The Use of Syntax in Word Completion Utilities

by

Afsaneh Fazly

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

The Use of Syntax in Word Completion Utilities

Afsaneh Fazly

Master of Science

Graduate Department of Computer Science

University of Toronto

2002

Current word-prediction utilities rely on little more than word unigram and bigram frequencies. Can part-of-speech information help? To answer this question, we first built a testbench for word prediction; then introduced several new prediction algorithms which exploit part-of-speech tag information. We trained the prediction algorithms using a very large corpus of English, and in several experiments evaluated them according to several performance measures. All the algorithms were compared with WordQ, a commercial word-prediction program. Our results confirm that strong word unigram and bigram models, collected from a very large corpus, give accurate predictions. All predictors, including that based on word unigram statistics, outperform the WordQ prediction algorithm. The predictor based on word bigrams works surprisingly well compared to the syntactic predictors. Although two of the syntactic predictors work slightly better than the bigram predictor, the ANOVA test shows that the difference is not statistically significant.

Dedication

To my parents,

my brother and sisters, Amir, Farzaneh, and Azadeh,

and my husband, Reza

For their true love and support.

Acknowledgements

I would like to extend my gratitude to several people whom without their help and support, accomplishment of this thesis would not have been possible.

First and foremost, my supervisor, Graeme Hirst, for providing great insights, detailed comments and tireless support through the development of this thesis. I also would like to give my special thanks to him for his availabiliy for discussions all the time. I am also very grateful to my second reader, Fraser Shein, for his help and his useful comments.

I would like to extend my appreciation to Suzanne Stevenson for her support and for the enlightening conversations she shared with me. I also want to thank Gerald Penn and Sam Roweis for their great help.

My special thanks goes to Tom Nantais, who has always been ready to answer my questions about WordQ, and has helped me with porting WordQ prediction module from Windows to Linux.

I also want to thank my friends in Computer Science Department, and especially those in Computational Linguistics group for their time, support and friendship during the past two years. Melanie Baljko, Alex Budanitsky, Bowen Hui, Diana Inkpen (Ziau), and Vivian Tsang for answering my questions and being always ready to help.

I am grateful for the financial support to the University of Toronto, and Communications and Information Technology Ontario (CITO).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Many people with learning, physical or cognitive disabilities have to cope with the physical and psychological problems imposed upon them. One of the problems that these people may have to overcome is their inability to communicate normally. The human's ability to communicate allows them to exchange ideas and concepts with each other, and the inability to communicate can lead people to several problems.

Therefore, computer-based technologies, also called Assistive Technology (AT), have become an essential support for many people with disabilities. Assistive technology refers to the use of technology to assist individuals with physical or cognitive disabilities to improve their abilities in performing functions that may be difficult or even impossible for them without help. Assistive technology consists of a broad range of technologies, such as wheelchairs, augmentative communication devices and alternative computer access hardware and software. Each individual needs a specific type of technology, depending on their impairments, and in order to make best use of the potential benefits of each assistive device, some products exploit *adaptive interfaces* to meet the specific needs of their users.

Communication is one of the activities that can be assisted using these technologies. People who have difficulty in movement or speaking may use computers as their writing or communicating tools. However, many people with severe speech and motor impairments face difficulties when using computer interfaces to communicate. For many of them, it is difficult to use the standard keyboards or any other computer input devices, due to physical, cognitive, or learning disabilities, and hence the rate of sentence composition can be very low.

Augmentative and Alternative Communication (AAC) systems are alternatives for such people. They attempt to facilitate the communication of the people with disabilities by increasing the communication rate and decreasing the effort. They help to enhance an individual's communication abilities, and may include an integrated group of strategies and techniques, such as symbols and picture boards, pencil and paper, and electronic communication devices.

In Figure 1.1 an example of an AAC device, called Gemini, is shown. In the picture, a handicapped user is using an input device specially designed to fit the needs of a user with disabilities. Gemini is a full-featured Macintosh computer, developed by Gemini Assistive Technology, Inc. It is a dedicated device to help people with learning, communication, or computer access difficulties.

Word prediction, the task of predicting words which are likely to follow a given segment of text, is a technique commonly used in AAC systems, which uses the inherent redundancy in natural language to enhance the sentence composition rate. Word-prediction systems have been in use as writing aids since the early 1980's. They were originally used by people with physical disabilities in order to reduce the amount of effort needed to enter a text, but later they were found to be also useful for people with learning or language impairments, such as people with difficulty in speaking, spelling, or grammar. A detailed description of a word-prediction system can be found in Section 1.2.2.

Our goal is to introduce more efficient word-prediction algorithms by incorporating

Figure 1.1: Gemini device while is being used by a user with disabilities (picture from Assistive Technology, Inc. *www.assistivetech.com*)

additional information into existing word-prediction systems. We evaluate an existing word-prediction software, *WordQ*. We also design, implement, and evaluate several predictors that use different sources of linguistic and statistical information in order to suggest words to the user.

1.2 Alternative and augmentative communication

Augmentative communication refers to the use of a method or device to supplement a person's ability to communicate. This may be a dedicated device or a software solution. Dedicated devices range from simple systems, such as single-message voice output products to large, complex systems able to handle comprehensive communication needs. In the following sections, several examples of AAC devices are explained.

1.2.1 Alternative access methods

Alternative access refers to the use of a method other than the regular computer input devices to operate a computer, such as speech recognition and synthesis, eye-tracking, direct-brain, and row-column scanning interfaces.

Alternative and virtual keyboards are keyboards with special and/or extra options. An alternative keyboard consists of a physical device and a language set, and maps the input entered by a user through the physical device into selections from the language set. An example of such keyboards is an on-screen keyboard, which appears as an image on the screen, and the user can use a pointing device to move the cursor and make keyboard selections. Row-column scanning devices are also examples of such keyboards. In these devices, the users control their computers by pressing a switch to make choices as items are highlighted one after the other. This highlighting of choices is called scanning.

Many alternative input devices can be augmented with additional processing, such as adding predictive capabilities, using speech recognition and synthesis software. Many AAC systems use speech recognition or synthesis software to allow their users to type by using their voice. Programs are different depending on whether or not they are completely hands-free, their ability to customize to meet individual needs, whether or not they give feedback or corrections to the user, and the method they use to correct recognition errors.

There are several AAC systems that exploit speech to help their users, for example a *screen reader* provides auditory feedback to a blind user about what is happening on the screen. Another type of a *reading system* may allow an individual who is blind or has low vision to use a scanner to scan documents, bills or books into the computer. The computer then reads the material aloud and aids the user to store it in specialized categories for later retrieval.

There is also software that can be used by individuals with learning disabilities. Word processing software such as talking word-processing or word-prediction programs help the users write by providing auditory feedback or suggestions for the current position of the sentence. This is useful for individuals who have difficulty typing letter by letter but who can recognize words. This is also useful for individuals with physical disabilities, since it reduces keystrokes.

1.2.2 Augmentative communication systems

The ability to communicate effectively is a very important issue to users of AAC and AT; and communication rate is a salient characteristic of effective communication. To help people with difficulties in speech or writing, various AAC devices have been designed, but all of them are much slower than the standard ones. Therefore, the text composition rate for these people is too low to maintain a normal conversation, usually less than 15 words per minute, see (Alm et al., 1992), compared to the average rate of normal conversations which is between 150 and 200 words per minute, see (Foulds, 1980). There are several rate-enhancement techniques available to AAC/AT users, such as *abbreviation expansion*, *semantic encoding*, *sentence compansion*, and *word prediction*, see (Demasco and McCoy, 1992). These methods are described in the following paragraphs.

Abbreviation expansion Abbreviation expansion is a simple and effective method to increase the communication rate of AAC devices. It helps the user produce words with relatively few keystrokes. Early systems exploited a lookup table maintaining the words and phrases and their corresponding abbreviations. Although this method allows the users to compose words with few keystrokes, it requires the user to memorize a large set of codes for accessing the words and the system to maintain the abbreviation database.

Some of the systems are rule-based, and hence do not maintain a predefined lookup table. These systems are more flexible, since the users can create their own abbreviations, but they need to remember the rules. An example of such systems is the one developed by Moulton et al. (1999), which requires the user to follow two intuitive rules to create their own abbreviations. The rules are simple, and help the system to find the user's intended words for input abbreviations. For example, one rule may be that an abbreviation should start with the same letter as the word itself.

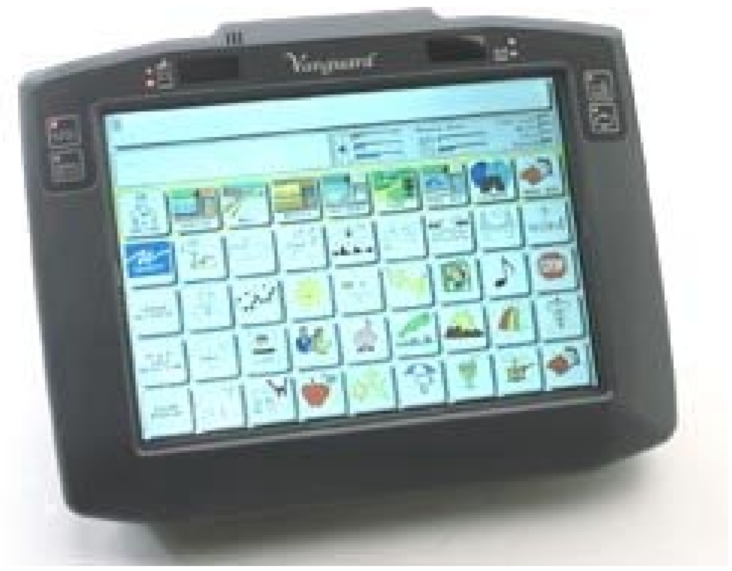


Figure 1.2: Vanguard screen display with its icons for semantic concepts (picture from Aroga Technologies *www.aroga.com*).

Semantic coding In this method, the vocabulary set consists of semantic primitives that can be used to form words. Each word is generated from a sequence of two or three semantic primitives. Semantic coding is a useful technique, but it also requires the user to remember a large number of sequences of semantic concepts for creating sentences.

Minspeak is an example of a semantic compaction system developed by Baker (1982), see also (Cross et al., 2001). The users of the system can produce complete sentences without having to choose letters, phonemes, or words. A number of simple, reusable sentences are stored within Minspeak; each one can be accessed by a short code.

The semantic concepts can be represented as icons, for example. Therefore, the keys on the keyboard bear images that stand for semantic concepts. Each key can have different meanings depending on the context. *Vanguard* is a communication device that combines a dynamic display screen with the Minspeak semantic coding system. A picture of this device is shown in Figure 1.2.

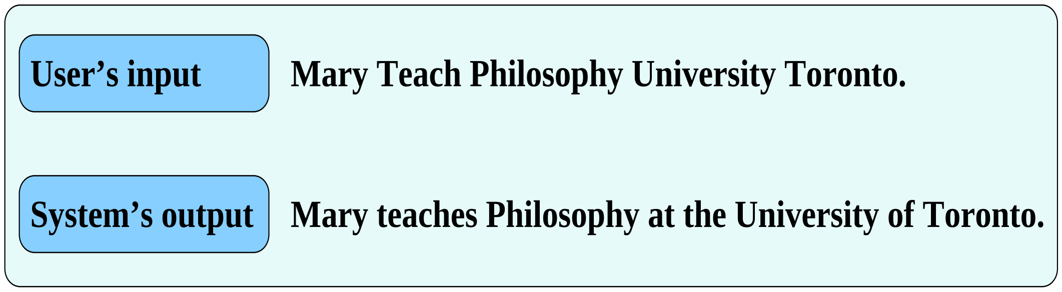
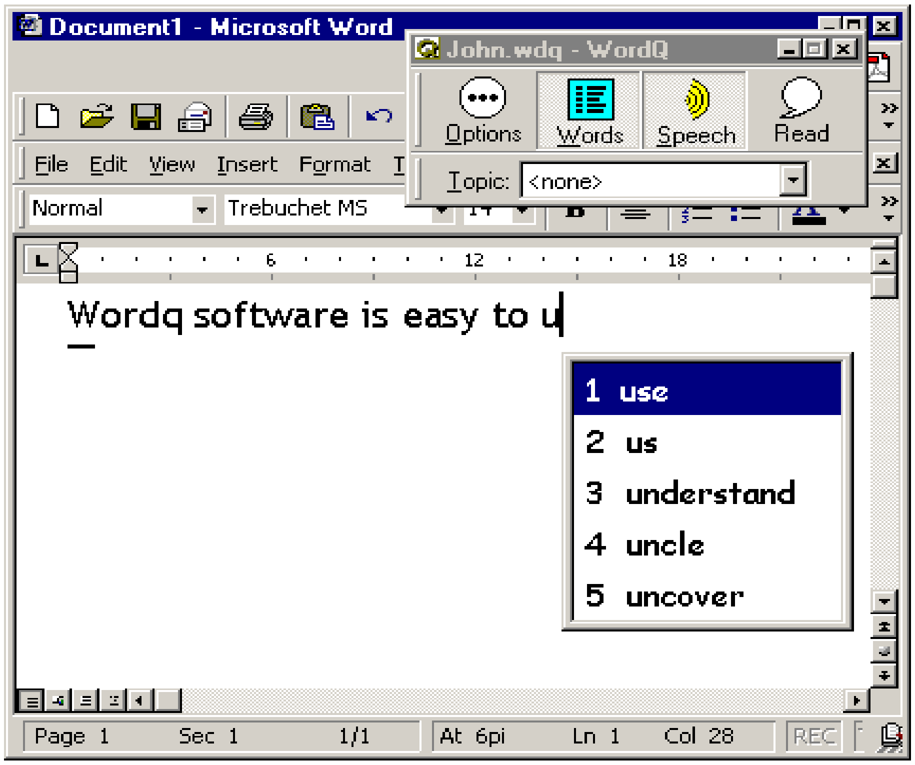

User's input
Mary Teach Philosophy University Toronto.
System's output
Mary teaches Philosophy at the University of Toronto.

Figure 1.3: An example of sentence companion.

Sentence companion A sentence companion system takes a compressed message as input and expands it into a well-formed grammatical sentence. It has a vocabulary set of uninflected content words that the user can choose from. One such system has been developed by Demasco and McCoy (1992). This system exploits a *semantic parser* that produces a semantic representation of the input words. A *representation translator* then translates this semantic representation into a deep structure, which is used as input for the *natural language generator* to produce grammatical sentences. An example is shown in Figure 1.3.

Word prediction Word prediction is the problem of guessing which words are likely to follow a given segment of a text. A word-prediction system typically operates by displaying a list of most likely words or phrases for the current position of the sentence being typed by the user. As the user continues to enter letters, the system updates the prediction list according to the new context. At any time, the user may select one of the suggestions with a single keystroke or mouse click. An example of word prediction by WordQ is shown in Figure 1.4.

Word prediction has applications other than being used as a communication aid for the people with disabilities. Word prediction, or in general predicting a missing word, part-of-speech tag, or any other token given its context, is one of the important tasks in most Natural Language Processing (NLP) applications such as part-of-speech tagging, speech recognition, word-sense disambiguation, context-sensitive spelling correction, and


Document1 - Microsoft Word
John.wdq - WordQ
Options
Words
Speech
Read
File Edit View Insert Format
Topic: <none>
Normal
Trebuchet MS
Page 1 Sec 1 1/1 At 6pi Ln 1 Col 28 REC

Wordq software is easy to u

1 use
2 us
3 understand
4 uncle
5 uncover

Figure 1.4: An example of word prediction by WordQ (picture courtesy of Fraser Shein).

machine translation.

It is known that all these methods impose a cognitive load on their users, resulting in little improvement in text composition rate. It is also indicated that although these techniques reduce the number of keystrokes, and the amount of users' effort as a result, they do not necessarily reduce the *time* needed to enter a text. In a word-prediction system, visually searching the list of suggestions and deciding whether or not the list contains the desired word, changing the point of gaze from the keyboard to the display and vice versa, are activities that increase the cognitive load on the user. Therefore, although a considerable number of keystrokes can be saved by using such a program, there is not always an improvement in overall text generation rate.

However, among various methods for enhancing the communication rate of AAC systems, word prediction is one of the most successful ones, and has been widely used.

Exploiting prediction techniques in an AAC system has several advantages: it can reduce the amount of physical effort; it helps the users spell words correctly and save effort by saving keystrokes; and it improves the quality and quantity of written work. Moreover, the increased cognitive loads may decrease over time as the user becomes familiar with the system.

1.3 Outline of the Study

Although word prediction might have other applications in the domain of computational linguistics, word prediction as a writing aid for people with disabilities is the focus of our study. In this study, various types of statistical information are added to a baseline prediction system to improve the accuracy of predictions, resulting in higher keystroke savings, and a corresponding reduction in the physical effort needed by the user. Therefore, our aim is to find prediction techniques that result in more accurate predictions.

So far most of the existing predictors do not consider the syntactic structure of the sentence. We hypothesize that if a word-prediction system uses syntactic information in addition to the other statistical information such as word unigram and bigram statistics, the predictions would be more accurate.

To test our hypothesis, we collect data from a large corpus of English. Various statistics about words and their part-of-speech tags are collected from the corpus, such as word unigram and bigram statistics, part-of-speech tag unigram, bigram and trigram statistics, and word-tag probabilities. Using this data, we will conduct several experiments in order to find out the effect of including more statistical information in a word-prediction system.

In Chapter 2, we provide a survey of related work in word prediction, and their advantages and disadvantages. Our proposed algorithms for the task of word prediction are presented in Chapter 3. In Chapter 4, we show how statistics are collected on our selected

n-grams. The experimental methodologies and the results are reported in Chapter 5.

Our results confirm that a strong word unigram or word bigram model, collected from a large corpus, is a good predictor. However, it seems that there is an overlap between the information embedded in a word bigram model and the information which can be found in a part-of-speech tag trigram model. Thus, adding part-of-speech tag information to word bigrams and attempting to predict using both information sources is not as helpful as we expected.

In the final chapter, we conclude by discussing the contributions and limitations of the current work, and suggest possible future extensions.

Chapter 2

Related work

2.1 Introduction

Statistical Natural Language Processing (NLP) aims to do statistical inference for the field of NLP. Statistical inference consists of taking some data for training and making inferences about unseen data in general, on the basis of the distribution and other statistical information of the training data.

An example of statistical estimation is the task of language modeling, where the problem is to predict the next word given the previous words. However, the methods developed for this task can be used in other applications such as word sense disambiguation, probabilistic parsing, part-of-speech tagging, etc., see (Jurafsky and Martin, 2000; Manning and Schütze, 1999).

Word prediction can be stated as attempting to estimate the probability of having a word in a desired sentence position, given the sequence of all its previous words in the context. Since there is not always enough data to consider each textual history separately, we need a method to group similar histories in order to give reasonable predictions for the next position in the sentence. One possible way is making a *Markov assumption*, which says that only the last $n-1$ words of the history affect the next word. This is called an $(n-1)^{th}$ order Markov model or an n-gram word model. Therefore, we estimate the probability of each word, w_n, considering only $n-1$ previous words: $w_1, w_2, ..., w_{n-1}$. The probability is defined to be $P(w_n \,|\, w_1, w_2, ..., w_{n-1})$.

Usually, we would like the n in an n-gram model to be sufficiently large to include most of the sequences that might happen in the language under consideration. Unfortunately, even if we have a large corpus as training data, there is always the problem of data sparseness for large values of n. Therefore, current systems mostly use unigrams ($n = 1$), bigrams ($n = 2$), and sometimes trigrams ($n = 3$).

Most of the existing word-prediction systems exploit *statistical* prediction algorithms. The information that is mostly used in such algorithms is *word frequency* statistics. Several systems incorporate syntactic information, by adding *word category* statistics or *grammar rules*, in order to improve the accuracy of word prediction. Other programs may use *semantic* information along with other information sources. There are also systems that *adapt* to the user's language by using specific vocabulary or through learning.

The general approach of any statistical word-prediction algorithm, regardless of the information sources it exploits, is as follows:

- Find a subset of the lexicon, called W, which contains all words starting with the *word prefix* entered so far by the user.
- For each candidate word $w_i \in W$, compute the probability of this word being in the desired sentence position. This probability is usually computed from various statistical information that is maintained along with every word in the lexicon.
- Suggest the top n words of the candidate word list, according to the probability computed for each word in the previous step.

In Sections 2.2 and 2.3 several existing letter-prediction and word-prediction systems are introduced. Section 2.4 describes some other word-prediction systems that incorporate syntactic information into their prediction algorithms, either by using word category information or grammatical rules.

2.2 Statistical letter prediction

There are several letter-prediction systems that attempt to help people with disabilities. Although our aim in this study is to improve the performance of word prediction, looking at the ideas of letter prediction might also be useful. In the following section, three such systems are described in detail.

2.2.1 Predictive keyboards

Foulds et al. (1987) have developed a communication model to reduce the number of keys representing the English alphabet, from 27 (a – z, and space) to 11. On the new keyboard, each key represents a cluster of three letters; therefore a disambiguation process is needed to determine which letter the user wants to type when pressing a key. It is possible that the disambiguation process fails to predict the user's desired letter correctly; therefore, one key on the keypad is used for error correction.

The model exploits the letter-quadgram transitional probabilities to predict the appropriate letter without using a lexicon of English words. The statistical information has been derived from the Brown Corpus. This letter-prediction system, or predictive keyboard, can be used by users with disabilities as a specific device. The authors claim that since the user does not have to choose among 27 keys, but only 11 keys, the effort and cognitive load imposed on the user is decreased compared to when the user works with a standard keyboard.

Such a device can be useful only if the number of keystrokes per character is not large. In this system, the final keystrokes per character achieved in one of the experiments, was about 1.14 for English.

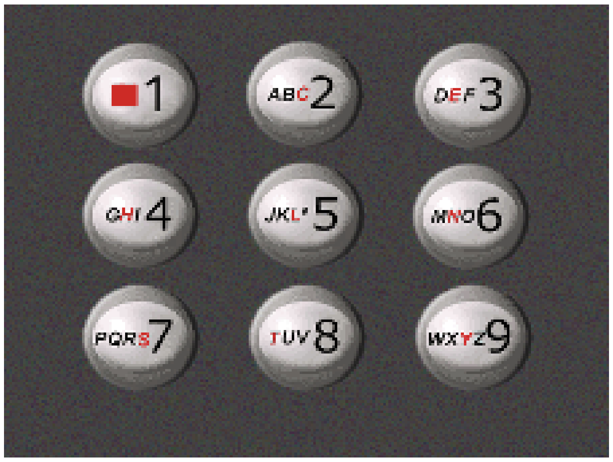


Figure 2.1: The standard 12-key telephone keypad (picture from Eatoni Ergonomics, Inc. *www.eatoni.com*).

There are other similar text entry methods and devices, developed to decrease the number of keystrokes. Most of these techniques are used on mobile phone keypads. Although the purpose of developing such systems is different from those which are used as AAC devices, the ideas are similar. An example of such keypads is shown in Figure 2.1

LetterWise is such a technique, developed by MacKenzie et al. (2001), and is used to enter text using a phone keypad. This technique is used to decrease the total number of keystrokes needed to enter a text by using the conventional text entry methods using a 12-key keypad, consisting of number keys 0 – 9 and two additional keys. Letters a – z and space character are spread over keys 0 – 9; thus, three or four letters are grouped on each key resulting in ambiguity.

In this method, each keypress is disambiguated according to the previously typed letters. In one implementation of LetterWise, up to three previous letters are considered in disambiguation. Thus, LetterWise includes databases of probabilities of letter quad-grams in the target language. Keystrokes per character for LetterWise in an experiment was 1.15 for English. There are also commercial systems, such as T9 (*www.t9.com*), that work on the keypress disambiguation for mobile phones.

The Reactive Keyboard, developed by (Darragh et al., 1990), see also (Darragh and

Witten, 1992) is another alternative interface for physically people with disabilities who have difficulty using a regular keyboard. It is embedded in an editor, and generates predictions from a text model that is created and maintained adaptively from previously entered text.

The Reactive Keyboard works by attempting to predict what the user might want to select next, on the basis of preceding input. It uses an n-gram model for characters, created from text samples and from the user's input. The model is stored in a special tree structure that allows partial matches between context and model to be found economically. The idea is to use the $n-1$ previous characters to predict the nth one, where possible. If matches cannot be found, the context is shortened by one character, and the process continues.

As a result of the prediction process at each point, a list of up to 128 likely characters for the next position of the current sentence is found. In the next step, a list of predictions, each beginning with one of the predicted characters, is displayed to the user. The user can choose one of them and click at a point; characters up to that point are inserted into the editing window. The prediction window is scalable and always contains 128 predictions ordered by a score assigned to them according the probability of their first letters being the next letter. Predictions beginning with the most likely next letter are in the highest position of the list.

2.3 Statistical word prediction

Language statistics are a useful information source for most of the existing commercial and academic word-prediction systems. The statistical information used in prediction systems is mostly n-gram language models.

Most of the early word-prediction programs, especially those that were developed as writing aids for people with disabilities in the early 1980s, used word frequency information, i.e., a word unigram model, to predict words for current or next position. These predictors ignore all the previous context, and use only word frequency information in their prediction process. In order to improve the accuracy of predictions, some predictors consider a larger context and use word sequence statistics, such as word bigram or trigram models.

A major drawback of both unigram and bigram predictors is that they do not consider the syntactic and semantic structure of the sentence, and therefore there is a possibility of predicting words which are syntactically and/or semantically inappropriate for the desired sentence position. Another disadvantage of such predictors is that they consider only a small context.

2.3.1 Existing N-gram predictors

WordQ software is a writing tool that was developed at the Bloorview MacMillan Centre, Toronto, Canada, with support from the Ontario Rehabilitation Technology Consortium (Nantais et al., 2001; Shein et al., 2001). WordQ can be used along with standard Windows word-processing software; it has a word-prediction module that suggests words to the user, and a speech module that provides spoken (text-to-speech) feedback.

The WordQ prediction module uses word unigram and word bigram statistics along with optional customization to each user's vocabulary in order to suggest appropriate words. Wherever the bigram probability exists, the WordQ prediction algorithm suggests words according to that. In this case, the predictor suggests words whose bigram probabilities according to the given previous word are higher. Otherwise, it uses the unigram language model to predict the next word, i.e., it ignores the previous word and predicts only according to the words' frequency of use. A few examples of prediction by WordQ are shown in Figure 2.2.

WordQ is adaptive, and thus the vocabulary used for prediction is customizable depending on the user's level and the specific topic selected by the user. The system

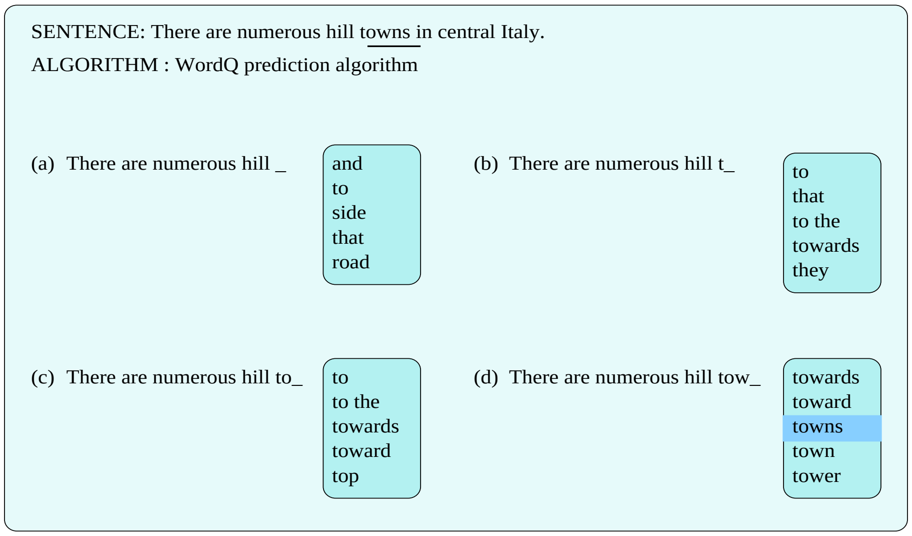

SENTENCE: There are numerous hill towns in central Italy.
ALGORITHM : WordQ prediction algorithm
(a) There are numerous hill _
and
to
side
that
road
(b) There are numerous hill t_
to
that
to the
towards
they
(c) There are numerous hill to_
to
to the
towards
toward
top
(d) There are numerous hill tow_
towards
toward
towns
town
tower

Figure 2.2: Sample predictions given by WordQ: (a) predictions after completion of the previous word *hill*; (b) predictions after entering the first letter of the current word; (c) the second letter of the word *towns* has been entered; (d) after entering the third letter, the user's desired word is predicted.

captures new words and adds them to the user's lexicon. Thus, there is a large background lexicon used to check the spelling of the new words before adding them to the user's lexicon, in order to prevent wrong words from entering the system's lexicons. It also dynamically changes its frequency information to adapt to the users' frequencies. A user frequency is assigned to each candidate word along with the unigram and bigram probabilities. The user frequency changes dynamically according to the use of the word by the user. This frequency affects the final probability of choosing a word for the desired position.

(Nantais et al., 2001) did several experiments on WordQ in which the keystroke savings of WordQ was measured on three test texts, 116,579 words in total. The predictor achieved between 37% and 53.1% keystroke savings, depending on the type of text, adaptation, and the type of frequency information used. In all experiments, the size of the prediction list is 5.

Predictive Adaptive Lexicon (PAL) is a computer program developed by Arnott et al.

(1984), see also (Swiffin et al., 1985), which assists users in composing text by suggesting a list of words which are appropriate for the desired position of the sentence. The user can choose from the list by an appropriate key press or continue typing their own words.

The prediction system consists of some standard lexicons; each is constrained to a maximum size of the order of 1000 words. But it is possible to create a new lexicon for a specific user, either online, by creating the lexicon while the user types in words, or offline, by creating the lexicon from an existing text written by the user. The lexicons contain statistical information for each word, such as word frequency and recency of use.

The prediction module uses word-frequency information and predicts words according to the current prefix. It is also adaptive, i.e., it uses words' recency information to suggest words. The lexicon and the statistical information used by the system are both adaptive and hence change according to their use by the user. PAL also automatically captures new words, and adds them to the lexicon for future predictions. In order to prevent wrongly spelled words going into the master lexicon, PAL is supplemented with a spelling corrector program.

The predictor was tested using a number of texts, including a technical manual, several newspaper articles, and the transcript of a TV program. With 10 words in the prediction list, the predictor achieved keystrokes savings between 30% and 46%, depending on the type of text.

Another word-prediction system, developed by Bentrup (1987), uses both the frequency of occurrence of individual words and the knowledge of their sequential dependencies, i.e., the transitional probabilities between words, in order to make better predictions. The system incorporates the n-gram models of language to make predictions.

Using a corpus, a lexicon is constructed containing the frequency of words and word sequences. The lexicon contains words and phrases of two to seven words. Since there are phrases of up to seven words in the master lexicon, an initial key of six previous words is used to retrieve words and phrases to suggest to the user. The transitional probabilities of words are used to find the most likely suggestions. If enough matches cannot be found using the initial key, the key is shortened by one word. The process of retrieving words and phrases that match the key is continued until either the desired number of words or phrases are found, or the key is an empty string. At this time, if there are not enough predictions in the list, a number of words and phrases are considered as suggestions according to their individual frequency of use, i.e., all the previous words are ignored in the prediction process.

Communication rate, the number of characters entered by user in one minute, and *communication efficiency*, the number of movements to enter one character, were used as measures of performance. The system was tested by five users with motor impairment who transcribed articles into the predictor. The effect of using specialized dictionaries was also measured for these users. On average, the communication rate was 18.1 characters per minute, and the communication efficiency was 1.98 movements per character (when no specialized dictionaries were used).

Profet is a statistical and adaptive word-prediction program for Swedish, developed by Carlberger et al. (1997a,b), and has been used as a writing aid since mid-1980s. When users are typing with Profet, a list of up to 9 likely words is presented to them and they can either accept a suggestion if their intended word appears in the list or continue typing, otherwise.

An old version of the program is strictly frequency-based and exploits three information sources, one at a time: the word unigram lexicon, the word bigram lexicon, and the user lexicon used for adaptation purposes. The word unigram and bigram lexicons are created from a subset of the Stockholm-Umeå Corpus (SUC).

The predictor uses a bigram lexicon in order to predict the next word when the user completes a word, but after the user enters letters from the new word, the system ignores any previously typed words, and considers only word unigram probabilities. Profet uses recency information to adapt to the user's typing behaviour. Therefore, the recently used words are more likely to be predicted when they match.

A text segment of about 200 words was used to measure the performance of the predictor. Profet achieved a keystroke savings of around 33%, when five suggestions and no adaptation were used.

To summarize: PAL does not consider the previous word; it uses only word frequency information and the word prefix entered so far. Both WordQ and Profet attempt to take the previous word into account by using word bigram statistics. But the bigram probabilities are not always available; therefore, they do not always use the information about the previous word. Another problem with these systems is that they do not consider the syntactic structure of the sentence.

2.4 Syntactic word prediction

Statistical prediction systems that do not consider syntactic information may suggest words which are not grammatically appropriate. Therefore, they impose an extra cognitive load on the user, decreasing the text composition rate. Also, it has been found in various studies that removing the syntactically inappropriate words from the list of suggestions improves the comfort of user, even if the actual saving in physical effort does not change much , see (Hunnicutt, 1987; Woods, 1970).

Therefore, various prediction systems have been developed that attempt to include syntactic information in their prediction process. The goal of syntactic prediction is to ensure that the system does not suggest grammatically inappropriate words to the user. Some of these systems consider part-of-speech tag information of words as syntactic information, while others use a parser to build the syntactic structure of the whole sentence.

However, an important issue is that the prediction system is required to be as fast as possible. But the more information sources the predictor uses to suggest words, the slower the system is. Also, it is found in some experiments that by including syntactic information in prediction, appropriate words are removed from the prediction list almost as often as inappropriate words are prohibited from appearing to the list, see (Hunnicutt, 1987).

Another possible drawback of a syntactic predictor is that it cannot adapt itself to the user's vocabulary easily since all the words in the lexicon need to be tagged with their syntactic category and some other grammatical features.

2.4.1 Existing syntactic predictors

The Predictive Program is a program used as a writing aid for people with disabilities. It has been developed at the University of Washington, see Treviranus and Norris (1987). When the user of the system enters the first letter of the desired word, the system presents 6 to 10 possible completions for the existing word prefix. The predictions are presented according to grammatical information of the preceding word(s) and the frequency of English words. Unfortunately, it is not clear how the grammatical information is used by the prediction algorithm. The system was tested by two persons with cerebral palsy. They had used the system for 6 and 9 months, respectively, and found the system useful for their writing needs.

Syntax PAL is a version of PAL that incorporates some grammar rules into its prediction module to offer more appropriate predictions. A set of 16 basic syntactic classes is used to tag the words in the lexicon and in the sentence being entered; each word is assigned its most common syntactic class. To predict words, the word frequency and the syntactic category bigrams are used, i.e., the following probability is assigned to each candidate word:

$$F(w) * P(syncat_2 \,|\, syncat_1),$$

where $F(w)$ is the frequency, or the unigram probability, of the candidate word w, and

$P(syncat_2 \mid syncat_1)$ is the first-order syntax transition probability, or the syntactic categories' bigram probability, in which $syncat_2$ is the possible syntactic category of the current word, and $syncat_1$ is the syntactic category of the previous word. The syntax transition matrix, containing all these bigram probabilities for syntactic categories, is calculated from a tagged training corpus in which every word is tagged with its most common syntactic category.

Syntax PAL also maintains a list of English function words along with a transition matrix indicating the possibility of occurrence of a particular function-word pair. This information is used to remove unlikely combinations from the prediction list, see (Booth et al., 1992; Swiffin et al., 1987, 1988).

Three texts of $3,500$ words each were used to test Syntax PAL. Syntactic information gives an improvement between 0.5% and 2%, depending on the text and the number of predictions offered.

The prediction system developed by VanDyke (1991) uses a grammar of English to provide the user with a list of grammatical suggestions. The predictor works by traversing the search space produced by constructing the parse tree of the input sentence. The parser holds all possible structures for the partial sentence entered so far, and thus at each point in the sentence, it knows what syntactic categories can be in the next position. This eliminates a number of words to choose from, resulting in more appropriate predictions, but it requires a considerable amount of work to partially parse the input sentence every time a new word is completed by the user. There is no evidence of testing the system to see whether the use of grammar helps improve the prediction performance.

In the prediction system developed by Wood (1996), Windmill, a context-free grammar (CFG) along with augmented phrase structure rules are employed to improve prediction performance through grammatical processing. At each point in the sentence, all possible continuations are considered for prediction purposes. In this way, all the possible syntactic categories for the current position can be used to extract appropriate words from the lexicon. This list of words is then sorted according to their likelihood using the statistical information stored along with the words. The first few words in the list are then offered to the user as suggestions.

Three pieces of text were used to test Windmill: an article, a scientific paper, and a set of transcripts of several conversations. The system was evaluated according to hit rate, keystroke savings and the position of the desired word in the prediction list. The predictor achieved average keystroke savings of between 30.44% and 55.10%, depending on the type of text, and the prediction algorithm: whether it uses syntactic information or not.

Garay-Vitoria and González-Abascal (1997) have developed a syntactic word-prediction system that exploits a chart parser to analyze the syntactic structure of the sentence. The chart-parsing technique is used to determine the most probable syntactic category which is appropriate for each position. The predictor uses a lexicon containing statistical, syntactic, and morphological information for each word. The system also uses a set of grammar rules, each of which is assigned a weight according to its frequency of use. Within a rule, information about gender and number are defined and can be used in addition to morphological information to properly adapt gender and number of the proposals, if necessary.

The system is also adaptive, meaning that the lexicon changes dynamically to adapt to the user's vocabulary, and the weights assigned to the grammar rules change according to their frequency of use by a specific user. Whenever a user enters a word that cannot be found in the lexicon, the system asks the user about the syntactic category of the new word and then adds it to the lexicon.

When the number of predictions proposed to the user is 5, the system achieved keystroke savings of about 58%, compared to those of a baseline which only uses frequency information, 56% (both predictors are adaptive). There is no more information about their experimental methodology: whether they tested their system in a real environment with real users or they used a test corpus for doing so.

Even-Zohar and Roth (2000) have incorporated additional information into the learning process of their word-prediction system, in order to learn better language models in comparison to prediction systems that use n-gram models. In the proposed prediction system, the local context information along with the global sentence structure are considered. For this purpose, a very large set of features characterizing the syntactic and semantic context in which the word tends to appear has been used, a representation is learned for each word in terms of the features, and a learning method that is capable of handling the large number of features is used. A language for introducing features in terms of the available information sources is also defined.

Each sentence is represented as a list of predicates, called the *Information Source* (IS) of the sentence. Features are defined as relations over the information source, or aspects of the structure of the sentence. A few examples of features are the *adjacency* relation between words, word *collocations*, and the part-of-speech tag assigned to each word. There are also complex features such as the *dependency* relation between words and role of each word or phrase inside the sentence, i.e., if it is a subject, object, etc.

The *Wall Street Journal* of the years 1988 – 89, about one million words, is used for training and testing. Two different feature sets and three different classifiers are used in order to compare the results. For testing purposes, a list of *confusion sets* is constructed, each including two different verbs whose occurrence is equally likely in the training corpus. In the test set, every occurrence of verbs v_1 and v_2 is replaced by the set $\{v_1, v_2\}$. A classifier then attempts to predict the appropriate verb for each position.

This work is different from other word-prediction systems, in three ways: First, the predictor assumes that the entire sentence is available, except for one word, which is being predicted. Therefore, it can use the context before and after the prediction position in the sentence. Second, the verb prediction task, as a specific application of word prediction, is implemented for evaluation purposes. Third, the prediction is limited to choosing from two possible verbs for each position. Therefore, the method might not be applicable to prediction systems that are used as writing aids for people with disabilities.

Hunnicutt (1989) conducted a number of experiments to measure the effect of adding syntactic and semantic information to a Swedish word-prediction system. To include syntactic information, a lexicon of about $10,000$ Swedish words was used, in which every word was marked by its part-of-speech tag. The predictor considers some of the syntactic classes to be unlikely for the current positions. Words which are from these classes are then removed from the prediction list.

In an early version of the system that did not include syntactic and semantic information sources, the predictions were chosen either according to their frequency of use stored in the lexicon, or on the basis of their usage in the text being typed by the user, taking both frequency and recency information of the words into account. Results of the experiments showed a small improvement in keystroke savings for syntactic information.

In order to enhance the accuracy of Profet, a new version of it has been developed. In the new system, developed by Carlberger et al. (1997a), the scope is extended to include part-of-speech tag trigrams and word bigrams at each prediction point. The new prediction algorithm exploits two interacting Markov Models: a second-order Markov Model for part-of-speech tags, and a first-order Markov Model for words, which is modified to consider the conditional probability of a word given the probability estimation of the tag obtained by the tag Markov Model.

The idea of the prediction algorithm is to first obtain a probability estimation for the tag of the next word, using the tag Markov model. In the next step, a probability estimation is found for the next word using the word Markov model. The tag probability estimation from the previous step is used to promote rank of the words with the most likely tag. The tag unigram, bigram, and trigram lexicons are created from the same corpus (SUC) used to build word unigram and bigram language models.

Three texts of about 10,000 words each, have been used for evaluation. The system achieved a keystroke savings of about 43.2%, when five suggestions and no adaptation were used.

2.5 Summary

Most of the systems introduced in this chapter have not been evaluated using a complete set of experiments that capture different aspects of performance of word prediction. Some of them have used very small test texts, and most of them have only used keystroke savings as a performance measure. Some of them were tested by real users with disabilities, but not on a test set.

Our goal have been to design and implement a testbench in which we can plug any word-prediction algorithm and can work with large training and test data. We have also introduced several prediction algorithms, some of them exploited syntactic information. A number of performance measures have been introduced, and several experiments on large corpora have been conducted to evaluate each one of the prediction algorithms, and compare them according to the measures.

Chapter 3

Prediction algorithms

Suppose the user is typing a sentence and the following sequence of words has been entered so far:

$$\cdots \quad ppw \quad pw \quad cw_{prefix},$$

where *ppw* and *pw* are the last two completed words, or the two previous words of the current position, and cw_{prefix} is a prefix of the current word that the user has typed so far. We define W to be the set of all lexicon words beginning with the prefix cw_{prefix}. A predictor attempts to choose the n (usually between 1 and 10) most appropriate words from the set of candidate words, W, according to the context, i.e., the sequence of words and the current prefix.

Each prediction algorithm may incorporate different types of information about words and the context. In the following sections, we will introduce several prediction algorithms that we have implemented and tested. Some of them use only statistical information about the words in the sequence, such as *Unigram predictor* and *Bigram predictor*, which are explained in more detail in Section 3.1 and Section 3.2. Others may also include syntactic information about the words through the use of part-of-speech tags. These predictors are introduced in Section 3.3.

Most of the word-prediction programs that are used by individuals with disabilities attempt to adapt to the user's behaviour in different ways, in order to improve the accuracy of the predictions. In Section 3.4, some of the adaptation techniques used in word-prediction systems, especially in WordQ and our predictors, are described.

3.1 Unigram predictor: the baseline

We have implemented a baseline word-prediction algorithm that uses only word unigram statistics to do prediction. The Unigram predictor sorts all lexicon words that match the word prefix entered so far, according to their frequency of use. Word frequency information, i.e., word unigram statistics, is gathered from a training corpus, which is a subset of the British National Corpus (see Chapter 4). The predictor then presents the top n words, according to their frequency, as suggestions to the user. An example of words suggested by the Unigram predictor is shown in Figure 3.1. It can be seen that in any of the prediction steps, only a few of suggestions are grammatically acceptable for the desired position.

3.2 Bigram predictor

The Bigram predictor expands the context and takes the previous word into account, in addition to the word prefix entered so far. The predictor first looks for all the lexicon words that match the existing word prefix. It then retrieves the bigram probability, $P(w_i \,|\, pw)$, for each candidate word w_i. In the next step, the predictor selects the top n words in the candidate list according to their bigram probability. These high-probability words create a list of suggestions for the user. An example of words suggested by the Bigram predictor is shown in Figure 3.2. Compared to prediction given by the Unigram predictor, most of the suggestions given by the Bigram predictor are more appropriate for the desired position.

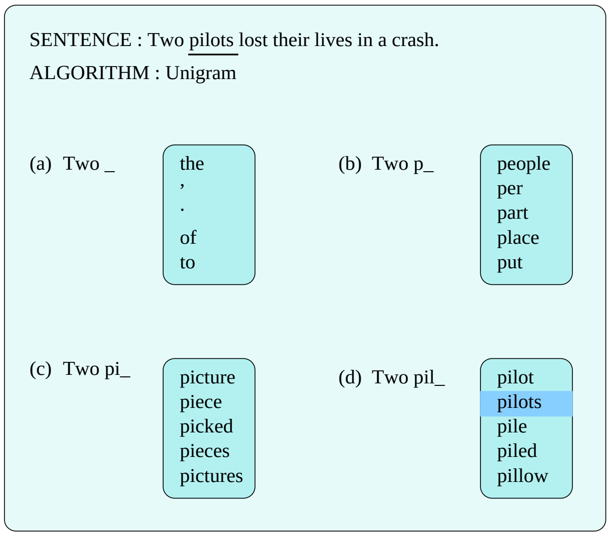


Figure 3.1: Sample predictions given by the Unigram predictor: (a) predictions after completion of the previous word *Two*; (b) predictions after entering the first letter of the current word; (c) the second letter of the word *pilots* has been entered; (d) after entering the third letter, the user's desired word is predicted.

The WordQ prediction algorithm, described in Chapter 2, is similar to our Bigram predictor. The most important difference between these two predictors is that in WordQ, when the bigram probability does not exist, the system backs off to the unigram probability. However, we use a backoff weight together with the unigram probability wherever the bigram does not exist. Backoff n-gram models were proposed by (Katz, 1987). If the n-gram of concern (here bigram) has not appeared or appeared k times or less (k is normally set to 0 or 1, in our case 0) in the training data, then an estimate from a shorter n-gram (here unigram) is used (for more details see Chapter 4, Sections 4.2.2 and 4.2.3).

3.3 Syntactic predictors

Our syntactic predictors make use of syntactic information by considering the part-of-speech tags assigned to the words by a part-of-speech tagger. Using part-of-speech tag information has two major advantages: first, it considers the syntax of the sentence,

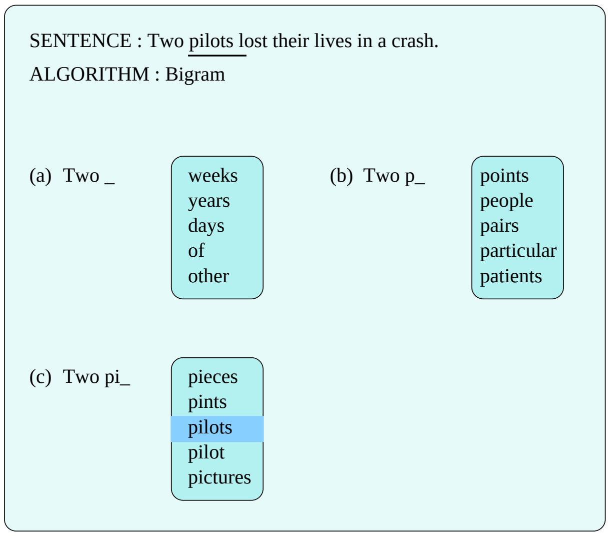


Figure 3.2: Sample predictions given by the Bigram predictor

since the tagger has to look at all the previous words and their tags to tag each word in the sentence; secondly, since the trigram model for part-of-speech tags is much smaller than the trigram model for words, the syntactic predictor can look at the tags of the two previous words when suggesting words to the user, resulting in a larger context.

A syntactic predictor assumes that a part-of-speech tagger tags the sentence words as the user types them, i.e., the tagger tags the sentence from the beginning every time a new word is entered by the user. A tagger that can tag each new word without having to tag the whole sentence from the beginning is of course a better choice.

Some part-of-speech taggers, such as the Brill's tagger, have to look at the context around each word several times to decide about the final tag assignment of the word. Thus, these taggers are not appropriate for the application of word prediction. Others may be looking at a minumum number of surrounding words and especially only the words prior to the current word to assign a tag. These taggers (with slight changes) can be used in word-prediction environments.

A syntactic predictor has access to the following sequence of words and part-of-speech tags to predict the current word:

$$\cdots \quad ppw/t_{ppw} \quad pw/t_{pw} \quad cw_{\mathit{prefix}},$$

where t_{ppw} and t_{pw} are the part-of-speech tags of the previous words *ppw* and *pw*, respectively.

There are different ways of incorporating the part-of-speech tag statistical information into the predictor. Some of these approaches are explained in the following sections. Our experiments and the results achieved by each of the approaches are explained in Chapter 5.

3.3.1 Part-of-speech tags only

In this method, we assume that the part-of-speech tag is sufficient for prediction. Therefore, we can assign a probability to each candidate word w_i by estimating the probability of having this word in the current position, given that the most likely tags for the two previous words are t_{ppw} and t_{pw}, respectively. This probability can be estimated as follows:

$$P(w_i \,|\, t_{ppw},\, t_{pw}) \;=\; \sum_{t_{ij} \in T(w_i)} \{P(w_i \,|\, t_{ij}) \times P(t_{ij} \,|\, t_{ppw},\, t_{pw})\}\,, \qquad (3.1)$$

where t_{ij} is the jth tag for w_i in which j varies from 1 to $|T(w_i)|$; $T(w_i)$ is the set of all possible part-of-speech tags that may be assigned to w_i; $P(t_{ij} \,|\, t_{pw}\,,\, t_{ppw})$ is the part-of-speech tag trigram probability; and $P(w_i \,|\, t_{ij})$ is the conditional probability of having word w_i as the current word of the sentence given t_{ij} as its part-of-speech tag. This probability can be calculated by Bayes's formula as follows:

$$P(w_i \,|\, t_{ij}) \;=\; \frac{P(t_{ij} \,|\, w_i) \times P(w_i)}{P(t_{ij})},$$

where $P(w_i)$ and $P(t_{ij})$ are unigram probabilities of English words and part-of-speech tags, respectively; and the conditional probability $P(t_{ij} \,|\, w_i)$ is calculated from a corpus for each word of the lexicon and all of its possible part-of-speech tags.

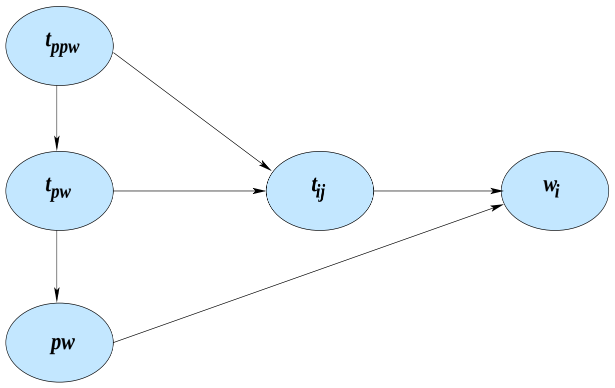


Figure 3.3: A bayesian network showing interdependencies among words and part-of-speech tags.

3.3.2 Previous word and two previous part-of-speech tags: Tags-and-words

Another approach to incorporating part-of-speech tag statistics into the prediction algorithm, is to estimate the probability of each candidate word w_i, given the previous word, pw, its part-of-speech tag, t_{pw}, and the part-of-speech tag of its preceding word, t_{ppw}; i.e., $P(w_i \,|\, pw,\, t_{pw},\, t_{ppw})$. The probability $P(w_i \,|\, pw,\, t_{pw},\, t_{ppw})$ can be estimated as follows:

$$P(w_i \,|\, pw,\, t_{pw},\, t_{ppw}) \;=\; \sum_{t_{ij} \in T(w_i)} \{P(w_i,\, t_{ij} \,|\, pw,\, t_{pw},\, t_{ppw})\}\,, \qquad (3.2)$$

where $P(w_i,\, t_{ij} \,|\, pw,\, t_{pw},\, t_{ppw})$ is calculated as follows:

$$P(w_i,\, t_{ij} \,|\, pw,\, t_{pw},\, t_{ppw}) \;=\; P(w_i \,|\, pw,\, t_{ij},\, t_{pw},\, t_{ppw}) \times P(t_{ij} \,|\, pw,\, t_{pw},\, t_{ppw}) \qquad (3.3)$$

In Figure 3.3 a simple Bayesian network is shown, in which the direct conditional dependencies among t_{ppw}, t_{pw}, pw, t_{ij}, and w_i are specified. Using the conditional independence assumptions, as can also be seen in the Bayesian network, we can assume that if the most likely tag for the current word has been found according to the two previous tags, we do not need to know the previous tags themselves. Therefore, $P(w_i \,|\, pw,\, t_{ij},\, t_{pw},\, t_{ppw})$ can be rewritten as $P(w_i \,|\, pw,\, t_{ij})$. Also we can assume that the tag of the current word is conditionally independent of the previous word itself, knowing its part-of-speech tag.

Thus, $P(t_{ij}\,|\,pw,\,t_{pw},\,t_{ppw})$ can be rewritten as $P(t_{ij}\,|\,t_{pw},\,t_{ppw})$. As a result of the above independence assumptions and using Bayes rule, the above probability can be estimated as follows:

$$\begin{aligned}
&P(w_i\,|\,pw,\,t_{ij}) \times P(t_{ij}\,|\,t_{pw},\,t_{ppw}) && (3.4)\\
&= \frac{P(pw,\,t_{ij}\,|\,w_i) \times P(w_i) \times P(t_{ij}\,|\,t_{pw},\,t_{ppw})}{P(pw,\,t_{ij})}\\
&= \frac{P(pw\,|\,w_i,\,t_{ij}) \times P(t_{ij}\,|\,w_i) \times P(w_i) \times P(t_{ij}\,|\,t_{pw},\,t_{ppw})}{P(pw,\,t_{ij})} && (3.5)\\
&\approx \frac{P(pw\,|\,w_i) \times P(t_{ij}\,|\,w_i) \times P(w_i) \times P(t_{ij}\,|\,t_{pw},\,t_{ppw})}{P(pw,\,t_{ij})}\\
&= \frac{P(w_i\,|\,pw) \times P(pw)}{P(w_i)} \times \frac{P(t_{ij}\,|\,w_i) \times P(w_i) \times P(t_{ij}\,|\,t_{pw},\,t_{ppw})}{P(t_{ij}\,|\,pw) \times P(pw)}\\
&= \frac{P(w_i\,|\,pw) \times P(t_{ij}\,|\,w_i) \times P(t_{ij}\,|\,t_{pw},\,t_{ppw})}{P(t_{ij}\,|\,pw)} && (3.6)
\end{aligned}$$

According to the Bayesian network, neither of t_{ij} and pw are directly dependent on the other, and therefore we can replace $P(t_{ij}\,|\,pw)$ (the denominator in line 3.6) by $P(t_{ij})$, and $P(pw\,|\,w_i,\,t_{ij})$ (line 3.5) by $P(pw\,|\,w_i)$. Thus, the probability $P(w_i,\,t_{ij}\,|\,pw,\,t_{pw},\,t_{ppw})$ (Equation 3.3 above) can be estimated by the following formula:

$$P(w_i,\,t_{ij}\,|\,pw,\,t_{pw},\,t_{ppw}) \approx \frac{P(w_i\,|\,pw) \times P(t_{ij}\,|\,w_i) \times P(t_{ij}\,|\,t_{pw},\,t_{ppw})}{P(t_{ij})}$$

Therefore, we can assign a probability to each candidate word according to the following formula (see Equation 3.2 above):

$$\begin{aligned}
P(w_i\,|\,pw,\,t_{pw},\,t_{ppw}) &= \sum_{t_{ij}\in T(w_i)} \{P(w_i,\,t_{ij}\,|\,pw,\,t_{pw},\,t_{ppw})\}\\
&\approx \sum_{t_{ij}\in T(w_i)} \frac{P(w_i\,|\,pw) \times P(t_{ij}\,|\,w_i) \times P(t_{ij}\,|\,t_{pw},\,t_{ppw})}{P(t_{ij})}\\
&= P(w_i\,|\,pw) \times \sum_{t_{ij}\in T(w_i)} \frac{P(t_{ij}\,|\,w_i) \times P(t_{ij}\,|\,t_{pw},\,t_{ppw})}{P(t_{ij})} \qquad (3.7)
\end{aligned}$$

An example of predictions given by the Tags-and-words predictor is given in Figure 3.4.

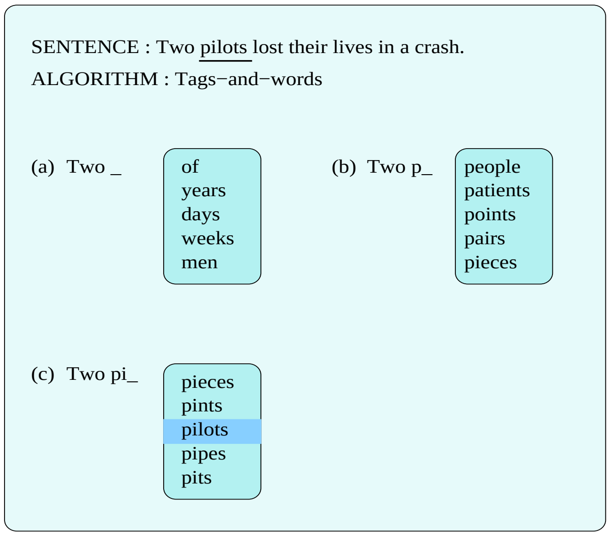


Figure 3.4: Sample predictions given by the Tags-and-words predictor.

3.3.3 Linear combination

The main idea of the *linear combination* approach is that the predictor first attempts to find the most likely part-of-speech tag for the current position according to the two previous part-of-speech tags. Then, it attempts to find words that have the highest probability of being in the current position according to the predicted part-of-speech tag. It then combines this probability with the probability of the word given the previous word. The two predictors, one that predicts the current tag according to the two previous part-of-speech tags, and the one that incorporates bigram probability to find the most likely word, are combined using a linear combination with a coefficient $0 \leq \alpha \leq 1$:

$$\alpha \times P(w_i \,|\, pw) + (1-\alpha) \times [P(w_i \,|\, t_{cw}) \times P(t_{cw} \,|\, t_{pw} , t_{ppw})] , \tag{3.8}$$

in which $P(w_i \,|\, pw)$ is the bigram probability, t_{cw} is the most likely part-of-speech tag for the current position that can be associated with the candidate word, w_i, and can be found using the following formula:

$$t_{cw} = argmax_{t_{ij}} \{P(t_{ij} \,|\, t_{pw} , t_{ppw}) \times P(w_i \,|\, t_{ij})\} .$$

One of the important issues in this approach is to determine the value of α. It is not very easy to determine what weight should be assigned to any of the prediction factors. Therefore, we find the value of α experimentally (see Chapter 5).

3.4 Adaptation

No matter what algorithm a prediction system uses, there are techniques for making predictions more accurate and more appropriate. These effective techniques are generally called *adaptation* and can be used along with any prediction algorithm to improve the accuracy of predictions.

There are various approaches to incorporate adaptation in a word-prediction system. Some of the systems may use an *adaptive lexicon*, which updates the words' frequency information according to the user's typing behaviour; i.e., the statistics collected from the training corpus are considered as basic frequency information that can be changed for every specific user. One way to update the statistics about words is to associate a user frequency to each word or sequence of words. The user frequency of a word or word sequence is increased each time the user enters that word (or word sequence). Words with higher user frequencies are preferred to those with lower frequencies. Different prediction systems may have different algorithms to decide how to choose the words according to both their base frequency (collected from the training corpus) and user frequency.

For example, WordQ prediction module assigns a user frequency to each word, in addition to the frequencies collected from a training corpus. The user frequency is zero for all words at the beginning. Every time the user uses a word, its user frequency is incremented. In a similar way, a user frequency is also associated with each word pair. Words or word pairs with non-zero user frequency are preferred over words with zero user frequency.

An adaptive lexicon may also use *recency information.* Recency information indicates how recently a word has been used and therefore increases or decreases the likelihood of it being used again in a similar context. Incorporating the recency information into the prediction module may be accomplished in different ways. One way to do this is to have a recency count for each word (or word sequence), and increment it each time a word is used within the desired context. Each document or each sequence of N words within the document may be considered as new contexts. An algorithm is also needed to determine how words should be selected according to both their base frequency and recency count.

For example, the dictionary of WordQ adapts to the users' vocabulary. Each user begins with a small dictionary of words. As the user enters new words, which are not in this lexicon, the system adds them as long as they can be found in the master dictionary in order to check the spelling. In this way, the predictions are adapted to the user's preferences, resulting in more appropriate predictions for the user.

Another way to adapt to the user's preferences is to include several topic-oriented lexicons other than the base lexicon, each containing vocabulary related to a specific topic. The users can then choose their desired topic. The advantage of having several lexicons is that it restricts the range of words to choose from and therefore increases the possibility of predicting the appropriate words. In WordQ it is possible for the users to select a topic of their preference, which also helps the predictions to be more appropriate for the users.

In our prediction algorithms we do not implement any of the above methods, but they can easily be incorporated into our system. The only adaptation technique that we implement is to remove repeated predictions. In this way, we do not predict those words that have been already rejected by the user for the same position. More details about this technique and the results achieved by incorporating it into the prediction system can be found in Chapter 5.

Chapter 4

Data collection

4.1 The British National Corpus

We used the World Edition of the British National Corpus (BNC) to train and test our word-prediction algorithms. This corpus was also used to test WordQ and compare its performance with the performance of other proposed predictors. BNC is a corpus composed of text samples from different eras and different genres, generally no longer than $45,000$ words. It contains a mixture of both spoken and written language, which are substantially the product of the speakers of British English. It includes about 100 million words, in total: 10% spoken and 90% written language, see (BNC manual, 2000). Table 4.1 presents the domains and the percentage of texts in each domain in BNC.

The BNC texts use the *reference concrete syntax* of SGML in which the beginning and the end of each element are marked by appropriate labels. The corpus consists of word labels, $< w >$, and punctuation labels, $< c >$, grouped into segment elements specified by segment labels, $< s >$. Each segment contains a portion of written or spoken text, more or less equivalent to a sentence, identified by the CLAWS segmentation scheme, a part-of-speech tagger used to tag the BNC.

Table 4.1: Domains and the percentage of BNC texts in each.

Domain	texts	words	percent
Applied Science	370	7,104,635	8.14
Arts	261	6,520,634	7.47
Belief and thought	146	3,007,244	3.44
Commerce and finance	295	7,257,542	8.31
Imaginative	477	16,377,726	18.76
Leisure	438	12,187,946	13.96
Natural and pure science	146	3,784,273	4.33
Social science	527	13,906,182	15.93
World affairs	484	17,132,023	19.62

4.1.1 CLAWS part-of-speech tagger

The Constituent Likelihood Automatic Word-tagging System (CLAWS), is a part-of-speech tagger for English with accuracy of around 96 – 97%. The latest version of the tagger, CLAWS4, and the C5 tagset were used to tag the BNC World Edition. C5 is a small tagset of 61 tags. This tagset is small because it was developed to annotate very large corpora. The description of each tag in the tagset is presented in Table 4.2.

A word labeled by the markup label, $< w >$, is a grammatical word according to the CLAWS definition of a word. An attribute, called **type**, is assigned to each word by the CLAWS tagger, and determines the word syntactic category according to CLAWS tag set. A punctuation character is also specified by a **type** attribute, which indicates the class assigned to it by the CLAWS system.

There are 30 special tags generated by the CLAWS tagger when annotating the BNC texts. These are called *ambiguity tags*, since each of them is a combination of two basic tags. CLAWS assigns an ambiguity tag to a word when it cannot decide which one is the correct tag. A list of the ambiguity tags and their interpretation is presented in Table 4.3.

4.1.2 Pre-processing the corpus

In each of the training or test steps, a specific format for the training texts is required by utility programs collecting statistical information or calculating performance measures. Therefore, various filtering are needed on the original BNC texts to make them appropriate as input to each one of the utility programs. The pre-processing steps are explained in the following sections.

Table 4.2: C5 tagset used to tag the BNC

Tag	Description
PUN	punctuation: general separating mark, i.e., ..., !, . : ; , - ?
PUQ	punctuation: quotation mark, i.e., ' or "
PUL	punctuation: left bracket, i.e., [or (
PUR	punctuation: right bracket, i.e.,] or)
AJ0	adjective (unmarked) (e.g. GOOD, OLD)
AJC	comparative adjective (e.g. BETTER, OLDER)
AJS	superlative adjective (e.g. BEST, OLDEST)
AT0	article (e.g. THE, A, AN)
AV0	adverb (unmarked) (e.g. OFTEN, WELL, LONGER, FURTHEST)
AVP	adverb particle (e.g. UP, OFF, OUT)
AVQ	wh-adverb (e.g. WHEN, HOW, WHY)
CJC	coordinating conjunction (e.g. AND, OR)
CJS	subordinating conjunction (e.g. ALTHOUGH, WHEN)
CJT	the conjunction THAT
CRD	cardinal numeral (e.g. 3, FIFTY-FIVE, 6609) (excl ONE)
DPS	possessive determiner form (e.g. YOUR, THEIR)
DT0	general determiner (e.g. THESE, SOME)
DTQ	wh-determiner (e.g. WHOSE, WHICH)
EX0	existential THERE
ITJ	interjection or other isolate (e.g. OH, YES, MHM)
NN0	noun (neutral for number) (e.g. AIRCRAFT, DATA)
NN1	singular noun (e.g. PENCIL, GOOSE)
NN2	plural noun (e.g. PENCILS, GEESE)
NP0	proper noun (e.g. LONDON, MICHAEL, MARS)
NULL	the null tag (for items not to be tagged)
ORD	ordinal (e.g. SIXTH, 77TH, LAST)
PNI	indefinite pronoun (e.g. NONE, EVERYTHING)
PNP	personal pronoun (e.g. YOU, THEM, OURS)
PNQ	wh-pronoun (e.g. WHO, WHOEVER)

Table 4.2: C5 tagset used to tag the BNC

PNX	reflexive pronoun (e.g. ITSELF, OURSELVES)
POS	the possessive (or genitive morpheme) 'S or '
PRF	the preposition OF
PRP	preposition (except for OF) (e.g. FOR, ABOVE, TO)
TO0	infinitive marker TO
UNC	"unclassified" items which are not words of the English lexicon
VBB	the "base forms" of the verb "BE" (except the infinitive), i.e. AM, ARE
VBD	past form of the verb "BE", i.e. WAS, WERE
VBG	*-ing* form of the verb "BE", i.e. BEING
VBI	infinitive of the verb "BE"
VBN	past participle of the verb "BE", i.e. BEEN
VBZ	*-s* form of the verb "BE", i.e. IS, 'S
VDB	base form of the verb "DO" (except the infinitive), i.e.
VDD	past form of the verb "DO", i.e. DID
VDG	*-ing* form of the verb "DO", i.e. DOING
VDI	infinitive of the verb "DO"
VDN	past participle of the verb "DO", i.e. DONE
VDZ	*-s* form of the verb "DO", i.e. DOES
VHB	base form of the verb "HAVE" (except the infinitive), i.e. HAVE
VHD	past tense form of the verb "HAVE", i.e. HAD, 'D
VHG	*-ing* form of the verb "HAVE", i.e. HAVING
VHI	infinitive of the verb "HAVE"
VHN	past participle of the verb "HAVE", i.e. HAD
VHZ	*-s* form of the verb "HAVE", i.e. HAS, 'S
VM0	modal auxiliary verb (e.g. CAN, COULD, WILL, 'LL)
VVB	base form of lexical verb (except the infinitive)(e.g. TAKE, LIVE)
VVD	past tense form of lexical verb (e.g. TOOK, LIVED)
VVG	*-ing* form of lexical verb (e.g. TAKING, LIVING)
VVI	infinitive of lexical verb
VVN	past participle form of lex. verb (e.g. TAKEN, LIVED)
VVZ	*-s* form of lexical verb (e.g. TAKES, LIVES)
XX0	the negative NOT or N'T
ZZ0	alphabetical symbol (e.g. A, B, c, d)

Step one: removing unnecessary SGML markup The only elements from the BNC texts needed in our experiments are words, punctuation marks, and their part-of-speech tags. Therefore, before starting any training we remove unnecessary elements from the texts. These include all SGML markup other than word, punctuation, and sentence boundary labels. Output of this filtering process includes words and punctuation marks and their part-of-speech tags. In the resulting texts, each sentence is delimited between two SGML labels, $< s >$ and $< /s >$, and is written on a separate line.

Step two: removing part-of-speech tag information In order to build n-gram models for words, we need BNC texts which contain only words, but not their part-of-speech tags. Therefore, we remove all part-of-speech tag information from the previously filtered texts. The output of this step is used as input by programs that collect n-gram statistics for words.

Step three: removing the words The syntactic predictors introduced in Chapter 3 use part-of-speech tag statistics in order to suggest a list of words to the user. In order to collect n-gram statistics for part-of-speech tags, we need another version of the training texts in which the words are removed, but their part-of-speech tags remain. We have written a script that reads in the training texts and extracts all part-of-speech tags.

4.2 Training: collecting the statistics

A portion of the BNC, about 81 million words chosen from the written-language section, is used to train the Unigram, Bigram and the syntactic predictors. The training data used for training WordQ is a text sample of around 5.8 million words collected from the Internet. The training phase mostly includes collecting various types of statistical information used by the predictors.

Different predictors use different statistical information to give suggestions to the user. As explained in Chapter 3, the predictors exploit both word and part-of-speech tag language models, along with word-given-tag probabilities.

Table 4.3: List of the ambiguity tags.

Ambiguity tag	Ambiguous between	More probable tag
AJ0-NN1	AJ0 or NN1	AJ0
AJ0-VVD	AJ0 or VVD	AJ0
AJ0-VVG	AJ0 or VVG	AJ0
AJ0-VVN	AJ0 or VVN	AJ0
AV0-AJ0	AV0 or AJ0	AV0
AVP-PRP	AVP or PRP	AVP
AVQ-CJS	AVQ or CJS	AVQ
CJS-AVQ	CJS or AVQ	CJS
CJS-PRP	CJS or PRP	CJS
CJT-DT0	CJT or DT0	CJT
CRD-PNI	CRD or PNI	CRD
DT0-CJT	DT0 or CJT	DT0
NN1-AJ0	NN1 or AJ0	NN1
NN1-NP0	NN1 or NP0	NN1
NN1-VVB	NN1 or VVB	NN1
NN1-VVG	NN1 or VVG	NN1
NN2-VVZ	NN2 or VVZ	NN2
NP0-NN1	NP0 or NN1	NP0
PNI-CRD	PNI or CRD	PNI
PRP-AVP	PRP or AVP	PRP
PRP-CJS	PRP or CJS	PRP
VVB-NN1	VVB or NN1	VVB
VVD-AJ0	VVD or AJ0	VVD
VVD-VVN	VVD or VVN	VVD
VVG-AJ0	VVG or AJ0	VVG
VVG-NN1	VVG or NN1	VVG
VVN-AJ0	VVN or AJ0	VVN
VVN-VVD	VVN or VVD	VVN
VVZ-NN2	VVZ or NN2	VVZ

N-gram language models, both for words and part-of-speech tags, are provided for the experiments by using the Carnegie Mellon University-Cambridge Statistical Language Modeling toolkit. Word-tag statistics are collected by a utility Perl program developed for this project. Both the statistical language modeling toolkit and the utility Perl program are explained in the following sections.

4.2.1 Statistical Language Modeling toolkit

The CMU-Cambridge Statistical Language Modeling toolkit is a suite of Unix software tools developed to make the task of language modeling easier, see (SLM documentation, 2001; Clarkson and Rosenfeld, 1997). Some of the programs are used to process general textual data into word frequency lists and vocabularies, word bigram and trigram counts, various backoff bigram and trigram language models, etc. Some others can be used to compute some measures over the resulting language models, such as perplexity and Out Of Vocabulary (OOV) rate.

We use the toolkit to collect the frequency of English words, word sequences, part-of-speech tags, and tag sequences from the training texts.

4.2.2 N-gram models for words

The vocabulary produced by the SLM toolkit programs contains about 65,000 most-frequent words. Using this vocabulary and the original texts, unigram and bigram models for words are constructed. The output file contains 65,001 unigrams (65,000 vocabulary words, and one symbol for all OOVs), and 7,053,433 bigrams (see Table 4.4). It also includes the backoff unigram and bigram models used in cases when one of the probabilities does not exist. According to the model created by the SLM toolkit, the conditional probability of a word given its previous word is defined to be:

$$P(w_2 \,|\, w_1) = \begin{cases} P(w_2, \,|\, w_1) & \text{if bigram exists} \\ P_{backoff}(w_1) \times P(w_2) & \text{otherwise} \end{cases},$$

where $P_{backoff}(w_1)$ is the backoff probability of the word w_1, collected from the training corpus by the SLM toolkit.

Using the toolkit options, we created an open-vocabulary model which allows for OOVs to occur, but all of them are mapped to the same symbol. In our training, we treat this symbol the same way as any other word in the vocabulary. The Good-Turing estimator is used as a discounting strategy to smooth the probabilities collected from the training data.

Table 4.4: Size of word and part-of-speech tag n-gram models.

Model	Size (elements)
word unigram	65,001
word bigram	7,053,433
tag unigram	80
tag bigram	5,257
tag trigram	132,285

4.2.3 N-gram models for part-of-speech tags

The same toolkit programs are used to build unigram, bigram, and trigram models for part-of-speech tags. The vocabulary in this case contains 79 part-of-speech tags, 61 basic tags along with 18 of the 30 ambiguity tags. The other 12 ambiguity tags have been replaced by the most probable tag (the first tag). These are mostly the ones that are frequently repeated in the training corpus and/or the two tags are in two different part-of-speech categories, such as *VVZ-NN2*. The output file includes 80 part-of-speech tag unigrams, 5,257 bigrams, and 132,285 trigrams (see Table 4.4). The discounting strategy is the same as the one exploited in building word language models, i.e., Good-Turing.

Similar to the n-gram models for words, the conditional probability of a part-of-speech tag given its two previous tags is defined to be:

$$P(t_3 \,|\, t_1,\, t_2) = \begin{cases} P(t_3,\, |\, t_1,\, t_2) & \text{if trigram exists} \\ P_{\textit{backoff}}(t_1,\, t_2) \times P(t_3 \,|\, t_2) & \text{if bigram}(t_1,\, t_2) \text{ exists} \\ P(t_3 \,|\, t_2) & \text{otherwise} \end{cases}$$

where $P_{\textit{backoff}}(t_1,\, t_2)$ is the backoff probability of the part-of-speech tag pair $(t_1,\, t_2)$, collected from the training corpus by the SLM toolkit.

4.2.4 Word-given-tag probabilities

The prediction module of the syntactic predictors exploits word and part-of-speech tag n-gram statistics, along with word-given-tag probabilities. We define word-given-tag probability to be the probability of having the word w, given that the tag of the word is t, i.e., $P(w \,|\, t)$. We also define the tag-given-word probability to be the probability of assigning the tag t to the word w by a part-of-speech tagger, in our case, the CLAWS part-of-speech tagger. Using Bayes's formula, we can compute each one of these probabilities, having the other one:

$$P(w \,|\, t) = P(t \,|\, w) \times \frac{P(w)}{P(t)},$$

in which $P(w)$ and $P(t)$ are the word unigram and tag unigram probabilities, respectively. Therefore, we compute the tag-given-word probability from our training data. Since we have the unigram probabilities both for words and part-of-speech tags, if we have tag-given-word probabilities, we can calculate word-given-tag probabilities at the time of need. In order to collect tag-given-word probabilities, we have written a Perl script that reads through training texts, and computes the desired probabilities for every word which is in the vocabulary constructed by SLM.

Chapter 5

Evaluation

5.1 Experimental methodology

We conducted several experiments to see how accurate the predictions are that are provided by each of the word-prediction algorithms introduced in Chapter 3, and to compare them with WordQ.

In order to do a thorough test of various aspects of the algorithms, we needed to select appropriate *test data* (Section 5.1.1), a *test procedure* (Section 5.1.2), a set of *parameters* that may affect the results (Section 5.1.3), and a number of *performance measures* for evaluation purposes (Section 5.1.4).

5.1.1 Test data

A small subset of the BNC (see Chapter 4), around 27 text files and around one million words, was chosen for evaluation. In selecting texts, our attempt was to choose texts from various domains. The name, size, and the domain of these texts are presented in Table 5.1. Most of the files contain between 30,000 and 40,000 words (the average is around 35,256), but there are some which contain less or more than this.

All the test texts were pre-processed and converted to the required format. The pre-processing steps were very similar to those exploited for the training data (see Chapter 4).

Table 5.1: List of the BNC texts used for testing the prediction algorithms.

Filename	Words	Domain
AR3	36433	Imaginative
AS0	30887	Applied science
AT8	34669	Commerce and finance
ARF	37153	Natural and pure science
ARG	35217	Belief and thought
BN3	34758	World affairs
BP0	40588	Imaginative
BPK	39983	Social science
CS2	38443	Belief and thought
CS6	33887	World affairs
CS7	32413	Social science
CTX	49013	Applied science
EX0	33825	Leisure
EW6	32327	Natural and pure science
EWA	43998	Arts
FUA	34467	Social science
FYS	36799	Commerce and finance
FYX	39109	Applied science
GX1	14792	Belief and thought
GX8	14365	Natural and pure science
HWB	42841	Arts
HWE	33001	Imaginative
HX8	41219	Leisure
HX4	32091	Commerce and finance
J2K	36715	Arts
J2L	37959	World affairs
K2U	34980	Leisure
Total	951,932	

5.1.2 Test procedure

We designed and implemented a *prediction testbench* in order to perform various tests with different prediction algorithms and different values for the test parameters. The testbench

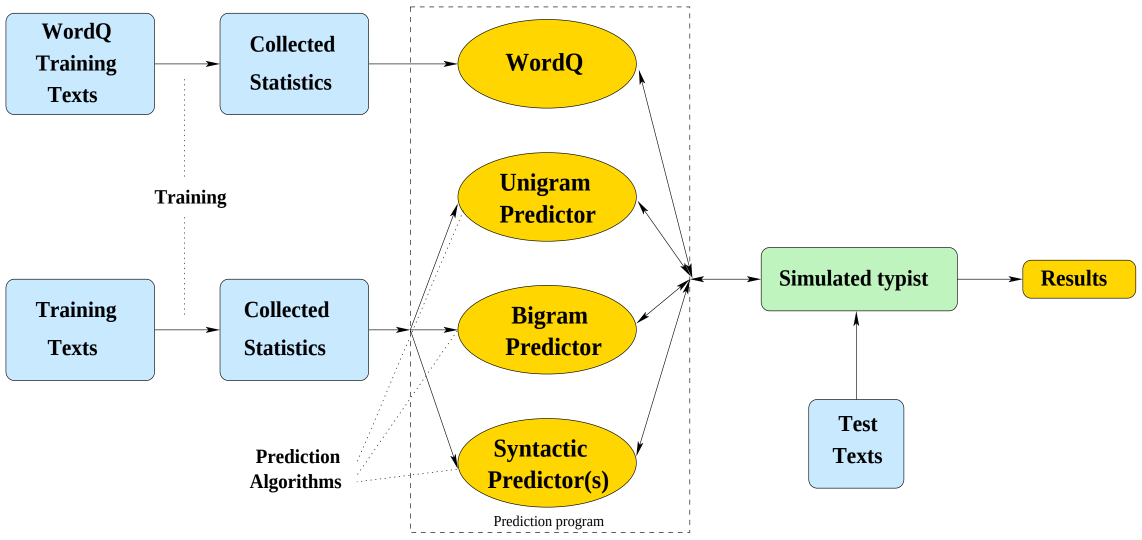


Figure 5.1: Architecture of the word-prediction testbench used in the experiments.

has three major components: a *simulated user* that is used to inject the characters into the prediction algorithm, a *prediction program* that can use any of the prediction algorithms to suggest appropriate words to the user, and a *database of statistics* collected from the training corpus. In Figure 5.1 the general architecture of the prediction testbench is shown.

In each experiment, the virtual user types in one of test texts letter by letter. The prediction program then suggests n words (usually between 1 and 10 in our experiments) to the user. The next step is to determine whether the user's desired word (in our case the correct prediction according to the test text) is among the predictions, and calculate the measures accordingly.

The virtual user is actually a program that simulates a user who types in the text, and is described in more detail in the following section. The prediction program is the program that gives suggestions to the user according to the given context, the prediction algorithm incorporated, and the statistical information. This module is explained in more detail in Section 5.1.2.2.

5.1.2.1 User simulation

The virtual or simulated user is a Perl program that reads in each test text letter by letter. After reading each letter, it determines what the correct prediction for the current (or next) position is. The prediction program then is called and a list of suggestions is returned to the user. The user searches the prediction list for the correct prediction. If it is found in the list, the user increases the number of correct predictions by the predictor, and calculates some other measures accordingly. The correctly predicted word is then completed and the user continues to read the rest of the text. If the correct word is not in the list of predictions, the user increments the number of cases in which the predictor cannot predict the correct word, and types in the next letter. This process continues until either the word is predicted by the predictor or the user completes the word.

The user is assumed to be a perfect user, i.e. if the correct word is in the prediction list, the user does not miss it. This is not always true in the real world when a prediction system is used by an individual with disabilities. There might be times when the users do not find their desired words in the list although they are among the suggestions. This depends on various factors, such as the number of words in the prediction list, the amount of cognitive load imposed on the user by the system, and the type and level of the users' disabilities.

5.1.2.2 Prediction program

The prediction program is a C program which is implemented in the Linux environment and in which any one of the prediction algorithms (or any other new algorithm) can easily be plugged for evaluation purposes.

Each time the user calls the prediction program, the prediction algorithm attempts to find the requested number of suggestions according to the given contextual information, such as previous words and/or their part-of-speech tags. It then gives a list of suggestions to the user. After entering each new letter from the test text, the prediction program updates the prediction list according to the new context. This program uses the statistics database, described in the next section, for the prediction task.

As mentioned in previous paragraphs, the prediction program is implemented in the Linux environment. In order to test each of the prediction algorithms, they should be plugged into the prediction program one by one. Thus, in order to measure the performance of the WordQ prediction algorithm, we needed to port WordQ from Windows into Linux. WordQ has several modules other than the prediction module, such as speech synthesis which we did not port. Therefore, it was also required to make small modifications to WordQ (without changing the prediction algorithm, of course) to make it work consistently with our prediction program.

5.1.2.3 Statistics database

A large amount of statistical information is used by the word-prediction algorithms. This information includes word and part-of-speech tag n-gram probabilities collected from the training data, and also a dictionary of word-tag probabilities. Our training data is very large (around 81 million words). Therefore, as can be seen in Table 4.4, n-gram models (especially the word bigram model) are large as well. This makes it difficult to store and access the information by the prediction program. In order to enhance the efficiency of the prediction program, several data structures have been used, which are explained in more detail in the following paragraphs.

Hash tables As mentioned in the previous paragraph, the statistical information used by the prediction algorithms is very large. The access rate to this information is high, especially because the test texts contain around one million words, in total. Also, we know that both in our experiments and in a real-world prediction system, the process of suggesting words to the user needs to be as fast as possible. Therefore, the statistical information used by the prediction algorithm should be maintained in main memory.

Due to the large size of data that is used during the prediction process and the fact that all data should be stored in main memory, it is necessary to find an efficient method of storing them, both in terms of memory space and memory access time. The data structures that we have used to manipulate the statistical information are a number of hash tables, one for each n-gram model.

Caches Since we test the predictors on a large test data, the running time of each experiment is an important factor. Thus, we use several caches to store previous predictions and use them wherever possible. The caches are good specially because they prevent us from repeating the time-consuming prediction process, when we can just retrieve the suggestions from a cache. This idea can also be applied to real-world prediction systems, in which it is important to provide the users with predictions within a reasonable response time.

5.1.3 Test parameters

There are three parameters that may affect the performance of the predictors. These parameters are explained in more detail in the following paragraphs. In order to measure the effect of each parameter on the performance, several experiments were conducted. The results of these experiments can be found in Section 5.2.5.

- Coefficient α

  α is the coefficient of the linear combination of two predictors in Formula 3.8 (see Chapter 3). Choosing a value for this parameter is not an easy task. It can range from 0 to 1, giving more weight to either of the predictors. The best value for this parameter must be found experimentally (see Section 5.2.5.1).

- Repetition of suggestions in consecutive predictions

  According to the assumption of having a perfect user, we can avoid repeating

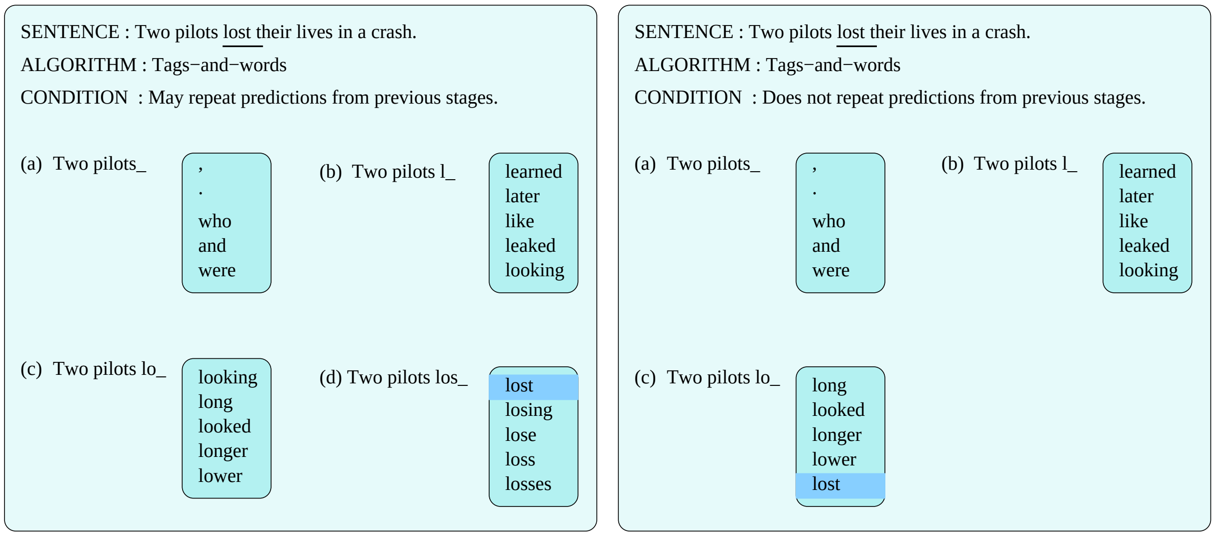


Figure 5.2: Sample predictions given by the Tags-and-words predictor when $n = 5$ for both conditions: repeating previously predicted words or not.

suggestions that have been previously rejected by the user for the same position. We hypothesize that if we avoid predicting the same words, the likelihood of having the appropriate word in the future suggestion lists will be higher.

An example of predictions by the Tags-and-words predictor in each situation is shown in Figure 5.2.

In order to evaluate the effect of this factor, we leave it as a test parameter. Thus, we can compare the results in both conditions: avoiding repetition, or not doing so. The results can be found in Section 5.2.5.3.

- Maximum number of words in the prediction list (n)

  It is clear that the higher the number of words in the prediction list, the higher the probability of having the appropriate prediction among the suggestions. But, larger values for n impose a cognitive load on the user since it makes the search time for the correct word longer. Therefore, it is more likely that the user misses the correct prediction. In a real-world prediction system, this parameter can be introduced as an option for the user. We selected the values 1, 5, and 10 for n to measure the effect of this parameter on the performance of prediction algorithms. The results can be found in Section 5.2.5.2.

5.1.4 Performance measures

In order to measure the performance of each word-prediction algorithm and compare them with each other, a number of measures are needed. The measures should be selected such that they consider various aspects of the performance in a prediction system. *Hit rate*, *keystroke savings*, and *keystrokes until prediction* are the three main measures that are explained in the following paragraphs.

- Hit rate

  Hit rate is the percentage of the times that the correct word appears in the prediction list. The algorithm that has a higher hit rate is a better algorithm compared to the others.

- Keystroke savings

  Keystroke savings refers to the percentage of the keystrokes that the user saves by using a word-prediction tool. The number of keystrokes that a user saves for each correctly predicted word is equal to the number of remaining letters in the word plus a space that is automatically inserted after the word by the predictor, minus one keystroke needed for choosing the appropriate prediction. According to this measure, the better algorithm is the one that results in higher keystroke savings.

- Keystrokes until prediction

  Keystrokes until prediction is defined to be the average number of keystrokes that the user must enter before the appearance of the correct word in the prediction list. The algorithm that gives a lower value for this measure is considered to be a better algorithm.

There are also other measures that are calculated during the experiments and can be used for specific purposes. These measures are as follows:

- Percentage of the words predicted at some point (accuracy)

  This measure shows the proportion of words that could be predicted by the predictor before they were completed by the user. This measure may also be called *accuracy* of the predictor. This cannot be used as a major measure to compare two prediction algorithms by itself, because a good predictor is one that not only predicts most of the words, but also can predict them in early stages. For example, a predictor that predicts all of the words right before the last letter of the word is entered by the user has an accuracy of 100%, but its hit rate and keystroke savings are small.

- Average number of suggestions at each position

  This measure can be used to determine whether a prediction algorithm can provide the user with a sufficient number of predictions at each position. The closer the average to n (the maximum number of predictions in the list), the better the algorithm. This measure can be used to find an appropriate value for n. If the average is much less than the maximum, then it means that this value of n is too high, such that the algorithm cannot even find sufficient words to suggest.

- Average length of predicted words

  This measure shows whether the predictor is able to predict very long words only, words of average length, or also short words. It helps in building dictionaries to enhance the performance of word prediction according to the algorithm and the users' needs.

5.2 Results

The remainder of this chapter contains the results of various experiments, comparisons, and discussion. For comparison purposes, several series of experiments were conducted. In each of them, either the prediction algorithm was varied among WordQ, Unigram, Bigram, and syntactic predictors, or one of the parameters was given different values to see the effect.

5.2.1 Major performance measures

In this section, the three major performance measures, i.e., hit rate, keystroke savings, and keystrokes until prediction, are analyzed. First, for a general comparison between the algorithms, we set $n = 5$ and we assume that the predictions may be repeated in consecutive positions. The results are shown in Table 5.2 and Figure 5.3.

Table 5.2: Values of hit rate, keystroke savings, and keystrokes until prediction for $n = 5$ when predictions may be repeated.

Algorithm	%Hit rate	%Keystroke savings	Keystrokes until prediction
WordQ	27.10	42.54	2.09
Unigram predictor	32.14	45.70	1.92
Bigram predictor	37.43	52.90	1.55
Part-of-speech tag predictor	34.93	49.80	1.72
Linear predictor($\alpha = .8$)	37.78	53.14	1.54
Tags-and-words predictor	37.49	53.30	1.53

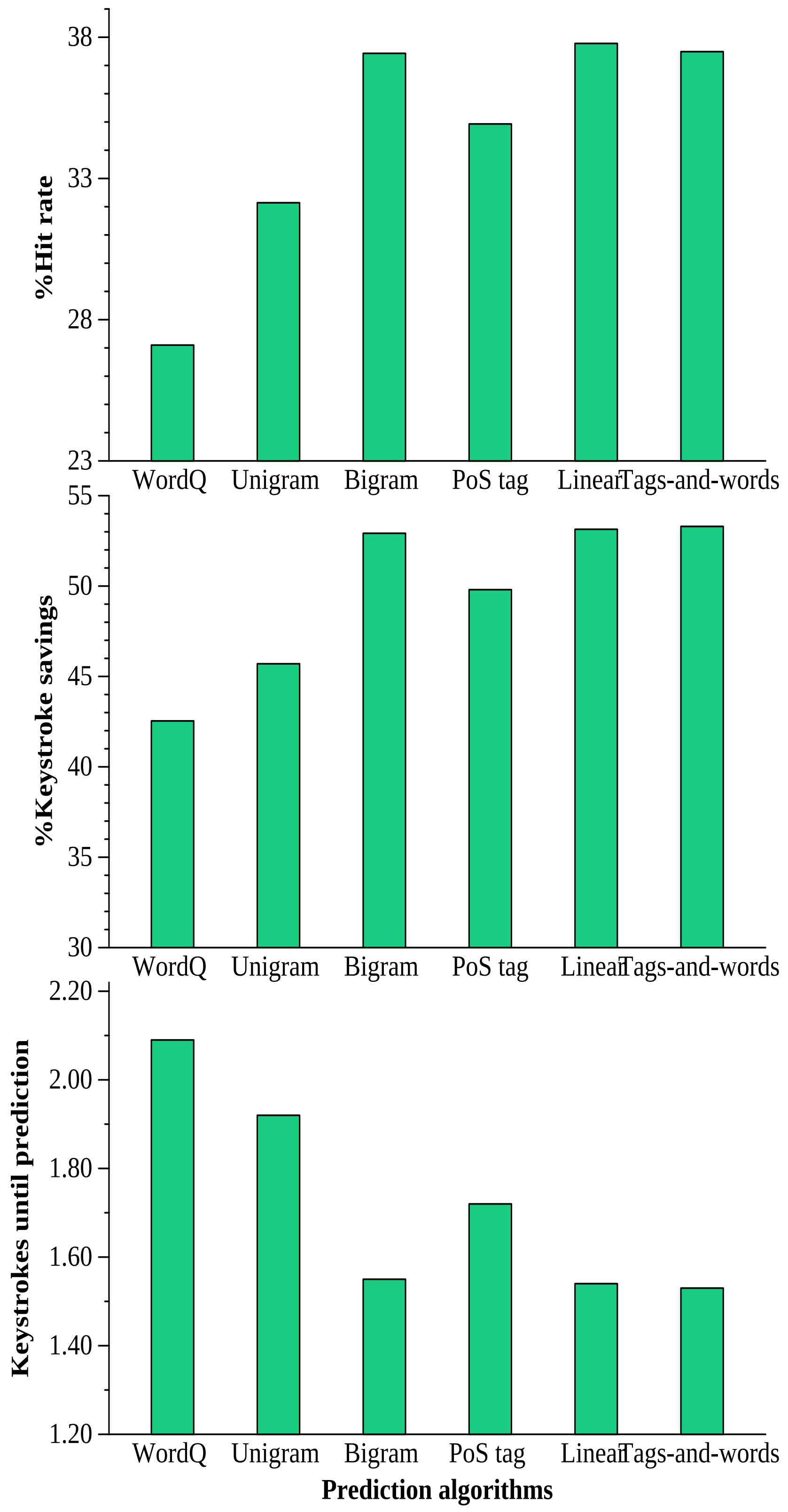


Figure 5.3: Comparison of hit rate, keystroke savings, and keystrokes until prediction among all prediction algorithms when $n = 5$ and predictions may be repeated).

5.2.2 Discussion

Although the WordQ prediction algorithm is similar to the Bigram predictor, it has the lowest value for hit rate and keystroke savings, even lower than the Unigram predictor, and the highest value for keystrokes until prediction. There are two major problems in comparing WordQ with other algorithms according to the results of experiments only.

- As mentioned in Chapter 4, the training data used for training WordQ is a text sample of around 5.8 million words collected from the Internet, while the training data we used in training all the other five algorithms is a subset of the BNC (around 81 million words). The word unigram model used by the WordQ prediction module contains 62,669 words, and the bigram table includes 1,302,731 entries. Thus, the word bigram models that are used by the WordQ prediction and the ones we used in our testbench, are very much different in terms of size.

- The training and the test data for WordQ are not necessarily from the same genres, but in all other algorithms the training and test data are distinct subsets of the BNC.

The above factors, i.e., the differences in size and genres of the two training sets used for training WordQ and the other predictors, may lead to some scenarios that cause the performance of WordQ to be worse than our Unigram and Bigram predictors. The WordQ prediction module uses word frequency (unigram) wherever the required bigram does not exist. Since the bigram model of WordQ is small and because the training and test data are not strongly similar, this scenario might happen quite often, resulting in using the unigram model for finding suggestions in most cases. If we accept that the unigram model in our predictors is more accurate than the unigram model of WordQ (both in terms of general coverage and similarity between training and test), it would be clear why WordQ works even worse than our Unigram predictor.

The training data used in training WordQ is a corpus of Canadian English in which a number of words have different spelling compared to their spellings in the BNC. Also, there are a number of local words, such as names of people and places, in the corpus which might be completely useless for testing on the BNC.

Therefore, we cannot actually compare the results of other prediction algorithms with the results of WordQ prediction algorithm without using the same or at least similar training data. Several experiments were accomplished by (Nantais et al., 2001) in which the keystroke savings of WordQ were measured on three test texts similar to the ones used for training. The results show that without adaptation and for $n = 5$, the average keystroke savings of WordQ is 45.3% (2.76% higher compared to 42.54% in our experiments).

The Part-of-speech tag predictor, which uses only part-of-speech tag information to predict words (see Chapter 3, Formula 3.1), performs better than the Unigram predictor, but worse than the Bigram predictor. This means that two previous part-of-speech tags are more predictive than the frequency of individual words only, but less predictive than the word bigram.

When we combine the Bigram predictor and a slightly different version of the Part-of-speech tag predictor by linear combination (see Chapter 3, Formula 3.8), the predictor outperforms both the Part-of-speech tag and the Bigram predictors.

The Tags-and-words predictor (see Chapter 3, Formula 3.7) outperforms all other predictors with a slight improvement in keystroke savings and keystrokes until prediction (but not hit rate).

5.2.3 Statistical significance tests

We used MATLAB to perform a one-way ANOVA test with Tukey-Kramer post-test and a 95% confidence interval (confidence level of 0.05) on the results of all predictors for the keystroke savings measure, when $n = 5$ and predictions may be repeated in consecutive positions.

Our test data consists of 27 text segments (each segment is simply a BNC text). We ran each experiment on all texts and calculated the mean and standard deviation for keystroke savings for the purpose of ANOVA test. The test shows that:

- All the improvements over WordQ are statistically significant at $p < 0.001$.
- Among the three algorithms Bigram, Linear, and Tags-and-words, the difference is not statistically significant ($p > 0.05$).
- All other improvements are statistically significant (Unigram compared to Bigram and all syntactic predictors; Part-of-speech tag predictor compared to Bigram, Linear, and Tags-and-words predictors).

5.2.4 Observations

Large word bigram model As mentioned in previous sections, we used a very large set of training data to build n-gram models for words and part-of-speech tags. Thus, the word bigram model that is used by the Bigram predictor is a very strong model that also captures the syntax of the word pairs. The sentences in the training data are mostly grammatically and syntactically correct sentences. Therefore, if we build a model that contains the probabilities of word pairs appearing in these texts, it would partially capture the syntactic relationship of the word pairs as well. For example, the probability of having a past-tense verb (VVD) after another of the same category is low. Thus, the bigram probability of having a sequence of two tags of this form is low. Similarly, every word bigram containing two past-tense verbs (such as *took*, *lived*) would have a low probability.

Banko and Brill (2001) have shown that for the specific task of confusion set disambiguation, using a very large corpus (around 1 billion words) has a higher effect than using a smarter and stronger classifier.

Thus, the overlap between the quantity of the information included in the word bigram model and the part-of-speech tag trigrams might be high. It is possible to do some experiments with a smaller subset of the BNC as training data to see if adding part-of-speech tag information would help when the word bigram model is not sufficiently large.

Conditional independence assumptions In Chapter 3, Section 3.3.2, we made some independence assumptions to be able to calculate the probability $P(w_i \,|\, pw,\, t_{pw},\, t_{ppw})$ by using the basic n-gram models for words and part-of-speech tags. Some of the random variables that we assumed to be conditionally independent of others might not be actually independent in reality.

For example, in estimating the above probability, we have assumed that t_{ij} and pw are conditionally independent given the tag of the previous word, t_{pw} (see the Bayesian network is Figure 3.3). Although this is a reasonable assumption to make, it is not clear how it may affect the results. If we did not make this independence assumption, it would have been necessary to calculate the probabilities $P(t_{ij} \,|\, pw)$ and $P(pw \,|\, w_i,\, t_{ij})$ directly from the corpus.

Errors in the CLAWS tagger The accuracy of the CLAWS tagger, used for tagging the BNC (see Chapter 4), is claimed to be around 96 – 97% (3 – 4% error). Also, we can see in the tagged corpus that there are many cases in which the tag associated with a word is ambiguous between two part-of-speech tags. This means that although we add a new source of information to the prediction algorithm (through addition of part-of-speech tags), at the same time we add more uncertainty to the predictions made using this new information source.

Granularity of the part-of-speech tags The C5 tagset used by the CLAWS tagger to tag the BNC contains around 61 tags. Looking at Table 4.2 reveals that there are distinctions between word classes that might not be important for the task of word prediction. It is possible that if we cluster the existing tags into a small number of categories (for example verbs, nouns, etc.) and use a coarser-grained set of tags, the part-of-speech tags would be more useful for prediction.

Separate n-gram models for words and part-of-speech tags All the prediction algorithms introduced in Chapter 3 use a combination of the basic word and/or part-of-speech tag n-gram models. If we can build n-gram models for word and tag together, it is possible that they would be more accurate predictors. For example, we might want to calculate the joint probability of having a word w_i and its part-of-speech tag t_i in the current position, given the previous words and their associated part-of-speech tags.

5.2.5 Parameters

In this section we investigate the impact of changing our test parameters on the performance of the selected algorithms. We tried to prune the potentially huge space of various experiments, since otherwise it would take too long to test all the predictors with different values of all the parameters. Thus, only one of the predictors is selected for most of the experiments in which the test parameters are involved: the Tags-and-words predictor. Only for determining the best value for coefficient α, the Linear predictor is chosen.

5.2.5.1 Coefficient

We performed several experiments with different values for α to choose the best value for it. A small set of development texts, around 36,000 words, was used in these experiments. The diagram in Figure 5.4 shows changes in keystroke savings when α ranges from 0 to 1. The changes for the other two measures, hit rate and keystrokes until prediction, were very similar to this (not shown). According to the experiments, the best value for α is

0.8. Thus, we have set α to 0.8 in all of the experiments in which the Linear predictor is involved.

However, choosing a constant value for α may not be the best way. The value of this parameter may change depending on the distribution of word unigram or bigram frequencies. It might be the case that for some words, with some distribution of word unigram or bigram frequencies, the part-of-speech tag has more effect. Therefore, the value of α should be such that it gives more weight to the second predictor (see Formula 3.8).

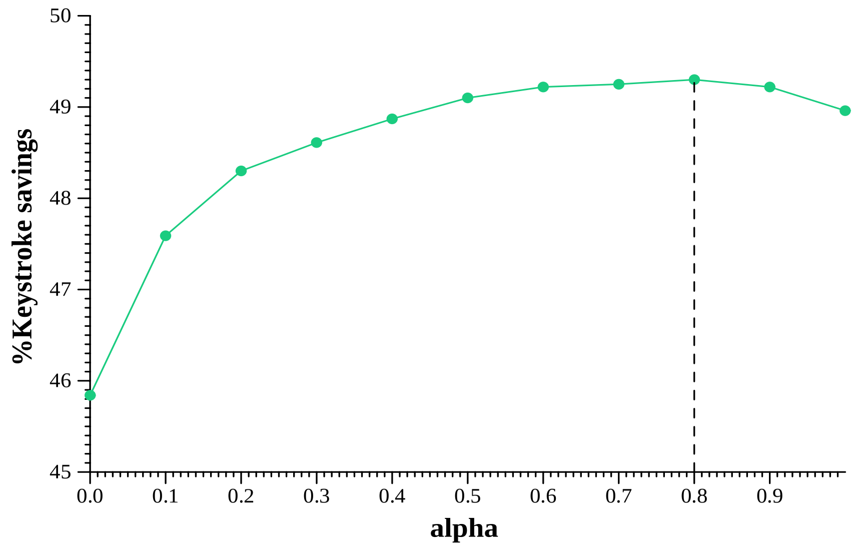


Figure 5.4: Keystroke savings of the Tags-and-words predictor for different values of α and $n = 5$ when predictions may be repeated.

5.2.5.2 Size of prediction list

The Tags-and-words predictor has the best values for keystroke savings and keystrokes until prediction measures (but not the best value for hit rate). Therefore, we selected this algorithm for an experiment in which we measure the effect of maximum prediction list size, n, on the performance. In this experiment, we chose three different values, 1, 5, and 10, for n and compared the results. The values for the three measures can be found in Table 5.3. Comparisons can be seen in Figure 5.5.

The results show that if the number of words in the prediction list size increases, the accuracy of the predictions will increase as well, resulting in higher hit rate and keystroke savings. However, choosing the right value for this parameter, especially in real-world prediction systems, depends on the type of users' disabilities. If the users have learning or cognitive disabilities, it is better to suggest fewer words to them. But if saving physical effort, i.e., saving as many keystrokes as possible, is a major gain for the user, suggesting more words (e.g., $n = 10$) would be a better choice. However, the best value of n depends on other factors, such as the average number of predictions in the list. If this average is considerably lower than the maximum, it shows that this is probably a high value for n. For example, in our experiment with the Tags-and-words predictor, the average size of the list is 9.28 which is close to maximum, i.e., 10.

Table 5.3: Effect of maximum prediction list size on performance of the Tags-and-words predictor when predictions are not repeated.

Size of prediction list	%Hit rate	%Keystroke savings	Keystrokes until prediction
$n = 1$	26.10	40.95	2.20
$n = 5$	38.50	54.37	1.47
$n = 10$	43.17	58.63	1.24

5.2.5.3 Repetition of previous predictions

We conducted an experiment using the Tags-and-words predictor as the prediction algorithm, with two different options for predictions: one with repeated predictions, another one without. The results for the three measures, hit rate, keystroke savings, and keystrokes until prediction, can be found in Table 5.4. A comparison of the values is represented in Figures 5.6.

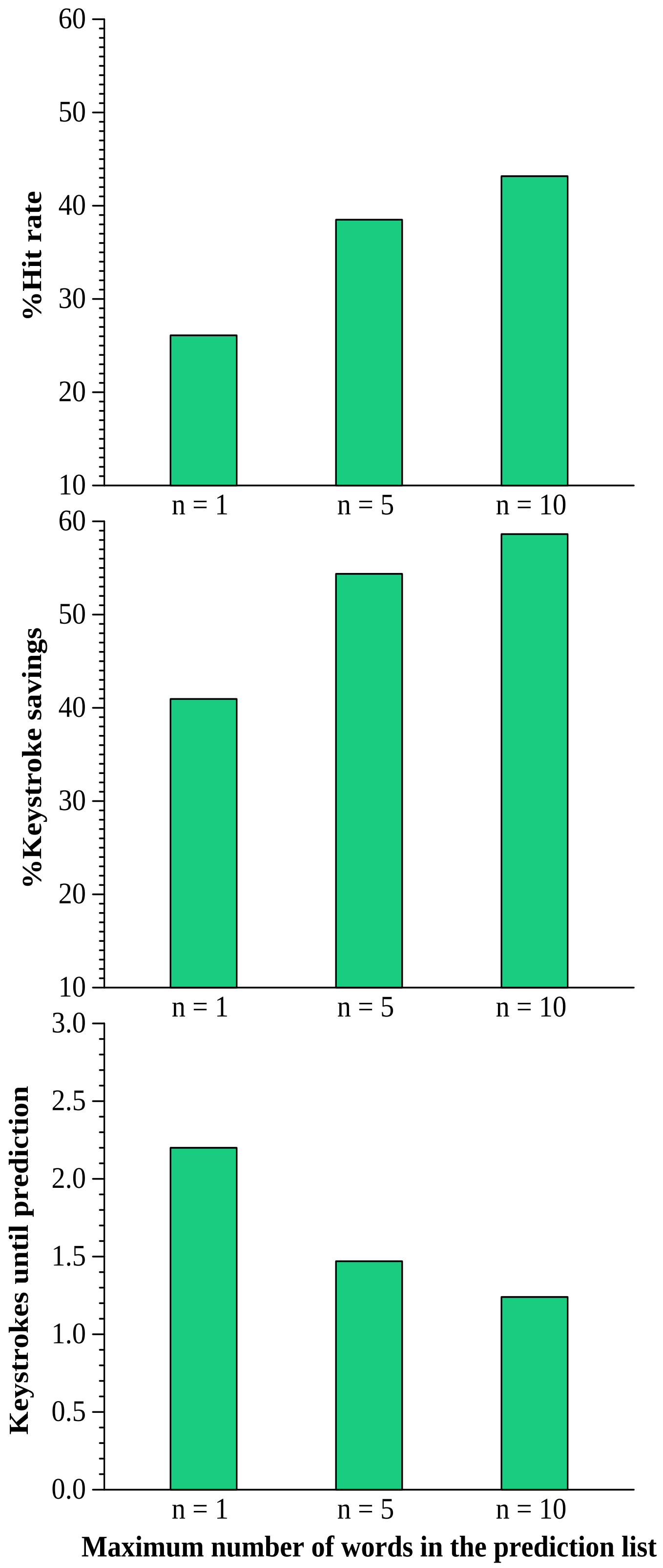


Figure 5.5: Effect of maximum prediction list size on hit rate, keystroke savings, and keystrokes until prediction of the Tags-and-words predictor when predictions are not repeated.

Table 5.4: Effect of prohibiting repeated predictions on performance of the Tags-and-words predictor when $n = 5$.

Experimental condition	%Hit rate	%Keystroke savings	Keystrokes until prediction
Repeated predictions are not removed	37.49	53.30	1.53
Repeated predictions are removed	38.50	54.37	1.47

Looking at the values for the three measures reveals that if the system does not repeat the same predictions for the same position, the likelihood of predicting the appropriate word will increase. Although the increase in the measures is not much, usually giving more accurate predictions decreases the amount of cognitive load imposed on the user and increases the level of the users' confidence in the system.

Although this might be true in most cases, it is not what we always want. For some individuals with learning or cognitive disabilities, it is not easy to scan the prediction list and therefore they may miss the correctly predicted word which is in the list. In real-world prediction systems, this parameter is often an option for the user.

5.2.6 Other measures

Table 5.5 contains the exact values for the three measures, accuracy, average number of predictions at each point, and average length of predicted words, for all of the prediction algorithms. Also in Figure 5.7 three bar-graphs are given for comparison purposes.

WordQ predicts 77.09% of the words at some point before the word is completed by the user. This means that around 23% of the words cannot be predicted by WordQ at all. Our Unigram predictor works much better and can predict around 90% of the words. The other four predictors, Bigram, Part-of-speech tag, Linear and Tags-and-words, have accuracy ranging from around 91% to 92.75%. The highest accuracy is that of the Linear predictor.

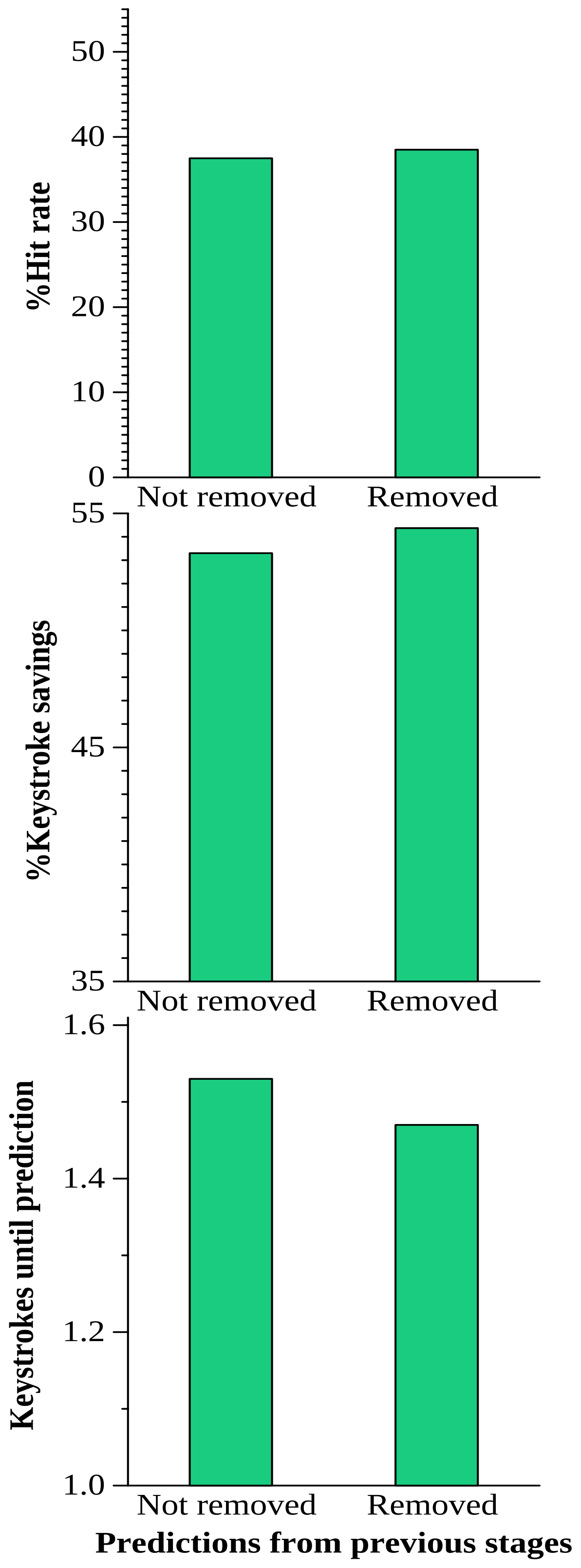


Figure 5.6: Effect of prohibiting repeated predictions on hit rate, keystroke savings, and keystrokes until prediction for the Tags-and-words predictor when $n = 5$.

Table 5.5: Accuracy, average number of words in the prediction list, and average length of predicted words for all prediction algorithms when $n = 5$ and predictions may be repeated.

Algorithm	%Accuracy	Average size of prediction list	Average length of predicted words
WordQ	77.09	4.11	4.81
Unigram predictor	90.34	4.82	4.47
Bigram predictor	92.07	4.73	4.41
Part-of-speech tag predictor	91.74	4.75	4.42
Linear predictor($\alpha = .8$)	92.75	4.73	4.39
Tags-and-words predictor	91.09	4.73	4.44

WordQ has the lowest value for the average size of the prediction list, considerably less than 5 (around 4.11). This means that the prediction algorithm cannot always find many words to suggest to the user. The reason might be the small word unigram and bigram models, or the small number of words in the working dictionary. The Unigram predictor has the highest value, 4.82; however this algorithm is not the best one according to the major performance measures. The word unigram and bigram models used in all the predictors (except WordQ) are sufficiently large; therefore, the value of this measure should depend on other factors as well. The Unigram predictor applies the fewest restrictions on the candidate words to find appropriate suggestions, and most probably that is the reason for its high average prediction list size.

Adding more information through part-of-speech tag n-gram models or considering larger context causes a small decrease in the average size of the prediction list. The reason is that there are more restrictions on the candidate words to choose from, and thus there are more cases in which the number of words that can match the restrictions is less than the number of words required.

WordQ has the largest value for the average length of words that were predicted correctly at some point before being completed by the user. This means that the predictor needs a longer prefix on average to predict the appropriate word.

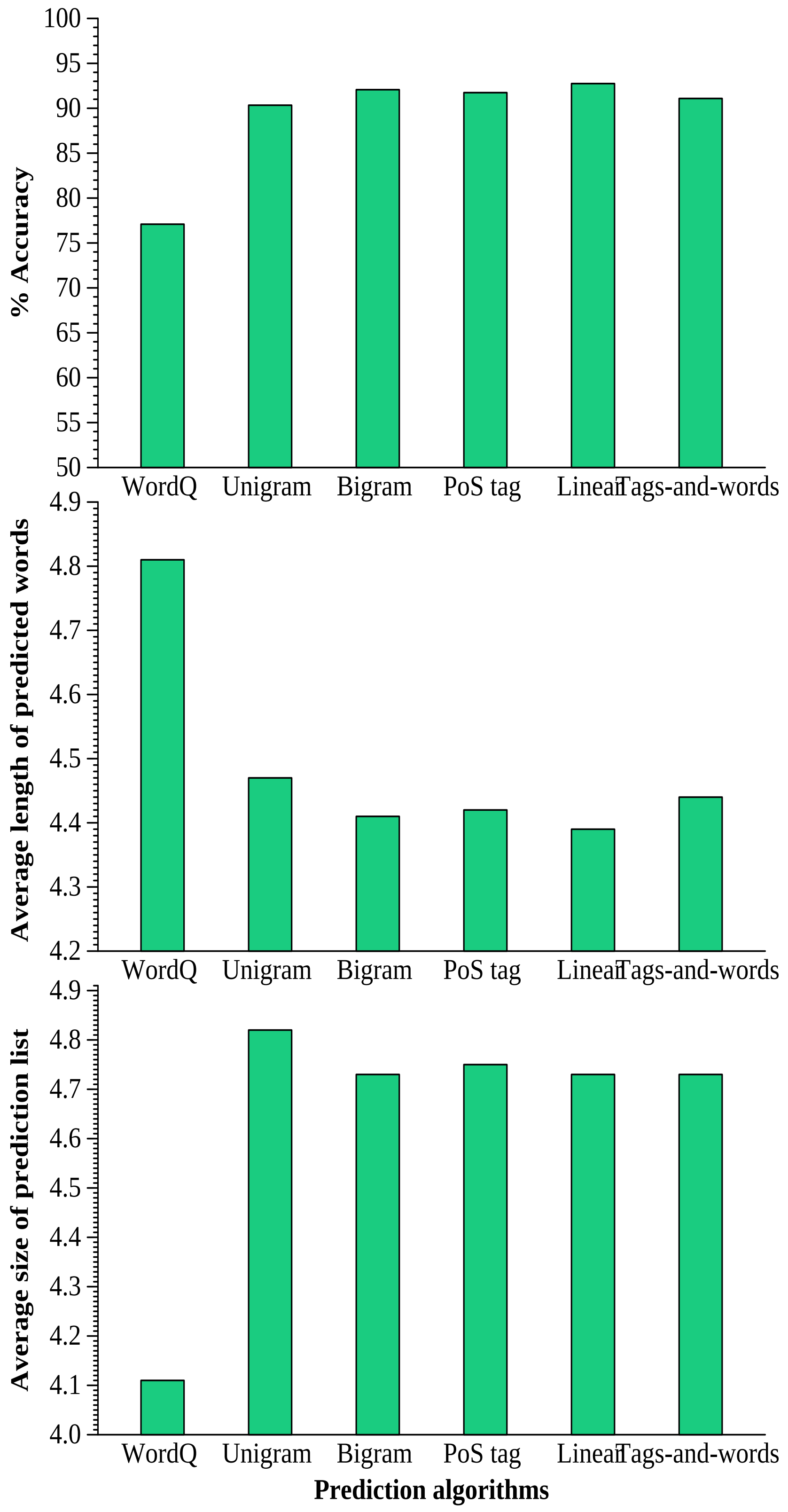


Figure 5.7: Comparison of the accuracy, average prediction list size, and average lenght of suggestions among all prediction algorithms.

Of course, as the length of the prefix entered by the user increases, the set of candidate words to choose from will be smaller. Thus, it is more likely to choose the correct word. As more information is added to the predictor, the average length of the predicted words should be less, which is what we can see in Table 5.5. However, the average length for all prediction algorithms except WordQ is very close (between 4.39 and 4.47).

5.3 Summary

In this chapter, we explained our experimental methodology, and presented the results of several experiments. The results show that our prediction algorithms outperform WordQ with a statistically significant difference. Also, it is confirmed that a word bigram model is more accurate than a word unigram model in predicting words.

The addition of part-of-speech tag information to the bigram model did not improve the performance of the predictor. The results show a small improvement in the performance measures of two of the syntactic predictors (Tags-and-words and Linear predictors) over the Bigram predictor. However, the ANOVA test shows that the difference is not statistically significant.

Chapter 6

Conclusions

Our goal in this work has been to investigate how we can improve the performance of word prediction by incorporating extra information in existing prediction systems, such as WordQ. We have implemented a number of existing prediction algorithms, such as the Unigram predictor, and introduced, designed, and implemented several new ones.

We have also performed a number of experiments to evaluate each of the prediction algorithms, to compare them with each other, and to measure the impact of various factors on the performance of the algorithms. In this chapter, we discuss contributions and limitations of our work and several possible extensions to it.

6.1 Contributions

A generic prediction testbench We have designed and implemented a generic word-prediction testbench that works with a large amount of statistical data. The testbench is designed to be fast since it was required to test each algorithm (with different values for each parameter) using a large set of test texts. Thus, several useful data structures, such as specific hash tables and caches, have been implemented to make this possible.

The testbench is designed in a way that it is possible to plug any of the prediction algorithms into it for test purposes. Any new prediction algorithm can easily be tested by being plugged into this generic testbench. Thus, for any further experiments in future we can use this generic testbench without any changes.

Prediction algorithms In order to test WordQ within the same environment as the other prediction algorithms, its prediction module has been ported from Windows to Linux. A Unigram predictor has been implemented that uses the word unigram model, only. These two algorithms are used as baselines in our comparisons.

Another prediction algorithm that has been implemented for test purposes is the Bigram predictor that uses the word bigram model created from the training corpus. Three different prediction algorithms that make use of syntactic information (through part-of-speech tags) have also been designed and implemented. These algorithms use the part-of-speech tag information in three different ways to see the impact of each method on the accuracy of the predictions.

Large training corpus A very large data set, the British National Corpus, is used both for training and testing. The corpus has been divided into three different parts: a training set, a development set and a test set. We have collected statistical information for words and part-of-speech tags from the training corpus by using a toolkit (SLM) and several other utility programs written for this specific purpose.

Adaptation As mentioned in Chapter 3, adaptation techniques can be exploited in any prediction system, no matter what prediction algorithm it uses. Adaptation helps a prediction system to improve the accuracy of the predictions and the level of the user's confidence in the system.

We have implemented one adaptation technique in our prediction testbench. The main idea of the technique is to avoid repeating the same suggestions that have already been rejected by the user for the same position. This technique has been implemented and the effect of adding it to the prediction process has been tested by testing the Tags-

and-words predictor in both conditions: using the adaptation technique or not using it.

Experiments An extensive number of experiments have been conducted to measure the performance of each algorithm and to compare them with each other. The experiments have been designed so that they test various aspects of a prediction algorithm, and the effect of different factors and parameters on the accuracy of the suggestions.

Comparison and statistical significance test The results show that we can build strong and accurate word unigram and word bigram predictors that outperform WordQ by using a very large amount of training data. It is confirmed that other prediction algorithms that make use of part-of-speech tag information, outperform WordQ as well. We have also presented results showing that the addition of part-of-speech tag information to the bigram model does not improve the accuracy of the predictions much.

We have also performed a statistical test (ANOVA) to find whether any of the algorithms is significantly different from the other ones. Although, there is a small improvement in the performance measures of two of the syntactic predictors (the Tags-and-words and the Linear predictors) compared to the Bigram predictor, the ANOVA test shows that the difference is not statistically significant.

Because of our positive and negative results, it is difficult to make a general statement regarding the use of part-of-speech tag information in prediction algorithms. There are many areas that deserve further investigation, and many new experiments to be done. For instance, our attempt has been to build prediction models using n-gram models for words and/or part-of-speech tags. However, it might be useful if we can investigate creating models using probabilities of sequences of words and part-of-speech tags together. In Section 6.2 several possible extensions to this work are discussed.

6.2 Future work

Combined n-gram models As mentioned in Chapter 5, adding part-of-speech tags to word bigrams does not improve the performance significantly. One of the reasons is that we have built separate n-gram models for words and part-of-speech tags; i.e., word unigram and bigram, part-of-speech tag unigram, bigram, and trigram. In light of this observation, we believe that we can benefit from a model that captures the probabilities of words and part-of-speech tags together. It is possible that if we use combined n-gram models, i.e., probabilities of sequences of words and part-of-speech tags together, we would have more accurate models and accordingly more accurate predictions.

A coarse-grained tagset As mentioned in Chapter 5, the C5 tagset contains 61 distinct syntactic categories. This number of distinct classes might not be needed for the prediction task. Thus, we think that if we combine some of the classes and create a coarse-grained tagset, it would be possible to have a decrease in the tagging error as well as an increase in the prediction accuracy.

Semantics Even if the predictions are syntactically and grammatically appropriate for the desired position, they might not as well be semantically appropriate. Therefore, one interesting extension to the current work is to add semantic information of the words and/or the context the word appears in to the predictor. There are several ways to add semantic information to the prediction algorithms. One is using a *lexical resource* such as WordNet, see (Fellbaum, 1998) together with a semantic similarity measure to make sure that the words suggested by the prediction algorithm are semantically relevant (by some degree) to the context, see (Hirst and Budanitsky, 2001). Another way is to use *lexical chains*, see (Morris and Hirst, 1991) to keep the semantic relevance of the context prior to the position for which predictions are required. Thus, the prediction algorithm can remove the words which are not semantically relevant to the context from the list of candidate words.

Adaptation We have only implemented and tested one adaptation technique in our prediction testbench. It is possible to easily add other techniques to our testbench and test them with any of the prediction algorithms. An extension to our work can be to find other adaptation techniques which might be worth implementing. In this way, we would be able to see the impact of using those adaptation techniques with our prediction algorithms.

Coefficient and the dictionary Figure 6.1 shows the probability distribution function of the dictionary words. As mentioned in previous chapters, the dictionary contains $65,000$ of the most-frequent words occuring in the training texts. It can be seen that the 250 most-frequent words, mostly function words, have about 60% of the probability distribution, and the $7,100$ most-frequent words have about 90% of the probability distribution.

Thus, there are many words in the dictionary that altogether account for a very small portion of the probability distribution. It is possible for a word-prediction system to suggest high-frequency words more often than they are really needed by the user, preventing the appropriate (probably less frequent) words from being predicted.

One of the experiments that can be done is to remove the high-frequency function words and/or short words (which do not have much effect in keystroke savings) to see how the performance changes. Also, it might be the case that for high-frequency words, the part-of-speech tag information helps more. Thus, instead of removing these words, it is possible to find a different value for α for every set of words according to their frequency.

Real users We have introduced six performance measures in Chapter 5. However, one important factor in the success of a prediction system is its acceptability by the users.

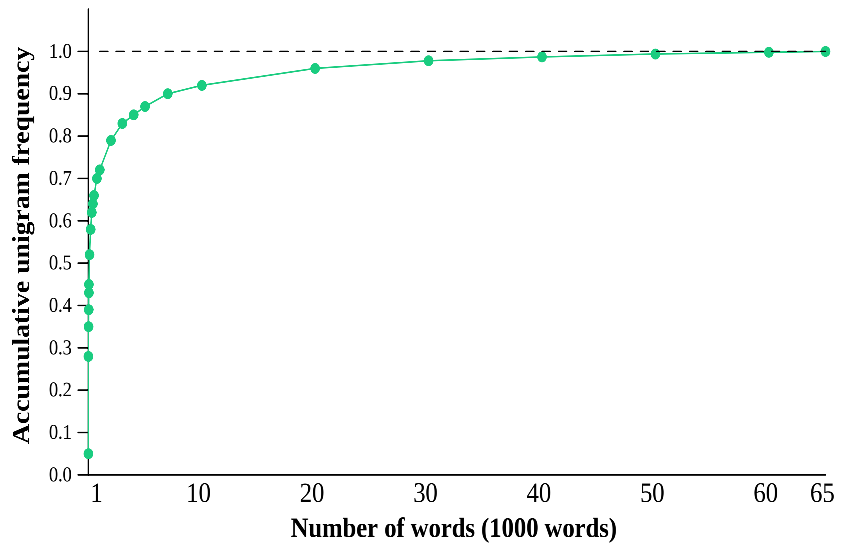


Figure 6.1: The probability distribution function of the dictionary words.

This means that even if a prediction algorithm has high values for those performance measures, it does not guarantee that the system is considered a good one by the users as well, especially since every user with disabilities might have different concerns and different problems to be solved by the system. It would be good to create an interface for the testbench, so that we can test the performance of the predictors by asking real users to use it for a period of time. Thus, we can evaluate the prediction algorithms according to the users' opinions as well as the selected performance measures.

Bibliography

Alm, N., Arnott, J. L., and Newell, A. F. (1992). Prediction and conversational momentum in an augmentative communication system. *Communications of the ACM*, 35(5):46–57.

Arnott, J. L., Pickering, J. A., Swiffin, A. L., and Battison (1984). An adaptive and predictive communication aid for the disabled exploits the redundancy in natural language. In *Proceedings of the 2nd International Conference on Rehabilitation Engineering*, pages 349–350, Ottawa.

Baker, B. R. (1982). Minspeak. *Byte*, 9:186–202.

Banko, M. and Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting and 10th Conference of the European Chapter of the Association for Computational Linguistics*, pages 26–33, Toulouse, France. ACL.

Bentrup, J. A. (1987). Exploiting word frequencies and their sequential dependencies. In *Proceedings of the 10th Annual Conference on Rehabilitation Technology*, pages 121–123. RESNA.

BNC manual (2000). *Reference Guide for the British National Corpus (World Edition).* www.hcu.ox.ac.uk/BNC, second edition.

Booth, L., Morris, C., Ricketts, I., and Newell, A. F. (1992). Using a syntactic word predictor with language impaired young people. In *Proceedings of the CSUN 7th Annual Conference on Technology for Persons with Disabilities*, pages 57–61, USA.

Carlberger, A., Carlberger, J., Magnuson, T., Hunnicutt, S., Palazuelos-Cagigas, S. E., and Navarro, S. A. (1997a). Profet, a new generation of word prediction: An evaluation study. In Copestake, A., Langer, S., and Palazuelos-Cagigas, S., editors, *Natural language processing for communication aids, Proceedings of a workshop sponsored by ACL*, pages 23–28, Madrid.

Carlberger, A., Magnuson, T., Carlberger, J., Wachtmeister, H., and Hunnicutt, S. (1997b). Probability-based word prediction for writing suport in dyslexia. In Bannert, R., Heldner, M., Sullivan, K., and Wretling, P., editors, *Proceedings of Fonetik'97 Conference*, volume 4, pages 17–20.

Clarkson, P. and Rosenfeld, R. (1997). Statistical language modeling using the cmu-cambridge toolkit. In *Proceedings of the ESCA Eurospeech 97*.

Cross, R. T., Baker, B. R., Klotz, L. V., and Badman, A. L. (2001). Static and dynamic keyboards: semantic compaction in both worlds. www.prentrom.com/aacassessment/assessmentindex.html.

Darragh, J. J. and Witten, I. H. (1992). *The Reactive Keyboard*. Cambridge University Press.

Darragh, J. J., Witten, I. H., and James, M. L. (1990). The reactive keyboard: A prediction typing aid. *IEEE Computer*, 23(11):41–49.

Demasco, P. W. and McCoy, K. F. (1992). Generating text from compressed input: An intelligent interface for people with severe motor impairments. *Communications of the ACM*, 35(5):68–78.

Even-Zohar, Y. and Roth, D. (2000). A classification approach to word prediction. In *Proceedings of the Conference, 2nd Meeting of the North American Chapter of the Association for Computational Linguistics*. ACL.

Fellbaum, C., editor (1998). *WordNet, An Electronic Lexical Database*. The MIT Press.

Foulds, R. (1980). Communication rates for non-speech expression as a function of manual tasks and linguistic constraints. In *Proceedings of the International Conference on Rehabilitation Engineering*, pages 83–87, Toronto. RESNA.

Foulds, R., Soede, M., van Balkorm, H., and Boves, L. (1987). Lexical prediction techniques applied to reduce motor requirements for augmentative communication. In *Proceedings of the 10th Annual Conference on Rehabilitation Technology*, pages 115–117. RESNA.

Garay-Vitoria, N. and González-Abascal, J. (1997). Intelligent word prediction to enhance text input rate (a syntactic analysis-based word prediction aid for people with severe motor and speech disability). In *Proceedings of the Annual International Conference on Intelligent User Interfaces*, pages 241–244.

Hirst, G. and Budanitsky, A. (2001). Correcting real-world spelling errors by restoring lexical cohesion. *MS, Submitted for publication.*

Hunnicutt, S. (1987). Input and output alternatives in word prediction. Technical Report STL-QPSR 2-3, Department of Speech Communication and Music Acoustics, KTH.

Hunnicutt, S. (1989). Using syntactic and semantic information in a word prediction aid. In *Proceedings of Eurospeech 1989*, volume 1, pages 191–193.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing*. Prentice Hall.

Katz, S. M. (1987). Estimation of probabilities from sparse for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing ASSP*, 35:400–401.

MacKenzie, S., Kober, H., Smith, D., Jones, T., and Skepner, E. (2001). Letterwise: prefix-based disambiguation for mobile text input. In *Proceedings of the ACM Symposium on User Interface Software and Technology — UIST 2001*, New York. ACM.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. The MIT Press.

Morris, J. and Hirst, G. (1991). Lexical coherence computed by thesaural relations as an indicator of the structure of text. *Computational Linguistics*, 17(1):21–42.

Moulton, B. J., Lesher, G. W., and Higginbotham, J. (1999). A system for automatic abbreviation expansion. In *Proceedings of the RESNA 22nd Annual Conference*, pages 55–57. RESNA.

Nantais, T., Shein, F., and Johansson, M. (2001). Efficacy of the word prediction algorithm in WordQ. In *Proceedings of the 24th Annual Conference on Technology and Disability*. RESNA.

Shein, F., Nantais, T., Nishiyama, R., Tam, C., and Marshall, P. (2001). Word cueing for persons with writing difficulties: WordQ. In *Proceedings of CSUN 16th Annual Conference on Technology for Persons with Disabilities*.

SLM documentation (2001). *The CMU-Cambridge Statistical Language Modeling Toolkit HTML Documentation*. http://svr_www.eng.cam.ac.uk/ prc14/toolkit_documentation.html.

Swiffin, A. L., Arnott, J. L., and Newell, A. F. (1987). The use of syntax in a predictive communication aid for the physically handicapped. In *Proceedings of the 10th Annual Conference on Rehabilitation Technology*, pages 124–126. RESNA.

Swiffin, A. L., Arnott, J. L., Newell, A. F., and Brophy, B. (1988). Function and content words in a predictive system. In *Proceedings of ICAART Conference*, pages 70–71, Montreal.

Swiffin, A. L., Pickering, J. A., Arnott, J. L., and Newell, A. F. (1985). PAL: An effort efficient portable communication aid and keyboard emulator. In *Proceedings of the 8th Annual Conference on Rehabilitation Technology*, pages 197–199. RESNA.

Treviranus, J. and Norris, L. (1987). Predictive programs: Writing tools for severely physically disabled students. In *Proceedings of the 10th Annual Conference on Rehabilitation Technology*, pages 130–132. RESNA.

VanDyke, J. A. (1991). A syntactic predictor to enhance communication for disabled users. Technical Report 92-03, Department of Computer and Information Sciences, University of Delaware.

Wood, M. E. (1996). *Syntactic pre-processing in single-word prediction for disabled people.* PhD thesis, Department of Computer Science, University of Bristol.

Woods, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606.