in which the source words or phrases are considered include fonts, bolding and italicizing.

7. The computer method according to claim 4, further comprising learning multiple probabilities that a summary word or phrase will appear in a summary string given a source word or phrase appears in the encoded text considering the various usages of the word or phrase in the encoded text.

8. The computer method according to claim 7, wherein the usages in which the source words are considered are syntactic usages.

9. The computer method according to claim 8, wherein the syntactic usages include the word or phases part of speech.

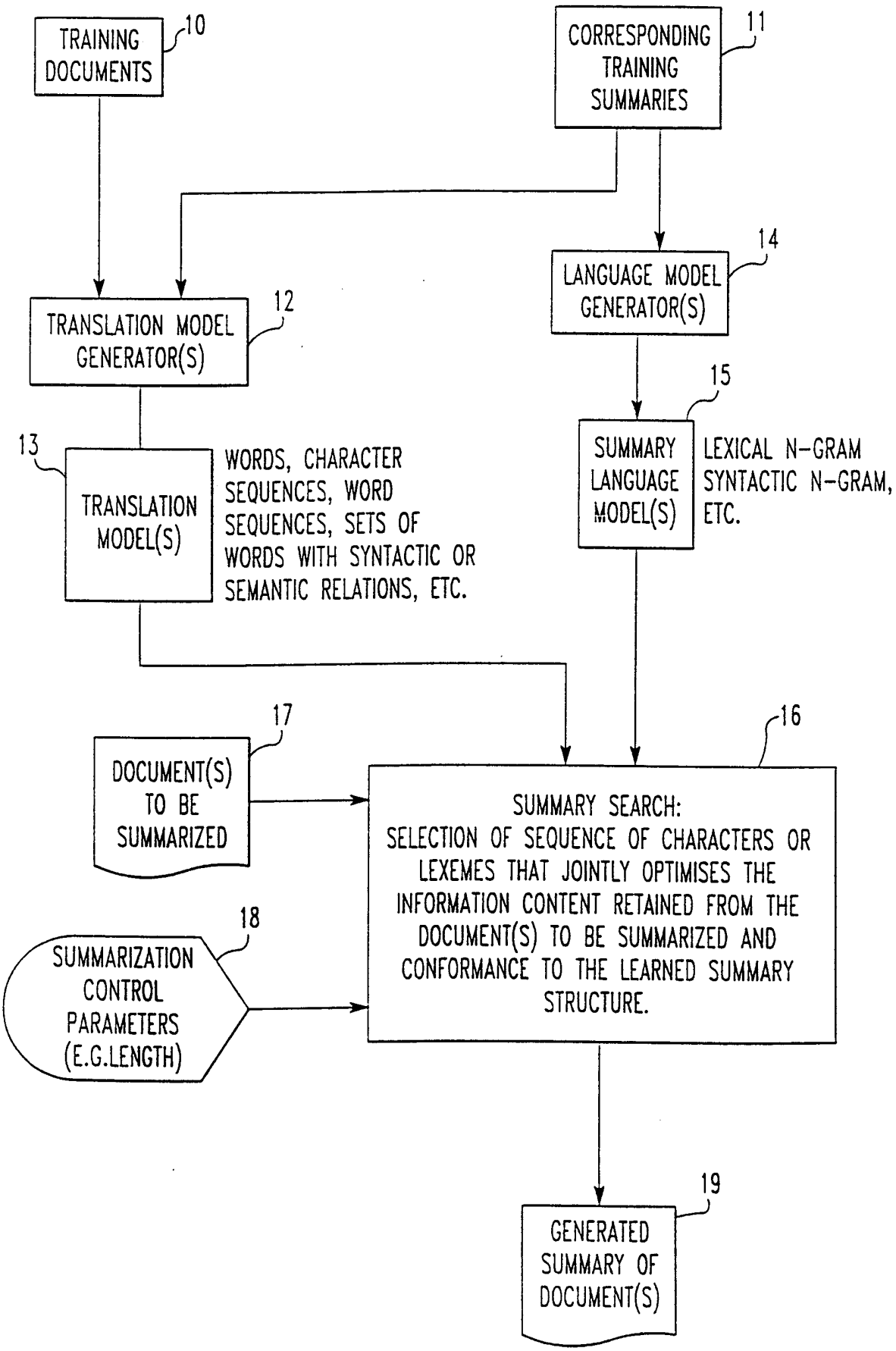
10. The computer method according to claim 7, wherein the usages in which the source words or phrases are considered are semantic usages.

11. The computer method according to claim 10, wherein the usages in which source words or phrases are considered include usage categories selected from the TIPSTER/MUC standards.

12. The computer method according to claim 10, wherein the usages in which source words or phrases are considered include usage categories selected from the group AGENT, CIRCUMSTANCE, CIRCUMSTANCE/TEMPORAL, COMMUNICATIVE_ACTION and OBJECT.

13. The computer method according to claim 4, wherein the step for comparing a corpus of encoded text documents with manually generated summary strings takes into consideration external information in the form of queries, user models, past user interaction and other biases to optimize the form of the summary strings constructed in the summary constructing step.

14. The computer method according to claim 1, comprising producing summaries in a different language from the source document by using a training set of an encoded text document in one language with manual summaries in another language.


TRAINING DOCUMENTS
10
CORRESPONDING TRAINING SUMMARIES
11
LANGUAGE MODEL GENERATOR(S)
14
TRANSLATION MODEL GENERATOR(S)
12
15
13
TRANSLATION MODEL(S)
WORDS, CHARACTER SEQUENCES, WORD SEQUENCES, SETS OF WORDS WITH SYNTACTIC OR SEMANTIC RELATIONS, ETC.
SUMMARY LANGUAGE MODEL(S)
LEXICAL N-GRAM SYNTACTIC N-GRAM, ETC.
17
16
DOCUMENT(S) TO BE SUMMARIZED
SUMMARY SEARCH: SELECTION OF SEQUENCE OF CHARACTERS OR LEXEMES THAT JOINTLY OPTIMISES THE INFORMATION CONTENT RETAINED FROM THE DOCUMENT(S) TO BE SUMMARIZED AND CONFORMANCE TO THE LEARNED SUMMARY STRUCTURE.
18
SUMMARIZATION CONTROL PARAMETERS (E.G.LENGTH)
19
GENERATED SUMMARY OF DOCUMENT(S)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/00268

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G 06F 17/27, 17/21, 15/00, 17/00

US CL : 704/1, 9, 10; 707/530, 531, 532

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 704/1, 9, 10; 707/530, 531, 532

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,778,397 A (KUPIEC ET AL) 07 JULY 1998, Abstract, col. 4, lines 7-65, col. 5, lines 17-32, col. 10, lines 5-13, col. 12, lines 3--64, col. 16, lines 1-65.	1-14
Y	US 5,638,543 A (PEDERSEN ET AL) 10 JUNE 1997, Abstract.	1-14
A	US 5,848,191 A (CHEN ET AL) 08 DECEMBER 1998, abstract.	1-14
Y	US 5,077,668 A (DOI) 31 DECEMBER 1991, Abstarct.	1-14
Y	US 5,297,027 A (MORIMOTO ET AL) 22 MARCH 1994, FIGURE 5, COL. 4, lines 41-67, col. 5, lines 1-26.	1-14

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance, the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance, the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search	Date of mailing of the international search report
05 APRIL 2000	26 APR 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	Authorized officer Patrick N. Edouard
Facsimile No. (703) 305-3230	Telephone No. (703) 308-6725

Form PCT/ISA/210 (second sheet) (July 1998)★

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/00268

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,708,825 A (SOTOMAYOR) 13 JANUARY 1998, Abstract.	1-14
Y	US 5,384,703 A (WITHGOTT ET AL) 24 JANUARY 1995, abstract.	1-14

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/00268

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

WEST/EAST
search terms: (summar$ or conden$ or abstract$) same (document or text or file) and (likelohood or probability or frequency near2 occurrence) and (704$.ccls. or 707$.ccls.)