Text Summarization

Partha Lal

June 13, 2002

Abstract

In this project, a text summarization system is developed. The system works by assigning scores to sentences in the document to be summarized, and using the highest scoring sentences in the summary. Score values are based on features extracted from the sentence. A linear combination of feature scores is used. Almost all of the mappings from feature to score and the coefficient values in the linear combination are derived from a training corpus. Some anaphor resolution is performed. The system was submitted to the Document Understanding Conference for evaluation. In addition to basic summarization, some attempt is made to address the issue of targeting the text at the user. The intended user is considered to have little background knowledge or reading ability. The system helps by simplifying the individual words used in the summary and by drawing the pre-requisite background information from the web.

Acknowledgements: I would like to thank my project supervisor Stefan Rüger for his useful help and advice, my 2^{nd} marker Ian Harries for his constructive criticism, and the GATE team at the University of Sheffield for their invaluable technical help.

Contents

I Introduction **6**
- 0.1 Motivation . . . 6
- 0.2 Potential applications . . . 6
- 0.3 Scope . . . 7
- 0.4 About This Document . . . 7

II Background **8**

1 Summarization **8**
- 1.1 Types of Summary . . . 8
- 1.2 Types of Evaluation . . . 9
- 1.3 Current state-of-the-art . . . 9
- 1.4 Multi-Document Summarization . . . 9
- 1.5 Use of Discourse Structure . . . 9
- 1.6 Use of Lexical Chains . . . 10
- 1.7 Use of Co-reference Chains . . . 10
- 1.8 Extractive Indicative Summaries . . . 10

2 Simplification **11**
- 2.1 PSET . . . 11
- 2.2 Background Knowledge . . . 12

3 Supporting Resources **12**
- 3.1 GATE . . . 12
- 3.2 WordNet . . . 13
- 3.3 MRC Psycholinguistic Database . . . 13
- 3.4 TREC Corpus . . . 15

III Design **16**

4 Sentence Scoring **18**
- 4.1 Position . . . 19
- 4.2 Length . . . 19
- 4.3 Coreference . . . 19
- 4.4 tf.idf . . . 20
- 4.5 Source . . . 20
- 4.6 Gathering document frequency values . . . 21

5 Anaphor Repair **24**

6 Simplification **25**
6.1 Lexical Simplification . . . 25
6.1.1 Implementation . . . 26
6.1.2 Problems . . . 27
6.2 Psycholinguistic Analysis . . . 28
6.2.1 `PsycholinguisticAnalyser` . . . 28
6.2.2 `SyllableCounter` . . . 28

7 Background Information **30**
7.1 Biographical Data . . . 30
7.2 Geographical Data . . . 31

8 Training **32**
8.1 Training Corpora . . . 32
8.2 Representing Training Data within GATE . . . 33
8.3 `AnnotationDiff` . . . 34
8.4 Feature scores . . . 34
8.5 Results Handling . . . 37
8.5.1 Histograms . . . 37
8.5.2 Modelling . . . 37
8.6 Coefficients . . . 38

9 Web Interface **41**

10 Abandoned Ideas **42**
10.1 RST-based summarization . . . 42
10.2 ContextScorer . . . 42

IV Evaluation **44**

11 DUC Evaluation **44**
11.1 Method of Evaluation . . . 44
11.2 Results of Evaluation . . . 46
11.2.1 Performance on each Quality Question . . . 46
11.2.2 Sentence Recall and Precision . . . 47
11.2.3 Unmarked Peer Unit Relevance . . . 49
11.2.4 Coverage . . . 49

12 Evaluating Simplification **52**

13 Strengths and Weaknesses **52**

V Conclusion **53**

14 Outcomes **53**

15 Challenges **54**

16 Future Work **54**
16.1 Document Visualisation 54
16.2 Simplification 55
16.3 Background Information 56
16.4 Sentence Representation 56

Part I
Introduction

0.1 Motivation

The intention of text summarization is to express the content of a document in a condensed form that meets the needs of the user [12]. Far more information than can realistically be digested is available on the World-Wide Web and in other electronic forms. News information, biographical information, minutes of meetings missed — it isn't possible to read everything one would want to read and so some form of information condensation is needed.

Secondly, the language of news media may be impenetrable to some people — for example, children or people learning English as a foreign language. Some method of language simplification would be useful, as well as a method of providing the background knowledge adults take for granted.

A solution that addressed both of those problems would enable a wider range of people to be aware of a greater amount of information.

0.2 Potential applications

Possible current uses of summarization [12]:

Summarizing medical data for doctors In [21] the authors intend to summarize medical information with the patients details in mind.

Multimedia News Summarization This involves summarizing data from different sources.

Producing Intelligence Reports Given a wide range of documents, an intelligence analyst may wish to read a biography of a person. A system exists which creates a dossier of information on a person from a text collection.

Text for Hand-held devices Due to the limited size of displays on WAP phones and palm-top computers, it is useful to condense text found in web pages browsed.

Convenient Text-to-Speech for Blind people The idea here is to scan in a page from a book, and then read out a summary of the page rather than the entire text.

Collating Search Engine hits Rather than read all the pages returned by a search, it would be better to read a summary of the top N hits.

Summarizing Meetings Combining summarization with automatic speech recognition produces a system which summarizes the salient points of a meeting.

0.3 Scope

The project is wide in scope — all of the limitations stated below may seem to contradict that, but they are the *only* restrictions applied.

This project looks at single document summarization — the area of multi-document summarization is not covered. Also, the summaries produced are largely extracts of the document being summarized, rather than newly generated abstracts. The parameters used are optimal for news articles, although that can be changed easily.

With regard to language simplification, only lexical changes were considered — syntactic changes were not. Background information was limited to biographical information and maps.

0.4 About This Document

The contents of the sections of this report are:

Part II Here we look at how various methods of summarization work, the current state-of-the-art and various supporting technologies that are used in the project.

Part III In this part we are discuss the design of the summarizer. Each component is described in a separate section. Finally ideas that were abandoned are mentioned.

Sections 4, 5, 6 and 7 Here we discuss in more detail the stages that the summarizer goes through.

Section 8 The majority of parameters in the system were derived from a training corpus, and it is in this section that the derivation process is described.

Part IV This part describes how the system was evaluated, largely describing the DUC[1] evaluation. A possible extrinsic evaluation of the simplification and background information modules is also described.

Part V Here we make some conclusions and consider potential future work.

This report and source code are available at

`http://www.doc.ic.ac.uk/~pl98/`

. The project can be downloaded as a CREOLE Repository (see Section 3.1) at

`http://www.doc.ic.ac.uk/~pl98/summarizer/`

.

[1] Document Understanding Conference

Part II
Background

1 Summarization

The phases that an automatic summarizer goes through can be split into the following [16].

Interpret This is where a representation of the document to be summarized is produced. Also known as analysis.

Transform This is where the representation of the document is turned into one of a summary of the document.

Generate Here, summary text is produced from the summary representation. Also known as synthesis.

1.1 Types of Summary

Summaries of text can be divided into different categories, some of them harder to automate than others. One division is based on the origin of the text in the summary:

Extractive This where the summary consists of sentences that have already appeared in the text.

Abstractive Here, some new text is generated by the summarizer.

Clearly, extractive summaries are the simpler option of the two, since they avoid the language generation problem.

Summaries can also be categorised by their purpose:

Indicative These summaries are meant to give the reader an idea as to whether it would be worthwhile reading the entire document. The topic and scope of the text should be expressed but not necessarily all of the factual content.

Informative This type of summary expresses the important factual content of the text.

Critical This sort of summary criticises the document. It expresses an opinion on — in the case of a scientific paper, say — the methods employed and the validity of the results.

Indicative summaries are the most feasible to automate, out of the three, and critical summaries probably the least. Informative summaries are a little harder than indicative ones, since fuller coverage of the information in the text is required.

1.2 Types of Evaluation

Evaluation of summaries is a critical problem in this area. Evaluations fall in to one of two types:

Intrinsic This is where the system is tested in of itself. Typically tests are done to measure summary quality and informativeness.

Extrinsic This is when the summarization system is measured relative to a real world task. Examples include reading comprehension tests — how much of the original content did the user gain from just reading the summary?.

1.3 Current state-of-the-art

NIST (National Institute of Standards and Technology) run the Document Understanding Conference (DUC) at which the latest summarizing techniques are compared. The proceedings of DUC-2001 [9] contains examples of the latest techniques used.

1.4 Multi-Document Summarization

This is where one summary needs to be composed from many documents. A side-problem is that of novelty-detection — given a ordered set of documents (search engine hits ordered by relevance, news articles ordered by date) summarize the first document, then summarize only the previously unseen information in all subsequent documents.

There are a number of issues in multi document summarization (MDS) that are different to single document summarization. First of all a higher compression is needed. A 10% summary may be sufficient for one document but with, say, ten documents, concatenating the individual 10% summaries will give a text as long as an average document in the collection — too long. A 1% summary is more like what is needed. An straightforward extractive summary is out of the question.

Also, information may be repeated in different documents and a decision will be needed on which of the intersecting sentences should be included. Rhetorical relations between sentences need to be established — does one sentence contradict another or does it say everything the other says and more? Many more relations exist and these can be used to judge which are the sentence that carry the most new information.

1.5 Use of Discourse Structure

Discourse Structure is to do with the relations between sentences. Sentences are not flung together in a random order, they bear specific relations to each other — one sentence might provide *background*, *evidence* or *elaboration* for another. Those are just some of the relations that exist in Rhetorical Structure Theory (RST) [27]. Automatically parsing RST relations is difficult, but it seems true that a summarizer using RST would perform well.

1.6 Use of Lexical Chains

Lexical chains are sequences of words that are related to each other in a defined way. There are mutually symmetric relations between words like hypernymy & hyponymy and meronymy & holonymy — the former pair is to do with one word being a type of the other, the latter is one being a part of the other. "Tree" is a hypernym of "oak", a hyponym of "plant", a holonym of "leaf" and a meronym of "woods".

Using these relations, chains of otherwise unrelated words can be built; the longest chains could indicate the central theme of the text and therefore which sentences to extract.

1.7 Use of Co-reference Chains

Another approach (not exhibited at DUC) is the use of co-reference [25].

In text, references are made to referents, for example in

> Dave went to visit his grandmother. She was surprised to see him.

the referents are Dave and his grandmother. The references to Dave were "Dave" and "him" whilst the references to his grandmother were "his grandmother" and "she". The reference "him" and "she" were *anaphoric* and "Dave" and "him" (as well as "his grandmother" and "she") can be said to co-refer. Anaphor resolution is the task of finding the referent ("Dave") of a reference ("he").

In this system anaphora are resolved and sequences of references to the same referent are extracted. Long chains are considered central to the document.

1.8 Extractive Indicative Summaries

There are various methods for selecting which sentences to use in summary. Early work was done on this by Luhn [11] and later Edmundson [10]. Luhn proposed finding the content words in a document; sentences could be selected if they contained those words. Content words would be those that occur between given frequencies — stop-words (conjunctions, determiners etc.) would be excluded and the frequency limits would be determined from a corpus.

Edmundson suggested scoring sentences with a linear combinations of four terms:

Cue words Sentences containing words such as "significant" or "conclusion" may be useful to include.

Key words These are like Luhn's content words. A variation on this is $tf.idf$ scores. $tf.idf$ values for a term t are the product of the term frequency of t (number of occurrences in this document) and some function of the inverse of the document frequency (number of documents containing this term). A high $tf.idf$ value for a term indicates that it could be a keyword, since it occurs often in the document but less often in general.

Location Good sentences may be more often found at the beginning of paragraphs. Heuristics for finding fertile positions can be found in [26, Section 2.1] and [2].

Title If a sentence uses words that occurred in the title of the document, the score is increased.

The optimal coefficients for the terms indicated that key words were the least useful term and that the combination of cue words, location and title was the best.

Problems with extractive summaries include those of *coherence*. There may be anaphoric reference (e.g. pronouns, "The event was...", "This action must be...") in the sentence(s) selected, the referent of which is in an unselected sentence — this is known as a *dangling anaphor*.

There may be gaps in the flow of the text, where adjacent sentences have nothing to do with each other.

Sentence might also have been extracted from within a discourse structure e.g. a list of points — "There are five reasons why we should..." followed by only two of the reasons would be an example.

Extractive summaries have the advantage of being more robust than abstractive ones, as well as cheaper to process. They do not require complicated natural language understanding and generation.

2 Simplification

2.1 PSET

The Practical Simplification of English Text project (PSET) [23] looked at the task of automatically simplifying English newspaper text. The solution they aimed for had the following structure:

Lexical Tagger Part of Speech tagging

Morphological Analyser Records the number, tense, case etc. of words

Parser Parses text

Anaphor Resolver Resolves anaphora

Syntactic Simplifier Turns passive into active voice, for example

Lexical Simplifier Swaps words with simpler words

Morphological Generator Applies the previously recorded inflection to this new word

2.2 Background Knowledge

The difference between simplification work done under PSET and the work undertaken here is related to one of the differences between aphasic readers and children — children do not have the same general knowledge as an aphasic adult. Even if a news article is expressed in simple syntax, it still will not be understood unless the reader recognises the entities referred to within.

One of the online sources of information that was be used here is the biography database at `www.s9.com`. Brief biographies of people can be extracted from the website by querying on name, date or keyword.

Another source of information is `www.wikipedia.com`, or even better `simple.wikipedia.com`. Both are free online encyclopedias and the latter is specifically written in simplified English. Unfortunately, the simplified Wikipedia contains only a handful of articles. The main version is now approaching the 20000-entry mark. Neither of those sites were used.

Maps could, perhaps, be extracted from "Map Machine" [22]. The site provides a tailor-made map of the place name the user provides. The interface may be difficult to handle automatically though, and the site is a commercial one, so using their content without some form of credit may be an issue. A cheeky alternative solution is to use the first image returned by the query "`map` *place-name* `site:.cnn.com`" — this has proven fairly successful in informal tests. The latter option was chosen in the end.

3 Supporting Resources

3.1 GATE

GATE[2] is made up of three elements[3]:

- An architecture describing how language processing systems are made up of components.
- A framework (or class library, or SDK), written in Java.
- A graphical development environment built on the framework.

GATE is an architecture in which text processing tools can be created and used. The aspects of GATE that make it so appealing for this project are the modular nature of it and the structure that it provides.

The representation of a document in GATE holds the contents of the document, sets of annotations on the content and a feature-value map. *ProcessingResource*s are executed upon *Document*s, or more specifically, on *Corpus*s of them.

CREOLE[4] is the feature of GATE that enables components to be loaded over the Internet. A URL can be provided which locates a "repository" — a

[2] General Architecture for Text Engineering
[3] from `http://gate.ac.uk`
[4] Collection of REusable Objects for Language Engineering

jar file containing the components and an XML file describing them (names, parameters...).

ANNIE[5] is a system built into GATE which performs named entity extraction. It consists of a series of *ProcessingResources* (PRs) run together using a `SerialLanguageAnalyserController` — the PRs used are:

`gate.creole.tokeniser.DefaultTokeniser` Tokenizes the input into words.

`gate.creole.splitter.SentenceSplitter` Finds sentence boundaries.

`gate.creole.POSTagger` Labels each token with its Part-of-Speech.

`gate.creole.gazetteer.DefaultGazetteer`

`gate.creole.ANNIETransducer`

`gate.creole.orthomatcher.OrthoMatcher` Recognised coreference between named entities that are similar but not identical. ("Tony Blair" and "Blair" for example)

3.2 WordNet

WordNet [28] is an online thesaurus. There also exists a Java interface to WordNet called `javaWN` [15] (amongst many other Java versions, amongst versions in many other languages...). A WordNet interface was being integrated into GATE towards the end of project.

The structure of the thesaurus is as follows. Individual words can have more than one sense (the word "bank" can refer to a river bank or a financial bank) and each sense can have more than one synonym. The set of synonyms for a sense is called a *synset*. An example is in Figure 1. Words are listed by their base form (that is, verbs are not conjugated, nouns are singular...) and their part of speech (one of noun, verb, adjective or adverb in this case).

Senses are listed in decreasing order of frequency, meaning the most commonly used sense (in a given corpus) occurs first. Disambiguating the sense of a word from the context within which it is used, is a difficult problem and one that is not practical for this project. Choosing the most frequent sense gives fair performance for no effort.

Note that pairs of words are included in the thesaurus — "financial institution" for example.

3.3 MRC Psycholinguistic Database

When it comes to language simplification, it is not necessarily true, that a word with fewer syllables will be more widely understood. Searching in WordNet for synonyms of *unintelligible* gave *opaque* which, although shorter, probably would not be understood by a reader who does not understand *unintelligible*. Ideally

[5] A Nearly New Information Extraction system

The noun "bank" has 10 senses in WordNet.

1. depository financial institution, bank, banking concern, banking company — (a financial institution that accepts deposits and channels the money into lending activities; "he cashed a check at the bank"; "that bank holds the mortgage on my home")
2. bank — (sloping land (especially the slope beside a body of water); "they pulled the canoe up on the bank"; "he sat on the bank of the river and watched the currents")
3. bank — (a supply or stock held in reserve especially for future use (especially in emergencies))
4. bank, bank building — (a building in which commercial banking is transacted; "the bank is on the corner of Nassau and Witherspoon")
5. bank — (an arrangement of similar objects in a row or in tiers; "he operated a bank of switches")
6. savings bank, coin bank, money box, bank — (a container (usually with a slot in the top) for keeping money at home; "the coin bank was empty")
7. bank — (a long ridge or pile; "a huge bank of earth")
8. bank — (the funds held by a gambling house or the dealer in some gambling games; "he tried to break the bank at Monte Carlo")
9. bank, cant, camber — (a slope in the turn of a road or track; the outside is higher than the inside in order to reduce the effects of centrifugal force)
10. bank — (a flight maneuver; aircraft tips laterally about its longitudinal axis (especially in turning))

Figure 1: The synset of the noun "bank"

some idea of the vocabulary of an n-year-old child is needed, and for that a psycholinguistic database is needed.

One such database is the MRC Psycholinguistic Database [20] — it contains information such as age of acquisition, frequency of use (from Kucera & Francis, Thorndike & Lorge and Brown corpora), syllable count and phonetic transcription. In all, 26 properties are recorded although not every property is recorded for every word.

Using this data would give a far clearer idea of which word would serve as a better replacement for others. The various features of a word could be combined in some manner to give an "ease of understanding" score to a word. In the end, syllable count and Kucera-Francis frequency were used.

3.4 TREC Corpus

The TREC[6] corpus is a large corpus of documents taken from

- Associated Press
- Financial Times
- Foreign Broadcast Information Service
- Los Angeles Times
- San Jose Mercury News
- Wall Street Journal

The data is in SGML, with headlines, lead paragraphs and captions, amongst other things, being marked up. The corpus consists of hundreds of files, each containing many `<DOC>`... `</DOC>` elements — each document resided within its own `DOC` element.

[6] Text REtrieval Conference, see `http://trec.nist.gov`

Part III
Design

The structure that the summarizer takes is described in Figure 2. The "ANNIE" in the diagram includes the Coreferencer module, as well an extra *ProcessingResource*, which takes the original markup area a sentence is in and adds that as a feature of the sentence. That means that sentences that occurred within, say, a `<head>`... `</head>` markup will have a features `source` with value `HEADLINE`.

The *ProcessingResource*s (PRs) that deal with simplification and background info are made optional. Just add a `simplify` feature with value `true` to *Document*s that are required to be simplified and they will be run through the appropriate PRs, otherwise not. Similarly a feature can be added to enable background information addition.

The PRs are in a deliberate order. ANNIE produced information that is needed to assign scores to sentences (`SentenceScorer`). Only after scores have been assigned can sentences be selected `SentenceSelector`. Then it is known which anaphora will need repair — `AnaphorRepairer`.

`PsycholinguisticAnalyser` could be run at any stage after ANNIE, but this is the latest point at which it can be placed. `SyllableCounter` is run before `PsycholinguisticAnalyser` so that any errors it makes which have their correction under `PsycholinguisticAnalyser` will be corrected.

Both of those modules need to have been run before `LexicalSimplifier`. The background section could swap places with the simplification section, and the biographical and geographical modules could also swap places.

Each of the PRs is described in the following sections.

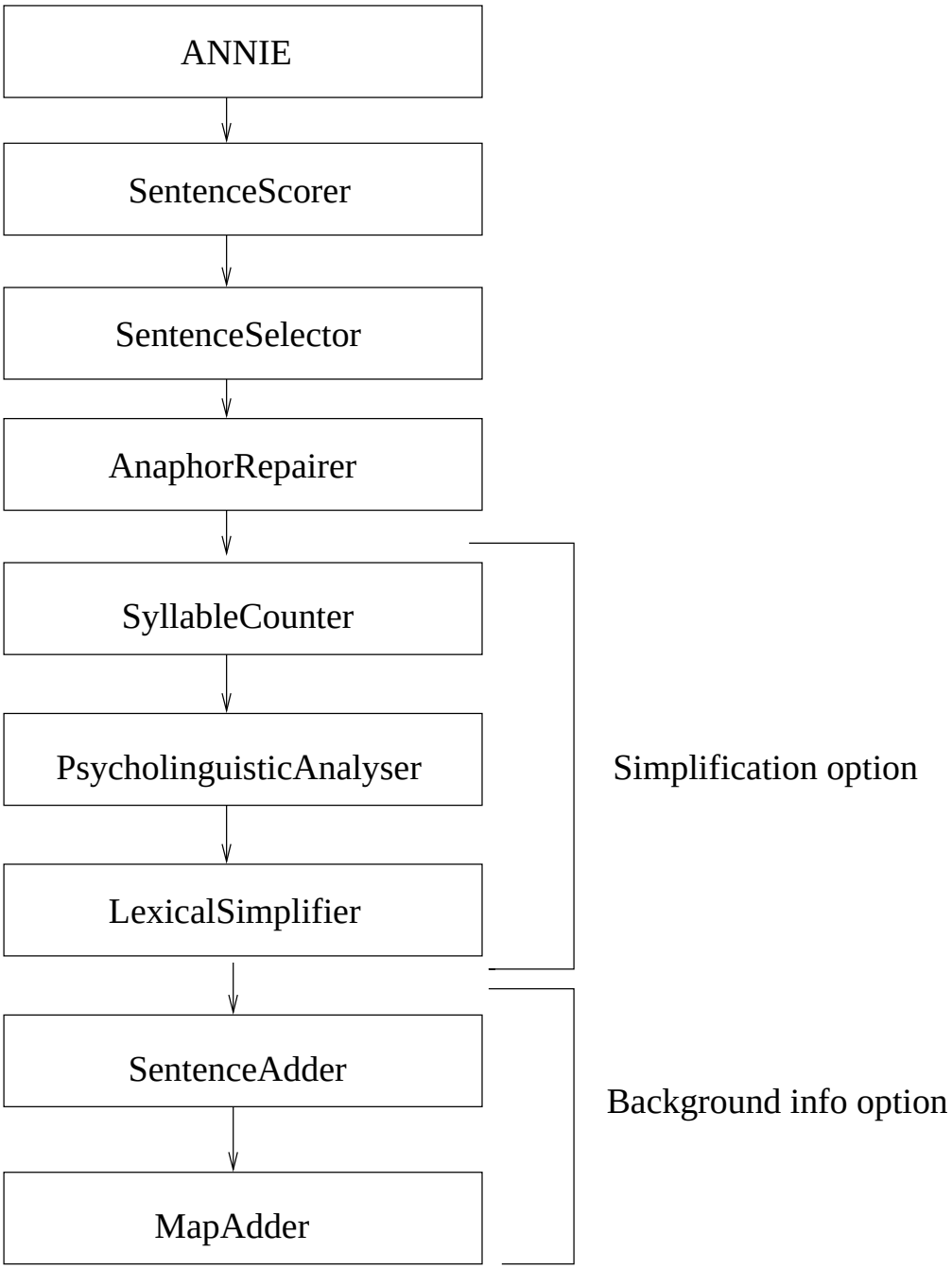


Figure 2: The overall structure of summarizer

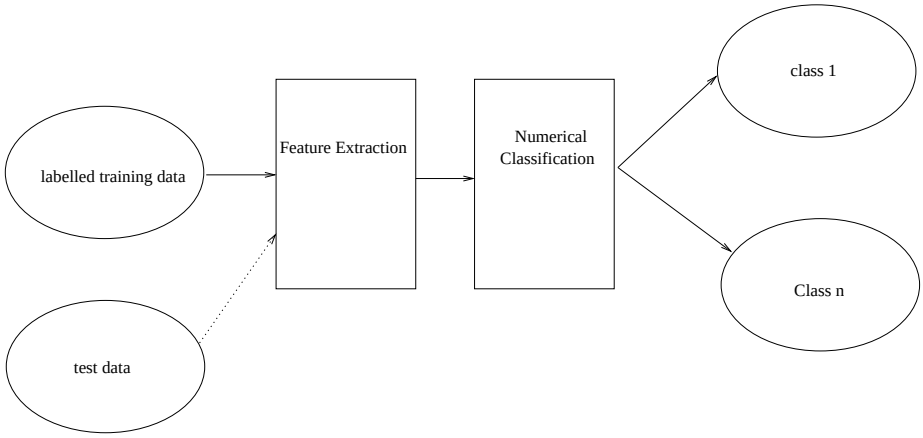


Figure 3: The general structure of pattern classification

4 Sentence Scoring

The task of scoring and selecting sentences can be seen as a classification problem. Classification can take the structure described in figure 3. In this instance, the training data consists of sentences labeled as either being in the summary of their document or not. Both the training and the test data came from DUC. The classes on the right are similarly just "in summary" and "not in summary". The training process is described in Part 8.

Feature extraction is the task of representing the data, i.e. sentences, numerically so that they can be easily classified. The representation chosen here resulted in a "score" for each sentence. The score for each sentence is simply a convex linear combination of the scores corresponding to different features, as in equation 1.

$$score(s) = \alpha S_{para}(s) + \beta S_{sent}(s) + \gamma S_{len}(s) + \delta S_{coref}(s) + \zeta S_{headline}(s) + \eta S_{source}(s) + \theta S_{tf.idf}(s) \quad (1)$$

where

$$S_{feature}(s) = P(extract | feature\ of\ s) \quad (2)$$

The numerical classification stage here is very simple — it was just the selection of the N highest scoring sentences for the "in summary" class.

For features involving named entities — S_{coref}, $S_{headline}$ and $S_{tf.idf}$ — places and organisations were considered but names of people were not. That decision is based on work cited in [12, page 62] which argued that newspaper articles are not so often about people.

The features measured were meant to be indicative of the sentence belonging to the "in summary" class. They were: paragraph number and sentence number (i.e. position), length of sentence, level of coreference, coreference with headline, source (within markup) and tf.idf. They are described in further detail in the following subsections.

4.1 Position

This covers both the position of the enclosing paragraph in the document and the position of the sentence within the paragraph. Position has been shown to be significant in finding sentences which are about the topic of the document [2] and also in summarization [26]. Based on [2], paragraphs were included into consideration.

First of all, sentence positions within a document were treated as a pair (P, S) — as was the case in [2] — the first number stating which paragraph the sentence is from and the second stating where *within that paragraph* the sentence is. These pairs were implemented as `summarizer.trainer.PositionPair` which was simply a class with two fields (for P and S) and access methods for each.

When it came to modelling the data (see section 8.5.2), things proved more difficult. The feature in that form is two-dimensional, and using any of the one-dimensional method described would not have worked. Listing the pairs in order of occurrence in the document, along with the score for that pair would have resulted in a sawtooth pattern — peaks at the top of each paragraph. In future, two-dimensional linear interpolation could be attempted.

It seemed that the two numbers would have to be separated into different features. So rather than equation 1 having $S_{position}$ it would have S_{para} and S_{sent}. This option was undesirable. To avoid having to do that, combining p and s in a number of ratios was tried — $S_{pos} = a \times S_{para} + b \times S_{sent}$ with $a = 1$ and $b = 2$ for example. The fixed ratio position feature was put through `summarizer.trainer.Trainer` (described in section 8.6) and the precision of the results were compared.

Eventually, that method was decided to be unsatisfactory, and the features were separated

4.2 Length

The length of a sentence has featured successfully in previous summarization systems [26] and was therefore included here.

4.3 Coreference

Coreference is when more than one word or phrase refers to the same thing. Coreference has been used before in summarization, for example [25]. GATE made these features easy to measure. Once ANNIE & the Coreferencer have been executed on a *Document*, each named entity will have a list of *Annotation*s it matches with. These will include not only matches derived from the OrthoMatcher (matching "General Colin Powell" with "General Powell") but those from the Coreferencer (anaphoric references). So every reference to a named entity will have a list of all the other instances of that NE in the document.

Two coreference-based features were considered.

Overall Level In a document it seems reasonable to expect that the "subject" of the document will be referred to a lot. So sentences containing words that refer to entities that are highly referred to, may be central to the document.

Coreference with Headline The entities mentioned in the headline of a document are usually important in the document, so sentences that refer to those same entities seem likely to be relevant. A bag of NEs that are mentioned in the headline needs to be generated. It would be far too inefficient to recalculate this bag every time a sentence is measured. So the bag is filled once, when processing of the document begins, and kept as a "feature" of the *Document* for future reference.

4.4 tf.idf

First, a recap of what tf.idf values mean. Given a term t (a term being either a word or a phrase) and a document d, the term frequency, $tf(t,d)$ is the number of occurrences of t in d — a similar definition can be applied to sentences. Given a corpus of documents, the document frequency $df(t)$ is the number of documents in the corpus containing at least one occurrence of t. Finally, a less used function is collection frequency $cf(t)$, equal to the total number of occurrences of t in the entire corpus. *idf* just signifies the *inverse* of the document frequency. The generic $tf.idf$ function is $f(tf(t,d)) \times g(idf(t))$ where f and g are one of a number of possible functions. The specific $tf.idf$ agreed upon for this project was

$$tf.idf(t,d) = tf(t,d) \times \log(N/df(t)) \tag{3}$$

where N is the number of documents in the corpus.

Clearly, term frequencies can be calculated by inspection of the document to be summarized. Document frequencies, however, need to be pre-computed from a large reference corpus. The process of deriving document frequencies from a corpus is described in Section 4.6.

The idea behind this feature is to take the mean of the tf.idf values of named entities occurring in the sentence. A high tf.idf value for an NE would indicate that the named entity occurred often (or at least once) in the sentence, but was rare in general over a reference corpus. Separate reference corpora were used, depending on the publication the document came from. The rationale behind this was that terms that occurred often in, say, *Financial Times* would not be so common in *LA Times*, because of the differing subject matter of the two newspapers. The mean of the tf.idf scores was taken rather than the sum, to keep this feature independent of sentence length.

4.5 Source

This idea is perhaps limited to use on data from the TREC corpus. The text within the headline may well be more useful than that in the main body of the text. In order to make this feature easier to calculate, the badly named PR

`summarizer.util.DUCtoDoc` is run before `SentenceScorer`. `DUCtoDoc` adds a source feature to every "Sentence" *Annotation* (and therefore can only be run after a sentence splitter has been run). The source feature indicate which region of the document the sentence is in.

A minor hurdle that was overcome was the variation in the different markups used by different publications — headlines could be marked as `HL` or `HEAD` or.... The solution was to simply conflate equivalent tags into one type. This problem occurred again when the scope was extended to cover HTML. With HTML, the idea of lead paragraph was abandoned, since they were not typically marked out, and the headline was considered to be anything with in the `HEAD` region.

4.6 Gathering document frequency values

To work out a tf.idf value for a sentence, document frequency values were needed. The possibility of searching for a pre-calculated table of *df*s was considered. But, even if such a resource were found, its use would be limited — the named entities being considered would be exactly those NEs that ANNIE finds.

The TREC Corpus was used as a reference for document frequencies, after all, the DUC test data would come from there too — the format of the corpus is described in Section 3.4. A Perl script that generates document frequency and collection frequency data for all non-stop words[7]was provided by Stefan Rüger. Unfortunately, this was not suitable for these purposes — the intention was to use the tf.idf of named entities and not non-stop words. Whilst the two sets of terms may intersect, the whole set of multiple word NEs would be omitted.

The task was realised to be this: using GATE, iterate through a "sufficient" amount of the TREC corpus, gathering df data. This took far longer than expected, due to various implementation problems related to GATE.

The aspect of the design that remained throughout the process is described in Figure 4.

Implementations of *CorpusFrequencies* are meant to be put in `SerialAnalyserController`s (for example, after ANNIE) and will maintain the `data` *Map* as indicated. The `execute` method does the following work:

1. Collect a `Bag` of `Object`s from the *Document*, using the `getBag()` method (implemented in subclasses).
2. Increment `numInsts` by the size of the `Bag` — `numInsts` is the number of `Object`s seen in the *Corpus*.
3. For each *type* of `Object` in the `Bag`
   (a) if *type* does not appear as a key in `data` then put a mapping from it to an empty `bag`.
   (b) Add the name of the current *Document* to the `Bag` that is mapped to by *type* in `data`.

[7] Stop words are function words — for example "the", "at" and "of".

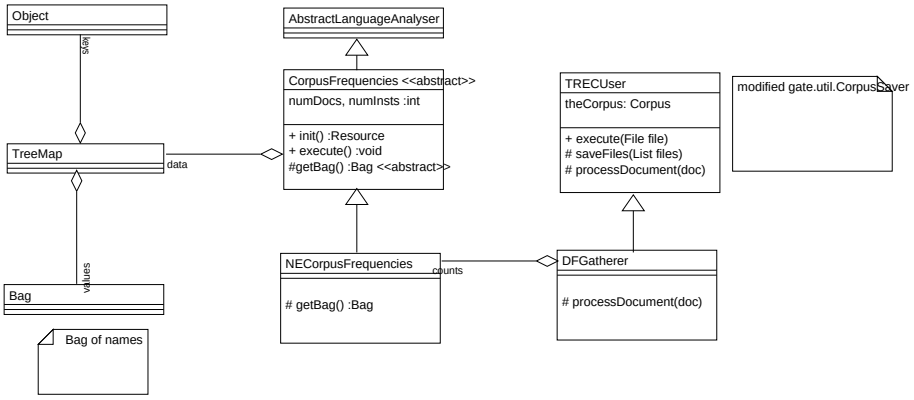


Figure 4: UML Diagram of an attempt to gather document frequencies from TREC corpus

4. Increment `numDocs` by one.

Corpus statistics can be collected using the following expressions:

$tf(t,d)$ `data.get(t).count(d)`

$df(t)$ `data.get(t).numTypes()`

$cf(t)$ `data.get(t).size()`

`NECorpusFrequencies` is simply an implementation of *CorpusFrequencies* such that the `getBag()` method returns a `Bag` of named entity strings. The XML file it outputs to is shown in Figure 5.

`TRECUser` runs ANNIE over a directory of TREC files and saves the documents therein in a corpus. The subclass of `TRECUser`, `DFGatherer`, adds the `NECorpusFrequencies` PR to series of PRs used. That was the design settled upon. Another design attempted involved implementing subclasses of `gate.corpora.DocumentImpl` and `gate.corpora.CorpusImpl` with `summarizer.util.TRECDocumentImpl` and `summarizer.util.TRECCorpusImpl`.

In the end, it was realised that the crux of the problem was keeping documents in memory for only as long as is needed. Populating a corpus from an entire TREC file is impractical. Also, once a *Document* has been deleted with `Factory.deleteResource(Resource res)`, that does not necessarily mean that it is ready to be garbage-collected by the Java Runtime Environment. There may still exist references to it — that is why it is easier to localise activity to one or two classes. An implementation based around `DFGatherer` was finally settled on. The memory leakage problem did not affect performance too badly, although usage still increased steadily throughout execution.

```
<?xml version="1.0" encoding="UTF-8"?>
<FREQUENCY_DATA NUM_DOCS="20655" NUM_INSTS="366248">
 <TERM DF="3" CF="15">'N</TERM>
 <TERM DF="2" CF="2">'N Pak Stores Inc</TERM>
 <TERM DF="1" CF="1">'N Save Corp.</TERM>
 <TERM DF="1" CF="1">'N Shop Inc</TERM>
 <TERM DF="1" CF="1">0009 New Way Found</TERM>
 <TERM DF="1" CF="1">0009 Soviet Road</TERM>

⋮


 <TERM DF="1" CF="1">western U.S.S.R.</TERM>
 <TERM DF="1" CF="1">western Virginia</TERM>
 <TERM DF="2" CF="2">western Washington</TERM>
</FREQUENCY_DATA>
```

Figure 5: An example of the XML files produced by `NECorpusFrequencies`

Another problem that was noted was with DOM[8]. Initially, the XML document outputted was held in memory as a DOM — considering the files grew to be megabytes large, holding the DOM in memory became impractical. To remedy this SAX[9] was used instead. SAX is an event-based model for XML — whilst DOM hold the tree structure of the document in memory, SAX effectively following a traversal of the tree. A `org.xml.sax.ContentHandler` outputs the correct markup and text specified by "start element", "end element", ... events. SAX does not hold the document in memory and so file size is only limited by disk space.

[8] Document Object Model
[9] Simple API for XML

5 Anaphor Repair

One of the problems with extractive summaries is *dangling anaphora*. This is best explained with an example:

> William Boynton, of the Lunar and Planetary Institute of the University of Arizona, US, is lead author on one of a trio of pioneering Mars papers published in this week's issue of Science magazine. He regards the sensational findings of ice below the surface of Mars as third time lucky.

Suppose only the second sentence of the two was included in a summary. In that case, the anaphor in it (the "he" referring to "William Boynton") would be left *dangling* — the referent it refers to would be unknown.

The solution being applied here is to make use of ANNIE's Coreferencer module. It will attempt to indicate the antecedent (i.e. the text it refers to) of every anaphor. So the `AnaphorRepairer` PR looks for anaphora whose antecedent does not occur within the extract. Once such an example is found, the text of the antecedent is brought up and inserted within square brackets directly after the anaphor.

Because of an unimplemented `edit` feature in GATE, the antecedent text cannot actually be inserted. So, the sentence is given a feature with value equal to its modified contents. The same feature is used when performing lexical simplification (Section 6)

The module was tested on an example document, although now that trained parameters have changed, the test cannot easily be repeated.

This application of the `Coreferencer` was discussed with its author [8]. The possibility of a confidence value was suggested (to indicate the confidence there was that the correct antecedent had been found) but the resolution algorithm was too simple to generate such a value. Initially it was thought that the precision of the module was too low to be of practical use, but referring to [17, page 58] indicated that acceptable use could be made if limited to he/she/her/him/herself/himself anaphora. Subsequently, performance was improved on I/my/me/myself anaphora and so they were repaired too.

The scope of this solution is limited. It only applies to pronominal anaphora, and even then only on types described above. It would be good to find a better anaphora resolution algorithm. To that end, I contacted the author of a comparative study in anaphor resolvers [3] and found that Kennedy & Boguraev's [4] system performed the best. Unfortunately, after contacting Chris Kennedy, I was told that there was no public implementation of their algorithm, since the work was done at Apple. Implementing the algorithm with just the description in the paper to work from was not practical in the time available, but may be worthwhile in future.

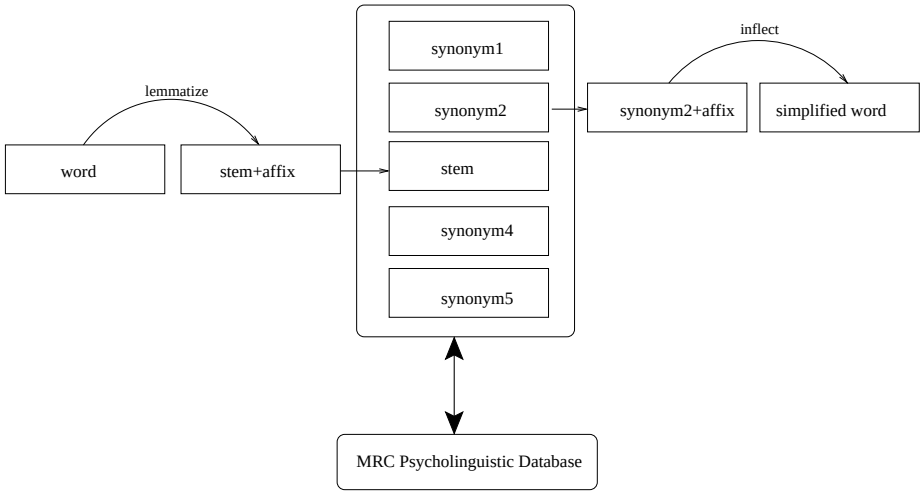


Figure 6: The process of word replacement

6 Simplification

6.1 Lexical Simplification

This stage boils down to replacing difficult words with easy ones. The task, however, is far harder than it sounds; a diagram is given in figure 6 and a walk-through example follows.

1. The open class words (nouns, verbs, adjectives and adverbs) in the document are sorted by decreasing order of "difficulty". Then, for each word:
2. Analyse the morphology of the word. For example, `publicised=publicise+ed`.
3. The Part-of-Speech of the word will be known and so a query can be made of WordNet. Here the query term is (`publicise`,`verb`).
4. The "difficulty" of each of the synonyms returned is evaluated, and the simplest one chosen. In our example, `air` would be picked.
5. The chosen word is given the inflection of the word it is to replace. So `air+ed` is calculated to be `aired`.
6. If applicable, the preceding "a"/"an" is corrected. So "*A* publicised event" becomes "*An* aired event".

The iteration continues until either all words have been "simplified" or the minimum required Flesch Readability Score is attained (see Section 6.2.1). The Flesch Score is recalculated whenever a change is made to the *Document* and stored as a *Document* feature.

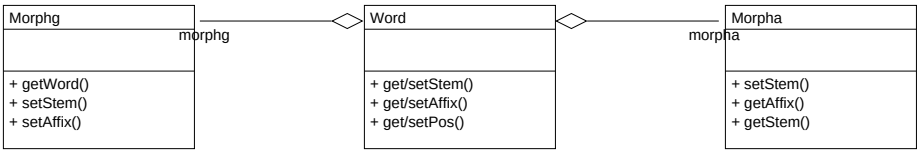


Figure 7: UML Diagram of morphology handling

6.1.1 Implementation

The implementation of that idea brought together three technologies.

MRC Psycholinguistic Database (see Section 3.3) This was used to quantify word difficulty.

`morphg`, `morpha` and `ana` The first two are morphology tools produced for PSET [23], they are used in the second and fifth stages above. `ana` is used in the final stage.

WordNet WordNet is accessed using the purely Java interface jwnl. It is used to find synonyms.

One issue that came of using different things in one place was the various data representations used. Each product had its own Part of Speech labels, for example — this meant writing tedious conversion methods from one set of PoS tags to another.

Difficulty The psycholinguistic data is accessed using `summarizer.util.PsycholinguisticAnalyser` (Section 6.2.1). The Flesch score of documents is also calculate there, although updates to the Flesch score caused by word replacement are done in `summarizer.simplifier.LexicalSimplifier`.

Comparison of words in terms of difficulty is done using a *java.util.Comparator* called `summarizer.util.psycholinguistic.PsychoComparator`. It operates like this:

1. If the number of syllables in either word is unknown, the words are equally difficult.
2. The word with fewer syllables is simpler.
3. If the Kucera-Francis frequency of either word is unknown, the words are equally difficult.
4. The word with the higher Kucera-Francis frequency is simpler.

Morphology The tools used were written using `lex` and so had to be wrapped in a Java class. The design settled upon is in figure 7. `morpha` analyses word morphology (`publicised`→`publicise+ed`) whilst `morphg` generates inflected words (`air+ed`→`aired`). A typical use case would be to

Construct a `Word` The constructor will invoke `Morpha`.

Change its stem with the `setStem` method Just changes the value of a field in `Word`.

Read the altered word with `getWord` Invokes methods of `Morphg`.

This code along with `Ana` described below, was tested by `summarizer.util.cogs.MorphTest`.

Indefinite Articles If the word being replaced is preceded with either "a" or "an" then the replacement may cause the article to be incorrect afterwards. Also, the problem of choosing which article is not as simple as "an" before vowels, "a" otherwise — consider "a one" and "an XML file". `Ana` wraps the `ana` Perl script and just corrects any such mistakes it finds. Conveniently, it also deals with capitalisation correctly, so if an article in the input is capitalised, it will be in the output.

The `ana` script itself had to be modified slightly, to force the output buffer it maintains to be flushed after every sentence is supplied. Also, sentences as defined by GATE can sometimes contain newline characters — giving those to `ana` confuses it and everything after the first newline gets returned upon the next query which will of course be on an entirely different sentence. To remedy that, newline characters are stripped out of the input.

WordNet The interface to WordNet chosen was jwnl. Initially javaWN was considered, but that involved use of native code via Java Native Interface (jni) which would not only ruin the portability of the code but also (from experience) be a nightmare to install. jwnl accesses WordNet files entirely in Java, and is in fact what the current GATE interface to WordNet is based on. Unfortunately that interface has only just been released, too late for use here.

6.1.2 Problems

Even when words are successfully substituted by this method — that is, the word is genuinely simpler and correctly inflected — the resulting text may not sound right. Examples of this are given in [6] — "many thanks" sounds sensible but "several thanks" sounds odd, even though "many" and "several" are synonymous.

The multiple word synonyms in WordNet are not considered, since their morphology is not handled by the morphological tools to hand. For example, providing `morphg` with `eat+ed_V` would return `ate_V` but giving `get up+ed_V` would not return `got up_V`.

Even though John is not normally given to a display of his deeper emotions, he allegedly has developed a profound affection for Mary, as compared to the more equable feelings he seems to have for Lucy, Fran and, to a lesser extent, Sue.

John has a profound affection for Mary.

John loves Mary.

Figure 8: These sentences should have Flesch scores of 32, 67 and 91

6.2 Psycholinguistic Analysis

6.2.1 `PsycholinguisticAnalyser`

MRC Data This PR adds psycholinguistic information about words gathered from the MRC Psycholinguistic Database [20]. The database comes in the form of an 11 Mb text file, sorted by alphabetical order. Loading the file into memory under Java would be impractical, and so it needs to be referred to on disk.

Again, performing a linear search through over 150 000 records would be time consuming (the search would have to be done for every word in the document) and so a binary search was implemented. Unfortunately, although the file was sorted alphabetically, words which can exist as different Parts-of-Speech did not have each PoS listed in a consistent order, and so the file had to be sorted further. The re-sorted database file is used.

Flesch Score The Flesch Readability Score of the *Document* is calculated when this PR is executed, and added to the *Document* as a feature. The formula for that score is

$$206.835 - 1.015s - 84.6w$$

where s is the mean sentence length in words and w is the mean word length in syllables.

Three test documents were used, based on information at [24] — they are shown in Figure 8. The Flesch scores calculated for them were 35.3, 66.8 and 91.0 respectively — the minor variation will be due to discrepancies in actual and calculated syllable counts.

6.2.2 `SyllableCounter`

The `PsycholinguisticAnalyser` does not have syllable counts for all words. Since the number of syllables is important for a number of other features, and it can be automatically calculated, this PR exists to provide syllable counts on words. The algorithm used was taken from [1] and simply counts contiguous vowel stretches whilst enforcing a minimum count of one and subtracting one from words ending in "e", as described in Figure 9.

```
count := 0;
for i = 0;i < word.length;i := i + 1 do
    if word[i] is a vowel then
        count := count + 1
    end if
    while i < word.length ∧ word[i] is a vowel do
        i := i + 1
    end while
end for
if word ends with e then
    count := count − 1
end if
if count ≤ 0 then
    count := 1
end if
```

word is the input and *count* the output; vowels are taken to include "y"

Figure 9: The algorithm used to count syllables

7 Background Information

As described before, there is little point in simplifying the language of a document if the people and places mentioned within it are unknown to the reader. These modules attempt to remedy that problem, by finding information on the named entities within the text and searching for information on them on the web.

7.1 Biographical Data

All of the code here is in `summarizer.selector.BiographyAdder` and relies on the results of ANNIE. The stages gone through are as follows:

List Person *Annotation*s in *Document* A `MatchesListComparator` is used to ensure each entity occurs only once — other references to the entity would have the same *matchesList* and so would be excluded.

Query Biographical Database A *Document* is constructed from a URL which includes the query term. For example, searching for "Bill Clinton" would result in constructing a *Document* with URL

`http://www.s9.com/cgi-s9/engine99.cgi?name=Bill+Clinton&`

`keyword=&date=&advanced=`

.

Process Results The page that is returned is one of three responses:

No match found In this case the person is ignored.

Website is overloaded, come back later Here, this message is passed on to the user so that he/she can try again later.

Results Any results found will be in a series of `<P>...</P>` markups. Each `P` area represents exactly one person with the name enquired after. There is opportunity here for interesting algorithms to chose which of the people is the one really sought after — the problem is not unlike word-sense disambiguation. A simple method is used here though — the longest `P` is used, based on the assumption that that person is the most "popular" person on the page and therefore also most likely to be the one in the summarized document. It is assumed (and seems true) that the results returned are in some alphabetical order — certainly not in an order that can be used for the disambiguation problem here.

Append data to *Document* The information gathered is made a "feature" of the *Document*, in the form of a *Map* from person name to biographical description text. It is then up to `SummarizerServlet` to incorporate that feature into the response to the user.

If the servlet sees a non-empty map as the *Document*'s "biographical data" feature, it forms an HTML `TABLE` containing the information.

7.2 Geographical Data

The way in which geographical data, i.e. maps, are gathered is similar to biographical data in the previous section. The website used this time was `http://www.picsearch.com` — `http://images.google.com`[10] would have been preferable but it made unavoidable use of frames, which do not agree with GATE's *Document* loader.

The search process is as follows:

- The URL is loaded

  `http://www.picsearch.com/search.cgi?`

  `num=16&q=map+PLACENAME&rp=same&pl=en`

  . The page returned contains a `TABLE` full of thumbnails of the actual images.

- The link enclosing the first image on the page is followed — this would lead to a page with frames but since GATE is downloading the page rather than a browser, a helpful page is returned containing the URLs of both the frames.

- One of the frames will contain the original page containing the image, the other will contain information about the image — loading the page at the latter URL will allow the URL of the image to be found.

- Once that URL is finally found, it can be inserted into a *Map* from place name to image URL, which will be made a feature of the summarized *Document*.

If anything goes wrong between the thumbnail image being found and the original image being found, the thumbnail is used. The servlet treats this data similarly to biographical data.

[10] Google's image search engine

8 Training

The questions that need to be answered through use of a training corpus are:

- Given a feature, what is the corresponding score?
- What are the optimum coefficients to apply to each term?

Given a general intuition about which sentences form a good summary, those questions could be answered with arbitrary values that "seem" right. Using a training corpus, however, gives the values used some justification.

On top of that, there are some instances where a guess at the score for a function cannot be agreed upon: are long or short sentences better? — an argument could be made for either side.

Tasks that were done as part of training:

- Training corpus found
- Training corpus made into a GATE usable form
- Conditional probability of extraction given feature ($P(extract|feature)$) calculated
- Conditional probabilities modelled
  - Gaussian model
  - Linear interpolation
- Model selected using χ^2 testing
- Optimal coefficients for feature combination chosen (equation 1)

8.1 Training Corpora

A problem in text summarization is the lack of training corpora. Ideally, there would be plenty of examples of text and their corresponding (extract-based) summary.

One solution is provided in [5], where extracts are derived from a corpus of (document,abstract) pairs. The documents are taken from the Ziff-Davis corpus, in which the articles are about computer retail. I contacted Daniel Marcu, author of [5], and was sent a copy of the corpus with extract-worthy sentences annotated.

However, another set of training data was made available soon after that. John Conroy[11] provided fellow DUC participants with extract-based summaries of 145 documents from the DUC training corpus.

The training data provided by Conroy was chosen for this purpose because it was taken from that same sort of data that the summarizer would be tested on — TREC data. Nothing from the Ziff-Davis corpus would be used in DUC

[11] John M. Conroy, Center for Computing Sciences, Institute for Defense Analyses

and so using that corpus could train the summarizer with incorrect values. For example, [12, page 51] states that the positions of salient sentences varies between the Ziff-Davis corpus and Wall Street Journal articles.

An annoying downside was that, whilst Marcu's data was conveniently marked up in XML (extracts were within `<I>...</I>` elements), this data came in a strange proprietary format that really required use of the Perl scripts they provided.

8.2 Representing Training Data within GATE

The files provided as training data were:

`meo_extracts.asc` A list of filenames each with a list of sentence numbers to be used in an extract.

`parsed` A directory structure with the mentioned in `meo_extracts.asc`. Appended to each of the files was:

- for each word — start & end offset, PoS, idf count, named entity type...
- for each sentence —- start & end offset, length, document position, paragraph position...

The best way to make use of this training data was via `gate.annotation.AnnotationDiff` (see Section 8.3), and so the files provided had to be turned into a *Corpus* of GATE *Document*s with *Annotation*s indicating which sentences should be extracted. That task was performed by `summarizer.trainer.MarkExtracts`. ANNIE would be run on the *Corpus* at this stage, to save having to run it every time the *Corpus* is used.

The action of `MarkExtracts` is described below:

`init()` Opens the *DataStore* that is to be used and initialises ANNIE & Coreferencer.

`execute()` Iterates through the lines in `meo_extracts.asc`. The lines each contain a filename, and for each file:

1. Open the file described (it is a file under the `parsed` directory).
2. Open the corresponding, unedited file from the original DUC training data directory.
3. Create a *Document* from the latter file.
4. Using the information provided in the former file, add "extract" and "don't extract" *Annotation*s to the *Document*.
5. Run ANNIE on the *Document*.
6. Save it to the *Corpus*, then delete it from memory.

`finalize()` Closes the *DataStore* that was being used.

The major hurdle in this step was problems aligning the offsets given for sentences with the actual sentences. Once *Document*s are created in GATE, offsets into them ignore markup text[12]. To compensate for this, parameters could be set at initialisation time, to ensure a record of the markup is kept so that offsets that include the markup can be transformed into ones that don't. Unfortunately this feature of GATE was relatively new, and in the course of implementing `MarkExtracts`, many bugs were reported to its author. Some bugs were ironed out, but in the end some annotations were inexplicably one or two characters out.

8.3 `AnnotationDiff`

A GATE utility that was used throughout training was `AnnotationDiff`.

The purpose it served was to answer the question: given the model summary in the training data and the system-generated summary, how close is the generated summary to the model?

Summaries at this stage were represented as "extract" *Annotation*s — the reference extracts were in the default *AnnotationSet* and the summarizer generated ones were in the "summarizer" *AnnotationSet*.

`AnnotationDiff` can do a number of things, all to do with the overlap of *Annotation*s. It is given the two *AnnotationSet*s to be compared and can return the recall, precision and F-measure[13] with which the sets match. It can measure these either strictly or leniently — the former requiring an exact overlap for two *Annotation*s to be considered a match, the latter requiring only some degree of overlap.

8.4 Feature scores

The intended outcome of this phase is to know, given a feature value f of a sentence (i.e. given its word count or how many references there are to entities in the headline), the probability that it should appear in the summary — $P(extract|f)$. This part took a couple of attempts to get right.

Initially, the probability recorded was simply $P(f \cap extract)$. In that implementation, a `Bag` of feature values that occur in extract sentences was maintained (using instances of *CorpusFrequencies* (see Section 4.6)). Once the training corpus had been iterated through, the `Bag` was such that `bag.count(f)` was proportional to $P(f \cap extract)$.

Then it was realised that that was the case and that the probability required

[12] Once the file has become a *Document*, the first character in `<TEXT>Example</TEXT>` is `E`

[13]

recall The proportion of possible correct answers that the system found

precision The proportion of answers that were correct

F-measure A weighted sum of recall and precision

```
<?xml version="1.0" encoding="UTF-8"?>
<GOOD_GIVEN_FEATURES>
 <FEATURE PROB="0.0">1</FEATURE>
 <FEATURE PROB="0.0">2</FEATURE>
 <FEATURE PROB="0.08333333333333333">3</FEATURE>
 <FEATURE PROB="0.02">4</FEATURE>

⋮

 <FEATURE PROB="0.0">119</FEATURE>
 <FEATURE PROB="0.0">151</FEATURE>
 <FEATURE PROB="0.0">166</FEATURE>
 <FEATURE PROB="0.0">198</FEATURE>
 <FEATURE PROB="0.0">240</FEATURE>
 <FEATURE PROB="0.0">314</FEATURE>
</GOOD_GIVEN_FEATURES>
```

Figure 10: An example file containing conditional probabilities for features

was $P(extract|f)$. So Bayes' Theorem was leapt upon

$$P(extract|f) = \frac{P(f|extract)P(extract)}{P(f|extract)P(extract) + P(f|not\,extract)(1 - P(extract))}$$

and `Bag`s were maintained of feature values that occur in non-extract sentences as well as those in extract sentences. This required non-extract sentences to be annotated in the training corpus — they hadn't been up until then.

Finally, it was noticed that

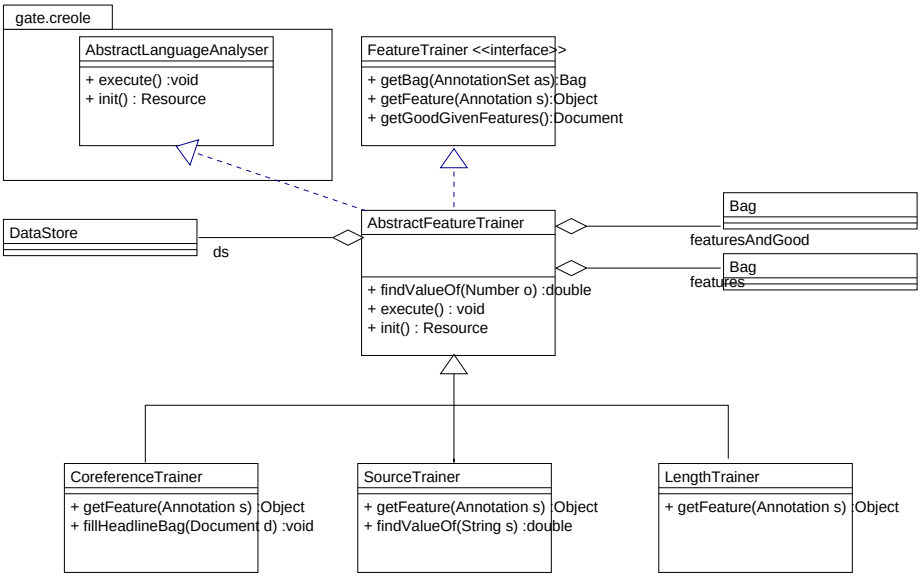
$$P(extract|f) = \frac{P(extract \cap f)}{P(f)}$$

so `Bag`s of features occurring in extracts and features occurring anywhere were maintained. `extracts.count(f)/all.count(f)` is proportional to $\frac{P(extract \cap f)}{P(f)}$ and therefore to $P(extract|f)$.

The results of the training were held in an XML file such as the one in Figure 10. The use of XML files means that the results of training can be described in the files, rather than be hard-coded into the application, and new training results could easily be used.

Unfortunately, there was a bug in the extraction of the overall coreference feature (see Section 4.3), and the XML file for that feature held nonsensical values. That rendered the feature invalid, as well as the coefficient assigned to that feature in Section 8.6. That bug has been fixed but was still in place when the DUC submission was made.

There was a fair deal of commonality between the trainer classes for each feature type. Duplication of code was avoided as far as possible — a diagram of the eventual feature training class structure is in Figure 11.



`summarizer.util.Bag` An implementation of *java.util.Collection*, behaves as a bag should — it is like a set that can contain multiple instances of the same object. It is implemented by means of a `java.util.TreeMap` from `Object` to `Integer` — the use of a `TreeMap` makes it quicker (log time versus linear time) but limits its applicability. The class was tested with `summarizer.util.BagTest`.

gate.DataStore An interface for the persistent store containing the training corpus described in Section 8.1. ANNIE + `Coreferencer` had already been run on the corpus.

Figure 11: UML diagram of feature training arrangement (not all subclasses of `AbstractFeatureTrainer` are shown)

8.5 Results Handling

The nature of some of the conditional probability data derived meant that the data had to be processed further before use.

8.5.1 Histograms

Some of the feature values derived were too sparse to be used raw. For example, tf.idf scores were real numbers and so when collected in a bag will never "pile up" — two numbers would have to be identical for the bag to count them together. A bag containing {0.1,0.11,0.09} would not indicate any clustering in its contents. To remedy this, some of the features were the histogram limits of the actual feature value returned for the sentence. Histograms were used for

- $tf.idf$
- Length
- Overall Coreference

8.5.2 Modelling

Sometimes the data would contain large gaps between values. For example, with sentence length, especially toward the higher end of the scale, there would be big differences between any non-zero probabilities. The solution was to model the data. All of this work is done in `summarizer.trainer.ResultsHandler`

Linear Interpolation Here, any zero probabilities are replaced with a value derived through linearly interpolating between the nearest non-zero values.

Gaussian Model The data could be modelled with a Gaussian distribution. Using this distribution requires that the values in the XML file could genuinely be considered to be a probability distribution — the sum of all the probabilities must equal one[14]. The mean and standard deviation were then used as inputs to public domain statistics library, to obtain the probability for the interval around that feature value.

χ^2 Comparison How could you tell which model to choose, that is, which model diverges least from the original? A modified version of χ^2 hypothesis testing was employed. The raw probabilities were treated as the observed values

[14] Actually, they must sum to $P(extract)$ but that is only a constant factor out.

$$\sum_{\forall i} P(extract|f) = P(extract)$$

O and the modelled data (by linear interpolation or normal distribution) were the expected values E. The formula usually used is

$$\sum_{\forall i} \frac{(O_i - E_i)^2}{E_i^2}$$

but some of the observations here were zero. $O_i = 0$ means that the term becomes independent of E_i, which is undesirable — smaller values of E_i should give smaller χ^2 values. The modification made was to add one to all numbers, giving

$$\sum_{\forall i} \frac{((O_i + 1) - (E_i + 1))^2}{(E_i + 1)^2}$$

Results of the comparison are given in table 1 — lower values are indicate a closer fit. These results dictated which model to use.

Feature	Gaussian	Interpolated
Length	0.014	0.035
Sent. No.	0.009	0.023
Para. No.	0.029	0.188
tf.idf	0.027	0.168
headline coref.	0.094	0.060
overall coref.	0.010	0.030

Table 1: χ^2 values for models of conditional probability distribution in various features.

Removal of Tails After gathering conditional probabilities for all features it was noticed that using them unreservedly might not be the best thing to do. For example, With features such as coreference, although the probability distribution may be roughly bell-shaped, that doesn't mean that after a certain maximum, more coreference is detrimental. It just means that no examples of higher amounts of coreference were found in the training data. As a result the probabilities for values greater that the maximum were deleted — this resulted in that maximal value being used for all greater values too.

Another modification to the data is in the position-based features. The distribution gleaned from the training corpus was simply to flat and ran counter to expectations. The file in Figure 12 was used in place of the derived data.

8.6 Coefficients

Any feature in the linear combination might not be as useful as another — some way of evaluating the relative worth of each feature is needed. This amounts to finding values for the coefficients in the linear combination of equation 1. The accuracy of this training was to the nearest 0.05 — higher accuracy would be

```
<NORMALIZED>
 <GENERATED PROB="1.0">1.0</GENERATED>
 <GENERATED PROB="0.8">2.0</GENERATED>
 <GENERATED PROB="0.6">3.0</GENERATED>
 <GENERATED PROB="0.4">4.0</GENERATED>
 <GENERATED PROB="0.2">5.0</GENERATED>
 <GENERATED PROB="0">6.0</GENERATED>
</NORMALIZED>
```

Figure 12: Replacement data for paragraph number and sentence number.

for $\alpha := \alpha_{min}; \alpha \leq \alpha_{max} \wedge \alpha \leq 1; \alpha := \alpha + \iota$ **do**
  for $\beta := \beta_{min}; \beta \leq \beta_{max} \wedge \beta \leq 1 - \alpha; \beta := \beta + \iota$ **do**
    for $\gamma := \gamma_{min}; \gamma \leq \gamma_{max} \wedge \gamma \leq 1 - \alpha - \beta; \gamma := \gamma + \iota$ **do**
      for $\delta := \delta_{min}; \delta \leq \delta_{max} \wedge \delta \leq 1 - \alpha - \beta - \gamma; \delta := \delta + \iota$ **do**
        for $\zeta := \zeta_{min}; \zeta \leq \zeta_{max} \wedge \zeta \leq 1 - \alpha - \beta - \gamma - \delta; \zeta := \zeta + \iota$ **do**
          for $\eta := \eta_{min}; \eta \leq \eta_{max} \wedge \eta \leq 1 - \alpha - \beta - \gamma - \delta - \zeta; \eta := \eta + \iota$ **do**
            $\theta := 1 - (\alpha + \beta + \gamma + \delta + \zeta + \eta)$;
            run summarizer at these parameters over corpus;
            record mean precision, recall and F-measure;
          end for
        end for
      end for
    end for
  end for
end for

Figure 13: Algorithm used to find optimum coefficient values

desirable. This training surprisingly showed that better performance occurred when the coefficients for S_{coref} and $S_{tf.idf}$ were zero — meaning that those features weren't at all useful (at least not at this level of accuracy).

The zero coefficient for S_{coref} was later pinned down to an incorrect parameter file. Correcting the value is simply a matter of running this section again. The corrected coefficient will be available very shortly.

The work was done in `summarizer.trainer.Trainer`, and was as Figure 13. The XML file produced was similar to that in Figure 14. The best coefficients were found by simply searching the file for high precision. Precision was chosen over recall because the summarizer is limited in how many sentences it can choose and so there may be an artifical ceiling on recall — so it could be that recall could have been high if only more sentences could have been included in summary. The word limit was chosen to make the reference and generated summaries similar in length, but the argument still remains for precision over recall.

First, the upper and lower limits for every coefficient was set to 0 and 1

```
 <?xml version="1.0" encoding="UTF-8"?> <TRAINING_RESULTS
CORPUS="extracts" CONTEXT_SCORED="false"> <EXPT_RESULTS
COREF="0.0" SENTNO="0.0" PARA="0.0" LENGTH="0.42"
TFIDF="0.0" SOURCE="0.28" F-MEASURE="0.28199539115032063"
PRECISION="0.3495640509725016" RECALL="0.24665178327150158" />
<EXPT_RESULTS COREF="0.04" SENTNO="0.0" PARA="0.0" LENGTH="0.42"
TFIDF="0.0" SOURCE="0.28" F-MEASURE="0.2679101620725045"
PRECISION="0.33122065727699523" RECALL="0.2355054178997841" />
<EXPT_RESULTS COREF="0.08" SENTNO="0.0" PARA="0.0" LENGTH="0.42"
TFIDF="0.0" SOURCE="0.28" F-MEASURE="0.274109596440953"
PRECISION="0.3394366197183099" RECALL="0.2400828826885165" />
⋮
</TRAINING_RESULTS>
```

Figure 14: Coefficient Optimisation XML file

respectively. The increment ι was set to 0.25. Then the same was done with ι=0.1, which was more useful but more time consuming. After examining the results of that run the upper and lower limits were narrowed to close in on a potential maximum and the search granularity ι was reduced.

Originally there were fewer features and so the algorithm had fewer nested loops. Running the algorithm shown $\iota = 0.1$ and maximum and minimum values being 1 and 0 would take an infeasible amount of time. Paramters were trained for so many features by starting with the original four features and then adding the next parameter, but with the original four confined to the neighbourhood of their optimal values.

The weights derived give equation 4.

$$score = 0.35 \times S_{length} + 0.05 \times S_{para} + 0.25 \times S_{sent} + 0.2 \times S_{headline} + 0.15 \times S_{source} \quad (4)$$

9 Web Interface

To bring the system produced to a wider audience, it has been wrapped within a Java servlet. The Tomcat servlet container is being installed on a machine, to contain this servlet. A web page has been written to interface with the user. These components have yet to be correctly installed and deployed, but will be ready very shortly.

The result will be a webpage at which URLs or plain text can be inputted, and a summary returned. Percentage compression or a word limit can be specified for the summary. Check boxes to select the simplification and background info options are included.

10 Abandoned Ideas

In this section ideas that were briefly, or not so briefly, considered are discussed. This starts with thoughts from initial discussion in Autumn Term and finishes with something that was implemented but then scrapped.

10.1 RST-based summarization

RST[15] describes the discourse structure of a text[16]. The idea considered here was to construct an RST parse of the document and then apply various transformations over it. For example, one transformation could be to remove the satellites of a rhetorical relation, if the relation is far enough away from the root relation. Another (unrelated, but considered at the time) was to perform sentence compression upon remaining leaf nodes in the tree.

This solution required an automated RST parser, after using the RSTLIST mailing list [19] I was told that an automatic parser was not yet available. Writing one was beyond the scope of the project.

10.2 ContextScorer

The plan here was to run this PR after `SentenceScorer`. Suppose a document was represented as in Figure 15. The nodes in the graph represent sentences and the weights on the arcs are sentence similarity coefficients. As a pre-condition, each sentence i has associated with it a score S_i. After execution the score assigned to each sentence will equal

$$S_i + \sum_{\forall j} (S_j \times sim(i, j))$$

where *sim* is the similarity coefficient for the two sentences. Various similarity coefficients exist, most look at which terms or named entities occur in each sentence. The one used here was the cosine similarity coefficient: it is based upon the Vector-space model[17]. The cosine similarity coefficient is the cosine of the angle between two document vectors.

The idea was that if a sentence was similar to another, there scores would be brought closer to each other — this was intended to correct mistakes made by the `SentenceScorer`.

[15] Rhetorical Structure Theory

[16] RST describes the inter-sentential structure of a document. Sentences may bear relations to each other, for example, one sentence may provide *evidence* for, or *elaborate* on another. These relations may also hold between groups of sentences. There are many different type of rhetorical relation — in a number of them one sentence takes the role of *nucleus* and the other(s) *satellites*. A document may be represented as a tree of rhetorical relations.

[17] In the Vector-space model each term forms a dimension of a vector space. Documents are represented by vectors with the n^{th} element of a vector equal to the term frequency of the n^{th} term in the document vocabulary. So documents with identical numbers of each term will have identical vector representations and the more difference there is between documents, the further apart the vectors will get.

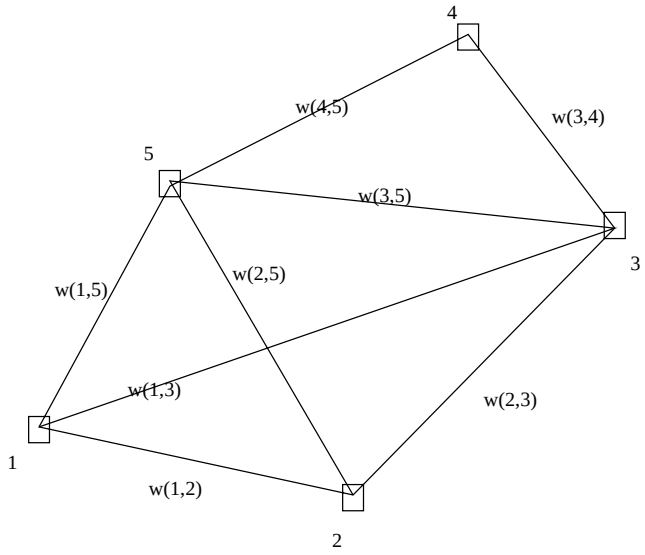


Figure 15: Document representation used in `ContextScorer`

After a little thought, it was realised that the transformation of scores described above is equivalent to pre-multiplying the vector of sentence scores S by the matrix of sentence similarity coefficients W, and then adding the product to the original S. So, A matrix library [14] was found and used, thus avoiding some unnecessary code-writing.

This idea was scrapped for a number of reasons.

Slow A typical document took thirty seconds to pass through this phase. That value was set to increase badly with document size — the matrix generated from the document would take quadratic time (in the number of sentences) to fill with values, and each value would take longer since the time taken to derive a similarity coefficient was proportional to the number of NEs.

This would not necessarily be a problem if it were not for the training that takes place in Part 8. There, the performance of the summarizer over a corpus of documents has to be measured at hundreds of different permutations. That would multiply the execution time — turning inconvenient into impractical.

Ineffective A manual examination of what the `ContextScorer` did to three documents gave unimpressive results. The selected sentences before the `ContextScorer` ran and after it ran were compared — it seemed to turn a reasonable summary into an entirely random selection of sentences.

Misguided The algorithm was expected to have a magical ability to fix all the ills of the `SentenceScorer`, an ability for which there wasn't any real basis.

Part IV
Evaluation

Producing any extractive summary is easy, producing a good one is the challenge and so some way of evaluating the effectiveness of a summary is a must. To evaluate the completed summarizer, the system was submitted to the DUC evaluation.

The DUC evaluation only examined the summarization aspect and not any of the simplification effort. The design of an extrinsic evaluation of that is described in Section 12.

11 DUC Evaluation

To encourage research into summarization, NIST[18] organised the Document Understanding Conference (DUC) Evaluation. DUC is a competitive evaluation of a wide range of summarization systems that has been run since 2000.

Systems evaluated in DUC include those produced by leading research teams at Columbia University and USC Information Sciences Institute, as well some from industry, Microsoft Research, Cambridge for example.

11.1 Method of Evaluation

The timetable of evaluation was:

29^{th} **November** Joined DUC, downloaded training data.

29^{th} **March** Test data made available. Once downloaded, no further modification of the summarizer was allowed, so download was postponed as far as possible.

15^{th} **April** Summaries of test data were submitted by this date.

31^{st} **May** Results made available.

11^{th}–12^{th} **July** DUC takes place in Philadelphia, PA.

Both training and test data was partitioned into "docset"s — sets of around ten documents, all about a common subject. That arrangement was for the evaluation of multi document systems. In all, there were 299 training documents provided, and extract-based summaries were provided for around half of them.

The test data provided had sentences boundaries already marked, for consistency between participants — so systems extracting sentences would all have the same idea of what a sentence is. The submissions to DUC were either single document or multi document summaries and either extracts or abstracts. The

[18] National Institute for Standards and Technology

extracts were evaluated automatically and the abstracts manually. Twelve systems were submitted for single document abstract evaluation — the category this system was submitted for. All of the multi document summaries were read and a random sample (five per docset) of single document summaries were read. This meant 295 of the 567 summaries submitted were read and evaluated.

Abstracts were evaluated in a number of ways, described in the following paragraphs. Model summaries were used for reference.

Quality Questions The questions used to assess the quality (in terms of language rather than information content) of the summary are given below.

1. About how many gross capitalisation errors are there?
2. About how many sentences have incorrect word order?
3. About how many times does the subject fail to agree in number with the verb?
4. About how many of the sentences are missing important components (e.g. the subject, main verb, direct object, modifier) - causing the sentence to be ungrammatical, unclear, or misleading?
5. About many times are unrelated fragments joined into one sentence?
6. About how many times are articles (a, an, the) missing or used incorrectly?
7. About how many pronouns are there whose antecedents are incorrect, unclear, missing, or come only later?
8. For about how many nouns is it impossible to determine clearly who or what they refer to?
9. About how times should a noun or noun phrase have been replaced with a pronoun?
10. About how many dangling conjunctions are there (”and”, ”however”...)?
11. About many instances of unnecessarily repeated information are there?
12. About how many sentences strike you as being in the wrong place because they indicate a strange time sequence, suggest a wrong cause-effect relationship, or just don't fit in topically with neighbouring sentences?

Answers returned would be one of

0 Zero errors

1 One to five errors

2 Six to ten errors

3 More than ten errors

Matching Peer Units with Model Units (marking) The model, human generated, summaries were split into Model Units (MUs). Similar sentence splitting of the system-generated summary (the "Peer" summary) occurred, giving Peer Units (PUs). If a PU can be matched with an MU, it is "marked".

Relevance of Unmarked Peer Units Having PUs that are unmarked but at least relevant is better than having irrelevant unmarked PUs, and that is reflected here.

Coverage within Peer Units This measure takes into account the proportion of the document content covered, and the number of words taken to do so. More specifically, the coverage metric is a weighted arithmetic mean

$$a \times c + (1 - a) \times b$$

where

$c =$ The proportion of the meaning of the Model Unit expressed by the marked Peer Units and b.

$b =$ Zero if the peer length is greater than some limit l, $\frac{l-length}{l}$ otherwise.

Length adjusted coverage sets $a = \frac{2}{3}$, length independent coverage had $a = 1$.

11.2 Results of Evaluation

Before evaluating the performance at DUC it should be remembered that the system submitted was produced over a period of months by one undergraduate student juggling other commitments, whilst other systems may have been the result of years of effort by teams of full-time researchers. So on that basis, it would be unreasonable to expect to be leading the field, and a below average performance would still be understandable.

System were assigned codes, the system produced here is 18. A baseline submission was created, and is included in the results below. Baseline summaries simply take the first 100 words or so, adding word sentences such that the word count doesn't exceed 115 words. It is indicative of how difficult a problem summarization is, when by a number of measures the baseline abstracts are the best. That is also perhaps a symptom of the structure of newspaper articles — the main points are usually mentioned in the first paragraph or so.

11.2.1 Performance on each Quality Question

Results in Table 2 shown where the mistakes were made, whilst Table 3 takes the mean score over all questions and compares systems. The table shows that the quality of the language was generally better than that produced by genuinely abstract-based systems, where new text would have to be generated. This is largely to be expected, since my system reuses the text of the original document

Question #	avg. error	mean avg. error
1	0.138983	0.34878
2	0.0372881	0.0426321
3	0.0135593	0.0194625
4	0.125424	0.179487
5	0.0813559	0.0920605
6	0.0169492	0.0231696
7	0.122034	0.0898981
8	0.227119	0.189682
9	0.0101695	0.0139018
10	0.0305085	0.0278035
11	0.0440678	0.0651838
12	0.19322	0.247451

Table 2: Quality of summaries produced, by question

and would therefore be of roughly the same quality as it. Although the quality of the summaries was generally better than that of other systems, it is worthwhile examining exceptions to that trend. The only questions performed worse on were numbers 7, 8 and 10.

7 This was the question about pronouns lacking (or having incorrect) antecedents. Some effort was put into solving this problem (in the form of `AnaphorRepairer`). One possible reason it persisted is errors made by the GATE Coreferencer. Also, no attempt was made to repair "it" anaphora because of the prohibitively low precision of the Coreferencer with them.

8 This asks how many nouns there are which are unclear as to who they are referring to. Performance could be improved by replacing named entity string with the longest string that corefers to it (derived from the NE's matches list).

10 This problem is to do with words such as "And" and "However" being out of place at the start of sentences. It could be solved in future by searching the summary for sentences beginning with such a word (a set would need to be defined) and including the preceding sentence in the summary if found.

11.2.2 Sentence Recall and Precision

Table 4 holds recall results and Table 5 holds precision.

$$recall = \frac{\#markedPUs}{\#MUs} \tag{5}$$

$$precision = \frac{\#markedPUs}{\#PUs} \tag{6}$$

System id	Mean Score
25	4.29938
16	1.69492
29	1.27456
28	1.1254
31	1.2
27	1.05763
18	**1.04067**
15	1.01356
21	0.92542
Baseline	0.768707
17	0.761092
19	0.745756
23	0.631038

Table 3: Quality of summaries produced, by mean score

System id	Recall
17	0.0834219
16	0.235564
18	**0.240376**
25	0.27139
23	0.279323
21	0.298814
28	0.301222
29	0.309238
15	0.31579
31	0.318465
19	0.334337
27	0.348226
Baseline	0.354338

Table 4: Model Unit Recall

System id	Precision
17	0.416683
25	0.578547
31	0.676387
Baseline	0.685807
15	0.69649
29	0.717294
27	0.746948
21	0.767861
18	**0.785932**
19	0.786962
16	0.794972
23	0.805402
28	0.822768

Table 5: Model Unit Precision

11.2.3 Unmarked Peer Unit Relevance

Table 6 states the fraction of PUs that are unmarked yet still relevant.

System id	Proportion relevant
17	0.233447
16	0.343729
18	**0.373559**
28	0.425085
23	0.428966
19	0.490847
21	0.493559
27	0.519322
29	0.573559
25	0.610884
15	0.624407
31	0.624407
Baseline	0.641497

Table 6: Relevance of unmarked Peer Units

11.2.4 Coverage

First the mean and median coverage with the length adjusted metric, and then the length independent metric.

System id	Mean Coverage
25	0.197078
16	0.201922
18	**0.21621**
15	0.221125
23	0.233541
31	0.240332
29	0.241305
21	0.246631
Baseline	0.246755
28	0.253224
27	0.259993
17	0.262188
19	0.266671

Table 7: Mean length-adjusted coverage metric

System id	Median Coverage
25	0.11285
16	0.120237
23	0.12821
18	**0.130942**
15	0.134519
Baseline	0.157153
29	0.161397
21	0.169047
31	0.172658
28	0.179119
27	0.185885
19	0.201336
17	0.217331

Table 8: Median length-adjusted coverage metric

System id	Mean Coverage
17	0.0819659
25	0.289643
16	0.302858
18	**0.322834**
15	0.331637
23	0.335252
31	0.360478
29	0.361227
Baseline	0.369806
21	0.369949
28	0.379617
27	0.383434
19	0.388553

Table 9: Mean coverage metric

System id	Median Coverage
17	0.0146758
25	0.163265
23	0.177241
16	0.180339
18	**0.194915**
15	0.201695
Baseline	0.235374
29	0.241356
21	0.253559
31	0.258983
28	0.268475
27	0.272203
19	0.290508

Table 10: Median coverage metric

12 Evaluating Simplification

Here an extrinsic evaluation would be most appropriate. A news article would be taken and put through the system. Test subjects would be school-children — half of them would be chosen at random to be given the original article and the other half would receive the summarized and simplified article. Both groups would be given the same amount of time to read the text and answer a series of questions on the subject of the document.

13 Strengths and Weaknesses

The strengths of this project are

Working system produced The system made has trained parameters wherever possible, anaphor repair, lexical simplification, information addition and a web interface.

GATE tested GATE is an upcoming technology in Language Engineering. This project adds to the growing number of applications of GATE and goes to show that it can be used for projects like this. Testing and bug reporting was done throughout, to the benefit of both GATE and this project.

Rigorous Evaluation The system was tested thoroughly in competition with leading summarization systems from around the world.

Things that could have been done better

Improved Quality This refers to the quality questions from DUC — some of them could have been addressed more directly in the design of the summarizer(and some didn't even required that much effort). If the quality questions had been referred to from the outset, that aspect of the evaluation would have improved.

Version Control The use of a version control utility such as CVS would have been wise. Although it wasn't necessary at first, it could have been useful in later stages.

Part V
Conclusion

14 Outcomes

The outcomes of this project were:

Working system produced With the following features

- **Trained parameters** Almost without exception, parameters to the sentence selection process were derived from a training corpus.
- **Anaphor repair** Where a sufficiently high precision was possible, dangling pronominal anaphora were repaired.
- **Lexical simplification** A number of different resources were brought together to simplify the words used within text.
- **Biography and map addition** Places and people were extracted from the text and used to search the web for background information.
- **Web interface implemented** A servlet was implemented to allow the system to be demonstrated on the web.

GATE tested GATE is an upcoming technology in Language Engineering. This project adds to the growing number of applications of GATE and goes to show that it can be used for projects like this. Testing and bug reporting was done throughout, to the benefit of both GATE and this project.

Rigorous Evaluation The system was tested thoroughly in competition with leading summarization systems from around the world at the DUC Evaluation. A paper is being submitted for inclusion in the Conference's proceedings and will be presented during the Conference itself.

Reusable utilities provided Due to the modular nature of GATE, and the flexible way in which my code was written, some of the modules produced could be reused in any other project that uses GATE. The `SyllableCounter` and `PsycholinguisticAnalyser` could be of most general use but any of the other components could be used in any information retrieval project. So if a document retrieval system were written in GATE, this project could provide summaries of found documents. Or if a rudimentary text-to-speech system were built around GATE, my `PsycholinguisticAnalyser` could provide pronunciations to the majority of the words in computer-readable format.

15 Challenges

A final year project isn't expected to run without difficulty. Some of the major challenges overcome include:

Working within GATE The initial plan, up until the Christmas break, was to use GATE as a utility for extracting named entities. I was to write my own classes to represent documents and sentences etc. Gradually it dawned on me that to do so would be a silly duplication of effort, since everything I needed existed within GATE. But the decision to use GATE entailed that my entire project would be reliant on it. This was problematic when I came across difficulties in using it, or bugs in its implementation. There wasn't any realistic way to reverse the decision to use GATE. But, as the final system shows, all GATE-related problems were overcome — thus improving GATE (with bug reports) and demonstrating its utility.

Discarding code There were times when code I had written had to be discarded. For example, as described above, the decision to use GATE wholly wasn't made from the outset. So a fair bit of code written before then had simply to be discarded. Similarly, `ContextScorer` required quite a lot of coding — matrix representation of sentences, similarity coefficients... — but it simply didn't make sense to retain it. Similarly when the coefficient for $S_{tf.idf}$ was found to be zero, that meant all the work on that feature, and on gathering document frequency values, would go unused. In both cases, it was a difficult but necessary decision to choose what would create the best system rather than retain code just because "it took ages to write that code, and I'm not throwing it away now".

16 Future Work

16.1 Document Visualisation

There are many immediate applications for this system. They include integration into a search engine, so that document summaries are provided instead of documents themselves. Integration into a document visualisation tool is another use. The *Information Navigator Suite* was extended in another final year project [18]. It can visualise documents or document clusters in a number of ways, including as points on a graph. Currently, holding the mouse over those points brings up the first few lines of the document — once this project has been integrated, a summary should appear instead. The same cannot be said for document clusters — that would require multi-document summarization.

Implementing a complete MDS system would quite easily be a final year project in itself. But if we refer to the stages of MDS in [12, page 180] we can see that some modification could be done to the current system to allow multi-document summarization.

Identify sentences to be extracted.

Match these sentences across documents, using some form of similarity metric.

Filter out repeated sentence, retaining the more salient ones.

Reduce sentences using generalisation and aggregation.

Present the information.

The first step has already been done for single document summaries. An option from here would be to:

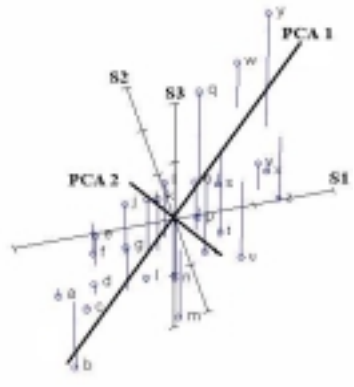
1. Plot the summary sentences for the entire cluster in a vector space.
2. Use Principal Components Analysis[19] to identify major topics in the cluster.
3. Retain the sentences most similar to the axes produced by PCA — I believe each of these axes will represent a topic. Only the closest sentence per axis should be kept, to prevent repetition of information.
4. Present these retained sentences.

Steps two and three correspond to matching and filtering. No reduction takes place and step four is obviously the present stage. Of course, patching a single document system to become a multi document one is hugely inelegant and starting from scratch would be preferable.

16.2 Simplification

There is much further potential in lanugage simplification. First in terms of lexical simplification, the problem of unnatural sounding replacements needs to

[19] Principal Components Analysis (PCA) is method which takes a set of multi-variable data points and rotates them such that the maximum amount of variablity is seen along the axes. For example here the original axes are $S1$, $S2$ and $S3$ — $PCA1$ and $PCA2$ are such that the range of values along those axes is maximised.



be resolved. One way would be to have a list of common collocations[20] against which the replacement could be checked.

The other area, practically untouched in this project but considered in initial discussions, is that of syntactic simplification. Potential simplifications facilitated by use of the Link Grammar [7] include turning a long sentence joined in the middle by an "and" into two sentences. Details were provided in the outsourcing report. Other ideas include turning passive sentences in to active ones and removing unnecessary adjectives and adverbs.

16.3 Background Information

The background information modules are essentially a prototype and further work could be done.

- Regarding biographical information, some disambiguation of the biographies returned could be done. A solution could be to choose the biography that contains dates and locations that also occur, or are nearest to, those mentioned in the document.
- The above problem also applies to maps.
- Sometimes the set of places in a document contains places that are close together. In that case it is confusing to have multiple maps when one would suffice. Instead a gazetteer could be used, which can map place names to longitude and latitude, and places close together could be confined to one map. Many public domain gazetteers are available.
- Another problem with maps is when unneeded maps are brought up. For example, the mention of a place may be just passing — "The strongest man in Britain" occurring in a document doesn't really call for a map of the UK. This might be solved by only bringing up a map after a threshold number of mentions are made. Or alternatively the system could be localised — the user would enter their location, causing the threshold number of mentions for a map insertion would decrease the further away the mentioned place was.

16.4 Sentence Representation

In the current design, the representation of a sentence is just one number — its score. Perhaps a vector could be used, each element being the score from each feature type.

Doing so would result in sentences being points in a seven-dimensional space[21]. This sort of data can readily be inputted into classification tools such as LNKnet [13]. LNKnet would allow more accurate models of the data to produced — the system as it is ignores which component contributed greatest to

[20] A collocation is simply a pair (or more) of word that often occur together in some pattern.
[21] since there are seven different features currently extracted.

the score and effectively partitions the space with a flat six-dimensional plane. LNKnet, on the other hand, could represent models that say things like "useful sentences have a high S_{coref} and low S_{length} or low S_{coref} and high S_{para}".

References

[1] Andrew Kennedy. Code to find Flesch readability index. http://rousseau.uwaterloo.ca/ ajkenned/courses/flesch.html.

[2] C-Y Lin and E. Hovy. Identifying Topics by Position. In *Proceedings of the* 5^{th} *Applied Natural Language Processing Conference*, pages 283–290, 1997.

[3] Catalina Barbu and Ruslan Mitkov. Evaluation tool for rule-based anaphora resolution methods.

[4] Chris Kennedy and Branimir Boguraev. Anaphora for everyone: Pronominal anaphora resolution without a parser. In 16^{th} *International Conference on Computational Linguistics*, 1996.

[5] Daniel Marcu. The Automatic Construction of Large-Scale Corpora for Summarization Research. In *Research and Development in Information Retrieval*, pages 137–144, 1999.

[6] Darren Pearce. Synonymy in Collocation Extraction. In *WordNet and Other Lexical Resources: Applications, Extensions and Customizations*, pages 41–46. Carnegie Mellon University, Pittsburgh, 2001.

[7] Davy Temperley, Daniel Sleator and John Lafferty. An Introduction to the Link Grammar Parser. http://www.link.cs.cmu.edu/link/dict/introduction.html.

[8] Marin Dimitrov. Personal Communication.

[9] Workshop on Text Summarization. http://www-nlpir.nist.gov/projects/duc/duc2001/agenda_duc2001.html.

[10] H.P. Edmundson. New Methods in Automatic Abstracting. In *Journal for the Association of Computing Machinery*, volume 16, pages 264–285, 1969.

[11] H.P. Luhn. The Automatic Creation of Literature Abstracts. In *IBM Journal of Research and Development*, volume 2, pages 159–165, 1958.

[12] Inderjeet Mani. *Automatic Summarization*. Natural Language Processing. John Benjamins Publishing Company, 2001.

[13] Information Systems Technology Group, MIT. LNKnet. http://www.ll.mit.edu/IST/lnknet/.

[14] Jean-Marie Dautelle. Java Addition to Default Environment. http://www.dautelle.com/jade.

[15] John Asmuth and Jesse Fischer. JavaWN — Java WordNet Interface. http://javawn.sourceforge.net.

[16] Karen Sparck-Jones. Automatic summarising: factors and directions. In Inderjeet Mani and Mark Maybury, editor, *Advances in Automatic Text Summarization*. MIT press, 1998.

[17] Marin Dimitrov. A Light-weight Approach for Coreference Resolution for Named Entities in Text. Master's thesis, University of Sofia, 2002.

[18] Matthew Carey. Retrieved Document Visualisation. Master's thesis, Imperial College, University of London, 2000.

[19] Matthew T. Bell. Personal Communication.

[20] M.D. Wilson. The MRC Psycholinguistic Database: Machine Readable Dictionary, Version 2. In *Behavioural Research Methods, Instruments and Computers*, volume 20, pages 6–11, 1988. http://www.bitd.clrc.ac.uk/PersonPublications/10209.

[21] N. Elhadad and K.R. McKeown. Towards Generating Patient Specific Summaries of Medical Articles. In *NAACL 2001, Automatic Summarization Workshop*, 2001.

[22] NationalGeographic.com and ESRI. The Map Machine. http://saltflat.nationalgeographic.com/mapmachine.

[23] Practical Simplification of English Text. http://www.cogs.susx.ac.uk/lab/nlp/pset/.

[24] Rudolf Flesch. How to write plain English. http://www.mang.canterbury.ac.nz/courseinfo/AcademicWriting/Flesch.htm.

[25] S. Azzam, K. Humphreys and R.J. Gaizauskas. Using Corefence Chains for Text Summarization. In *Proceedings of the Workshop on Coreference and its Applications*, 1999.

[26] Satoshi Sekine and C Nobata. Sentence Extraction with Information Extraction Technique. In *Workshop on Text Summarization*, 2001.

[27] W.C. Mann and S.A. Thompson. Rhetorical Structure Theory: Towards a Functional Theory of Text Organization. In *Text*, volume 8, pages 243–281, 1988.

[28] WordNet. http://www.cogsci.princeton.edu/wn.