AUTOMATIC TERM EXTRACTION IN TECHNICAL DOMAIN USING PART-OF-SPEECH AND COMMON-WORD FEATURES

by

Nisha Ingrid Simon

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
May 2018

Table of Contents

List of Tables

List of Figures

Abstract

Extracting key terms from technical documents allows us to write effective documentation that is specific and clear, with minimum ambiguity and confusion caused by nearly synonymous but different terms. For instance, in order to avoid confusion, the same object should not be referred to by two different names. In the modern world of commerce, clear terminology is the hallmark of successful RFPs (Requests for Proposal) and is therefore a key to the growth of competitive organizations. While Automatic Term Extraction (ATE) is a well-developed area of study, its applications in the technical domain have been sparse and limited to certain narrow areas such as the biomedical research domain. We present a method for Automatic Term Extraction (ATE) for the technical domain based on the use of part-of-speech features and common words information.

The novelty of this thesis lies in the domain to which ATE is applied. Our method is evaluated on a C programming language reference manual as well as a manual of aircraft maintenance guidelines, and has shown comparable or better results to the reported state of the art results. In addition, we also compared our system to another method (TBXTools statistical) and obtained favorable results.

List of Abbreviations and Symbols Used

Collocation	An expression of two or more words that represent a conventional way of stating an idea
F-measure	A weighted harmonic mean between precision and recall
False Negatives	Items that are not returned but are relevant
False Positives	Items that are returned (identified) but not relevant
NLTK	Natural Language Toolkit
POS	Part of Speech
POS Tagging	Assigning grammatical identifiers to words
Precision	The percentage of true positives out of all returned items
Recall	The percentage of true positives out of all relevant items in the dataset
Tokenizaton	Refers to the removal of punctuation and the splitting of a sentence into individual words
True Positives	Items that are both returned (identified) and relevant

Acknowledgements

Thank you to all the people without whom it would not have been possible to complete this thesis. I would especially like to thank my supervisor, Professor Vlado Kešelj, for his continuing support, patience, and guidance throughout my graduate degree program. I greatly appreciate his taking the time away from his many other responsibilities to meet with me regularly, provide suggestions and discuss various research approaches and ideas. Thank you to my committee members, Professor Evangelos E. Milios and Dr. Abidalrahman Mohammad, for their advice and suggestions. Thank you to everyone at the Faculty of Computer Science for creating a stimulating and welcoming research environment. I would like to express my thanks to numerous Dalhousie University faculty and staff members for their assistance and advice during my graduate studies; Dr. Carolyn Watters, Professor Norbert Zeh, Dr. Alex Brodsky and many others. Special mention to Sittichai Jiampojamarn, Heather Sutherland and Anne Bartlett for their support and advice. Thank you to my colleagues in the Dalhousie Natural Language Processing group for sparking many thought-provoking research discussions. And last but not least, to my family, thank you for all your encouragement, advice, patience, and love. Thank you for believing in me.

Chapter 1

Introduction

Automated Term Extraction (ATE) from technical documents has become an important problem since extracting key terms from technical documents allows us to write high-quality documentation that is specific and clear, with minimum ambiguity. For instance, in order to avoid confusion, the same object should not be referred to by two different names. In the world of business, clear terminology is the hallmark of successful RFPs (Requests for Proposal). ATE in general is an important area of study because it has applications in IR (Information Retrieval) such as text summarization, text categorization, opinion mining and document indexing.

While in the sphere of general or creative writing, such as various literary works which could include prose and poetry and even journalism, the authors tend to make their writing more vivid and interesting by choosing different synonymous words for the same concept. This would generally be a weakness and a source of confusion in technical writing. For example, if a technician is following some assembly instructions, referring to the same part with different names in different parts of the documents could be confusing and would increase the chance of error in assembly. This is one of the reasons why a more uniform and standardized terminology is highly regarded in the technical domain. Therefore a tool that can detect terminology and help a writer to create a better document would be beneficial. A similar service would also be useful to reviewers and other assessors of technical proposals and similar documents.

The number of technical terms is constantly growing, especially in specialized areas such as computer science, engineering and medicine [FAM00]. Manually assigning key terms is both tedious and time consuming [FPW+99]. With the pace of knowledge acquisition that is required to maintain currency in modern technical fields, it is helpful for a user to have access to a method of quickly extracting part of an index that is comprised of the key terms of a document [DGBPL00].

Although much work has been done on ATE, the state of the art performance is

still low as found by Hasan et al. [HN14]. Major factors that affect ATE performance measures are the length of documents, a lack of structural consistency (for instance, a lack of a defined index or abstract), topic changes, and the presence of uncorrelated topics in the same text.

1.1 Research Objective

Our main hypothesis is that the Automatic Term Extraction (ATE) can be successfully performed in a (novel) technical domain to extract key phrases using Part-of-Speech information with additional information about commonality and frequency of domain-specific words. We achieve this research objective by building and evaluating a system for automated term extraction based on Part-of-Speech and common-word feature information, and evaluating it on the C programming language standards. To be more specific, we use a C Reference Library that incorporates the C Standards (henceforth merely referred to as *C Program Language Standards* or *C Standards*). As a gold standard we use the manually prepared indexes of terms found at the end of these standards. In addition, we apply our methods on a manual of aircraft maintenance guidelines [Hig90] (referred to in the thesis as the *aircraft manual data set*), and compare our results to that obtained from the C programming language standards document. Results are then compared to those obtained from the TBXTools system [OGVG15].

1.2 Contributions

The novelty of this thesis lies in the domain to which ATE is applied. While ATE has been used in extremely specific areas such as the biological or biomedical domain [AFT15, LVJRT16], Archaeology or Chemistry [BBR+16], or general areas such as newspapers [CD16], to the best of our knowledge, ATE has not been used earlier in the technical domain on documents such as C programming language reference manuals or aircraft maintenance guidelines manuals. The previously published methodology [DSD05] had to be adapted since a new domain was used. While Da Sylva produced a hierarchical index where secondary terms were listed under primary terms, e.g. “planet” would be a primary term and “earth” and “moon” would be secondary terms

that were nested below it, we aim to create a list of stand-alone, non-hierarchical key terms.

Another original part of our methodology is that while the majority of earlier research focuses on extracting only a limited set of the most important keywords by rank; i.e., cutoffs are used [FPW+99, Tur00, MI04], while our research is aimed at generating all valid index terms from a document for the purpose of replicating a back-of-the-book index.

1.3 Outline of the Thesis

The thesis is structured as follows: Chapter 2 presents the background of ATE and an overview of related work. We present relevant published research and provide an overview of automatic term extraction. Chapter 3 describes the system and methodology: tools and equipment used, data pre-processing, POS tagging, candidate detection and keyword selection. We describe our method of stop word and common word selection and usage. Chapter 4 explains the experiment design and the experimental results obtained, while Chapter 5 contains a discussion of performance measures used, and the evaluation of the results. The conclusion is presented in Chapter 6, summarizing the contributions and outlining directions for future work.

Chapter 2

Background and Related Work

2.1 Background

Automatic keyphrase extraction is the identification of phrases in a document that together summarize the contents and major ideas contained within that document. The key phrases can be any terms that appear in the document. This is contrasted with keyphrase assignment or text categorization, where a pre-defined list of constrained key words is assigned to a document based on the document's contents, and the key words may not directly appear in the particular document [Tur00]. Applications and related areas of use of ATE include:

- Information Retrieval. The retrieval of documents based on the keywords they contain can be improved if ATE is used [PL01, DGBPL00].
- Glossary creation. Creating a glossary can involve building dictionaries, thesauruses or translation memories using ATE [OGVG15].
- Named Entity Recognition. Some examples of Named Entity Recognition are extracting company names, locations or industry specific terms from newspaper articles [TKSDM03], both of which can be made more accurate by the use of ATE.
- Document Clustering. Grouping documents by their shared features is another area where the use of ATE can provide improvements in efficiency by using keywords as features [KMKB13].
- Text Summarization. Summarizing long documents into shorter pieces of text can be improved using ATE, since the key terms can be extracted and then used to create a precis of the document. Keyphrases can therefore provide "semantic metadata indicating the significance of sentences and paragraphs in which they appear" [KMKB13].

While ATE is a well developed area of study, its applications in the technical domain have been sparse. The Related Work section which follows will provide a summary of prior related literature.

2.2 Related Work

There has been much work on Automatic Term Extraction based on document features and statistical approaches [KK09, SS15]. As early as 1987 the use of "phrase relationships", or a sequence of words that "occur together in a document" in a given delimited piece of text such as a sentence or paragraph was reported on by Fagan [Fag89]. These are not dependent on the meaning of the individual words that comprise a phrase, but rather can be seen as a form of using collocations, such as "text analysis" and "book review". He noted that a problem which needs to be considered is "the structural ambiguity of many complex noun phrases". Collocations are also discussed in [PPZ05] where they determine *unithood*, which is described as the "strength or stability of syntagmatic collocations". Syntagmatic words are those that form some sort of relationship. Unithood is contrasted with *termhood* which is the degree to which "a linguistic unit is related to domain-specific concepts"[PPZ05]. Unithood and termhood were also discussed in [KU96]. Therefore, finding collocations in the text takes us further in compiling a list of potential key terms.

In the course of comparing the performance of the C4.5 decision tree algorithm and the GenEx algorithm on email messages and web pages, Turney [Tur00] defined automatic keyphrase extraction as "the automatic selection of important topical phrases from within the body of the document". A human-generated keyphrase was considered to be the same as a machine-generated keyphrase if they had the same sequence of *stems*. A stem of a word is what remains of the word when its suffix is removed.

KEA was presented by Witten et al. [WPF+99], as an "algorithm for automatically extracting key phrases from text" using a Naive Bayes classifier that is trained on documents from the New Zealand Digital library, to extract features from text and apply them to other documents. The researchers evaluated their system by comparing the number of matches between their system's output and the number of author-assigned keyphrases. They concede that KEA found "less than half the author's

phrases", but recognize that even different human evaluators will vary on the set of key phrases that they select from the same document.

A method to create a back-of-the-book index for *Stargazers* text using lexical classes was described by Da Sylva [DSD05]. This work involved the main steps of noun phrase extraction (including lemmatization and part-of-speech tagging), text segmentation, candidate term weighting and index compilation. Da Sylva also differentiated between SSTV (specialized scientific and technical vocabulary) which is "specific to each discipline, science or trade" and "contains the perfect candidates for indexing", and BSV (Basic Scientific Vocabulary) that consists of nouns that are "general words used in all scientific domains" [DS09]. She contrasted this with CV (common vocabulary) which is made up of everyday words normally learned in primary grades and is not generally a good source of index words. Identifying these three groups of words is a key part of building an index. However she notes that these three groups, rather than having well-delineated boundaries, can in fact overlap and merge with one another. For instance the word *application* can have either a BSV meaning or a SSTV meaning, such as when it is used in the sense of *computer program* [DS09].

Other related work includes Ferrari et al. [FdSG14]. TBXTools which is a tool built in Python to extract key terms from controlled corpora, was presented by Oliver and Vàzquez [OGVG15]. The term *controlled* in this sense means that the key terms are known in advance. Statistical and linguistic methods are combined to extract multiword terms. A general corpus of news articles and a specialized corpus of telecommunications documents were used by Drouin [Dro03] to extract key terms based solely on nouns and adjectives.

While some prior research chose to use a large number of documents, smaller datasets have also been used. There is an instance where ATE has been used on only the abstracts of scientific articles, as opposed to the entire documents [Hul03]. Shorter text data sets were also used in [MI04] where a co-occurrence matrix was used to extract the top ranked fifteen keyterms from a single document (a paper by Alan Turing) without the benefit of a corpus. It has been noted in the literature that the size of the corpus is less important than its design and representativeness of the domain, especially in the case of specialized domains [PPMM15].

An overview of the advances in Natural Language Processing was presented by

Hirschberg and Manning [HM15] where it is observed that "simple methods using words, part-of-speech (POS) sequences... or simple templates can often achieve notable results when trained on large quantities of data."

Named Entity Recognition has been done using machine learning techniques [JCK05]. The goal of the paper was to find biological terms in scientific publications using n-grams. Feature attributes such as upper and lowercase, prefix and suffix characters were used. N-grams were used instead of searching for noun phrases since "some biological terms are partial noun phrases". Named Entity Recognition was also the subject of the CoNLL 2003 shared task [TKSDM03] where data from English and German newspapers was used, after "a tokenizer, a part-of-speech tagger, and a chunker were applied to the raw data".

ATE systems are generally categorized into statistical, linguistic and hybrid systems [PPMM15]. Statistical systems use purely statistical techniques such as measures of frequency. Linguistic systems rely on analyzing the linguistic characteristic of the corpora such as its part of speech tags. While linguistic systems generally give better performance than statistical systems, linguistic systems are heavily dependent on the language of the corpus to be analyzed. It was observed by Castellví that statistically based systems tend to "produce too much silence" while linguistically based systems tend to "produce a great deal of noise" [CBP01] "Noise [refers to] the rate between discarded candidates and accepted ones. Silence [refers to] those terms contained in an analyzed text that are not detected by the system" [CBP01]. The problem of "silence" in statistically based ATE systems and "noise" in linguistic systems have also been commented in Conrado et al. [dSCDFPR14] . Hybrid systems use a combination of statistical and linguistic techniques, such as the C value and NC value.

The *C/NC* method was described by Frantzi et al. [FAM00] for the purposes of analyzing eye pathology medical records. The *C value* was used to assign *termhood* and the *NC value* provided context information. Strings that matched a linguistic filter and a certain threshold were extracted. The *C/NC* method has also been used by Milios et al. to extract key terms from a neural network corpus consisting of 100 papers. The first 820 words of each paper were used and the *C-value* and *NC-value* were calculated. The *C-value* "is a domain specific method used to automatically

extract multi-word terms... [It] combined linguistic knowledge (which consists of part-of-speech tagging, linguistic filters [POS sequences], stop list etc.) and statistical information to obtain a *termhood* value." [MZHD03] They also observed that using different filters will produce varying levels of precision and recall. The *NC value* uses "context words" which are "nouns, verbs and adjectives that either precede or follow" real terms (from the top 10 candidate terms) that have been manually labeled by domain experts [MZHD03] and are dependent on the domain. *NC values* are used to re-rank the candidate terms found by the *C-value* method.

In summary, although much work has been done on ATE, to the best of our knowledge there has been little work on technical documents such as C programming language standards or aircraft manual maintenance guidelines. We believe that these data sets provide a unique opportunity to study ATE in the technical domain, using carefully selected domain-specific stop words and POS information. We have focused our experiments on retrieving all relevant key phrases, in contrast to much of the previous literature which focused on retrieving only a selected list of ranked key phrases. After having considered an overview and a summary of the related work in the area of ATE in this chapter, we now move on to discussing our specific methodology in the next chapter.

Chapter 3

Methodology

3.1 Methodology Overview

This chapter presents our methodology and the rationale for our experiment design. Our overall methodology and system architecture is represented in Figure 3.1 and corresponding sections of the thesis are shown in brackets. The main components of our system are Data Pre-processing, POS Tagging, Candidate Detection, Stop words and Common words and Keyword Selection. Data pre-processing forms an important part of the process as described in Figure 3.1 . Both Document and Index data needed to be modified and cleaned. In order to perform the POS tagging, the data must then be split into sentences and also tokenized (*tokenizaton* refers to the removal of punctuation and the splitting of a sentence into individual words). POS tagging is then performed in order to obtain candidate terms. A list of stop words (common words) and calculation of the frequency of word occurrence are then used to find likely keywords. Stop words and common words selection is explained in greater detail in section 3.4.

The design of our methodology was driven by the nature of ATE. ATE can be considered to be a supervised learning task [Tur00]. A part of ATE can also be seen as a search for collocations [MS$^+$99]. A collocation is an expression of two or more words that represent a conventional way of stating an idea; e.g., "strong tea", "broad daylight". Collocations are characterized by three main features as discussed by Manning and Schütze [MS$^+$99]:

- *non-compositionality*, i.e. the meaning of the whole collocation does not correspond to the meaning of its parts.

- *non-substitutability*, i.e. other words cannot be substituted for the components of a collocation. For example the phrase *powerful tea* would sound odd to the ear of a native speaker of the English language, when what is actually meant is

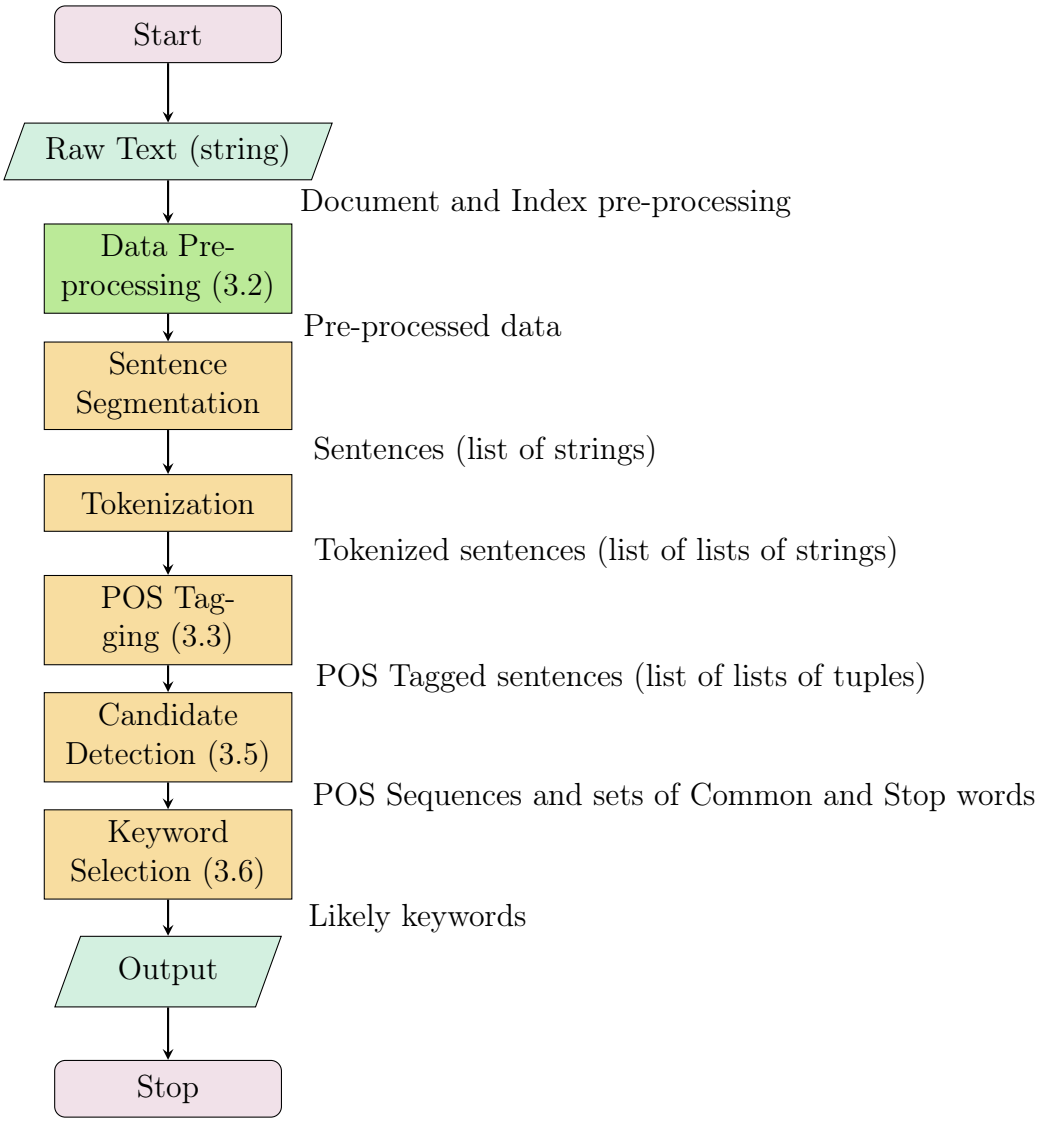

Figure 3.1: Architectural Overview of the System

strong tea.

- *non-modifiability*, i.e. a collocation cannot be modified with additional lexical material or grammatical transformations. For instance *declaring union variables* is a suitable index term that makes logical sense, while *declaring the union of variables* is not.

One way of identifying collocations and key terms is by using part-of-speech tags. In order to perform part of speech tagging, a grammatical tag is given to each word of the text [FAM00]. A method of selecting the most frequent bigrams (consisting

of taking two consecutive words at a time from the text) and passing them through a POS (Part of Speech) filter of "likely phrase" sequence patterns was proposed by Justeson [JK95] . For example, if some likely part-of-speech patterns for selecting terms are *adjective-noun* or *noun-noun*, the filter would search for patterns *JJ NN* or *NN NN* in the text, such as "real number" or "certification requirement". A list of Penn Treebank POS tags that were used in this project is provided in table 3.1.

Noun phrases (NPs) are especially useful for identifying key words [dSCDFPR14]. An earlier review of the ATE landscape [CBP01] showed that a number of systems focused on extracting NPs since "there is a high rate of terminological NPs in specialised texts". On the other hand, it was also recognized that "all specialised languages have their own verbs ... no matter how low the ratio is in comparison with nouns" [CBP01]. An example of POS tagging that can be used on a sentence to extract features such as noun phrases is shown in Figure 3.2. The given sentence is *"The real number types provided in C are of finite precision"*.

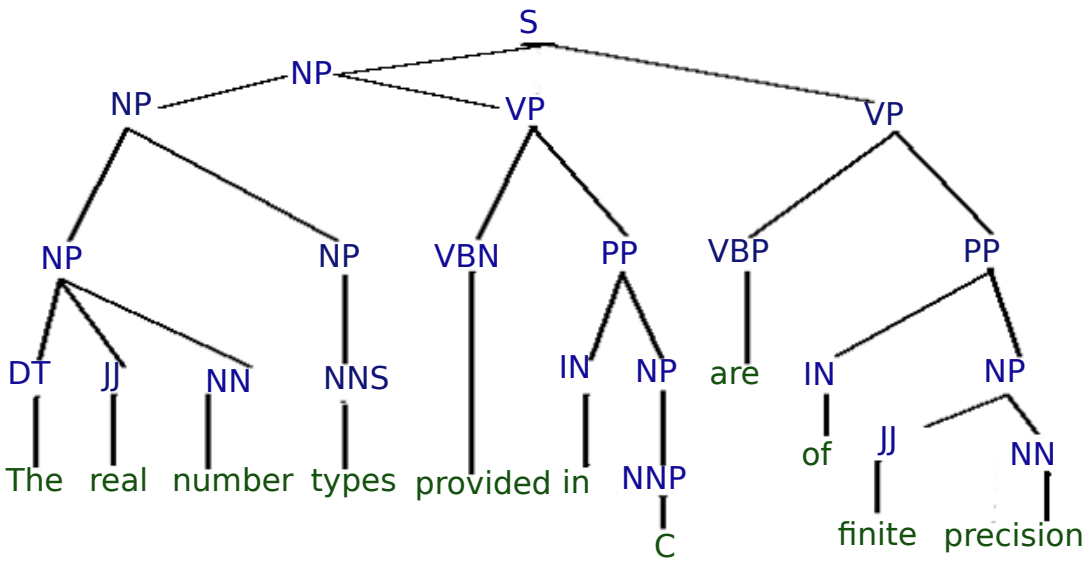

Figure 3.2: Noun Phrases based on POS Tags

The corresponding sentence tree is given as:

(S (NP (NP The/DT real/JJ number/NN) (NP types/NNS))
(VP provided/VBN (PP in/IN (NP C/NNP)))
(VP are/VBP (PP of/IN (NP finite/JJ precision/NN))))

Table 3.1: POS Tags and their Descriptions

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential *there*
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal Verb
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	Infinitive *to*
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

The above approach was followed as well by Hulth [Hul03]. "Fifty-six POS patterns based on tag sequences of manually assigned keywords, [...] that occurred ten or more times" were used on 2000 abstracts from the Inspec database. Manually assigned keywords were treated as the gold standard. In terms of evaluating results, it was observed by Hulth [Hul03] that this is "the most severe way to evaluate a keyword extractor, as many terms might be just as good, although ... not chosen by a human indexer".

In our research we use C programming language standards as our first dataset. The index of each C programming document was used as the gold standard to measure how well the system performed. The original C programming standards document was converted to plain text from HTML. It was then split into a text file (body of the document) and an index file. The index file, which consisted of the index terms as well as page numbers and the section of the document in which the index terms appeared, underwent further processing to remove the page numbers and section headers. This left a plain text file consisting only of index terms, with one term per line. The Table of Contents and section headings were removed from the text file. An initial count of POS tags based on sentences and words was performed. After individual words were tagged, POS sequences were then extracted.

A second data set that was used was a manual of aircraft maintenance guidelines, referred to in this thesis as *aircraft manual data*. This was a set of PDF documents that were converted to plain text format. As a gold standard, the glossary at the end of this document was used to validate candidate terms. A major difference between the aircraft manual data and the C Programming Standards data was that the former contained a large number of acronyms.

Once our system generated candidate terms, we evaluated our performance measures when compared to that of the TBXTools system. The Odds Ratio test provided a value of greater than one in favour of our system when compared with TBXTools statistical. Further details are provided in the Experimental Results section.

The sub-steps of the Methodology are presented in greater detail in the following sections. These sub-steps cover Data Pre-processing, POS Tagging, Candidate Detection, Stop words and Common words and Keyword Selection.

3.2 Data Pre-processing

The first set of data used in the experiments were C programming language standards specifications [LSM+89]. Wherever possible, the HTML format of the file was used and this file was then converted to a plain text format. Where this was not the case, PDF files were used and then converted to plain text. The text was converted completely to lowercase to remove duplicates that were based solely on case differences.

In initial trials, the resulting plain text file was filtered to remove anything other than alphanumeric characters. In later trials, the "words" that were composed solely of numbers were also excluded, as it was observed that the index of each document contained only key terms that were purely alphabetic in nature; i.e., contained only letters. The above technique of filtering out of non-alphanumeric characters and stand-alone numbers was used in [Hul03]. The pre-processing method is illustrated in Appendix A Listing A.2. See code A.2. The pseudocode for data pre-processing is described in Algorithm 1.

The document Index contained both the index (keyword) and also the section name, within the document, in which the particular keyword was found. Therefore in order to produce an index document consisting of only the index terms themselves, a Python script was run on the Index file to remove the section names. The relevant script can be found in Appendix A as Listing A.3. See code A.3.

The aircraft manual data consisted of multiple PDF files that were combined into one large plain text file. Extraneous material such as acknowledgments and publisher information had to be removed from the document since otherwise, a search for POS tags of the form $<NN>$ $<NN>$ would find invalid terms such as *"McGraw Hill"* or proper names of authors.

3.3 POS Tagging

POS or Part of Speech tagging was performed using the default Maximum Entropy Penn Treebank POS tagger that is available with the NLTK toolkit. POS tagging uses POS tag sequences as a "within collection" (as opposed to external) syntactic feature, in order to identify candidate key phrases.

Algorithm 1 Algorithm for Data pre-processing

Require: Input text file

Ensure: Pre-processed Input text file

textlines ←text file

for all non-alphanumeric characters **do**

remove character and replace with space character

textlines ←remaining characters

end for

while code snippet tagged for removal **do**

textlines ←textlines without code snippet

end while{Use the above WHILE statement if code snippets are to be removed from text}

$output file$ ←text lines

return Filtered output text file

Table 3.2: Frequency of POS Tags in the C Standards Document

POS Tag	Example Word	Frequency
NN	bit	8435
NNS	arrays	2456
JJ	manual	1431
RB	commonly	710
VBP	are	637
VBG	declaring	483
VBD	wrote	476
VBN	based	456
IN	of	325
VB	include	212
VBZ	describes	208
CD	five	193
MD	can	167
JJR	less	62
DT	a	40
JJS	smallest	23
RBR	earlier	9
WP$	what	9
CC	and	5
PRP	it	2
RP	up	1

NN (noun) is the most frequently used tag in the C standards document as can be seen from Table 3.2 and also from the figure 3.3. This is followed by *NNS* (noun plural) and then *JJ* (adjective), *RB* (adverb), *VBP* (verb singular present) and *VBG* (verb gerund or present participle).

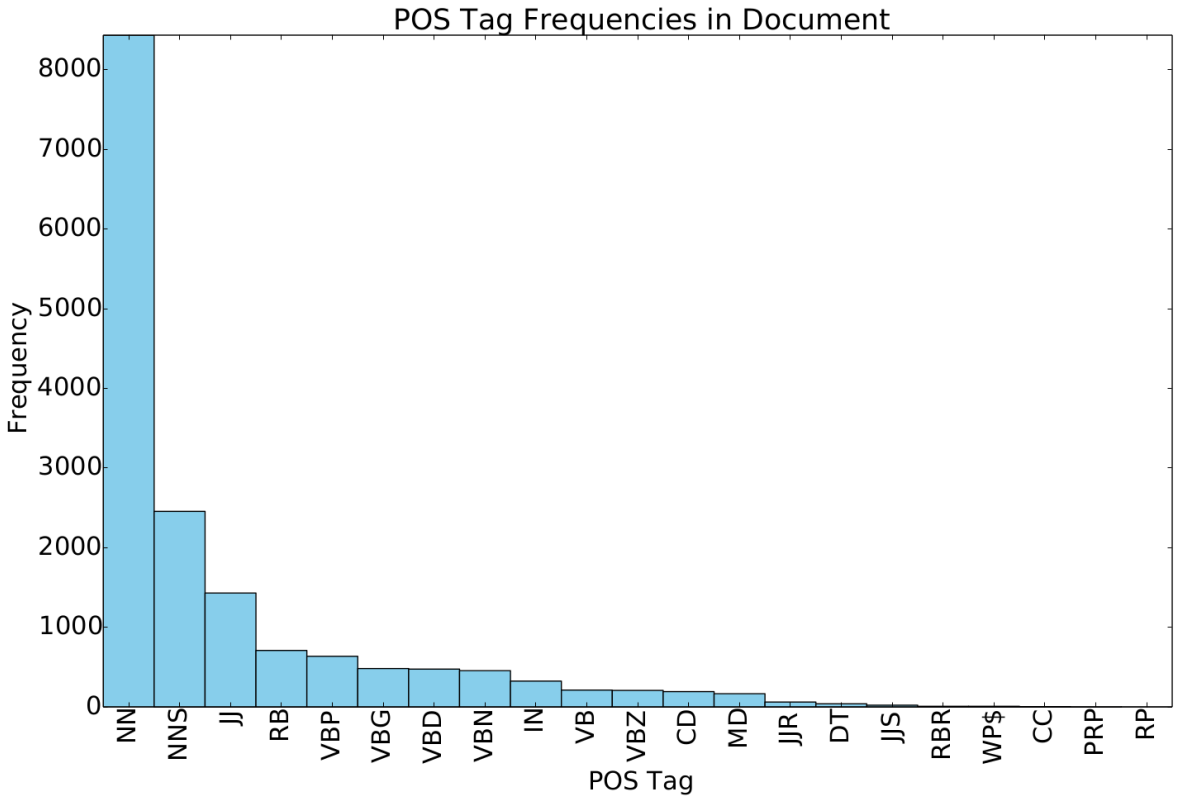

Figure 3.3: Frequencies of each type of POS Tag in the C Standards document

NN (noun) is the most frequently used tag in the C standards index as revealed by Table 3.3 and also Figure 3.4. This tag is followed by *NNS* (noun plural) and then *VBG* (verb gerund or present participle) followed by *JJ* (adjective) and *VBP* (verb singular present). The frequency of each **type** of POS tag in the C standards index are listed in Table 3.3. The *frequency* of each **sequence** of POS tag in the Index are shown in Table 3.4. The sequences of POS tags in the index were used to find the most appropriate expressions to use as filters on the body of the document.

While the frequency of nouns appears prominent among the index terms (as expected and discussed above) e.g. *arithmetic operators, operator precedence, program structure*, one must also consider the presence of verbs, which come a close second

Table 3.3: Frequency of POS Tags in the C Standards Index

POS Tag	Frequency
NN	244
NNS	213
VBG	61
JJ	40
VBP	31
VBD	14
RB	4
JJR	4
VBN	3
VBZ	3
IN	2
PRP$	1
RBR	1

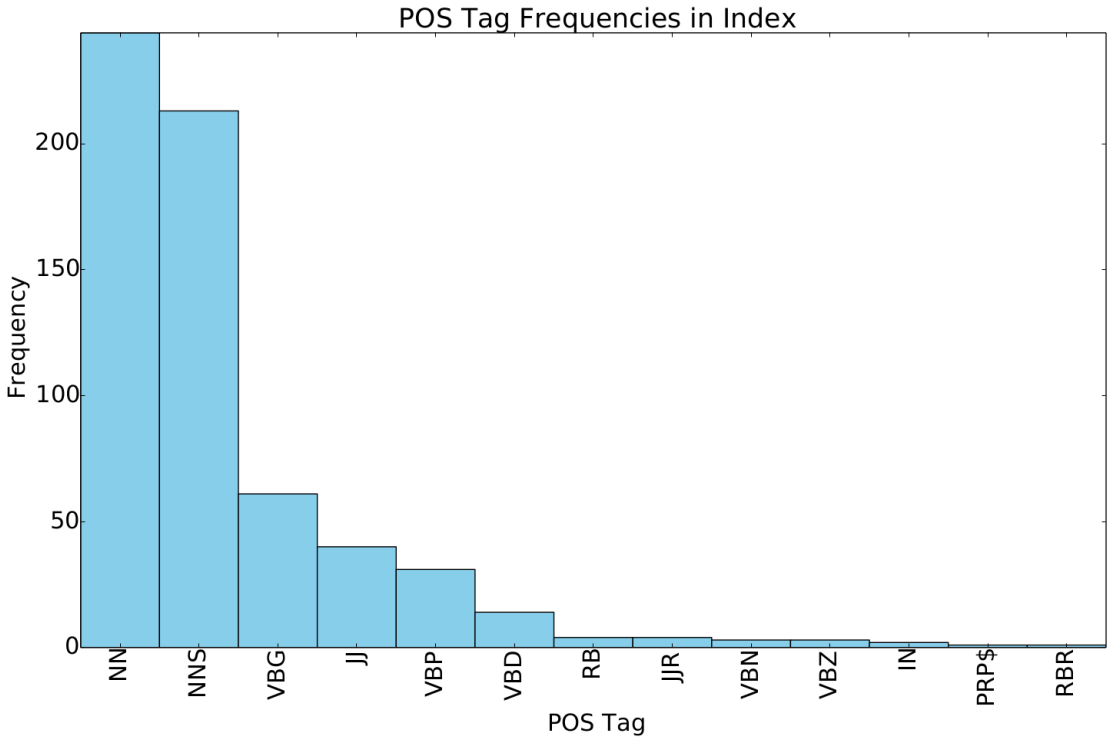

Figure 3.4: Frequencies of each type of POS Tag in the C Standards Index

e.g. *accessing array elements, initializing arrays, declaring pointers.*

3.3.1 Index Terms Flipping

In order to ensure consistent matching between terms as they appeared in the text of the document and terms that appeared in the index, index terms which used commas to alter the order of words were pre-processed by having their commas removed, and the order of words "flipped" so as to make them grammatically correct. "Grammatically" correct in this instance is taken to represent a meaningful, coherent phrase that can be inserted as it is into a full sentence. For instance, *"array elements, accessing"* was converted to *"accessing array elements"* or in POS terms, ***NNS NNS comma VBP*** would have been changed to ***VBP NNS NNS***. It was commented by Da Sylva that "human indexers often reword phrases in a document" [DS13]. The frequency of some of the most common of these **modified (flipped)** terms can be seen in Table 3.4. The POS sequences in the **unedited** index are shown in Table B.3 .

3.4 Candidate Detection

Candidate words were selected based on POS patterns. Candidate terms of one character length as well as non-alphanumeric terms were removed from consideration. The goal of this step is to avoid "incorrect" keyphrases while using pruning to create the smallest possible number of candidates [HN14]. This pruning is necessary because reducing the number of candidates increases the value of the precision measure. Automatic term extraction can therefore be considered a classification task. Each candidate term is either a key term or not [FPW+99].

Regular Expressions Rules for Candidate Detection

Regular Expressions based on patterns in the Index terms were used to find POS sequences. For instance, `<N*> <N*>` would match `<NN> <NN>` or `<N N>` etc. `<VB[DGP] <N.*>?` was used to find sequences of the form `<VBD> <NN>`, `<VBGD> <NN>` and so on.

Table 3.4: Frequency of **"flipped"** sequences of POS Tags in the C Standards Index

POS Tag	Frequency
NN NNS	31
NN NN	26
VBG NNS	22
VBG NN NNS	21
JJ NNS	18
NNS	17
VBG NN NNS IN NN	8
NN	7
VBD JJR NN	6
NNS IN NNS	6
NN NNS NNS	6
IN NN	5
NN NN NNS	5
NN NN NN NN	4
VBD NNS	4
NNS TO NNS	4
JJ NN NNS	4
NN IN NNS	4
NN VBD	3
NN JJ	3
NN JJ NNS	2
JJ NNS NNS	2
JJ NN	2
NNS VBP NNS	2
NN NNS NN	2
VBD NN NNS NN	2
JJ NN NNS NNS	2
RB VBZ	1
JJ NNS NN	1
RB VBP NNS NN	1

POS Tag	Frequency
JJ NN NN NN	1
VBG	1
JJ NN NN	1
VBD PRP$	1
VBD RB RB JJ NNS NN	1
VBG NN NNS NNS	1
RB JJ NNS NN	1
VBD VBN NNS NN	1
NNS NNS	1
JJ NN NN NNS	1
NNS IN JJ NNS	1
IN NN NN	1
NN VBG NN NNS	1
VBD RB JJ NNS NN	1
NNS VBG NNS	1
IN NNS NN	1
VBG IN NN NNS	1
IN NNS	1
NNS IN NNS NNS	1
NN VBZ IN NNS	1
JJ NN NNS NN	1
RB RB JJ NNS NN	1
NN NN NNS NNS	1
NN VBG	1
VBP NN	1
NN NN IN JJ	1
JJ NN NN NN NNS	1
NN VBD NN NNS	1
VBD JJ NN NNS NN	1

3.5 Stop Words and Common Words

When identifying key terms in technical documents, it must be remembered that general, yet very frequent words such as "looks" or "for" cannot be considered keywords. It has been observed by Church [CGHH91] that "the most common words cause the most trouble". These common words must therefore be filtered out of the list of technical key words. To accomplish this, the text is compared to a list of common words called "stopwords" and any terms from the text that also appear in the list of stopwords are excluded. The NLTK toolkit provides such a list in the form of the stopwords corpus. In addition, the least frequent terms from the text document (occurrence of three times or less) as well as word bigrams from the Brown Corpus in the *editorial* category, were added to the set of stopwords to refine the list of candidate terms. This list of custom stop words was also manually edited to include other selected words based on the C dataset. For the aircraft manual data set, we used both this C data custom list with no modifications and in turn, a custom list built specifically for the aircraft manual data for the sake of comparison. The Brown Corpus is an electronic corpus of English, which was created in 1961 at Brown University. It contains text from 500 sources and is categorized by genre. A method of selecting and using high frequency words from a sample of their corpus as stop words was used by Frantzi et al. [FAM00]. The effect of stopword removal (number of tokens) can be seen in Table 3.5 and this table also shows that a significant number of words in the document were in fact stopwords.

Table 3.5: Effect of stopword removal on C Standards document

Process	Without filtering	Stopwords removed
Number of Tokens	29171	17510

Table 3.6: Effect of stopword removal on Aircraft Manual document

Process	Without filtering	Stopwords removed
Number of Tokens	114948	113872

Five different types of these stopwords and common words were used. Their

sources are listed below:

1. NLTK
2. Brown Corpus
3. Technical terms (custom list)
4. Operator terms
5. Least frequent words in the given text

The first set was a basic list of stopwords provided by NLTK. The second was a list of stopwords from the Brown corpus. Next a custom list that consisted of a set of common technical words that were prevalent in the technical domain (one for the C data and one for the aircraft manual data) was manually created. The aircraft manual data was processed using its own custom list, and then it was also processed using the unmodified C custom list for comparison. In addition, a list of words that could be considered stopwords but were actually valid key terms was also created. The Operators list shown in table 3.7 is a list of common words that could be considered stopwords in a generic domain, but were in fact specialized terms and therefore valid key words in the technical domain.

Domain specific stop words, which are stop words that were based solely on the frequency of words in the particular document, were also used. The Least frequent words list was found by calculating the number of occurrences of each word in the document. Then NLTK stopwords were removed from the resulting list. Following this step, a threshold was set at a particular value e.g. 3 or 10, and all words of that frequency or less were added to the stop words list. For the aircraft manual data, one list was created for words that appeared in the aircraft manual text with a frequency of three or less and another for words that appeared in the aircraft manual text with a frequency of ten or less. Numbers were excluded from these "frequency" stop word lists. The pseudocode for the construction of the list is shown in Algorithm 2.

These modifications such as custom stop words and common words are necessary because the prior research indicates that "performance can be boosted by exploiting

domain specific information about the likelihood of keyphrases" ... [and] "the modification significantly improves the quality of keyphrases extracted" [FPW+99]. Thus domain knowledge increases the effectiveness of keyword selection [Tur00].

Algorithm 2 Algorithm for Construction of Least Frequency words Stop Words List

Require: Input text file and threshold value

Ensure: Pre-processed Input text file

textlines ←text file
textwords ←textlines
textwords ←textwords in lowercase
for all punctuation characters **do**
  remove character and replace with space character
  textwords ←remaining characters
end for
if word in NLTK Stop Words list **then**
  textwords ←textwords without word
end if{Use the above IF statement to remove NLTK Stopwords from the text}
for all words in textwords **do**
  if frequency of word in text file <= threshold value **then**
    $leastfreqentwordsstopwordlist$ ←word
  end if
end for
$outputfile$ ←leastfreqentwordsstopwordlist
return Stop words list of Least Frequent words

Stop words were extracted after candidate terms were found, so as not to prematurely exclude valid key terms. Stop words lists could also include stop word *phrases* i.e. stop words of more than one word in length. Candidate terms that were exclusively made up of stopwords or that *contained* stopwords as part of their structure were removed from consideration. For instance, if the phrase *"spot check"* was included in a stop word list, then the candidate terms *"spot", "check", "spot check", "spot clean", "check mate"* and *"routine check"* would be removed from the final list of terms.

Table 3.7: Operator terms for C Standards document

Term	Term
and	union
or	type
not	functions
structure	cast
character	casts
array	field
assignment	fields
type	operators
comparison	bit

3.6 Keyword Selection

In preliminary trials, a small sample of the text was used as a proof of concept. The relevant script can be found in Appendix A as Listing A.1. See code A.1. Tokens that were extracted from this sample text are shown in Figure 3.6. The POS tags that were extracted from this sample text appear in Figure 3.5.

In Experiment set A, the focus was on finding exact matches, which has been noted in the literature to be quite a harsh rule for evaluation of the system. In certain experiments of Experiment set B, the above restriction was relaxed, as variations in candidate terms were also allowed as matches. The pseudocode for Keyword Selection is described in Algorithm 3. Note that when "word" is used in the algorithm this can also refer to "terms" i.e. stop words, common words and operators can consist of more than one word per term.

3.7 Additional Data Sets

The second dataset that was used was a manual of aircraft maintenance guidelines. This new dataset provided a contrast to the C Programming Language Standards. The same methods described above were implemented on this second dataset to determine the degree of transfer learning that may occur and whether the methodology had to be modified to accommodate this new data set.

Algorithm 3 Algorithm for Keyword selection

Require: Set of text candidate terms one term per line

Ensure: Set of key terms one term per line

stopwords ←NLTK stopwords list

operators ←operator terms list

commonwords ←common words list

bigramslist ←bigrams from Brown corpus

stopwords ←stopwords and commonwords {Use this step if needed by the particular experiment}

stopwords ←stopwords and bigrams list {Use this step if needed by the particular experiment}

stopwords ←stopwords *without* Operators {Use this step if needed by the particular experiment}

for all terms in candidate terms list **do**

tokenize line

remove non-alphanumeric characters from candidate term

convert candidate term to lowercase

if term exists in stopwords **then**

candidatelist ←candidate list without candidate term

end if

end for

keywords ←candidate list

return Set of keywords and Statistics for calculation of P, R, F values

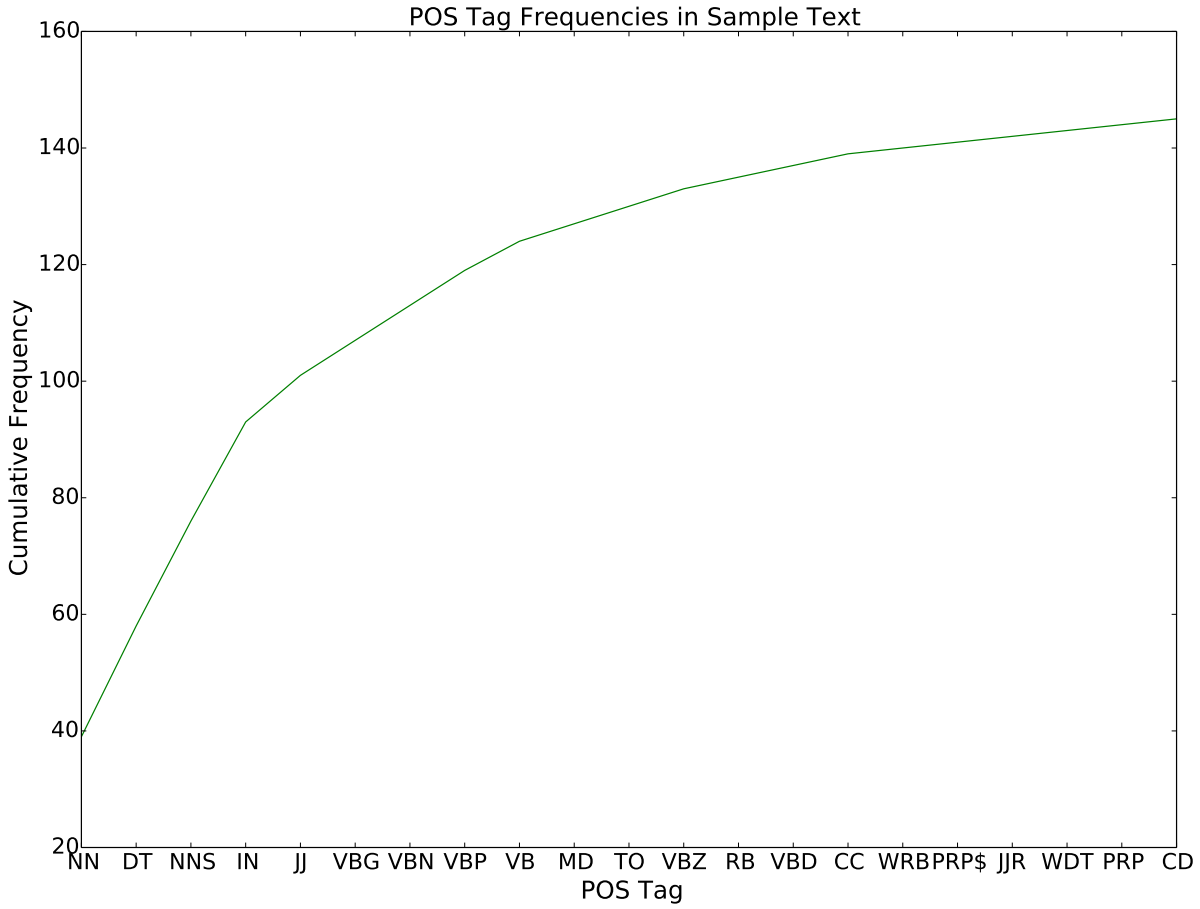

Figure 3.5: Tags in C Standards sample text

3.8 System Comparison

In order to further evaluate our system's performance, we compared our results to those of the TBXTools system [OGVG15]. TBXTools is a "free automatic terminology extraction tool that implements linguistic and statistical methods for multiword term extraction". TBXTools is implemented in Python and allows statistical methods of term extraction while providing a basic list of stopwords.

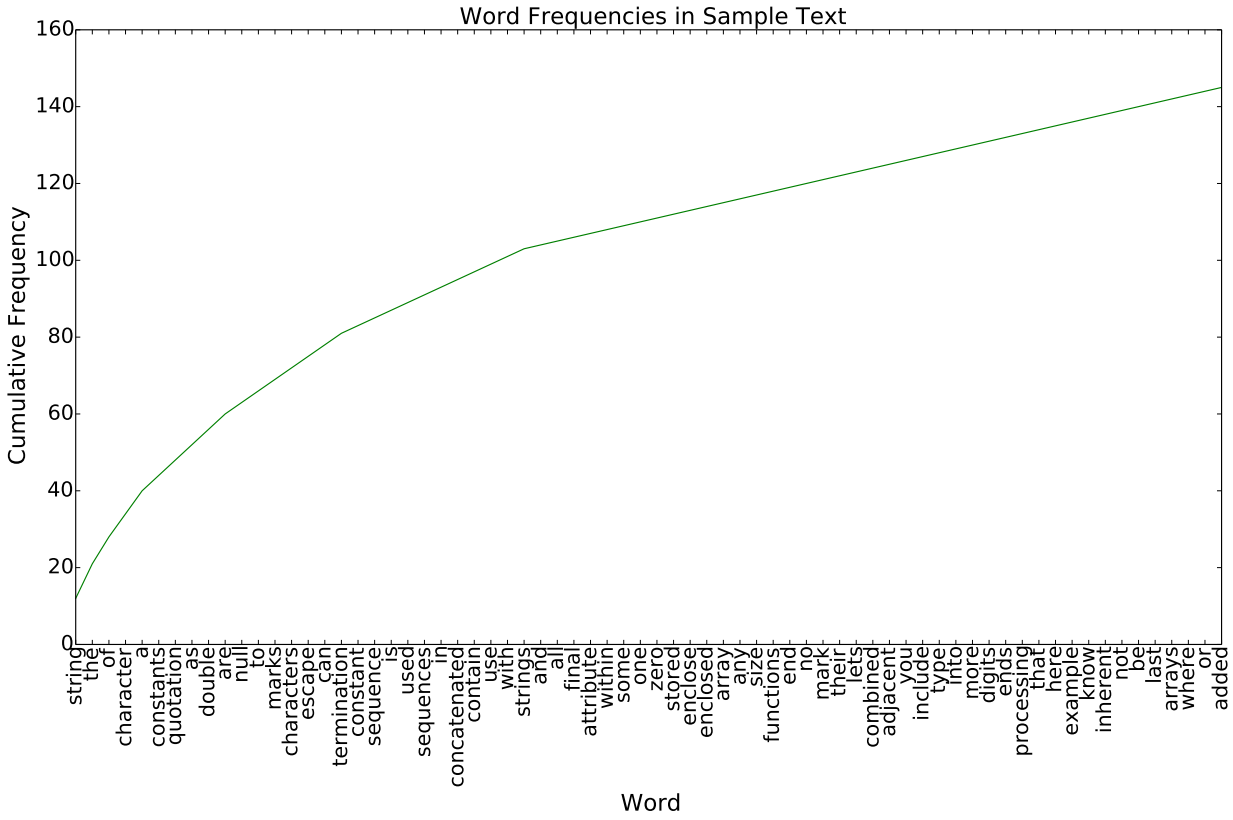

Figure 3.6: Tokens in C Standards sample text

3.9 Methodology Summary

In this chapter we present our methodology and describe our system architecture. Our overall methodology and system architecture is represented in Figure 3.1. The main components are Data Pre-processing, POS Tagging, Candidate Detection, Stop words and Common words and Keyword Selection. We have explained the rationale behind the methods used while considering ATE as both a search for collocations as well as a supervised learning task. The use of POS Sequences, especially the importance of noun phrases in recognizing candidate terms, was emphasized. The evaluation method used was to compare the list of generated keywords to the list of all the index terms or "gold standard", which has been recognized in the literature as

being a severe performance measure. Data pre-processing was described for both data sets and differences between the characteristics of both data sets were mentioned. It was noted that Index terms flipping was used to ensure data consistency between the index and text terms. The use of custom Stop words lists were explained and their value in pruning candidate terms was recognized. Finally our system was compared with a different system (TBXTools), to produce another evaluation measure. In the next chapter we lay out our experimental set-up and results. In the conducting of our experiments we used the NLTK Toolkit, a basic text editor, and Python to manipulate text files and create statistics of performance measures.

Chapter 4

Experimental Results

4.1 Experiments Overview

This chapter presents our experiments. Experiments were divided into sets A, B and C as described below. The C Standards document was designated as Data Set 1 while the Aircraft maintenance guidelines manual (aircraft manual data) was designated as Data Set 2. Experiment Set A was focused on the C standards document and on attempting to replicate the back-of-the-book index that had been provided with this document. Experiment Set B was designed to measure **transfer learning** i.e. how much of the information learned from manipulation of the previous data set could also be applied to this new data set. In Experiment Set C, in order to compare the performance of our system with that of another, we used the TBXTools system to evaluate the resulting candidate terms when compared to our results .

4.1.1 Rationale for Data Set Selection

The C Standards are a venerable, curated, important and well established dataset that has been in existence for a long period of time. The C data set was used to establish and refine our methodology. The aircraft manual data set is a newer, more specialized dataset that was used to validate our system performance by transferring our methodology to a different text.

4.2 Implementation Tools

The NLTK toolkit (Natural Language Toolkit) described by Bird et al. [BKL09] was used to process the data. This toolkit provides modules to study and manipulate a corpus of data. The research used NLTK version 2.0.4 and Python version 2.7.13./2.7.5. The experiments were run using a Windows 8.1, 64-bit Intel i7 2.40 GHz processor with 8 GB RAM, and on an Intel i5 3GHz 64 bit Linux machine running

Fedora 20.

4.3 Experiment set A — Data set 1

The C data set was used in preliminary trials and to refine our methodology. The original C standards document was manually split into a text file and an index file. The C programming language standards text file consisted of 28,288 distinct alphanumeric words and 1,81,407 tokens overall. The details of the individual experiments are discussed below.

Experiment A1 (POS pattern: <NN> <NN>) The WordNetLemmatizer was used on the index file. The observed performance is indicated in Table 4.1.

Experiment A2 (POS pattern: <N*> <N*.>?) The WordNetLemmatizer was used on the index file.

Experiment A3 (POS pattern: <VB[DGP] <N.*>?) The WordNetLemmatizer was used on the index file. The observed performance is indicated in Table 4.1.

Experiment A4 (POS patterns: *<N*> <N*.>?* and *<VB[DGP] <N.*>?* together) The WordNetLemmatizer was used on the index file. The observed performance is shown in Table 4.1.

Experiment A5 (POS pattern: <N*> <N*.>?) The WordNetLemmatizer was used on the index file. Also, C language code examples which contained irrelevant words that were used merely for illustration, such as *rye bread*, *pumpernickel*, and *tutti frutti ice cream*, were manually filtered out of the document. This pre-processing task of removing irrelevant code snippets can be considered the "removal of noisy symbols" as mentioned in [MFO16]. Experiment 5 was conducted using a **small** portion of the full C programming standards document. The observed performance is indicated in Table 4.1.

For instance, as indicated in the following segment of text:

```
 Here is an example of defining a simple union for holding an
integer value and a floating point value:  <code union numbers {
```

```
int i; float f; };> That defines a union named numbers, which
contains two
members, i and f, which are of type int and float, respectively.
```

was converted to

```
 Here is an example of defining a simple union for holding an
integer value and a floating point value:  That defines a union
named numbers, which contains two members,i and f,which are of
type int and float, respectively.
```

Experiment A6 (POS pattern: <N*> <N*.>?) The WordNetLemmatizer was used on the index file. Also, code examples were filtered out of the document. The observed performance is shown in Table 4.1 .

Experiment A7 (POS pattern: <NN> <NN>) The pattern is applied on **both** the index and text file. Duplicates in each file were removed using the *set* command. Also, code examples were manually filtered out of the document.

Experiment A8 (POS pattern: <NN> <NN>) The pattern is applied on **both** the index and text file. Duplicates in each file were removed using the *set* command. Also, terms in both files were converted to lowercase and code examples were manually filtered out of the document.

Experiment A9 (POS pattern: <NN> <NNS>) The pattern is applied on both the index and text file. Duplicates in each file were removed using the *set* command. Also, terms in both files were converted to lowercase and code examples were manually filtered out of the document.

Experiment A10 (POS pattern: <VBG> <NNS>) The pattern is applied on both the index and text file. Duplicates in each file were removed using the *set* command. Also, terms in both files were converted to lowercase and code examples were manually filtered out of the document.

Experiment A11 (POS pattern: <VBG> <NN> <NNS>) The pattern is applied on both the index and text file. Duplicates in each file were removed using the *set* command. Also, terms in both files were converted to lowercase and code examples were manually filtered out of the document.

Experiment A12 (POS pattern : <NN> <NNS>) The pattern is applied on both the index and text file. Duplicates in each file were removed using the *set* command. Also, terms in both files were converted to lowercase and code examples were manually filtered out of the document. An expanded list of stopwords was used. Terms that occurred the least frequently in the document (three times or less), were also added to the stopwords list.

Experiment A13 (POS pattern: <NN> <NNS>) The pattern is applied on both the index and text file. Duplicates in each file were removed using the *set* command. Also, terms in both files were converted to lowercase and code examples were manually filtered out of the document. An expanded list of stopwords was used. An extended list of "operator" words which consisted of words that were incorrectly regarded as stopwords and therefore had to be removed from the stopwords list, was used.

4.3.1 Experiment set A Summary

A summary of the experimental results in terms of counts is presented in Table 4.1. The column labeled *Tokens* indicates the number of unique words (reflecting the approximate text length) that were used in the particular experiment. It can be seen that Experiment number A5 was run on a smaller text sample while using the same POS sequence as Experiment A3. The *Index Terms* refers to the number of index terms that corresponded to the particular POS sequence of that row, and the *Candidates* are the number of candidate terms which were found by using that POS sequence. *Matched* refers to the number of candidates of that particular POS sequence which matched actual index terms. As a larger number of appropriate stopwords (common words) were used, the performance measures improved. Also stopwords that were actually valid index terms and which had been incorrectly removed by the

Table 4.1: Experimental Results — Counts of C Standards Data

Exp.#	Tokens	POS	Index Terms	Candidates	Matched
A1	28288	<NN> <NN>	251	1134	76
A2	28288	<N*> <N*.>?	251	5157	83
A3	28288	<VB[DGP] <N.*>?	251	1087	18
A4	28288	*Exp.# 2 and 3 together*	251	6244	84
A5	3031	<N*> <N*.>?	181	210	18
A6	28288	<N*> <N*.>?	181	1065	62
A7	28288	<NN> <NN>	20	352	15
A8	28288	<NN> <NN>	20	352	16
A9	28288	<NN> <NNS>	20	178	16
A10	28288	<VBG> <NNS>	12	18	3
A11	28288	<VBG> <NN> <NNS>	12	13	1
A12	28288	<NN> <NNS>	20	108	13
A13	28288	<NN> <NNS>	20	71	13

common words filter, later needed to be added back into the list of candidate terms in order to improve the performance measures. Examples of such terms are *addition*, *custom* and *option*.

4.4 Additional Data Set

We then used a second data set to see if our methodology would transfer to this new data set. The second data set that was used was a manual of aircraft maintenance guidelines. The frequency of individual POS tags in the index of the document are shown in Figure 4.1. The frequency of POS sequences in the index of the document are shown in Table B.1. It can be seen that noun phrases form a large majority (about 50%) of the index terms. The occurrence of other types of terms declines sharply after the noun phrases have been accounted for. This is dissimilar to the C Programming Standards document where there were also a large number of verb-based index terms.

4.5 Experiment Set B — Comparison with Data set 2

Experiment B1 (`Acronyms`) Acronyms were extracted from both the index file and the text file using a regex expression to detect sequences of capital letters that may also be interspersed with lower case letters, periods, slashes and ampersands e.g. *PP&C, ETOPS, IATA*. It should be noted that to facilitate the retrieval of

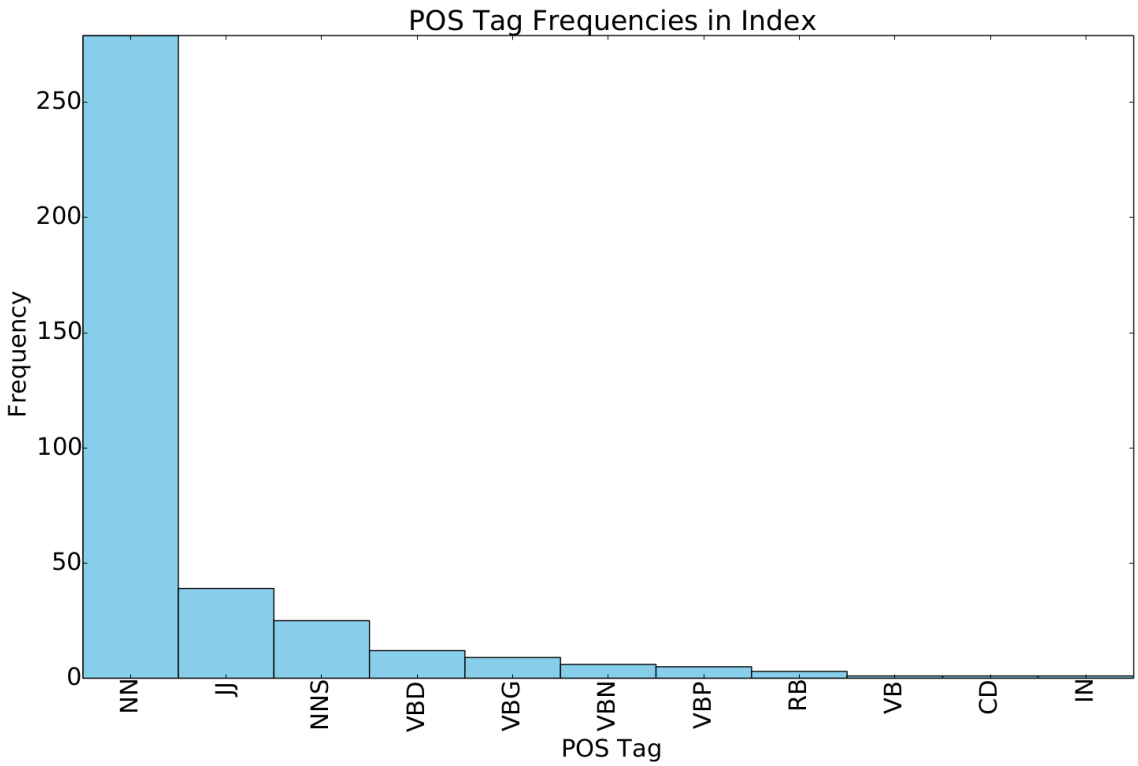

Figure 4.1: Frequencies of each type of POS Tag in the Aircraft Maintenance Guidelines manual

acronyms, in this case, unlike the C Programming Standards document, the text was not converted to lowercase.

Previously, candidates of one character length were excluded as they were unlikely to be key terms in the C Programming Standards document. However for the aircraft manual data, the candidate terms that contained words of one character length could not be excluded, as some of these candidate terms were valid key terms e.g. *"C check"*. This necessitated a modification to the list of valid stop words that were used to prune the set of candidate terms.

Experiment B2 (POS pattern: <NN> <NN>) When the pattern *<NN> <NN>* was used on the aircraft manual data (similar to experiment A1 on the C Programming Standards data), the results indicated high recall but low precision due to a large number of valid candidate terms being rejected as they contained what

were incorrectly determined to be stopwords.

Experiment B3 (POS pattern: <NN> <NN>) While the pattern *<NN> <NN>* was used on the aircraft manual data, the stopwords list was not used, and so a larger number of matches were found. Recall improved dramatically, and there were also improvements in precision and f-measure values as well. This demonstrates that proper stop word selection plays a vital role in key term extraction.

Experiment B3b (POS pattern: <NN> <NN>) While the pattern *<NN> <NN>* was used on the aircraft manual data, the stopwords list was *not* used, and so a larger number of matches were found. Recall improved greatly from Experiment B2, and there were also improvements in precision and f-measure values as well. This indicates that proper stop word selection plays a vital role in accurate key term extraction. In addition, variations in matched terms were allowed. *"on-the-job-training"* was considered equivalent to *"job training"* and *"failure effect"* was considered to be the same as *"failure effects"*. These variations were found by manual inspection of the text candidate terms and the index terms.

Experiment B3c (POS pattern: <NN> <NN>) While the pattern *<NN> <NN>* was used on the aircraft manual data, the *same* custom stopwords list as was used for the C data was used for the aircraft maintenance guidelines manual data. The NLTK stop words list and the operators list were used, but the Brown corpus editorial stop words were *not* used. Recall improved greatly from Experiment B2, and there were also improvements in precision and f-measure values as well. This shows that proper stop word selection plays a vital role in accurate key term extraction. In addition variations in matched terms were allowed. *"on-the-job-training"* was considered equivalent to *"job training"*. These variations were found by manual inspection of the text candidate terms and the index terms.

Experiment B3d (POS pattern: <NN> <NN>) While the pattern *<NN> <NN>* was used on the aircraft manual data, the *same* custom stopwords list as was used for the C data was used for the aircraft maintenance guidelines manual data. The NLTK stop words list and the operators list were used, and the Brown corpus

editorial stop words were used. Recall improved greatly from Experiment B2, and there were also improvements in precision and f-measure values as well. This indicates that proper stop word selection plays a vital role in accurate key term extraction.

Experiment B3e (POS pattern: <NN> <NN>) While the pattern *<NN> <NN>* was used on the aircraft manual data, the stopwords list consisting of words of frequency less than three (excluding numbers) from the aircraft manual text were used on the aircraft manual data. The NLTK stop words list and the operators list were used, but the Brown corpus editorial stop words were not used. Variations were not allowed. Recall improved greatly from Experiment B2, and there were also improvements in precision and f-measure values as well. This indicates that proper stop word selection plays a vital role in accurate key term extraction.

Experiment B3f (POS pattern: <NN> <NN>) While the pattern *<NN> <NN>* was used on the aircraft manual data, the stopwords list consisting of words of frequency less than ten (excluding numbers) from the aircraft manual text was used on the aircraft manual data. The NLTK stop words list and the operators list were used, but the Brown corpus editorial stop words were not used. Variations were not allowed. Recall improved greatly from Experiment B2, and there were also improvements in precision and f-measure values as well. This indicates that proper stop word selection plays a vital role in accurate key term extraction.

4.5.1 Experiment set B Summary

A summary of the experimental results in terms of counts is presented in Table 4.2. The column labeled *Tokens* indicates the number of unique words (reflecting the approximate text length) that were used in the particular experiment. The *Index Terms* refers to the number of index terms that corresponded to the particular POS sequence or Method of that row, and the *Candidates* are the number of candidate terms which were found by using that POS sequence. *Matched* refers to the number of candidates of that particular POS sequence which matched actual index terms. Using the same Common words and Operator words from the C Standards data proved to be less effective in extracting terms from the aircraft manual data than the C Standards

Table 4.2: Experimental Results — Counts of Aircraft Manual data

Exp.#	Tokens	POS Sequence or Method	Index Terms	Candidates	Matched
B1	67735	*Acronyms*	91	267	77
B2	67735	<NN> <NN>	8	802	1
B3	67735	<NN> <NN>	8	2787	5
B3b	67735	<NN> <NN>	8	2787	7
B3c	67735	<NN> <NN>	8	2352	6
B3d	67735	<NN> <NN>	8	776	1
B3e	67735	<NN> <NN>	8	1892	5
B3f	67735	<NN> <NN>	8	1377	5

data did. Allowing for variations greatly improved recall. Only one index term was missed (not matched) in experiment B3b as shown in Table 4.2. This is a result of permitting variations.

4.6 Experiment Set C — System Comparison

The TBXTools software was used to extract key terms and the performance was compared with our methodology. By default the TBXTools statistical package produces bigrams, so longer candidate expressions were not considered in initial trials.

Experiment C1 (`TBXTools`) Candidate bigrams were extracted from the C Standards document (unfiltered), using statistical analysis provided by the TBXTools software. The unfiltered text was not pre-processed to remove code snippets, therefore irrelevant terms such as *"pumpernickel bread"* and *"fruit"* when used in code examples, were allowed to remain in the text.

Experiment C2 (`TBXTools`) Candidate bigrams were extracted from the C Standards document (filtered), using statistical analysis provided by the TBXTools software. The filtered text was pre-processed to remove code snippets, therefore irrelevant terms such as *"pumpernickel bread"* and *"fruit"* when used in code examples, were removed from the text. This is analogous to experiment A5 and later Set A experiments.

Experiment C3 (`TBXTools`) Candidate bigrams were extracted from the aircraft manual document (filtered), using statistical analysis provided by the TBXTools software.

Table 4.3: Experimental Results — TBXTools

Exp.#	Tokens	Method	Index Terms	Candidates	Matched
C1	28288	*TBXTools statistical unfiltered*	179	1087	49
C2	28288	*TBXTools statistical filtered*	179	779	52
C3	67735	*TBXTools statistical filtered*	157	6100	42

4.7 Overall Experiments Summary

The above experiments considered specific POS sequences and characteristics. However we also studied the overall performance of our system in order to give a high level view of its efficiency. For the overall experiment with C Standards data, the POS sequences used were: `<NN> <NN>` , `<VGB> <NNS>`, `<VGB> <NN> <NNS>`, `<N*> <N*.>?`, `<JJ> <NNS>` and `<NNS>`, which cover the most frequent sequences of POS Tags in the index. For the overall experiment with aircraft manual data the POS sequences used were `<NN> <NN>` and *Acronyms* which cover the most frequent sequences of POS Tags or index term characteristics in the index.

Overall experiment counts for the C data and aircraft manual data using our method (denoted by *ATE*) are presented in Table 4.4.

Table 4.4: ATE Overall Experimental Results — Counts

Exp.#	Tokens	Index	Candidates	Matched
C data	28288	178	325	64
Aircraft Manual data	67735	99	3054	84

Basic statistics about the experiments that were conducted are presented in this section. Evaluation measures are presented in the next chapter. In the next chapter we discuss the significance of our results and present a method to evaluate those results.

Chapter 5

Discussion and Evaluation

The C programming standards documents included an index, which was used as the gold standard to measure how well the system performed. The aircraft maintenance guidelines manual contained a glossary which served the same purpose. A standard approach to the calculation of evaluation metrics for ATE was laid out by Kim et al. [KMKB10, KMKB13] and Hasan et al. [HN14] as well as Tjong Kim Sang et al for the CoNL-2003 shared task (named entity recognition) [TKSDM03] and [KU96]. This approach involves mapping the key phrases in the "gold standard" document (i.e. the index file) to the key phrases that are output by the system, using an exact match. The mapping is then scored using precision, recall and F-measure values. The "gold standard" is a pre-built list of reference terms that provides "reproducibility of results, tunability of parameters, and comparison between different methods on one dataset" according to Astrakhansev [AFT15].

Therefore the methods of evaluation used in the experiments were:

- precision,
- recall, and
- F-measure

Precision is the measure of *correctness* or the percentage of positives out of all returned keywords:

$$P = \frac{TP}{TP + FP} \tag{5.1}$$

Recall is the percentage of true positives out of all relevant keywords:

$$R = \frac{TP}{TP + FN} \tag{5.2}$$

F-measure is a weighted harmonic mean between precision and recall:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{5.3}$$

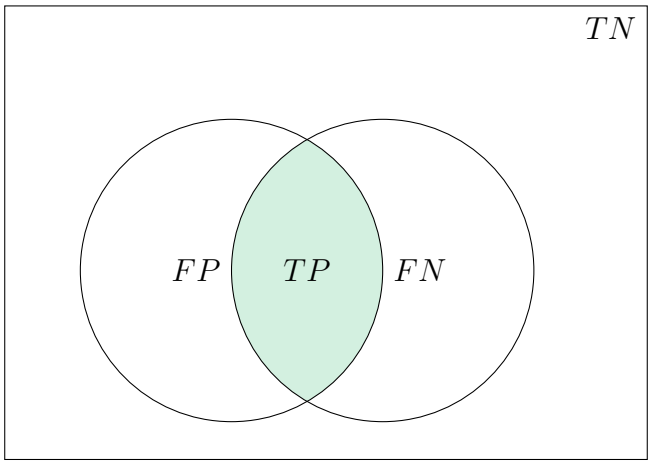

Figure 5.1: Evaluation Diagram

In general, we set

$$\beta = 1$$

in order to make precision and recall have equal weights, and thus we obtain:

$$F = \frac{2PR}{P+R} \tag{5.4}$$

where TP = True Positives, FP = False Positives and FN = False Negatives.

TP refers to True Positives, which are items that are both returned (identified) and relevant, *FP* stands for False Positives, which are items that are returned (identified) but not relevant and *FN* represents False Negatives which are items that are not returned but are still relevant. A description of the elements of each of these sets is shown in graphical form in the Venn Diagram of Figure 5.1

The question of how to calculate recall must be considered. Milios et al. calculated recall in the following manner: First, the terms above a frequency threshold of 3 are taken as candidate terms. Second, the number of real terms in this list are counted and thirdly, a *C-value* threshold of 0 is used to filter the terms further. "Recall is then taken to be the ratio of terms after step 2 to the number of terms after step 1." [MZHD03] Hulth calculated recall by considering the number of manually created key phrases [Hul03]. While recall can be calculated based on a set of given gold standard index terms, when the document has no index this calculation becomes more problematic.

The experimental performance measures are presented in Table 5.1. The use of more relaxed POS sequences e.g. <N.> <N.>* resulted in a greater number of candidates being identified, but only a small number of these candidates were actual index terms. Using more restrictive POS sequences e.g. <NN> <NN> provided better matches between the text and index terms. The values are presented in decimal form i.e. **0.3347 refers to 33.47%**

Table 5.1: Experimental Results — C Standards Data Performance

Exp.#	Precision	Recall	F-measure
A1	0.0670	0.3028	0.1097
A2	0.0161	0.3307	0.0307
A3	0.0166	0.0717	0.0269
A4	0.0135	0.3347	0.0259
A5	0.0857	0.0994	0.0921
A6	0.0582	0.3425	0.0995
A7	0.0426	0.7500	0.0806
A8	0.0455	0.8000	0.0860
A9	0.0899	0.8000	0.1616
A10	0.1667	0.2500	0.2000
A11	0.0769	0.0833	0.0800
A12	0.1204	0.6500	0.2031
A13	0.1831	0.6500	0.2857

A difficulty arises in POS tagging when words have ambiguous meanings. For instance, *book* can be both a verb (as in *book a train ticket*) as well as a noun (*the book was on the table*). The built-in POS tagger that was available with the NLTK software was therefore lacking in accuracy when it came to certain terms in our domain. Also, there was a discrepancy between tags in the text file and in the index file due to ambiguity of word usage in various sentences as opposed to the index file. It should be noted that the text file consisted of whole paragraphs while the index file comprised of index terms that were listed one term per line, which may indicate that the POS tagger had less information on which to base its tagging decision when tagging the index file as compared to the text file.

It has been discussed earlier that the state of the art performance in ATE is still low. State of the art values for Precision in general vary between 0.27 and 0.35, Recall varies between 0.28 and 0.66, while F-measure (which decreases as the

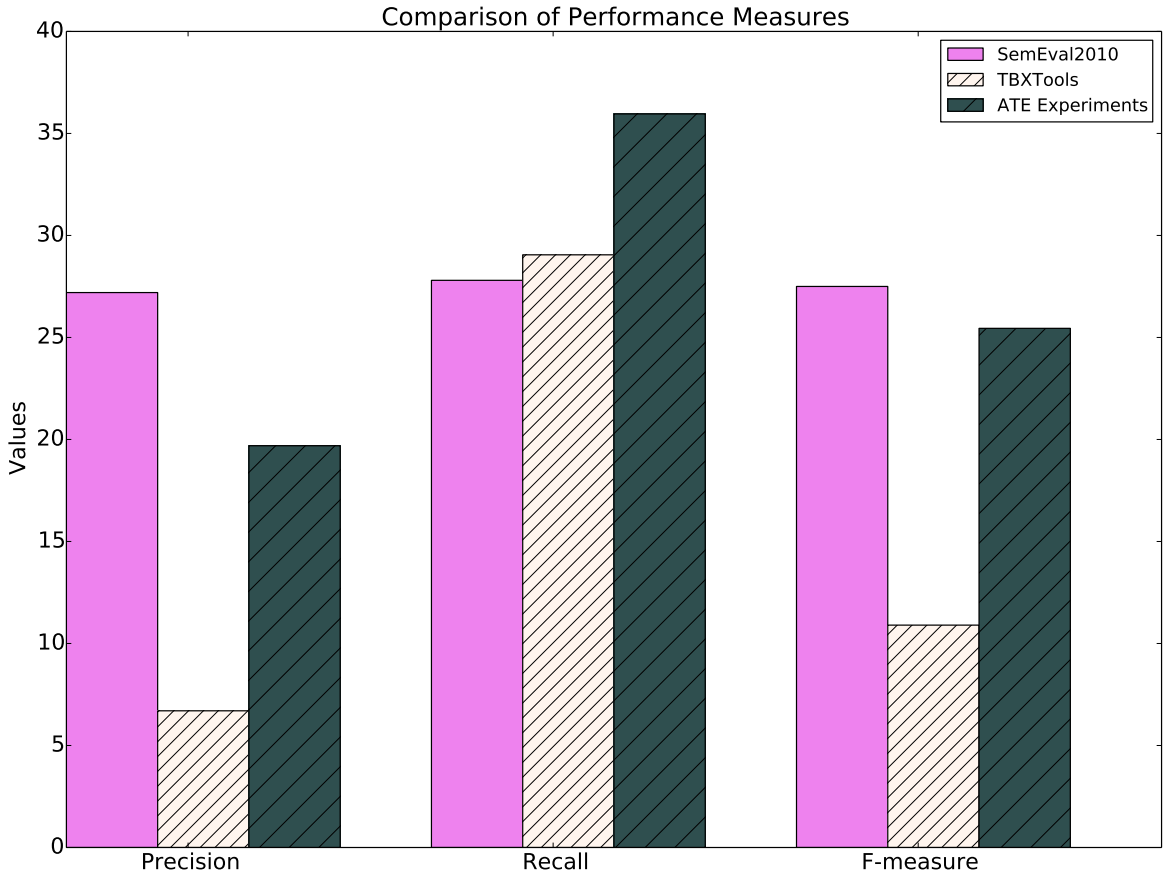

Figure 5.2: Performance Measures Comparison on C Standards Data

document length increases) falls between 0.27 and 0.45, as explained by Hasan et al. [HN14] . For performance measures on scientific papers in particular (SemEval 2010) [KMKB13], which corresponds most closely to our dataset, the corresponding values are 0.27, 0.28 and 0.28 respectively. The highest obtained performance measures for our experiments (in percentage values) are shown in Figure 5.2, which shows that our results are comparable to the previously mentioned state of the art performance values.

The fact that "keyphrase extraction is a subjective task" was observed by Kim et al. and it was noted that even "the top performing systems return F-scores in the upper twenties" [KMKB10, KMKB13]. Indeed it has been remarked that the actual

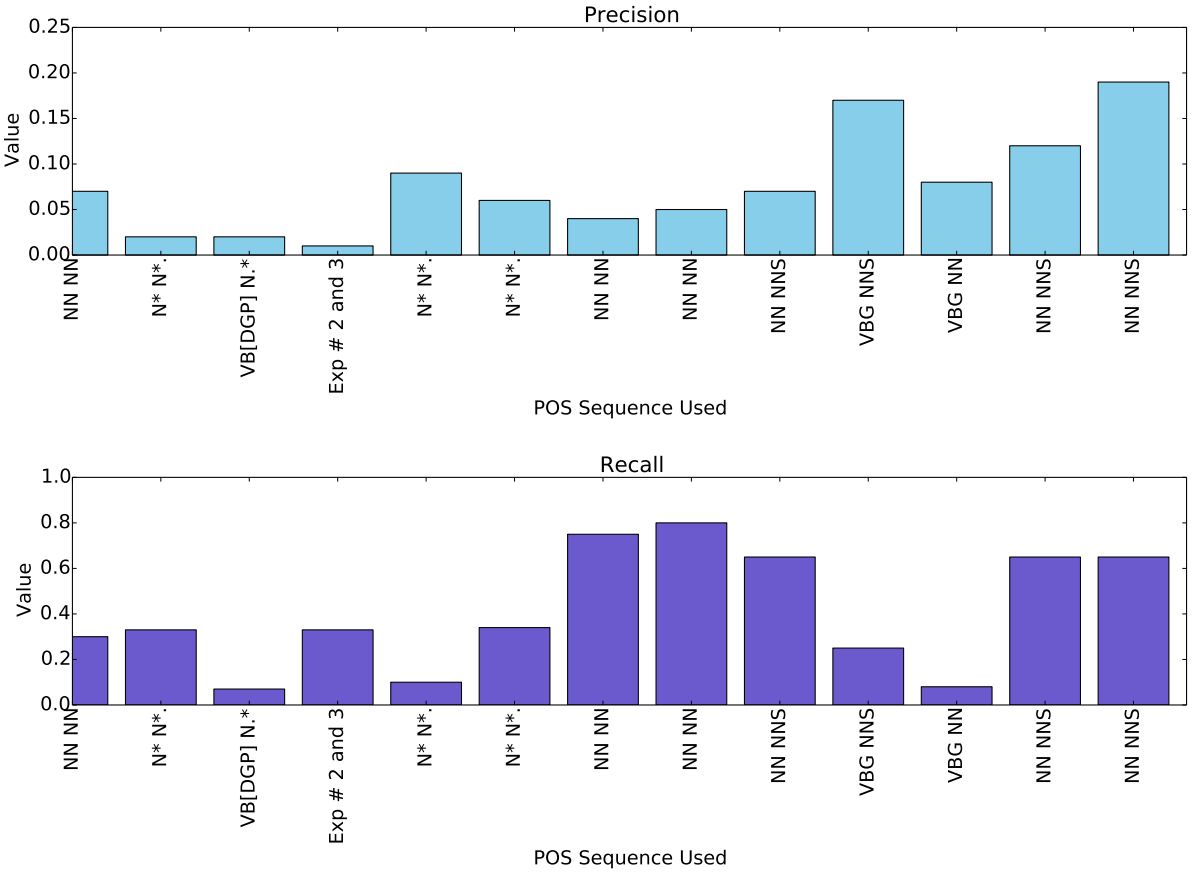

Figure 5.3: Performance Measures — Precision and Recall on C Standards Data

precision value has no real meaning, but instead we should ask "what percentage of keyphrases ... are acceptable to a human reader" [Tur00]. Thus Turney emphasizes that the actual precision and recall values are less important than what they represent.

Although precision, recall and F-measure are standard evaluation measures, Astrakhansev et al. [AFT15] note that:

> *"The formulation of the term recognition problem is [itself]...far from being entirely formal, which makes it quite difficult to compare term recognition methods and to evaluate their efficiency. As a result there are currently no commonly-accepted datasets and methodologies for efficiency evaluation, and developed methods are often domain- and application-specific."*

As expected, precision varies inversely with recall as shown in Figure 5.3. This trade-off becomes inevitable as returning more candidates improves recall but lowers

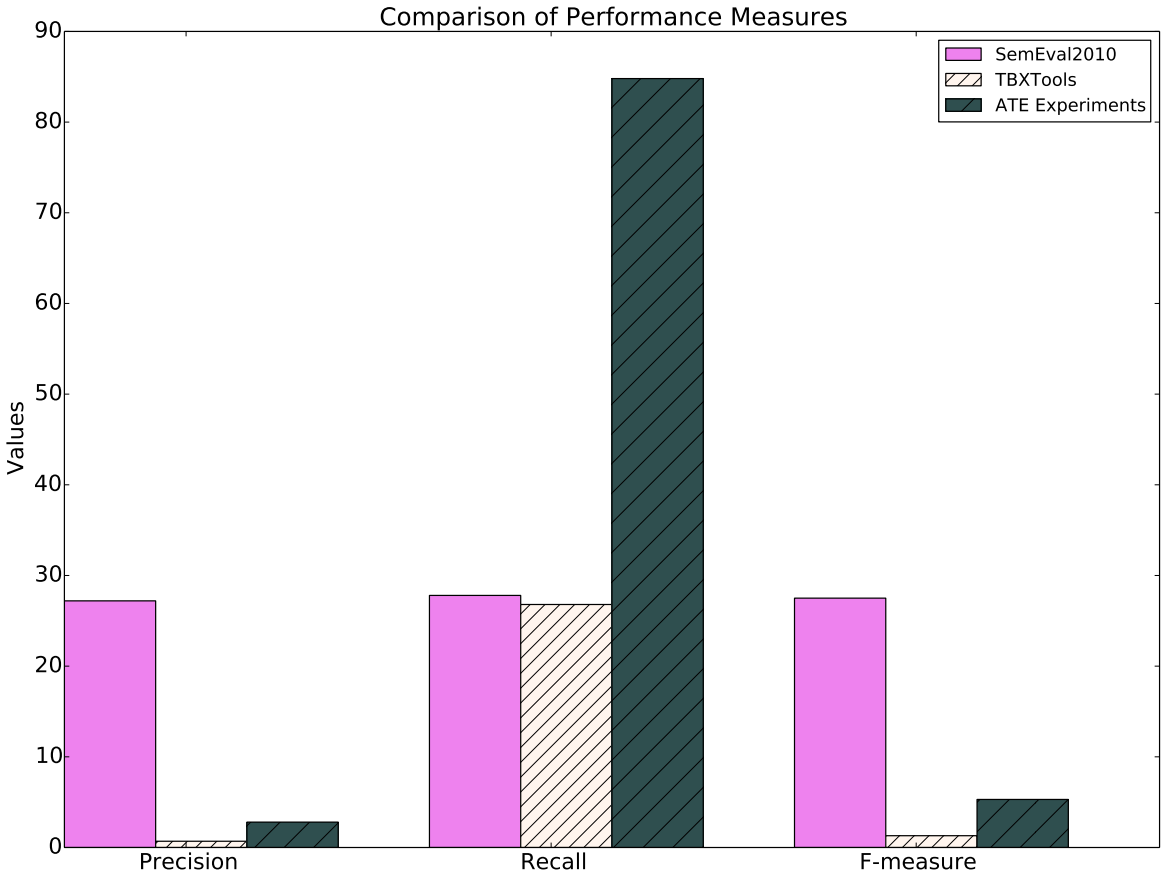

Figure 5.4: Performance Measures Comparison on Aircraft Manual Data

precision, while pruning the number of candidate terms improves precision while lowering recall.

The values for Precision, Recall and F-measure for the aircraft manual data are shown in Table 5.2. The values are presented in decimal form i.e. **0.8462 refers to 84.62%**

Precision and recall are improved by removing code snippets and pre-processing the data as illustrated in Table 5.3. However, it should be noted that by default the TBXTools system only produces word bigrams as a set of candidate terms, thus its results will overlook matches with longer index terms such as *"accessing array elements"*, leading to a drop in recall and precision when compared to a system that

Table 5.2: Experimental Results — Aircraft Manual Data Performance

Exp.#	Precision	Recall	F-measure
B1	0.2884	0.8462	0.4302
B2	0.0012	0.1250	0.0025
B3	0.0018	0.6250	0.0036
B3b	0.0025	0.8750	0.0050
B3c	0.0026	0.7500	0.0051
B3d	0.0013	0.1250	0.0026
B3e	0.0026	0.6250	0.0053
B3f	0.0036	0.6250	0.0072

accounts for these longer candidate terms.

Table 5.3: Experimental Results — Performance with TBXTools

Exp.#	Precision	Recall	F-measure
C1	0.0451	0.2737	0.0774
C2	0.0668	0.2905	0.1086
C3	0.0069	0.2675	0.0134

A summary of the observed performance is illustrated in Table 5.4. *Exp.* is the number of the particular experiment. The *Process* column describes some additional changes to each experiment. *WL* stands for the use of the WordNetLemmatizer. *CS* indicates the removal of code snippets. *S* shows that only a sample of the whole text was used. *D* denotes that duplicates were removed using a Python *set* command. *P* in the Process column shows that an expanded list of stopwords was used. *F* shows that words of low frequency were added to the stopwords list. The Experiment numbers are prefixed with "A", "B" or "C" with "A" showing data for the C Language document, "B" showing data for the aircraft maintenance guidelines manual, and "C" describing the values when the TBXTools software [OGVG15] was used on the C language document and on the aircraft manual data. "P", "R" and "F" in the table header stand for *Precision*, *Recall* and *F-Measure* respectively. Note that figures are presented in decimal format i.e. **0.8765 represents 87.65%**

For the overall experiment with C Standards data, the POS sequences used were: `<NN> <NN>` , `<VGB> <NNS>`, `<VGB> <NN> <NNS>`, `<N*> <N*.>?`, `<JJ> <NNS>` and `<NNS>`, which covered the most frequent sequences of POS Tags in the index.

For the overall experiment with aircraft manual data the POS sequences used were

Table 5.4: Experimental Results Summary
WL - WordNetLemmatizer, CS - Code Snippets removed, S - Sample text, D - Duplicates removed, P- Expanded Stopwords list, F - Low Frequency words, A - C Language data, B - Aircraft data, C - TBXTools

Exp	Process	Method	Matches	P	R	F
A1	WL	<NN> <NN>	76	0.0670	0.3028	0.1097
A2	WL	<N*> <N*.>?	83	0.0161	0.3307	0.0307
A3	WL	<VB[DGP] <N.*>?	18	0.0166	0.0717	0.0269
A4	WL	*Exp.# 2 and 3 together*	84	0.0135	0.3347	0.0259
A5	WL CS S	<N*> <N*.>?	18	0.0857	0.0994	0.0921
A6	WL CS	<N*> <N*.>?	62	0.0582	0.3425	0.0995
A7	D CS	<NN> <NN>	15	0.0426	0.7500	0.0806
A8	D CS	<NN> <NN>	16	0.0455	0.8000	0.0860
A9	D CS	<NN> <NNS>	16	0.0899	0.800	0.1616
A10	D CS	<VBG> <NNS>	3	0.1667	0.2500	0.2000
A11	D CS	<VBG> <NN> <NNS>	1	0.0769	0.0833	0.0800
A12	D CS P F	<NN> <NNS>	13	0.1204	0.6500	0.2031
A13	D CS P F	<NN> <NNS>	13	0.1831	0.6500	0.2857
B1	D CS P	*Acronyms*	77	0.2884	0.8462	0.4302
B2	D CS	<NN> <NN>	1	0.0012	0.1250	0.0025
B3	D CS	<NN> <NN>	5	0.0018	0.6250	0.0036
B3b	D CS	<NN> <NN>	7	0.0018	0.89	0.005
B3c	D CS	<NN> <NN>	6	0.0026	0.7500	0.0051
B3d	D CS	<NN> <NN>	1	0.0013	0.1250	0.0026
C1	P	*TBXTools statistical unfiltered*	49	0.0451	0.2737	0.0774
C2	CS P	*TBXTools statistical filtered*	52	0.0668	0.2905	0.1086
C3	CS P	*TBXTools statistical filtered*	42	0.0069	0.2675	0.0134

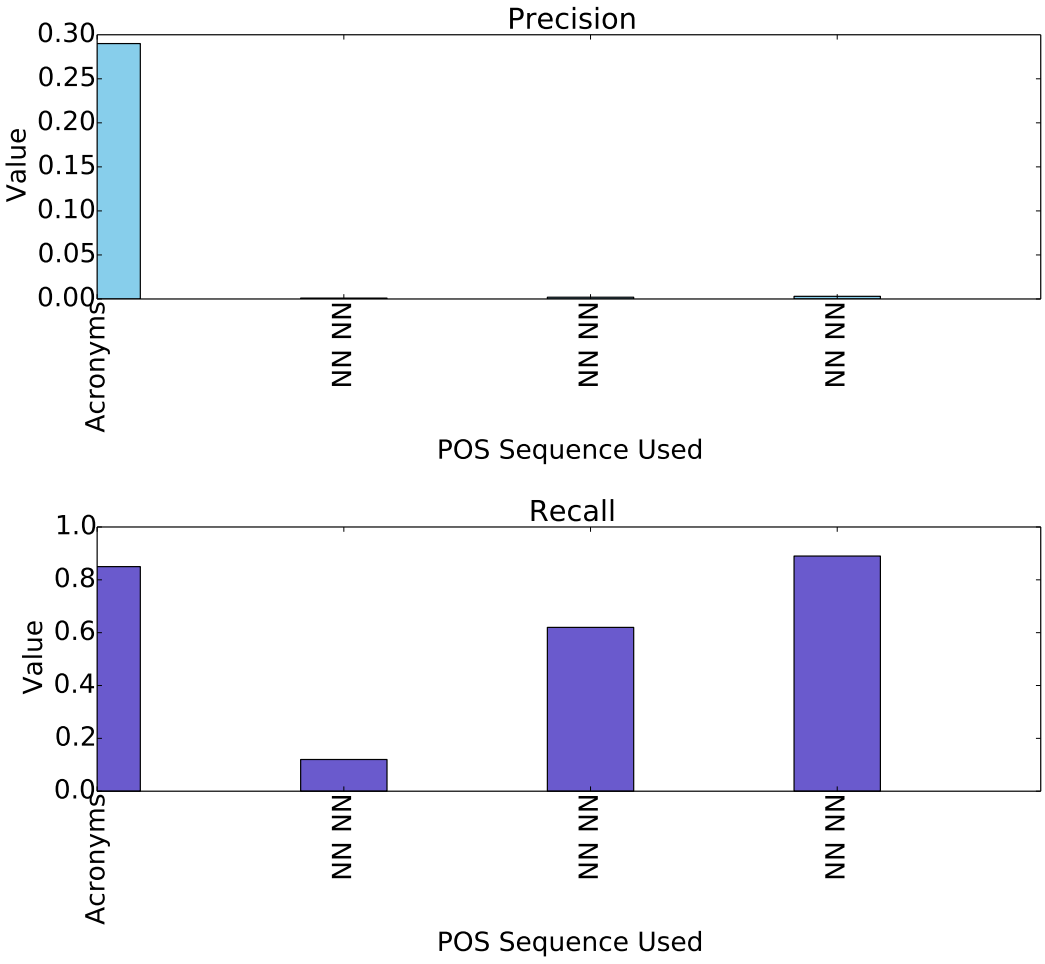

Figure 5.5: Performance Measures — Precision and Recall on Aircraft Manual Data

<NN> <NN> and *Acronyms* which covered the most frequent sequences of POS Tags or index term characteristics in the index.

The Odds Ratio test was used to compare our performance to that of TBXTools statistical.

$$Odds = \frac{P1/(1-P1)}{P2/(1-P2)} \tag{5.5}$$

where

P1 = Group 1 = Probability of a match between candidate term and index term using ATE

and

P2 = Group 2 = Probability of a match between candidate term and index term using TBXTools

Obtaining a value of $Odds > 1$ implies that a match is more likely to occur in Group 1 (ATE).

Table 5.5: Odds Ratio Test

Value	C data	Aircraft Manual Data
P1	0.36	0.85
P2	0.27	0.27
Odds	1.51	15.32

5.1 Evaluation Summary

The data in Table 5.6 are presented in decimal form i.e. **0.8485 represents 84.85%**

Table 5.6: Overall Experimental Results

Exp.#	Precision	Recall	F-measure
ATE C data overall	0.1969	0.3596	0.2545
TBXTools C data overall	0.0668	0.2905	0.1086
ATE Aircraft Manual data overall	0.0275	0.8485	0.0533
TBXTools Aircraft Manual data overall	0.0069	0.2675	0.0134

Hulth [Hul03] has observed that using "POS tag patterns gives a higher recall compared to extracting n-grams", and this is reflected in the difference in recall values when using our system as compared to TBXTools as can be seen in Table 5.3.

While examining the results obtained by our system we found that some candidate terms were in fact valid key terms that a human evaluator would consider suitable for inclusion in the index. However, the given documents' index did not contain these keywords. This means that our obtained precision values are lower than they should be.

As expected, nouns and noun phrases played a dominant role in candidate detection in the aircraft manual data set. Noun phrases also were a key factor in the C dataset, but they had to be balanced by accounting for verbs as well. This has to

do with the manner in which the indices of these documents are structured. For instance, the C data set contained terms such as *"accessing array elements"*, *"declaring pointers"* and *"defining structures"*, while the aircraft manual data set consisted of terms such as *"system element"* and *"airworthiness certificate"*.

As illustrated in the data of Table 5.6, ATE (our methodology) outperforms TBXTools (statistical) in terms of both recall and precision. ATE on the C Standards data produces a precision of alomst 20%, which is close to state of the art as measured by the SemEval 2010 exercise [KMKB10]. Recall and F-measure for the C data using ATE were better than the reported SemEval values. The difference in recall between ATE and TBXTools is much larger on the aircraft manual data set, where a recall rate of close to 85% was achieved by ATE. This serves to indicate that our results exceed those of the TBXTools system and also approach or exceed the state of the art as measured by SemEval values.

The following chapter presents our conclusions and directions for future research endeavors.

Chapter 6

Conclusion

In the preceding sections we have shown that Automatic Term Extraction (ATE) can be successfully performed, as indicated by the Odds Ratio test, in the technical domain to extract key phrases using Part-of-Speech information with additional information about commonality of some words. We have evaluated and demonstrated a method for the automatic creation of a list of the most relevant key terms from a C programming standards document and a manual of guidelines for aircraft maintenance. The results compare favorably to the state of the art, although the state of the art performance itself is still relatively low in terms of precision and recall.

There are some key points to note about our study. As described in the Discussion and Evaluation section, the default NLTK POS Tagger did not provide the required level of accuracy in tagging. For instance "string" was identified as a verb instead of a noun in the phrase "string constants". Indeed it is difficult to accurately tag words such as *bit*, *compound*, or *string* as either nouns or verbs without also obtaining further background knowledge of the domain of the document, as well as a more in-depth analysis of the sentence in which these words appear. For instance, the word *compound* might appear as a noun in an article whose subject matter is Chemistry, or in a Mathematical journal, or both as a noun and as a verb in an article devoted to the study of English Grammar. It has been noted in the literature that "POS disambiguation is one of the most important error sources" [CBP01] in ATE. However, it must be pointed out that the POS tags from the text were based on complete sentences, while the POS tags from the Index were based on index terms that appeared by themselves in the index terms file, one term per line. This may have altered the observed efficiency of the tagger.

While most indexers tend to focus largely on noun phrases [NM02, KU96], we have endeavored to include verb phrases as well, since it has been noted that index terms are more complex than mere sequences of nouns [KK09, PBB02].

Term extraction techniques are found to be highly domain- and genre- specific [MZHD03], which implies that a method that is used on one corpus may not necessarily produce equal levels of precision and recall when used on another.

In our calculation of performance measures, we used all the keyphrases that were generated, unlike experiments that placed a cutoff on the number of generated keyphrases. Using an exact match between gold standard terms and the resulting candidate terms has been recognized as quite a harsh method of evaluating the precision and recall of an ATE system. Despite this restriction we were able to achieve recall values of 62%, which improved to 89% when variants were allowed, which exceeds the values obtained by state of the art. Prior research was focused more on either using stemming to find matches, or on accepting variants of index terms.

We included variant terms in our candidate detection since, as Bougouin et al. state, "exact matches between extracted keyphrases and human assigned reference keyphrases ... leads to overly pessimistic scores since variations in the extracted key phrases that might be judged as correct cannot be taken into account" [BBR+16]. As noted by Kim et al. [KMKB10], when we "use a strict matching metric for evaluation,... semantically equivalent keyphrases are not counted as correct".

Some candidate terms found by our system were in fact valid key terms that a different human evaluator would have seen fit to be added to a generated index. However, they were not included as a part of the documents' original gold standard index. It has been noted by Drouin that "different terminologists will identify different terms in the same document and that the same phenomenon could be observed with one terminologist looking at the same corpus over a period of time" [Dro03]. This factor affects our evaluation measures by artificially lowering both our obtained recall and precision values and therefore also the F-measure.

It has been recognized that using "manually assigned keywords as the gold standard ... is the most severe way to evaluate a keyword extractor, as many terms might be just as good, although ... not chosen by the [original] human indexer" [Hul03]. It is also acknowledged by Witten et al. [WPF+99] that "it is highly unlikely that even another human would select the same set of key phrases as the original author". The issue of valid terms being found by the system but not being present in the set of gold standard terms was also recognized by Pazienza et al. [PPZ05] and by Pantel

et al. who noted that "experts often disagree on the correctness of a term list for a corpus" [PL01]. The recognition of key terms is therefore still a subjective task [dSCDFPR14].

6.1 Future Work

Future work will include the addition of external features e.g. data from Wikipedia article titles or other sources, in order to provide more domain-specific background information.

An alternate POS tagger will be used to find POS sequences, and the results will be compared with that of the default POS tagger provided by NLTK, in order to determine if this new tagger will increase the accuracy of POS tagging.

To extend the research further, keyword generation of key terms based on concepts in the document, but where the key terms are not explicitly contained in the document, could also be explored.

Bibliography

[AFT15] NA Astrakhantsev, Denis G Fedorenko, and D Yu Turdakov. Methods for Automatic Term Recognition in Domain-specific Text Collections: A Survey. *Programming and Computer Software*, 41(6):336–349, 2015.

[BBR+16] Adrien Bougouin, Sabine Barreaux, Laurent Romary, Florian Boudin, and Béatrice Daille. Termith-eval: a French Standard-based Resource for Keyphrase Extraction Evaluation. In *Language Resources and Evaluation Conference (LREC)*, 2016.

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, Inc., 2009.

[BMMI15] Slobodan Beliga, Ana Meštrović, and Sanda Martinčić-Ipšić. An Overview of Graph-based Keyword Extraction Methods and Approaches. *Journal of Information and Organizational Sciences*, 39(1):1–20, 2015.

[CBP01] M Teresa Cabré Castellví, Rosa Estopa Bagot, and Jordi Vivaldi Palatresi. Automatic Term Detection: A Review of Current Systems. *Recent Advances in Computational Terminology*, 2:53–88, 2001.

[CD16] Damien Cram and Béatrice Daille. Termsuite: Terminology Extraction with Term Variant Detection. *ACL 2016*, page 13, 2016.

[CGHH91] Kenneth Church, William Gale, Patrick Hanks, and Donald Hindle. Using Statistics in Lexical Analysis. *Lexical acquisition: Exploiting On-line Resources to Build a Lexicon*, 115:164, 1991.

[CW14] Erik Cambria and Bebo White. Jumping NLP Curves: a Review of Natural Language Processing Research [review article]. *IEEE Computational Intelligence Magazine*, 9(2):48–57, 2014.

[DGBPL00] Gaël Dias, Sylvie Guilloré, Jean-Claude Bassano, and José Gabriel Pereira Lopes. Combining Linguistics with Statistics for Multiword Term Extraction: A Fruitful Association? In *Content-Based Multimedia Information Access-Volume 2*, pages 1473–1491. Le Centre De Hautes Etudes Internationales D'informatique Documentaire, 2000.

[Dro03] Patrick Drouin. Term Extraction using Non-Technical Corpora as a Point of Leverage. *Terminology*, 9(1):99–115, 2003.

[DRS13] Anurag Dwarakanath, Roshni R Ramnani, and Shubhashis Sengupta. Automatic Extraction of Glossary Terms from Natural Language Requirements. In *Requirements Engineering Conference (RE), 2013 21st IEEE International*, pages 314–319. IEEE, 2013.

[DS04a] Lyne Da Sylva. A Document Browsing Tool based on Book Indexes. 2004.

[DS04b] Lyne Da Sylva. A Document Browsing Tool based on Book Indexes. 2004.

[DS09] Lyne Da Sylva. Corpus-based Derivation of a Basic Scientific Vocabulary for Indexing Purposes. *Journal of Linguistics*, 45(1):167–201, 2009.

[DS13] Lyne Da Sylva. Integrating Knowledge from Different Sources for Automatic Back-of-the-book Indexing. In *Proceedings of the Annual Conference of CAIS/Actes du congrès annuel de l'ACSI*, 2013.

[dSCDFPR14] Merley da Silva Conrado, Ariani Di Felippo, Thiago Alexandre Salgueiro Pardo, and Solange Oliveira Rezende. A Survey of Automatic Term Extraction for Brazilian Portuguese. *Journal of the Brazilian Computer Society*, 20(1):12, 2014.

[DSD05] Lyne Da Sylva and Frédéric Doll. A Document Browsing Tool: using Lexical Classes to Convey Information. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 307–318. Springer, 2005.

[Fag89] Joel L Fagan. The Effectiveness of a Nonsyntactic Approach to Automatic Phrase Indexing for Document Retrieval. *Journal of the American Society for Information Science*, 40(2):115, 1989.

[FAM00] Katerina Frantzi, Sophia Ananiadou, and Hideki Mima. Automatic Recognition of Multi-word Terms:. the C-value/NC-value Method. *International Journal on Digital Libraries*, 3(2):115–130, 2000.

[FdSG14] Alessio Ferrari, Felice dell'Orletta, Giorgio Oronzo Spagnolo, and Stefania Gnesi. Measuring and Improving the Completeness of Natural Language Requirements. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 23–38. Springer, 2014.

[FPW+99] Eibe Frank, Gordon W Paynter, Ian H Witten, Carl Gutwin, and Craig G Nevill-Manning. Domain-Specific Keyphrase Extraction. In *16th International Joint Conference on Artificial Intelligence (IJCAI 99)*, volume 2, pages 668–673. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[Gan01] Aryya Gangopadhyay. Conceptual Modeling from Natural Language Functional Specifications. *Artificial Intelligence in Engineering*, 15(2):207–218, 2001.

[GEL+14] Shalini Ghosh, Daniel Elenius, Wenchao Li, Patrick Lincoln, Natarajan Shankar, and Wilfried Steiner. Automatically Extracting Requirements Specifications from Natural Language. *CoRR, abs/1403.3142*, 2014.

[Gro11] International Standardization Working Group. *C11 Standards WG14 N1570*. GNU, 2011.

[Hea97] Marti A Hearst. Texttiling: Segmenting Text into Multi-paragraph Subtopic Passages. *Computational Linguistics*, 23(1):33–64, 1997.

[Hig90] Lindley R. Higgins. *Maintenance Engineering Handbook*. McGraw-Hill, 1990.

[HM15] Julia Hirschberg and Christopher D Manning. Advances in Natural Language Processing. *Science*, 349(6245):261–266, 2015.

[HN14] Kazi Saidul Hasan and Vincent Ng. Automatic Keyphrase Extraction: A Survey of the State of the Art. In *ACL (1)*, pages 1262–1273, 2014.

[Hos06] Mark Hoske. Functional Specifications. In *Control Engineering*, pages 69–73. Peer Reviewed Journal, 2006.

[Hul03] Anette Hulth. Improved Automatic Keyword Extraction given more Linguistic Knowledge. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 216–223. Association for Computational Linguistics, 2003.

[JCK05] Sittichai Jiampojamarn, Nick Cercone, and Vlado Keselj. Biological Named Entity Recognition using N-grams and Classification Methods. In *Proceedings of the Conference Pacific Association for Computational Linguistics, PACLING'05*, Meisei University, Hino Campus, Hino-shi, Tokyo, 191-8506 Japan, August 2005.

[Jia05] Sittichai Jiampojamarn. Automatic Biological Term Annotation using N-gram and Classification Models. Master's thesis, Faculty of Computer Science, Dalhousie University, 2005.

[JK95] John S Justeson and Slava M Katz. Technical Terminology: Some Linguistic Properties and an Algorithm for Identification in Text. *Natural Language Engineering*, 1(01):9–27, 1995.

[JM09] Daniel Jurafsky and James H Martin. *Speech and Language Processing an Introduction to Natural Language Processing, Computational Linguistics, and Speech 2nd ed.* Pearson Education, 2009.

[KK09] Su Nam Kim and Min-Yen Kan. Re-examining Automatic Keyphrase Extraction Approaches in Scientific Articles. In *Proceedings of the Workshop on Multiword Expressions: Identification, Interpretation, Disambiguation and Applications*, pages 9–16. Association for Computational Linguistics, 2009.

[KMKB10] Su Nam Kim, Olena Medelyan, Min-Yen Kan, and Timothy Baldwin. Semeval-2010 Task 5: Automatic Keyphrase Extraction from Scientific Articles. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 21–26. Association for Computational Linguistics, 2010.

[KMKB13] Su Nam Kim, Olena Medelyan, Min-Yen Kan, and Timothy Baldwin. Automatic Keyphrase Extraction from Scientific Articles. *Language Resources and Evaluation*, 47(3):723–742, 2013.

[KU96] Kyo Kageura and Bin Umino. Methods of Automatic Term Recognition: A Review. *Terminology. International Journal of Theoretical and Applied Issues in Specialized Communication*, 3(2):259–289, 1996.

[LFV16] Lucelene Lopes, Paulo Fernandes, and Renata Vieira. Estimating Term Domain Relevance through Term Frequency, Disjoint Corpora Frequency-Tf-Dcf. *Knowledge-Based Systems*, 97:237–249, 2016.

[LL08] Marina Litvak and Mark Last. Graph-based Keyword Extraction for Single-document Summarization. In *Proceedings of the Workshop on Multi-source Multilingual Information Extraction and Summarization*, pages 17–24. Association for Computational Linguistics, 2008.

[LSM+89] Sandra Loosemore, Richard M. Stallman, Roland McGrath, Andrew Oram, and Ulrich Drepper. *The GNU C Library Reference Manual*. GNU, 1989.

[LVJRT16] Juan Antonio Lossio-Ventura, Clement Jonquet, Mathieu Roche, and Maguelonne Teisseire. Biomedical Term Extraction: Overview and a New Methodology. *Information Retrieval Journal*, 19(1-2):59–99, 2016.

[MFO16] Zakariae Alami Merrouni, Bouchra Frikh, and Brahim Ouhbi. Automatic Keyphrase Extraction: An Overview of the State of the Art. In *Information Science and Technology (CiSt), 2016 4th IEEE International Colloquium on*, pages 306–313. IEEE, 2016.

[MI04] Yutaka Matsuo and Mitsuru Ishizuka. Keyword Extraction from a Single Document using Word Co-occurrence Statistical Information. *International Journal on Artificial Intelligence Tools*, 13(01):157–169, 2004.

[MMC14] Stephen G MacDonell, Kyongho Min, and Andy M Connor. Autonomous Requirements Specification Processing using Natural Language Processing. *arXiv preprint arXiv:1407.6099*, 2014.

[MS+99] Christopher D Manning, Hinrich Schütze, et al. *Foundations of Statistical Natural Language Processing*, volume 999. MIT Press, 1999.

[MZ10] Indrit Myderrizi and Ali Zeki. Current-steering Digital-to-Analog Converters: Functional Specifications, Design Basics, and Behavioral Modeling. *IEEE Antennas and Propagation Magazine*, 52(4):197–208, 2010.

[MZHD03] E Milios, Y Zhang, B He, and L Dong. Automatic Term Extraction and Document Similarity in Special Text Corpora. In *Proceedings of the Sixth Conference of the Pacific Association for Computational Linguistics*, pages 275–284. Citeseer, 2003.

[NKLB12] David Newman, Nagendra Koilada, Jey Han Lau, and Timothy Baldwin. Bayesian Text Segmentation for Index Term Identification and Keyphrase Extraction. In *COLING*, pages 2077–2092, 2012.

[NM02] Hiroshi Nakagawa and Tatsunori Mori. A Simple but Powerful Automatic Term Extraction Method. In *COLING-02 on COMPUTERM 2002: Second International Workshop on Computational Terminology-Volume 14*, pages 1–7. Association for Computational Linguistics, 2002.

[OGVG15] Antonio Oliver González and Mercè Vàzquez Garcia. Tbxtools: A Free Fast and Flexible Tool for Automatic Terminology Extraction. Association for Computational Linguistics (ACL), 2015.

[PBB02] Youngja Park, Roy J Byrd, and Branimir K Boguraev. Automatic Glossary Extraction: beyond Terminology Identification. In *Proceedings of the 19th International Conference on Computational Linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002.

[PL01] Patrick Pantel and Dekang Lin. A Statistical Corpus-based Term Extractor. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 36–46. Springer, 2001.

[PPMM15] Carlos Periñán-Pascual and Eva M Mestre-Mestre. DEXTER: Automatic Extraction of Domain-specific Glossaries for Language Teaching. *Procedia-Social and Behavioral Sciences*, 198:377–385, 2015.

[PPZ05] Maria Teresa Pazienza, Marco Pennacchiotti, and Fabio Massimo Zanzotto. Terminology Extraction: an Analysis of Linguistic and Statistical Approaches. In *Knowledge Mining*, pages 255–279. Springer, 2005.

[Rai48] Victor C Raimy. Functional Specifications for a Sound Recorder for the psychological Clinic. *American Psychologist*, 3(11):513, 1948.

[RY07] Trevis Rothwell and James Youngman. *C99 Standards WG14 N1256*. GNU, 2007.

[SS15] Sifatullah Siddiqi and Aditi Sharan. Keyword and Keyphrase Extraction Techniques: A Literature Review. *International Journal of Computer Applications*, 109(2), 2015.

[TKSDM03] Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 Shared task: Language-independent Named Entity Recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics, 2003.

[Tur00] Peter D Turney. Learning Algorithms for Keyphrase Extraction. *Information Retrieval*, 2(4):303–336, 2000.

[Wid07] Henry Widdowson. Jr firth, 1957, Papers in Linguistics 1934–51. *International Journal of Applied Linguistics*, 17(3):402–413, 2007.

[WPF+99] Ian H Witten, Gordon W Paynter, Eibe Frank, Carl Gutwin, and Craig G Nevill-Manning. KEA: Practical Automatic Keyphrase Extraction. In *Proceedings of the fourth ACM conference on Digital libraries*, pages 254–255. ACM, 1999.

Appendix A

Sample Code

POS Tagging:

```
import nltk
import re
from collections import Counter

import numpy as np
import matplotlib.pyplot as plt
from operator import itemgetter

#POS Tags from a small test sample of the C programmng language standards

text = """A string constant is a sequence of zero or more characters,
digits, and escape sequences enclosed within double quotation marks. A string constant is
of type ‘‘array of characters’’. All string constants contain a null termination character
(\0) as their last character. Strings are stored as arrays of characters, with no inherent
size attribute. The null termination character lets string-processing functions know
where the string ends.

Adjacent string constants are concatenated (combined) into one string, with the null
termination character added to the end of the final concatenated string.

A string cannot contain double quotation marks, as double quotation marks are used
to enclose the string. To include the double quotation mark character in a string,
```

```
23 use the \" escape sequence. You can use any of the escape sequences that
       can be used
24 as character constants in strings. Here are some example of string
      constants:"""
25
26 #convert text to lowercase
27 text = text.lower()
28
29 #remove non-alphanumeric characters
30 text = re.sub('[^0-9a-zA-Z]+',' ',text)
31 text = re.sub('[\s*\d+\s+]',' ', text)
32
33 #tokenize
34 text = nltk.word_tokenize(text)
35
36
37
38 #apply POS tags
39 tagged = nltk.pos_tag(text)
40
41 print tagged
42
43
44 with open('output_test_tags1.txt', 'w') as f:
45     for line in tagged:
46         strs=" ".join(str(x) for x in line)
47         f.write(strs+"\n")
48
49 counts = Counter(tag for word,tag in tagged)
50 print counts.most_common()
51
52 countswords = Counter(word for word,tag in tagged)
53 print countswords.most_common()
54
55
56 plt.figure()
57 counts = counts.items()
```

```
58 counts.sort(key=itemgetter(1), reverse=True)
59 labels, values = zip(*counts)
60 indexes = np.arange(len(labels))
61 width = 1
62
63 plt.xticks(indexes, labels)
64 plt.title("POS Tag Frequencies in Sample Text",fontsize=20)
65 plt.xlabel("POS Tag ",fontsize=20)
66 plt.ylabel("Cumulative Frequency",fontsize=20)
67 plt.tick_params(axis='x', labelsize=18)
68 plt.tick_params(axis='y', labelsize=18)
69 cumulative = np.cumsum(values)
70 plt.plot(cumulative, c='green')
71
72 #plt.show()
73
74 plt.figure()
75 countswords = countswords.items()
76 countswords.sort(key=itemgetter(1), reverse=True)
77 labels, values = zip(*countswords)
78 indexes = np.arange(len(labels))
79 width = 1
80
81 plt.xticks(indexes, labels)
82 plt.title("Word Frequencies in Sample Text",fontsize=20)
83 plt.xlabel("Word",fontsize=20)
84 plt.ylabel("Cumulative Frequency",fontsize=20)
85 plt.xticks(rotation=90)
86 cumulative1 = np.cumsum(values)
87 plt.tight_layout()
88 plt.plot(cumulative1, c='green')
89 plt.tick_params(axis='x', labelsize=18)
90 plt.tick_params(axis='y', labelsize=18)
91 plt.show()
```

Listing A.1: POS tagging with small sample text - premilinary trials

The above code produces the following output:

```
a DT
string NN
constant NN
is VBZ
a DT
sequence NN
of IN
zero NN
or CC
more JJR
characters NNS
digits NNS
and CC
escape NN
sequences NNS
enclosed VBD
within IN
double JJ
quotation NN
marks NNS
a DT
string VBG
constant NN
is VBZ
of IN
type NN
array NN
of IN
characters NNS
all DT
string VBG
```

```
constants NNS
contain VBP
a DT
null NN
termination NN
character NN
as IN
their PRP$
last JJ
character NN
strings NNS
are VBP
stored VBN
as IN
arrays NNS
of IN
characters NNS
with IN
no DT
inherent JJ
size NN
attribute NN
the DT
null NN
termination NN
character NN
lets NNS
string VBG
processing NN
functions NNS
know VBP
```

```
where WRB
the DT
string NN
ends VBZ
adjacent NN
string VBG
constants NNS
are VBP
concatenated VBN
combined VBN
into IN
one CD
string VBG
with IN
the DT
null JJ
termination NN
character NN
added VBD
to TO
the DT
end NN
of IN
the DT
final JJ
concatenated VBN
string VBG
a DT
string NN
can MD
not RB
```

```
contain VB
double JJ
quotation NN
marks NNS
as IN
double JJ
quotation NN
marks NNS
are VBP
used VBN
to TO
enclose VB
the DT
string NN
to TO
include VB
the DT
double JJ
quotation NN
mark NN
character NN
in IN
a DT
string NN
use NN
the DT
escape NN
sequence NN
you PRP
can MD
use VB
```

```
any DT
of IN
the DT
escape NN
sequences NNS
that WDT
can MD
be VB
used VBN
as IN
character NN
constants NNS
in IN
strings NNS
here RB
are VBP
some DT
example NN
of IN
string NN
constants NNS
```

POS Tagging of C89 text:

```
import nltk
import re
from collections import Counter

import numpy as np
import matplotlib.pyplot as plt
from operator import itemgetter

#POS Tags from text file of C programmng language standards

# Open data file
with open('C89fromHTMLnoindex.txt', 'r') as myfile:
    text=myfile.read().replace('\n', ' ')

# Close opened file
myfile.close()

#convert text to lowercase
text = text.lower()

#remove non-alphanumeric characters and individual numbers
#as all index terms consist only of letters
text = re.sub('[^0-9a-zA-Z]+',' ',text)
text = re.sub('[\s*\d+\s+]',' ', text)

#tokenize
text = nltk.word_tokenize(text)

#remove stopwords
stopwords = nltk.corpus.stopwords.words('english')
content = [w for w in text if w.lower() not in stopwords]


#apply POS tags
tagged = nltk.pos_tag(content)

```

```
#print tagged

with open('output_test_tags3.txt', 'w') as f:
    for line in tagged:
        strs=" ".join(str(x) for x in line)
        f.write(strs+"\n")

f.close()
print "Completed POS tagging\n"

counts = Counter(tag for word,tag in tagged)

print counts.most_common()


counts = counts.items()
counts.sort(key=itemgetter(1), reverse=True)
labels, values = zip(*counts)
indexes = np.arange(len(labels))
width = 1

#plt.bar(indexes, values, width, align='center',color='skyblue')
plt.bar(indexes, values, width,color='skyblue',align='center')
plt.xticks(indexes, labels)
plt.title("POS Tag Frequencies in Document",fontsize=25)
plt.xlabel("POS Tag", fontsize=22)
plt.ylabel("Frequency", fontsize=22)
plt.tick_params(axis='x', labelsize=22)
plt.tick_params(axis='y', labelsize=22)
plt.xticks(rotation=90)
plt.tight_layout()
#cumulative = np.cumsum(values)
#plt.plot(cumulative, c='green')
plt.axis('tight')
plt.margins(0.01, 0.01)
```

```
plt.show()
```

Listing A.2: POS tagging with C89 text - Ths code illustrates the pre-processing method

```
import re

indexdata=""

# Open data file
with open('C89fromHTML_index.txt', 'r') as myfile:
  for line in myfile:
    sep = ':'
    indexdata+= line.split(sep, 1)[0]
    indexdata+="\n"

with open('output_index1.txt', 'w') as f:
  f.write(indexdata+"\n")

# Close opened files
myfile.close()
f.close()

print "Completed index filtering\n"
```

Listing A.3: Index filtering

```
import re
import nltk
import pprint

# Open data file
with open('C89fromHTMLnoindex.txt', 'r') as myfile:
    text=myfile.read().replace('\n', ' ')

# Close opened file
myfile.close()

#convert text to lowercase
```

```
13 text = text.lower()
14
15 #remove non-alphanumeric characters and individual numbers
16 #as all index terms consist only of letters
17 text = re.sub('[^0-9a-zA-Z]+',' ',text)
18 text = re.sub('[\s*\d+\s+]',' ', text)
19
20 sentences = nltk.sent_tokenize(text)
21 sentences = [nltk.word_tokenize(sent) for sent in sentences]
22 sentences = [nltk.pos_tag(sent) for sent in sentences]
23
24 #print sentences
25
26 with open('output_test_tags5.txt', 'w') as f:
27   for line in sentences[0]:
28     strs=" ".join(str(x) for x in line)
29     f.write(strs+"\n")
30
31 f.close()
32 print "Completed POS tagging\n"
```

Listing A.4: POS Tagging of Sentences in preparation for Chunking

Appendix B

Data Sets

The following section shows some more detailed information about the data sets that were used in the experiments.

Table B.1: Frequency of sequences of POS Tags in the Aircraft Maintenance Guidelines manual

POS Tag	Frequency
NNP NNP	20
NNP NN NN	11
NN	9
NN NN	8
NNP	7
NNP NN	7
NN NN NN	7
NN NNP	6
JJ NN	6
NNP NNP NNP NNP	3
NNP NNS NN	3
NNP NNP NNP	3
NNP NNP IN NNP	3
JJ NN TO VB	2
NNP NNS	2
NN CC NN	2
NNP VBP NN	2
NNP VBD NN	2
NN VBD	2
VBG	2
NNP NN JJ	2
JJ NN NNS VBP	1
JJ NN NNP NNP	1
NN IN NN	1
JJ NN NNS IN JJ NN	1
NNP NNP IN NNP CC NNPS	1
NNP IN NN	1
IN NN	1
VBN NNP	1
RB NNP	1
JJ NN NN	1
NN VBG NNS NN	1
NN VBG CC NN	1
VBG NN	1
JJ VBN NNP	1

POS Tag	Frequency
NNP NN NN NN	1
NNS	1
VBG NN CC NN NN	1
VBG NNP NNP NNP	1
NNP NNP NNP IN NNP	1
NNP CC NN	1
NNP NNP NNP NNP IN DT NNP	1
JJ NNP	1
NNS VBP	1
NNP IN NNP NNPS	1
JJ	1
NNP NNP NNPS	1
NN NNS	1
NNP CC NN NN	1
NN NN CC NN NN	1
RB JJ NN	1
DT NNP	1
JJ NN IN VBN NNS	1
NNS VBG	1
JJ NN NN NN	1
NN NNP NNP	1
DT NN VBD	1
NN VBG NN	1
NN VBG	1
NNS NNS	1
NN JJ NN	1
NNP NNP CC NNP NNP	1
NN NN NNP NNP	1
NNP VBP	1
NN VBD NN	1
JJ NN CC NN	1
VBG NNP	1
NNP NNS IN NN	1
NNP IN JJ NNS	1

Table B.2: Partial list of Custom Stop Words

Term	Term	Term	Term	Term
occurs	constrain	truth	foo	appendix
discards	wont	ie	thus	uses
redirects	easier	termination	either	support
custom	common	making	body	modifies
outermost	natural	everyone	built	token
lets	crash	safe	slight	maintainer
enum	performs	continues	doubt	dereferencing
derived	basics	fish	machine	structures
dollar	sizet	library	equals	referred
project	consist	anywhere	prepend	whatever
addresses	earlier	deduce	enclose	recommend
month	third	adjacent	step	squares
wraps	returned	nonstandard	containing	invalid
looks	underscore	year	exist	done
consists	mark	duration	method	right
line	furthermore	decrementing	depends	difference
issues	year	declarator	purpose	similar
indirect	chapter	nonstandard	whether	output
impact	wont	therefore	assumptions	prefix
special	produced	handler	provide	nowadays
group	executing	precede	bar	braces
innermost	whatever	subsequent	starts	serves
available	bar	ensures	likewise	concatenated
represents	sample	date	counter	silent
produces	rectangle	digit	go	source
methods	causes	handler	check	addition
day	apply	posix	error	evaluate
examples	actions	use	subscript	lets
gives	yields	month	added	mark
seen	nine	extract	plus	handles

Table B.3: Frequency of **Unedited** sequences of POS Tags in the C Standards Index

POS Tag	Frequency
NN NN	19
NNS	17
NNP	16
NN NNS	15
NNS comma NN	15
VBG NNS	12
VBG NN NNS	11
NNS comma JJ	10
NNS comma VBG	10
NN	9
JJ NNS	8
NN NNS comma VBG	7
IN NN	5
NNS NNS comma NN	5
NN comma NN	5
VBG NN NNS IN NN	4
NN NN NN NN	4
NN NNS comma VBG IN NN	4
VBD JJR NN	3
NN comma VBD NN	3
NN VBD	3
NNS NNS comma JJ NN	3
NNS comma JJ NN	3
NNS IN NNS	3
NNS comma NNS IN	3
NN NN NNS	3
NN JJ	3
VBD NNS	2
NN IN NNS	2
NNS comma VBD	2
NN comma JJ	2
NNS VBP NNS	2
NNS comma VBP IN	2
NN NNS NN	2
space	2
VBG NNS comma VBG	2
VBD NN NNS NN	2
NNS comma VBG NN	2
JJ NN	2
JJ NN NNS	2
JJ	2
NNS TO NNS	2
NNS comma NNS TO	2
DT	1
NNS comma NNS IN NNS	1
VBD JJ NN NNS NN	1
NNS NNS comma JJ	1
RB VBZ	1
NNS comma NN NN	1
VBD PRP$	1
NNS NNS comma VBP	1
RB JJ NNS NN	1
NN VBG	1
NNS IN JJ NNS	1
JJ NNS NN	1
JJ NNS NNS	1
VBD RB JJ NNS NN	1
NN NNP NNP	1
NNS NNS comma VBG NN	1
PRP	1
NNS NNS	1
NN NNS comma NN NN	1
NN JJ NNS	1
VBD VBN NNS NN	1
JJ NN NN NN	1
VBG	1
NN VBZ comma IN NNS	1
RB RB JJ NNS NN	1
NNS VBG NNS	1
NNS comma VBP	1
NN NN IN JJ	1
VBP NN	1
LS	1
JJ NN NN	1
IN NNS NN	1
NN NNS comma VBG IN	1
JJ NN NNS NN	1
JJ NNS comma VBP	1
VBD RB RB JJ NNS NN	1
NN NN NNS comma JJ NN	1
IN NN NN	1
RB VBP NNS NN	1
IN NNS	1
JJ NN NN NNS	1

Table B.4: NLTK Stop Words List

Term	Term	Term	Term	Term	Term
i	them	does	before	both	d
me	their	did	after	each	ll
my	theirs	doing	above	few	m
myself	themselves	a	below	more	o
we	what	an	to	most	re
our	which	the	from	other	ve
ours	who	and	up	some	y
ourselves	whom	but	down	such	ain
you	this	if	in	no	aren
your	that	or	out	nor	couldn
yours	these	because	on	not	didn
yourself	those	as	off	only	doesn
yourselves	am	until	over	own	hadn
he	is	while	under	same	hasn
him	are	of	again	so	haven
his	was	at	further	than	isn
himself	were	by	then	too	ma
she	be	for	once	very	mightn
her	been	with	here	s	mustn
hers	being	about	there	t	needn
herself	have	against	when	can	shan
it	has	between	where	will	shouldn
its	had	into	why	just	wasn
itself	having	through	how	don	
they	do	during	all	should	
weren	won	wouldn	any	now	

Appendix C

Algorithms

A list of algorithms that were used is found on the next page.

List of Algorithms