6-6-2014 12:00 AM

Cosine Similarity for Article Section Classification: Using Structured Abstracts as a Proxy for an Annotated Corpus

Arthur T. Bugorski, *The University of Western Ontario*

Supervisor: Dr Robert Mercer, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

COSINE SIMILARITY FOR ARTICLE SECTION CLASSIFICATION: USING STRUCTURED ABSTRACTS AS A PROXY FOR AN ANNOTATED CORPUS

(Thesis format: Monograph)

by

Arthur Bugorski

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

During the last decade, the amount of research published in biomedical journals has grown significantly and at an accelerating rate. To fully explore all of this literature, new tools and techniques are needed for both information retrieval and processing. One such tool is the identification and extraction of key claims.

In an effort to work toward claim-extraction, we aim to identify the key areas in the body of the article referred to by text in the abstract. In this project, our work is preliminary to that goal in that we attempt to match specific clauses in the abstract with the section of the article body to which they refer. For our data, we use journal articles from PubMed with structured abstracts.

Our technique is based on the cosine-measure of feature vectors using a bag-of-words approach. We refine our technique through the application of five different experimental variables: feature-weighting, word and bi-gram based feature-sets, text pre-processing, fixed-expression filtering, and different classifier heuristics.

We found that the choice of classifier dominates all other considerations, and while their performance with feature-weighting is synergistic, other variables were found to have little or no effect.

Keywords: Cosine Measure, Text Similarity, Structured Abstracts, Claim Extraction

Acknowlegements

I would like to express my appreciation to my supervisor Professor Dr. Mercer upon whose experience I leaned on heavily.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The project began following an extended discussion with Professor Pete Rogan from the Department of Biochemistry at The University of Western Ontario and President of Cytognomix Inc. Cytognomix sells DNA probes for the purpose of confirming the presence, or absence of a gene, corresponding to the given probe in a sample. These probes can be used by cytogeneticists and genetic counsellors for diagnosing congenital conditions. Often, a genetic counsellor will take note of a patient's phenotypes, postulate an underlying genetic condition, and then order probes to test an acquired genetic sample for the condition.

Professor Rogan was interested in a system that could mine a biomedical corpus and match phenotypes to conditions and these conditions to specific genetic mutations. He also desired that such a system would be able to handle contradictory claims by assessing the relative merit of each claim using context: repetition of the claim within the corpus, agreement with the rest of the corpus, citations counts, the impact factor of the journal, and the publishing history of the authors. It was hoped that this system would allow genetic counsellors to input the patient's phenotypes and the system would, upon additional consideration of the incidence rate of any relevant conditions, determine in which order probes should be tested to optimize reaching a diagnosis.

While it was clear that such a system was beyond the means of our project, we attempt to solve one of the components necessary for such as system: focusing on identifying and

extracting the main claims of biomedical research articles. Dr Barbara White completed her PhD at The University of Western Ontario on inter-annotator agreement of human annotators coding claims in biomedical journal publications [1]. She had been hoping to receive a grant to further her work by enlarging her corpus, and we had been hoping to use her corpus both as our own and as a gold standard for our work on claim extraction. Unfortunately, she was not able to continue her work on her corpus, and we judged its current state insufficient for our needs.

Furthermore, we hoped that we could compensate for the lack of a sufficiently large annotated corpus by using what we called the "greatest hits" model of abstract writing. It was our belief, at the time, that the sentences in the abstract were sourced from the most important sentences in the journal article's argumentation. By matching each sentence from the abstract with a sentence in the body of the article, we would then uncover loci of the most important argumentation in the article and be able to extract the main claims from them. Additionally, we believed that our biggest challenge would be accounting for changes in tense, pronoun replacement, etcetera that had been made for reasons of grammar and conciseness to the sentence as it had been copied from the body to abstract.

This approach, however, was discouraged when we met with a subject matter expert, Dr Derek McLaughlin, who instructs undergraduate students in the biomedical field in professional writing for the sciences. He informed us that the lifting of entire sentences from the article body was considered a bad practice, and, if anything, we should look for similarities at the clause level rather than the sentence level, due to the compressed nature of an abstract.

Our discussion with Dr McLaughlin left us wondering how to establish a corpus. We then looked into the MEDLINE collection of journals, since they often have a structured abstract. A structured abstract is one where the abstract is organized into the same sections as the sections which appear in the body of the article. We realized that these journal articles with their structured abstracts could serve as an annotated corpus. MEDLINE also had the advantage of being online and being stored in an XML format.

Upon revisiting our earlier goal of matching sentences from the abstract to those of the

article body for the purpose of the establishing loci of argumentation ripe for extracting, we came to see it as a problem that could be approached in three separate stages. The first is selecting the corresponding section for each sentence in the abstract; the second is sub-selecting from that section a specific sentence to match the one from the abstract; and third is mining the matched sentence's loci for claims. By using the structured abstracts as an annotated corpus, we felt we could meaningfully investigate the first stage of the problem. However, following Dr McLaughlin's advice, we decided to focus on matching clauses from the abstract instead of sentences.

1.1 Research Question

Since our approach is novel to the field, our work, by definition, had to be exploratory; thus, we have no pre-established performance goals or results to which to directly compare our results. Our research is driven by the following question: how can we maximize the performance of using word similarity to label clauses in abstracts to their referent article body sections? We implemented several different classifiers that use the cosine measure of similarity and applied various techniques to them that have been used in other natural language processing or classification tasks attempting to improve the heuristics' performance.

1.2 Approach

Our approach was to try out various techniques and to be experimental and exploratory. In our search of the literature, we did not find any related work which could serve as a benchmark for our own work. Thus, we felt, for this approach to be fruitful in the long term, it would be up to us to establish a baseline for further work by others to improve upon. To this end, we applied various techniques that have been known to be efficacious in both the fields of artificial intelligence and natural language processing research, and we measured their effect on our ability to classify clauses in the abstract according to their referent sections of the article. Our

underlying and unquestioned assumption was that anything that improved our classifiers would ultimately improve the ability to assign the clauses from the abstract to specific sections of text.

1.3 Organization

This thesis, like our corpus, is organized following the conventional IMRaD structure: an Introduction, followed by a Literature Review, and then Methods, Results, and Discussion sections. The Introduction consists of the backstory that led us from the original inspiration for our research to the final research question upon which we settled, then a clear statement of our final research question.

The Literature Review examines the published work that most closely resembles our own. The work is generalized along the lines of the corpora other researchers used, their methodologies and the features they used for their own classification. Following that is an examination of how our work is both an extension of the field and a novel outgrowth of it.

The Methods chapter starts with a discussion of the composition of our corpus along with statistics concerning word frequency. Then, we define and describe our motivation and procedures for our five experimental variables: feature weighting, feature-sets, text pre-processing, and classification heuristics.

The Results chapter begins with an overview of the section and then reviews the results of the experimental variables in the order in which they are presented in the Methods section. Each method is evaluated fully. For all but the final experimental variable (the classifiers), each variable is explored full, but only select configurations are carried forward to be used as the basis for the exploration of the next experimental variable. However, for the classifiers, there is a full review of how each classifier performs with each other experimental variable.

Finally, this thesis ends with a Conclusion chapter. This section states the conclusions that can be drawn from our work. It also suggests steps to both further our work and how to apply the results of our work to other related work.

Chapter 2

Literature Review

2.1 Corpus Construction

There has been a tremendous growth occurring in the amount of research being performed in the biomedical disciplines, and this increase has lead to a large volume of high-quality research articles being published in journals. As happens with such an influx of material, diffusing this information has become an issue. For this reason, research articles from biomedical journals often form the corpora of natural language processing research [2-14]. Specifically, in the literature, there is a focus on randomized control trials [7,12,15,16]. Fittingly, the other major discipline whose research is found in corpora is computational linguistics. This is most likely because the researchers are better able to assess the material with which they working, especially when manually annotating it [17]. This is important because, as discussed below, access to subject-matter experts is an issue which restricts the size of corpora available for analysis.

In the literature, there are roughly three sizes of corpora, and the size of the corpus is generally a function of the methodology and tooling used. Whenever the research methodology incorporates manual annotation as a part of the methodology, the corpus sizes range from 6 to 20 articles [9,10,13,17]. When the corpus is processed solely by researcher-developed tools, the corpus sizes are generally in the hundreds of articles [3,14,15,18]. It is only when

research leverages pre-existing tools, such as search engines, do the corpora grow to thousands of journal articles [13,16], even up to including the entire PubMed corpus.

According to Liakata et al., "abstracts of much high-quality work remain unstructured" [16]; however, many researchers focus on structured abstracts, using the structure as additional information. Some use structured abstracts as labelled training data [4], and others use journal articles with structured abstracts for its corpus exclusively [6,12,15,16]. Demner-Fushman and Lin leverage MeSH ("Medical Subject Headings") headings [16] that are applied by trained annotators [19].

While MeSH tags are generated by hundreds of trained annotators, those annotators are not employed as part of any specific research project but rather directly for PubMed itself, and no individual research project employs nearly as many annotators [19]. Nigam et al. utilize data that is labelled into the classes that research is classifying [20]. Other data [4,9,10,15,18] are annotated by only one annotator and some with expert guidance [4,18]. While Yamamoto and Takagi's corpus has 202 manually labelled articles [4], only the abstracts are annotated (1652 sentences). While the corpus in Mullen et al.'s work annotated entire articles [10], it only has 20 articles. Wilbur et al.'s work had 12 annotators [11]: the three article authors and nine graduate students in the sciences; however, they only annotated 101 sentences. From 148 RADIUM-structured journal articles, Agarwal and Yu chose 5 sentences from each section for 2960 sentences [6], 2000 of which were annotated by the article author and one of five biomedical researchers. Shatkay et al. had eight well-trained annotators [13], and while each sentence was annotated by three different annotators, they annotated only 10,000 sentences. The largest corpus is Liakata et al.'s work [14], which had 20 annotators who were a mix of chemistry PhD s and post-doctoral students, and yet they annotated only 265 articles. This corpus is used as the gold standard for the methods discussed in the next section [14].

2.2 Methodology

In the literature reviewed, the most commonly used metrics for benchmarking are the F_1-score and the kappa-score. Generally, the F_1-score is used to measure classifiers classifying previously manually annotated data, whereas the kappa-score is used to measure the ability of independent annotators to reach the same conclusion with regards to annotating a given piece of text:

$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall} F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$

The F_1-score is used with classification tasks as a combined measure of a classifier's precision and recall. It is calculated by comparing the classification output of the classifier with the independently determined "gold standard" classification. This gold standard is presumed to be completely accurate and is generally established beforehand. Precision is the percentage of classifications that are correct, whereas recall is defined as the percent of available classifications actually made by the classifier. By default, an equal weighting is given to both precision and recall, but by tuning the β-parameter, the score can be adjusted to place greater emphasis on either statistic [12,15,18]; in fact, McKnight and Srinivasan explicitly state that they use 1 as the value for the β-parameter [12].

$\kappa = \frac{Pr(a) - Pr(e)}{1 - Pr(e)}$

The kappa score is the difference between observed classifications and what one would expect by random chance. It is used when there are multiple independent annotators annotating a given corpus. Their annotations are then compared to one another's, and their inter-annotator agreement is measured. It is important to note what distinguishes the kappa-score from the F_1-score. The F_1-score requires a definitive annotated copy against which the other annotated copies are compared, and the kappa-score does not. The kappa-score treats each annotated copy as equal and presumes that, a priori, there is no reason to prefer one annotated copy over another. Thus, it is used when analyzing the reproducibility of an annotation scheme [6,17].

The most common method for classifying text is through the use of machine learning with approaches in support vector machines (SVM), conditional random fields (CRF), and Naive Bayes. SVM is the single most common technique [4,9,11,13,15]. Yamamoto and Takagi

along with Liakata et al. started with linear kernel [4, 14], but Yamamoto and Takagi switched to using polynomial kernel [4]. The Naive Bayes classifier is a well-studied and widely-used classifier based on observed conditional probability [21]. From the training data, the probability of each classification in the presence of feature is observed (i.e. calculated). Then, when classifying, the presence or absence of each feature is determined, and the overall probability is calculated for each class. Finally, the one with the highest probability is selected.

Mizuta et al. and Teufel et al. [10,17], in attempting to maximize reproducibility, structure their annotation guidelines as decision trees. This approach is generally seen when the annotation is being performed by humans. Nigam et al. uses a statistical technique called "maximum entropy" [20]. The key assumption is that in the absence of additional information, one should presuppose a uniform distribution of classes. Then, additional facts are observed, such as when feature A is present, the class is Z 40% of the time, and the other 60% of the time, the classification is spread uniformly amongst the remaining classes. In the absence of feature A, a uniform distribution amongst all the classes is once again the case. Chung uses a similar technique called "Conditional Random Fields" [15]. Both maximum entropy and conditional random fields can be generalized together as different forms of "log-linear models" [22].

As is common with many machine-learning tasks, most projects [6,9,12,13,18] use k-fold cross-validation. Nearly all of the projects use 10-fold cross-validation [6,9,12,18] with a project [15] using more (fifteen) and another [13] using less (five). McKnight and Srinivasan state that they randomly split the data into folds [13]. Agarwal and Yu randomly split the data into folds of random sizes between 100 and 1,000 in increments of a 1,000 [6].

In the reviewed works, little emphasis is placed on pre-processing of input documents. Nigam et al. explicitly state that they do not filter stop-words or use stemming [20]. In Agarwal and Yu [6], they replaced all numbers with the special token "#NuMBeR" which is unlikely to occur in the source text. Chung tried a variation of the approach where researchers replace all numbers with either "INT" or "REAL", depending on whether the number in the source text represents an integer or real number value [15], where others replace citations with the token "CITE" [9].

An open question in the literature is the size of the unit of text used for annotation: should annotation occur at the sentence or clause-level? Liakata et al. admit that there is "no general consensus" [14]. Mizuta et al. choose to annotate at the clause-level [10], and Mullen et al. do both [9]. Wilbur et al. [11], by contrast, choose to annotate at the sentence level while allowing annotators to, at their own discretion, annotate at the clause level. This later causes problems when comparing annotations between annotators when the units do not match up. No reviewed research articles propose a unit of annotation larger than the sentence (e.g. such as paragraph) nor smaller than the clause (for example, the word or noun-phrase level).

2.3 Features

Within the domain of using natural language processing to classifying text in biomedical journal articles with regards to rhetorical status, most of the work reviewed does not use decision trees for their classification (however, Mizuta et al. do [10]) but instead use machine-learning. Thus, it is of the utmost importance that careful attention is paid to the selection of features used. Unsurprisingly, the features are generally derived from the words in the text. Generally, most of the reviewed literature uses what is called a "bag-of-words" approach where the focus is on the word tokens present, with little to no leveraging of the grammatical or the rhetorical structure of the document or other higher-level concepts (the use of sequences of 'n' words, called n-grams, is still considered a bag-of-words). Only one study truly veers from the model using cue phrases, syntactic structures, and word order [7].

Like most of the other articles, Shatkay et al. [13] use individual words as features, but they augment the individual word features with additional features from both the bag-of-words approach and features, which leverage deeper syntactic information. From the bag-of-words approach, they use both bi- & tri- grams, which they call "statistical phrases" and leverage deeper syntactical structure with noun- and verb- phrases, which they refer to as "syntactic phrases."

Liakata et al. [14], like Shatkay et al. [13], use individual words as features and also

augment it with bi-grams, using only features whose frequency is greater than three. Liakata et al. [14] have some other local features such as the presence of citations, sentence length, verb parts of speech, passive voice, grammatical subjects, and grammatical structures.

Without a doubt, the most consistently relied upon feature in the literature is the presence of words in the text fragments. While some researchers do incorporate a part-of-speech alongside the textual representation of a word, the words are treated, regardless of their part-of-speech, as if the part-of-speech was just part of the word token itself, and all parts-of-speech are treated uniformly. However, some researchers treat certain parts-of-speech differently. The main thrust of this has been focusing on the role that verbs play as seen in varying degrees [4,7,10,11,14]. Some researchers [7,10,11], specifically focus on verb tense; for example, Wilbur et al. find that the usage of "were" usually precedes the exposition of a finding [11]. Outside of verb tense, no other single aspect of verbs is studied as broadly. Yamamoto and Takagi examine for the presence of an auxiliary verb [4], Mizuta et al. the main verb [10], de Waard the modality of verbs [7], and Liakata et al. the passive voice [14].

After words, the most widely used feature in the literature is bi-grams. The bi-grams do not have to be directly extracted from the corpus; for example, Mullen et al. lemmatize them first [9]. Some [9,13,14] only use the bi-grams whose frequency crosses a given threshold. McKnight and Srinivasan do not use bi-grams but mention that they plan to do so in their future work [12]. Shatkay et al. do include some tri-grams [13], but their prevalence in the literature is significantly less than that of bi-grams.

An important non-word based feature that reoccurs in the literature is that of the position of the text fragment relative to the document. This is found in one degree or another in much of the research [4,6,9,12,14,15,18]. As in Yamamoto and Takagi [4] or McKnight and Srinivasan [12], the measurement is often represented as a percentage in the range of 0 to 1, where 0 is the very first sentence, and 1 is the very last. Liakata et al. use both the position of a sentence within a paragraph and within a section [14].

Despite bag-of-words being the predominant approach used in the literature reviewed, researchers do attempt to incorporate the broader context of the document through the use of

"windowing." Windowing is a technique in which, when classifying a piece of text, information derived from surrounding text is considered. For example, Agarwal and Yu use an estimated tag for the preceding sentence as a feature for classifying a sentence [6]. Kim et al. do that as well [18], but it also simply reuses features from previous sentences like Chung does [15].

Most of the articles reviewed used feature-vectors either directly (e.g. with a cosine measure) or as input into machine-learning algorithms. In either situation, how to represent the various features numerically poses a challenge: specifically, how does one quantify the presence of words and bi-grams in a text sample? Shatkay et al. [2] use simple binary values to indicate the presence or absence of a feature. Nigam et al. [20] use the term frequency in their vectors. Yamamoto and Takagi [4] use the term frequency in the sample relative to the term frequency in the corpus (i.e. term-frequency-inverse-document-frequency). McKnight and Srinivasan initially try multiple schemes [12], but "all performed similarly," so the researchers chose a binary representation.

2.4 Comparison to Field

Our research methodology clearly follows the groundwork established by others in the field. Like many others, we use biomedical publications from PubMed with structured abstracts. The size of our corpus (which is approximately 100 articles) corresponds with what is generally seen when the corpus is processed programmatically (i.e. without human intervention) but while not leveraging external resources such as the MedLine search engine, MeSH tags, or UMLS ("Unified Medical Language System")[23]. Our evaluation methodology uses the F_1-measure present in nearly all of journal articles in the field surveyed. The use of pre-processing, which we heavily employ, is seen in the literature, but not as commonly as one might expect. While the features, words, and bi-grams we employ are nearly universally used, the cosine measure is not. It does see some usage in the literature [3]; however, as most research heavily incorporates machine-learning, the value of the cosine measure would be dubious. While the

cosine measure-could be used as a feature for machine learning, as the values that went into the cosine-measure are already present, the value is questionable when one has much richer information already available for classification.

2.5 Contrast to Field

While, like the literature, our work does incorporate the F_1-measure, we, unlike many, do not calculate the kappa statistic. In the literature, the kappa statistic is applied to determine agreement on the annotations made between annotators, but we do not employ any manual annotation. If we considered each of our heuristics as an annotator, then we could calculate a kappa-score for the clause-level agreement between heuristics, but it is not certain that the measure would be indicative of the information we wish to pursue. This is not entirely unprecedented: our approach is supported since the kappa is not used in any of the machine-learning approaches reviewed. However, while our approach has a passing resemblance to machine learning (as least when contrasted with decision trees and manual annotation), it is a purely statistical technique. Over-fitting is principally a concern of machine-learning-based approaches, which our approach is not; thus, we do not perform cross-validation because we are concerned with over-fitting. Also, fairly notably, we do not exploit text-position as a feature: the abstracts that we use are structured, and the article body sections appear in a fixed order. The clause position would correlate too highly for the classes into which we are classifying for the clause position to be useful for us as a feature, and it would hamper generalizability.

2.6 Value Proposition

Our work contributes to the body of knowledge in the field chiefly in four ways: it supplements an existing body of knowledge regarding the processing of natural language in biomedical journal articles; we place a high emphasis on pre-processing, and our results are applicable outside of the biomedical niche; we explore the utility of using word similarity as a measure of

sourcing the referent section of clauses from abstracts, and our work can be used to augment existing work; finally, our work establishes a platform for enabling annotator-less research into biomedical processing.

Both the biomedical field, and the resulting field of the processing of its articles, are increasingly active areas of research. As such, anything that explores this field, especially in a novel way, is a welcome addition to the knowledge base. Our work presents a host of techniques that can be grafted onto existing approaches. Unlike a lot of other work in this domain, our work is focused solely on leveraging lexical similarity. The techniques we apply are those whose intent is only to lay bare lexical similarity that may have been occluded by grammatical necessity. Therefore, we do not attempt to leverage domain-specific resources and pre-existing tools such as UMLS or MeSH tags, etc. As such, our techniques are applicable outside the domain in which we are working. This will be increasingly important as more and more researchers start branching out beyond the present biomedical niche into other fields.

Abstracts are believed to be the distilled points of argumentation of an academic article. Therefore, there is hope that clauses from abstracts could be leveraged to indicate which areas of a research article may be the most important (i.e. important enough that the writers felt inclined to include them in the abstract) for further information extraction. Labelling the referent section is a proxy-goal that works in this direction.

One of the persistent problems within most sub-fields of natural language processing is that of finding sufficient annotated materials with which to work. It usually becomes a trade-off between the size, the accuracy, and the cost of establishing the corpus. As such, most research is done on small corpora. Our approach is annotator-free and allows for other researchers to apply our work to their own corpora.

Chapter 3

Methodology

3.1 Introduction

We have found no references in the literature to anyone doing comparable work. As such, there is no previous research which has established techniques that are known to work. Therefore, we have selected a combination of techniques that are either known in the field of natural language processing, suggested through discussions with subject matter experts, or created by our own intuition. Thus, we consider our work to be exploratory and to suggest techniques for use by others in the future.

3.2 Corpus Composition

3.2.1 Overview

Our final corpus consists of 92 biomedical journal articles, as some articles produced errors when parsed with the chosen tools. There are a total of 29,186 sentences composed of 45,778 clauses, with a total of 594,236 words. The most frequently appearing word is “the”, and it appears 29,300 times. The most common part-of-speech in the top 100 words is IN (“preposition

Filter	Words in Corpus	Corpus Size Reduction
NullFilter	36,341	
LowerCaseFilter	32,321	4,020
PoSFilter	32,832	3,509
PoSLemmatisationFilter	35,269	1,072
PunctuationFilter	36,336	5
StemmerFilter	35,023	1,318
StopWordFilter	36,092	24
SymbolFilter	36,328	13
POS_JUNK	28,516	7,825
ALL	26,153	10,188

Table 3.1: The number of unique words in the corpus depends on the filter being used.

or subordinating conjunction"), but the part-of-speech that makes up the most of the corpus is NN ("noun, singular or mass").

Of course, the choice of filter affects the number of unique words in the corpus and varies with the filter used in the configuration (see table 3.1). The choice of filter can reduce the number of words in the corpus by nearly 30%.

3.2.2 Hapax Legomena

Of the over half-million words in the corpus, 36,341 of them are distinct (using the case-sensitive string composition and part-of-speech definition). Nearly half of them (48%; 17,391 words) appear only once. Words that appear only once are called hapax legomena. Surprisingly, only 5.3% of them are numbers. As the natural sciences involve many precise measurements, one would expect that would result in specific numbers that are very large and very small numbers that appear only once. Nouns combined compose over 64% of the hapax legomena (NN, singular nouns alone were 54%), with all verbs combined (i.e. VB, VBD, VBG, VBN, VBP, VBZ) being less (9.79%) than just the adjectives (JJ, 18.52%). Adverbs (RB) formed

2.47% of the hapax, which means that the ratio of hapax legomena adverbs to verbs (16:64) is similar to that of adjectives to nouns (19:64).

Of the distinct nouns (i.e. NN, NNS, NNP, NNPS), half of them are hapax legomena; by contrast, only 37% of distinct verbs (i.e. VB, VBD, VBP, VBZ) are. The hapax legomena represent 5.4% of the nouns but only 2.4% of verbs. It is not unexpected for there to be a greater variety in nouns than in verbs, as scientific discourse requires naming very specific items.

3.2.3 Word Frequencies

On average, each word in the corpus appears 15 times; however, due to the number of hapax legomena, on average, a word that appears more than once will appear 28 times. This high frequency is also deceptive because some parts-of-speech have very few words, but those words have very high frequencies. Such a class is TO (which represents both the preposition and infinitive marker "to"), which contains only two words but represents a total of 9,583 words from the corpus. When one looks at the average frequency of words that are not hapax legomen and do not belong to a high-frequency part-of-speech (i.e. average frequency >100; there were 13 such parts of speech none of which were parts expected to convey important context), that number is 19 appearances.

The ratio of adjectives to nouns is fairly similar, whether or not one looks at unique words or total number of words (30% and 28% respectively). This also holds true for the ratio of adverbs to verbs (23% and 24% respectively). The fact that they behave similarly is interesting because adjectives are only 18% of the unique words and 10% of the total, and adverbs are only 2%. Despite there only being 37 distinct words classified as determiners (i.e. DT), they make up close to ten-percent (9.49%) of the total words.

The average noun appears 9.3 times in the corpus. Nouns of the most common class of nouns (i.e. NN, "singular or mass") appear on average less, at only 8.4 times, whereas the second-most common class of nouns (i.e. NNS "plural nouns") appear on average 13.8 times.

3.2.4 Top 100 Words

In the corpus, the most common word is “the”, followed by “of”, and then the period (“.”), the comma (“,”), and then “and”. The words with the IN part-of-speech make up 16% of the top 100 most frequent words, but their combined frequency makes up 30% of the combined frequency of the top 100. Words with NN part-of-speech (i.e. singular or mass nouns) comprise 15% of the words in the top 100, but their combined frequency is just under 5%. DT make up 11% of the top 100 and 19% of the combined frequency. There seems to be no correlation between the distribution of frequencies across the top 100 words and across the entire corpus.

Of the top 20 most frequent words in the corpus, all of them are either stop-words or punctuation. “Stop-words [are] common words such as ’the’ or ’and,’ which help build ideas but do not carry any significance themselves” [24]. Of the top 40 most frequent words, only 7 are content; extending to the top 50 words, the number of content words only rises to 11. The first “content” word is the 23rd most frequently appearing in the corpus: it is the plural noun “cells.” This is interesting because there are many more singular nouns than plural nouns in the corpus. The first singular noun is “protein” at 27th, followed by “genes”, which is 31st. The word “cell” is very close in the frequency ranking to “cells” (36th and 23rd overall, respectively). If they were counted as one word, their frequencies combined (1310 and 2111) would make cell/cells the 16th most frequent word. This finding reinforces the notion that stemming and/or the inverse-corpus-frequency feature-weighting are worth exploring.

Although stop-words dominate the higher end of the word frequency spectrum, they do not span the entire gamut. The stop-word “got” appears at the very bottom with only one appearance in the entire corpus. This is somewhat surprising as one may have expected all of the stop-words to appear frequently.

Nouns have a lot more repetition than verbs. The most frequent non-stop verb is “using”, 33rd overall, followed by “used”, which is 56th. The 76th is “shown”. The most common verbs all tend to be different forms of “to be” (i.e. “is”, “were”, “was”, “are”, “be”, “been”), followed by verbs related to the IMRAD (“Introduction, Methodology, Results, and Discussion”) structure rather than to the biological domain of the work being described (e.g. “found”,

"observed", "described"). In fact, if one combines that category of words (i.e. "shown", "described", "found", "observed", "compared", "reported", "shows", "show", "suggest", and "determine"), then it would be the 20th most frequent word found in the corpus. The most frequent biology-related verb is 215th overall ("expressed", as related to the concept of "gene expression" with a frequency of 253rd), and the next one is "treated", 255th overall, appearing only 227 times. This information regarding the frequencies of nouns and verbs suggests that nouns are much more likely to reflect the domain than the verbs. However, the research does not consider weighting different parts of speech.

3.2.5 Distribution Curve

The most common word in the corpus is "the", and it appears 29,300 times, even more times than ".", constituting an entire 5% of the corpus. Considering that there is a total of 36,341 distinct words in the corpus, that means for every distinct word in the corpus, "the" appears 0.8 times. The lowest recorded frequency is, not unexpectedly, 1, and there are 17,391 words with that frequency. Thus, the most frequent word appears more times than the number of words in the corpus that appear only once; "the" actually appears 1.7 times more than the number of hapax legomena.

The composition of our corpus is arguably atypical. While the three most frequent words in our corpus (ignoring the period and the comma, which are not considered words in the Brown corpus [25]) are the same (i.e. "the", "of", and "and") and in the same order, and the following three (i.e. "to", "a", and "in") are not, the rest of the list is noticeably different. The frequency distribution is also quite different. "The" represents only 5% of our corpus instead of 7% as in the Brown corpus [25,26]. Furthermore, our corpus also does not follow Zipf's law. Whereas Zipf's law predicts that frequency of works in the corpus is inversely proportional to the rank, in our corpus frequency decays exponentially with rank. In the Brown corpus, the 135 most frequent make up half the corpus, in ours it is the top 153 [27]. What causes our corpus to be different was not investigated in this study, but it could be due to the subject matter and/or our definition of a word being case-sensitive and including a part of speech.

Half of the corpus is made up of 36,118 distinct words with a frequency of 284 or less; the other half is made of the top 186 words. The top 3 words make up nearly 10% of the corpus; nearly a quarter of the corpus is made up of the top 8 most frequent words. On the other end of the spectrum, words with a frequency of 7 or less (representing 29,603 distinct words) make up nearly 10% of the corpus. For a quarter of the corpus, one needs only 34,655 distinct words that appear 44 or less times in the corpus.

In the corpus, there is at least one word with a frequency for every number between 1 and 142; for numbers equal to or greater than 699, if there are words with that frequency, there is only one such word. Only 5 frequencies greater than 368 have 2 words or more words.

3.2.6 Sentence Lengths

The sentences in our corpus range in length from one to a hundred words long. The average sentence is 20 words long, the median is 18.5 words, and the mode is 16 words. Nearly, without exception, as the absolute value of a sentence length from 16 increases, the frequency decreases. The notable exceptions are an uptick when going from sentences of length two to length one and amongst the last 16 sentence lengths (i.e. 66-100) where the frequencies are in the low single-digits.

It is important to remember that the corpus was not tagged by hand but rather using PostMed. As such, one would expect oddities in the data from the use of classifiers. While it is possible to form a complete sentence with only a single word—"Stop!" is a complete sentence—our corpus contains 532 one-word sentences; most of these are titles and errors in determining sentence boundaries. One such problem is when periods are not used to end a sentence but rather to indicate an abbreviation such as "Dr.". Likewise, there is a sentence in the corpus that is 100 words long.

3.3 Processing Pipeline

When processing the corpus it is done in the following steps. First, each paper is processed with StepByStep, a derivative of the STEPS2 framework. This extracts the article content from the .nxml files, tags the words according to their part of speech, and then the sentences are parsed using the Collins parser. The parsed sentences are then read in line by line where a graph representing the parse tree structure is constructed. Any input filtering is applied at this time.

When creating the words, as per our definition, a fly-weight pattern (via a factory pattern) is used so that the words themselves are "instance controlled". This means that only one instance of every word will exist. By using a factory pattern we can also easily keep track of the usage of the word in the corpus (i.e. the "corpus frequency"). When we are doing fixed expression filtering then we perform this phase twice; first, to identify the fixed expressions from the text, and again to replace the fixed expressions. We reset the word counts after the initial pass as not to distort the frequency counts for words replaced by the fixed expressions.

To perform the classification we iterate through the clauses in the abstract, creating a feature vector for each one. Then, for each clause, we iterate through the rest of the document in a manner specific to the classifier's implementation. Feature vector values are modified by the feature weightings as they are created.

Once the classifier has finished processing the document, the classifier returns it's classification. This is compared with the section of the structured abstract from which the clause was taken. If the classification is correct it is considered a "true positive" for the expected article body section type (e.g. 'results'), for the purposes of the F_1 measure. Otherwise, it is counted as a "false positive" for the article body section returned by the classifier, and a "false negative" for the known section. The F_1 measure is calculated for each article body section and their scores are combined into an arithmetic average weighted by the expected numbers of clauses for that article body section.

3.4 Feature Weighting

The use of a cosine measure to gauge similarity between text fragments was part of our initial experimental design. However, when the implementation of the cosine measure began, it became apparent that a variety of implementations of the cosine calculation are in fact possible. The basic idea of each value corresponding to a feature of that text fragment (presence or absence of a word from the corpus in our case) remains constant. However, it occurred to us that using such a method implies (by resulting in the same score for the measure) that two text fragments sharing just the word "the" are equally as similar in content and intention as two text fragments sharing the word "haemoglobin". In other words, the cosine measure considers both sets of text fragments as being equally similar even though "the" is considered a stop-word and "haemoglobin" is a much rarer word whose appearance is generally restricted to the biomedical domain.

Upon realizing this deficiency, the first approach involves sub-selecting words (such as stop-words) that would not count towards the feature vector. While this approach inspired the filter experimental variable, it is problematic. Even if a certain class of words (such as stop-words) is ignored, there remain issues with other classes of words. For example, should we consider text fragments just sharing the word "blood" (just over a trillion results in Google) to be judged as similar in content as two fragments just sharing the word "haemoglobin"? We felt the answer to that was "no" but also that two fragments just sharing "blood" should be scored as being more similar in content and intent than two fragments just sharing the word "the". It became apparent that the logical conclusion results not in two categories of words but rather a near infinite hierarchy of categories of words and that categories could be defined by their repetition within the corpus (i.e. corpus frequency). The generalization became then that text fragments sharing rare words should be judged as being more likely to have the same intent as fragments sharing more mundane words. Thus, the question was formed: can we get higher performance if we weight features (i.e. words) by their rarity (specifically, by the inverse frequency in the corpus) when creating feature vectors representing those fragments? We then extend this question to see if both fragments repeating the same words the same number of

times is indicative of both having the same intent (i.e. term frequency).

3.5 Feature-Set

Our features are represented within our system by the means of a "feature vector"—an array of measurements where each position within the array corresponds consistently with a feature. Therefore, by comparing values at the same index from multiple feature vectors, one can compare how they vary along the corresponding measure [28]. We compare our feature vectors by computing their dot-product, which is the measure of the cosine between the two vectors. When the feature vectors are identical, the value of their cosine is 1. When the feature vectors are entirely dissimilar (i.e. orthogonal, not negated), the cosine will be 0 [28].

The purpose of feature-vector cosines is to use the measure of the similarity between the feature vectors of two text fragments as a proxy for measuring the similarity of their semantic intent, which guides us in matching the clauses from the abstract to sections from the article body. The unstated assumption is that a similarity of words between text fragments is an effective indicator of similarity. To explore this assumption, we compare the performance of using the sharing of words —as a measure of overlap— to that of sharing bi-grams. To perform this comparison, we replace the presence of words in the feature vectors with the presence of bi-grams. A third option we explore is whether combining the feature-sets (that is appending a bi-gram feature vector to a word feature vector) improves the performance.

So to restate our experiment, for the experimental variable of a feature-set, we have three values: words, bi-grams, and the union of words and bi-grams (the hybrid feature-set).

3.5.1 Word Feature-Set

In the word feature-set, each position in the feature corresponds to the measure (e.g. 1 for presence or 0 for absence) of the word from the corpus corresponding with that location. This is the most likely approach and is the one used in the preceding section exploring the performance of the feature-weightings.

3.5.2 BiGram Feature-Set

The second value for this experimental variable is the usage of bi-grams. N-grams are sequences of tokens (whether the tokens are words, syllables, letters, etcetera depends on the implementation, but here we are using words), which are often used in computational linguistics. As the size of a text grows, the number of n-grams grows exponentially, especially for larger values of N, most of which appear quite rarely. When the number of n-grams is very large, their distribution is unlikely to be reflective of the texts of which the corpus is a sample. Due to these limitations and the size of our corpus, we only use n-grams of size two, which are usually called "bi-grams"; this is our second feature-set. N-grams of size 3, 4, and 5 have only recently begun to be used by researchers at Google with a corpus that is one terabyte in size and contains over a trillion words [29].

3.5.3 Hybrid Feature-Set

The third value for this experimental variable is a combination of the feature-sets. This is done by coupling the word feature-set and the bi-gram feature-set. Each position in the feature vector is still associated with only one element from the feature-set. This experimental variable is important because even if one of the other two feature-sets has clearly superior performance to the other, we know if the less performing feature-set can augment the other for increased performance. When a feature-vector is constructed for a text-fragment, the feature-set reflects the entire corpus and not just those features that are present in that article.

3.6 Filters

As stated earlier, our operative definition of a word is a part-of-speech and a case-sensitive sequence of characters denoting the text token of the word. Two words are only considered equal if both the text token and the part of speech are the same. Therefore, a word is only equal to itself, even if the two words are so similar to readers that they would consider them semantically the same word. This means that there are situations for which human readers will find a

match between two words, but our classifiers will fail to. For this reason, we chose to explore the effects of what we called "text filters". The purpose of a text filter is to systematically transform the incoming text, in a deterministic way, to remove differences between words which are believed to be superficial and to expose to the classifiers to the similarity that appears to the reader. By applying these filters to the text during input, they will allow the classifiers to make some matches similar to that of a human reader. When filters are included in configuration, words that have been processed by the filter are used in the matching process. The term "filter" is borrowed, as a metaphor, from the colour filters that are sometimes used in photography.

It is our assumption that clauses in the abstract, those which refer to specific sections of the body, would often have minor differences than the actual words with which they were composed, to express their ideas to adjust for different tenses being used or to better fit the surrounding text for flow. Other times, clauses in the abstract get rephrased for concision, especially when multiple sentences are united (e.g. a word that would have been originally capitalized at the beginning of a sentence no longer is).

When brainstorming, we established eight different filters: the null filter; the lower-case filter; the part-of-speech filter; the part-of-speech lemmatization filter; the punctuation filter; the Porter-stemmer filter [30]; the stop-word filter; and the symbol filter. A ninth, a meta-filter, was created which could combine any number of other filters. For our research, we created two such combinations of filters, which are discussed last after all other filters: the POS_JUNK and ALL_FILTER (see table 3.1).

3.6.1 Individual Filters

Null Filter The null filter is the most basic filter. It does nothing to text but it returns the unmodified word with both the original word token and the part-of-speech tag. It is used as the default filter for a configuration when no filter is otherwise specified.

Lower-Case Filter The lower-case filter leaves the part-of-speech tag unchanged, but it converts the word token to lower-case. The motivation for using this filter was that the capitaliza-

tion of the first letter of a word appearing at the beginning of a sentence is enough for it to be considered a different word by our definition. Thus, even a simple reordering could prevent what might seem like an obvious match. Consider the following two phrases: "Blood was viscous" and "The blood was viscous." Using our operative definition of word, they share only two words: "was" and "viscous" but not "blood"/"Blood". However, when the lower-case filter is applied, there are three words shared by those text fragments. When this filter is applied, it effectively removes the case-sensitivity criteria from our definition of equality between words, since all words will be entirely lower-case. Employing this filter will then illustrate the effect on performance that considering case-sensitivity has.

Part-of-Speech Filter The part-of-speech filter removes the part-of-speech from words so that they are judged entirely on the basis of their text token. Removal of the part-of-speech tag from words is implemented to reassign all words to have the same part-of-speech. This filter provides a base against which the part-of-speech tag can be compared.

Part-of-Speech Lemmatization Filter The Collins-parser tags words to nearly 50 different parts-of-speech categories. The part-of-speech lemmatization filter was designed to see if, perhaps by merging existing categories, using fewer ones would improve performance. A "lemma" is defined as "a set of lexical forms having the same stem, the same major part of speech, and the same word-sense" [28]. This process is known in natural language processing as "lemmatization". This "merging" takes all the various parts-of-speech and breaks them into distinct groups, where one member of the group is selected to be representative of that whole group, and all words with tags in that group are retagged to have the part-of-speech that was chosen as representative.

We have two distinct hypotheses for suggesting that filters aid classification. The first is that with the size of our corpus, there may be too little repetition to accurately gauge the frequency of words, and perhaps by merging categories, we can simulate the effects of a larger corpus. Confirming that this filter approximates a larger corpus is out of the scope of this particular study and is left for future work.

The second is guided by the belief that clauses are often slightly modified from the form they have in the article body when they are repeated in the abstract. These most common modifications make the clause fit in the abstract grammatically, which results in changes to the part-of-speech of some of the words in the clause. This is the part-of-speech analog of the Porter-stemmer filter and the lower-case filter [30], and it is likely that this filter will work well in combination with those filters. The following table shows the part-of-speech tags that were chosen to be representative of entire groups of parts-of-speech and which specific parts-of-speech lemmatize to them (see table 3.2).

Lemmatized Part of Speech	**Constituent Parts of Speech**
JJ	JJ, JJR, JJS
RB	RB, RBR, RBS, WRB
DT	DT, PDT, WDT
NN	EX, NN, NNP, NNS, NNPS, PRP, PRP$, WP, WP$
VB	MD, RP, VB, VBD, VBG, VBN, VBP, VBZ
SYM	DOLLAR, HASH, C_PAREN, CD, COLON, COMMA, LS, O_PAREN, PERIOD, QUOTE, SYM
left unchanged	_OTH, CC, FW, IN, POS, BOUNDARY, TO, UH

Table 3.2: With the part-of-speech filter all the parts of speech in the right-hand column are mapped to the part of speech on the left.

Punctuation Filter The punctuation filter removes all punctuation from the text fragments. When conceived, this was suspected mainly to improve performance in configurations using

the binary feature-weighting or the term-frequency feature-weighting. It would have little effect with the other feature-weightings that do not consider word frequency (i.e. the inverse-corpus-frequency and term-frequency-inverse-corpus-frequency feature-weighting), since all punctuation is so numerous in the corpus that the value of the features should be negligible. We also suspect that punctuation is a poor indicator of semantic intent, and thus the effect of its presence is "noise" rather than "signal".

Stop-Word Filter Stop-words are words that have little to no individual meaning, but they are used in language primarily to serve grammatical functions. These are some of the most commons words in the English language and are unlikely to indicate the semantic intent; further, we suspect they have a negative impact on performance as their presence or absence does not affect the intent and are considered "noise". The stop-word filter removes these words from the text.

Porter-stemmer Filter The Porter-stemmer [9.5] attempts to deterministically find the root word, or word-stem, for a given word [30]. The Porter-stemmer filter applies the Porter-stemmer to each word token while not modifying the part-of-speech [30]. If clauses are reworded in the abstract to adjust for a change in tense, then the word token would not match. In principle, however, the word root should remain the same and be discovered by the Porter-stemmer so that they could match.

POS_JUNK Filter The POS_JUNK filter is conceived as a "junk filter". It uses the filter chain to combine the filters we feel remove content that is not indicative of the content of text-fragments and whose inclusion is more likely to mislead the classifiers than to guide them: the stop-word filter, the punctuation filter, and the lower-case filter. The intuition is that a text-fragment with the POS_JUNK filter applied is still readable to humans with the original intent clearly preserved but without any unnecessary artifacts.

ALL_FILTER The ALL_FILTER is another combination filter, which includes all the filters developed (except for the part-of-speech filter, since that filter negates the effect of the part-of-speech lemmatization filter). It is composed of the punctuation filter, symbol filter, lower-case filter, part-of-speech lemmatization filter, stop-word filter, and the Porter-stemmer filter. The motivation is to distill the intended content of the text fragment being discussed by stripping away any grammatical or rhetorical edifice. Unlike the POS_JUNK filter, the meaning of the sentences would be affected since information crucial to meaning will get removed. The prefix "un-" and the suffix "-d" are removed by the Porter-stemmer filter so that "unsaturated" and "saturate" become indistinguishable and thus are treated as matching. This, however, may be undesirable. For example, if in the abstract the phrase "unsaturated" is used but the body uses the phrase "not saturated", the intent is presumably the same but only apparent to the classifier with the stemming applied. Intuitively, this can be thought of as extracting the idea that "saturation" is being discussed but not necessarily about whether or not saturation is actually taking place.

3.7 Fixed-Expressions

A simplistic treatment of words in natural language is to think of each one as being mapped on to a single meaning. While all speakers of language know that this is not the case, many language processing tools —this one included, for the most part— behave as if it were true. An issue that confronts all research into natural language processing is dealing with the fact that words can have multiple meanings. Distinguishing between multiple meanings is called "word-sense disambiguation" [28]. This is something we partially address by using a part-of-speech tagger, which allows us to distinguish "tear" as a verb (as a synonym for "to rip" or "to shred") and "tear" as a noun but not between the two different meanings of the noun "tear" (i.e. distinguishing between referring to "a rip" or "clear saline fluid secreted by the lacrimal gland" [31]). Another issue that arises with a simplistic treatment of meaning is situations in which a combination of words means something different from that which would be inferred

by an individual treatment of its parts. Idioms are the most common example: often there is not even “one hand” let alone another. Other examples are subtler: a “motor home” is not a term that can be applied to just any home that has been motorized; a house boat with an outboard motor is not something that we would call a “motor home”, despite fulfilling the meaning of each constituent word. For something to be called a “motor home”, it must be a motorized land vehicle with at least 4 wheels, an interior cabin with at least enough height for an average adult to stand-up in, with a certain expected set of amenities, etcetera. The meaning of “motor home” is an idea that is quite different than which a reductionist treatment of its constituent words would reveal. In languages like German [32], these kind of situations often result from a compound noun rather than through an adjective/noun or noun/noun pairing.

What complicates matters further is that once context is established, the entire combination is often referred to using only part of the phrase. A text, having firmly established the context of motor homes, may use just the word “home” fully intending the reader to understand the word “home” that in this situation it refers to the idea of a motor home rather than the more general notion of “home” that the word usually implies. This is often done for aesthetic or convenience reasons. In our planning phase, while we manually reviewed elements of our corpus, we observed several instances of this kind of short hand occurring in biomedical literature. For example, in “Factors Associated With Ischemic Stroke During Aspirin Therapy in Atrial Fibrillation: Analysis of 2012 Participants in the SPAF IIII Clinical Trials” [33], having established the context of “alcohol-containing drinks” in the sentence “Consumption of >= 14 alcohol-containing drinks per week was associated with a reduced risk of ischemic stroke”, the authors follow with “This effect was not significant with consumption of 7 to 13 drinks per week” where the word “drinks” is clearly intended by the authors to be understood by readers as meaning specifically “alcohol-containing drinks” and not the more general class of drinks [33].

Thus, it is hoped that if such a heuristic is developed, it can be used as an indicator that two text samples are referring to the same more specific idea than an isolated treatment of their constituent words would indicate, and it would, therefore, improve the performance of

the section-matching classifiers.

The phrases with fixed meanings are called “fixed expressions”. To utilize them in improving classifier performance, we needed to develop an operational definition of a fixed-expression. First, we generated a list of bi-grams that have relatively high frequencies in our corpus, and then we examine all the frequent bi-grams to discern any patterns:

022926 ./. /BOUNDARY	000479 ,/, a/DT	000305 ,/, as/IN	000219 /BOUNDARY Our/PRP\$
005647 of/IN the/DT	000479 ,/, but/CC	000300 is/VBZ the/DT	000218 as/RB well/RB
004346 /BOUNDARY The/DT	000473 as/IN a/DT	000300 as/IN the/DT	000212 the/DT cell/NN
003538 in/IN the/DT	000466 can/MD be/VB	000291 between/IN the/DT	000211 ,/, respectively/RB
001845 ,/, and/CC	000458 number/NN of/IN	000286 cells/NNS were/VBD	000209 ,/, or/CC
001674 ,/, the/DT	000452 with/IN a/DT	000286 expression/NN of/IN	000206 respectively/RB ./.
001591 to/TO the/DT	000439 /BOUNDARY However/RB	000286 such/JJ as/IN	000203 was/VBD used/VBN
001342 /BOUNDARY In/IN	000438 /BOUNDARY These/DT	000284 in/IN this/DT	000203 in/IN Figure/NN
001174 and/CC the/DT	000436 /BOUNDARY For/IN	000283 based/VBN on/IN	000199 using/VBG a/DT
000950 with/IN the/DT	000420 However/RB ,/,	000278 ,/, it/PRP	000199 may/MD be/VB
000944 for/IN the/DT	000390 has/VBZ been/VBN	000270 it/PRP is/VBZ	000196 the/DT number/NN
000841 from/IN the/DT	000367 using/VBG the/DT	000265 due/JJ to/TO	000196 the/DT two/CD
000841 /BOUNDARY This/DT	000366 have/VBP been/VBN	000264 is/VBZ not/RB	000195 Thus/RB ,/,
000827 ,/, we/PRP	000363 of/IN these/DT	000257 In/IN the/DT	000195 cells/NNS ,/,
000811 /BOUNDARY A/DT	000357 /BOUNDARY To/TO	000252 into/IN the/DT	000193 well/RB as/IN
000796 that/IN the/DT	000354 to/TO a/DT	000251 /BOUNDARY As/IN	000193 /BOUNDARY All/DT
000738 of/IN a/DT	000354 presence/NN of/IN	000248 /BOUNDARY Thus/RB	000190 amino/JJ acid/NN
000711 /BOUNDARY We/PRP	000344 cells/NNS ./.	000243 within/IN the/DT	000189 genes/NNS ./.
000701 ,/, which/WDT	000343 is/VBZ a/DT	000237 compared/VBN to/TO	000187 gene/NN expression/NN
000684 to/TO be/VB	000339 the/DT same/JJ	000237 and/CC a/DT	000186 E./NNP coli/NN
000664 on/IN the/DT	000338 analysis/NN of/IN	000236 shown/VBN in/IN	000186 \%/NN of/IN
000633 by/IN the/DT	000330 the/DT presence/NN	000235 of/IN this/DT	000186 could/MD be/VB
000603 in/IN a/DT	000329 ,/, in/IN	000235 ,/, with/IN	000185 did/VBD not/RB
000508 /BOUNDARY Figure/NN	000327 /BOUNDARY It/PRP	000229 used/VBN to/TO	000184 wild/JJ type/NN
000490 at/IN the/DT	000312 involved/VBN in/IN	000229 by/IN a/DT	000184 the/DT other/JJ

Figure 3.1: The 100 most frequent bigrams in the corpus with their frequencies.

In the corpus, we found 68,957 unique bi-grams from a total of 460,583 bi-grams with an average frequency of 6.7 appearances per bi-gram. However, as can be expected, the distribution is heavily skewed with only 12,537 bi-grams (i.e.˜18%) appearing more than average. The most common bi-gram was, unsurprisingly, ./. /BOUNDARY. “Boundary” (see figure 3.1), as you will recall, was a special word added before and after each sentence to mark the boundary of the sentence, so the bi-gram ./. /BOUNDARY is of a period terminating a sentence and represents 5% of all bi-gram appearances. Also, not unexpectedly, many of the most common bi-grams are artifacts of English grammar, such as starting sentences with “We”, “The”, or

"This" (these are bound to be frequent due to English grammar fixed word order placing the subject at the beginning of a clause) or containing stop-words. However, near the bottom of the top 100 most frequent bi-grams, bi-grams which we would anticipate in biomedical articles and would consider fixed-expressions such as "amino acid", "gene expression", and "wild type" start appearing.

Part of Speech	**% of Bigrams Containing**
NN	52%
NNS	18%
JJ	21%
DT	12%
IN	23%
VBN	7.2%
VBD	4.5%
RB	4.8%

Table 3.3: Breakdown by type, percentage of bi-grams containing a given part-of-speech, showing only the most common. Please note, the bi-grams percentages will not add up to 200% on account of double counting since a single bi-gram can count towards two different categories. In fact, most do: if both words in a bi-gram have the same part-of-speech it is only counted once (which occurs in 5,853 -8.5%- of bi-grams).

We also examined the composition of bi-grams. Table 3.3 shows what percentage of bi-grams contain a given part of speech. Of those bi-grams where both words were of the same POS, 83% were NN, 10% were JJ, with several (IN, VB, VBN) reaching up to 1.75%, and the rest at near 0% levels. In line with our expectations, over 73% of bi-grams contain a noun (NN, NNS, NNP, NNPS, PRP, PRP$), with over half (52%) containing a singular noun (NN). Verbs (VB, VBD, VBG, VBP, VBZ) were seen in 23% of bi-grams, and 22% had adjectives (JJ, JJR, JJS). These number are not weighted by frequency, by which we mean that bi-grams having

the same part of speech are only counted once. The reason these parts of speech do not appear more frequently in the top 100 bi-grams is because, as stated earlier, the most frequent are the result of English grammar and despite appearing very often do not have much variation.

When bi-grams are considered on a per-article basis, things change slightly. The ./boundary bi-gram is the most common in every article and appears around 250 times per article. The second most popular bi-grams have an average frequency of approximately 70. On average, journal articles in our corpus have 6.75 bi-grams in the top 10 consisting of less-meaningful parts-of-speech like DT, IN, and CC. The bi-grams that do not have nouns are the ones that one would expect to be prevalent in any English text. These were often mundane phrases like "to the", "within the", ", the", "BOUNDARY The", "of a" et cetera. These bi-grams appear at all frequencies, as a bi-gram like "for the" can appear as few as 6 times in a given article.

On the other hand, there are meaningful bi-grams that appear with high frequencies. The journal articles in the corpus have, on average, 3.42 NOUN bi-grams in top the 10. This is different from the corpus as a whole, since nouns are more likely to be article specific, so they do not appear as often in other journals. On average, each article has about 3.4 bi-grams in the top 10 with nouns as the second word. These are more promising since they represent concepts such as "binding sites", "growth rate", "edn promoter", "comparative modelling", et cetera.

The features that stood out to us were that the words were either adjective/noun or noun/-noun pairs. This is not all together unexpected since these were our initial examples when we were discussing the concept, but we had left ourselves open to the possibility of adverb-verb fixed expressions, and it is possible that they would in other corpora. To prevent too many false positives, we chose a certain frequency threshold before a bi-gram is considered a fixed expression. To preserve context, we only consider fixed-expressions on a per-article basis. We chose seven for our frequency threshold, since, at that threshold, we felt confident about the likelihood of an expression being important in the article. This gave us 1,296 fixed-expressions for the whole corpus. Of course, employing filters changes the number of fixed-expressions found. Use of filters can increase the number of fixed-expressions by discovering new ones or reduce the number by merging existing ones. The idea was to generate a unique word that

would not only replace the bi-gram in the articles in which it appears but also replace instances of the second noun on its own. For example, if in a certain article the phrase “blood cell” is determined to be a fixed expression, then, for that article, not only is every instance of that phrase “blood cell” replaced with the same randomly generated word but also all instances of the lone word “cells” are replaced, even though they do not appear as part of “blood cells”. It is hoped that this will help the classifiers find matches, especially when paired with feature-weighting that incorporates word-frequency.

Of course there are limitations to our approach. For example, the phrase “big dogs” refers to domesticated dogs that happen to be of larger than average size; however, the phrase “big cats” refers species related to undomesticated cats such as lions and does not refer to tabbies even if they may be unusually rotund. In the first case, the adjective “big” merely narrows the scope of “dog”, whereas in the second case, the same adjective “big” combines with the word “cats” to create a new concept: no attempt is made to explore or exploit this distinction.

3.8 Classifiers

3.8.1 Overview

From the very onset of this project, it was decided that we would be using the cosine-measure of feature vectors as an indicator of text-fragment similarity. From there, we moved on to how to use text-fragment similarity to decide which article section to assign to each clause from the abstract and as to how large the sections from the article body against which we were comparing the abstract clauses should be. Our original model was that the abstract represents a compilation of the most important statements from the body of the journal article. So, we expected to find that we could often match a clause from the abstract to a nearly verbatim clause somewhere in the article body. The most common difference we expected to find was modification accounting for tense changes, the usage of pronouns, and brevity. Initially, we expected our work to focus mostly on applying filters to adjust for these deviations. However, as indicated earlier, upon consultation with a subject-matter expert, we were informed that such

verbatim copying was considered a very poor practice and unlikely to be broadly encountered. Instead, we should expect a clause from the abstract to be condensing entire portions of the article.

We then reconsidered our initial approach and examined how we could leverage the text surrounding the target clause to aid in classification. We developed an idea where the text that immediately precedes and follows a clause informs the context in which a fragment of text finds itself. Therefore, for a clause in the abstract to capture the intent of a target clause, it must also capture of some of the content from the surrounding text.

Consider the situation where a clause in the abstract summarizes three consecutive sentences from the body. The abstract clause could contain three words which do not appear jointly in any of the individual sentences; each of those sentences contains only one of those words —a different word— per sentence and none of the others. Then, by combining the target clause with its paired sentence as well as the previous and following sentences, we can provide classifiers with additional information with which they can make their classifications.

The question thus becomes, how much of that context is desirable to include for optimal classification? This, it seems, depends on the underlying nature of the articles in the corpus. Assuming that each abstract clause summarizes each of the main points in the article's argumentation, then the size of the optimal context should reveal how much text one needs to understand a single point. Thus, the size of the context should also indicate how much text is needed for understanding the clause's meaning.

3.8.2 Clause Section Classifier

The clause section classifier is in essence the default classifier. It classifies abstract clauses to article body sections by choosing the section containing the clause which most closely matches, as indicated by the cosine measure for the abstract clause. If the main points worth summarizing are contained in individual clauses, then this classifier would be expected to perform best. Such a situation would reflect the "greatest hits collection" model of abstract writing we initially envisioned. However, as a result of our discussion with the subject matter expert,

```
clause_vec := make_feature_vector( clause )
best_score := 0
best_section := none
for each section in article body
    for each sentence in section
        for each sentence_clause in sentence
            sentence_clause_vec := make_feature_vector( sentence_clause )
            cosine_val := cosine( clause_vec, sentence_clause_vec )

            if ( cosine_val > best_match ) then
                best_section := section
                best_match := cosine_val
            end if
        end for
    end for
end for

return best_section
```

Figure 3.2: The clause section classifier compares the clause from the abstract with every clause in the body. It labels the abstract clause with the section from which it found the best matching clause.

we no longer consider our initial scenario as feasible and thus do not expect this to be the best performing classifier. Thus, its relevance will act primarily as a baseline for the performance of the other classifiers.

3.8.3 Clause-Triplet Classifier

The clause-triplet classifier combines the target clause with its preceding and following clause but only if those clauses are within the same sentence. If this was the best performing classifier, this would suggest that the key arguments of a biomedical article are packed tightly into individual sentences.

3.8.4 Sentence-Triplet Clause Classifier

The sentence-triplet clause classifier combines the target clause with all the clauses in its sentence and the preceding and following ones as well. Considering that the first and last sentences of a paragraph are often “linking” sentences that serve to maintain the flow between

```
clause_vec := make_feature_vector( clause )
best_score := 0
best_section := none

for each section in article body
    for each sentence in section
        for each sentence_clause in sentence
            clause_triplet :=  sentence_clause

            if( sentence_has_clause_after( sentence, sentence_clause ) ) then
                second_clause := get_next_clause_in_sentence( section, sentence )
                clause_triplet := clause_triplet + second_sentence

                if( section_has_sentence_after( sentence, second_clause ) ) then
                    third_clause := get_next_clause_in_sentence( sentence, second_clause )
                    clause_triplet := clause_triplet + third_clause
                end if
            end if

            clause_triplet_vec := make_feature_vec( clause_triplet )
            cosine_val := cosine( clause_vec, clause_triplet_vec )

            if ( cosine_val > best_match ) then
                best_section := section
                best_match := cosine_val
            end if
        end for
    end for
end for

return best_section
```

Figure 3.3: The clause-triplet classifier compares the given clause from the abstract not with individual clauses from the body, but with triplets of adjacent clauses (only from the same sentence). The section of the best matching triplet becomes the label for the abstract clause.

paragraphs, a triplet of sentences approximates a paragraph. If this is the best performing classifier, then that would suggest that abstract clauses generally summarize approximately an entire paragraph.

3.8.5 Section-Blob Classifier

The section-blob classifier takes the idea of including the context of a clause with its ultimate conclusion: it incorporates the text of a target clause's entire section. However, if this were to be the best performing classifier, then the implications are not as clear. It would not

```
clause_vec := make_feature_vector( clause )
best_score := 0
best_section := none

for each section in article body
    for each sentence in section
        s := sentence

        if( section_has_sentence_after( section, sentence ) ) then
            second_sentence := get_next_sentence_in_section( section, sentence )
            sentence_triplet := sentence_triplet + second_sentence

            if( section_has_sentence_after( section, second_sentence ) ) then
                third_sentence := get_next_sentence_in_section( section, second_sentence )
                sentence_triplet := sentence_triplet + third_sentence
            end if
        end if

        sentence_triplet_vec := make_feature_vec( sentence_triplet )
        cosine_val := cosine( clause_vec, sentence_triplet_vec )

        if ( cosine_val > best_match ) then
            best_section := section
            best_match := cosine_val
        end if
    end for
end for

return best_section
```

Figure 3.4: The sentence-triplet clause classifier compares the given clause from the abstract not with individual sentences from the body but with triplets of adjacent sentences (only from the same section). The section of the best matching triplet becomes the label for the abstract clause.

suggest that abstract clauses summarize entire sections from the body of the articles, as we already know that is not the case as there are multiple sentences (each containing at least one clause but most containing multiple clauses) summarizing each section. Instead, it would suggest that the writing in every section is so substantially different from the writing in the others that a single feature vector for the whole section is sufficient to identify clauses in the abstract as referring to the section uniquely. Since the only features we consider are words (and bigrams), this would indicate that each section has a corresponding shift in vocabulary. It would be convenient if this were the best-performing classifier as it calculates much more quickly

```
clause_vec := make_feature_vector( clause )
best_score := 0
best_section := none

for each section in article body
    s := ''''

    for each sentence in section
        s := s + sentence
    end for

    section_vec := make_feature_vector( s )

    cosine_val := cosine( clause_vec, sentence_vec )
    if ( cosine_val > best_match ) then
        best_section := section
        best_match := cosine_val
    end if
end for

return best_section
```

Figure 3.5: The section-blob classifier treats each article body section as if it were a sentence (i.e. "blob") and then finds which of these "sentences" best matches the clause to label the clause as belonging to that section.

than the other classifiers since only one feature vector is generated per section.

3.8.6 Bottom-Up Classifier

The bottom-up classifier is a hybrid of both ends of the spectrum of context size (i.e. the clause-section classifier and the section-blob classifier). It selects the three closest classifier clauses from the body of the article. Then, it selects from amongst their corresponding sections by determining which section, amongst those sections, matches that clause most closely using the same calculation as the section blob classifier. If this were the best performing classifier, it would indicate that the context of section helps to correctly identify which section, amongst several containing similar clauses, is the correct match.

3.9 Evaluation of the Proposed Methods

3.9.1 Overview

Each combination of different values for our experimental variables constitutes a configuration. Our research is focused on which of these configurations are best suited to classifying clauses for the abstract; then we look for trends in the performance of the various configurations. To evaluate the performance of various configurations, we propose using the arithmetic mean (a.k.a. "average") of the weighted F_1 scores for each article whose abstract's clauses are classified. F_1, borrowed from the field of information retrieval, will be used since it is the standard for classification tasks [28].

Since in the calculation of F_1 score everything is calculated as if it were a binary classification task and there are five different sections in the bodies of BMC articles (i.e. background, results, discussion, conclusions, and methods), we propose to first treat the classification of each section as a separate task, to calculate an F_1 for each task, and then to calculate the weighted mean of the F_1 scores where the weighting is the number of clauses in that section of the abstract. This weighted mean would become the performance score for the configuration on that article itself. The overall score for a configuration would be the arithmetic mean of its score on each article in the corpus.

The following subsections deal with establishing the definitions necessary for defining our definition of performance, by which we shall judge the efficaciousness of the different values for our experimental variables.

3.9.2 True Positives, False Positives, & False Negative

A true positive is where a classifier identifies a sample as belonging to class, and it is known that, in fact, that sample does belong to that class. A false positive is where the classifier believes that a sample belongs to class, but in actuality, it does not. In contrast, a false negative is where the classifiers believes it does not belong to a class, but it is known that it does [21].

3.9.3 Precision

Precision is the percentage of samples that are labelled as being of category Σ, that actually belong to category Σ. It can be seen as the rate of correctness. For example, if one were labelling sentences as to whether or not their rhetorical status is that of belonging to the conclusion, it would be trivial to have high precision if you only selected sentences beginning with "We conclude that...". The formal definition of precision is given as follows:

$$Precision = \frac{true-positives}{true-positives+false-positives}$$

3.9.4 Recall

Recall is the percentage of samples that actually belong to category Σ, that are labelled as belonging to category Σ. It can be seen in the rate of coverage. For example, if one attempted to classify paintings as forged or legitimate, the recall would be the rate of forged paintings that were identified as such. It is trivial to achieve high recall by labelling all paintings as forged and not attempting to identify which are not. The formal definition of recall is given as follows:

$$Recall = \frac{true-positives}{true-positives+false-negatives}$$

3.9.5 F_1 Score

The F_1 score is an F_β score where recall is valued equally with precision (i.e. β=1); it is the harmonic mean of precision and recall where both are equally weighted. The harmonic mean is used when averaging rates. In this context, precision can be viewed as the rate of correctness, whereas recall can be seen as the rate of coverage. The importance of the F_1 score is that it combines both precision and recall into a single score because for both precision and recall it is trivial to achieve a high score independently. By combining both of those parameters, the F_1 score allows us to meaningfully compare configurations since it recognizes that is desired for classifiers to be mistaken as little as possible while returning as many matches as possible. The formula for calculating the F_1 score is given below:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision+recall}$$

```
clause_vec := make_feature_vector( clause )

first_best_score := 0
first_best_section := none

second_best_score := -1
second_best_section := none

third_best_score := -2
third_best_section := none

for each section in article body
    for each sentence in section
        sentence_vec := make_feature_vector( sentence )
        cosine_val := cosine( clause_vec, sentence_vec )

        if ( cosine_val > first_best_score ) then
            third_best_section := second_best_section
            third_best_score := second_best_score

            second_best_section := first_best_section
            second_best_score := first_best_score

            first_best_section := section
            first_best_score := cosine_val
        else if ( cosine_val > second_best_score ) then
            third_best_section := second_best_section
            third_best_score := second_best_score

            second_best_section := section
            second_best_score := cosine_val
        else if ( cosine_val > third_best_score ) then
            third_best_section := section
            third_best_score := cosine_val
        end if
    end for
end for

top_sections := set( first_best_section, second_best_section, third_best_section )

best_score := 0
best_section := none

for each section in top_sections
    s := ''''

    for each sentence in section
        s := s + sentence
    end for

    section_vec := make_feature_vector( s )

    cosine_val := cosine( clause_vec, sentence_vec )
    if ( cosine_val > best_match ) then
        best_section := section
        best_match := cosine_val
    end if
end for

return best_section
```

Figure 3.6: The bottom-up classifier finds the top three sentences that best match the given clause. Then it compares the clause each against the entire text of each of those three sections. This classifier will label the clause as being from the section, of those three, that most closely matches the clause.

Chapter 4

Results

4.1 Overview

The results section is divided into several subsections. As the composition of the corpus affects each experimental variable, it is important to understand the corpus upon which the variable will be evaluated before examining the individual results. Thus, the first subsection examines the various facets of the corpus's composition such as word frequency distribution and hapax legomena. From then on, each subsequent section focuses on an individual experimental variable. The exploration of the experimental variables is arranged by the increasing scope of each variable: feature-weightings; feature-sets; word filters; fixed-expression filtering; and classifiers.

As the classifiers are the central focus of our research, they are run along-side the examination of every other experimental variable. Also, a classifier is essential for matching and thus establishing a comparison between the different values for the experimental variable. However, they are only discussed in detail extensively at the very end, as they are the experimental variable with the largest scope.

As there are 4 different feature-weightings, 3 feature-sets, 96 filter combinations, 2 settings for the filter of fixed expressions settings, and 5 different classifiers, the number of possible configurations is large, specifically 11,250. It is not feasible to examine each and every con-

figuration. For this reason, each experimental variable is explored independently, in order, and the most performing configurations from the previous section are carried over.

4.2 Performance Calculation

4.2.1 Feature-Vector Creation

For each text-fragment to be compared, a feature-vector is created. A vector whose length is the cardinality of the feature-set is also constructed. The feature-set is sorted (the method used is irrelevant as long as it is consistent) and then reiterated. Starting with the first position in feature-vector and moving to the next subsequent position for each subsequent feature in the feature-set, a value of one is stored if the feature is found in the text-fragment; otherwise a zero is placed in that position.

Then, the feature is weighted according to the feature-weighting scheme selected in the configuration. If the feature-weighting selected has a term-frequency component (i.e. term-frequency feature-weighting or term-frequency-inverse-corpus-frequency feature-weighting), then the value in that position is multiplied by just how many times that feature is found in that text-fragment. If the feature-weighting has an inverse-corpus-frequency component (i.e. inverse-corpus-frequency feature-weighting and term-frequency-inverse-corpus-frequency feature-weighting), then the value present is divided by the number times that feature is found in the entire corpus.

4.2.2 Classification Procedure

The evaluation procedure is to go through each journal article in the corpus, and, for each journal article, to go through every clause in the abstract. A feature-vector is then created for each such abstract clause. Each heuristic then traverses the document, extracting text-fragments in a manner that is specific to that heuristic. For each such text-fragment, a feature-vector is created, and it is compared to the feature-vector of the abstract clause currently being classified

by calculating the cosine-measure between the two vectors. The higher the value, the more similar the text fragments are judged to be. The clause from the abstract is then labelled as referring to the article section from which the most similar text fragment (as judged by the cosine-measure) was created.

As the abstracts we are using are structured, we can tell to which section they refer. If the label returned by the classifier matches the section indicated by the structure of the abstract, this is counted as a true-positive. If the labels do not match, then it is counted as a false-positive for the label provided by the classifier and as a false-negative for the label intended by the abstract. From these three values, the precision and recall, and ultimately the F_1-measure, are calculated for each class.

The per class (i.e. article body section) F_1-measure is computed into an F_1-measure for the entire article as a weighted average where the weighting is the known number clauses in the abstract belonging to that class. Then, the per-article F_1-measures are averaged —without weighting for article length— to determine the overall F_1-measure for the experimental configuration.

4.3 Feature-Weighting

4.3.1 Motivation

From the beginning, our approach was to attempt to leverage similarity between overlapping text fragments; however, it was not certain how much weight we should apply to any specific word. We decided then to experiment with weighting individual features along two axes. The first was to consider the frequency of that feature within the text-fragment. The second was to consider the frequency of that feature within the overall corpus. These two axes combined gave us a total of four permutations. For more, please see the corresponding subsection in the “Methodology” section.

4.3.2 Feature-Weighting

From these two ideas (i.e. whether both corpus rarity and repetition within fragment matter), we created four feature-weightings (one for each permutation): the binary feature-weighting ignores word repetition within a sentence and corpus rarity; the term-frequency feature-weighting ignores corpus rarity but incorporates inter-fragment repetition; the inverse-corpus-frequency feature-weighting ignores inter-fragment repetition but incorporates corpus rarity; and the term-frequency-inverse-corpus-frequency feature-weighting incorporates both.

4.3.3 Results

Classifier	Feature Weighting			
	bin	**tf**	**icf**	**tf-icf**
Sentence Triplet Classifier	.413	.383	.367	.369
Clause Section Classifier	.421	.414	**.405**	**.407**
Triplet Clause Classifier	**.429**	.407	**.405**	.391
Section Blob Classifier	.250	.398	.365	.394
Bottom Up Classifier	.409	**.416**	.386	.400
Average	.384	.404	.385	.392

Table 4.1: The scores here for the word feature-set, **bold** scores indicate best performance, using the F_1-score, for each feature weighting. **bin** is short for "binary", **tf** for "term-frequency", **icf** for "inverse-term-frequency", and **tf-icf** for "term-frequency-inverse-term-frequency".

The effect on performance that the choice among feature-weightings has is on the same order of magnitude as the choice among classifiers (see table 4.1). At the most extreme data points, the choice of feature-weighting increases a classifier's (section-blob classifier) worst score by 59.5% to its best score (from an F_1-score of 0.250 with the binary feature-weighting to 0.398 with the term-frequency feature-weighting; the average change was 19%), but the choice of classifier increases a feature-weighting's (binary feature-weighting) worst score by

71% (from an F_1-score of 0.250 with the section-blob classifier to 0.429 with the clause-triplet classifier, the average was 25%). However, all of these results are quite misleading, as we shall see further, due to an exceptionally ill-performing classifier/feature-weighting pairing.

The combination of the section-blob classifier along with the binary feature-weighting is notably worse than any other combination; it is the lowest performance score recorded, 35% worse than the second-worst score recorded (that score is also with the section-blob classifier, but with the inverse-corpus-frequency feature-weighting instead of the binary feature-weighting). Ignoring this unusually poor showing decreases the biggest improvement of a classifier's (sentence-triplet classifier) performance from worst to best to just 12.6% (the average was 8.7%) and with a 17.7% improvement on feature-weightings (still binary). When the performance-ranges are adjusted for the outlier, the effect on performance of choosing a feature-weighting remains comparable to that of choosing a classifier.

While the section-blob classifier does poorly in general (even ignoring the outlier, it is still the worst performing classifier on average), the extreme nature of the outlier demands analysis. As discussed earlier, the section-blob classifier creates its feature-vector using an entire section of an academic journal article. Within any given section of an academic article, the majority of words that do appear in that article appear at least once. If only the presence or absence of a word is measured, as is the case with the binary feature-weighting, then there is almost nothing to distinguish a feature-vector generated by one section from one generated by another; thus, the classification performed by the classifiers essentially becomes random. For this reason, we expected the performance of the section-blob classifier when paired with the binary feature-weighting to vary directly with the number of features.

When the section-blob classifier/binary feature-weighting interaction was ignored, the aggregate data changed in notable ways. First of all, once the section-blob classifier/binary feature-weighting pairing was omitted, the binary feature-weighting went from having the worst average performance to having the best average performance. Until the outlier was recognized, the binary feature-weighting was considered the worst feature-weighting, which is terribly misleading, as three of the classifiers have their best performance with the binary

feature-weighting. The non-section-blob classifier that did not have its best performance with the binary feature-weighting (i.e. the bottom-up classifier) is only 1.7% worse with the binary feature-weighting than with its best feature-weighting: the binary feature-weighting is second overall for that classifier.

The term-frequency feature-weighting performs strongly with all classifiers; the classifiers that have their best performance with the binary feature-weighting all have their second-best with the term-frequency feature-weighting. This contrasts with inverse-corpus-frequency feature-weighting, which performs poorly; every classifier has its worst or second-worst performance with it. The term-frequency inverse-corpus-frequency feature-weighting behaves as one would expect the product of the term-frequency feature-weighing and the inverse-corpus-frequency feature-weighting to perform. Generally, it performs worse than the universally, well-performing term-frequency feature-weighting but better than the poorly performing inverse-corpus-frequency feature-weighting. However, an interesting exception is that the clause-triplet classifier has its worst performance with the term-frequency-inverse-corpus-frequency feature-weighting, 4% worse than with the inverse-corpus-frequency feature-weighting.

4.3.4 Conclusion

The performance figures indicate that the frequency of shared words between text fragments is a better indicator of similarity than the rarity of shared words. This follows from the fact that the binary feature-weighting and the term-frequency feature-weightings are consistently top-performers for our classifiers whereas the term-frequency-inverse-corpus-frequency feature-weighting is consistently one of the worst feature-weightings for any classifier. The binary feature-weightings (along with the clause-triplet classifier) is the feature-weight used in the best performing classifier/feature-weighting configuration, which is perplexing since it captures less information regarding the context of the words in the text fragment (as discussed with regards to classifiers in the Methodology section) than any other feature-weighting and since it is the simplest, most naive implementation. Still, the results indicate that it is the overall

best-performing feature-weighting. Since the term-frequency feature-weighting captures more context than the binary feature-weighting without incorporating corpus-frequency (whose effect on performance we were initially unsure), we expected it to have performed better than the binary feature-weighting, but none of the results support our initial assumption. This suggests that information from the term-frequency can overwhelm the information regarding the presence of other words that two text fragments have in common. This is counter-intuitive as additional information generally improves performance, especially when the new information is a super-set of the older information. However, one can consider both the term-frequency and the binary feature-weightings as part of a continuum of feature-weightings where feature values are calculated by taking the term-frequency and raising it to an exponent (i.e. for the binary feature-weighting, the exponent would be zero, whereas for the term-frequency feature-weighting, the exponent would be one). Then, one can ask that if the exponent 1 is too large since it over-emphasizes repetition,‘ is there an exponent in the range 0 to 1 which would fare better than the binary feature-weighting, such as the square-root of frequency (i.e. the exponent is one-half)? The scope of this question, however, is beyond the parameters of this research.

Also, it is apparent that aggregate performance figures can be highly deceptive since not all configurations exhibit similar performance characteristics. Notably, specific classifier/feature-weighting combinations can produce results that are not predictable from other trends: for example, the clause-triplet classifier, unlike any other classifier, performs better with the inverse-corpus-frequency feature-weighting than with the term-frequency-inverse-corpus-frequency feature-weighting.

4.4 Feature-Set

4.4.1 Motivation

When looking for overlapping features between two text-fragments, it is very important to consider which features to use. The two most commonly used features in natural language processing are individual words or an ordered pair of words called a “bi-gram”. With larger

corpora, tri-grams or even tetra-grams can be used, but for a corpus of our size, bi-grams are the largest n-grams that are feasible to use. We experiment with both individually and combined, a feature which we call the hybrid feature-set. For more, please see the corresponding subsection in the "Methodology" section.

4.4.2 Overall

		Feature Weighting				
		bin	tf	icf	tf-icf	Average
Feature Set	Word	.384	.404	.385	.392	.391
	Bigram	.382	.379	.358	.365	.371
	Hybrid	.381	.371	.361	.368	.370
	Average	.382	.385	.368	.375	

Table 4.2: Average score for each feature-set feature-weight pairings.

		Feature Weighting			
		bin	tf	icf	tf-icf
Feature Set	Word	4	1	3	2
	Bigram	1	2	4	3
	Hybrid	1	2	4	3

Table 4.3: Relative ranking of feature-weightings per feature-set.

Before examining the implications of these results upon the feature-set experimental variable, it is worth examining how these new results reflect the trends that the previous data suggested. Overall, the term-frequency classifier has the highest average across the feature-sets, but when the feature-sets are considered individually, this was only true of the word feature-set (see tables 4.2 and 4.3). The highest overall average still belongs to the pairing of the word feature-set and term-frequency feature-weighting. In both the bi-gram and hybrid feature-sets, the binary feature-weighting performed best, followed by the term-frequency

feature-weighting, the term-frequency-inverse-corpus-frequency feature-weighting, and finally the inverse-corpus-frequency feature-weighting. This is the same ranking that occurs just with the word feature-set when the pairing of the section-blob classifier and the binary feature-weighting is ignored (which remains the worst configuration). This further suggests that this particular result is an outlier and should not be depended upon too heavily to inform future experimentation.

Other than for the outlier pairing of the section-blob classifier and the binary-scorer, no other classifier/scorer pairing has its performance with the word feature-set improved upon by switching to either the bi-gram feature-set or the hybrid feature-set (see table 4.4). Despite the various permutations, no classifier/feature-weighting/feature-set configuration, other than the clause-triplet classifier with the binary feature-weighting, scored closer to the optimal configuration from the previous section than the bottom-up classifier/term-frequency feature-weighting/word feature-set configuration already does. This means that despite the additional permutations of configurations resulting from the addition of the bi-gram and hybrid feature-sets, the best performing variations are the clause-triplet classifier/binary feature-weighting pairings followed by the second-best configuration from the original results. So, despite the addition of feature-sets, our understanding of optimal configurations has not changed. This means that even with the additional results from the new configurations with the addition of the bi-gram and hybrid feature-sets, the best performing configurations remain the same. The largest decrease in performance observed with the alternate feature-sets was seen with the section-blob classifier and the term-frequency feature-weighting, which drops 22% when the bi-gram feature-set is used in place of the word feature-set.

The pairing of feature-set and feature-weighting with the worst overall performance is the bi-gram feature-set and the inverse-corpus-frequency feature-weighting; however, the combination of the hybrid feature-set with the inverse-corpus-frequency score is not significantly better (<0.2%). The bigram and hybrid feature-sets do more poorly than the word feature-sets because the bi-grams may be too sparse to give interesting bi-gram similarities but do better at the section level because the probability of a bi-gram existing is higher than at the sentence

Classifier	Word				Bigram				Hybrid			
	bin	tf	icf	tf-icf	bin	tf	icf	tf-icf	bin	tf	icf	tf-icf
Sentence Triplet	.413	.383	.367	.369	.4013	.366	.363	.363	.399	.376	.365	.365
Clause Section	.421	.414	**.4047**	**.407**	.4012	.396	.373	.366	.415	**.406**	.376	.369
Triplet Clause	**.429**	.407	**.405**	.391	**.416**	**.398**	**.390**	.371	**.422**	.403	**.393**	.374
Section Blob	.250	.398	.365	.394	.294	.328	.286	.332	.278	.273	.291	.336
Bottom Up	.409	**.416**	.386	.400	.397	.409	.379	**.395**	.391	.397	.381	**.397**
Average Score	.384	.404	.385	.392	.382	.379	.358	.365	.381	.371	.358	.368
" Rank	4	1	3	2	1	2	4	3	1	2	4	3
Average w/o SB	.418	.405	.391	.392	.404	.392	.376	.374	.407	.396	.379	.376
" Rank w/o SB	1	2	4	3	1	2	3	4	1	2	3	4

Table 4.4: Scores grouped by feature set, **bold** indicates best classifier/feature-weighting per feature set, light grey is best overall, and dark grey is worst overall.

level. All classifiers perform better with the hybrid feature-set when using the inverse-corpus-frequency feature-weighting or the term-frequency-inverse-frequency feature-weighting. For the binary feature-weighting and term-frequency feature-weighting, the section-blob classifier and the bottom-up classifier perform better with the bi-gram feature-set whereas the others mostly perform better with the hybrid feature-set than they do with the bi-gram feature-set (the sentence-triplet classifier with the binary feature-weighting is the sole exception). The biggest change in performance between the two alternate feature-sets is when the pairing of the section-blob classifier is paired with term-frequency feature-weighting and is switched from the bi-gram feature-set to the hybrid feature-set: the performance is lowered by nearly 17%. However, the overall difference performance (calculated as averages of classifiers grouped by feature-weightings) is minimal with a maximum difference of 2.13% and an average difference of 0.17%.

4.4.3 Breakdown by Classifier

The addition of the results for the alternate feature-sets does little to test the edges of performance possible. The best score still remains from the clause-triplet classifier paired with the word feature-set and the binary feature-weighting. Likewise, the worst score still remains the section-blob classifier using the same pairing. In fact, when relative scores are considered, only the section-blob classifier improved at all compared to its score with the word feature-set and, even then, only for its poor pairing with the binary feature-weighting. That classifier, again the only one which showed performance better than that shown with the word feature-set, also performs better with the bi-gram feature-set when it is not augmented with the word feature-set (i.e. the hybrid feature-set). This underscores that the binary feature-weighting/word feature-set combination is a particularly poor performer when used with the section-blob classifier. This is understandable when one considers the amount of text the section-blob classifier considers as its unit of comparison (i.e. an entire section of an article); most of the words that appear in the entire article will appear at least once in that section. Therefore, with the binary feature-weighting, almost all of the features will have the value 1 indicating word presence,

meaning that different sections will differ in fewer ways.

Classifier	Bigram				Hybrid			
	bin	tf	icf	tf-icf	bin	tf	icf	tf-icf
Sentence Triplet	-.012	-.018	-.004	-.006	-.014	-.007	-.002	-.004
Clause Section	-.020	-.018	-.031	-.040	-.006	-.008	-.029	-.038
Triplet Clause	-.013	-.010	-.016	-.019	-.007	-.004	-.012	-.017
Section Blob	+.044	-.070	-.079	-.062	+.028	-.125	-.074	-.058
Bottom Up	-.003	-.007	-.007	-.005	-.018	-.019	-.005	-.005
Average Improvement	-.003	-.024	-.027	-.027	-.003	-.033	-.027	-.024

Table 4.5: Improvements on classifier/feature-weighting pairings over the word feature-set.

Classifier	Word				Bigram				Hybrid				Best
	bin	tf	icf	tf-icf	bin	tf	icf	tf-icf	bin	tf	icf	tf-icf	Rank
Sentence Triplet	8	34	44	42	17	46	51	50	20	38	48	47	8
Clause Section	3	7	15	12	18	26	40	45	6	13	37	43	3
Triplet Clause	1	11	14	31	4	22	32	41	2	16	29	39	1
Section Blob	60	21	49	28	55	54	57	53	58	59	56	52	21
Bottom Up	9	5	33	19	25	10	36	27	30	24	35	23	5

Table 4.6: Overall ranking, grouped by feature-set, top five rankings in light grey, next five in grey, and bottom five rankings in dark grey.

Three of the top five configurations (specifically the 1st, 3rd, & 5th; see table 4.6) are achieved under the word feature-set. The other two are still from the clause-triplet classifier (i.e. the classifier which is used in the best-overall configuration) using the binary feature-weighting. In fact, aside from the anomalous binary-word pairing with section-blob classifier, all binary-scorer/word feature-set performance scores appear in the top ten. The five worst scores all belong to the section-blob classifier whose best performing configuration's performance score is 21st overall. Even considering only two of the four experimental variables, we

can state confidently that it is unlikely that the section-blob classifier will have a configuration where it will be the highest performer. However, it is worth investigating further to see what other configurations, if any, can cause it to perform like its outlier. Not surprisingly, as the bottom-up classifier contains elements of the section-blob classifier's heuristic in its own heuristic, the bottom-up classifier and the section-blob classifier both show better performance with the term-frequency feature-weighting than with the binary feature-weighting.

Aside from the section-blob classifier's anomaly, every classifier/feature-weighting pairing performs better when used in conjunction with the word feature-set (see table 4.7). Thus, this suggests that feature-sets are not a fruitful field for further study as our most naive feature-set is the best performing, and the hybrid feature-set is almost always better than the bi-gram feature-set, meaning that augmenting the word feature-set does not increase performance. To reduce the number of configurations for which to analyze results, the effects of the remaining experimental variables should be inferred from their effects on the bi-gram feature-set, since the two feature-sets perform well with each other.

4.4.4 Conclusion

Compared to the word feature-set, the word and hybrid feature-sets did not perform well. No pairing of heuristic and feature-weighting, with the exception of the section-blob heuristic, is improved by substituting the word feature-set with either the bi-gram or hybrid feature-sets. This is surprising since we expected that, per our model of abstract creation, shared bi-grams would be strong indicators of the referent text-fragment of an abstract clause and thus a good predictor of article-body section.

We also expected the hybrid feature-set to out perform the bi-gram feature-set and the word feature-set due to the additional features which should allow a better measure of similarity. While the hybrid feature-set did generally improve upon the bi-gram feature-set (see table 4.8), it did not improve upon the word feature-set(see table 4.5). It appears that the only reason configurations utilizing the hybrid feature-set out perform ones with the bi-gram feature-set is because they behave like a configuration, using the word feature-set but with additional noise.

Classifier	bin			tf			icf			tf-icf		
	word	bigram	hybrid	word	bigram	hybrid	word	bigram	hybrid	word	bigram	hybrid
Sentence Triplet	8	17	20	34	46	38	44	51	48	42	50	47
Clause Section	3	18	6	7	26	13	15	40	37	12	45	43
Triplet Clause	1	4	2	11	22	16	14	32	29	31	41	39
Section Blob	60	55	58	21	54	59	49	57	56	28	53	52
Bottom Up	9	25	30	5	10	24	33	36	35	19	27	23

Table 4.7: Overall ranking, grouped by feature-weighting. For each pairing of classifier and feature-weighting, the top score is in light grey. In dark grey are the bottom 5 scores. Note, the best score for the pairing of the bottom-up classifier and the binary feature-weighting is the sixth lowest score overall.

Classifier	bin			tf			icf			tf-icf		
	word	bigram	hybrid	word	bigram	hybrid	word	bigram	hybrid	word	bigram	hybrid
Sentence Triplet	1	2	3	1	3	2	1	3	2	1	3	2
Clause Section	1	3	2	1	3	2	1	3	2	1	3	2
Triplet Clause	1	3	2	1	3	2	1	3	2	1	3	2
Section Blob	3	1	2	1	2	3	1	3	2	1	3	2
Bottom Up	1	2	3	1	2	3	1	3	2	1	3	2
Average	1.4	2.2	2.4	1	2.6	2.4	1	3	2	1	3	2

Table 4.8: Relative rank across feature-weightings per classifier/feature-set pairing. Configurations where the bi-gram feature-set performs better than the hybrid feature-set are in light grey.

4.5 Filters

4.5.1 Motivation

Our filters are preprocessing steps that can be applied on input text to transform it as it gets read in. We can use these filters to remove certain information from factoring in during later processing. This allows us to establish the effect on performance of any particular feature —whether a particular feature is part of the signal or just noise.

Our definition of a word is a sequence of characters and a part-of-speech. This means that when a word appears at the beginning of a sentence and thus has its first letter capitalized, it is no longer considered to be equal to the uncapitalized version, which would otherwise be considered "the same word". We felt that not matching such a word between the two fragments solely due to a single capitalized letter seemed spurious. Conversely, matching text-fragments to overlapping common stop-words seemed incorrect as well. So, we pre-processed the article text to expose to the algorithms matches we felt they should be making. For more, please see the corresponding subsection in the "Methodology" section.

4.5.2 Results

When the ALL_FILTER is applied, the clause-triplet classifier paired with the binary feature-weighting improves the score that is achieved with the null filter that is used in the experiments above (see table 4.10); it is worth remembering that the null filter is the filter used when no other filter is specified). This is noteworthy since this is the first configuration that performs better than the initial clause-triplet-classifier/binary feature-weighting. Not all filters have a positive impact, as the clause-triplet classifier achieves its lowest score in a configuration using the binary feature-weighting and the POS_JUNK filter. This result may not be significant, as the range between configurations across filters of clause-triplet classifier and the binary feature-weighting pairing is only 0.79%; the clause-triplet classifier shows fairly uniform performance.

Filter Name	Unique Words	Words Removed Rank	Best Score Rank	Average Score Rank
Null	36,341	10	5	2
Lower Case	32,321	3	2	8
Part of Speech	32,832	4	4	6
Lemmatisation	35,269	6	3	5
Punctuation	36,336	9	7	3
Porter Stemmer	35,023	5	6	4
Stop Word	36,092	7	8	7
Symbol	36,328	8	9	1
POS_JUNK	28,516	2	10	10
ALL	26,153	1	1	9

Table 4.9: Relative ranking of filters

The only classifier that shows any significant variation is the section-blob, which fares nearly 15% worse under its worst filter configuration compared to its best filter configuration.

There is not much that can be generalized about the impact of our various filters under the binary feature-weighting, as each classifier has its best performance under a different filter, and only the sentence-triplet classifier and the clause-section classifier have their worst performance with the same filter (i.e. the part-of-speech filter). The only constant across classifiers is that none of them performed best under the null filter, meaning that for all classifiers, there is a filter that improves their performance. We find it surprising that this holds for the section-blob classifier as well. The earlier results suggest that it suffers when the number of features is reduced, which all these filters do, leading us to expect that it would perform worse with any filter applied and thus best with the null filter. Yet, it still manages to improve on its null filter performance with four different filter configurations.

The results of filters on configurations using the term-frequency feature-weighting are more

	Classifier										
	Sentence		**Clause**		**Triplet**		**Section**		**Bottom**		**Average**
Filter:	**Triplet**		**Section**		**Clause**		**Blob**		**Up**		
NullFilter	.413		.421		.429		.250		.409		
LowerCaseFilter	.412	-0.001	.426	0.006	.430	.001	.250	-.000	.398	-.012	-.001
PoSFilter	.406	- .007	.418	-.003	.429	.000	.254	.004	.410	.001	-.001
PoSLemmatisationFilter	.410	-0.003	.420	-.001	.430	0	.252	0.002	.409	-0.001	0
PunctuationFilter	.413	.000	.419	-.002	.429	-.000	.248	-.002	.407	-.003	-.001
StemmerFilter	.4143	0.001	.421	0	.429	0	.253	0.004	.408	-0.001	0.001
StopWordFilter	.4145	.002	.421	.000	.429	-.000	.217	-.033	.4117	.002	-.006
SymbolFilter	.413	.000	.422	.001	.429	-.001	.253	.004	.4125	.003	.001
POS_Junk	.410	-.003	.42704	.006	.4278	-.002	.227	-.022	.403	-.007	-.005
ALL_FILTER	0.41	-.003	0.42695	.006	0.431	.002	0.222	-.028	0.404	-.006	-0.006
Average	.412		.422		.430		.243		.407		
max	.4145		0.42704		.431		.254		.4125		
min	0.406		0.418		0.428		0.217		0.398		
spread	0.008		.009		.003		.036		.015		
variance as % of avg	1.97%		2.12%		0.79%		14.98%		3.66%		

Table 4.10: All the pairings between filters and classifiers with the *binary* feature-weighting and word feature-set. Indicating improvement of each filter over NullFilter.

informative on account of being more consistent (see table 4.11). Whereas under the binary feature-weighting only the section-blob classifier/stop-word filter pairing had its worst performance with the stop-word filter, now all the classifiers other than the section-blob classifier have their worst performance in configurations using the stop-word filter. As for the section-blob classifier, now it has slightly even worse performance with the ALL_FILTER than it does with the stop-word filter. The sentence-triplet classifier, clause-section classifier, and the clause-triplet classifier all have their best performances configured with the lower-case filter. Even the bottom-up classifier, which has its worst performance while using the binary feature-weighting with the lower-case filter, improves its performance over the null filter when the lower-case filter is applied. Also worth noting is that both the section-blob and the bottom-up classifiers have their best performing configurations using the term-frequency feature-weighting and beat their best performing configurations using the binary feature-weighting, but no other classifiers do. This reflects what we observe with both of those feature-weightings when configured with the

	Classifier										Average
	Sentence		**Clause**		**Triplet**		**Section**		**Bottom**		
Filter:	**Triplet**		**Section**		**Clause**		**Blob**		**Up**		
NullFilter	.383		.414		.407		.398		0.4157		
LowerCaseFilter	.406	-0.001	.420	0.006	.417	.001	.391	-.000	.412	-.012	-.001
PoSFilter	0.3819	-.007	.412	-.003	.402	.000	.396	.004	.410	.001	-.001
PoSLemmatisationFilter	.383	-0.003	.413	-.001	.406	0	0.3991	0.002	.415	-0.001	0
PunctuationFilter	0.3825	.000	.414	-.002	.409	-.000	.396	-.002	.4163	-.003	-.001
StemmerFilter	.388	0.001	.415	0	.406	0	.397	0.004	.415	-0.001	0.001
StopWordFilter	.3816	.002	.391	.000	.398	-.000	.377	-.033	.392	.002	-.006
SymbolFilter	.385	.000	.413	.001	.408	-.001	0.3993	.004	.4160	.003	.001
POS_Junk	.385	-.003	.403	.006	.408	-.002	0.3993	-.022	0.416	-.007	-.005
ALL_FILTER	.391	-.003	.408	.006	.410	.002	.374	-.028	.403	-.006	-0.006
Average	.387		.410		.406		.390		.409		
max	.406		.420		.417		0.3993		0.4163		
min	0.3816		0.391		0.398		0.374		0.392		
spread	0.025		.029		.019		.025		.024		
variance as % of avg	6.44%		7.09%		4.69%		6.47%		5.85%		

Table 4.11: All the pairings between filters and classifiers with the *term-frequency* feature-weighting and word feature-set. Indicating improvement of each filter over NullFilter.

null filter. This suggests that while filters can improve performance, they cannot compensate for the feature-weighting selected. Supporting this, the worst performing configurations use the term-frequency feature-weighting for both the section-blob and bottom-up classifiers and outperform their optimal configuration utilizing the binary feature-weighting. The performance of the clause-triplet classifier follows this pattern with its worst configuration with the binary feature-weighting besting its best configuration with the term-frequency feature-weighting, and this pattern nearly continues with sentence-triplet classifier with the binary and term-frequency feature-weightings (the worst configuration with the binary feature-weighting has an F_1-score of 0.4064 whereas the best configuration with the term-frequency feature-weighting has an F_1-score of 0.4065). In fact, only the clause-section classifier has a significant overlap in performance between configurations using the binary feature-weighting and the term-frequency feature-weighting.

Under the inverse-frequency feature-weighting, the data is not as consistent as with the

	Classifier										
	Sentence		**Clause**		**Triplet**		**Section**		**Bottom**		**Average**
Filter:	**Triplet**		**Section**		**Clause**		**Blob**		**Up**		
NullFilter	.383		.396		.393		.364		0.39		
LowerCaseFilter	.345	-0.018	.362	-0.034	.375	-0.018	.329	-0.035	.360	-0.03	-0.027
PoSFilter	0.359	-0.004	.395	-0.001	.391	-0.002	.351	-0.013	.367	-0.023	-0.009
PoSLemmatisationFilter	.339	-0.025	.389	-0.007	.388	-0.006	0.354	-0.01	.384	-0.006	-0.011
PunctuationFilter	0.362	-0.002	.398	0.002	.392	-0.001	.361	-0.003	.383	-0.007	-0.002
StemmerFilter	.365	0.001	.390	-0.006	.391	-0.002	.358	-0.006	.384	-0.005	-0.004
StopWordFilter	.364	0.001	.394	-0.001	.390	-0.003	.365	0.001	.391	0.001	0
SymbolFilter	.363	0	.397	0.001	.391	-0.003	0.368	0.004	.394	0.004	0.001
POS_Junk	.345	-0.018	.370	-0.025	.373	-0.02	0.337	-0.027	0.367	-0.022	-0.023
ALL_FILTER	.353	-0.01	.371	-0.025	.383	-0.01	.321	-0.043	.363	-0.027	-0.023
Average	.356		.386		.387		.351		.378		
max	.365		.398		.393		0.368		0.394		
min	0.339		0.362		0.373		0.321		0.36		
spread	0.026		.036		.019		.047		.034		
variance as % of avg	7.26%		9.42%		5.01%		13.37%		8.94%		

Table 4.12: All the pairings between filters and classifiers with the *inverse-corpus-frequency* feature-weighting and word feature-set. Indicating improvement of each filter over NullFilter.

term-frequency feature-weighting, but there are still some discernible patterns (see table 4.12). When configured with the binary feature-weighting, the bottom-up classifier has its best configuration with the symbol-filter, and the section-blob classifier's configuration with the symbol-filter is a close second. Similar results occur when they are configured with the term-frequency feature-weighting; the bottom-up and section-blob classifiers switch places, with the bottom-up classifier having its second best configuration with the symbol filter and the section-blob having its best. With the inverse-corpus-frequency feature-weighting in the configurations, both of these classifiers perform best when configured with the symbol filter. If we had many more feature-weightings, this would be a phenomenon worthy of further study; however, as we do not, we cannot attest to the consistency of this result. Unlike with the term-frequency feature-weighting, with the inverse-corpus-frequency, the stop-word filter does not have a significantly negative effect on performance (it does slightly with the clause-triplet classifier). In fact, all the F_1-scores for configurations with the stop-word filter are within 1% of each classifier/best

filter configuration with the inverse-corpus-frequency feature-weighting.

Configured with the inverse-corpus-frequency feature-weighting, there is little consistency as to which filter causes each classifier to have its best or worst performing configuration. This suggests that the effectiveness of filters may depend on the feature-weighting applied. What is interesting is that with the inverse-corpus-frequency feature-weighting, we see that a configuration exists (i.e. the clause-section classifier with the punctuation filter) that outperforms the best configuration for the clause-triplet classifier. As for the performance overlap in the ranges between configurations using the term-frequency feature-weighting and inverse-corpus-frequency feature-weighting, there is only a significant overlap for the clause-section classifier and just barely any overlap for the section-blob classifier.

When the F_1-scores for various configurations are grouped together by filter, a clear pattern emerges: each filter achieves its best F_1-score with the binary feature-weighting using the clause-triplet classifier. This suggests that, despite there being performance implications from the interaction of filters and feature-weightings, nothing can nullify the importance of the underlying classifier chosen.

4.5.3 Conclusion

From the results of configurations utilizing the filters, there are five generalizations about the effects on performance that are strongly supported by the data. First, the best performance is still achieved in configurations with the binary feature-weighting. Second, the symbol filter consistently improve scores. Thirdly, the only consistent pattern is that, other than for the section-blob and bottom-up classifiers, the term-frequency feature-weighting has a negative impact on performance when compared to the binary feature-weighting and that the inverse-corpus-frequency feature-weighting has an even more negative impact than the term-frequency feature-weighting. This effect is more pronounced on the section-blob classifier than with bottom-up classifier. Fourthly, rarely is it optimal for the configuration to use the null filter, as almost any combination of the other experimental variables can be improved with a judicious selection of a filter. Fifthly, and finally, while filters do affect performance, their effect cannot

Classifier	Feature Weighting	Sentence Triplet	Clause Section	Triplet Clause	Section Blob	Bottom Up
NullFilter	binary	0.413	0.421	0.429	0.25	0.409
	tf	0.383	0.414	0.407	0.398	0.4157
	icf	0.363	0.396	0.393	0.364	0.39
LowerCaseFilter	binary	0.412	0.426	0.43	0.25	0.398
	tf	0.406	0.42	0.417	0.391	0.412
	icf	0.345	0.362	0.375	0.329	0.36
PoSFilter	binary	0.406	0.418	0.429	0.254	0.41
	tf	0.3819	0.412	0.402	0.396	0.41
	icf	0.359	0.395	0.391	0.351	0.367
PoSLemmatisationFilter	binary	0.41	0.42	0.43	0.252	0.409
	tf	0.383	0.413	0.406	0.3991	0.415
	icf	0.339	0.389	0.388	0.354	0.384
PunctuationFilter	binary	0.413	0.419	0.429	0.248	0.407
	tf	0.3825	0.414	0.409	0.396	0.4163
	icf	0.362	0.398	0.392	0.361	0.383
StemmerFilter	binary	0.4143	0.421	0.429	0.253	0.408
	tf	0.388	0.415	0.406	0.397	0.415
	icf	0.365	0.39	0.391	0.358	0.384
StopWordFilter	binary	0.4145	0.421	0.429	0.217	0.4117
	tf	0.3816	0.391	0.398	0.377	0.392
	icf	0.364	0.394	0.39	0.365	0.391
SymbolFilter	binary	0.413	0.422	0.429	0.253	0.4125
	tf	0.385	0.413	0.408	0.3993	0.416
	icf	0.363	0.397	0.391	0.368	0.394
POS_JUNK	binary	0.41	0.42704	0.4278	0.227	0.403
	tf	0.385	0.399	0.402	0.375	0.395
	icf	0.345	0.37	0.373	0.337	0.367
ALL_Filter	binary	0.41	0.42695	0.431	0.222	0.404
	tf	0.391	0.408	0.41	0.374	0.403
	icf	0.353	0.371	0.383	0.321	0.363

Table 4.13: Scores for all configurations of feature-weightings, filters, and classifiers, all with the word feature-set.

overcome the selection of a feature-weighting.

The performance of the section-blob classifier is intriguing because since it performs so poorly as compared to the other classifiers, it has a great deal of room for improvement. Thus, it serves an excellent environment in which to compare benefits of different variables. The best performing configuration with the section-blob classifier is achieved with the term-frequency feature-weighting and the symbol filter. However, even this configuration performs poorer than any configuration with any other classifier, the binary feature-weighting, and the null filters. It is also quite clear that the filters do affect performance. While often the effects are small, they can be quite positive as seen by the lower-case filter and the sentence-triplet classifier pairing with the term-frequency feature-weighting. However, not all scores can be improved with filters since the clause-triplet classifier, with the inverse-corpus-frequency feature-weighting, is best with the null filter.

There is little indication that the examination of a filter alone can predict how it will effect the performance of a classifier other than that generally reducing word variety hurts the section-blob classifier. Thus, further exploration of different filters is suggested since the potential for the exploration is quite broad. Moving forward, we keep using each classifier's best and worst performing configuration with respect to the selection of feature-weighting and filter. This allows us to speculate on the effect of the other filters on both extreme ends of performance. This will leave us with only 10 configurations per experimental variable, which is a workable balance between being large enough to be indicative, while being small enough to still be manageable.

4.6 Fixed-Expressions

4.6.1 Motivation

A lot of human communication relies on context for disambiguation and for general relaying of information. One of the things that we do to be more succinct is not to fully qualify the thing that we are discussing every time we mention it and to instead refer to it in short form

and count on the reader to infer our meaning. For example, in an article about the genes of a fruit fly, the fruit fly may be referred to in some sentences solely by the term "fly" with the "fruit" qualifier being intended to be inferred. These bi-grams are called fixed-expressions, and we wanted to see if we could detect them and replace them to aid text-fragment matching. For more, please see the corresponding subsection in the "Methodology" section.

4.6.2 Results

The results of the effect of fixed-expression filtering are less uniform than those for our other experimental variables. The range of performance scores show both improvement and deterioration of about the same order of magnitude; however, those for deterioration are slightly larger and outnumber those of improvement. Enabling fixed-expressions with either the bi-gram feature-set or word feature-set can be expected to have an approximate effect of 4% on the final score. The individual effects range from a 16% improvement to a 24% decrease.

The best-case configurations universally suffer degraded performance or at best, in the case of the clause-section classifier, no change at all. However, the worst-case configurations have a good chance of improving with fixed-expression filtering, with seven out of ten configurations improving with it. Of course, it goes without saying that the worst-case configurations have more room for improvement. Despite the fact that there was more room for improvement with configurations using the bi-gram feature-set (see table 4.14), there we observe more improvement with configurations using the word feature-set (see table 4.15). Also, the sentence-triplet classifier does especially and consistently poorly with the fixed-expression filtering, which is odd since it is closely related to the clause-section and clause-triplet classifiers, which show no such weakness. It is worth noting that none of the worst-case configurations that managed to improve with fixed-expression filtering were able to better the performance of their best-case configurations. However, some, such as the bottom-up classifier/word feature-set pairing, come relatively close, meaning that there must exist some imaginable classifiers where the use of fixed-expression filtering would improve a worst-case configuration beyond a best-case configuration.

The worst-case configurations of the clause-section classifier and bottom-up classifier, without fixed-expression filtering, perform worse with the word feature-set than they do with the bi-gram feature-set. However, with fixed-expression filtering enabled, those classifiers are able to better their unfiltered bi-gram performance with the word feature-set. This is interesting since it shows again that synergy between two experimental variables (i.e. feature-set and fixed-expression filtering) can improve scores. Note, however, that this does not occur with worst-case configurations where there is more room for improvement due to the lower scores.

4.6.3 Conclusion

Fixed-expression filtering undoubtedly, under the right conditions, can dramatically improve performance. However, its effect on performance is quite variable and appears to be uniformly negative when used with near-optimal configurations. No best-case configuration is improved by the usage of fixed-expression filtering, meaning that no classifier has its best performance using fixed-expression filtering. Enabling fixed-expression filtering is not enough to compensate for a poor selection of other experimental variables, as enabling it is not enough to create a worst-case configuration to out perform a best-case configuration—at least not for any of our classifiers, but some came close so the possibility for others is quite real. The enabling of fixed-expression filtering is capable of compensating for the selection of feature-set even thought it is not able to compensate for a complete misconfiguration of the other experimental variables.

4.7 Classifiers

4.7.1 Motivation

While early on we had decided to focus on the cosine measure of similarity, we were not certain how to best apply it. Therefore, we wanted to explore different heuristics that would use the cosine measure with different strategies and different sizes of text-fragments. We ended up

developing different strategies for our heuristics, with one being scaled for three different text-fragment sizes, the other with only one scale, and finally a hybrid-strategy heuristic, for a total of five heuristics. For more, please see the corresponding subsection in the "Methodology" section.

4.7.2 Results

Clause-Section Classifier

Feature-Weightings The relative performance of the clause-section classifier is consistent across different the feature-weightings. While other classifiers jumped around in rankings, the clause-section classifier was second overall for three of the feature-weightings and was the best with the term-frequency-inverse-corpus-frequency feature-weighting.

Feature-Sets The relative performance of the classifier is also consistent across the different feature-sets. Under every feature-weighting, it is third with the bi-gram feature-set, and it is second overall under three of the four feature-weightings. There are two third place rankings and a single second place finish (with the binary feature-weighting) with the hybrid feature-set and a first place ranking (with the term-frequency feature-weighting) that is only 0.7% better than the second place.

Filters While the filters have little effect, the clause-section classifier has even less effect than the other classifiers. With the binary feature-weighting and the lower-case filter (and the combined filters), the clause section classifier shows an improvement, but nothing significant appears with any of the other filters. With the term-frequency feature-weighting, it still shows some improvement; yet, it is noticeably negatively affected by the stop-word filter to the same degree as the section-blob classifier and the clause-triplet classifier.

Fixed Expressions The clause-section classifier is unique in that with fixed-expression filtering enabled (with both the word and bi-gram feature-sets), the best-case configurations show

no change in performance. However, things are more interesting with the worst-case configurations. Despite performing almost 4% worse when configured with the word feature-set rather than the bi-gram feature-set, the word feature-set improved nearly 16% with fixed-expression filtering, whereas that same filtering causes the bi-gram feature-set to drop its performance by nearly 8.5%. There is nothing in the performance of other classifiers to suggest that this ought to be the expected result of applying fixed-expression filtering on the bi-gram or word feature-sets. The only explanation is that the effect is synergistic with the combination of this specific classifier with these feature-sets.

Clause-Triplet Classifier

Feature-Weightings The clause-triplet classifier does well with the binary feature-weighting, having the best performance of all the classifiers across all the feature-weightings with the binary-scorer. Despite performing well with the binary-scorer, it does poorly with both the term-frequency and term-frequency-inverse-corpus-frequency feature-weightings. It is the only classifier that performs better with the inverse-corpus-frequency than with the term-frequency-inverse-corpus-frequency feature-weighting.

Feature-Sets With the bi-gram feature-set, the clause-triplet classifier performs relatively better with the feature-weightings without term-frequency (i.e. binary feature-weighting and inverse-corpus-frequency feature-weighting) than with the feature-weightings incorporating term-frequency (i.e. term-frequency and term-frequency-inverse-corpus-frequency), but the effect is less drastic (1st vs 2nd). This pattern is preserved exactly with the hybrid feature-set. The best performing score overall is still from the word feature-set with the binary feature-weighting. With either the bi-gram or hybrid feature-sets, the clause-triplet classifier is always first or second overall.

Filters With the binary feature-weighting, the clause-triplet classifier is almost entirely unaffected by filters; although, it shows some improvement with the lower-case filter. This continues with the term-frequency feature-weighting but not with the inverse-corpus-frequency

feature-weighting, which is not unexpected because most filters have a markedly negative impact on performance for all classifiers with the inverse-corpus-frequency feature-weighting. However, the triple-clause classifier does better with the ALL_FILTER than the POS_JUNK filter (with the inverse-corpus-frequency feature-weighting), meaning that adding additional filters, which have negative impacts on performance to the POS_JUNK filter, significantly increased the performance (which was still worse than with just the null filter), showing that multiple filters can interact with classifiers to have a synergistic performance that cannot be extrapolated from other indicators. Only the section-blob classifier exhibits this effect.

Fixed Expressions The fixed expression filtering has no effect on the worst-case configurations with either feature-set.

Sentence-Triplet Classifier

Feature-Weightings The sentence-triplet classifier shows generally poor performance, and, like the clause-triplet classifier, it does not work well with term-frequency; both times term-frequency is counted (i.e. with the term-frequency term-frequency-inverse-corpus-frequency feature-weighting), it performs even worse than the under-performing section-blob classifier. It performs best with the binary-scorer.

Feature-Sets When the performance of the sentence-triplet classifier is examined with the other feature-sets, the performance trends become clearer. The sentence-triplet classifier's relative performance is consistent, and it performs poorly in general; it is only better than the section-blob classifier. With the other feature-sets, it no longer exhibits a penalty from using the term-frequency feature-weighting; it does equally poorly with the inverse-corpus-frequency feature-weighting as it does with the term-frequency feature-weighting and the term-frequency-inverse-corpus-frequency feature-weighting. This is perhaps not surprising as it barely outperformed the section blob classifier under the word feature-set with the inverse-corpus-frequency feature-weighting. It definitely does perform optimally with the binary feature-weighting with second-best performance with the bi-gram feature-set.

Filters Using the binary feature-weighting, the sentence-triplet classifier is indifferent to filters with the exception of being moderately negatively impacted by the part-of-speech filter. However, it is greatly improved by the lower-case filter under the term-frequency feature-weighting. In fact, that pairing shows the greatest improvement by a filter of any feature-weighting/classifier pairing seen. This could be in some way related to the fact that with the word feature-set, the sentence-triplet classifier has its worst performance with the term-frequency feature-weighting. With the inverse-corpus-frequency feature-weighting, the classifier performs negatively with all filters like all the other classifiers; however, it does especially poorly with the part-of-speech lemmatization filter. We could not find a method for determining if this could be related to the negative impact seen with the part-of-speech filter under the binary feature-weighting or if this is a coincidence. Regardless, the sentence-triplet classifier seems to exhibit an unusual sensitivity to filters which generally have little to no impact on overall performance.

Fixed-Expressions Fixed-expression filtering is not conducive to increasing performance for the sentence-triplet classifier. For the best-case configurations for both the word and bi-gram feature-sets, it causes a large drop in performance, 14% and 26% respectively. For the bi-gram feature-set, this is the largest drop observed in best case performance and, furthermore, is the biggest change in performance for fixed-expression filtering observed. In fact, the next two largest changes in performance were with the worst-case configurations with the sentence-triplet classifier with the word (-24%) and bi-gram (-20%) feature-sets. In the word feature-set, it is the only classifier to show a decrease in performance for their worst case post filtering. The results from the experimentation with feature-set filtering further support that the sentence-triplet classifier's performance is somehow especially sensitive to the effects of filtering.

Section-blob Classifier

Feature-Weightings The overall performance of the section-blob classifier is poor. While it has a similar distribution of rankings as the sentence-triplet classifier, its average score is in fact

much lower, in part due to its performance with the binary feature-weighting, which is the worst recorded F-score by far. In fact, the pairing of the section-blob classifier and the binary feature-weighting is so low that when we sorted by average score, the binary-scorer performed the weakest. Yet, when that score is ignored, the binary-scorer has the best average performance. It is the only classifier to have its worst performance with the binary feature-weighting. The performance, in general, is underwhelming, but the results with the binary-scorer are still truly anomalous and extreme.

Feature-Sets With the other features-sets, the section-blob classifier's performance goes from poor but mixed to unquestionably poor. Regardless of the feature-weighting or the feature-set (i.e. bi-gram or word), it is the worst performing. When the 60 configurations of feature-weighting, classifier, and feature-set are ranked by F-score, nine of the lowest scores belong to the section-blob classifier.

Filters The section-blob classifier has a strong negative impact on performance when using the binary feature-weighting and the stop-word filter, which drops the previously worst F-score by a further 13%. The stop-word filter also has the same negative impact with the term-frequency feature-weighting, but the lower-case filter has a moderately negative impact as well. As expected, filters do not improve this classifier performance with the inverse-corpus-frequency feature-weighting, but the lowercase filter generates the worst drop in performance recorded by a filter. Ironically, the stop-word filter has a slightly positive effect on performance with the inverse-corpus-frequency feature-weighting.

Fixed-Expressions The biggest recorded improvement on a worst-case configuration from fixed-expression is on the section-blob classifier with the bi-gram feature-set where it improved by 16%. However, with the worst case on the word filter, fixed-expressions had no change. The best-case configurations—like all the other classifiers beyond clause-section classifier—have decreased with the filtering, with the section-blob classifier showing the biggest decrease in performance with the word feature-set. However, its best-case configuration has the smallest

decrease in performance with the bi-gram feature-set.

Bottom-Up Classifier

Feature-Weightings The bottom-up classifier exhibits a wide range of rankings in performance, from best (with the term-frequency feature-weighting) to almost last (i.e. 4th overall with the binary feature-weighting). Despite being second-last with the binary-scorer, its performance is much closer to that of the other classifier rather than worse than the section-blob classifier; thus, it is hard to determine how the performance is affected by its "parentage" of the section-blob classifier. Like the section-blob classifier, it has its two-best performances with the term-frequency feature-weighting and the term-frequency-inverse-corpus-frequency feature-weighting. With the term-frequency-inverse-corpus-frequency feature-weighting, the bottom-up classifier is second-overall, while its parent classifiers, clause-section classifier & section-blob classifier, are 1st and 3rd respectively. Again, it is hard to determine how the overall ranking is actually affected by the parent classifiers and how much is coincidence.

Feature-Sets Across the feature-sets, the bottom-up classifier displays wide-ranging performance scores from performance only better than the section-blob classifier with the binary-scorer, to two first overall finishes with the term-frequency-inverse-corpus-frequency feature-weighting. It no longer appears that the overall performance of the bottom-up classifier mimics that of the section-blob classifier, as the section-blob classifier is consistently the worst here. Concerning the relative performance with the bi-gram feature-set, it follows that of the word feature-set with a third place overall score and with the inverse-corpus-frequency feature-weighting a close second. However, the pattern does not hold with the hybrid feature-set where it surprisingly did poorly with the term-frequency feature-weighting. One thing that is consistent, both with itself and with the section-blob classifier, is that it fares poorly with the binary-scorer, which seems to favour term frequency based feature-weightings.

Filters With the binary feature-weighting, the lower-case filter has no effect on most of the classifiers (including the section-blob classifier). It improved the performance of the clause-

section classifier, but it caused the performance for the bottom-up classifier to drop significantly. With the term-frequency feature-weighting, the classifier is slightly negatively impacted by the lower-case filter and the part-of-speech filter, both which also had slightly negative impacts on the section-blob classifier, but the effect is slight enough that it is hard to say if it is meaningful. By contrast, the performance is seriously impaired by the stop-word filter, but so is the performance of every other classifier other than that of the sentence-triplet classifier, so that is not especially notable either.

With the inverse-corpus-frequency feature-weighting, the bottom-up classifier has a strong negative interaction with the lower-case filter again, like it does with the binary feature-weighting and unlike what it has with the term-frequency feature-weighting. This could be attributed to the lack of a term-frequency term in the feature-weighting, but once again, it is difficult to discern causality.

With the term-frequency feature-weighting, it performs poorly with the stop-word filter, and that is the worst configuration for that classifier/feature-weighting pairing; however, that is also true for all classifiers other than the sentence-triplet (which still performs poorly, but has its worst term-frequency feature-weighting configuration with the ALL_FILTER). As far as its deterioration with the filter compared to the null filter, it is the median result. In contrast, with the inverse-corpus-frequency feature-weighting, it has a strong, negative interaction with the part-of-speech filter unlike any other classifier.

Fixed-Expressions Like the section-blob classifier, the bottom-up classifier has the performance for its best-case configuration severely impacted (-20% and -19% respectively) with the bi-gram feature-set. Again, like the section-blob classifier, its best-case configuration for word deteriorated but not as badly. Fixed-expression filtering provides a considerable improvement in the worst-case configuration for word, but a decrease in performance of around the same magnitude with the bi-gram feature-set.

Conclusions

It is important to remember that the focus here, in this section, is that we examine the relative performance of the various classifiers tested here and not their absolute performance. The reason is that if we can make generalizations about, for example, the effect of different feature-weightings, then we should be able to anticipate the effect of better feature-weightings in the future, no matter how much better they perform on average.

What is most intriguing is how consistent the relative performance of the various classifiers remains, as the other experimental variables change. The epitome of this is the clause-section classifier, which is consistently ranked in the middle of the results. This validates our initial choice of this as our first classifier, as it is the simplest, the most intuitive, and it makes for a solid baseline. The fact that it is almost never the most performing supports our idea that additional context should be included for matching.

Judging from the consistent top performance, and the highest individual scores, the triple-clause classifier is the best performing of all the test classifiers. If our assumption that each clause in the abstract summarizes a key piece of a journal article's argumentation that would indicate that each argument is stated as an individual sentence. As the context grows to, roughly, a paragraph (i.e. the sentence-triplet classifier) or an entire section (i.e. the section-blob classifier), performance drops. However, the relatively strong performance (at least relative to its context) of the bottom-up classifier suggests that additional context can be useful when leveraged appropriately, but that mere inclusion into the feature vector is not sufficient.

The stellar performance of the clause-triplet classifier with the binary-scorer, as opposed to the section-blob classifier's poor performance with that same feature-weighting, shows just how synergistic classifiers can be with feature-weightings. The classifiers that use larger contexts perform better with feature-weightings that incorporate term-frequency. Further, classifiers with larger contexts do not do well with the binary feature-weighting since that means that there is not enough to differentiate between vectors, making the classification process essentially random. With other feature-weightings, the feature vectors for different fragments contain a lot of the same words and can still be easily distinguished if their proportions are

different.

	Heuristic:	Filter:	Mode:	Scorer:	FixExp Filtering:	F_1 Score:	Improvement:	Improvement %:
Best	Sentence Triplet Classifier	StopWordFilter	BIGRAM	1	TRUE	0.3435	-0.100	-26.30%
					FALSE	0.3819		
	Clause Section Classifier	POS_JUNK	BIGRAM	1	TRUE	0.4111	0.000	0.00%
					FALSE	0.4111		
	Section Blob Classifier	SymbolFilter	BIGRAM	tf	TRUE	0.3163	-0.022	-6.88%
					FALSE	0.3235		
	Bottom Up Classifier	PunctuationFilter	BIGRAM	tf	TRUE	0.3859	-0.046	-11.31%
					FALSE	0.4044		
Worst	Sentence Triplet Classifier	PoSLemmatisationFilter	BIGRAM	icf	TRUE	0.3305	-0.073	-20.38%
					FALSE	0.3564		
	Clause Section Classifier	LowerCaseFilter	BIGRAM	icf	TRUE	0.3677	-0.032	-8.46%
					FALSE	0.3799		
	Triplet Clause Classifier	POS_JUNK	BIGRAM	icf	TRUE	0.3955	0.000	0.00%
					FALSE	0.3955		
	Section Blob Classifier	LowerCaseFilter	BIGRAM	1	TRUE	0.2995	0.047	16.48%
					FALSE	0.2860		
	Bottom Up Classifier	LowerCaseFilter	BIGRAM	icf	TRUE	0.3729	-0.031	-8.05%
					FALSE	0.3849		
Average							-0.0286	-4.64%

Table 4.14: Effect of fixed-expression filtering on the best and worst performing configurations per classifier with the bigram feature-set.

	Heuristic:	Filter:	Mode:	Scorer:	FixExp Filtering:	F_1 Score:	Improv ement:	Improve ment %:
Best	Sentence Triplet Classifier	StopWordFilter	WORD	1	TRUE	0.3870	-0.100	-26.30%
					FALSE	0.4113		
	Clause Section Classifier	POS_JUNK	WORD	1	TRUE	0.4271	0.000	0.00%
					FALSE	0.4271		
	Section Blob Classifier	SymbolFilter	WORD	tf	TRUE		-0.022	-6.88%
					FALSE			
	Bottom Up Classifier	PunctuationFilter	WORD	tf	TRUE	0.3701	-0.046	-11.31%
					FALSE	0.4019		0.00%
Worst	Sentence Triplet Classifier	PoSLemmatisationFilter	WORD	icf	TRUE	0.3862	-0.073	-20.38%
					FALSE	0.4186		
	Clause Section Classifier	LowerCaseFilter	WORD	icf	TRUE	0.3096	-0.032	-8.46%
					FALSE	0.3369		
	Triplet Clause Classifier	POS_JUNK	WORD	icf	TRUE	0.3848	0.000	0.00%
					FALSE	0.3639		
	Section Blob Classifier	LowerCaseFilter	WORD	1	TRUE	0.3857	0.047	16.48%
					FALSE	0.3853		
	Bottom Up Classifier	LowerCaseFilter	WORD	icf	TRUE	0.2441	-0.031	-8.05%
					FALSE	0.2441		
Average							-0.0215	-4.64%

Table 4.15: Effect of fixed-expression filtering on the best and worst performing configurations per classifier with the word feature-set.

Chapter 5

Thesis Conclusions

5.1 Conclusions of Our Work

5.1.1 Feature Weighting

Aside from the choice of classifier heuristics used, no other experimental variables have as large of an impact on performance as the choice of the feature weighting method. The others, generally, had little to negative impact. An important discovery for us was that the choice of feature weighting is synergistic with the choice of classification heuristic. While most classifiers performed very well with the binary feature weighting, the section-blob classifier had the worst recorded performance. This led us to learn two important lessons: the importance of heuristic/feature-weighting synergy and the dangers of aggregate statistics, which had suggested that binary feature-weighting was poor.

5.1.2 Feature-Set

The results we observed were quite surprising. Of the earlier configurations, which are pairings of a heuristic and a feature-weighting scheme, none of the configurations were improved except for the markedly poorly performing section-blob heuristic/ binary feature-weighting. Even then, three of the five worst performing configurations used the hybrid feature-set (al-

though those three were with the section-blob classifier, see table 4.7).

Another surprising discovery is that while we expected the hybrid feature-set to out-perform the bi-gram feature-set, which it did, we also expected that it would out-perform the word feature-set. Our assumption was that with more information available than with either the word or bi-gram feature-sets, heuristics would leverage that additional information to discriminate more effectively. However, judging from the results, it appears that the only reason configurations using the hybrid feature-set out-perform ones with the bi-gram feature-set is because they behave like a configuration using the word feature-set but with additional noise. Thus, there are some fundamental problems with applying bi-grams to our task. While further examination into this may prove revealing, it does not currently appear to be a fruitful avenue for improving performance, or at least not without a tremendous amount of further work first.

5.1.3 Filters

Generally, the effect of text pre-processing on performance was disappointing. For most configurations, there was little or no change in performance (the range was from an increase of 6% to a decrease of 8%), and when there was a change in classification performance, more often than not it was negative. However, there were some encouraging findings: each heuristic was able to best its previous most-performing configurations with different types of text pre-processing.

The implementations were very easy to put into place. Also, the variations we considered were endless. Upon taking into consideration the proven (albeit muted) efficacy of this technique, we can readily support that future work should incorporate some of our text pre-processing techniques and experiment with others.

5.1.4 Fixed Expressions

For the pairing of heuristic and feature-sets (excluding hybrid as our earlier results had shown it moot), we chose the feature weighting and filter pairings that both were the best

and worst performers. We then ran each such configuration with and without fixed-expression filtering to note the impact on classification performance. While the fixed-expression filter was definitely capable of producing significant effects on performance—ranging from a 16% increase to a 24% decrease—the improvements only appeared with the least-performing configurations and even then not consistently. With the best-performing configurations, at best, the fixed-expression filtering had no effect. We were interested if perhaps the fixed-expression filtering would compensate for some negative synergies and allow previously poorly performing configurations to catch up to some of the better performing ones. Even when a significant increase of performance was noted (which was only in the worst performing configurations), it was never enough to significantly close the gap between the worst-performing configurations with fixed-expression filtering and the better-performing without it.

Unlike the text pre-processing, which enabled us to top our previous best-performing configurations, no best configuration for a heuristic was improved through the usage of fixed-expression filtering. Therefore, considering the complexity of implementation and the difficulty in implementing fixed-expression filtering (especially in comparison to a technique like text-preprocessing), we cannot, at present, encourage further work using this technique.

5.1.5 Classifiers

Overall, the best performing classification heuristic is the clause-triplet. It is the best or second-best classifier in all but two configurations (word/term-frequency and word/term-frequency-inverse-corpus-frequency) from the permutations of feature-weightings, feature-sets, and classifiers.

Clause-section, sentence-triplet, and the clause-triplet classifiers are variants on the same approach, differing only on the size of the text-fragment against which they are comparing the clauses from the abstract. Generally, their F_1-measures follow a pattern, with clause-triplet classifier, outperforming the clause-section classifier which bests the sentence-triplet classifier. While sometimes the clause-section classifier is better than the clause-triplet classifier, and sometimes the sentence-triplet classifier beats the clause-section classifier, never does the

sentence-triplet classifier exceed the clause-triplet classifier. The clause-section classifier is usually right behind the clause-triplet classifier in performance (e.g. the word/inverse-corpus-frequency, word/binary, bi-gram/term-frequency configurations). This proves one of our initial assumptions: context is an important variable.

Overall, the section-blob classifier is the worst-performing heuristic. However, in some configurations, it does do better. The section-blob classifier is not the least performing classifier in two feature-weighting/feature-set pairings: word/term-frequency (fourth) and word/term-frequency inverse-corpus-frequency where it is third (its best showing) and outperforms the sentence-triplet classifier. The improvements show that it is possible to achieve decent results with it, but the factors from which it would benefit would likely be unlike any other.

Of all of our classifiers, the bottom-up classifier's algorithm is the most complex and its behaviour the most interesting. Depending on the configuration, the bottom-up classifier can be the best (i.e. word/term-frequency), and sometimes second-worst (but never worse than the section-blob classifier). It also appears to be relatively unaffected by the bi-gram feature-set. The F_1-measure score of its composite classifiers (i.e. clause-section and section-blob) saw an average decrease of 0.027 and 0.042, respectively, relative to their F_1-scores with the word feature-set; however, the bottom-up classifier only decreased 0.008 on average, the lowest average decrease observed. The bottom-up classifier also poses interesting questions regarding context. First of all, should its performance be attributed to the small context of the initial clause matching phase of the heuristic or the large context of the secondary section matching phase? It nearly almost was worse than the clause-section classifier then would could dismiss it; however, it does well in circumstances where the clause-section, clause-triplet, and sentence-triplet classifier did poorly, such as with the term-frequency and term-frequency inverse-corpus-frequency feature-weightings. This indicates that if the binary feature-weighting is not optimum (or some other experimental variable unexplored by us), it is possible the bottom-up classifier could be the best performing. Further work on classifiers, like the bottom-up classifier, needs to be done.

5.2 Overall Conclusions

We have found that smaller and simpler generally work best. The best performing configuration we observed was the clause-triplet classifier with the word feature-set, binary feature-weighting, and all the classifiers. The clause-triplet classifier scope is approximately the size of a sentence. The second best was the clause-section classifier, also with the same feature weighting, feature-set, and filter. Considering that the debate about the proper unit of classification is between the sentence and the clause, this is an unsurprising result.

The best performing feature-weighting scheme was the binary feature-weighting. We included this, intending it to serve as a baseline; however, it seems to be the best starting choice. The more complicated methods usually did not do well; however, sometimes they did (the section-blob classifier did better with any other feature-weighting with every feature-set and the bottom-up classifier with the word feature-set performed best using the term-frequency feature-weighting and with the bigram and hybrid feature-sets using the term-frequency-inverse-corpus-frequency feature-weighting). The feature-weighting is very synergistic with the classification heuristics, where the interaction is a key component in the performance. This is unlike the text pre-processing, which, while able to boost performance, did have as wide and varied effect on performance and did not seem as tied to the choice of feature-weighting or classification heuristic.

Whenever our best F_1 scores are compared to the F_1 scores seen in other work in natural language processing tasks that use biomedical articles for their corpus, we see that our scores are much lower. However, as our methodology does not use machine-learning, nor do they attempt the same task that we do, this is not an appropriate comparison. Instead, we have explored a new area that requires further study and research.

The techniques which we applied were intended to aid classification by revealing similarity may have been obscured by our strict definition of a word. However, these techniques did not significantly improve performance even though we can construct examples in which it is clear we would like to be considered similar. Thus it seems that similarity may not be that important in classification. This definitely questions our initial and underlying assumption about how

similar the contents of abstracts are to the body of articles.

5.3 Applications of Our Work

5.3.1 Proving Ground for Pre-processing

The most wide-reaching application of our work will be in relation to our work on the pre-processing of articles. Pre-processing can be applied to any corpus and used with any technique; therefore, it has the broadest applicability to the natural language processing field. Our research provides a promising ground for measuring whether or not a given pre-processing technique exposes word and bi-gram-level similarity. It allows for the development of new, perhaps more sophisticated pre-processing techniques; further, it would be useful to evaluate their effectiveness as compared to the approaches we have developed for pre-processing text so that each word is replaced with its most common synonym. It would also be interesting to see how the efficacy of the pre-processing varies across techniques or tasks.

5.3.2 Classification as a Feature

Most of the literature we reviewed employs machine-learning for classification. Our work could be used as a feature in theirs— either directly by using our classifiers' labels as a feature or just using a cosine similarity measure. For structured abstracts, many already use the position of the sentence as a feature and that correlates directly with the referent article body section. Our classifiers have only been used with structured abstracts, and without testing them with annotated unstructured-abstracts, there is little evidence that the writing conventions which enable our classifiers in structured abstracts are also present in the unstructured abstracts.

5.4 Future Work

5.4.1 Evaluation Towards Goal

The initial vision of the project was to extract and to weigh the support for claims from biomedical journals in order to aide investigation of genetic disorders. As such a project was beyond our means, we decided initially to focus on attempting to extract the main claims, guided by our intuition that the main claims are summarized —or at least referenced— by the abstract. Thus, by matching the clauses from the abstract with the text-fragments to which they refer, we could identify the main claims of the journal article's argumentation and have them identified for extraction.

Unfortunately, we found that we lacked an appropriate corpus of sufficient size for us to judge the effectiveness of any technique developed. We had, therefore, settled on using structured abstracts as annotated abstracts where; in lieu of annotating the clauses from the abstract with their target text-fragment, the structure of the abstract annotates each clause's target text-fragment's section.

Ideally, having done the work described, the next prudent step would be to examine whether or not matching target clauses to sections of the article body allowed us to identify the sections to which they refer. During our research, the best performing classifiers all compared the similarity of words between a clause from the abstract and text-fragments (whose size depends on the particular heuristic), and, since the smaller windows performed better, there is a chance that the best-matching text-fragments are actually the referent text-fragments we sought. One should be able to extract the text-fragment that the classifiers judged as best matching the clause from the abstract. These text-fragments should then be examined to determine if they do in fact represent to what the clauses refer.

Validating this would require an annotator who is not necessarily trained but is steeped in the domain. This effort would be significantly less labour intensive than full manual annotation. The initial effort ought to be in having one or more trained annotators read through the abstract and then displaying to them each clause from that abstract and the corresponding text-fragment

from the article body. Then, the annotator could decide if or if not the clauses from the abstract are indeed summarizing that fragment from the body article. Whether or not the annotator would need to have read the article in its entirety would be part of the investigation. Since our method does not place any special constraints upon the construction of the corpus beyond that of requiring structured abstracts, it should not be problematic to pair annotators with a corpus concerning matters with which they are familiar.

If the results from human annotators were negative, it would be informative to run the classifiers we have developed against an annotated biomedical corpus where each clause from the abstract (whether or not the abstract is structured) is assigned to its referent text-fragment.

5.5 Ensemble Classification

One of the avenues that we left completely unexplored was using our classifiers in conjunction with each other. We are not certain if there were particular sentences that gave our classifiers trouble. If each classifier had issue with different ones, it might be worthwhile to have each classifier independently label a text fragment and then use the majority label. Alternate schemes where different weights are assigned to different classifiers are also still possible.

5.6 Other Domains

Finally, one of our goals was to leverage word similarity without using domain-specific tools or knowledge to allow our results to be applicable to a larger array of corpora than just biomedical. To that end, it would be interesting to see what the results would be of applying our results to other scientific disciplines or even to corpora based on the humanities. As a large share of work on research article processing is being performed in the biomedical field, it would be beneficial to have some broader applicability into other domains which have not received as much attention.

Chapter 6

Glossary

Configuration

The collection of values to which the experimental variables were set when a test was run. This includes classifier, feature-weighting, feature-set, text pre-processing filters, and whether or not fixed-expression filtering was enabled. When not mentioned explicitly, experimental values are understood to be at their default setting: for feature-weighting this is binary, for feature-set this is words, for text pre-processing filters this is the NullFilter, and by default fixed-expression filtering is not enabled.

MeSH

The National Library of Medicine's controlled vocabulary thesaurus, which consists of terms naming descriptors in a hierarchical structure that permits searching at various levels of specificity [19].

Referent

A thing (in our case a text-fragment) to which another text-fragment refers. For example, the referent of "our thesis" in "which supports our thesis" refers to the thesis of the given article.

Text Fragment

A contiguous section of text from an article, post-processing, that is going to be represented as a feature-vector.

Word

A word is considered to be a string of characters with a part-of-speech tag.

Chapter 7

Stop Words

Starting Letter	Stop Words
a	a able about across after all almost also am among an and any are as at
b	be because been but by
c	can cannot could
d	dear did do does
e	either else ever every
f	for from
g	get got
h	had has have he her hers him his how however
i	i if in into is it its
j	just
k	
l	least let like likely
m	may me might most must my
n	neither no nor not
o	of off often on only or other our own
p	

Starting Letter	Stop Words
q	
r	rather
s	said say says she should since so some
t	than that the their them then there these they this tis to too twas
u	us
v	
w	wants was we were what when where which while who whom why will with would
x	
y	yet you your
z	

Chapter 8

Bibliography

[1] B.E. White, Annotating a Corpus of Biomedical Research Texts: Two Models of Rhetorical Analysis. Western University, London, ON, 2010.

[2] L.N. Soldatova and R.D. King, An ontology of scientific experiments," J.R. Soc. Interface, vol. 3, pp. 795-803, 2006.

[3] H. Shatkay, S. Edwards, W.J. Wilbur, and M. Boguski, "Genes, themes, and microarrays: Using information retrieval for large-scale gene analysis," in Proc. of the International Conference on Intelligent Systems for Molecular Biology, 2000, vol. 8, pp. 317-28.

[4] Y. Yamamoto and T. Takagi, "Experiments with sentence classification: A sentence classification system for multi biomedical literature summarization," in Proc. of the 21st International Conference on Data Engineering Workshops, 2005, pp. 1163-1168.

[5] L. Tanabe and W.J. Wilbur, "Tagging gene and protein names in biomedical text," Bioinformatics, vol. 18, pp. 1124-1132, 2002.

[6] S. Agarwal and H. Yu, "Automatically classifying sentences in full-text biomedical articles into introduction, methods, results, and discussion," Bioinformatics, vol. 25, pp. 3174-3180, 2009.

[7] A. de Waard, "A pragmatic structure for research articles," in Proc. of the 2nd International Conference on Pragmatic Web, 2007, pp. 83-89.

[8] S. Raychaudhuri, J.T. Chang, F. Imam, and R.B. Altman, "The computational anal-

ysis of scientific literature to define and recognize gene expression clusters," Nucleic Acids Research, vol. 31, pp. 4553-4560, 2003.

[9] T. Mullen, Y. Mizuta, and N. Collier, "A baseline feature set for learning rhetorical zones using full articles in the biomedical domain," ACM SIGKDD Explorations Newsletter, vol. 7, pp. 52-58, 2005.

[10] Y. Mizuta, A. Korhonen, T. Mullen, and N. Collier, "Zone analysis in biology articles as a basis for information extraction," International Journal of Medical Informatics, vol. 75, pp. 468-487, 2006.

[11] W.J. Wilbur, A. Rzhetsky, & H. Shatkay, "New directions in biomedical text annotation: Definitions, guidelines and corpus construction," BMC Bioinformatics, vol. 7, pp. 356, 2006.

[12] L. McKnight and P. Srinivasan, "Categorization of sentence types in medical abstracts," AMIA Annu Symp Proc, 2003, pp. 440-444.

[13] H. Shatkay, F. Pan, A. Rzhetsky, and W.J. Wilbur, "Multi-dimensional classification of biomedical text: Toward automated, practical provision of high-utility text to diverse users," Bioinformatics, vol. 24, pp. 2086-2093, 2008.

[14] M. Liakata, S. Saha, S. Dobnik, C. Batchelor, and D. Rebholz-Schuhmann, "Automatic recognition of conceptualization zones in scientific articles and two life science applications," Bioinformatics, vol. 28, pp. 991-1000, 2012.

[15] G. Y. Chung, "Sentence retrieval for abstracts of randomized controlled trials," BMC Medical Informatics and Decision Making, vol. 9, 2009.

[16] D. Demner-Fushman and J. Lin, "Answering clinical questions with knowledge-based and statistical techniques," Computational Linguistics, vol. 33, pp. 63-103, 2007.

[17] S. Teufel, J. Carletta, and M. Moens, "An annotation scheme for discourse-level argumentation in research articles," in Proc. of the 9th Conference on European Chapter of the Association for Computational Linguistics, 1999, pp. 110-117.

[18] S.N. Kim, D. Martinez, L. Cavedon, and L. Yencken, "Automatic classification of sentences to support evidence based medicine," BMC Bioinformatics, vol. 12, 2011.

[19] National Library of Medicine, "Fact sheet: Medical subject headings (MeSH)," National Library of Medicine, 2013. [Online]. Available https://www.nlm.nih.gov/pubs/factsheets/mesh.html

[20] K. Nigam, J. Lafferty, and A. McCallum, "Using maximum entropy for text classification," IJCAI-99 Workshop on Machine Learning for Information Filtering, 1999.

[21] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2010.

[22] S. Maskey, Statistical Methods for NLP: Maximum Entropy Markov Models, Conditional Random Fields, 2010. [PDF document]. Available http://www.cs.columbia.edu/ smaskey/CS6998/slides

[23] National Library of Medicine, "UMLS quick start guide," National Library of Medicine, 2013. [Online]. http://www.nlm.nih.gov/research/umls/quickstart.html

[24] A. Rajaraman, J. Leskovec, and J.D. Ullman, Mining of Massive Datasets. Cambridge: Cambridge University Press, 2014. [E-book] Available http://infolab.stanford.edu/ ullman/mmds/book.pdf.

[25] W.N. Francis and H. Kucera, Brown Corpus Manual: Manual of Information to Accompany a Standard Corpus of Present-Day Edited American English, For Use With Digital Computers, Providence, Rhode Island: Brown University, 1979.

[26] M.A. Russell, Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Google+, Github, and More, 2nd ed. Sebastopol, CA: O'Reilly Media, Inc., 2014.

[27] S. Fagan and G. Ramazan, "An introduction to textual econometrics," in A Handbook of Emperical Economics and Finance, A. Ullah and D.E.A. Giles, Eds. CRC Press, 2010, pp. 133-153.

[28] D. Jurafsky and J.H. Martin, Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics, 2nd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2009.

[29] P. Norvig, "All our n-gram are belong to you," Google research blog, 2006. [Online]. Available http://googleresearch.blogspot.ca/2006/08/all-our-n-gram-are-belong-to-you.html.

[30] M.F. Porter, "An algorithm for suffix stripping," Program, vol. 14, pp. 130-137, 1980.

[31] "Tear [Def. 1a]," MedlinePlus, 2014. [Online]. Available: http://www.merriam-

webster.com/medlineplus/tear.

[32]J.B. Voyles, “German noun and adjective compounds,” Language Learning, vol. 17, pp. 9-19, 1967.

[33] R.G. Hart, L.A. Pearce, R. McBride, R.M. Rotherbart, and R.W. Asinger, “Factors associated with ischemic stroke during Aspirin therapy in atrial fibrillation: Analysis of 2012 participants in the SPAF I-III clinical trials,” Stroke: Journal of the American Heart Association, vol. 30, pp. 1223-1229, 1999.

Curriculum Vitae

Name:	Arthur Bugorski
Post-secondary Education and Degrees:	The University of Western Ontario London, Ontario, Canada 2002 - 2008 B.Sc.H. The University of Western Ontario London, Ontario, Canada 2008 - 2014 M.Sc.
Related Work Experience:	Teaching Assistant The University of Western Ontario 2008 - 2010