Conversational News Agent

by

John Lai

A Thesis submitted to the School of Computer Science and Engineering

THE UNIVERSITY OF
NEW SOUTH WALES

SYDNEY · AUSTRALIA

In partial fulfillment of the requirements for the Degree of Bachelor of Engineering in Computer Engineering

November 2002

Supervisor: Prof. Claude Sammut

Assessor: Dr. Andrew Taylor

Abstract

Research in *Information Filtering* (IF) and *Agent Technology* have reached a stage where integrated implementations are rapidly flourishing in numbers. This thesis documents the research and implementation of "*CNA*", a novel integration of a voice-enabled *Conversational Agent* with an *Information Filtering* software, which fetches the latest news articles and learns topic preferences of the user.

The relevant fields of *Text Categorisation* and *Machine Learning* were surveyed, and various techniques were empirically verified in search of the ultimate prediction engine. The system is then integrated with a script-based Conversation Agent called "*Probot*." The end system engages users in *Natural Language* dialogues, delivering customised news articles.

Acknowledgements

First and foremost, I'd like to acknowledge my endless gratitude to my parents for their never-ending support and faith. Without their love and dedication, this life milestone in life would not be reached, when the completion of this thesis marks the completion of my undergraduate education.

I'd like to thank my friends Fiona Luk and Simon Bowden, whose support were indispensable in the most discouraging times. I would especially like to thank Harvey Tuch, who spent countless hours sharing his limitless thoughts and ideas, whenever I'm in need. I'd also like to thank Paul Winning for his kind assistance in testing the end system.

Finally, I would like to thank Waleed Kadous and Claude Sammut, whose expertise and guidance have steered me clear of troubled waters, and homed me in to destination.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Rationale

With the widespread adoption of the Internet as a medium for news delivery, the need for a tool to automatically filter uninteresting articles is desirable. In any single day, over 100 news articles may be posted on CNN.com, although an end-user may only be interested in a few. However, the concern that an aggressive filter that may incorrectly filter important articles is perhaps the greatest deterrent from user adoption. A major component of this thesis will be the analysis and benchmarking of current AI techniques in the application of user preference prediction, to decide whether automated filters are sufficiently reliable for their purpose.

As a result of Internt adoption, the routine by which users receive news had been revolutionised. With traditional broadcasting methods such as television or radio news, the users assume a passive role in news delivery, as the users have no direct control over the order nor content of news reporting. On the other end of the spectrum, Web-based news delivery requires users to undertake the active role in browsing headlines and deciding on interesting articles to read. However, with the vast amount of information delivered on a day-to-day basis, searching through headlines for the minority of interesting articles can be quite tiresome on the user. This thesis aims to implement a system that allows users to play a *semi-active* role in news delivery, by using a voice-capable *Conversational Agent* interface to let users to *chat* to the system to discover interesting news of the day, while uninteresting articles are algorithmically filtered.

1.2 Thesis Proposal

> *"What is this Conversational News Agent you're trying to build?"*
>
> *"It's a computer program that talks you about news of the day, and learns the topics you like."*

The proposed system is named "CNA", pronounced "*See-na*", is short for "*Conversational News Agent*." Elaborating on the meaning of title, the "*Conversational*" part means that the user will engage in *Natural Language* conversations with the system to achieve desired operations. The "*News*" part refers to the issue of news delivery, while the word "*Agent*" characterises the system as an intelligent software that learns about its user through interaction.

The envisaged system should allow the following interaction:

> **User:** ***What's happening in Afghanistan?***
> CNA: *Three detainees returned to Afghanistan. Would you like*

to read it?

User: ***Yeah go ahead***

CNA: *Three Afghans who had been in U.S. custody at Guantanamo Bay, Cuba, have been returned to Afghanistan, the International Committee of the Red Cross said Monday. Should I go on?*

User: ***No, can you go through today's headlines?***

CNA: *West Nile virus linked to 2 more deaths, read it?*

User: ***Not interested***

CNA: *Jupiter-like planet found near distant star, read it?*

User: ***Yes that I like***

Through conversations, CNA actively learns the news topics of interest and disinterest to the user, so that dull articles encountered in the future can be automatically filtered to enhance the user experience.

In order to implement a Natural Language interface, a Conversational Agent framework is required. The "*Probot*" is one such framework developed by Professor Sammut, the supervisor of this thesis, and it is also the aim of this project to investigate potential applications on this novel agent framework. The system will therefore utilise Probot as the user interface to support conversational interaction.

1.3 Thesis Overview

To implement the envisaged system, it is vital to survey existing literature in related AI research fields. Chapter 2 begins with an introduction to the Probot agent framework on which the system is to be deployed, and then provides an in-depth review of research in *Text Categorisation*, which is the crux of the problem of user preference prediction. Text Categorisation belongs to the research fields of *Information Filtering* and *Information Retrieval*, of which two main approaches to solving this problem, *Collaborative Filtering* and *Cognitive Filtering*, will be examined. Introductions to well-known *Machine Learning* algorithms, used extensively in Cognitive Filtering, will also be presented, followed by the statistical techniques that will be used to evaluate their performances in predicting user preference. Finally, systems previously built by researchers in this field will be summarised.

Chapter 3 will describe the system requirements of the Conversational News Agent, and lay out the design for integrating various components of the system. Chapter 4 will then document the implementation of the CNA in detail.

In order to find the best user-preference prediction engine for the system, Chapter 5 documents the experimental process undertaken to bechmark various Text Categorisation techniques, along with analytical comparisons to previously published results.

Finally, a discussion the outcomes of this thesis is presented in Chapter 6.

Chapter 2

Background

2.1 Chapter Overview

This chapter presents an introduction to research fields related to this thesis, starting with a brief introduction to agents and the Probot system, then focusing on the user preference prediction problem.

In *Information Filtering* (IF) systems, users interact with the system over an extended period of time, during which the user preference is learnt by the system. More specifically, the problem of classifying textual documents into categories, such as "*interesting*" or "*uninteresting*", is known as *Text Categorisation*.

IF stemmed from *Information Retrieval* (IR), which researches the problem of retrieving documents from a large database in response to user requests, such as a web search engine. IF systems are distinguished from IR in that results are tailored to the interests of each individual user, as opposed to serving users indiscriminately. Although the CNA is an IF rather than an IR system, it is important to present a brief overview of both research areas, since performance measures from IR is used extensively in IF.

Collaborative Filtering is a popular approach to the IF problem, where profiles about a group of users are gathered at a central database, and article recommendations for each user are derived from the feedback of other users with similar interests. However, Collaborative Filtering systems require a large user base in order to deduce meaningful correlations between users, and thus this approach is beyond the feasibility of this undergraduate project.

The other popular approach to IF is *Cognitive Filtering*, where prediction is based on the *content* of articles. The system would maintain a history of articles that a user has liked and disliked, and make new predictions by computing the similarity of words in the new document to historic ones. Before such computations can be made however, pre-processing must be performed to convert textual documents into a machine tangible representation. The *Porter Stemming* algorithm is commonly used to trim word suffixes so that words such as "*connected*" and "*connecting*" will be transformed to their common word root, "*connect*." This then allows documents to be represented as *vectors* in the Linear Algebra sense.

Once documents are converted to a *Vector Representation*, *Machine Learning* algorithms can then be applied. Several popular classification algorithms will be benchmarked in this thesis, including the *Support Vector Machine*, which has often been recognised as the state-of-the-art classifier in the literature. The statistical techniques widely used in the field to benchmark classifier performances will also be described.

Finally, related systems implemented by researchers in this field will be discussed, to identify their similarities and differences to the CNA.

2.2 Agents

2.2.1 Agents Overview

An *agent* is an intelligent system that interacts with its environment through sensors and effectors. Intelligent systems were originally expected to perceive through audio-visual input, communicate in natural language, have ability to reason, solve problems and learn from its own experience. However, this turned out to be much more difficult than expected, and research in intelligent systems has since branched into many fields, including *Natural Language Processing*, *Theorem Proving*, *Machine Learning* and *Vision*, so that each function can be studied in isolation. AI is now at a stage where integration of isolated functions into intelligent systems is feasible, and such systems are termed *Intelligent Agents*.

A *Learning Agent* is one that can acquire knowledge autonomously from the user or other agents, by means of direct instruction or by observation and imitation. Learning Agents could also learn from information databases or its own experience. The typical architecture of a Learning Agent adopted from [Tec98] is shown in Figure 2.1.

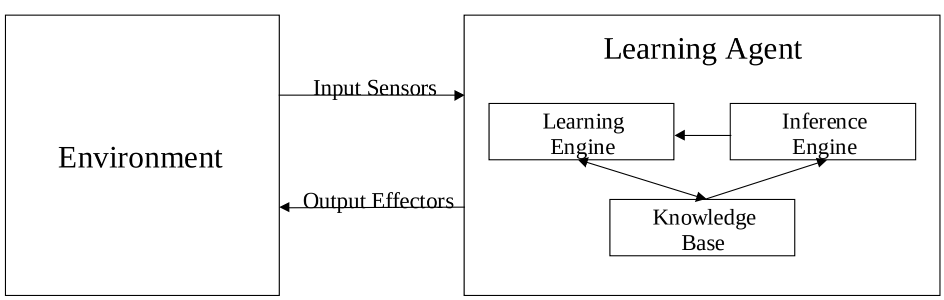


Figure 2.1: Learning Agent Architecture (from [Tec98])

However, one must bear in mind that developing a learning engine that captures the cognitive process of learning is a very difficult task, and thus most existing Intelligent Agents are arguably unqualified to be called Learning Agents.

Machine Learning is an AI research area focused on developing algorithms for systematic learning of concepts. Some learning strategies identified in the field are: [Tec98]

- *Empirical Inductive Learning From Examples* - Learn concepts by comparing the similarity and differences between positive and negative examples, then inductively generalise a description of the similarities of positive examples.
- *Conceptual Clustering* - grouping examples into different classes and learning a description of each class.
- *Reinforcement Learning* - updating knowledge based on feedback from environment.
- *Genetic Algorithms* - using genetic models of evolution to create a population of individuals over a sequence of generations.

The Conversational News Agent of this thesis is inherently a Reinforcement Learner; articles presented to the user will be scored, from which the agent will accumulate its knowledge base of the user's interests. Besides learning user preferences, which will be examined in great depth in later sections, the CNA will also engage in *Natural Language* interaction with the user. The conversational framework to be used by CNA is the "*Probot*" agent, which will be introduced in the next section.

2.2.2 Probot Scripting

A *Conversational Agent* must allow high level *Natural Language* interactions between the user and the system. Traditional *Natural Language Processing* methods have attempted to parse Natural Language input into syntactical representations, from which *semantics* and *pragmatics* are derived to allow machine understanding of the input. However, the complexity of such approach meant that existing Conversational Agents are far from their expected performances.

[Sam01] proposed a *rule-based system* called *Probot* which uses *shallow parsing techniques* as opposed to complex linguistic processing. Library of rehearsed responses are written in Probot scripts, allowing *scripted responses* to be delivered when built-in rules are successfully *pattern matched* to input utterances. These rules are trivial in general, and can be grouped into *contexts*, so that the scope of various rules can be restricted only to specific topics discussion. The system is built on top of the author's Prolog implementation, which can be called upon to perform more advanced logical processing.

The most commonly encountered constructs in the Probot scripting language are:

- `*`
  Wildcard for 0 or more words
- `~`
  Wildcard for 0 or more characters
- `#Prolog_Call`
  External call to Prolog

- `{ ... | ... }`
  List of rules or responses separated by pipe characters. (Random selection)

- `[ ... | ... ]`
  List of responses separated by pipe characters. (Sequential selection)

- `<` x `>`
  x represents a non-terminal symbol. For example, `<aff>` can be encoded as a non-terminal to represent affirmative words such as "*yes*", "*ok*", "*alright*" and "*yeah*" etc.

Response expressions are scripted with two types of constructs. Alternative responses listed in with braces ("{", "}") indicate that any of the alternative response may be chosen randomly for output when the same rule is triggered multiple times, while square brackets ("`[`", "`]`") would sequentially select an output on each rule trigger. Consider the example script below:

```
welcome ::
    init ==>
    [
        Hi there! Would you like to hear some news?
    |
        Hello! Shall we go through the latest headlines?
    ]

    * <aff> * ==>
    [
        #goto(read_headlines, [init])
    ]

    * <neg> * ==>
    [
        What topics would interest you then?
        #goto(search_news, [init])
    ]
```

```
{* computer~ * | * network * } ==>
[
    {
        Let me find some computer news!
    |
        I'll see if I can get some computer news!
    }

    #goto(computer_news, [init])

]
```

The first line indicates that the context is called “`welcome`”. Upon entry, the “`init`” rule is called first, which will greet the user with the “*Hi there!...*” response. The user must then responsd with a sentence of utterance, which will be patterned matched to rules in a top down fashion. For example, if the user says “*Well, alright then.*”, then the “`* <aff> *`” rule will be matched successfully, and will shift the conversation to the “`read_headlines`” context to read through the headlines. On the other hand, if the user gave a negative response, the “`* <neg> *`” rule will output “*What topics would interest you then?*”, and then jump to the “`search_news`” context to search for articles.

A third possible user response would contain no affirmative nor negative words, so that both the ‘`* <aff> *`” and “`* <neg> *`” rules would be skipped. If it then contained either “*network*” or a word starting with “*computer*”, the final rule in the “`welcome`” context will be triggered to jump to the “`computer_news`” context. However, if no rules were successfully matched, a *backup context* will be called upon to provoke temporary conversations, hoping that new inputs will be recognised by the system. In addition to the backup context, Probot also supports a *filter context*, which is a set of rules that will be attempted on the user utterance before the rules in the active current context are attempted. This allows universal rules to be defined, to avoid unnecessary duplication of rules. The full description of the Probot system can be found in [Sam01].

2.3 Information Retrieval and Filtering

2.3.1 Information Retrieval

Information Retrieval addresses the issues in *retrieving data from large document collections in response to user queries.* [She94] IR traditionally resolves around search engines where users enter keywords to search databases. It is a well-established field of information science, and three major paradigms have been identified in IR:

- *Statistical* - Selecting articles based on statistical correlations such as word counts in documents and document collections.
- *Semantic* - Selects articles by attempting to capture the meaning of the documents and user queries through *Natural Language Processing*.
- *Contextual/Structural* - Selects articles by taking advantage of the structural and contextual information

Although the CNA is not strictly an IR system [1], the IR literature has established well defined metrics that will be used to evaluate user preference prediction algorithms. These metrics will be defined in Section 2.3.3.

2.3.2 Information Filtering

Information Filtering addresses the problem of *finding desired information and eliminating those that are undesirable.* Filtering USENET newsgroups,

[1] CNA allows article searching by interfacing the search engine at CNN.com, but does not implement a retrieval system of its own.

where an enormous amount of messages are posted every day, is one of the most popular research areas of IF. [RIS+94, Lan95, She94] In contrast to IR, where the user's query is satisfied *within a single session*, the user interacts with the IF system through *multiple sessions* so that the user preference can be *learned over a period of time*.

Three approaches to Information Filtering have been identified in [MG87]:

- *Social Filtering* - Selects articles based on the relationships between people and their subjective judgments, recommending documents based on the feedback of other users.
- *Cognitive Filtering* - Selects articles based on the content of the articles. Keyword filtering is one of the most primitive forms of this approach.
- *Economic Filtering* - Selects articles based on the economic cost and benefit to the user through economic pricing mechanisms.

Since the CNA is not intended as a fee-based news service, the Economic Filtering approach is not applicable. Social and Cognitive Filtering however, are highly relevant approaches for the desired application, and will be described in further depth in later sections.

2.3.3 Evaluation Metrics

Formal evaluation metrics are necessary to characterise the performance of Information Retrieval and Filtering systems. The classical metrics of the literature are:

- *Accuracy* - The proportion of the system's interesting and uninteresting predictions that agree with the user's assessment.
- *Precision* - The proportion of the system's interesting predictions that agree with the user's assessment.
- *Recall* - The proportion of items of interest to the user that are presented to the user.

To avoid ambiguities, these metrics shall be formally defined using the definitions in [Alv02].

Formal Definitions

More formally, let a set of n news articles be the set U_+.
Let U_+ be a subset of news articles in U that the user rates as *interesting*.
Let U_- be a subset of news articles in U that the user rates is *uninteresting*.
Using *binary classification*, where each article is either *interesting* or *uninteresting*, it follows that:

$$U = U_+ \bigcup U_-$$

Let S_+ be a subset of news articles in U that the system predicts as *interesting*.
Let S_- be a subset of news articles in U that the system predicts as *uninteresting*.
Again, using binary classification, it follows that:

$$U = S_+ \bigcup S_-$$

Each news article in U will be classified into one of U_+ or U_-, and also one of S_+ or S_-. This means that each article will belong to one of four possible classification combinations:

Type 1: $S_+ \bigcap U_+$
Type 2: $S_+ \bigcap U_-$
Type 3: $S_- \bigcap U_+$
Type 4: $S_- \bigcap U_-$

Further, let the number of articles in each of the four combinations be represented by:

$$n_1 = |S_+ \bigcap U_+|$$
$$n_2 = |S_+ \bigcap U_-|$$
$$n_3 = |S_- \bigcap U_+|$$
$$n_4 = |S_- \bigcap U_-|$$

Then the accuracy a of the system is defined as:

$$a = \frac{n_1 + n_4}{n}$$

The precision p of the system is defined as:

$$p = \frac{n_1}{n_1 + n_2}$$

The recall r of the system is defined as:

$$r = \frac{n_1}{n_1 + n_3}$$

Interpretation

To assist in understanding the meaning of these metrics, the expected performance in accuracy, precision and recall will be examined from four general classifiers:

- *Random* - Randomly predict articles as interesting or uninteresting.
- *Present All* - A predictor that indiscriminately predicts all articles as interesting.
- *Present None* - A predictor that indiscriminately predicts all articles as uninteresting.
- *'Good' Predictor* - This characterises the predictive behaviour expected of a 'good' predictor.

For the purpose of comparing these classifiers, assume that a set of articles has been labelled by the user for testing, of which 50% are labelled *interesting* and 50% *uninteresting*. The expected performances are presented in Table 2.1.

Predictor	Accuracy	Precision	Recall
Random	50%	50%	50%
Present All	50%	50%	100%
Present None	50%	0%	0%
'Good' Predictor	$\gg$ 50%	$\gg$ 50%	$\gg$ 50%

Table 2.1: Expected Performance of General Predictors

Combined Metrics

Since multiple metrics exist for benchmarking classifiers, one needs to formalise how comparisons can be made decisively. Two common measures are:

- "*Precision-Recall Break-Even Point*" - For a given classifier, the precision and recall values often depend on internal parameters, which can be tuned to trade one for another. Thus the Precision-Recall Break-Even Point of a classifier is defined to be the point where precision is identical to recall through tuning of internal parameters.

- *F-Measure* - This was introduced by van Rijsbergen [VR79], and is defined to be:

$$F_\beta(p, r) = \frac{(\beta^2 + 1)pr}{\beta^2 p + r}$$

where β is the weighting parameter between precision and recall. When precision and recall are to be weighted equally, β is set to one, and thus:

$$F_1(p, r) = \frac{2pr}{p + r}$$

The F_1-Measure is effectively the *unweighted harmonic mean* of precision and recall. For the purpose of the evaluating classifiers from a *black-box* perspective, the F_1-Measure is a more suitable metric than the Precision Recall Break Even Point.

The F_1-Measure is perhaps the most commonly used metric in the IF and IR literatures, since combining only precision and recall means that the metric is based aound *positive classifications.* The set of articles classified as *uninteresting* by *both* the predictor and the user (Type 4) is not directly accounted for in the F_1-Measure. This can be justified by the following reasoning: If presenting articles that are actually interesting to the user (Type 1) are considered as *enhancements* to the end user experience, and Type 2 and 3 are considered as *detractions*, then Type 4 articles are *neutral* because they neither enhance nor detract the user experience. Such treatment of Type 4 articles are commonly accepted in the literature for Text Categorisation, but in some experiments in this thesis, it would be desirable to take into account of those articles through the accuracy measure. Thus the arithmetic mean of accuracy and the F_1-Measure will be used in some experiments when it is more appropriate than using F_1-Measure alone, although it must be noted that this procedure is not as commonly used.

2.4 Collaborative Filtering

2.4.1 Collaborative Filtering Overview

Collaborative Filtering is a form of *Social Filtering*, which classifies articles based on the *subjective evaluation of other readers*. In general, a database of user preferences is maintained so that predictions on new articles can be made for users with similar interests.

The basic steps involved in Collaborative Filtering are:

- Collect preference data for all users
- Train prediction algorithm on collected data
- Apply prediction algorithm on new articles
- Deliver articles most likely of interest to user

Two general classes of algorithms are identified:

- *Memory-Based* - performing computations over the entire database of historic user preference data to make each prediction.
- *Model-Based* - using the database to train a model, which is then used to make each prediction.

Both classes will be briefly described in the next two sections.

2.4.2 Memory-Based Algorithms

The general form of the *Memory-Based* prediction formula is

$$p_{a,j} = \overline{v_a} + \kappa \sum_{i=1}^{n} w(a,i)(v_{i,j} - \overline{v_i})$$

where $p_{a,j}$ is the predicted score on a new item j for user a, $\overline{v_a}$ is the average score of articles previously scored by user a. $w(a,i)$ is a *weighting function* for the *similarity between* users a and i, so that the score given to item j by user i can be used to predict the score for the same item for user a. κ is simply a normalising factor so that the sum of the absolute values of the weights is equal to one.

Qualitatively, if two users have similar interests, the weighting function would produce a high value, so that the score given to a new article by one user will be weighed strongly in making predictions for the other user. Common weighting functions (*Correlation* and *Vector Similarity*) as well as extensions to Memory-Based algorithms (*Default Voting*, *Inverse User Frequency*, *Case Amplification*, *Correlation Threshold* and *History Threshold*) can be found in [BHK98, Gok99].

2.4.3 Model-Based Algorithms

The general form of *Model-Based* prediction formula is

$$p_{a,j} = E(v_{a,j}) = \sum_{i=0}^{m} i \Pr(v_{a,j} = i | v_{a,k}, k \in I_a)$$

where $p_{a,j}$, the predicted score on a new item j for user a, is computed as the *probabilistic expected value* of the actual score $v_{a,j}$ using some underlying *probabilistic model* trained over the entire user preference database, given that the user has previously given scores to the set of articles I_a.

Qualitatively, the formula states that underlying model needs to determine the expected value of a score given the historic data about a user. The generality of this formula means that popular Machine Learning techniques like *Clustering* and *Bayesian Networks* can be applied in Model-Based Collaborative Filtering. [UF98, BHK98]

2.4.4 Suitability for Application

Having a large user base is essential in Collaborative Filtering. The time and resource constraints on this project means that the Collaborative Filtering approach would infeasible; gathering sufficient users required for successful predictions on an on-going basis is beyond the scope of an undergraduate thesis project.

2.5 Cognitive Filtering

2.5.1 Cognitive Filtering Overview

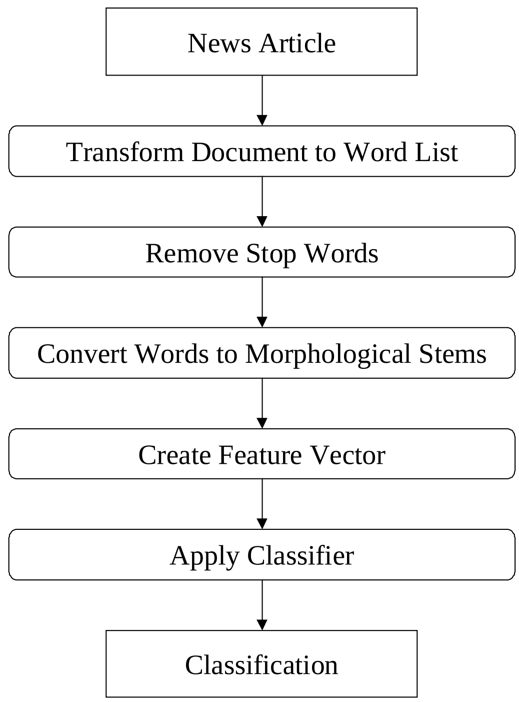


Figure 2.2: Cognitive Filtering Flow Model

Cognitive Filtering makes predictions based on the *words* in articles. Its most primitive implemention, known as *Keyword Filtering*, will scan articles for a set of user defined *keywords*, and use the *Boolean presence or absence* of those words to decide whether the articles are *interesting*. This is one of the most commonly used filtering techniques in E-mail clients.

Although the concept of Keyword Filtering appears too trivial, most of the more advanced Cognitive Filtering techniques still use Keyword Filtering as a pre-processor to eliminate "*Stop Words*", which are words so commonly used in every day English that they are considered valueless in the prediction process. For example, words such as "*the*", "*at*", "*by*" and "*said*" would fall in the class of common stop words.

In English and many other languages such as German and French, words often have multiple *morphological forms*. For example, the verb "*show*" has morphological forms "*showing*" and "*showed*", and the fact that all three words are semantically equivalent needs to be considered in the prediction process. It is therefore necessary to convert all words to their *word stems* so that documents about "*dog*" and "*dogs*" would be considered the same topic. The *Porter Stemming* algorithm is a popular method for converting words into their morphological stems.

In the IF and IR literatures, the *Vector Space Representation* of documents is commonly used. [She94, Sal83] These vectors, also commonly referred to as *Feature Vectors*, *Stem Feature Vectors* and *Attribute Vectors*, are used represent documents, where each *Feature Dimension* contains the *frequency* of some unique word stem. Using Feature Vector representation, various classifier algorithms can be applied to learn user preferences and make new predictions. These classifiers will be described in detail in later sections.

2.5.2 Porter Stemming Algorithm

Stemming algorithms *conflate* words with similar, if not identical, meanings to their common root, so that the meaning of these equivalent words are recognised in the process of Information Retrieval. For example, the following words are all of the same root, and should therefore be conflated to an identical stem:

`CONNECT, CONNECTED, CONNECTING, CONNECTION, CONNECTIONS.`

The following advantages of stemming are identified in the IR literature:

- To conflate words with identical/similar meaning to increase retrieval accuracy
- To reduce the total number of words, and hence reduce size and complexity of data in the IR system.

For the purpose of IR, the stemmer may produce stems that are not actual English words, provided that different words with the same meaning are converted to the same stem. For example, "*accessory*" may be converted to "*accessori*", as long as "*accessories*" also get converted to the same stem.

Stemming could be done by utilising a large dictionary mapping every word in the English vocabulary to their stems, but such approach would be resource intensive. The *Porter Stemming* algorithm was devised by Martin Porter, based on the idea that word suffixes are mostly combinations of smaller and simpler suffixes. This means that a set of *suffix replacement rules* can be devised and applied to achieve the desired result. The Porter Stemming algorithm is composed of five steps, in each of which *transformation rules* are *pattern matched* to the input word. On successful pattern matches, the transformation rule will be applied to remove parts of the stem, before passing the result to the next step for further stem removal.

Without the use of a stem dictionary, Porter noted that: [Por80]

- The suffixes are stripped simply to improve IR performance, not as an exercise in linguistics.
- The correctness of stemming using suffix-stripping rules will be significantly less than perfect. For example, if "*sand*" and "*sander*" are both converted to the same stem of "*sand*", then both "*wand*" and "*wander*" will likely be stemmed to "*wand*", but this would be incorrect. The "-*er*" in "*wander*" had been mistreated as a removable suffix when it is actually part of the stem. If more rules were added as special cases were encountered, the rule set will eventually reach a stage where *performance improvements in one area results in degradation elsewhere, and the system will have become more complex than necessary.*

Despite the minor imperfections, Porter's empirical results demonstrated that the simple algorithm performed no worse than the elaborate systems of its time. Over twenty years since its inception, the Porter Stemming algorithm has gained popular usage in the IR literature, and is widely used as the *de facto* standard stemmer. The original paper, as well as the author's own implementation of the algorithm, can be found at:

`http://www.tartarus.org/~martin/PorterStemmer/`

2.5.3 Feature Vector Representation

In the IF and IR literature, the *Feature Vector / Vector Space Representation* of documents is commonly used. [She94, Sal83] *Feature Vectors*, or *Document Vectors*, are used represent documents, where each *Feature Dimension*, or *Attribute*, contains the *frequency* of some unique word stem. As an example, let the Feature Dimensions be:

```
(australia, bush, blair, iraq, weapons)
```

Consider example phrase 1:

> *"Bush and Tony Blair have discussed war on Iraq"*

This would have a Feature Vector of

```
(0, 1, 1, 1, 0)
```

Consider example phrase 2:

> *"Bush visited Australia to discuss free trade between Australia and the US"*

This would yield a Feature Vector of

```
(2, 1, 0, 0, 0)
```

The example above used an extremely small set of Feature Dimensions. In common English, Feature Vectors may require over tens of thousands of dimensions to cover commonly used English words. Such high dimensions present high computation costs for Cognitive Filtering, and it is desirable to reduce the size to a feasible number. [YP97]

The Vector Space Representation of documents disregards the relative positions of words within a document, as IR literature claims that these *temporal properties* of documents are of minor importance for the purpose of *Text Categorisation*. [Joa98]

Normalisation of Feature Vectors to unit length is commonly used to *abstract the absolute size* of documents. Using the example Stem Feature set from above, consider the following two Feature Vectors:

```
A: (0, 20, 20, 0, 0)

B: (1, 1, 0, 0, 0)
```

Observe that article A has a total of 40 words, while B has only 2. Using a *distance measure* such as the *Euclidean Distance* function, it would appear that the two topics are dissimilar, when in fact both 50% of both articles are about "*Bush*," the second Feature Dimension. On the other hand, the normalised vectors would reduce, if not eliminate, the effect of absolute document sizes:

```
A’: ( 0, 0.5, 0.5, 0, 0)

B’: (0.5, 0.5, 0, 0, 0)
```

The two vectors A' and B' are now identical in the Feature Dimension representing "*Bush.*"

2.5.4 TFIDF Vector Scaling

Another popular vector transformation algorithm is the *Term Frequency Inverse Document Frequency* (TFIDF), which is applied prior to normalisation. This algorithm claims that *common words are not as useful in classifying documents as uncommon words.* By collecting word occurrence statistics over a large *corpus* (training data), *prior probabilities* of each word occurrence can be computed. Then, for each Document Vector to be classified, the occurrence of each word (*Term Frequency*) is multiplied by the inverse of its prior probability, so that those higher-than-expected word occurrences will be *accentuated* in the Document Vector. Several variants of the TFIDF equation exist, and the one tested in the system is of the following form:

$$\mathrm{TFIDF}_i = \mathrm{TF}_i \log(\frac{N}{\mathrm{DF}_i})$$

where TF_i is the term frequency of the ith word stem in a document. $\frac{N}{\mathrm{DF}_i}$ is the inverse of the prior probability of occurrence of the ith word stem, where N is the number of documents in the training corpus, and DF is the number of documents in which the ith word stem occurs in the training corpus.

It is often acknowledged in the IR literature that TFIDF improves classifier performance. [Joa98] However, without assuming the claim that accentuating uncommon words improves performance, tests will be conducted to empirically verify the usefulness of TFIDF.

2.6 Machine Learning Algorithms

Machine Learning is an active research area of AI where patterns in data are algorithmically mined. In application to the CNA, Machine Learning algorithms will be applied to learn topics of interest to the user, so that uninteresting articles encountered in the future can be automatically filtered. The six algorithms to be evaluated in this thesis will be described in this section, ranging from toy classifiers like *ZeroR* and *OneR*, to well regarded classifiers such as the *Naïve Bayesian Classifier*, *Support Vector Machine*, *J4.8* (*C4.5* variant) and *k-Nearest Neighbour*.

2.6.1 ZeroR

ZeroR is the most naïve classifier tested. It is a *majority predictor*, as it always predicts the *majority class of training data*. For example, if the majority of training data were *positive examples*, then ZeroR will always predict a positive score, regardless of the input.

This naïve, and arguably useless, classifier is included to serve as a bottom line benchmark. If some classifier performed worse than ZeroR, it would signal serious errors such as *over-fitting*, or that the classifier in question is not worth using at all.

2.6.2 OneR

OneR is another classifier serving as a bottom line performance benchmark. Given the training set of Feature Vectors, OneR will pick just one of possibly thousands of Feature Dimensions as the sole judging criterion. In the training phase, the training data will be split into a positive (interesting) and a negative (uninteresting) set, and a threshold value will be computed for each Feature Dimension so as to minimise prediction error on the training data, based on the single feature. The Feature Dimension with the least number of prediction errors will then be used as the sole prediction attribute.

2.6.3 Naïve Bayesian Classifier

The *Naïve Bayesian Classifier* is based on *Baye's Formula for Conditional Probability*:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

This states that the probability of A occurring given B has occurred is equal to the probability of A and B occurring simultaneously, divided by the probability of B occurring. When applied to Text Categorisation, the conditional probabilities of an article being classified as `HIT` (interesting) or `MISS` (uninteresting) are computed for each Feature Dimension, i.e. the probability of an article being a `HIT` or `MISS` given the word frequencies of each dimension.

When a new article is encountered, each Feature Dimension (i.e., unique word stem) will be examined in turn, and the probability of that article being a `HIT` and a `MISS` will be computed based on its word frequency. Two total probabilities, one for each of the `HIT` and `MISS` classes, will be deduced by multiplying together the individual probabilities of each of their respective Feature Dimensions. The class with the higher probability will be the outcome of the prediction.

The classifier is *Naïve* in that words from the Feature Dimensions are assumed independent, i.e., the conditional probability of a word occurrence given a class is assumed independent from the conditional probabilities of other words in the category. [YL99] An example to illustrate the naïveness of this assumption is as follows: The words "*September*" and "*Eleventh*" have very little meaning by themselves, but together they have a very specific reference to the terrorist attacks on the World Trade Centre in New York on September 11th 2001. The Naïve Bayesian Classifier however, would treat the two words as if they had no mutual dependence.

2.6.4 Support Vector Machines

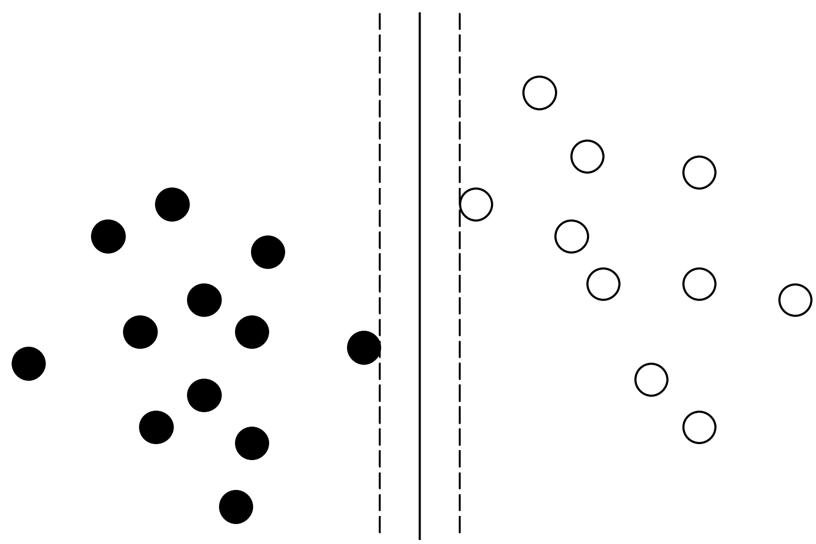


Figure 2.3: SVM - Non Optimal Decision Surface (from [YL99])

The *Support Vector Machine* was introduced by Vapnik in 1995 for solving two class problems, by finding the *optimal decision hyperplane*. [Vap95, YL99]

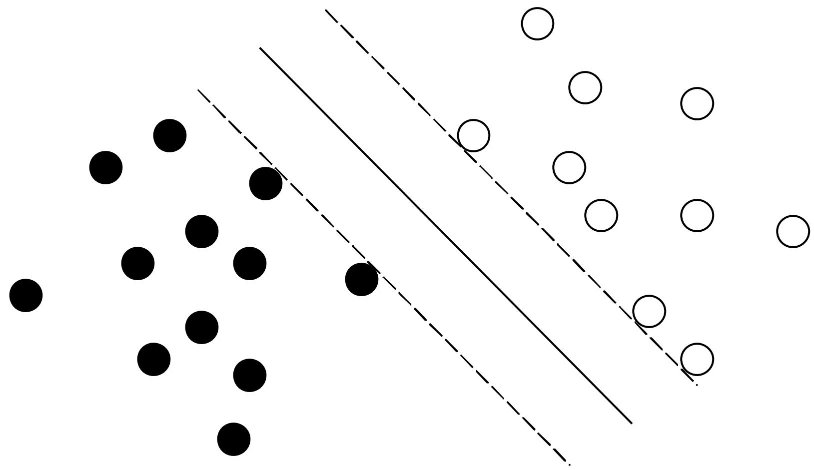


Figure 2.4: SVM - Optimal Decision Surface (from [YL99])

Given a set of training vectors representing documents, the optimal hyperplane is defined to be the one that *maximises the margin between the two classes*, as illustrated by Figures 2.3 and 2.4 with a simplistic two-dimensional example. Despite the fact that both *decision surfaces*, the solid lines, separate the two classes without any misclassifications, the one in 2.4 had a much wider margin than 2.3. This idea can be generalised to high dimensional spaces for Text Categorisation by using the *Structured Risk Minimisation* algorithm [Vap95] find the 'best' decision hyperplane separating the two classes.

Unlike the k-NN and Naïve Bayes algorithms, where every point in the training form some part of the derived model, only points on the margin of the decision surface, called *'support vectors'*, are significant in describing the decision surface. These points are those touching the dashed lines in Figure 2.4. With the decision surface determined by solely by these support vectors, classifications are performed without regard to all other points in the training set, leading to significant performance advantage for the SVM. It is worth noting that SVM is often regarded as the best performing classifier in the literature, and this claim shall be verified in Chapter 5.

2.6.5 J4.8 Classifier

J4.8 is an implementation of the *C4.5 Decision Tree* classifier by Quilan, who extended his previous *ID3* algorithm to take into account for unavailable values, continuous attribute value ranges, pruning of Decision Trees and rule derivation etc. Each node of a Decision Tree represents a decision based on a certain attribute of the Feature Vector, and the node branches to children nodes based on the value of the attribute. When given a new instance to classify, the tree is traversed from the root node downwards until reaching a leaf node, which represents a classification. A toy Decision Tree is shown in Figure 2.5.

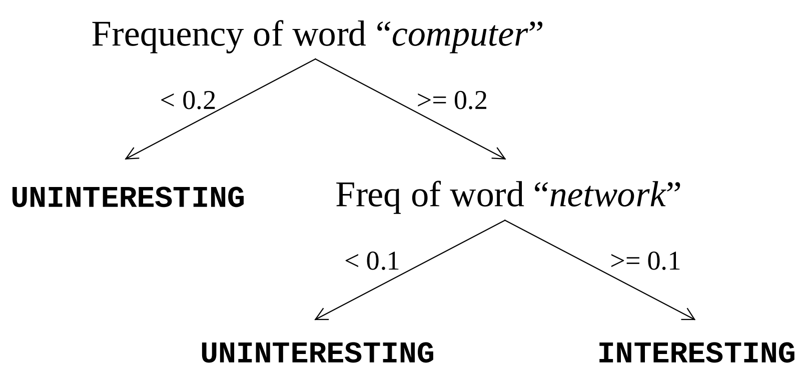


Figure 2.5: A Simple Decision Tree

The C4.5 algorithm can handle three types of attribute testing: [Qui93]

- *Standard Discrete Attribute* - The standard test on discrete attributes, with one outcome and branch for each possible value of each attribute.

- *Complex Discrete Attribute* - A more complex test, also based on discrete attributes, in which the possible values are allocated to a variable number of groups, with one outcome for each group rather than each value.

- *Continuous Attributes* - If attribute A has continuous numeric values, a binary test, with outcomes $A < Z$ and $A \geq Z$, can be performed by comparing the value of A against some threshold value Z.

Since the Feature Vectors used in this thesis are normalised to unit vectors, which have continuous numeric values, the third type of testing is used. The chosen threshold for each decision node must maximise performance, which can be deterministically calculated with a simple method: Given m vectors, sort them by their values in the attribute A being considered. Although having continuous numeric values means that there will be infinite number of equally optimal thresholds, on of these m values of attribute A must also be one of the optimal candidates. Therefore all m possible thresholds can be enumerated to find the optimal threshold. Using *memoisation*, the process can be performed in one pass through the m sorted values, by updating the distributions to the left and right of the threshold on the fly. [Qui93]

In order to explain how the most 'useful' attributes are identified by the algorithm, it is necessary to define *Entropy*, *Information Gain* and *Gain Ratio*.

Let each article be represented by a Feature Vector of i dimensions. Let S be the full training set, and let p_+ and p_- be the proportion of *interesting* and *uninteresting* articles in S respectively. The *Entropy* of S is defined as:

$$\text{Entropy}(S) = -(p_+)\log_2(p_+) - (p_-)\log_2(p_-)$$

This measures gives a measure of the impurity of S. The *Information Gain* from splitting the set S using Feature Dimension i is then given by:

$$\text{InformationGain}(S,i) = \text{Entropy}(S) - \sum \frac{|S_v|}{|S|}\text{Entropy}(S_v)$$

where S_v are the subsets of S containing vectors with values within certain ranges in Feature Dimension i. Consider this example: Since the Feature Vectors represent normalised word frequencies (continuous numeric values), a numeric threshold will be chosen to split S into two S_v subsets, the first being the set of vectors with values *less than* the threshold in dimension i, and the other being the set of vectors with values *greater than or equal to* the threshold.

C4.5's predecessor, the ID3 algorithm, uses the Information Gain criterion to determine the best attributes for classification. However, the author observed that the Information Gain suffered a serious deficiency: Information Gain biases in favour of tests with many outcomes. Although this is not as apparent in the continuous numeric attribute case, since there are only two outcomes from each decision node, this bias was rectified in the *Gain Ratio* criterion, defined by:

$$\text{GainRatio}(S,i) = \frac{\text{InformationGain}(S,i)}{\text{SplitInfo}(S,i)}$$

where

$$\text{SplitInfo}(S, i) = -\sum_{j=1}^{n} \frac{|S_j|}{|S|} \log_2(\frac{|S_j|}{|S|})$$

and j iterates through the n possible outcomes for attribute i, forming a subset S_j of vectors with equal values in attribute i. In application to continuous numeric attributes, n would be set to 2, since the set S is split into two subsets using a numeric threshold at each decision node.

In building the Decision Tree, the attribute with the highest Gain Ratio will be used as the root of the tree. The process is then repeated using the divide-and-conquer approach to generate the remaining sub-trees.

2.6.6 k-Nearest Neighbour

The *k-Nearest Neighbour* classification has been studied intensively in the field of *Pattern Recognition*. It an *Instance-Based Learner*, which classifies unseen cases by recalling similar remembered cases. The algorithm itself is trivial: Given an unlabelled item, use some distance metric to find the k nearest neighbours in the labelled training set, and label the new item as the majority class of the k nearest neighbours.

The distance metric most commonly used is the Euclidean distance, which given two vectors A and B both of dimension N, the distance is given by

$$\sum_{i=1}^{N} \sqrt{(A_i - B_i)^2}$$

where A_i and B_i are the ith dimensions of vectors A and B.

Another commonly used distance metric is the *cosine angle vector similarity* - given two normalised unit vectors, the cosine of the angle is simply the *dot product* of the two vectors. This results in a numeric value for similarity between -1 and 1, where an extreme value of 1 would mean that the two vectors are identical. A value of -1 would mean that the vectors are 180 degrees apart, i.e. have no similarities at all.

In binary classification, where each item is classified as either a `HIT` (interesting) or a `MISS` (uninteresting), k can be set to a odd number so that no ties are possible when determining the majority class of the k nearest neighbours.

2.7 Statistical Techniques

For comparisons between classifiers to be made, rigorous statistical testing is required. A common practice in the literature is to perform 10 runs of *ten-fold cross-validation* for each classifier, and then use the *Paired-Sample t-test* to compare the results. Both procedures are described in this section.

2.7.1 Cross-Validation

In theory, in order to deduce fair values for the accuracy and F_1-Measure of a classifier, two large sets of independent data should be used, one for training and the other for testing. However, it is often difficult to obtain another large set of labelled testing data on top of the training set, and therefore the *cross-validation* method is commonly used to generate internal estimates of the classifier performance using a single set of labelled data.

For a *N-fold cross-validation*, the labelled set is randomly partitioned into N subsets of nearly if not equal sizes. N runs of testing are then executed, with each run holding out one of the N subsets as testing data, while the other N-1 subsets are used as training data. The average of the N runs is then used as the output result.

The advantage of using N-fold cross-validation is that the random partitioning of eliminates any arbitrariness in the formation of training and testing data, making the results more statistically reliable than using fixed sets of training and test data. For this reason, the ten-fold cross-validation is perhaps the most reliable and commonly used method for evaluating classifiers in the literature.

2.7.2 Paired-Sample t-test

The *ten-fold cross-validation* testing is used to evaluate the performance of a classifier, but another statistical technique is required to compare classifiers to determine if one significantly outperformed the other. Suppose that ten runs of ten-fold cross-validation were performed for two different classifiers. Comparing the raw average performance of the two classifiers would be statistically unacceptable, since the statistical variations would not have been taken into account. However, comparing the standard deviations of the two classifiers could be just as unfruitful, since the performance difference *between runs* may dominate the performance difference *between classifiers*.

In the Paired-Sample t-test, both classifiers are subjected to same series of tests, and the *difference* between the classifiers' performances in each of the individual tests are used to compute a mean and standard deviation.

The t-test can then be used to determine whether the performance difference is significantly different from zero. A *P-Value*, representing the probability of the two classifiers being *indifferent* in performance, will be calculated from the difference values using the *t-distribution*. If the P-Value is below a desired significance threshold, then the test would conclude that one classifier has outperformed another. Otherwise, the two classifiers are considered statistically indifferent/equivalent in performance.

For the purpose of comparing two classifiers, ten runs of ten-fold cross-validation will be performed for each classifier. Random seeds will be used to ensure that the random partitioning is different between runs, but kept consistent within each run for both classifiers. This would subject both classifiers to the same series of tests as required by this statistical method.

2.8 Related Works

Resnick *et. al.* developed the *GroupLens* open architecture [RIS+94] which uses the *Collaborative Filtering* approach to filter USENET newsgroups. News reader clients compatible with the architecture display articles to users along with their customised predicted scores, and allows user feedback to be gathered at a central rating server. The server uses the *Correlation* approach to make predictions, using the heuristic that people who agreed in their scores in the past are likely to agree again.

PHOAKS is another Collaborative Filtering system built around USENET newsgroups, but instead of recommending articles, the system recognises and tallies web URLs mentioned in newsgroup articles to recommend popular websites. [THA+97]

In [She94], Sheth developed *Newt* as a collection of agents using the author's proposed keyword based filtering algorithm to personalise USENET newsgroup filtering. The GUI displays multiple *Information Filtering Agents*, with each specialising in a different news topic. The agent system supplements manual browsing of news, rather than being a complete environment. The article being displayed can be scored by giving a positive or negative feedback to one or more of the agents. For example, if the current article being viewd by the user is an interesting one about sports, the user would click the "+" button on the agent for sports. A power user can also examine the internal state of each agent to edit behaviour configurations. After sufficient training, the user would click on a specific agent's icon to read the articles retried by that specialist agent.

The *Newsweeder* system in [Lan95] is also a USENET agent, allowing users to access newsgroups through a web interface. Initially, the user reads and actively scores in existing newsgroups, from which user preference is learned. After some training, the user can then access a virtual newsgroup, which is a collection of interesting articles extracted from real newsgroups.

[PS96] presents a *Learning Personal Agent* which scouts sources on the WWW for conferences and *Request For Proposals* that fit the research interests of the user. The architecture splits the task of polling USENET newsgroups and learning user preference using two separate agents, utilising a defined communication protocol for article exchange between the two. Two prediction engines were tested for the system: *Cluster Mean Classification* using the *cosine angle distance metric*, and a *Neural Network*. However, thorough development and testing was not evident in this research.

The Conversational News Agent being developed for this thesis is different from the existing agents in the following regards:

- The agent is a complete news delivery environment, rather than a supplementary component.
- News articles are delivered instead of newsgroup posts.
- Implicit scoring is used to reduce user burden.
- A conversational interface is used, allowing news articles to be delivered in conversational segments rather than on-screen text dumps.
- Voice-enabled interaction is supported.

Chapter 3

System Framework

3.1 Chapter Overview

This chapter will describe the framework designed for the implementation of the CNA. First, the general requirements of the system will be summarised, followed by a high-level block diagram illustrating the interaction between system components. After the high-level overviews, more specific functional requirements will then be presented, followed by a set of *Data Flow Diagrams* showing the detailed internal data processing of various components.

3.2 General Requirements

- Support multiple user profiles.
- Use CNN.com as the news source, since it is a well respected news provider, with topics covering World News, Weather, Business, Sports, Politics and Technology etc. The broad coverage makes a single news provider sufficient for the desired application.
- Support conversational voice interaction with user, to let users undertake a relaxed *semi-active* role in receiving news.
- Allow latest headlines to be cycled through when the user has no specific news requests.
- Allow searching of specific topics when the user has specific news requests.
- Underlying prediction engine filters uninteresting articles to enhance the end user experience.

3.3 System Overview

The high-level block diagram of the system components is shown in Figure 3.1 The user interacts with the scripted Probot agent, which supports a text-based interface, as well as an animated face called "*Sophie*", which supports voice recognition and synthesis. Beneath the front-end lies the CNAClient, which accesses the headline database compiled by the CNAServer and fetches news articles from the news source.

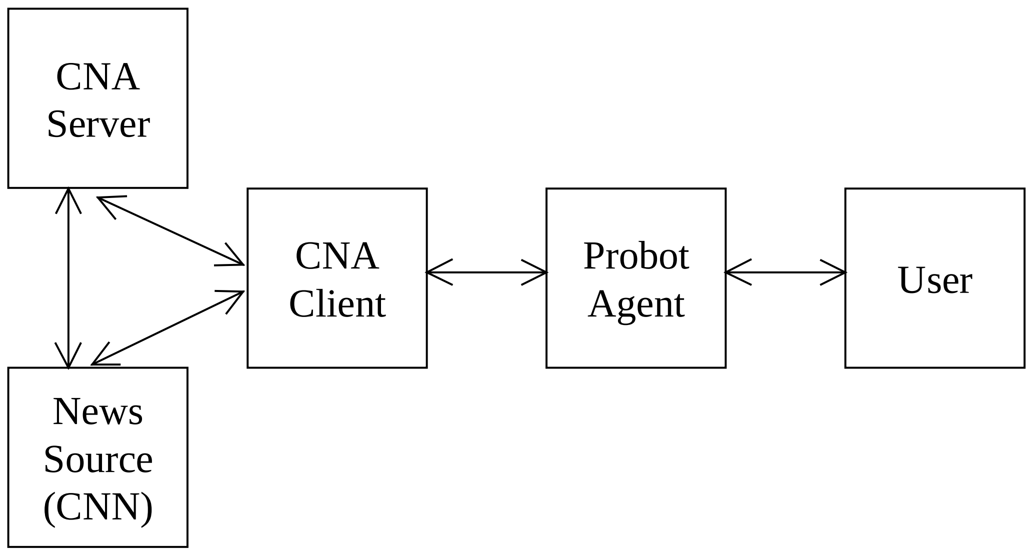


Figure 3.1: Overview of System

3.4 Functional Requirements

Having divided the system into manageable components, the specific functional requirements of each component is presented in this section. These specific functions, when integrated, allows the desired system to be achieved in a logical *divide and conquer* engineering approach.

3.4.1 CNAServer

- Extract headlines and article links from CNN
- Maintain local database of recent headlines
- Extract text-only articles from CNN

- Cache articles for mass retrieval by CNAClient

3.4.2 CNAClient

- Well defined I/O interface to accommodate Probot front-end.
- Update headlines and article caches from CNAServer
- Support multiple user profiles
- Allow transparent access to CNN's search engine
- Allow user to cycle through latest headlines
- Extract text-only articles from CNN
- Utilise prediction algorithm to filter uninteresting articles using each user's historic profile

3.4.3 Probot Front-End

- Serve as the *Natural Language* interface between user and the CNA-Client
- Allow user to access news in a conversational fashion
- Utilise "*Sophie*" to support voice interaction, which integrates *Voice Recognition* and *Voice Synthesis* sub-systems.

3.5 Data Flow Diagrams

The *Data Flow Diagram* (DFD) is a software engineering tool by which data processing functions of the system can be illustrated. In this section, DFDs of the CNA system is presented, starting with the Context Level Diagram summarising the high-level interaction between the system and external enterties, namely the user and the news source. The Level 0 DFD shows the overall interaction between components within the system, and the more sophisticated components, namely the CNAClient and the CNAServer, will expanded further into Level 1 and 2 diagrams to demonstrate the internal operations. Having the system functions modelled with Data Flow Diagrams, implementation can then be carried out in an organised fashion.

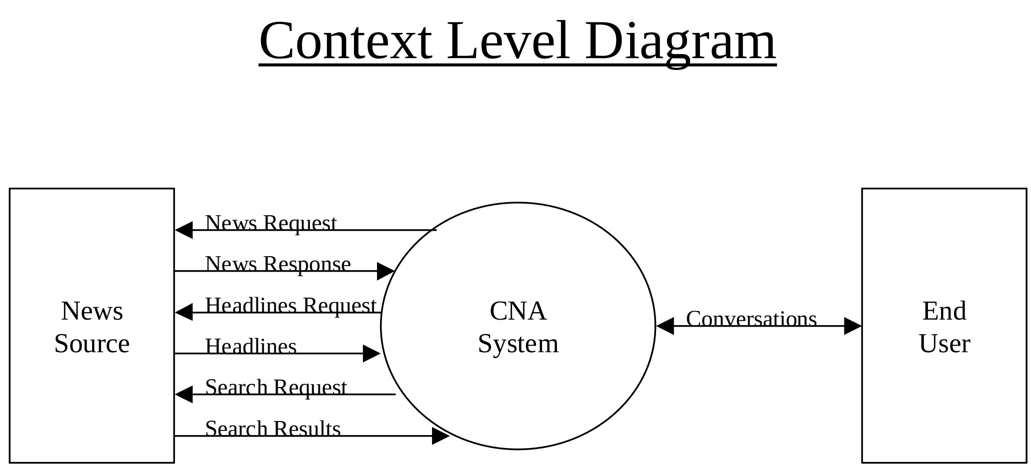
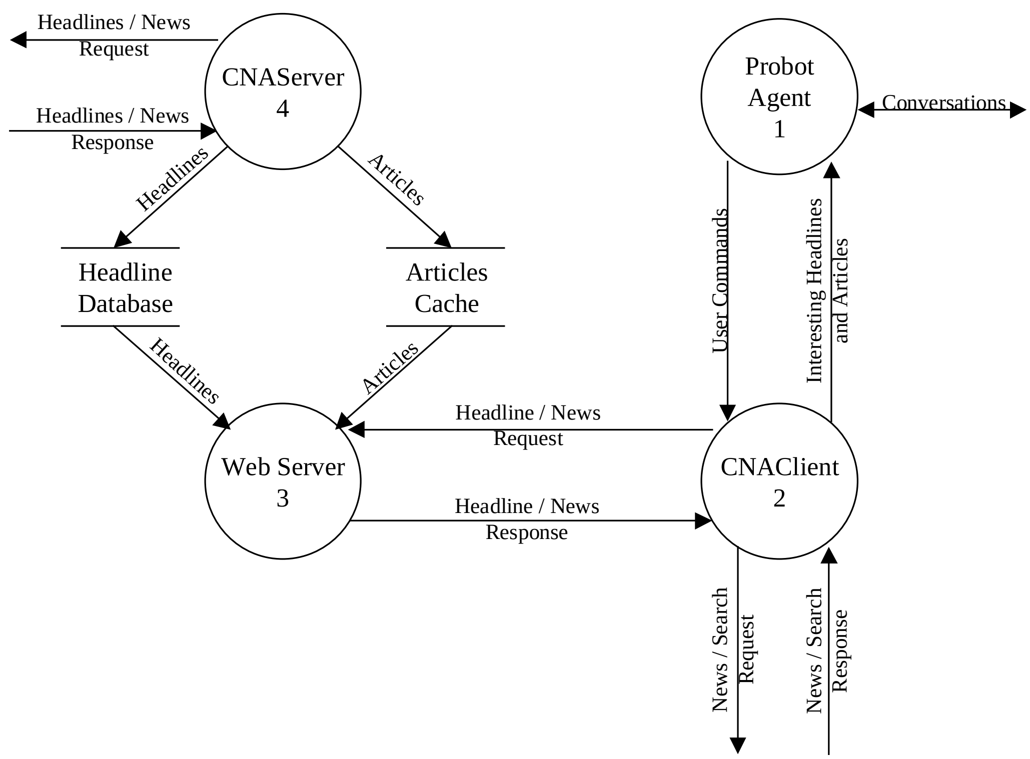


Figure 3.2: Data Flow Diagram - Context Level Diagram

Level 0 Diagram



Note: Processes 2 and 4 can both independently send and receive News Requests/Responses to and from the external news source, but for clarity, the news request/response flows had been merged with other flows in this diagram. In addition, note that the CNAClient in process 2 can fetch news from through Process 3 from the CNAServer's articles cache, as well as directly from the news source.

Figure 3.3: Data Flow Diagram - Level 0 Diagram

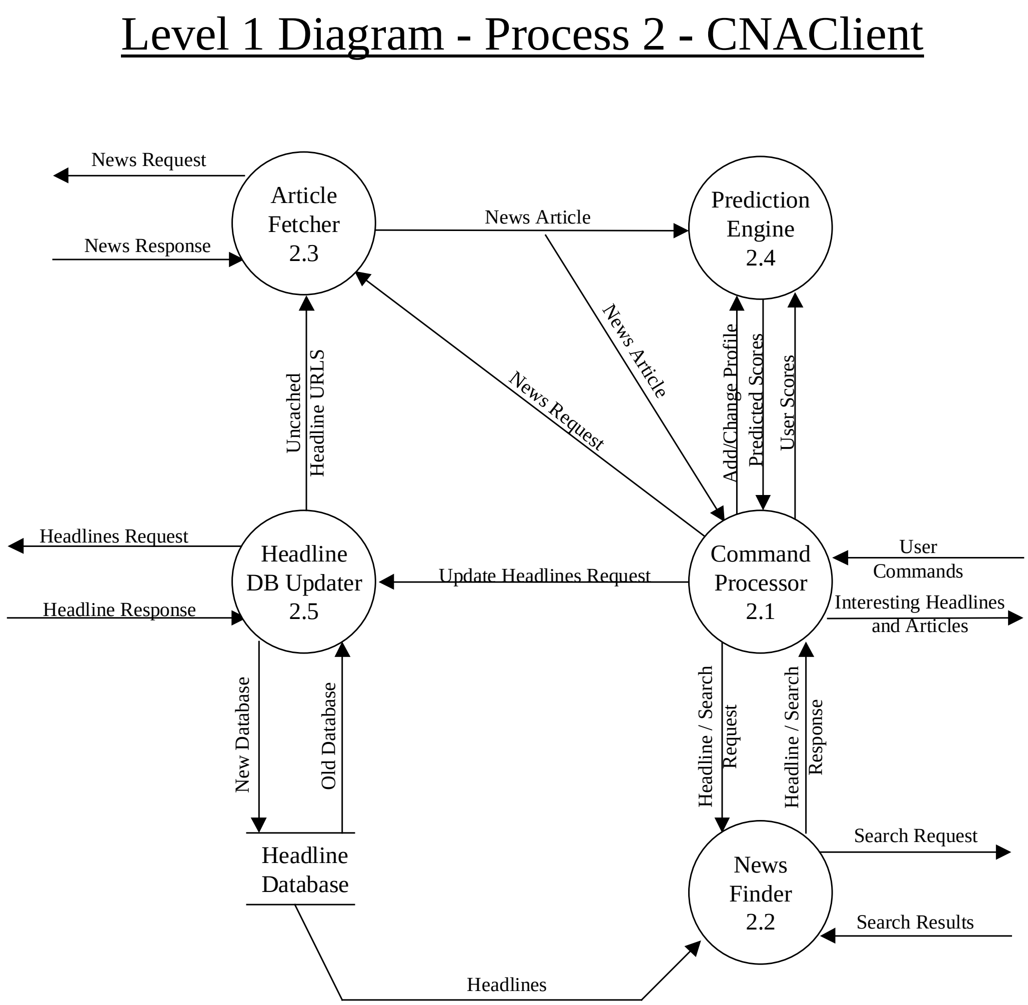


Figure 3.4: Data Flow Diagram - Level 1 Diagram of CNAClient

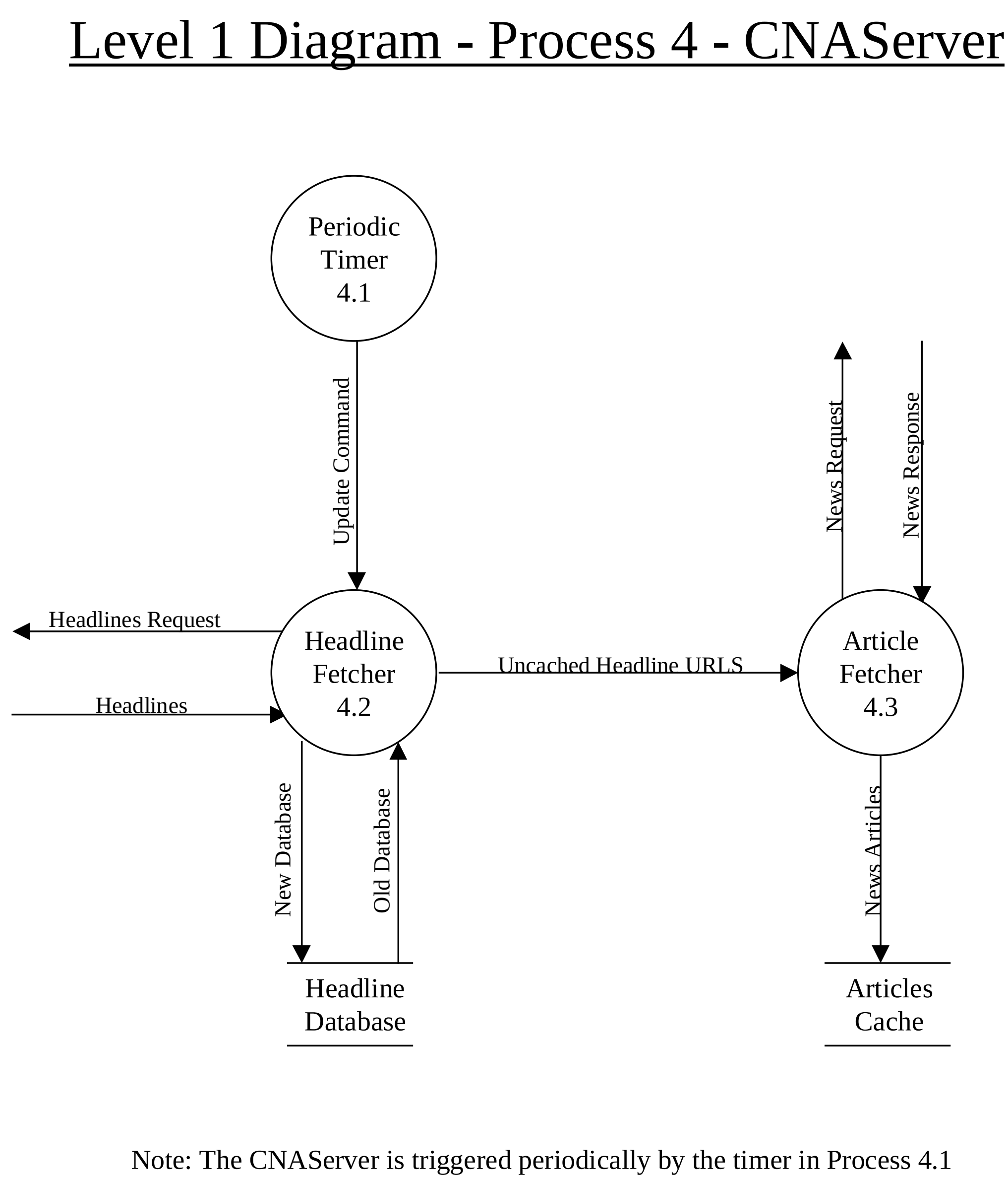


Figure 3.5: Data Flow Diagram - Level 1 Diagram of CNAServer

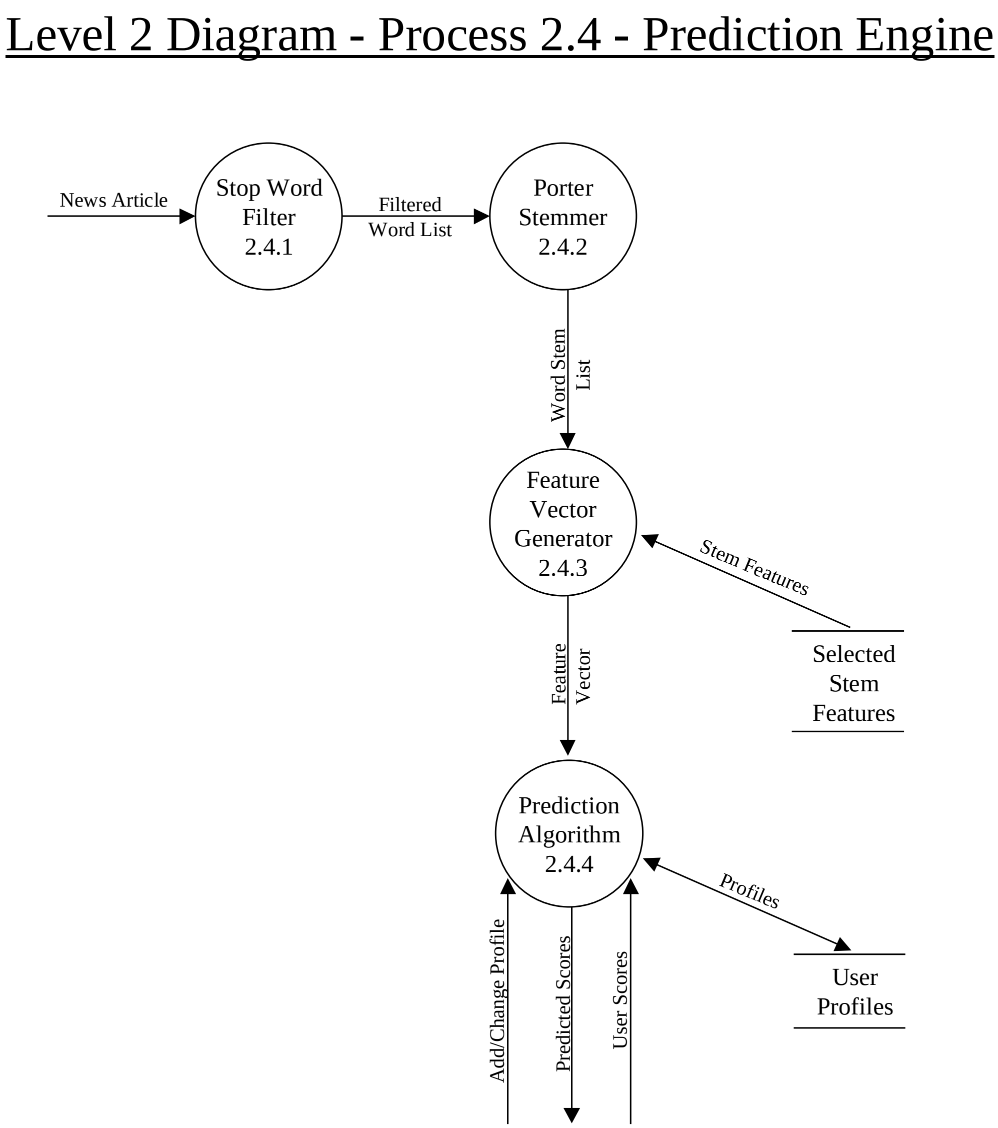


Figure 3.6: Data Flow Diagram - Level 2 Diagram of CNAClient's Prediction Engine

Chapter 4

System Implementation

4.1 Chapter Overview

In this chapter, technical descriptions of the system implementation will be presented, together with some design justifications made during the implementation process, when appropriate. The chapter begins with a discussion of the hardware and software resources chosen for the system, and then the implementation details of each of the three major software components, CNAServer, CNAClient and the Probot script, will be described.

4.2 Resources

Feature Vector Representations of documents are high-dimensional in nature, and thus require significant space and processing resources when prediction algorithms are to be trained and executed. It would be desirable for the client side software to run on a *Portable Digital Assistant*, but the resource constraints on existing PDAs make this infeasible. However, if the intensive processing can be shifted off-PDA, then a portable CNA would be conceivable. In fact, the Probot had already been ported to an iPaq handheld, which is connected via a wireless network card to a PC that performs the intensive processing. The implementation hardware and software platform can therefore be targeted for the standard PC, and execution on the iPaq handheld would be a free bonus.

Java was chosen as the software platform for the CNAClient and CNAServer, primarily for its platform portability. The current PC version of Probot was written for the Linux platform, and would thus require CNAClient to run on Linux as well. However, if the Probot were ported to Windows in the future, using the Java platform means that the CNAClient would be portable without any code modifications. Further, if handheld hardware does reach a stage where intensive processing can be carried out on-board, the CNAClient should be portable to the handheld device with relative ease.

The total network bandwidth required by the CNAClient is quite significant by today's standards: Downloading the entire article cache would require over 10MB of network traffic. However, the average size of pure-text articles is only around 3KB, and since the content delivery bandwidth to the end user is limited to the speed of human speech, only a few articles would be needed before there are more than enough content to keep the user occupied. Therefore the peak network bandwidth required by the CNAClient is not very demanding, and a 56Kbps modem connection would suffice.

The benchmarking of Machine Learning algorithms in Chapter 5 was conducted with a laboratory of 40 Pentium III-888MHz machines at the *School of Computer Science and Engineering at UNSW*. Without running the experiments in parallel, the experimental results would have been unattainable.

The final set of experiments leading to the results in Chapter 5 took three days to complete on the 40 machines, without taking into account the many series of initial testing undergone to refine the statistical reliability of testing procedure.

4.3 CNAServer

4.3.1 CNAServer Overview

The main task of CNAServer is to extract headlines from CNN, and also to cache articles for mass retrieval by the CNAClient. The CNN news source does not provide direct access to their articles database, and therefore it is the responsibility of the CNAServer to constantly monitor CNN's website and extract the latest headlines, as well as indexing them into a local headline database. Further, since CNN does not provide direct access to their articles in pure text form, the CNAServer must extract them from the marked up webpages, and cache them for the CNAClient.

4.3.2 Command Line Arguments

The CNAServer is a *non-persistent* server process, which means is periodically instead of being a permanent server process. The command to execute the CNAServer is as follows:

```
java CNAServer [ProxyURL ProxyPort]
```

ProxyURL and *ProxyPort* are the optional proxy server URL and port number respectively.

Example Usage (No Proxies):

```
java CNAServer
```

Example Usage (With Proxy):

```
java CNAServer www-proxy.cse.unsw.edu.au 3128
```

4.3.3 CNN Headline Extractor

The role of the CNN Headline Extractor is to fetch the latest headlines from CNN. Since CNN do not offer their headlines in a format that can easily be parsed such as XML, the best available alternative is the *CNN Desktop Headlines* web page at the following URL:

```
http://www.cnn.com/desktop/content.html
```

The CNN Desktop Headlines web page contains HTML patterns that allow pure text headlines to be extracted. Documentation of the precise specification of the page format is not publicly available, therefore the parsing process can at best be considered *ad-hoc*. Should CNN decide to alter the page format, a new Headline Extractor will have to be devised.

4.3.4 Headline Database

The headline database maintains data about the latest headlines, including the article link, category and time of publication. Unique IDs will also be assigned to each story so that the CNAClient can avoid presenting the user with duplicate stories. The database is stored a plain text file named "`headlines.db`", where each line in the file is tab-delimited with the data:

```
HeadlineID   LinkURL   Title   ContentProvider   Category   Time
```

The headlines are sorted in chronological order, i.e., the most recent headline will be at the end of the text file. The IDs of the articles are assigned sequentially, so that the latest articles will have the highest numeric IDs.

Since the plain text database will be transferred to the CNAClient in its entirety whenever the client updates the headlines, the file size need to be restricted. It was observed over a month long period that CNN typically publish 70 news articles daily. To maintain the database at a downloadable size, the database was tuned to keep only the 3000 most recent headlines, sufficient to cover a month's worth of CNN headlines. With this upper bound, the raw file size of the database is typically below 500KB, and therefore low bandwidth client connections would be tolerable.

4.3.5 Article Caching

In the initial prototype of the system, excessive delay was observed in the prediction engine of the CNAClient. The delay was attributed to the fact that full text of articles must be fetched in order to execute the prediction engine, and that many articles may be needed before a positive prediction is made. If, for example, 10 articles were filtered as *uninteresting* before encountering an *interesting* one, the client would have had to establish 10+1 TCP connections to the CNN website, and the time accumulated by TCP handshakes and data transfer resulted in unacceptable wait times for the user. For this reason, article caching became a necessary function in ensuring efficient system operation.

As with any network resource, the article links are occasionally incorrect or unreachable when caching is attempted. It is therefore necessary to re-attempt caching in subsequent sessions. However, if there were large numbers of problematic links in the database, the caching mechanism will access the CNN server excessively through regular request of large numbers of broken articles. Such actions may possibly be interpreted as a *Denial-of-Service* attack and may lead to IP banning, and thus the number of articles cached per execution of the server is bounded by a system parameter. This number was tuned so that the rate of caching exceeds the rate of headline publication, with some margins for re-attempting previously unreachable links.

4.3.6 Deployment

The CNAServer was deployed on `wagner.cse.unsw.edu.au` to collect the latest CNN headlines every 15 minutes, meaning the server is executed 4x24 = 48 times a day. The upper bound of caching rate was set to two articles per execution, so that at most 96 articles are cached every day. This exceeds the average headline publication rate of 70 per day, and gives sufficient margin for re-attempting previously unreachable links as required by the caching mechanism.

To serve the headline database and the cached articles to the CNAClient, the files are made available through a web server at the following URLs:

Headline Database:

```
http://www.cse.unsw.edu.au/~johnlai/comp4910/headlines.db
```

Article Cache Directory:

```
http://www.cse.unsw.edu.au/~johnlai/comp4910/articles/
```

4.4 CNAClient

4.4.1 CNAClient Overview

The CNAClient is the most sophisticated component of the system implementing facilities for updating, searching and presenting headlines. Further, a prediction engine is used to filter articles that are unlikely of interest to the user based on historic profile. The required operations are performed through a strict set of input commands and output responses, which was designed to accommodate the Probot front-end.

4.4.2 Command Line Arguments

The command to execute the CNAClient is as follows:

```
java CNAClient [ProxyURL ProxyPort]
```

ProxyURL and *ProxyPort* are the optional proxy server URL and port number respectively.

Example Usage (No Proxies):

```
java CNAClient
```

Example Usage (With Proxy):

```
java CNAClient www-proxy.cse.unsw.edu.au 3128
```

4.4.3 Input/Output Interface

INPUT SPECIFICATIONS

The full specification of input commands are presented in Table 4.1.

Operation	Syntax
Specify Current User	`setuser` *`USERNAME`*
Create New User	`adduser` *`USERNAME`*
Fetch New Headlines	`updateheadlines`
Cycle Through Headlines	`cycleheadlines`
Continue Cycling Headlines	*`blank line`*
Article Keyword Search	`search` *`KEYWORDS`*
Continue Cycling Through Results	*`blank line`*
Fetch Current Article	`fetchcurrent`
Continue Reading Article	*`blank line`*
Stop Reading Article	`back`
Quit the Program	`quit`

Table 4.1: Summary of CNAClient Commands

Notes:

- successful `adduser` *`USERNAME`* operation means that: *`USERNAME`* didn't already exist, a new profile for *`USERNAME`* was created and the current user was set to *`USERNAME`*.

- When the `cycleheadlines` or `search` command is issued, subsequent *blank lines* in the input may follow, meaning the user wishes to skip to the next headline/result.

- When the `fetchcurrent` command is used, subsequent *blank lines* in the input may follow, each meaning that the user wishes to read the next paragraph. When the article is finished or when a `back` command is issued, the program would take the user back to state prior to the execution of the `fetchcurrent` command.

OUTPUT SPECIFICATIONS

For every operation, there will be at least one line of output to indicate the success of the operations:

- `true` - The operation was completed successfully

- `false` - The operation failed. This could mean that the ***`USERNAME`*** doesn't exist, headline database server unreachable, no more articles left or no more content exists, depending on the context.

Then at most one line will follow, which if the Probot interface is used, will be displayed directly to the user. This could be a single headline or a single paragraph of an article, depending on the context. The strict I/O interface is necessary to simplify the integration process with the Probot front end.

4.4.4 CNN Text Extractor

For the purpose of content delivery to the user through a voice interface, the CNN articles need to be in pure-text form. Further, the prediction engine operates only on the text of articles. Unfortunately, like the headlines, the CNN web site does not offer their content in a pure text format. Therefore the CNN Text Extractor was written to process the source HTML of CNN articles to filter out advertisements, photos and menu bars, returning only the content of the article in pure text. This component software is also utilised by the CNAServer's caching mechanism to save processing times on the client end when possible.

4.4.5 CNN Search

A system requirement is to allow keyword search of news articles. The CNN Search utility was written as a component software to access CNN's own web search facility, submitting queries and extracting search results from the HTML source. Again, without the specifications of the input/output format to the CNN web search facility, this utility is based on patterns in the HTML sources of the CNN web search, and may need revision if the existing format is altered.

4.4.6 User Profiles

The system is required to support multiple user profiles, so that the prediction engine can produce tailored results for each user of the system. Each profile stores the IDs of articles already presented to the user, as well as the *score* associated with each presented article, for the purpose of preference prediction. Rather than explicitly asking the user to constantly assign scores to articles, implicit scoring is used: articles that the user has accepted to read are given a score of +1, and articles that the user has rejected are given a score of -1. This frees the user from having to constantly assign explicit scores while reading articles, which can be an annoying distraction.

When the user cycles through headlines, new articles will be passed to the prediction engine in reverse chronological order (most recent first) until a positive prediction is produced. That headline will then be presented to the user, which in turn will be scored and added to the training data, when the user decides to read or skip the article.

4.4.7 Weka

Weka is a collection of Machine Learning algorithms written in Java, and is publicly available under the GNU licence at:

`http://www.cs.waikato.ac.nz/ml/weka/index.html`

The package includes implementations of popular classifiers such as the *Support Vector Machine*, *Naïve Bayesian Classifier* and *k-Nearest Neighbour*, and will were benchmarked rigorously to find the best classifier for predicting interesting articles. The experimental process is described in detail in Chapter 5.

4.5 Probot

The Probot scripting agent is used as a front end to CNAClient, so that user interaction can be made more informal through Natural Language dialogue, instead of the rigid command syntax required by CNAClient.

The Probot Script will use four main contexts to model user conversations:

- IDLE - waiting for user to initiate system.
- IDENT - Identify new/existing user to create/load user profile.
- FIND - Find articles of interest to user. Has 2 sub-contexts, one of which allows headlines to be cycled through, while the other allows keyword searches to be performed.
- READ - Read article to user, prompting frequently to check whether the article should continue to be read.

The Probot context transition diagram, used to implement the Probot scripts, is presented in Figure 4.1.

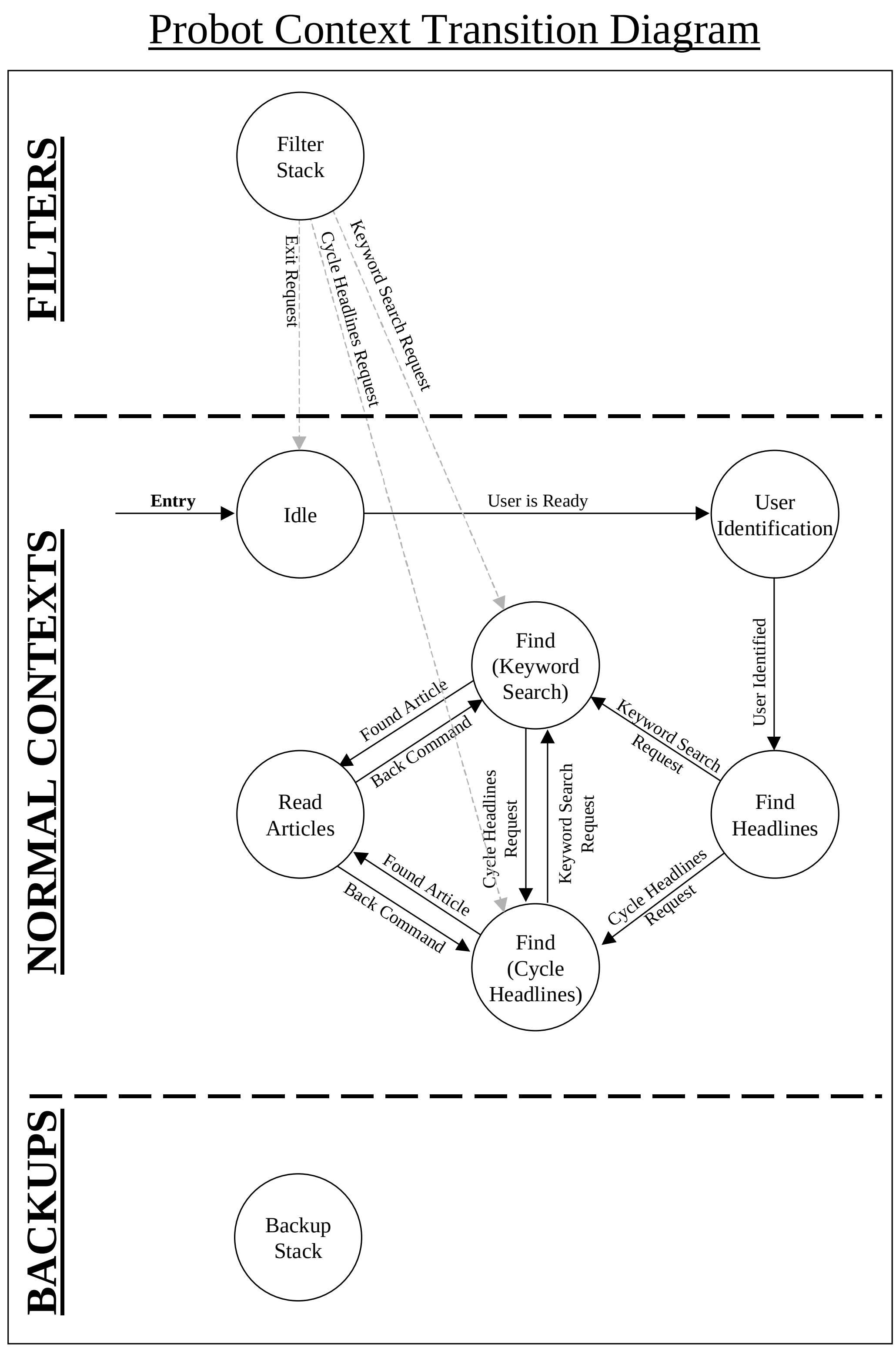


Figure 4.1: Probot Context Transition Diagram

4.6 Java CNA Source Structure

This section lists the files in the Java source directories of CNAClient and CNAServer, along with brief descriptions of their functionalities. This section would be a useful guide to navigating the source code attached to this report.

Common CNA Files

These files/directories are common to both the CNAServer and Client:

- `articles/`

  Articles cache directory

- `CNNTextExtractor.java`

  Used to extract text content from CNN articles

- `Headline.java`

  Represents a headline, including title, URL and date etc.

- `HTMLConverter.java`

  Converts an HTML marked-up document to text

- `headlines.db`

  Headline database file

- `Proxy.java`

  Allows HTTP proxy to be set conveniently

CNAServer Files

These files/directories are specific to the CNAServer:

- `CNAServer.java`

  The main code for the CNAServer

- `CNNHeadlineExtractor.java`

  Used to extract latest headlines from CNN.com

- `nextID.txt`

  Used to keep track of assigned unique IDs

CNAClient Files

These files/directories are specific to the CNAClient:

- `CNAClient.java`

  The main code for the CNAClient

- `headlineServer.url`

  Configuration file storing the URL of the CNAServer's headline database

- `cacheServer.url`

  Configuration file storing the URL of the article cache server

- `stemfeatures.txt`

  List of selected word stems to be used as Feature Vector Dimensions.

- `commonWords.txt`

  List of common words to be filtered by CommonWordsFilter class

- `ArticleFetcher.java`

  Used to cache articles in the background

- `CNNSearch.java`

  An interface to CNN.com's search engine

- `profiles/`

  Directory storing user profiles

- `CommonWordsFilter.java`

  Used to convert sentences into word lists, while common words are filtered

- `Stemmer.java`

  Used to convert words to their stems

- `FeatureExtractor.java`

  Used to count the word frequencies in a given article

- `UserProfile.java`

  Abstract class defining the behaviour of a user profile

- `UserProfileWeka.java`

  Instance of UserProfile using Weka package as the prediction engine.

- `weka.jar`

  JAR archive of Weka package, which is used for prediction and experiments

Experimental Files

These files are used in the experiments deriving classifier benchmarks and selecting feature stems, and are not used in the main system:

- `ARFFGenerator.java`
  Used to generate ARFF files from CNN articles and user scores
- `MassStemmer.java`
  Used to collect all word stems in a set of articles

Miscellaneous Files

- `CoTrainer.java`
  Implements the Co-Training algorithm proposed by [KM01], but the benefits of this algorithm have yet to be rigorously verified, and thus not used in the main system.

Chapter 5

Experiments

5.1 Chapter Overview

The aim of this section is to find the optimal classifier for use in the system through rigorous statistical testing, and to verify various claims from the Text Classification literature. More specifically, the following questions are to be addressed:

- Which classifiers benefit from TFIDF vector scaling?
- Which of the Feature Selection Algorithms available in *Weka* allow classifiers to perform optimally? The algorithms are $\{\chi^2$, Document Frequency (DF), Gain Ratio, Information Gain (IG), ZeroR$\}$.
- What is the best size configuration for each classifier?

A set of labelled data is required for use in the cross-validation experiments. Although the Reuter's corpus [1] is often used as the *de facto* standard data set for IR experiments, it would be more appropriate to use news articles from CNN.com, where the system actually sources its news. The author of this thesis collected 1710 CNN articles, and manually labelled them as either `INTERESTING` or `UNINTERESTING` according to his personal preferences. From the text of those articles, 25769 unique word stems were extracted. Those occurring in only one document were filtered, producing 14968 word stems as the full Feature Stem set.

In any individual test, several variables need to be controlled:

- The classifier being used for prediction. (7 possibilities from 6 different classifiers. The kNN algorithm was tested for kNN-1 and kNN-3 configurations.)
- The Feature Selection algorithm used to determine the Feature Dimensions. (5 possibilities from the Weka package)
- The Feature Dimension size (10 chosen sizes: 32, 64, 96, 128, 256, 384, 512, 1024, 2048 and 4096)
- TFIDF scaling or standard normalisation of Document Vectors. (2 possibilities)

[1] There are many collections of Reuters newswire documents with human assigned category labels that are frequently used for testing in the IR literature. One such collection, the Reuters-21578, is available at `http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html`

Each combination of the four variables would result in a different experimental configuration. Ideally, each of the $7 \times 5 \times 10 \times 2 = 700$ configurations would have ten rounds of ten-fold cross-validation conducted, but the time required conduct the experiments would be intractable by projective estimates. Therefore, the three questions posed would have to be addressed sequentially.

5.2 TFIDF

Which classifiers would benefit from TFIDF?

The goal of this test is to determine whether TFIDF scaling of document Feature Vectors improve performance. Ten-fold cross-validation was performed for each classifier under various feature selection algorithms and feature sizes, with and without TFIDF. This means that each classifier would have had $5 \times 10 = 50$ ten-fold cross-validations conducted with TFIDF, and then another 50 without TFIDF. The results are tabulated in Table 5.1, together with P-Values from Paired-Sample t-tests to determine whether TFIDF has significantly affected the performance.

Contrary to expectations, TFIDF scaling produced better results than without TFIDF with only one classifier, the OneR. In fact, TFIDF has led to significantly worse results in three of the remaining six classifiers. Therefore TFIDF will only be used with OneR in future tests.

Classifier	No TFIDF (Mean Acc. and F-1)	With TFIDF (Mean Acc. and F-1)	t-test (P-Value)	Conclusion
J48	64.9%	63.8%	0.000	$\gg$
N. Bayes	66.9%	65.2%	0.000	$\gg$
OneR	49.9%	50.3%	0.002	$\ll$
SMO	67.5%	66.4%	0.000	$\gg$
ZeroR	34.1%	34.1%	1.000	˜
kNN1	65.6%	65.8%	0.222	˜
kNN3	61.4%	61.3%	0.541	˜

Table 5.1: Performance Enhancements with TFIDF Vector Scaling
$\gg$ means TFIDF performed significantly *worse* ($P \leq 0.01$)
$\ll$ means TFIDF performed significantly *better* ($P \leq 0.01$)
˜means TFIDF showed *insignificant* performance difference ($P > 0.01$)

5.3 Stem Feature and Size Selection

[YP97] showed that by applying effective Feature Selection algorithms, the Stem Feature Dimensions can be vastly reduced to improve speed performance, and surprisingly, prediction accuracy as well. To decide on the most suitable Feature Selection algorithm and Feature Size for each classifier, two series of tests were conducted: the first to distinguish the best feature selection algorithms, and the second to find optimal feature sizes.

5.3.1 Feature Selection Algorithm

> *Which feature selection algorithms allow classifiers to perform best?*

Ten-fold cross-validation tests were performed for every classifier under every feature selection algorithm, with feature sizes of 32, 64, 96, 128, 256, 384, 512, 1024, 2048 and 4096, with and without TFIDF. Each derived accuracy and

F_1-Measure pair was averaged to derive a single metric, and Paired-Sample t-tests were then used to compare the performance difference between every pair of Feature Selection algorithms. The results are summarised in Table 5.2 and Table 5.3.

	χ^2	DF	GainRatio	IG	OneR
χ^2	1.0000	0.0000	0.0000	0.9785	0.5239
DF	0.0000	1.0000	0.4829	0.0000	0.0002
GainRatio	0.0000	0.4829	1.0000	0.0001	0.0000
IG	0.9785	0.0000	0.0001	1.0000	0.5850
OneR	0.5239	0.0002	0.0000	0.5850	1.0000

Table 5.2: P-Values for Paired-Sample t-test of Feature Selection Algorithms

	χ^2	DF	GainRatio	IG	OneR
χ^2	˜	$\gg$	$\gg$	˜	˜
DF	$\ll$	˜	˜	$\ll$	$\ll$
GainRatio	$\ll$	˜	˜	$\ll$	$\ll$
IG	˜	$\gg$	$\gg$	˜	˜
OneR	˜	$\gg$	$\gg$	˜	˜

Table 5.3: Comparison of Feature Selection Algorithms using Paired-Sample t-test

$\ll$ means the algorithm in this row was significantly *worse* than the one in this column ($P \leq 0.01$)
$\gg$ means the algorithm in this row was significantly *better* than the one in this column ($P \leq 0.01$)

˜means the algorithm in this row was *indifferent* to the one in this column ($P > 0.01$)

This can be summarised as

$$\{\chi^2, \text{IG}, \text{OneR}\} \gg \{\text{DF}, \text{GainRatio}\}$$

The result of those feature selection algorithms tested by [YP97] agrees with the results presented here, except that the Document Frequency (DF) method performed worse than χ^2 and IG. DF assumes that rare words are not useful in Text Classification, which conflicts with many popular beliefs in the literature, and is often regarded as an *ad-hoc* approach. However, [YP97] presented empirical results to suggest that DF has strong correlations with IG and χ^2 under kNN and the author's own *Linear Least Squares Fit algorithm*. The statistical results presented here, based on more classifiers and more rigorous statistical testing (Paired-Sample t-test on ten-fold cross validated results) would disagree with its claim, and has confirmed that DF performs worse than IG and χ^2.

5.3.2 Optimal Feature Size

What is the best size configuration for each classifier?

The first test showed that χ^2, IG and OneR were the best selection algorithms, and that their performance differences were insignificant across classifiers. The second part of the feature selection test seeks to determine the optimal feature size for each classifier. The desirable Feature Dimension size should be as small as possible to maximise system speed performance, while maintaining almost optimal prediction capability.

For each classifier under each size configuration, ten-fold cross-validated results under each of the best statistically indistinguishable selection algorithms (χ^2, IG and OneR) were averaged. From the averages, the optimal size configuration was determined. Paired-Sample t-tests were then used test the significance of the difference between the optimal configuration and each size configuration, so that the feature size can be minimised to improve runtime performance, in situations where filtering performance will not degrade significantly. Paired-Sample t-tests were used to test the significance of performance differences, and the P-Values are presented in Table 5.4.

In each column, the entry containing 1.00 indicate the optimal stem feature size for the classifier in question. For example, 2048 is the optimal feature size for the J4.8 classifier. Each P-Value entry represents the probability of the given classifier performing equivalent to the optimal configuration under the given size configuration. For example, the P-Value of the J4.8 classifier with stem size of 1024 is 0.84. This means that there is a high probability of 0.84 that the classifier performs just as well with 1024 stem features as the optimal configuration with 2048 features. On the other hand, J4.8 with only 16 features has a P-Value of 0.02, suggesting very significant performance degradation.

Features	J4.8	N. Bayes	OneR	SMO	ZeroR	kNN1	kNN3
16	0.02	0.01	*1.00*	0.03	*1.00*	0.07	0.40
32	0.16	0.05	0.27	0.07	*1.00*	0.29	0.43
64	0.13	0.06	0.18	0.02	*1.00*	0.14	0.99
96	0.31	0.03	0.18	0.04	*1.00*	0.46	0.88
128	0.28	0.00	0.18	0.02	*1.00*	0.28	0.98
256	0.51	0.36	0.18	0.05	*1.00*	0.54	1.00
384	0.47	0.27	0.18	0.03	*1.00*	*1.00*	0.02
512	0.45	0.14	0.18	0.00	*1.00*	0.44	0.04
1024	0.84	*1.00*	0.18	0.29	*1.00*	0.17	0.11
2048	*1.00*	0.32	0.18	*1.00*	*1.00*	0.01	0.08
4096	0.74	0.24	0.13	0.13	*1.00*	0.01	0.01

Table 5.4: P-Values of Paired-Sample t-tests Between Optimal- and Various Sized Configurations

Although the three best feature selection algorithms were insignificant in their performance differences, one algorithm must be ultimately chosen for each classifier. Hence, for each classifier under its optimal feature size, the selection algorithm with the highest absolute performance was chosen. The resulting optimal combinations of TFIDF, feature selection algorithm and feature size is presented in Table 5.5. This will allow comparisons to be made between each classifier under its respective optimal configuration in the next section.

Classifier	TFIDF	Feature Selection	Feature Size
J4.8	No	IG	2048
N. Bayes	No	χ^2	1024
OneR	Yes	IG	16
SMO	No	IG	2048
ZeroR	No	Any	16
kNN1	No	χ^2	384
kNN3	No	IG	256

Table 5.5: Optimal Configurations for Each Classifier

5.4 Classifier Benchmark

Having tuned each classifier for optimal performance, ten rounds of ten-fold cross-validation were conducted to benchmark their performances. The summary of result is presented first in Table 5.6.

Classifier	Accuracy	(Std. Dev.)	F-1	(Std. Dev.)	Avg. Acc. & F-1
SMO	85.0%	(0.4%)	75.0%	(0.5%)	80.0%
N. Bayes	77.7%	(0.2%)	70.8%	(0.2%)	74.2%
J4.8	75.8%	(0.5%)	61.6%	(0.8%)	68.7%
kNN1	76.5%	(0.4%)	60.3%	(0.9%)	68.4%
kNN3	76.3%	(0.6%)	58.8%	(1.2%)	67.6%
OneR	68.7%	(0.4%)	59.5%	(1.3%)	64.1%
ZeroR	68.2%	(0.0%)	0.0%	(0.0%)	34.1%

Table 5.6: Classifier Benchmark Over 10 Runs of Ten-Fold Cross-Validation.

Based on Accuracy in Table 5.7, the classifiers are ranked as follows:

$$\text{SVM} \gg \text{Naïve Bayes} \gg \{\text{kNN-1, kNN-3, J4.8}\} \gg \text{OneR} \gg \text{ZeroR}$$

Classifier	SMO	N. Bayes	kNN1	kNN3	J4.8	OneR	ZeroR
SMO	˜	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
N. Bayes	$\ll$	˜	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
kNN1	$\ll$	$\ll$	˜	˜	$\gg$	$\gg$	$\gg$
kNN3	$\ll$	$\ll$	˜	˜	˜	$\gg$	$\gg$
J4.8	$\ll$	$\ll$	$\ll$	˜	˜	$\gg$	$\gg$
OneR	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	˜	$\gg$
ZeroR	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	˜

Table 5.7: Classifier Accuracy Comparison Using Paired-Sample t-test
$\ll$ means the classifier in this row was significantly *worse* than the one in this column ($P \leq 0.01$)
$\gg$ means the classifier in this row was significantly *better* than the one in this column ($P \leq 0.01$)
˜means the classifier in this row was *indifferent* to the one in this column ($P > 0.01$)

Classifier	SMO	N. Bayes	J4.8	kNN1	OneR	kNN3	ZeroR
SMO	˜	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
N. Bayes	$\ll$	˜	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
J4.8	$\ll$	$\ll$	˜	˜	$\gg$	$\gg$	$\gg$
kNN1	$\ll$	$\ll$	˜	˜	˜	$\gg$	$\gg$
OneR	$\ll$	$\ll$	$\ll$	˜	˜	˜	$\gg$
kNN3	$\ll$	$\ll$	$\ll$	$\ll$	˜	˜	$\gg$
ZeroR	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	˜

Table 5.8: Classifier F_1-Measure Comparison Using Paired-Sample t-test
$\ll$ means the classifier in this row was significantly *worse* than the one in this column ($P \leq 0.01$)
$\gg$ means the classifier in this row was significantly *better* than the one in this column ($P \leq 0.01$)
˜means the classifier in this row was *indifferent* to the one in this column ($P > 0.01$)

Based on F_1-Measures in Table 5.8, the classifiers are ranked as follows:

$$\text{SVM} \gg \text{Naïve Bayes} \gg \{\text{J4.8, kNN-1, OneR, kNN-3}\} \gg \text{ZeroR}$$

The results showed that SMO (SVM) performed significantly better than every other classifier, producing 85% accuracy and 75% F_1-Measure, and will therefore be chosen as the prediction algorithm for the system.

The empirical data from this experiment disagrees with those presented in [YL99], where kNN was a top performer along with SVM, both of which consistently outperformed Naïve Bayes, the worst performer in every test conducted.

Chapter 6

Summation

6.1 Conclusion

The design and implementation of the *Conversational News Agent* had been presented through this thesis report.

The end system has demonstrated that useful applications with *Natural Language* interfaces can be developed effectively, using shallow parsing techniques and scripted responses, without using complex *Natural Language Processing* engines that often fail to deliver the performance expected of *Conversational Agents*.

The *Vector Space Representation* of documents have been used to allow Machine Learning algorithms to learn user interests in the AI field of *Text Categorisation*, and rigorous statistical testing have been applied to verify several claims in the literature.

The TFIDF vector scaling 'tweak' is often assumed in the literature to offer prediction performance improvements, but experimental results have suggested otherwise. In fact, TFIDF have demonstrated insignificant performance improvements, if not worse, in all but one classifier (OneR) tested.

Feature Selection algorithms allow aggressive speed improvements for Machine Learning algorithms, and several algorithms were benchmarked. Empirical results are in agreement with those of [YP97], except for one finding: The Document Frequency (DF) method was confirmed as an inferior *ad-hoc* method, as it had performed significantly worse than every other algorithm tested. [YP97] also acknowledged the *ad-hoc* appearance of the DF method, but its results have suggested that DF performed just as well as the best performing methods. The empirical results from this thesis disagree with this claim, and would rank the general performance of feature selection methods as follows:

$$\{\chi^2, \text{Information Gain, OneR}\} \gg \{\text{DF, Gain Ratio}\}$$

Several *Machine Learning* algorithms were tuned to their optimal configurations on the training data, and up to 75% F_1-Measure and 85% accuracy had been demonstrated by the best performing classifier - the Support Vector Machine. This is in agreement with the literature's informal recognition of the SVM as the current state-of-the-art classifier.

From the classifiers benchmarks in this thesis, two sets of ranking were derived. By the less commonly used *accuracy* performance measure, the classifiers as ranked in the following order:

$$\text{SVM} \gg \text{Naïve Bayes} \gg \{\text{kNN-1, kNN-3, J4.8}\} \gg \text{OneR} \gg \text{ZeroR}$$

By the more commonly used F_1-Measure in the IR literature, the classifiers are ranked in the following order:

$$\text{SVM} \gg \text{Naïve Bayes} \gg \{\text{J4.8, kNN-1, OneR, kNN-3}\} \gg \text{ZeroR}$$

In both rankings, the SVM has outperformed every other classifier.

Having integrated the current state-of-the-art Machine Learning algorithm with a Natural Language interface for customised news delivery, the next section identifies some research that could lead on from this thesis, in the never-ending quest for better prediction techniques and natural language interfaces.

6.2 Future Work

News providers world-wide have embraced the Internet as a news delivery medium in recent years, and despite the fact that most large players have well established web presences, revolutionary methods of tailored news services remain to be found.

Google has recently launched its news service, which scouts 4000+ worldwide news sources to algorithmically compile a collection of the most relevant articles. CNN itself has launched a *CNN Newswatch* service, where subscribers use an exclusive software application to monitor topics of their interest, much like the user preference prediction function provided by this thesis project. Customised news is therefore an active area with enormous research and commercial interest.

Research following on from this thesis could investigate the following suggestions, both to advance research in this area as well as to the CNA software of this thesis:

- [KM01] recently proposed the Co-Training algorithm, which could significantly reduce the number of labelled samples needed in order to train classifiers to their peak performance. The algorithm is simple: Use the minimal initial training data to train the classifier, then use the classifier to label other unlabelled data, and add the most confident ones to the training set. Facility for co-training had already been implemented in the current CNA code, and preliminary (but inconclusive) results have shown large improvements under certain conditions. For example, for both the SVM and Naive Bayes classifiers with only

17 initial training data, 35% improvements in accuracy have been seen through co-training. Rigorous statistical testing of the Co-Training algorithm remain to be conducted, and the results would be highly useful if proven effective.

- The current Probot script for the CNA can be enhanced to exhibit more semantic and pragmatic behaviour. For example, if the user is reading an article about "*Republicans campaigning for the Senate*", it would be desirable to let the user ask for specific articles relating to the current article, such as asking "*Is Bush campaigning with* them*?*" Traditionally, this would require complex semantic and pragmatic processing, but the same desired effect could possibly be achieved with simple yet clever Probot scripts.

- With the voice-mode interaction mode of the current Probot, the user would have to wait for the script to finish its sentence before issuing another command. If the user decides on a new course of action half way through a Probot *monologue*, the user cannot *pre-empt* the monologue to issue the new command. The end-user experience could be improved drastically if the pre-emption capabilities were implemented. Having pre-emption also means that Probot can read through articles without having to prompt the user every paragraph, asking whether or not to continue reading.

- *Latent Semantic Analysis/Indexing* is relatively new to Information Retrieval, using Linear Algebra theory (*Singular Value Decomposition*) to extract and represent documents. LSA seeks to address the shortcoming of current Machine Learning approaches where dimensions in Feature Vectors represent literal words, rather than concepts. Although stemming algorithms can trim word suffixes, concepts with several literal representations (synonyms) would not be captured in the same Feature Dimensions. Also, literal words often have multiple meanings (polysemy), and current Feature Vectors would likely lead to retrieval of irrelevant data. LSA attempts to overcome these problems by statistically deriving relationships between words and concepts, rather than making comparisons based on individual words themselves. Early works have shown promising results in automatic assessment of student essays, retrieval across different language (English and French) as well as Information Retrieval. [BDO94]

Appendix A

Sample User Session

Below is a sample session that demonstrates the actual behaviour of the end system, assuming, of course, that the end user is being cooperative. User responses are styled ***this way***.

> Let me know when you like to hear some news!
>
> ***alrighty let's go.***
>
> Hi there, have we met before?
>
> ***no we haven't.***
>
> What user name you would like?
>
> ***can I have phantom?***
>
> That user name is already taken... Let's try another one.
>
> ***how about john?***
>
> OK, let's catch up on some news! Would you like to go through the headlines?
>
> ***ok what's new?***
>
> Oct 23, 2002 3:30 PM - Military group urges rebellion against Chavez

Read it?

nope.

Oct 23, 2002 12:15 PM - N. Korea agrees to nuclear dialogue

Read it?

sure.

From CNN Seoul Bureau Chief Sohn Jie-Ae

Keep going?

yeah.

SEOUL, South Korea (CNN) – South and North Korean negotiators have agreed to cooperate aggressively to resolve all bilateral issues, including the nuclear issue, through dialogue. More?

ok.

The agreement came after four days of talks in Pyongyang that had been scheduled before North Korea's admission earlier this month

that it had a secret nuclear weapons program.

More?

more.

"In order to guarantee peace and stability on the Korean Peninsula, the South and North will actively cooperate in resolving all the issues, including the nuclear issue, through dialogue," said the agreement, which was announced early Wednesday and reported in pool accounts by South Korean journalists.

More?

no that's enough.

Oct 23, 2002 12:00 PM - Victim's mom, killer's friend helped nab Ohio sniper

Read it?

is there any news about the Bali bombing?

22-Oct-02 - Nightclub terror plans revealed

Read it?

yes.

From CNN Jakarta Bureau Chief Maria Ressa

Keep going?

ok.

JAKARTA, Indonesia (CNN) – An al Qaeda operative now in U.S. custody has told the FBI about plans to attack popular bars and nightclubs in Southeast Asia, according to classified documents.

More?

yeah.

His name is Mohammed Mansour Jabarah, an al Qaeda operative arrested

in Oman last March.

More?

do you have anything on the new Hannibal movie?

13-Oct-02 - 'Red Dragon' holds at No. 1

Read it?

yeah that's it.

LOS ANGELES, California (AP) – Hannibal Lecter stood out in a crowd of new movies.

Keep going?

yeah.

"Red Dragon," the third thriller starring Anthony Hopkins as serial killer Lecter, was the No. 1 movie for a second weekend, taking in $17.6 million and fending off an onslaught of debut films, according to studio estimates Sunday.

More?

ok.

Reese Witherspoon's romantic comedy "Sweet Home Alabama" remained in second place with $14.1 million.

More?

nah.

25-Jul-02 - Review: Myers, cameos make 'Goldmember' groovy

Read it?

yeah groovy!

By Lisa SchwarzbaumEntertainment Weekly

Keep going?

yea.

(Entertainment Weekly) – Anything short of a billion gajillion dollars isn't pay enough to spoil the fun by describing the opening of "Austin Powers in Goldmember." Well, maybe for a million – okay, for 10,000 simoleons. More?

no, I've gotta go now, bye!

See you later! Let me know when you like to hear some news!

Bibliography

[AFS01] James F. Allen, George Ferguson, and Amanda Stent. An architecture for more realistic conversational systems. In *Intelligent User Interfaces*, pages 1–8, 2001.

[AKB] Kartik Agaram, Stephen W. Keckler, and Doug Burger. A characterization of speech recognition on modern computer systems.

[Alv02] Sergio A. Alvarez. An exact analytical relation among recall, precision, and classification accuracy in information retrieval. 2002.

[BDO94] Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. Using linear algebra for intelligent information retrieval. Technical Report UT-CS-94-270, 1994.

[BHK98] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. pages 43–52, 1998.

[CBB+98] J. Cassell, T. Bickmore, M. Billinghurst, L. Campbell, K. Chang, H. Vilhjlmsson, and H. Yan. An architecture for embodied conversational characters, 1998.

[CBB+99] Justine Cassell, Timothy W. Bickmore, Mark Billinghurst, L. Campbell, K. Chang, Hannes Hogni Vilhjalmsson, and H. Yan. Embodiment in conversational interfaces: Rea. In *CHI*, pages 520–527, 1999.

[Cha00] Chakrabarti. Data mining for hypertext: A tutorial survey. *SIGKDD: SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery and Data Mining, ACM*, 1, 2000.

[Coh95] Paul R. Cohen. *Empirical methods for artificial intelligence*. The MIT Press, 1995.

[EMM96] B. Eric, I. Mani, and T. MacMillan. Representational issues in machine learning of user profiles, 1996.

[Fla98] Sharon Flank. A layered approach to NLP-based information retrieval. In Christian Boitet and Pete Whitelock, editors, *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics*, pages 397–403, San Francisco, California, 1998. Morgan Kaufmann Publishers.

[Gla99] James Glass. Challenges for spoken dialogue systems, 1999.

[Gok99] Anuja Gokhale. Improvements to collaborative filtering algorithms, 1999.

[GSM] Norbert Gerfelder, Ulrike Spierling, and Wolfgang Mller. Novel user interface technologies and conversational user interfaces for information appliances.

[HKW] Max Hfferer, Bernd Knaus, and Werner Winiwarter. Cognitive filtering of information by evolutionary algorithms.

[HL99] Wen-Lin Hsu and Sheau-Dong Lang. Classification algorithms for NETNEWS articles. In *Proceedings of CIKM-99, 8th ACM International Conference on Information and Knowledge Management*, pages 114–121, Kansas City, US, 1999. ACM Press, New York, US.

[HSD00] P. Husbands, H. Simon, and C. Ding. the use of singular value decomposition for text retrieval, 2000.

[Joa97] Thorsten Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In Douglas H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 143–151, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[Joa98] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[JPK98] A. Joshi, C. K. Punyapu, and P. Karnam. Personalization asynchronicity to support mobile web access. In *Workshop on Web Information and Data Management*, pages 0–, 1998.

[KM01] Svetlana Kiritchenko and Stan Matwin. Email classification with co-training. 2001.

[Lan95] Ken Lang. NewsWeeder: learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pages 331–339. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1995.

[LG94] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In W. Bruce Croft and Cornelis J. van Rijsbergen, editors, *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval*, pages 3–12, Dublin, IE, 1994. Springer Verlag, Heidelberg, DE.

[LR94] David D. Lewis and Marc Ringuette. A comparison of two learning algorithms for text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 81–93, Las Vegas, US, 1994.

[MG87] T. W. Maone and Grant. Intelligent information-sharing systems. 30(5):390–402, May 1987.

[MR95] Kathleen R. McKeown and Dragomir R. Radev. Generating summaries of multiple news articles. In *Proceedings, 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 74–82, Seattle, Washington, 1995.

[NT97] G. Nakhaeizadeh and C.C. Taylor, editors. *Machine learning and statistics: the interface*. John Wiley and Sons, 1997.

[Ore] Nir Oren. Improving the effectiveness of information retrieval with genetic programming.

[Por80] Martin F. Porter. *An Algorithm for Suffix Stripping Program*. 1980.

[PS96] A. Pannu and K. Sycara. A learning personal agent for text filtering and notification, 1996. Submitted to AAAI 96.

[PSA98] D. Perzanowski, A. Schultz, and W. Adams. Integrating natural language and gesture in a robotics domain, 1998.

[Qui93] John Ross Quinlan. *Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[RIS+94] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, 1994. ACM.

[Sal83] G. Salton. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[Sal88] Gerard Salton. *Automatic Text Processing*. Addison-Wesley Publishing Company, 1988.

[Sam01] Claude Sammut. Managing context in a conversational agent. 2001.

[SC93] Tomek Strzalkowski and Jose Perez Carballo. Recent developments in natural language text retrieval. In *Text REtrieval Conference*, pages 123–136, 1993.

[She94] Beerud D. Sheth. A learning approach to personalized information filtering. Master's thesis, January 1994.

[SM98] Sam Scott and Stan Matwin. Text classification using WordNet hypernyms. In Sanda Harabagiu, editor, *Use of WordNet in Natural Language Processing Systems: Proceedings of the Conference*, pages 38–44. Association for Computational Linguistics, Somerset, New Jersey, 1998.

[SM99] Sam Scott and Stan Matwin. Feature engineering for text classification. In Ivan Bratko and Saso Dzeroski, editors, *Proceedings of ICML-99, 16th International Conference on Machine Learning*, pages 379–388, Bled, SL, 1999. Morgan Kaufmann Publishers, San Francisco, US.

[SS98] A. Smola and B. Scholkopf. A tutorial on support vector regression, 1998.

[SW94] E. Schweighofer and W. Winiwarter. Intelligent information retrieval: Konterm - automatic representation of context related terms within a knowledge base for a legal expert system, 1994.

[Tec98] Gheorghe Tecuci. *Building Intelligent Agents: An Apprenticeship Multistrategy learning theory, methodology, tool and case studies.* Academic Press, 1998.

[THA+97] Loren Terveen, Will Hill, Brian Amento, David McDonald, and Josh Creter. PHOAKS: A system for sharing recommendations. *Communications of the ACM*, 40(3):59–62, 1997.

[Tip00] M. Tipping. The relevance vector machine, 2000.

[Ucl] Daniel Greening Ucla. Experiences with cooperative moderation of a usenet newsgroup.

[UF98] L. Ungar and D. Foster. Clustering methods for collaborative filtering, 1998.

[Vap95] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[VR79] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.

[Yan99] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1/2):69–90, 1999.

[YCB+99] Y. Yang, J. Carbonell, R. Brown, T. Pierce, B. Archibald, and X. Liu. Learning approaches for detecting and tracking news events, 1999.

[YL99] Y. Yang and X. Liu. A re-examination of text categorization methods. In *22nd Annual International SIGIR*, pages 42–49, Berkley, August 1999.

[YP97] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In Douglas H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 412–420, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.