UNIVERSITY OF BIRMINGHAM

Final Year Project

Extracting Key Phrases and Relations from Scientific Publications

Dissertation for B.Sc in Computer Science

School of Computer Science, University of Birmingham

Author
Thomas Clarke (1443652)

Supervisor
Dr Mark Lee

April 2018

Declaration

The material contained within this thesis has not previously been submitted for a degree at the University of Birmingham or any other university. The research reported within this thesis has been conducted by the author unless indicated otherwise.

Acknowledgements

I would like to give acknowledgement to those who helped me throughout the completion of this project.

Firstly, a thank you to Dr Mark Lee for being a supportive and informative supervisor, as well as an entertaining host during project meetings.

I also wish to thank my friends and family in supporting me during the year leading preceding this dissertation, ensuring I kept on track and in a good frame of mind.

Abstract

This project presents solutions developed to solve the SemEval 2017 ScienceIE task - analysis of scientific publications to extract key information. This includes three subtasks: *(A)* key phrase extraction, *(B)* classification and *(C)* relation extraction.

To achieve subtask A, the text of a paper is parsed to find it's semantic tree. Then, each word in succession is tested in a Support Vector Machine (SVM), based around a words' semantic attributes to determine if it should be a, or part of a, key phrase. Each phrase generated is also sanitised to reduce excess information. Subtask B involved treating each key phrase as a Bag-Of-Words, and calculating the phrases' distance to each classification type using Word2Vec. Finally, subtask C experimented with using the Word2Vec representation of a phrase and the relative distances between phrases combined with an SVM to try too detect relations.

Scores of NLP go here

To explore how this system could be used, a website was created hosting the information. This used Spring Boot to create a Java based web project which supported not only an archive of processed papers, but also the means to search using query strings and automatic processing of submitted papers to the system (through using the most successful versions of systems described above). *Evaluation summary goes here...*

Keywords

Natural Language Processing, Key Phrase Extraction, Classification, Relation Extraction, Support Vector Machine, Word2Vec, Spring Boot

Contents

List of Figures

List of Tables

Chapter 1

Introduction

When conducting scientific study, being able to search existing literature around a subject can be vitally important. A search system which can automatically sort scientific papers into order, returning the one likely to be most useful first, can speed up the process of gathering this information. A system which can go further and extract important pieces of information from the paper to help present answers to user queries has the potential to be even more effective.

At SemEval 2017[1], a task which heavily applied to the above was presented: ScienceIE[2]. This natural language processing (NLP) based task was to analyse scientific papers to extract key pieces of information, classify those pieces and attempt to draw relations between them. In short, this is a information extraction problem, specifically for scientific papers. The idea behind it is to support faster research as systems will be presented with information to help better gather relevant research when querying databases of existing literature.

1.1 Aims and Objectives

This project shall initially target the main goals of ScienceIE, and one of the methods of evaluation shall be through processing of the sample data and execution of the marking tools supplied as part of the task. Explicity, the overall task is split into 3 subtasks:

- **A**: The identification of all the key phrases in a scientific publication
- **B**: The classification of each key phrase into one of the following categories:
  - **Process** (scientific models, algorithms, processes)
  - **Task** (an application, end goal, problem, task)
  - **Material** (resources, materials)
- **C**: The identification of relationships between identified key phrases, where the relation is either none, or one of the following:
  - **Hyponym-of** (where the semantic field of key phrase A is included in that of key phrase B's semantic field, but not vice versa)
  - **Synonym-of** (where the semantic field of key phrase A and B are the same)

Therefore, through research of the systems created during ScienceIE and other research in the field, the largest and most obvious goal of this project is to create a system where any scientific paper can be input, some processing happens (with no time constraints) and the desired key phrase information is produced as an output, in the expected format specified for ScienceIE. This is the *brat* annotations format, which houses all of information described above about a paper in a single text document, saved separately from the original paper.

ScienceIE have supplied sample development, training and testing data for use in those participating in the task. An example *paper* can be see in appendix A and the annotations file that goes with it can be seen in appendix B.

[1] http://alt.qcri.org/semeval2017/
[2] https://scienceie.github.io/

The above can be refereed to as the *NLP system* part of the project. To evaluate the NLP system, not only will the marking tools be used, but further analysis of the information extracted shall also be conducted; for instance exploring the differences in key phrases automatically extracted compared to the expected results (seeing cases where a shorter or longer key phrase was extracted and what difference this might make when using the generated data).

There currently existing many search engines that specifically deal with research papers; Google Scholar[3] and ScienceDirect[4] are well known, popular choices currently. As an extension to the ScienceIE task, motivated by existing search engines publicly available on the web, the secondary goal of this project is to create a *proof-of-concept (POC) product* based on the NLP system. It should use information extracted by the NLP system to present useful information to the user, given suitable input through a graphical user interface (GUI).

It should maintain a collection of scientific papers that are prepared for user query to effectively help them navigate to the most useful piece of information relating to their query first. As a minimum requirement, it should host at least the test data supplied by ScienceIE. The papers should be able to be read in full, or simply have the extracted information presented (at least in the brat format described above) for the users convenience.

The goal of this *POC system* section of the project is to be able to explore the potential effectiveness of the extracted information in relation to a researcher trying to find relevant research and how effectively it can be presented to aid in understanding it.

1.2 Report Outline

This document begins with a background to the field, which feeds into specification of what is to be explored and implemented. This will cover the NLP system in detail and outline the GUI requirements, as this shall be discussed more towards the latter parts of the paper. Following that, the NLP system implementation is reported on, concluded in the next section which evaluates the strengths and weaknesses of the NLP system. Once the NLP system has been discussed, the POC concepts shall be explained in full, the implementation discussed and evaluation completed. To sum up, a final discussion section shall review the project as a whole, and reiterate the strongest positives and note some of the points for improvement or expansion.

[3] https://scholar.google.co.uk/
[4] https://www.sciencedirect.com/

Chapter 2

Background and Literature Review

2.1 Definitions and Descriptions

Throughout this literature review there are several key natural language processing and machine learning concepts discussed. Rather than defining them as we arrive at them, a list of useful definitions is constructed here.

F1 Score

The F1 score is a metric used to evaluate predictions. A common concept to find whe evaluating binary decisions is a *confusion matrix* from which ROC analysis can be completed (Fawcett 2006). This is a system which records *true positive* and *true negative* where the gold standard and predicted data match, and *false positive* and *false negative* where gold and predicted data do not match. From this, various values can be calculated.

Accuracy is one, but often doesn't show the full story as if, on a data set where 9 out of 10 items are 'false' and just 1 item is 'true', predicting all false will get an accuracy of 90%, but no *true positive* occurrences will appear - which is bad.

Two better metrics can be calculated, which are precision and recall. They look at the rates of correct and incorrect predictions, and can be combined together (and often are in the NLP world) to produce an F1 score. This is what ScienceIE's scripts calculate, given ScienceIE's gold standard data and a researchers predictions, comparing instances where the researcher has correctly predicted key phrase boundaries, classification and relations against the gold standard data.

Tokenization

Tokenizatoin is a simple concept where a document is broken down, from one long string into individual words or symbols.

Bag-Of-Words Representation

Bag-of-Words is another simple concept, where each token is considered independently of is semantic meaning.

Term Frequency - Inverse Document Frequency (TF-IDF)

TF-IDF is a metric for assessing how important a word is in a piece of text and has many applications in NLP, including (which will be discussed later) document query (Ramos 2003). It shall also see use throughout this report.

To be calculated for a given token, the document that token came from is required and a set of documents for comparison is required. The set of documents used in this project shall be the ScienceIE training set.

The theory is, a word like "the" - which will likely appear many times in almost every document - should a very low TF-IDF score (close to 0). Unusual words however, such as "xylanases" which are likely very specific to the paper they are contained in will probably have a much higher TF-IDF value than that of "the".

Parse Trees and Part-Of-Speech (POS) Tagging

A *parse* of a sentence is a tree structure where the root node is a *sentence*, each node below that is a *POS tag* and the leaves of the tree are the words or symbols in the sentence. The POS tag tells us about the semantic meaning of that node and its children - or sub tree - where the POS tag could be *verb phrase* or *noun phrase* for example.

Support Vector Machine (SVM)

A SVM is a supervised machine learning mechanism. The input to a SVM is a series of vectors generated from some original input data. Each of these vectors is a set of features - which are simply values calculated based on the original input data. For training, these data points are labelled, indicating their class. Once trained, a SVM can be used to *predict* the label for a new data point.

The training involves attempting to find a well fitting hyperplane with a maximal margin, that separates the labelled data, after mapping that data into a higher dimensional space. This involves an algorithm for finding the distance between the mapped data points, for which a *kernel* can be specified. Furthermore, the hyperplane that is fitted can be allowed to make errors. This is where it allows training data points to be within the hyperplane margins (so may be miss classified if tested against). This can be tuned to increase SVM performance at the cost of run time increasing as well.

Clustering

Clustering is simply the idea of grouping items together that are similar. The result should be a set of sets of items, where within the group there is a high average similarity, while inter-group similarity is much lower. This is often used for classification and there are a variety of clustering algorithms that can be applied, with various algorithms performing better for various applications (Rai & Singh 2010).

2.2 ScienceIE Proceedings

Evaluating the outcome of ScienceIE at SemEval indicates potential paths for future systems and document very recent activity in the key phrase extraction area. Three papers were published from the event regarding this task.

Firstly, an overview of how successful the task was shall be conducted. The highest end-to-end F1 score achieved by any team was measured to be 0.43 for all three sub-systems combined (Augenstein, Das, Riedel, Vikraman & McCallum 2017), with each of subtasks A, B and C were 0.56, 0.44 and 0.28 respectively.

For subtask A, it was evaluated that while many high scores were achieved with recurrent neural networks, the highest scoring system was a SVM using a well-engineered lexical feature set. SVMs and neural networks were also popular choices for subtask B. For subtask C, many methods were attempted and while a convolutional neural network was the most effective, various other methods (including SVM and multinomial naïve Bayes) all achieved very similar and reasonably accurate scores (the best had an F1 score of 0.64 when evaluated solely on subtask C).

– Needs rewriting after this bit –

The best end-to-end ScienceIE team used a long short-term memory (LSTM) approach for phrase extraction, with labelling completed by a CRF based sequence tagging model (Ammar, Peters, Bhagavatula & Power 2017). Their sequence tagging model employed gazetteers built from scientific words extracted from the web. Another team (Marsi, Sikdar, Marco, Barik & Sætre 2017) also had similar ideas, using CRFs to complete some of the task using WordNet[1] as a data source for the classifier they created. Both teams here also used sensible rules to help improve their score, such as intuitively marking all instances of a key phrase as a key phrase upon finding one instance (so if *carbon* is extracted and labelled as a *material*, then all other instances are labelled to match) and exploiting hypernonym relationship's bidirectional property (so if word 1 is a hypernym of word 2, the reverse is also true and therefore recorded).

Unfortunately, while extraction and classification were generally well handled, relation extraction has very low accuracy across all teams taking part with the average F1 score only being 0.15, with the highest score being 0.28. (Ammar et al. 2017) achieved this using gazetteer built from Wikipedia[2] and

[1] https://wordnet.princeton.edu/
[2] https://en.wikipedia.org

freebase. This seems more appropriate than hand written rules, which may seem appealing as they can be somewhat tailored and provide high accuracy, but require much more effort from the developer and may not work well on unseen conditions providing low accuracy (Manning & Jurafsky 2012). As mentioned earlier, WordNet is also a potential source of information for building a classifier for relation extraction, and a study by (Snow, Jurafsky & Y. Ng 2013) compared building a classifier off of Wordnet and Wikipedia for hypernym-only extraction. The result of this shows that Wikipedia may be more suited to creating this type of classifier as it achieved an F1 score higher than using WordNet (the Wikipedia based classifier got 0.36 while the WordNet based classifier got 0.27). While an improvement, it is not ultimately a huge increase and there is no evidence either Wikipedia is better for the specific area of scientific papers (as the 2013 study was completed on a generic set of data).

The results of ScienceIE demonstrate there are several potential systems that could be implemented to answer this problem, with the best system potentially being a combination of algorithms and a voting system to select and label key phrases. The product would likely involve supervised learning and previous knowledge for some algorithms, along with unsupervised learning sections as well.

2.3 Other Revelant Background Information

Several teams from ScienceIE chose to back up key phrase extraction with a CRF. CRFs can be used for key phrase extraction alone as well (Zhang, Wang, Liu, Wu, Liao & Wang 2008), and while studies imply that CRFs (shown to have F1 scores of 0.51) are more accurate than an SVM (the most accurate at ScienceIE) this paper is slightly older than papers produced at SemEval 2017 and so even if the SVM information used then was the best that was available at the time (the SVM F1 score was 0.46), the SVM implemented at ScienceIE beat both of these scores considerably achieving an F1 of 0.56 as mentioned above.

A method not attempted at ScienceIE was unsupervised learning by clustering key phrases, a method which has potentially very accurate results that also could not only be robust again new unseen data but even different languages. The idea is that candidate key phrases are selected by some heuristic and other phrases are clustered about them. With the simplest approach, the center of a cluster is the key phrase. Various clustering methods were attempted by (Liu, Li, Zheng & Sun 2009) on top of a candidate selection process built on semantic term relatedness. They ran tests on relatively short articles and while at maximum they only achieved an F1 of 0.45, there was several improvements suggested which apply to the task at hand concerning scientific papers. Firstly, an achievable improvement for this project would be to cluster directly on noun groups as they found most clusters consisted of groups of nouns anyway, which is backed up by Augenstein et al. (Augenstein et al. 2017), who reports 93% of all key phrases are noun phrases. Furthermore, improving their initial filtering to extend it further than stop words may help reduce errors as well; improving this may be possible by employing a words TF-IDF score with some threshold. Finally, they suggested a similar algorithm be applied to longer scientific papers. ScienceIE's test data consists of extracts of scientific texts (i.e. short paragraphs), however, any unsupervised system created for this task could be ran again entire papers and then only those sections compared for evaluation later – allowing this suggestion to be evaluated.

2.4 Word2Vec

2.5 The Supplied Data Set

The ScienceIE data set consists of 50 development, 350 training and 100 test documents.

Some analysis conducted at ScienceIE (Augenstein et al. 2017) showed some characteristics of the sample key phrases included:

- Only 22% of key phrases had 5 or more tokens,
- 93% of key phrases were noun phrases,
- Only 31% of key phrases seen in the training set were also in the test set.

This means that key phrase extraction appears quite difficult, as an algorithm needs to search for short phrases, processing phrases that it likely hasn't seen instances of before. Most of the key phrases

	Minimum	Average	Maximum	Standard Deviation
Nnumber of KPs	4	19	46	8
Minimum tokens per KP	1	1	3	0.4
Average tokens per KP	1	3	8	1
Maximum tokens per KP	2	9	25	4
Number of relations	0	2	13	2
Total tokens in document	60	159	264	46

Table 2.1: Key phrase (KP), token and relation analysis for the ScienceIE training set.

being noun phrases, however, is valuable information as it helps to identify a simple heuristic that can be used when processing.

Other useful and interesting characteristics about the training set, found during this study, can be seen in table 2.1.

Papers in the ScienceIE data set have many key phrases associated with them. With an average of 19 key phrases per paper, an average of 3 tokens per key phrase (meaning on average 57 key tokens per paper) and the average document containing only 159 tokens in total, around a third of all tokens are part of key phrases. This is partly due to the documents supplied by ScienceIE being very short (all are just one paragraph) and are *extracts* of papers rather than full publications. It is not a problem that the documents for processing are short - in fact that may help as the longer the document, the harder it is to choose key phrases (Hasan & Ng 2014) - however, it may mean any algorithm created here may not scale well to full scientific papers. It seems the ScienceIE task is looking for localised key phrases, choosing several from one paragraph; while the author of a paper may choose to select just five or ten key phrases from the whole paper. While this project will focus on the ScienceIE task with the given test data, a brief look longer or full papers shall be considered.

Chapter 3

Analysis and Specification

Say what I'm going to do, but probably a bad idea to have a section for this. It may work better to just have a all of the 'what im doing' in each section when we get there.

3.1 Project Architecture

With any large software project, it is sensible to choose a platform with all the necessary tools available so the developer can achieve their goals.

3.1.1 Language

Due to the past experience of the author, Java was an obvious choice. Given extensive time working in the language during university and in industry, a thorough understanding of the programming language was already achieved, which allowed for planning of a sensible software architecture to optimise code quality and (implicitly) the potential of increased success of the systems created.

Furthermore, Java is a very popular and accessible language world wide - backed up by the active StackOverflow community (casual and professional alike) with Java being one of the most popular technologies for at least the last five years, evidenced through their user surveys 2018[1], 2017[2] and 2016[3]. Due to this, Java has extensive support for many common problems people encounter, with issues being discussed and solutions proved across various forums.

Not only is Java's popularity good for increasing support availability, many libraries and utilities are available to help developers with tasks. Along side other technologies used for more specific tasks throughout completion of the NLP system and the POC system (which shall be discussed when used), common technologies used during the development of the entire project are described below. Throughout development of the project, very little issue was caused by lack of Java support for common processes or lack of Java capability when attempting to program some process (which was a critical part of evaluating which language should be used).

Finally, Java serialisation was used throughout (and will be noted when is). Serialisation allows the system to save a Java object to disk (any file name can be chosen, but classically its postfix is `.ser`), and later be reloaded.

As a brief aside, Python is another extremely popular language used for NLP and likely could have been used for at least the first half of this project producing similar results.

log4j

log4j 2[4] is a popular and robust library developed under the Apache Software Foundation to do logging in Java. It's useful features include:

- Automatic output of logs to both terminal and file: As well as immediate visual feedback, log files can be used for later processing and evidence gathering.

[1] https://insights.stackoverflow.com/survey/2018
[2] https://insights.stackoverflow.com/survey/2017
[3] https://insights.stackoverflow.com/survey/2016
[4] https://logging.apache.org/log4j/2.x/

- Timing of events: Timing is very useful as during long runs of a system (for example, some sections of the NLP task could take hours to complete) the logs can be analysed to see how long systems take to process data, which can be considered when going forward; for instance in terms of formulating efficiently timed tests plans.
- Labelling of logs into levels such as *debug*, *info*, *error* and *fatal* messages: This can be used when analysing the logs to catch where things went wrong (filtering for error messages) and then to try to debug the system by finding information logged prior to that (with debug). During development an excellent use of this feature is to output all levels aside from 'debug' to terminal, so monitoring progress isn't overloading the executor with information, but if something does go awry the steps leading up to the bad event can be analysed in the log saved to disk.

While direct output of this will not be present in the rest of this report, it is worth noting this was an extremely useful tool for developing all of the systems to follow.

Maven

Apache Maven[5] is another important tool. Like log4j, it is developed by the Apache foundation.

Maven is a tool to help with project management and has many uses. It is based around a *project object model* (POM) configured in a `pom.xml` file at the root of a Java project, which itself has a structure defined by Maven. The key uses utilised in this project are:

- Project compilation: Maven can be used to build a project and automatically run specified or all tests, with more detailed and well formatted output than compiling Java code by hand. Therefore, compilation and testing can more easily be scripted and output more clearly analysed. It also handles importing libraries used in a Java project when compiling (which can be very troublesome when completed by hand), which is discussed below.
- Library import: The `pom.xml` can specify dependencies of the Java project. While custom, third party repositories exist, Maven has a central repository[6] with many libraries available. This includes log4j described above, and all other libraries used in this project. Dependencies are downloaded to the systems local Maven repository at compile time.
- Library export: As discussed in the introduction, the NLP system shall be used in a POC system. Rather than combining these two systems into one large package, or doing a confusing copy of the required resources, Maven can be used to export the compiled NLP system to the local Maven repository. Then, the POC system can simply list the NLP system as a dependency, and Maven shall include it as a library when building the executable program.

Maven is used as the management backbone throughout the development of software discussed in this report. When libraries are used in a project, a link to their dependency configuration for Maven's `pom.xml` shall be included. As a good example, log4j[7] has an extensive page providing a detailed description of how to import the library.

JUnit

JUnit is a popular Java framework for testing. It is simple to use, catching unexpected (or expected) exceptions and ensuring values are correct with `assert` statements.

Maven also integrates with it, so that (by default) when you build a Java project with Maven, all of the methods marked with `@Test` annotation in the test source directory are executed to ensure the program is working as expected (as far as the tests ensure that). It will then provide a report and trace of any issues once complete. Maven will also automatically exclude the test files from the final packaged product to reduce waste space for deployments of projects.

While working through this project many JUnit tests were constructed (all of which are still available in the Git repository for this project). Somewhat unconventionally, there is a divide between tests: while some are based around ensuring functionality works as expected, many are actually building the NLP systems, training them (if required), testing them and comparing the predictions made to the gold standard data.

The tests can also be ignored[8] which is very useful, as many of the tests written are base around

[5] https://maven.apache.org/
[6] http://repo.maven.apache.org/maven2/
[7] https://logging.apache.org/log4j/2.x/maven-artifacts.html
[8] http://maven.apache.org/surefire/maven-surefire-plugin/examples/skipping-tests.html

evaluating the algorithms created rather than testing functionality; so not only does not every algorithm need to be retested at every compilation time, but if they were it would take many hours (and probably more memory than the standard computer has) to build and test the application.

Word2Vec

The interesting Word2Vec technology is utilised in this project in various places. The original Word2Vec library implementation was in Python. However, the Deep Learning For Java (DL4J) team have included, as part of their machine learning and deep neural network library, Word2Vec funcitonality[9]. This supports training a Word2Vec model, using the model, and saving and loading models.

The models used in this project are the Google News model[10] and the Freebase model[11]. While neither of these are made up of scientific articles, they both have a large vocabulary size (3 million and 1.4 million tokens respectively), and both based off of a 100 GB large samples, which should allow them to perform relatively well.

Attempts were made to use a Wikipedia based model (Wiki2Vec[12]) but unfortunately no successful attempt was made to use it in this project (there were various problems converting the model to a Java readable format and loading it). While potentially of lower quality semantics (as Wikipedia isn't officially maintained) it may have had more of the vocabulary the ScienceIE data supports as Wikipedia covers many topics including those of scientific nature so could have increased coverage of the model when finding similarities between various scientific tokens.

3.1.2 Platform

While Java is cross platform (another excellent reason for using it), some of the underlying system libraries that Wor2Vec relies on to function are included by default in many Linux distributions. It can be made to work on Microsoft Windows operating systems, but it requires a large amount of complex configuration and generally not worth the pay off. Therefore, this system was built on the Ubuntu 16.04 distribution of Linux, as this involved the least amount of configuration to get working.

Furthermore, some of the algorithms created as part of this paper are able to fill up available memory on a computer very quickly. The memory available as part of this project was 16 GB. Linux swap space was configured (an *overflow* area for memory usage) but generally one would not like to use this, as it is slow to read from and will add some wear to the solid state drive in the host system available (due to many fast reads and writes) which isn't good. This is another reason for using Linux as the platform to build these systems on, as the memory overhead from the operating system is much smaller when compared with Microsoft Windows 10 (in the order of gigabytes of memory saved). Furthermore, Linux can also be run headless (without a GUI) to further reduce the operating systems memory usage, which on Ubuntu 16.04 saves approximately an extra gigabyte of memory. With support for Secure Shell (SSH) to remotely connect to the system, to run tests and read results, Linux is an excellent choice of platform for optimising memory usage while running these algorithms.

[9] https://deeplearning4j.org/word2vec.html
[10] https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing
[11] https://docs.google.com/file/d/0B7XkCwpI5KDYaDBDQm1tZGNDRHc/edit?usp=sharing
[12] https://github.com/idio/wiki2vec

Chapter 4

The ScienceIE Task: Design and Implementation

To complete the ScienceIE task, the plan was made to have one Java project containing three sub systems, where each of which could be called independently. As such, this section shall step through each subtask's design and implementation in order, beginning with a description of the preprocessing that was implemented, as it is generic to all subtasks.

After this section has finished describing the design and implenetation of the varous algorithms used in this project, the following section shall describe the results associated with them.

4.1 Data Preprocessing

To support processing in later systems, all data (development, training and test) had to be preprocessed. The idea of this piece of computation is to prepare the data for analysis, and to also reduce computation time (doing this process once for the entire system rather than once for each sub system). To further reduce experiment run time, Java serialisation was also used to save all the following preprocessing information for later retrieval.

In Java, for each paper file from ScienceIE, a `Paper` object was constructed. This held many important pieces of information about the paper in question, including location on disk, text extracted from its source file, and all preprocessing information. `Paper` itself is a *plain old Java object*, only holding information and is an abstract class, with `TextPaper` and `PDFPaper` classes extending from it which could be instantiated. These extended classes inherited the data storage features and utilities from `Paper`, but their constructors are customised to extract information from their given type of file:

- `TextPaper` is for `.txt` files and simply extracts the text from the document. It sees the title of the text document as the title of the paper.
- `PDFPaper` is for `.pdf` files. This uses Apache PDFBox[1] (imported through Maven[2]) to extract the text from a PDF. The title, once again, is the title of the document. As alluded to earlier, with the ScienceIE test set not only being just text files but also being short documents, longer PDF papers was not usually used, so little development to properly sanitise PDFs happened, meaning all titles and references were also captured in this text extraction. If more PDF files were to be processed this would have been looked at, however, due to its lack of use the time needed to fix this was deemed not worth it.
- It was initially planned that there would be a *HTML* and *WebPDF* classes although, for similar reasons to why the `PDFPaper` text extraction was not developed further, these two classes were never implemented. The main reason for wanting them was to later support the POC system, as this would allow that system to dynamically grab papers from the web and add them to itself. Importing of papers to the POC system shall be discussed later at a more relevant time.

The bulk of the preprocessing came in the form of using a parser to calculate the parse tree of a text. As discussed, many teams at ScienceIE used spaCy. As the plan for this project was to complete

[1] https://pdfbox.apache.org/
[2] https://pdfbox.apache.org/2.0/dependencies.html

it in Java (creating a single, self contained system) the Stanford CoreNLP package was used (Manning, Surdeanu, Bauer, Finkel, Bethard & McClosky 2014) (imported through Maven[3]). While offering a range of useful NLP features, the main ones utilised by the project were tokenization and finding the parse tree of the text (which naturally included POS tagging). An `Annotator` class was constructed which accepted a `Paper` input and annotated the text contained using the CoreNLP library.

Further processing on this information was also completed, where (at the time of saving the CoreNLP parse information) a token *map* was created. This *map*'s key set was all tokens present in the document, with the associated value being the number of times the token was in the document. This was to help when calculating TF-IDF scores later in processing.

The final part of preprocessing was to load existing annotation information. Of course this was only possible for ScienceIE data, which were all supplied with the relevant `.ann` files in BRAT format. These records were loading into a list of `Extraction` abstract entities, where each entry to the list could be either of a *KeyPhrase* or *Relationship* extending type, which each held all the information supplied in the annotation files (including classifications, the types of relations and more).

4.2 Subtask A - Key Phrase Extraction

Subtask A at ScienceIE was considered the hardest, reinforced by both the maximum and average scores for each independent subtask. This paper dedicated most of its NLP effort to this task out of the three subtasks as this is currently the hardest part of information extraction (out of the given subtasks) under current research.

Two attempts at this subtask were made. Initially, a *safer* design involved a SVM which considers some of the key features about key phrases suggested in the literature around this topic. Then, an even more experimental trail shall be described which involves clustering based around Word2Vec similarities between words in a document.

4.2.1 Method 1: Support Vector Machine

Inspired by the highest success at ScienceIE, a SVM approach was adopted to attempt to provide a solution to subtask A. Initially, a small set of support vectors were selected and tested, with more being added as research continued.

Processing Data

Two approaches were considered when designing the input and output data. One was based around passing each token in individually and in order, while the other was based around using the parse information obtained by using CoreNLP to pass sections of a sentence.

Working with each individual token was selected for several reasons. Firstly, it was very easy to simply iterate through every token in a document in turn. Furthermore, the CoreNLP data is still available (evidences as that is what returns the tokens of the document) and can be passed to the SVM to be used when calculating support vectors. While using sections of a sentence should help keep any key phrase extracted more semantically correct (i.e. it should avoid missing the end of a noun phrase by accident which a check could be added for anyway), it poses a large issue: Any section selected as a key phrase would likely be *locked down* as such to the specific tokens inside that section, meaning there may be no way to get rid of excess information or added extra if the gold standard key phrase requires something slightly different to the key phrase chosen by the SVM. In terms of extra information needed, a system could be implemented to join adjacent key phrases but that would like see extra information over what is needed being included. If, to try and solve this issue, some system which could extend or retract by a token or two was implemented, it is getting closer to the original option anyway where the system is processing the entire document as individual phrases. Therefore, a system based around processing each token individually was decided upon.

This resulted in a total of 65447 different training points (the total number of individual tokens in all of the training data).

[3]https://stanfordnlp.github.io/CoreNLP/download.html

Support Vector Description	Value Range
The length of the token divided by the maximum token length in the training set.	$\texttt{svLen} \in \mathbb{R}$, $0 \leq \texttt{svLen} \leq 1$
Whether the token is a noun (using Part-Of-Speech tagging).	$\texttt{svPos} \in \{0, 1\}$
The TF-IDF score of the token.	$\texttt{svTfIdf} \in \mathbb{R}$, $0 \leq \texttt{svTfIdf} \leq 1$
The token index divided by the number of tokens.	$\texttt{svDepth} \in \mathbb{R}$, $0 \leq \texttt{svDepth} \leq 1$
The token index in the current sentence divided by the number of tokens in the sentence.	$\texttt{svDepthSentence} \in \mathbb{R}$, $0 \leq \texttt{svDepthSentence} \leq 1$
Whether the token is in the first sentence of the paper.	$\texttt{svFS} \in \{0, 1\}$
Whether the token is in the last sentence of the paper.	$\texttt{svLS} \in \{0, 1\}$
Whether the previous token was part of a key phrase.	$\texttt{svLWKP} \in \{0, 1\}$

Table 4.1: Initial key phrase support vector features used. A set of these features is generated for each token. When defining the value range, the variable is named as it is in the Java code.

Defining Support Vectors

It is clear that current trends view the position of key phrases are very important in the document and should definitely be considered when trying to learn how to predict them. A tokens proximity to other tokens semantically and as part of the document as whole seem to significantly help us identify where key phrases lie. Furthermore, some attributes about individual phrases also seem to play a large part. For example, the length of the word is a valid feature to evaluate, as the average length of a key phrase token (7 characters) is slightly different to the average of all key phrases (8 characters).

Thankfully, the idea behind using an SVM is to find what separates key phrases from just normal phrases. Therefore, I was able to create an initial range of features, as defined in table 4.1. Here it is evident most of the features are based around trying to gather information as to the whereabouts of the token. It also, importantly, considers the sequence of key tokens.

Training

To train the SVM, a *problem* must be created. The *problem* contains an array of vectors, where the vector is the set of features described above. Each of these data points must be labelled. The label is what we are trying to predict on the test data, so here the label is where or not the token is a key phrase (`0` for *normal*, or `1` for key phrase).

Model Selection

As the nature of the data is unknown, an educated guess can be made as to which kernel to use. A common kernel to begin working with is the *Radial Basis Function* (RBF) kernel (Chih-Wei Hsu, Chih-Chung Chang & Lin 2008). This is because it can handle non-linear data, which it is assumed the training data here is to be. The RGF kernel function to find the similarity between two data points is listed below:

$$K_{\text{RBF}}(x_i, x_j) = exp(-\gamma||x_i - x_j||^2)$$

There are two parameters which can be configured and tuned to optimise performance of the SVM:

- The cost C parameter. This influences the misclassification allowance, where a small value lets the SVM select a large hyper-plane for separating data but allows for more misclassification, and a large value will allow the SVM to attempt to find a smaller hyper-plane that has less misclassification. Several values will be explored, of the set $\{5, 50, 100, 200\}$.
- The RBF kernel has a single parameter γ. From the same source that recommended the RBF kernel, as a initial value the SVM shall be configured to 0.5, as this is 1 divided by the number of

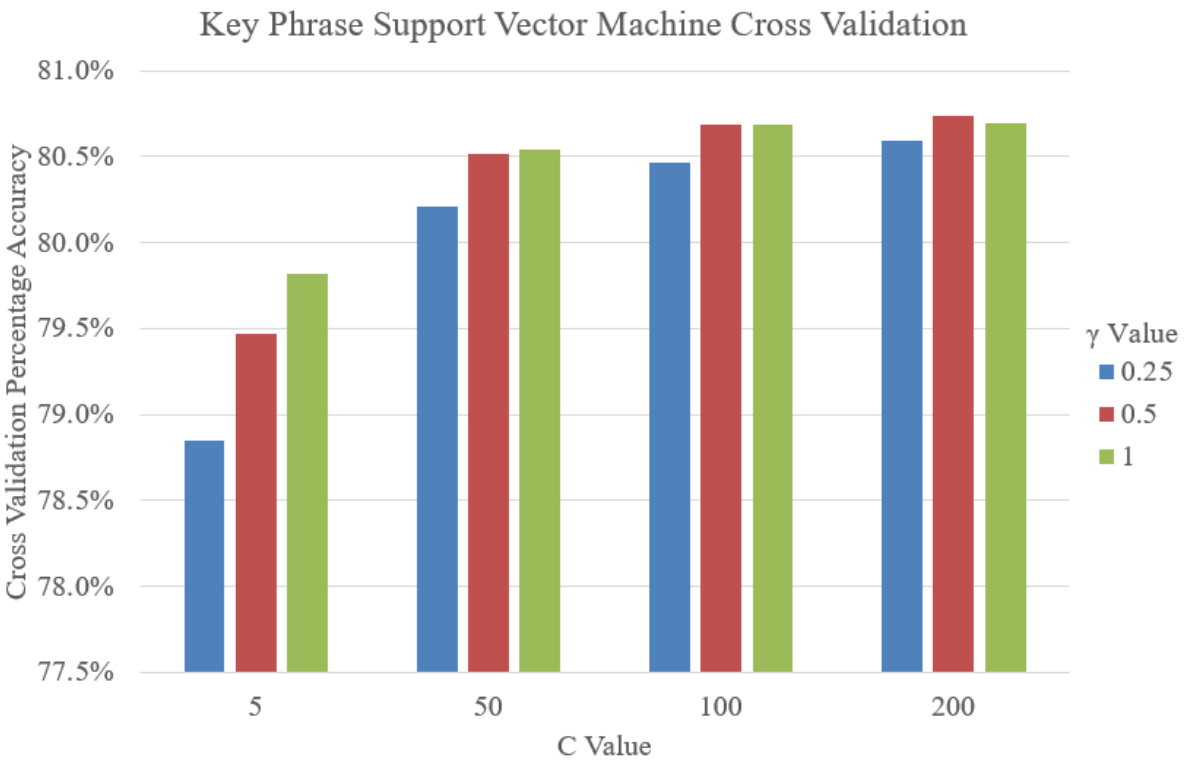

Figure 4.1: The cross validation results on the key phrase SVM.

features (we have 2 labels). However, other values shall be explored to attempt to find the best accuracy, and these values shall be {0.25, 0.5, 1}.

Development

Having decided on how to use the concept of an SVM, there was a need for a concrete implementation. The idea of implementing an SVM was considered, however, with a responsibly high implementation complexity and a high risk of getting something subtly wrong (therefore being hard to detect and fix) a pre-existing solution was searched for. Furthermore, with the author having never handled an SVM before, a pre-existing solution with additional usage information was desired.

A popular SVM package was found in libsvm (imported through Maven[4]). Originally written in C and ported over to Java (as well as many other languages), libsvm was designed to be flexible, supporting various kernels and suitable for beginners through to advanced users. It has support for the core use of SVMs - training and predicting, but also has features to aid in parameter selection such as a cross-validation function (which will be covered in more detail shortly), a visualiser for the training data and a data scaling tool.

Cross Validation

Cross validation is an important part when trying to optimise performance of an SVM. It allows for tuning key parameters by running repeated tests. Rather than using the testing data, which could introduce bias, the training data is split up into n folds (or groups of data from within the training set). $n = 5$ folds were used in this instance. In turn, the SVM is trained with 4 of the 5 folds and then evaluated against the remaining fold. This is repeated for all combinations of folds and then the accuracy of the SVM can be calculated. A higher accuracy should mean better performance, although there is the problem of over fitting to consider. If the model is built to run perfectly on the training data, real world performance may actually suffer. This is why we cannot stop testing the SVM after just cross validation, as evaluating against the unseen test set will tell us how well it really performs.

[4]https://mvnrepository.com/artifact/com.datumbox/libsvm/3.22

The values discussed for C and γ were used in cross validation and their outputs compared. The cross validation results can be seen in figure 4.1. As show by this chart, very little change is accuracy is observed. There is a upward trend as C and γ increase, but this appears to plateau as higher values are evaluated. No smaller values need testing as (given the trend continues) the accuracy will significantly diminish. In terms of increasing the values, there is potential for extremely small gains; however, the return from increasing these values will be almost worthless and the training time required as the C value increase significantly rises, as the SVM is working harder to find a better fitting hyperplane. Therefore, C = 100 and $\gamma = 0.5$ shall be used when completing full experiments. While, a C value of 200 is 0.05% more accurate with the same γ value, the training time is roughly doubled (from some hours to many hours) and the reward is not deemed worth it.

4.2.2 Method 2: Clustering

Concept

Clustering has been shown to be effective in key phrase and other information extraction. A seemingly effective method was using *term relatedness* to group terms, and then find an exemplar term at the centre of the clusters which can be used as a key phrase (Liu et al. 2009).

Inspired by this, this paper proposes a similar process, where the similarity of terms is based around their Word2Vec similarities. Conceptually, it is possible that gathering similar terms will have the effect of creating clusters of important concepts, from which key phrases can be extracted. With the most similar words at the centre, these could be considered key words, and the phrase around these words could be extracted from the document. This should also exclude unimportant or stop words, as these may have large distances to the key concepts.

An important note here is that the document is treated as a *bag-of-words*, where we ignore the semantic meaning of the words. Word2Vec does this anyway, as when using the library the word is simply passed as plain text with no extra information, and Word2Vec uses its own ideas about the words semantic meanings to evaluate it. Using a bag-of-words approach means words from any part of the document could be clustered together, which is ok as if both are close to the centre of a cluster, it may mean both of them are key phrase worthy and should be selected (a full phrase from their origin extracted to be a key phrase).

The clustering algorithm selected was hierarchical clustering (Rai & Singh 2010). Bottom up (agglomerative) hierarchical clustering works as follows:

1. Each element begins in their own cluster (so n elements means initially there are n clusters).
2. Given some distance metric, the distances between all clusters are calculated.
3. The closest two clusters are combined.
4. This process is repeated until a single cluster is left.

This algorithm was chosen for several reasons:

- Firstly, it is a relatively straight forward clustering algorithm to implement, so should allow results to be found quickly.
- While the term similarities are based off of Word2Vec similarities, the distances between clusters could be evaluated in a number of ways. These include *single* (the shortest distance between terms in each cluster), *average* (the average distance between each term in one cluster to each term in another) and *complete* (the largest distance between terms n each cluster). These are called the *linkage criteria*.
- If successful, the benefits of using clustering could be two fold. Not only might this produce effective key phrases, but it may even aid in classification. Once a number of clusters have formed key phrases, the hierarchical clustering could continue potentially all the way to producing just a few clusters where each could be classified into one of our 3 target classes. This will only be successful if the key phrase extraction is successful. If doing this clustering was purely for classification, k-means clustering may be more appropriate with a $k = 3$ where each cluster would be a different classification.

When working with hierarchical clustering, an important aspect to consider is how far to iterate through the algorithm. In theory, the algorithm should run until there is just one cluster left with every element in it - but this is not useful for anything. The more useful cluster states will be part of the way along the iterative cycle. To find this *sweet spot* will require manual tuning, via inspecting the progression of the clusters to try to identify a good range where key phrase information can be extracted. In theory finding the sweet spot could be automated and learnt (by trying to extract key phrases at all levels and evaluating against the gold standard phrases to see how well the algorithm does) but to fully develop this kind of system would require a lot of time, which itself is very expensive, and if a failure it would be a big waste of time.

A large difference between this and SVM usage is that this method is unsupervised learning and does not require training data, while using an SVM is supervised learning. This means that, given this method works for this testing scenario, its application may scale better in the *real world* as is it not tied to the quality of the training data. Furthermore, given a suitable Word2Vec model, differences in key phrase output may be seen. This means that this algorithm may suffer here given the Word2Vec models currently available are not based on scientific publications, but running this algorithm on test data with similar context to the Word2Vec model may improve things, which means it may even cope with different languages.

Development

The process of turning this theory into a practical implementation is straight forward. To allow for future expansion if clustering was to be used again for anything, a generic abstract `Cluster` class was created, which was formed of a list (where type is specified at creation time) of items and declaration of functions to find the distance between a given cluster object and another cluster, and to create a new cluster by combining a given cluster object with another.

A `Linkage` enumeration was created which could be used when testing to specify the method for finding the distance between two clusters.

Actually commencing the clustering begins by splitting the document into all of its tokens, and removing duplicates. Then, stop words and unimportant words are removed. Unimportant words are classified based on their TF-IDF scores. All words were sorted according to their TF-IDF score, and the bottom 15% were removed. This percentage was manually set, after generating a list of all tokens and their TF-IDF scores and evaluating where the cut off should be to remove all words that are not in any key phrase and that are not very interesting words. Some care needed to be taken, as for example, "results" has a low TF-IDF score of 0.006, but is in the key phrase "generalization of these results to the NSI case". Of course, that phrase also includes other words like "of", "the" and "to" which would be removed, but when we have some key words we could then reintroduce the semantic information and try to form well formed snippets, and as "results" would connect the first and second half of the sentence to help form the full phrase it is important to not get rid of all unimportant words. Therefore, 15% was found to be a reasonable compromise to remove the particularly low TF-IDF valued tokens and to leave the rest of the some what to very interesting tokens in the bag-of-words.

Then, the process of hierarchical clustering happens as described above. This happens for all test data, with all three linkage methods, with results being saved to disk. The output is then evaluated on a sample of the processed documents (as manually reviewing all 100 test documents cluster patterns across 3 different linkage criteria is too much for a single reviewer to achieve) to try to evaluate where key phrases can be extracted from.

4.3 Subtask B - Key Phrase Classification

4.3.1 Word2Vec Usage

With a large amount of influence coming from Word2Vec, the decision was made to focus on exploiting this technology to try to classify key phrases.

As the whole concept of Word2Vec is word similarities based on their semantic meaning, the idea is proposed that simply examining the distance from a key phrase to the relevant classification term or similar may result in a reasonably accurate classification.

An important thing to consider is that Word2Vec generally accepts individual tokens. This contrasts to what we are trying to classify which is a string of tokens. Therefore, a way to find the similarity between a string of tokens and a singular token must be devised.

Target Class	Similar Word Set to use when Testing
Task	Task, Application, Goal, Problem
Process	Process, Model, Algorithm
Material	Material, resource

Table 4.2: The target classes and the set of words that are of a similar domain to be used in testing of the Word2Vec classifier.

Two methods for achieving a distance metric are proposed and shall be tested:

- Average distance: each token in the string of tokens are compared to the target token and their distances are averaged,
- Shortest distance: each token in the string of tokens are compared to the target token the shortest distance (highest value in terms of *similarity*) is used.

There is also the problem of token importance - as it is like not all tokens will aid in classifying a key phrase. For example, "determine the lowest energy configuration" has the word "the" as a token. This token is bad as it can be used in a key phrase with any classification, and likely not make any difference. Therefore, it is theorised that stop words and very low TF-IDF words being removed could help improve classification results. There is a potential downside to this, as some key phrases may appear as only containing stop words. "He" for example is a key phrase and is a synonym of "helium", but without that relation knowledge (which at this stage of the competition we in theory doesn't have) it would be seen as a stop word and removed. In this case it would actually nullify the key phrase so should be left and classification attempted anyway, but this is why this algorithm should be tested with and without unimportant words being removed, as not all of them may actually be unimportant.

Finally, there is the question of what token to find similarity too. A simple way forward is comparing the tokens to the word identifying the class we're trying to find out about, so simply testing for similarity to "test", "process" and "material". However, if the quality of the Word2Vec model isn't very good (it could have an incorrect idea about what a *task* is) the classification may be of poor quality as well. To try to solve this problem, more words based around the same concept should be tested against to try to avoid the problem of a single word being in the incorrect space in Word2Vec vector space. Therefore, when testing this algorithm the similarities will be based on the word of the defining class ("task", "process", "material") on one set of runs, and on the other set of runs the maximum similarity to words related to that of the class shall be found. The words that shall be used are defined in table 4.2. The original word for the class shall still be evaluated when finding the maximum similarity from a set of similar words.

4.3.2 The Word2Vec Vocabulary Problem

An anticipated problem, proven through early testing, was that not every token in the scientific papers supplied by ScienceIE appeared in the Word2Vec models. With no obvious way around this issue, the unclassifiable tokens in the ScienceIE test set were checked to see their original classification, the result of which can be seen in figure 4.2. While many key phrases are included in this set that cannot be directly classified by Word2Vec due to the vocabulary problem, almost two thirds of these key phrases are *material* key phrases. Therefore, a default class of *material* will be assigned.

4.3.3 Development

As the design of this classifier is only concerned about classifying one key phrase at a time, and Word2Vec doesn't need (and can't take) more information about a word at a time, the only piece of information needed to run the classifier is the key phrase string. Therefore, only one public method needs to be exposed as part of the library to do this classificatiion, requiring the key phrase string and the Word2Vec model. The class implemented has no need to be stateful (as the method simply takes all it needs as parameters) so the implemented `W2VClassifier` cannot be instantiated and contains only `static` methods. For testing purposes, extra parameters are able to be passed to configure items like whether to use the single or multiple words for find distance to a classification, and technically two methods are exposed to the word as one is for closest distance and one is for average distance.

Upon calling the method, the key phrase string is split into its individual tokens. These tokens are then individually checked, firstly to see if they are a stop word or of extremely low importance, and then to

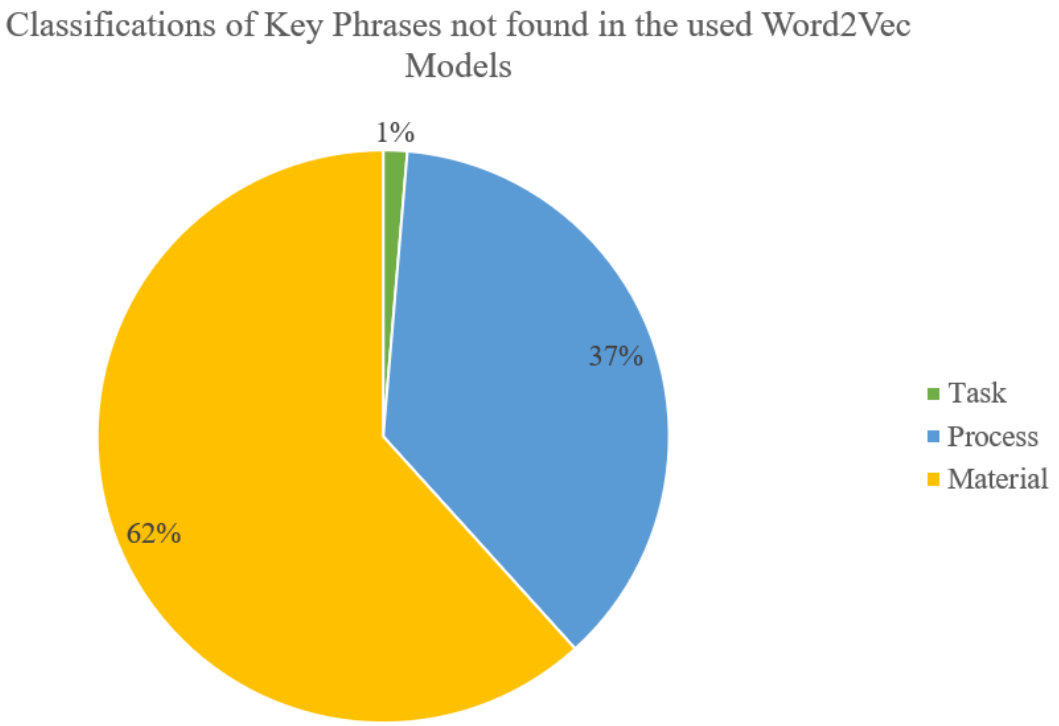

Figure 4.2: Gold standard key phrase classifications for key phrases that are not in the Word2Vec models used. 222 key phrases are included in this set.

ensure they are actually in the Word2Vec model (with the convenient `.hasWord(word)` method). Given a token passes both of these checks, its similarity to the target word (or words) for each classification is found and either accumulated (if based on average distance) where the sum value is divided by the number of checked tokens at the end, or compared to the current maximum similarity found (if based on closest) and selected if larger. To find this similarity, Word2Vec has a `.similarity(word2, word2)` method, which returns the cosine similarity of the two words in vector space (where the value must be between 0 and 1). There is always one *tracker* value (average accumulator or maximum similarity) per classification which is updated every time a token is processed.

Upon finishing checking each token, the *tracker* values are compared and the largest one selected, resulting in the classifiication this value is related to be returned (rather than handling strings, there is a `Classification` enumeration class to handle this functionality better). If all of the *tracker* values are 0, it means no token in the key phrase passed the initial checks; meaning we cannot classify the key phrase with Word2Vec and therefore must return a default value as discussed above.

4.4 Subtask C - Relation Extraction

4.4.1 Word2Vec Usage

Having considered Word2Vec's similarity function in the previous subtask, it shall again be utilised for this subtask - relation extraction. The goal here is to find relationships between key phrases, that can be categorised as hyponym (*is-a*) and synonym (*same-as*) relationships.

To achieve this, we consider the relative distances tokens in a Word2Vec vector space hold to each other. In theory, if words A and B, of the same type of nature (for example cities), are connected in the same way (in natural language) to words C and D, of the same type of nature (for example countries), respectively, the relative distance between A and C should be very similar to that of B and D. It should also stand that the relative distance A to B is very similar to C to D.

Therefore, the experiment here is to test if pairs of items that are synonyms share approximately the same relative distances between each other, and the same for hyponyms. The hope is to find a way to match synonym distances and hyponym distances.

To achieve this, as usage of support vector machine was leveraged in subtask A, it shall again be used here to construct a system that can interpret the relative distance information produced by Word2Vec for given pairs of key phrases. Two proposed ideas for feature sets based off of the Word2Vec vector information, and their development, are presented below.

4.4.2 SVM - Many Features

The first attempt at constructing a SVM to interpret the meaning of Word2Vec distances shall be based on the literal vector space difference. The vector spaces in the model have dimension sizes of 300 and 1000 for Google News and Freebase respectively. As 1000 is extremely large, and Google News has a larger vocabulary than the Freebase model, this experiment will only work with the Google News Word2Vec model. Therefore each item added to the support vector machine shall have 300 features.

The features shall be the difference between two key phrases' vectors, taken from Word2Vec. Word2Vec has a method, `.getWordVector(word)`, which returns the vector of a given word in Word2Vec space. With two vectors representing two key phrases, one vector can be subtracted from the other, and this is a data point.

Of course, as described in subtask B, Word2Vec cannot take a string of words, so instead we again tokenize the key phrase and process each token individually, summing their vectors as each is processed. Again, it is possible that some tokens in a key phrase are not significant to a phrase when trying to mathematically represent a word, so three variants on calculating the final vector of a key phrase are proposed:

- Simply summing the Word2Vec vector representation of all tokens in a key phrase,
- Removing stop and unimportant words and summing the vectors of the remaining tokens (if the key phrase is only unimportant words as mentioned in subtask B, the algorithm reverts to summing the vectors of all the tokens anyway so the result isn't null),
- Using the parse information about the phrase to attempt to extract the root noun from it. Not all key phrases have nouns, and when this happens all tokens are summed together, but as the overwhelming majority of them do have a noun, selecting just this to find relations between seems sensible as this is the key part of the key phrase, and the part that would make it a synonym or hyponym.

To generate the training data, all key phrases were paired with all other key phrases (within their papers). The vector differences were all calculated, and the key phrase pairs that were hyponym or synonym relations labelled. Rather than one SVM that handled multiple labels, two SVMs were trained, one with the hyponyms labelled and the other with synonyms labelled. The thought behind this was that it would be simpler to manage training and testing as hyponyms and synonyms could be dealt with separately. As with the key phrase SVM, the RBF kernel was once again used.

Cross validation was applied to both SVMs, however, the range of values tested did not seem much change. The top row of figure 4.3 show the cross validation results for this set of SVMs, and do not really indicate any trend. For both hyponym and synonym of extraction, $C = 5$ and $\gamma = 1$ appeared to have slightly higher accuracy, but overall very little difference can be seen. The reason the accuracy is extremely high (all results are less than 0.5% from 100%) is because of the low amount of *true case* relationship training points compared to the whole data set: 674 total relations out of 144410 possible combinations. This may prove problematic when testing on real world data, as the SVM effectively has a lack of good training data to learn off of (so may struggle to fit a good separating hyperplane).

This *many features* SVM was implemented in `RelationshipSVM` in the Java project accompanying this report.

4.4.3 SVM - Few Features

Mentioned above, as the data was sparsely labelled and each point had a high feature count in the *many features* SVM approach, this *few features* approach aimed to reduce the complexity of the data presented

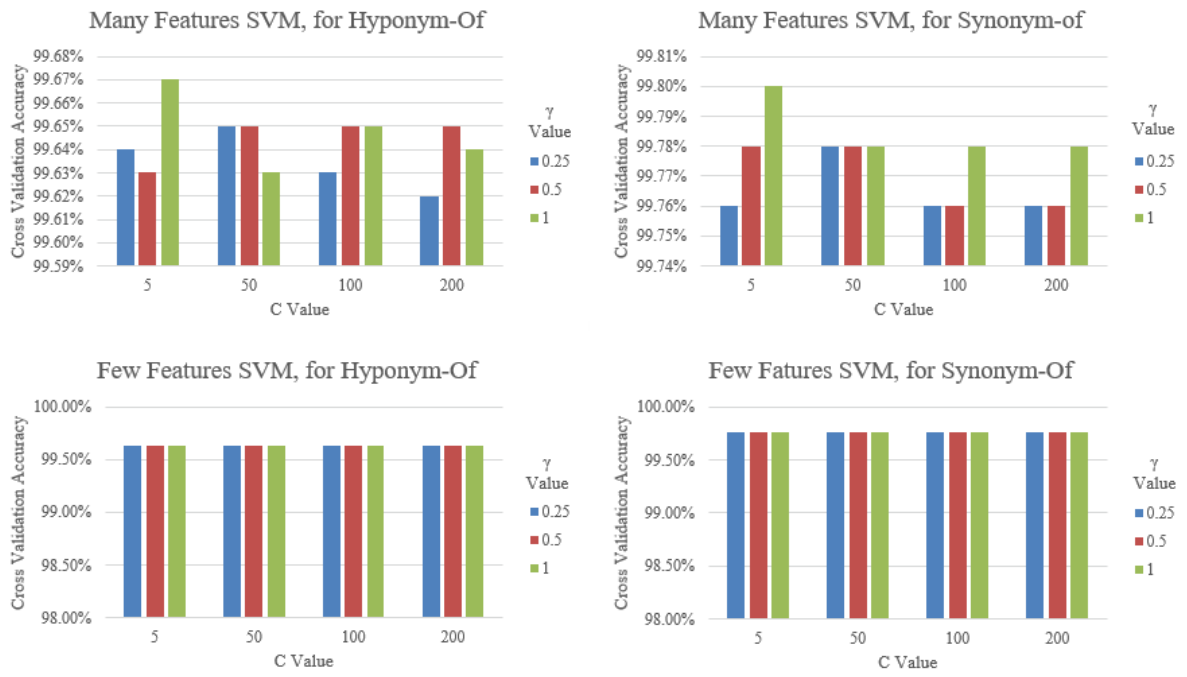

Figure 4.3: The cross validation results on both the hyponym and synonym variants of the *many features* and *few features* relationship SVMs.

in a hope it would allow for easier hyperplane fitting, meaning significantly reduced training times and hopefully a higher accuracy rate.

The idea was to reduce the vectors generated by Word2Vec to retain much of the same information in a simpler format. The result of this idea was to generate, given two key phrase vectors, the angle and the distance between the two vectors. The reason for these two metric is because these should preserve relative differences between vectors, as it is still based on the change in vector space between them.

To achieve this, the algorithm for generating a data point initially generates a vector sum of the tokens in each key phrase. With the two vectors prepared, we then calculate the following:

- Euclidean distance between the two vectors:

$$d(\overrightarrow{u}, \overrightarrow{v}) = ||\overrightarrow{u} - \overrightarrow{v}|| = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2 ... + (u_{300} - v_{300})^2}$$

- The angle between the two vectors based on the dot product:

$$cos(\theta) = \frac{\overrightarrow{u} \cdot \overrightarrow{u}}{||\overrightarrow{v}|| \cdot ||\overrightarrow{v}||}$$

These two values are then used as the features of each data point. A point is generated per pair of key phrases in a document, and labelled as to whether or not they are a label. As with the *many features* version, two SVMs were trained, one for hyponyms and one for synonyms. The RBF kernel was used in these SVMs.

Unfortunately, completing cross validation for this *few features* SVM approach was not fruitful. The result can be seen in figure 4.3, with the relevant charts below that of the *many features* cross validation results. Across the board for both SVMs, there is no change is accuracy with the different values tested. This likely means that no selection of relations is happening (the high accuracy is again because most of the data points should not be labelled as relationships), which does not bode well for real world tests. Given no trend has happened, it is hard to evaluate how to try to fix this issue. Theoretically, increasing the C value may allow for the SVM to find a better hyperplane which may help, but with when $C = 200$, the training time is many hours and the computer this is running on (equipped with 16 GB of memory) does not have enough memory to train further than that. Increasing C this further simply used many gigabytes of Linux swap space on the actual disk which slows it further, and if Java is not configured to be allowed to use that much memory, it can interrupt testing as well.

This *few features* SVM was implemented in `RelationshipSVM2` in the Java project accompanying this report.

Chapter 5

The ScienceIE Task: Evaluation

With solution systems proposed for the various subtasks of ScienceIE, each were given data to train on and predict on. Below are results where the algorithms have been tested end-to-end (where they are run through in order, with the data from one being passed to the next) and independently (where they are given the gold data as a starting point, and operate on that information). There is also some self evaluation conducted, to explore the effectiveness of evaluating under conditions of varying strictness.

5.1 Subtask A - Key Phrase Extraction

5.1.1 Method 1: Support Vector Machine

Self Evaluation

5.1.2 Method 2: Clustering

5.1.3 Conclusion

5.2 Subtask B - Key Phrase Classification

Key phrase classification under the outlined Word2Vec classifier was quick to execute. After the initial, one time per system boot, wait for the Word2Vec model to be loaded into memory, the actual calculations required are very fast to execute, giving this classifier a very high throughput.

In terms of result, it can be seen that several configurations allow for over 50% of the key phrases to be classified correctly. The full range of results for each configuration can be seen on table 5.1.

The obvious thing to note here is that all tests under Freebase without a default class and some Google News tests without default classifications have a 0% accuracy (i.e. it didn't classify any key phrase correctly). With a little investigation, if all key phrases were labelled as a *material*, the accuracy would be 44% - which is what was achieved by all configurations with the Freebase model and a default classification of *material*. Therefore, Freebase clearly isn't an effective vocabulary for working with scientific publications.

For Google News, things are less clear. In some cases where the distance metric was finding the *closest* token to the target classification, the results mirror that of when using Freebase. Generally results were better when unimportant and stop words were removed, however, the best overall left all stops words in.

Furthermore, using various words when finding similarity generally decreased the accuracy of classification. When using *average* similarity, results were averagely 2% higher (on runs that got higher than the base 44%) when compared to the same configuration bar using multiple words to gauge similarity.

Google News does show some promise for good classification. In some cases where the default classification does not have a null, it can still achieve relatively high results.

5.2.1 Other Experimentation

As a small experiment, given the SVM for subtask A had undergone quite some development...

Word2Vec Model	Distance Metric	Default Class	Remove words?	Use many words?	Accuracy
Google News	Average	Unknown	No	No	**47.90%**
Google News	Average	Unknown	Yes	No	**47.76%**
Google News	Average	Unknown	No	Yes	**45.47%**
Google News	Average	Unknown	Yes	Yes	**45.47%**
Google News	**Average**	**Material**	**No**	**No**	**54.58%**
Google News	Average	Material	Yes	No	**54.43%**
Google News	Average	Material	No	Yes	**52.14%**
Google News	Average	Material	Yes	Yes	**52.14%**
Google News	Closest	Unknown	No	No	0.00%
Google News	Closest	Unknown	Yes	No	**45.96%**
Google News	Closest	Unknown	No	Yes	0.00%
Google News	Closest	Unknown	Yes	Yes	43.91%
Google News	Closest	Material	No	No	44.05%
Google News	Closest	Material	Yes	No	**52.63%**
Google News	Closest	Material	No	Yes	44.05%
Google News	Closest	Material	Yes	Yes	**50.58%**
Freebase	N/A	Unknown	N/A	N/A	0.00%
Freebase	N/A	Material	N/A	N/A	44.05%

Table 5.1: The above table is the various configurations of the Word2Vec classifier, running with every possible configuration for the five parameters are listed. The result in bold line is the highest scoring configuration, with bold results being notable results. All Freebase results were not listed as there was no change in result other than for the *default class* variable, so *N/A* is present instead of each iteration of those variables. These results are based on classifying all 2052 ScienceIE key phrase test data points.

5.2.2 Conclusion

5.3 Subtask C - Relation Extraction

5.3.1 Conclusion

Chapter 6

Creating a Proof of Concept Use for ScienceIE data

Having completed systems to handle the information extraction, the next major part of the project was to explore using this information in a generally useful way.

6.1 Further Research

Discuss the resources used to design maybe? Make sure to include research on searching I did...

6.2 Design and Implementation

How it was pulled off

6.3 Web Interface

Exactly what was achieved

6.4 Testing

(Get) user feedback

6.5 Conclusion

Overall impact of the GUI on the project

Chapter 7

Discussion

Talk about overall results

7.1 Improvements and Extensions

Bibliography

Ammar, W., Peters, M., Bhagavatula, C. & Power, R. (2017), 'The AI2 system at SemEval-2017 Task 10 (ScienceIE): semi-supervised end-to-end entity and relation extraction', *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)* **10**, 592–596.
URL: *http://www.aclweb.org/anthology/S17-2097*

Augenstein, I., Das, M., Riedel, S., Vikraman, L. & McCallum, A. (2017), 'SemEval 2017 Task 10: ScienceIE - Extracting Keyphrases and Relations from Scientific Publications', pp. 546–555.
URL: *http://arxiv.org/abs/1704.02853*

Chih-Wei Hsu, Chih-Chung Chang & Lin, C.-J. (2008), 'A Practical Guide to Support Vector Classification', *BJU international* **101**(1), 1396–400.
URL: *http://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.pdf*

Fawcett, T. (2006), 'An introduction to ROC analysis', *Pattern Recognition Letters* **27**(8), 861–874.

Hasan, K. S. & Ng, V. (2014), 'Automatic Keyphrase Extraction: A Survey of the State of the Art', *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* pp. 1262–1273.
URL: *http://aclweb.org/anthology/P14-1119*

Liu, Z., Li, P., Zheng, Y. & Sun, M. (2009), 'Clustering to Find Exemplar Terms for Keyphrase Extraction', *Language* **1**, 257–266.
URL: *http://portal.acm.org/citation.cfm?doid=1699510.1699544*

Manning, C. & Jurafsky, D. (2012), 'Using Patterns to Extract Relations'.
URL: *https://www.youtube.com/watch?v=VodeEgvxgtA*

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. & McClosky, D. (2014), 'The Stanford CoreNLP Natural Language Processing Toolkit', *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations* pp. 55–60.
URL: *http://aclweb.org/anthology/P14-5010*

Marsi, E., Sikdar, U. K., Marco, C., Barik, B. & Sætre, R. (2017), 'NTNU-1$@$ScienceIE at SemEval-2017 Task 10: Identifying and Labelling Keyphrases with Conditional Random Fields', *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)* pp. 937–940.
URL: *http://www.aclweb.org/anthology/S17-2162*

Rai, P. & Singh, S. (2010), 'A Survey of Clustering Techniques', *International Journal of Computer Applications* **7**(12), 1–5.
URL: *http://www.ijcaonline.org/volume7/number12/pxc3871808.pdf*

Ramos, J. (2003), 'Using TF-IDF to Determine Word Relevance in Document Queries', *Proceedings of the first instructional conference on machine learning* pp. 1–4.

Snow, R., Jurafsky, D. & Y. Ng, A. (2013), 'Learning syntactic patterns for automatic hypernym discovery', *Journal of the American Medical Informatics Association* **20**(1), 1–11.
URL: *http://dx.doi.org/10.1186/s12859-015-0606-0%5Cnhttp://dx.doi.org/10.1016/j.jbi.2015.02.004%5Cnhttp://d*

Zhang, C., Wang, H., Liu, Y., Wu, D., Liao, Y. & Wang, B. (2008), 'Automatic Keyword Extraction from Documents Using Conditional Random Fields', *Journal of Computational Information* **43**, 1169–1180.
URL: *http://www.jofci.org*

Appendix A

Example ScienceIE Training/Test Document

The following is ScienceIE test paper file S0010938X15301268.txt:

Fig. 9 displays the growth of two of the main corrosion products that develop or form on the surface of Cu40Zn with time, hydrozincite (Fig. 9a) and Cu2O (Fig. 9b). It should be remembered that both phases were present already from start of the exposure. The data is presented in absorbance units and allows comparisons to be made of the amounts of each species between the two Cu40Zn surfaces investigated, DP and HZ7. The tendency is very clear that the formation rates of both hydrozincite and cuprite are quite suppressed for Cu40Zn with preformed hydrozincite (HZ7) compared to the diamond polished surface (DP). In summary, without being able to consider the formation of simonkolleite, it can be concluded that an increased surface coverage of hydrozincite reduces the initial spreading ability of the NaCl-containing droplets and thereby lowers the overall formation rate of hydrozincite and cuprite.

Appendix B

Example ScienceIE Training/Test Annotation Data

The following is ScienceIE test paper annotations file S0010938X15301268.ann:

T1 Material 46 64 corrosion products T2 Material 104 110 Cu40Zn
T3 Material 122 134 hydrozincite
T4 Material 149 153 Cu2O
T5 Material 378 384 Cu40Zn
T6 Material 408 410 DP
T7 Material 415 418 HZ7
T8 Material 530 536 Cu40Zn
T9 Material 552 564 hydrozincite
T10 Material 566 569 HZ7
* Synonym-of T9 T10
T11 Material 587 611 diamond polished surface
T12 Material 613 615 DP
* Synonym-of T11 T12
T13 Material 678 691 simonkolleite
T14 Material 751 763 hydrozincite
T15 Material 809 833 NaCl-containing droplets
T16 Material 883 895 hydrozincite
T17 Material 900 907 cuprite
T18 Process 456 471 formation rates
T20 Process 280 296 absorbance units
T19 Task 308 406 comparisons to be made of the amounts of each species between the two Cu40Zn surfaces investigated
R1 Hyponym-of Arg1:T3 Arg2:T1
R2 Hyponym-of Arg1:T4 Arg2:T1
T21 Material 480 492 hydrozincite
T22 Material 497 504 cuprite
T23 Process 665 691 formation of simonkolleite
T24 Process 776 793 initial spreading
T25 Process 865 879 formation rate