

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 content of this dissertation has been produced by the author as part of the BSc Computer Science at the University of Birmingham. None of the material has previously been submitted for a degree at the University of Birmingham or any other university. The research conducted has been by the author unless indicated otherwise.

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 its 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. Clustering based off of the Word2Vec distances between words was also experimented with, but was not able to produce satisfactory results. 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 to detect relations.

The best solutions found in this paper for subtasks A, B and C, under the ScienceIE script evaluation, saw F1 scores of 0.2, 0.11 and 0.02 in end-to-end tests and scores of 0.2, 0.55 and 0.1 when tested individually, respectively.

To explore how this system could be used, a proof-of-concept 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). The search was the main feature developed, which aimed to use the key phrase information and the phrases' tokens' TF-IDF scores to help prioritise the results more relevant to the user if they use key words from the target papers. A reasonable solution was found for this, although further testing with a larger range of papers would confirm its usefulness. Two data visualisations, a donut chart and a set of word clouds, were also implemented to display snapshots of the information extracted by the earlier systems.

All code and resources produced and used throughout the production of this project are available at:

<https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2017/tbc452>

Appendix E gives instructions on how to run the program. Similar instructions are also available in the README.md file at the root of the Git repository listed above.

Keywords

Natural Language Processing, Information Extraction, Classification, Relation Extraction, Support Vector Machine, Word2Vec, Java, Spring Boot

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 while working.

Contents

1	Introduction	5
1.1	Aims and Objectives	5
1.2	Report Outline	6
2	Background and Literature Review	7
2.1	Definitions and Descriptions	7
2.2	ScienceIE Proceedings	8
2.3	Further Background	9
2.4	Word2Vec	10
2.5	The Supplied Data Set	10
3	Project Architecture	12
3.1	Language	12
3.2	Platform	14
4	The ScienceIE Task: Design and Implementation	15
4.1	Data Preprocessing	15
4.2	Subtask A - Key Phrase Extraction	16
4.2.1	Method 1: Support Vector Machine	16
4.2.2	Method 2: Clustering	21
4.3	Subtask B - Key Phrase Classification	23
4.3.1	Word2Vec Usage	23
4.3.2	The Word2Vec Vocabulary Problem	24
4.3.3	Development	24
4.4	Subtask C - Relation Extraction	24
4.4.1	Word2Vec Usage	24
4.4.2	SVM - Many Features	25
4.4.3	SVM - Few Features	26
5	The ScienceIE Task: Evaluation	28
5.1	Subtask A - Key Phrase Extraction	28
5.1.1	Method 1: Support Vector Machine	28
5.1.2	Method 2: Clustering	30
5.2	Subtask B - Key Phrase Classification	31
5.3	Subtask C - Relation Extraction	32
5.4	Conclusion	33
5.4.1	Improvements	34
6	Creating a Proof of Concept Use for ScienceIE data	35
6.1	Concept and Specification	35
6.2	Background and Technology Review	35
6.3	Core Component Design	37
6.4	Implementation	39
6.5	Web Interface	43
6.6	Testing	46
6.7	Conclusion	50
7	Conclusion	51

A	Example ScienceIE Training/Test Document	54
B	Example ScienceIE Training/Test Annotation Data	55
C	Stop Words List	56
D	Google Scholar and ScienceDirect Search Pages	57
E	How to Run the Project from Source	58

List of Figures

4.1	Key Phrase SVM Cross Validation	20
4.2	Key Phrase Classifications not in Word2Vec Model	25
4.3	Relation SVM Cross Validation	26
5.1	Key Phrase SVM Self Evaluation Through Iterations	29
5.2	Visual Representation of Key Phrase Clustering	31
6.1	Database Entity Relationship Diagram	39
6.2	Configuration to set the NLP system as a dependency in Maven	40
6.3	The add of the website	43
6.4	An example view page of the website	45
6.5	The home page of the website	46
6.6	The home page of the website	47
6.7	An example search of the website	48
6.8	The Word Clouds with Word Occurance As The Size Metric	50

List of Tables

2.1	ScienceIE Training Set Analysis	11
4.1	Initial Key Phrase SVM Features	17
4.2	Additional Key Phrase SVM Features	18
4.3	Strictness Descriptions for Self Evaluation of Key Phrases	21
4.4	Word2Vec Classification Target Words	23
5.1	Summary of Results Evaluated With ScienceIE Scripts	28
5.2	Word2Vec Classification Results	32
5.3	Relation Extraction Specific Results	33
6.1	Search Times with and without parallel execution	42
6.2	Available Requests the POC Website Supports	44

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 2017¹, a task which heavily applied to the above was presented: ScienceIE². 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 an 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. Explicitly, 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 seen 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 referred 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 Scholar³ and ScienceDirect⁴ 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 report initially considers the background and current achievements for information extraction among researchers, along with discussion over the data provided at ScienceIE. Following this, the architecture of the implemented solutions shall be discussed, before going into depth about both the NLP system and the POC system. Both systems shall have concepts and requirements discussed, their implementations described and the outcomes evaluated. While each system will have its own conclusion which will include extensions, the report shall end with a final conclusion reviewing what has been achieved as part of this project.

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.

An F1 score of 1 is perfect, and an F1 score of 0 means nothing was classified correctly at all.

Tokenization

Tokenizatoion 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.

Stop Words

Stop words are words that are commonly filtered out due to their lack of specific meaning and generally do not contribute to useful input, although exceptions can be made if there is little other information. For example, common search engines are likely to ignore the word "the" in most searches, but this process of removal needs to be conducted carefully as, for example, when searching for "the who", "the" is an important part of that query.

The stop words concept is used in this report, and the list of stop words used is taken from the Stanford CoreNLP GitHub repository¹ and shown in appendix C.

¹<https://github.com/stanfordnlp/CoreNLP/blob/master/data/edu/stanford/nlp/patterns/surface/stopwords.txt>

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 have 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 documents very recent activity in the information and 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, with each of subtasks A, B and C were 0.56, 0.44 and 0.28 respectively (for end-to-end tests) (Augenstein, Das, Riedel, Vikraman & McCallum 2017).

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, multinomial naïve Bayes and Conditional Random Fields (CRFs)) all achieved very similar and reasonably accurate scores (the best had an F1 score of 0.64 when evaluated solely on subtask C).

Furthermore, a common preprocessing technique was to use the spaCy NLP pipeline to analyse the given texts to achieve knowledge about their semantics (Honnibal & Johnson 2015). Then, the key

phrases were directly annotated on to the text model with a flavour of tagging scheme (typically Inside-Outside-Begin or Begin-Inside-Last-Outside-Unit) so that algorithms operating on the produced models have the annotation information to learn from. Algorithms would then predict these tags on the test data, which would then be post processed into the BRAT annotation format for evaluation.

The best end-to-end ScienceIE team applied sequence tagging models, which in turn used long short-term memory (LSTM) neural networks and CRFs to solve each subtask separately (Ammar, Peters, Bhagavatula & Power 2017). Their sequence tagging models also employed gazetteers built from scientific words extracted Wikipedia and Freebase, which appear to perform suitably in the context of scientific papers, as their high results (scoring first or second in all of the scenarios evaluated) indicate the vocabulary supported by these two resources covered much of the test data (their algorithms would likely have not scored so high if a large portion of the vocabulary was missing from the gazetteers created).

Another contribution also included the use of CRF based models (Marsi, Sikdar, Marco, Barik & Sætre 2017). This team actually created a range of CRFs, each with a different intention. They built a range of models, each targeting a specific goal (general key word extraction, task key word extraction, task classification and so on), and each with their own set of features chosen through cross validation. These features generally focussed on the format of the word, the position of the word, and (for material key phrases) comparison to known hyponyms, using WordNet² as a data source. A problem encountered here was that they found some classes had few examples given, so as a solution they removed all sentences that didn't contain a given class (to only focus on areas with positive examples to help balance class distributions). Finally, as there were overlap in the CRF classifiers created, they could be run in parallel and a voting system used to choose a final prediction.

Both of the solution sets above 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 in the same document are labelled to match). The teams also exploited hyponym relationship's bidirectional property (so if word 1 is a hyponym of word 2, word 2 is a hypernym of word 1), meaning their classifier could classify the relation in either direction and the final *hyponym-of* relation could be returned.

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 to select and label key phrases, either directly working together or separately and then their results being passed through some voting system to choose a final prediction. The majority of the solutions presented at ScienceIE involve supervised learning, which requires the use of the training data, with little mention of unsupervised approaches. This suggests the current best solutions require learning from examples, which could cause a problem if the training data is of poor quality or (as one team worked around) there a lack of examples for a given class. Furthermore, as training data is being used, over-fitting needs to be considered, where solutions presented may work well for the test set but not in the real world, but unfortunately it is hard to evaluate the application of the produced algorithms on random scientific articles as the research ended after testing with ScienceIE data in most cases.

2.3 Further Background

Now a review of research in a larger field is reviewed to explore other contributions to information extraction.

A CRF based key phrase extraction model was completed for use on Chinese documents which saw very high results. Before describing their findings, this paper targets a language other than English, so their use of semantics may have different affects if applied in the same way to English sentences. The CRF model designed contained 22 different features, where almost all of the features were based around the position of the token in the text. For their test data set, the largest F1 score measured was 0.51, which is very similar to the best scoring mechanism used on the ScienceIE data set. This proves CRF models can be effective when applied to multiple languages, but also infers that little progress has been made in the decade since this research, given (although different data sets were used) very similar results.

As mentioned earlier, WordNet is also a potential source of information for building a classifier, although other knowledge bases exist. One study compared building a classifier using WordNet and Wikipedia for hyponym-only extraction (Snow, Jurafsky & Y. Ng 2013). The concept of the system produced was that it could learn the lexical patterns of example hyponym pairs so that could be used

²<https://wordnet.princeton.edu/>

to find hyponym relations. Using Wikipedia scored higher (F1 score of 0.36 on their dataset) than when using WordNet (F1 score of 0.27), showing it may be more suited to creating this type of classifier. While an improvement, it is not ultimately a huge increase and there is no evidence either Wikipedia is more useful for the specific area of scientific papers, as the dataset used in this study was not focussed. However, Wikipedia has more information than WordNet, and would likely cover more of the scientific concepts covered by any ScienceIE data. It is also updated on a more regular basis meaning new definitions are likely to be available much earlier than if using WordNet. With the study being completed in 2013, as Wikipedia matures it may, over time, increase or decrease the result of the experiment if repeated, depending upon the quality of the information contained in Wikipedia, which if tested would help evaluate its effectiveness.

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 against new unseen data but even different languages. A study used the idea that words can be clustered together through relatedness, with the goal of producing a set of clusters representing topics in a document (Liu, Li, Zheng & Sun 2009). To extract the key phrases, using the simplest approach, the centre of a cluster is the key phrase. Various clustering methods were attempted by this study. They ran tests on relatively short articles and while at maximum they only achieved an F1 of 0.45, there were several improvements suggested which apply to the ScienceIE task concerning scientific papers. Two suggested covered how the clusters were formed, including clustering on noun groups (which most key phrases at ScienceIE are) and creating a heuristic for co-occurrence based and Wikipedia based relatedness. Another suggestion was to improve the removal of unimportant words. The study only used stop words, but introducing a TF-IDF filter could also be possible.

2.4 Word2Vec

Through recommendations, this project also considers the use of Word2Vec. Word2Vec is an application of neural networks, that processes text and creates a vector space containing each word in the text. This vector space can be used to find the similarity and differences between words (Mikolov, Chen, Corrado & Dean 2013). This technology is relatively new (initially designed in 2013) and its uses are still being discovered, with a similar project called GloVe being worked on the following year (Pennington, Socher & Manning 2014).

Given a large set of input texts, Word2Vec aims to learn the meaning of words. It processes all examples of where every word in the document library has been used contextually, and builds a vector representation of each word. These vector representations can be compared to try to extract the similarity, or relative similarity between words. For example, the relative distance from a *knee* to a *leg* may be similar to that of *elbow* to *arm*. It has been proven to be effective, although very difficult to understand (Goldberg & Levy 2014).

In terms of information extraction, there has been a recent study into its use to find extra hyponym information from text (Nayak 2015) using GloVe. This study designed a function based on the GloVe vector space to create a system to predict hyponym pairs. Reasonable results were achieved, scoring up to 35% precision in some cases. However, a problem discussed is that words can have multiple meanings, which can effectively confuse the model, which decreases its quality for determining similarity and therefore reducing the effectiveness of a system built.

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

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. The following describes the environment and technologies used generically through out the entire project.

3.1 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 2018¹, 2017² and 2016³. 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 ²⁴ 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.
- 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.

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/>

- 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.
- Specified layout of logs: the developer of a project can detail what information (and the precision of the information) is included in a log statement (for example, time of the log, source of the log). This, along with all other configuration for using log4j 2, is completed in `log4j2.xml` in a Java projects `resource` folder.

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 Maven⁵ 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 repository⁶ 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, log4j⁷ 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.

⁵<https://maven.apache.org/>

⁶<http://repo.maven.apache.org/maven2/>

⁷<https://logging.apache.org/log4j/2.x/maven-artifacts.html>

The tests can also be ignored⁸ which is very useful, as many of the tests written are base around 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 functionality⁹. This supports training a Word2Vec model, using the model, and saving and loading models.

The models used in this project are the Google News model¹⁰ and the Freebase model¹¹. 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. This is the reason Word2Vec is being used over GloVe, as these models have much larger vocabulary sizes that the pre-trained GloVe models found, so in theory have a better chance of covering the vocabulary of the ScienceIE data set¹².

Attempts were made to use a Wikipedia based model (Wiki2Vec¹³) 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.

3.2 Platform

While Java is cross platform (another excellent reason for using it), some of the underlying system libraries that Word2Vec 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.

⁸<http://maven.apache.org/surefire/maven-surefire-plugin/examples/skipping-tests.html>

⁹<https://deeplearning4j.org/word2vec.html>

¹⁰<https://drive.google.com/file/d/0B7XkCwpI5KDYnINUTtISS21pQmM/edit?usp=sharing>

¹¹<https://docs.google.com/file/d/0B7XkCwpI5KDYaDBDQm1tZGNDRHc/edit?usp=sharing>

¹²This GitHub page contains a selection of popular models for both Word2Vec and GloVe: <https://github.com/3Top/word2vec-api>

¹³<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 impenetation 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 PDFBox¹ (imported through Maven²) 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 Maven³). 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>

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

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 Features

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 RBF 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

Feature Description	Value Range
The Word2Vec distance from the given token to "task".	$\text{svTask} \in \mathbb{R}, 0 \leq \text{svTask} \leq 1$
The Word2Vec distance from the given token to "process".	$\text{svProcess} \in \mathbb{R}, 0 \leq \text{svProcess} \leq 1$
The Word2Vec distance from the given token to "material".	$\text{svMaterial} \in \mathbb{R}, 0 \leq \text{svMaterial} \leq 1$
The depth of the token in the parse tree for its parent sentence, divided by the maximum depth of any token in that sentence.	$\text{svParseTreeDepth} \in \mathbb{R}, 0 \leq \text{svParseTreeDepth} \leq 1$
Whether or not the token is a stop word.	$\text{svIgnoreWord} \in \{0, 1\}$

Table 4.2: Additional features to be calculated for each token added to the SVM. These were implemented in order and their benefit or reduction in performance is measured.

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.

To allow for further SVM usage, an abstract `BaseSvm` was implemented so that any SVM created as part of this project can extend from this and use these functions generic to all SVMs. This has a default configuration set, a generic training method (calling `.svm_train(problem, configuration)`), a generic predict method and a `.makeNewNode(index, value)` method. This `makeNewNode` method creates a feature for a data point in the format the SVM library can understand (a *vector* is an array of `svm_nodes`). It also validates the value of the feature, setting it to 0 is somehow an infinite or *NaN* has been passed in.

This base SVM class also holds a cross validation function (`.doCrossValidation(C, γ)`), which runs the cross validation procedure and returns the accuracy as a percentage.

From the above, the `KeyPhraseSVM` class was created which contained the functionality to build the required feature vectors of input data and add labels to the training data, which completed the requirements of the SVM.

To actually produce key phrases for a `Paper` object, once a token was *predicted* to be part of a key phrase, its start and end position was noted. The following sequence of tokens would all have their status of key phrase or not key phrase predicted, and while there is a string of tokens predicted to be key phrases, the end position is updated to cover this range of tokens in the original document. Once a token is predicted to not be a key phrase, the `.makeKeyPhrase` method of `Paper` is called with the start and end positions of the tokens to be inside the key phrase, which creates the `KeyPhrase` object.

Additional Features and Post Processing

While working on development of the key phrase SVM, further features were considered and implemented. To evaluate how effective they were, a baseline score was achieved and then each of these were added idea by idea to test if they increased the system performance or not. The additional items, in order, are listed in table 4.2.

⁴<https://mvnrepository.com/artifact/com.datumbox/libsvm/3.22>

The initial three focus on Word2Vec's `.similarity(word1, word2)` method, which finds the distance between two words in the Word2Vec space. The distances chosen were to the three types of classification laid out in the task description. The theory behind this is: as we are looking for tasks, processes and materials, words which are closely related to these concepts may be likely to have a higher similarity to those words in Word2Vec's vector space. Including this information as a feature in the SVM may help distinguish those data points which should be selected as key phrases (as they may be particularly close to one of the three target concepts).

Next, the parse depth of the token is considered. The idea of this is that if you imagine a parse tree, the leaves that are further away from the original sentence (has more branches leading to them) may generally be the more important parts of the sentence, as the other parts of the sentence (leaves less far down the tree structure) simply lead up to the key phrase and add extra information. Therefore, this should test to see if how *central* a token is contributes to whether it should be seen as a key phrase.

Finally, there is a flag to described whether the token is a stop word. This should work with the TF-IDF feature described earlier to help identify words that are unimportant and should not be included in key phrases.

Along side these features, the results produced as part of development and manual analysis of the key phrases created gave incentive to *tidy up* the key phrases with some post processing. This was to try to remove unwanted parts of key phrases, or bad key phrases entirely, and in conjunction with the above features, the following post processes were applied:

- *TF-IDF filter* - In an attempt to remove key phrases which were effectively just *noise*, each key phrase was treated as a candidate key phrase and would only become a true key phrase if the total TF-IDF of the tokens inside of it was above 0.02. This value was set by reviewing the key phrases produced in earlier runs, calculating their TF-IDF values, ordering them and finding the first key phrase that had suitable words (what we'd want to see in a key phrase) in it. Rounded slightly down, this value was set as a threshold and removed all extremely low TF-IDF key phrases.
- *Sanitisation on creation* - When a new key phrase was found by the SVM, the `.makeKeyPhrase` method is called. This method was amended to try to only produce well formed key phrases. This includes attempting to remove redundant symbols, blank space and stop words at the start and the end of the key phrase. The goal was to try to close the gap between strict and inclusive evaluation by removing some of the redundant information. Empty key phrases were also discarded.
- *Final sanitisation changes* - The sanitisation added to the `.makeKeyPhrase` was revisited. The previous changes were refined and improved, and it was ensured all key phrases have some content in them (including support for special scientific characters such as " α "). A new process added here was the splitting of key phrases if there is a bracket half way through the phrase. This creates two, more semantically correct key phrases (as originally, either side of the bracket would have been from different parts of the sentence), where at least one is more likely to be correct as no gold standard key phrase go across a bracket boundary and doesn't return later.

Through some analysis of the output, the final changes to sanitisation also gave this system the opportunity to match obvious synonyms: If the original key phrase was "Support Vector Machine (SVM)", the two split phrases would be "Support Vector Machine" and "SVM". A simple check was implemented to see if the capitalised characters of the first phrase ("Support Vector Machine" would be filtered to "SVM") were part of the second and vice versa, and if so a synonym relation was drawn between the pair of key phrases.

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.

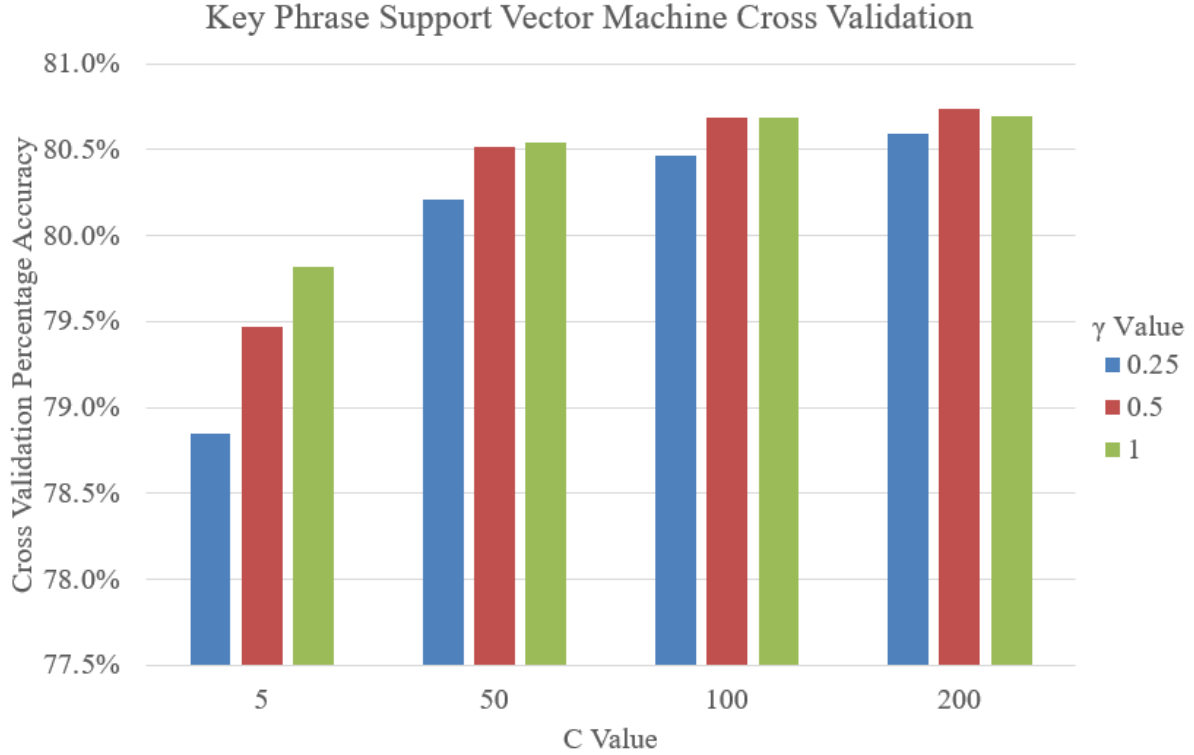


Figure 4.1: The cross validation results on the key phrase SVM. These were completed with all initial features and all extension features in place.

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 shown by this chart, very little change in accuracy is observed. There is an 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 increases 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.

Self Evaluation

A problem referenced in the literature review was evaluation of key phrase extraction. What was discussed is how to classify good and bad systems. It is easy to accurately say how well a system performed compared to a set of gold standard data, but this paper considers whether that is enough to say whether a given extraction system is a success or not.

Therefore, in preparation for evaluation, several metrics will be used. The metric titles and descriptions can be seen in table 4.3. The point of these different metrics for deciding whether a prediction is good or not, is to try to capture a sense of how much of the information we are aiming to extract we actually achieve, potentially foregoing some extra information. Of course, a metric equal to that of the ScienceIE test scripts is included so we can get an expected value for when running with those scripts and a good comparison value for the other metrics used in this paper.

Firstly, the *matching* variant is believed to be perfectly reasonable: it results in exactly the same textual information output, it is merely less concerned with the information's original location. A higher score in this should be considered very positive, as when a reader is looking through key phrases extracted from various papers (for example when trying to pick one out that is useful to them), they likely don't care exactly which sentence the text came from. Even when looking at an individual paper as part of the ScienceIE data set, origin sentence likely doesn't matter as the documents are short and it should

Strictness	Description
Very strict (REALLY_STRICT)	Both the predicted key phrase string and its boundaries need to match the gold standard string and boundaries. This is of equivalent strictness to the <i>BRAT</i> evaluation completed by the ScienceIE scripts.
Matching (STRICT)	Just the predicted key phrase string needs to match the gold standard string.
Inclusive matching (INCLUSIVE)	The predicted key phrase string must be equal too or include the gold standard string.
Generous matching (GENEROUS)	The predicted key phrase string can include the gold standard string, or the gold stand string can include the predicted.

Table 4.3: The strictness levels used in self evaluation of key phrase extraction. The *strictness* title also includes the **Strictness** enumeration value as in the Java code for reference.

be easy to see where the phrase could have come from. This relaxed metric compared to the ScienceIE evaluation may be less favourable if the data set was made up of larger documents, as in a large document position makes more of a difference if trying to find information surrounding a key phrase; however, here it should be reasonable to just check successful key phrase extraction in this way.

There are also two much more lenient metrics. While it is not thought these should be official ways to evaluate an extraction technique, it is worth considering what these metrics are telling us about what information has been extracted. Evaluating with the *inclusive* metric will inform the reader about how many of the key pieces of information have been found, but potentially with some extra information included. This extra information lowers the quality of the key phrases, but does not stop the key pieces of information from being extracted. Therefore, if using this metric scores a high result, the extracted pieces of information is likely still very useful, even if stricter metrics return significantly lower results. The *generous* metric is less useful, as it will likely allow many false positive *true positives* through an evaluation system - given a *true positive* can be awarded when just part of a key phrase is extracted - which does not guarantee key words within the key phrase are captured. However, similar *inclusive* and *exclusive* scores that are both high should mean there are very few instances where only part of a key phrase has been captured, while most key phrases predicted are with exactly right, or have some extra pieces of information attached.

The evaluation described above shall be used to evaluate SVM progression as each idea is added, with the *best* SVM version found there used when testing on the ScienceIE scripts.

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 in 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 it is 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

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

Table 4.4: The target classes and the set of words that are of a similar domain to be used in testing of the Word2Vec classifier.

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.

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 4.4. 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 classification, 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 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(word1, 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 classification 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.

Classifications of Key Phrases not found in the used Word2Vec Models

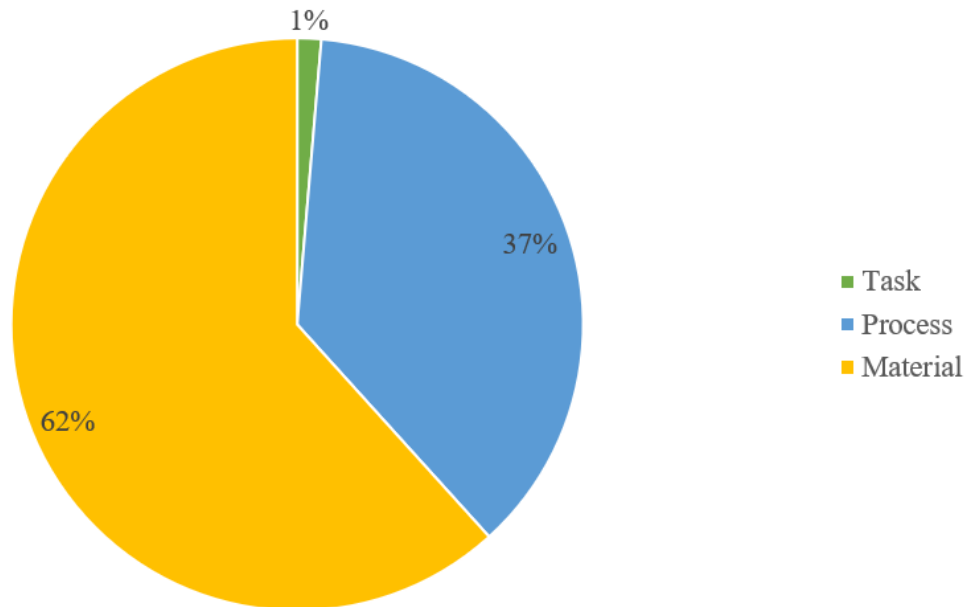


Figure 4.2: Gold standard key phrase classifications for key phrases that are not in the Word2Vec models used. 222 key phrases are included in this set.

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

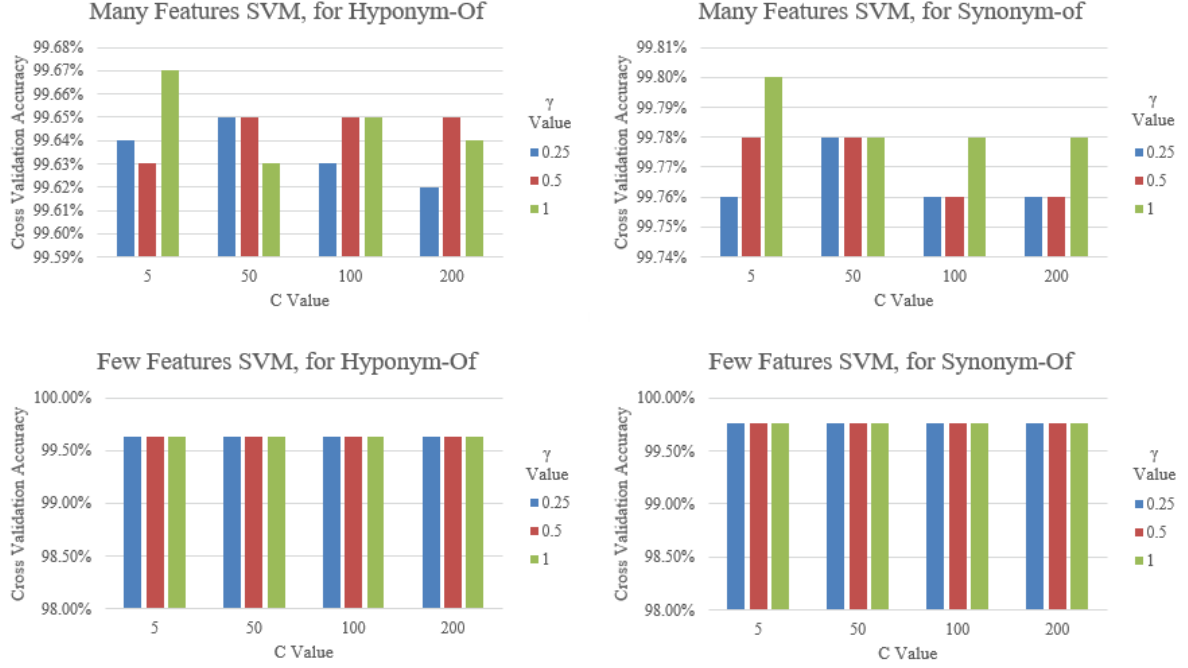


Figure 4.3: The cross validation results on both the hyponym and synonym variants of the *many features* and *few features* relationship SVMs.

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** (which extends from the previously described **BaseSvm**) 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 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:

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

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

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

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 in 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** (which also extends from **BaseSvm**) 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.

A summary of evaluation using the ScienceIE scripts, which covers all subtasks using the best algorithms found within this paper, and also includes the ScienceIE scores for comparison, is presented in table 5.1. These will be discussed throughout this section.

5.1 Subtask A - Key Phrase Extraction

5.1.1 Method 1: Support Vector Machine

As described in the design section of this report, the SVM created for extracting key phrases began with a set of features which then underwent several iterations with other features and post processing being added. The full results, from start to finish, can be seen in figure 5.1.

The results of the initial SVM are interesting. The strict metric shows middling results below that of ScienceIE’s average. However, the two more generous metrics are very high. This implies two good things: Firstly, as the *inclusive* metric is high, it shows that while some unwanted information is being extracted, much of the key phrase data is been chosen. Furthermore, the *generous* metric scoring not too much higher implies that a lot of the other key phrases incorrectly extracted are just noise produced by the SVM rather than the gold key phrases with some missing information. This shows that the SVM isn’t performing terribly as much of the required information is outputted, however, it has some overflowing of extra information which needs to be reduced. The redundant information could be removed by improvements to the SVM, as well as some post analysis and processing to remove extra, unwanted pieces of information, which is why the processes discussed above were developed and their analysis is to follow.

The Word2Vec features added generally slightly improved things across all metrics for at least one Word2Vec model. Oddly, while using the Google News model increased the score for the more generous

Subtask	ScienceIE		This Report	
	Individual Best / Average	End-To-End Best / Average	Individual Best	End-To-End Best
KP Extraction	0.56 / 0.38	0.56 / 0.38	0.2	0.2
KP Classification	0.67 / 0.57	0.44 / 0.26	0.55	0.11
Relation extraction	0.64 / 0.43	0.28 / 0.07	0.1	0.02
Overall	N/A	0.43 / 0.25	N/A	0.11

Table 5.1: The best F1 scores achieved in this paper evaluated with the supplied ScienceIE scripts. This includes both tests for end-to-end data production, and individual subtask tests (where the gold standard data from the previous subtask is fed in). Summarised ScienceIE results are also included, extracted from the ScienceIE proceedings(Augenstein et al. 2017).

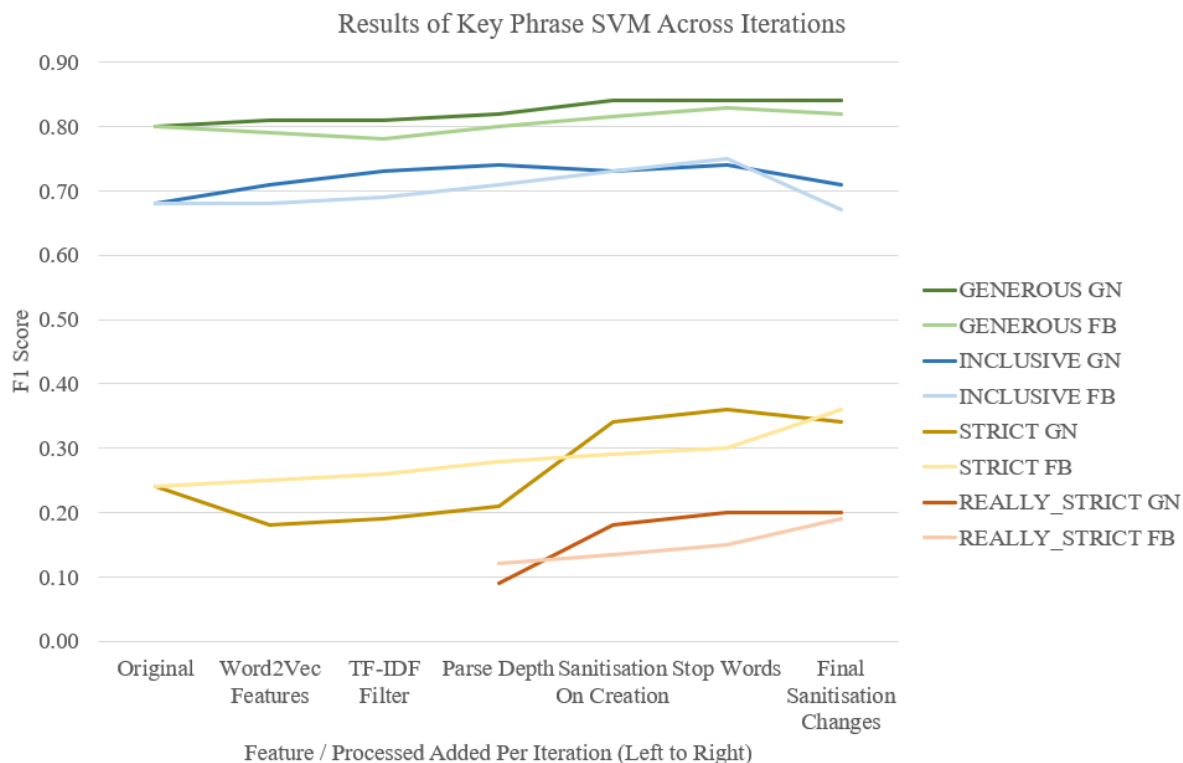


Figure 5.1: The key phrase SVM self evaluation results for varying strictness across iterations, where each iterations changes 1 thing. The coloured lines denote strictness as defined in table 4.3, with the legend assigning each colour to each strictness level. With each assignment, the darker of variant of the same colour is when the Google News (GN) Word2Vec model is used, and the lighter for when the Freebase (FB) model is used. Very strict evaluation was only created part way through development, so some early results are missing for that, but they would likely follow a similar patten to that of matching or *strict* evaluation.

two metrics, it negatively impacted the strict metric, reducing the F1 score by 6% compared to the original score. The Freebase model was the opposite of this, although less significant differences were observed, as for the more generous metrics things slightly degraded, and for the strict metric things slightly improved. For this feature, it would imply Freebase was the best model to use as it did increase the results of the strictly measured predictions.

The first post processing effect was added here with the TF-IDF filter. This gave all configurations minor improvements (the average F1 score rose by 0.01) as expected.

Including the parse depth feature saw averagely 0.01 score improvements across the board as well. The graph showing progression also begins to include the very strict metric as well. This metric is in-line with how the ScienceIE evaluation works, matching both the key phrase string and position in the document. The result saw a 60% score reduction compared to the, just string matching, strict metric when using both Freebase and Google News. This means the system is choosing useful pieces of information from the papers, but unfortunately roughly half of the key phrases chosen are not from the correct place in the text.

Following this, the largest change to the SVM based key phrase extraction system happened; the inclusion of the *sanitisation on creation* changes saw large improvements in F1 score - averagely 0.1 improvement for the Google News based strictly and very strictly evaluated tests. This was a very positive step, as the two generous metric scores didn't change very much. This mean that this post processing was effectively transforming a number of key phrases from being inclusive of the gold standard to matching the gold standard.

Including the stop word flag feature again slightly improved things for the actual SVM performance, again averaging an increase of 0.01 F1 score.

The final change to the SVM process was to make the final changes to the key phrase creation sanitisation. While averagely this made no difference to the F1 results, for some cases it significantly

damaged scores; namely when using the inclusive metric with the Freebase model, the F1 score dropped by almost 0.1. The Freebase model, under these presented set of features and processes, performs worse than the Google News model for all metrics aside from *strict*, where the difference is only 0.02 F1 score. This change was still positive however, as it made the key phrases much more pleasant to look at. Cases such as "support vector machine (svm" was tidied up, and generally words were not missing final letters which was a problem from earlier in SVM's life. Furthermore, this last change saw an opportunity to quickly generate some synonyms for almost no cost. Using this method saw 14 synonyms correctly generated when processing the test data.

From this, the best configuration can be seen to be when all proposed features and processing effects are used, with the Google News model being used for the Word2Vec features. Running this configuration and evaluating the predicted key phrase with the ScienceIE test scripts produces an overall F1 score of 0.2. This is perfectly in line with the self evaluation conducted under the strictest metric, which reinforces the self evaluation as trustworthy (as it got the correct score), and through that helps to validate claims the other evaluations give the information about the system they are meant to.

Experimenting With Longer Documents

The ScienceIE proceeding paper SemEval 2017 (Augenstein et al. 2017) was chosen to be tested in the key phrase extraction system. The SVM with all extensions and post processed was used to extract key phrases from the paper, and the results analysed. For this, the paper was manually convert from its PDF format to a text document, stripping all headings, footnotes, pre and post amble, leaving only the abstract and main content.

From the long paper, only 121 key phrases were extracted, with only 69 unique phrases. Furthermore, of the unique key phrases, many were very similar, only different because of line breaks, spaces or punctuation throughout the word (for example, "scenario teams" and "scenario teams", "annotator" and "anno-tator"). There were 201 words across all 121 key phrases, meaning that most of the key phrases were very short (an average of 1.7 words per key phrase) which is quite low compared to the average 3 in the ScienceIE test data.

Something that didn't happen was an overload of key phrases. While around triple the largest number of key phrases for a single document in the ScienceIE training set, given the length of the document, 121 is actually quite low. Only 5.7% of the tokens in the document were labelled as a key phrase, which is significantly lower than the 36% average in the ScienceIE training set.

In terms of the quality of the key phrases, there were some important pieces of information extracted (such as "annotations of hypernym", "SVM", "scientific articles of the Computer Science"), but most were a single word which didn't mean anything. It repeatedly selected "subtask" and "documents", and while these are important to the document they are not necessarily key phrases of it. Aside from learning a little about the ScienceIE task, and the name of some of the teams attempting solutions, the extracted key phrases provide little information to help describe the actual content of the paper.

5.1.2 Method 2: Clustering

While the SVM method showed some good results, unfortunately clustering was not as successful.

A problem encountered during development was that not every word was in the Word2Vec vocabulary, meaning not every word could be clustered. This is actually quite a major problem as, given the context of the words we'd like to cluster is scientific papers, many of these unusual words not in the corpus are likely part of the key words describing the key pieces of information in the paper. Given we want to be able to cluster based on those terms to find them in future papers to extract as key phrases, this limits the capability and robustness of the algorithm. The only thing that could be done was to continue clustering until we are left with one cluster with everything that can be clustered in, and a group of clusters with one element each, where those singular terms are not in the vocabulary of Word2Vec. This problem didn't occur for every test paper processed, but has shown some papers to have over 20 tokens that were excluded from the Word2Vec model.

Next came the problem of how the clusters formed. Ideally, we would like to see the case where, for many iterations, many clusters are present with many items each. This could then be used to split the document into many clusters with a (for instance) key phrase in each, taking the tokens at the centre of each cluster at a given level to be part of its own key phrase. What was generated instead was a pattern where, generally, two items clustered together which were there joined by a third item on the next iteration, then a fourth on the next, and so on, while no other tokens clustered together in other clusters. This is significantly less useful, as a bunch of words are grouped together in a growing group,

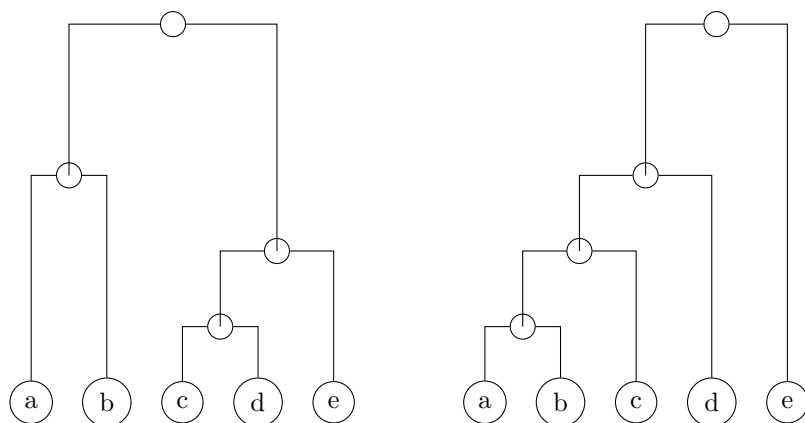


Figure 5.2: Examples of hierarchical clustering on a set of items, a , b , c , d and e , through to when they are all in the same cluster. The left drawing is an example of ideally what we would like to have seen. The right is a representation of what was very common generated.

while all others are left on their own until they are included in this large group. Very occasionally, other clusters grew to have a few items rather than immediately join the large cluster, but this was rare to see and still not very useful to draw multiple key phrases from.

To aid in the understanding of what happened, an illustration is provided in figure 5.2. The left drawing is the ideal case, and the right is what commonly occurred. It didn't necessarily have to happen in token order, but the point is that aside from picking tokens that are very close together, this does not give us much insight into the key topics of the paper.

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 5.2.

The obvious thing to note here is that all tests under Freebase without a default class 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, seemingly containing one of the supplied words.

When using the Google News model, things are less clear. Generally, using this model a decent accuracy can be achieved, even without the support of a default classification. This shows the model does include many of the terms used in the publications presented. In terms of distance metric, *average* was slightly better; it was part of the best configuration (scoring 54.6%), and also the average of all of the *average* distance configurations was 2% higher than the average of the *closest* configurations (50% and 48% respectively).

The configuration change that made the least difference was whether or not the unimportant and stop words were removed. This saw changes, averagely, of 0.1% when comparing two configurations differing in only whether these words were removed. When some words were not removed the algorithm performed better, indicating that these words are more important than first thought for deciding class. This may be down to the edge cases discussed such as "He" or similar which, despite being a stop word, we would want to keep.

Using various words when finding similarity decreased the accuracy of classification, and was the largest detrimental parameter. When using many words to find similarity, accuracy decreased an average of 2.2% (50.3% down to 48.1%). This means the position of the target classes in the Word2Vec model are likely quite accurate when compared to words that can be used in a similar context.

Using the best configuration found my testing within this paper, evaluating this classifier through the ScienceIE scripts gives an F1 score of 0.55 when evaluated individually (given the gold standard key

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	46.05%
Google News	Closest	Unknown	Yes	No	45.96%
Google News	Closest	Unknown	No	Yes	44.05%
Google News	Closest	Unknown	Yes	Yes	43.91%
Google News	Closest	Material	No	No	52.73%
Google News	Closest	Material	Yes	No	52.63%
Google News	Closest	Material	No	Yes	50.73%
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 5.2: 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 those above the baseline score of 44.05% which is where every key phrase is simply classified as a *material*. 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.

phrases). This isn't a bad score, as it almost matches the average ScienceIE F1 score for this section done independently.

Other Experimentation

While working on key phrase classification, having seen some decent results produced by the SVM created in subtask 1, a short experiment was conducted to see how well using position in a document could be used to determine classification.

The process of trying this was very simple: re-purpose the subtask A SVM (with all described extensions included) code to, when training, label data according to a class rather than to key phrase. Two versions of this were attempted:

- The SVM is given 4 different labels, one for each classification and one for no classification (i.e. not part of a key phrase).
- 3 different SVMs were trained, each only having one classification labelled.

Given the only real semantic meaning for a token in the SVM was based around whether the token was a noun, this was not expected to perform well as semantics and the specific words surrounding what we are trying to classify generally are expected to be important, rather than the position of the token in the text which gives us little context.

As expected, results were terrible, with F1 scores for both versions being less than 0.1. This confirms that the position of tokens, at least as far as this paper's positional information is concerned, is not useful in classifying tokens. It may have been interesting to try again leaving out non key phrase tokens from the training data generated for the SVM to learn from, but these low results didn't encourage further study into this idea when other, more efficient and productive concepts were being worked on.

5.3 Subtask C - Relation Extraction

Both SVMs concepts were tested and the results for the hyponym and synonym variants combined for full evaluation.

SVM Model	Vector Generation	True Positives	False Positives	F1 Score
<i>Many Features</i>	All tokens included	13	84	0.09
<i>Many Features</i>	Root noun selected	2	28	0.02
<i>Many Features</i>	Unimportant words removed	0	0	0
<i>Few Features</i>	All tokens included	0	0	0

Table 5.3: Results from doing relation extraction with the proposed SVMs. The results are calculated after the relations extracted by the hyponym and synonym SVMs are combined. The *vector generation* column described how a vector for a key phrase was found (for example, where the unimportant words are removed). The bold result is the highest F1 score achieved.

As presented in summary table 5.1, the best overall results for this synonym extraction was an F1 score of 0.1%. This was from using the *many features* SVM with the vectors from each token in a key phrase being used to calculate a key phrases total vector. A more detailed breakdown of how well each configuration performed can be seen in table 5.3.

From this table it is clear to see that the *many features* SVM concept performed better than the *few features*. While the hope was to allow the SVM to work on information with a lower complexity to help it find a better fitting hyperplane, this clearly hasn’t worked as it didn’t pick any relations at all; compared to *many features* with the same vector generation, which chose a total of 101 relations.

In terms of the different methods of generating vectors for key phrases, keeping all of the information in the key phrase appears to be the best method. Most true positives were found when all tokens are kept, while less true positives came from where the root nouns are selected, and no true positives were found when stop words and unimportant words were removed. This mirrors what happened in subtask B, where potentially unimportant information was removed and a decrease in the quality of results is observed; although it is more noticeable here given it is the difference between some and no relations being extracted (there was only a 0.1% change in subtask B’s results).

Furthermore, while leaving more important information in the key phrase for vector generation did improve the system performance, the false positives increases a large amount as well. Selecting just the root noun saw 14 times more false positive results than true positive; compared to when all key phrase information was left in, only 6 times as many false positives to true positives were seen. Not excluding data clearly improves the system accuracy, and increases the F1 score.

5.4 Conclusion

This paper set out to provide a set of solutions to the ScienceIE task and its subtasks. Considering the summary presented in table 5.1, these experimental systems have performed in a satisfactory manner.

The key phrase extraction using a range of features in a support vector machine and some effective post processing achieved reasonable results considering, if we ignore the location of the key phrase extractions, a maximum F1 score of 0.36 which is very close to the average at ScienceIE. For comparisons sake, it would be very interesting to run the same analysis on the results produced at ScienceIE, to see if the best teams realistically came very close to extracting 100% of the required information (if the position of key phrases could be ignored, and potentially a little bit of extra information allowed). Clustering with Word2Vec was not successful, as not useful clusters could be formed.

Given the extraction examination completed on a much longer document, this algorithm is not suitable for long documents. This places the algorithm’s purpose perhaps at being used to extract information from abstracts or similar, which should hold several key points from the paper. This can then be used to help identify papers for reading further into.

The key phrase classification produced good results, again being very close to the average at ScienceIE. While the Freebase Word2Vec model did not serve this use well, the Google News model did a very good job of classifying key phrases, producing comparable results with or without a default classification to give if it couldn’t understand a key phrase.

Finally, the relation extraction did not perform very well, even when attempting just hyponym or synonym extraction. In fact, the one simple rule included in the post processing for subtask A produced 1 more correct synonym than any synonym SVM did during testing with gold standard data as an input. From this, unfortunately using just Word2Vec as a basis for a SVM feature set cannot be recommended, as it has very poor performance under these conditions. It is believed, however, that part of this is because of the lack of testing data as input, given there were many examples given to the SVM of what

a relationship didn't look like, where there were few examples of what the relationships did look like.

5.4.1 Improvements

From this, a range of improvements and extensions could be applied:

- **A more appropriate Word2Vec model.** Word2Vec was used in all three subtasks, but unfortunately only two models were tested with the systems, and neither had the context of scientific papers. If a system was created to crawl the internet fetching many scientific publications and have them all processed by Word2Vec, there may be a potential for a more suitable model to be created. Not only would the vocabulary be laid out in vector space in a way that should likely fit the testing data presented here better, but more of the tokens in this testing data should also appear in the model, which would improve things significantly for a start. This would likely impact the solution to subtask B the most, although it may significantly help the clustering attempts in subtask A if it can produce a more balanced progression clustering.
- **Expansion of key phrase SVM features.** If future work was to continue on the SVM created in subtask A, a way that is likely to produce better results is to include more features about each token. Specifically:
 - Features giving more information about surrounding key phrase tokens (for example more information about recent key phrase tokens, information about overall key phrases so far for the given document),
 - The Word2Vec distance to the previous token (in an attempt to string together related tokens),
 - More lexical or semantic information about the current phrase.
- **More training time and memory capacity for parameter tuning.** Time and memory limitations didn't allow for much more training time to be given to the *many features* SVM, but potentially an increased C value may help to find a hyperplane which could work better. For the key phrase SVM, while the parameters tested in cross validation seemed to plateau, further investigation may have revealed increased performance.
- **Better data for training the relation SVMs.** As alluded to above, give the relation SVMs training data which includes a higher percentage of actual relations cases. This could be done through adding more papers with relations in, or removing some or all papers without any relations in.
- **Experiment with full scientific publications.** While this subject was touched upon briefly in this report, working with full publications rather than short extracts would help contribute to the problem of information retrieval, which currently needs development in the context of long documents. Clearly, the algorithm proposed here is not useful when applied to full publications (in the limited testing done), as no meaningful information is extracted beyond what the title of the document should give. Potentially, the strongest algorithms presented produced at ScienceIE could be recreated, tested and altered to work with longer papers.
 - One method that may be worth exploring, given the algorithms proposed here are targeting short documents, is that processing could be completed on individual paragraphs producing many key phrase for a document. Then a system could be developed to just look at the generated key phrases for each paragraph to choose key pieces of information to represent the entire paper.

Chapter 6

Creating a Proof of Concept Use for ScienceIE data

6.1 Concept and Specification

Having completed systems to handle the information extraction, the next major part of the project was to explore how this information can be utilised for researchers.

The most obvious domain for using summary information about a set of resources is search. Being able to efficiently search through a large set of documents to quickly find more useful information is a very useful thing. The focus of this section is to explore using the key phrase information in an efficient and convenient way to aid in user query across the ScienceIE data set.

The ScienceIE data set is specifically being used because that is what the algorithms in this project are designed for. The problem of handling longer, full publications has been discussed, so to not complicate this investigation further, the standard short documents of ScienceIE shall form the database of documents.

The following requirements are presented for the POC system to conform to are:

1. As a basis to make the system usable, the ability to list papers in the database, and the ability to present all the information about a paper on one screen. The ability to download the extractions in the BRAT format should also be made available.
2. A search system for the key phrase information gained. This should allow the input of a user query (plain natural language text), with an option to give some indication of the classification of the information they are interested in (i.e. *task*, *process* or *material*).
3. The ability to add papers dynamically to the database throughout run time. This means not only should the system support the creation of new paper records, but also the automatic processing of that paper through all 3 subtasks of ScienceIE, with the results being available to the search system.
4. Some way of conveying summary information about the extractions contained in the system. While not essential for search, having a convenient way for the developer and users to take a snapshot of what is contained in the system is useful for evaluation purposes, including giving an indication of how many papers are included and their processing status.

6.2 Background and Technology Review

Existing Search

Two popular, publicly available search engines have been examined to extra some well designed features from. These are Google Scholar¹ and ScienceDirect², and appendix D shows screen shots of both web sites given the same query. Common features are:

¹<https://scholar.google.com/>

²<https://www.sciencedirect.com/>

- Limited results per page (10 on Google Scholar, 25 on ScienceDirect),
- The total number of documents selected and the time it took to do this is displayed to the user,
- Filtering options, including by date,
- The title of a result is followed by a snippet of the document, with relevant words highlighted in some way,
- The title is a link to an individual results page which presents information about the document, but there are also links to directly get the document downloaded.

Some custom features on each website are:

- ScienceDirect supports searches with multiple parameters, including query fields specifically for author, title, pages, etc...
- ScienceDirect also supports custom numbers of search results per page; 25 is the default, but the user can receive 50 or 100 results per page if they wish,
- Google Scholar present *related searches* to allow the user to navigate sideways towards their target paper, rather than straight down to it,
- ScienceDirect features a *recommended* section at the top where it showcases several publications it believes to be the most relevant.

Given the data set available in this project, most of these ideas presented by Google Scholar and ScienceDirect can be implemented. However, as ScienceIE's data is purely text snippets of documents, author, real title, publish information and more are unavailable, meaning any system created here cannot support filtering by these features, or display that information. An original hope of the developer was to include a more advanced search, potentially including a PageRank type system (Page, Brin, Motwani & Winograd 1998) where the links would be based on referencing, but unfortunately without even the reference information for each paper available, this is impossible to achieve with the ScienceIE dataset.

The search on these two websites is likely to be based on latent semantic analysis (Landauer, Foltz & Laham 1998, Aswani Kumar, Radvansky & Annapurna 2012) which looks at the similarities between search terms and attempts to take into account context. This method of representing words in a vector space prepared for querying is a popular method of document retrieval. This method of querying could potentially be modified to involve more weighting of tokens if they are included in key phrases, or be used to inspire a new algorithm that uses key phrase information to help with weighting word importance.

Available Technologies

The goal with this project is to prove a use of the data produced as part of ScienceIE through some application. To make this application available for use, a web based service will be created that allows any user to join without the need for complex setup. The benefits of this include not having to distribute installation media and update media, along with centralised management of any database of information and being able to immediately make available new features.

In terms of storing information, popular database technologies were surveyed. The stack overflow survey 2017 listed MySQL³ as the most used SQL platform by developers on the site⁴. One reason for this is that some competitive SQL platforms require licensing (especially when commercial use is considered), such as Oracle SQL, while MySQL is open source and can be used more freely. The platform can be easily acquired and installed on various platforms (with major Windows and Linux operating systems supported), with largely straight forward initial configuration (for a project of this scale). The standard MySQL installation also includes the MySQL Workbench, which includes a convenient tool for building a relational database schema⁵, with the ability to export SQL scripts to build the designed database in a single execution (including tables, relations and users with certain access rights).

One of the most popular Java web frameworks available is Spring. Spring provides functionality to route requests to certain Java classes for processing, including the passing of any parameters with some degree of automatic validation, and the ability to process Java Servlet Pages (JSPs) which are compiled

³<https://www.mysql.com/>

⁴<https://insights.stackoverflow.com/survey/2017#technology-databases>

⁵<https://dev.mysql.com/doc/workbench/en/wb-data-modeling.html>

at run time to delivery varying pieces of information to the end users. It supports the standard *model-view-controller* (MVC) design pattern. To keep the focus on the use of data processed in this project, a convenient web container is desired, and that is what Spring Boot⁶ provides. This packages the Spring technology stack, web server technology (for example, Apache Tomcat) and any extra libraries required into a single, executable, Java project, making heavy use of annotations for the minimal configuration required. This makes it an extremely portable and (in terms of environment and system configuration) light weight application to set-up and run, which reflects as part of their projects mission statement to get developers "up and running as quickly as possible". It is a young project, with the original version being released in 2014 and version 2.0 being released during the course of this project, meaning it is built to work with the current standard of industrial Java technologies - so while this project is aiming to build a simple application, it shall have some powerful technologies standing behind it.

One of these technologies is Hibernate⁷. This integrates with the Java Persistence API (JPA), supported by Spring, for extremely simply database interactions with many functions being supported by default. It combines a simple Java object which holds fields relating to each of the column in a database entity (table), and an abstract `CrudRepository` class which can be extended from to make a data access object which can retrieve records from a database and update them as well. Methods can be added to the extended class to make more complex queries, and either plain English can be used in the method name to define the query, or the specific SQL query statement can be specified if required⁸. Furthermore, when fields are foreign keys of other entities, rather than being of an *ID* type such as `Integer` or `Long`, the class representation of the foreign entity can be used instead, and Hibernate will automatically populate it with the correct object.

6.3 Core Component Design

Design of the Services

There are four key requirements of the POC system which cover a large amount of usability. The first requirement is very simple. Two pages are required for this:

- The first is simply a list of papers, each selectable. When selected, the user will be taken to the next paper described.
- The second page, for displaying information about a paper, should begin with the title at the top, a section in the middle for the original text, and finish with a list of key phrases and relations. The key phrases, as a minimum, should be annotated on the main text as well.

The second requirement is covering the question of search. A new algorithm is proposed that includes the key phrase information about a paper. Using the concept of combining the TF-IDF scores of both the tokens from documents as well as from in the query to try to prioritise papers is utilised with a scaling factor, defined by the presence of key words in a paper. When searching, the following process should happen:

1. The query string shall be tokenized, and every token shall have its TF-IDF score calculated (relative to the training set, as all TF-IDF scores are calculated).
2. A set of papers are selected, with the criteria of needing to have at least one of the tokens from the query as a substring of the entire text.
3. For every paper selected, the tokens that relate to the search query are extracted. The extracted token may not be the same as the query token, as the query token could be a substring of the extracted token.
4. For every paper, for every unique token extracted in the previous step, the TF-IDF value of the extracted token is multiplied by the TF-IDF value of the query token it relates too as a *TF-IDF weighting*.
5. For every paper, for every token extracted, a *scaling factor* is then calculated. Beginning at one, it is incremented by one for every key phrase (for the given paper) a query token appears in.

⁶<https://projects.spring.io/spring-boot/>

⁷<http://hibernate.org/>

⁸A guide to using this technology is freely available on Spring's website and is presented in a very simple manor. It can be viewed at <https://spring.io/guides/gs/accessing-data-jpa/>

- If a user is *focussing* on a certain classification of information (e.g. *task*), then only the specified types of key phrases are used when scaling.
6. The *TF-IDF weighting* for each token extracted from a paper is then multiplied by the *scaling factor* calculated for the token.
 7. These multiplied values are then summed for each paper. The sum is the papers score.
 8. The papers are ordered by score, with the largest score being first to be presented to the user, and the lowest last.

This should, as a base line, retrieve all relevant papers to a users query with a reasonable ordering relative to the TF-IDF values of the query words. Furthermore, it should prioritise terms that are part of key phrase of documents, which should list documents specifically addressing certain concepts higher than documents mentioning them, which is an ideal use for the key phrase information extracted. A problem which may occur is when a search term with a high TF-IDF value is found (as it has a low occurrence in the document it appears in) and a low TF-IDF word which happens to appear in key phrases. As the multiplier may be large for the low TF-IDF valued word, it may overtake the other, more important word in overall paper score, which may not be idea for the user. However, this isn't expected to happen often, as low TF-IDF valued words are unlikely to be part of key phrases, as (implied by their TF-IDF score) they are not critical pieces of information in the document.

The next requirement is the ability to add new papers. In terms of user input, this is fairly simple, as a string will be taken in and can be treated as a file descriptor. Given many other jobs need completing in this project, and a web resources paper type wasn't created in the NLP system, to for fill this requirement, any paper location added shall be treated as a file URI and looked for on the host system's disk. This can then be loaded into the system and processed through each of the subtask systems.

To complete the processing, a paper will need a status associated with it. This status should describe what the next stage processing the paper should experience is. In order, the processes are: *preprocessing*, *key phrase extraction*, *key phrase classification*, and *relation extraction*. To complete the subtasks, the solutions with the best found configurations in the NLP system shall be used. The use of the set of SVMs this will require can be optimised: rather than training for every paper, the first time a SVM is used it can be trained, then serialised and saved to disk. Every time after that, it can be loaded back into memory without the need for hours or retraining. Furthermore, given the SVMs and Word2Vec take up a large amount of memory, rather than processing one paper at a tie end-to-end, all papers needing an action being completed can be processed at once with its required resources being loaded and reused throughout, before unloading ready for the next stage of processing which will work on all papers in that status as well.

Overall, this should all the efficient processing of all papers in the system. As the extracted content is predicted, it can be immediately added to the database allowing the search and other systems to immediately update with the latest information.

Finally, the last requirement is a little open to interpretation, as all that is required is the display of some summary information about what is in the database. Two data visualisations are proposed:

- The first is a simple pie chart. It shall have three segments, each representing the percentage of key phrases that are each classification. Next to the pie chart (aside from a legend or similar describing each segment) the number of key phrases should also be displayed, so the number of each classification can be understood.
- The second is a larger visualisation. Three word clouds should be created, made up of the tokens of key phrases extracted, one for each classification. To determine the size of a token in a word cloud, two values are proposed: the words commonality (i.e. each words occurrence is counted), and it's TF-IDF value should be used. While both shall be tested, it is expected using TF-IDF will produce more interesting results, as this will allow words with high importance to stand out over words which are still key phrases, but less important. This should give a user an insight into what information is in the system, and ideas for what to search for to achieve better results given they know what words are in the key phrases, rather than highlighting the potentially less interesting words.

Database Design and Entity Relationship Diagram

The support storage and query of the data produced by the ScienceIE task, a suitable database must be designed and created. Given a lot of the required entities had been created as part of the NLP system,

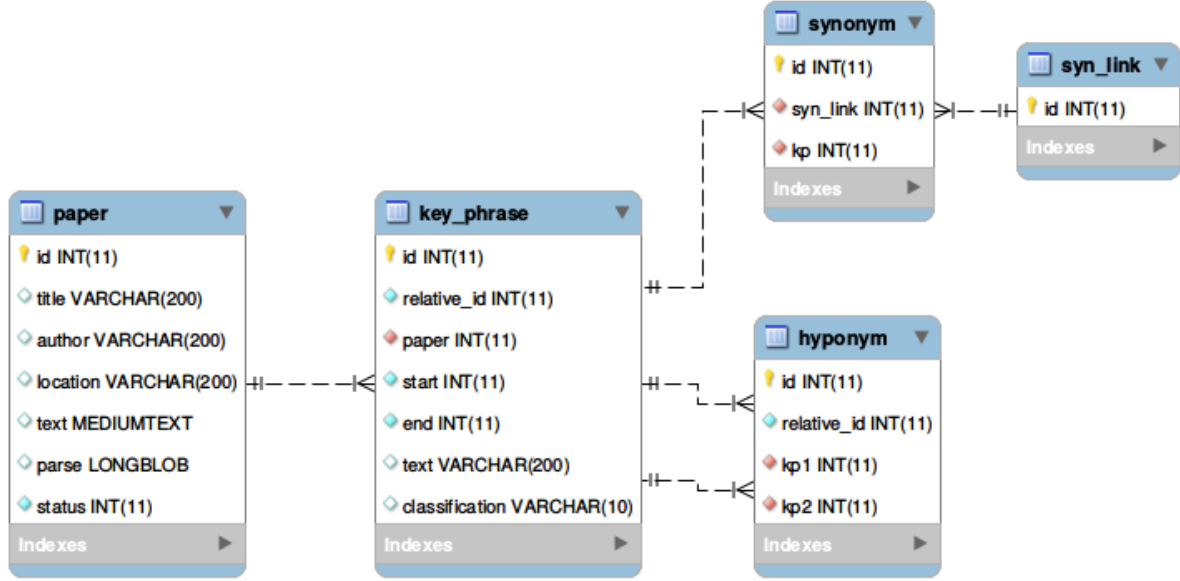


Figure 6.1: The entity relationship diagram that supports this proof-of-concept system, as exported from the MySQL Workbench tool kit. Primary keys are represented by gold keys, foreign keys are red diamonds, normal fields are blue diamonds and hollow blue diamonds indicate the field is nullable. Types for all fields in all entities are also shown. The Java representation of these entities share the same name in all cases, converted to camel case, aside from the entity titles which have DAO appended (standing for *data access object*) as without this the class names would clash with those used in the NLP system, which would have been confusing to program with. As examples, the `paper` entity was `PaperDAO` in the Java project, but an `id` field would remain `id` in the Java code to follow the standard.

the data base design heavily inherits the classes and fields from this. The full design for the database is shown in figure 6.1

The `paper` and `key_phrase` entities are largely the same as their equivalents in the NLP system. Of course, rather than having a list of key phrases held within the `paper` entity, the `key_phrase` entity has a field to store a reference to its parent `paper`. The `paper` entity also has two fields no in the NLP system: a `status` which was discussed above, and a `parse` which holds a serialised `Paper` Java object. This allows the preprocessing to take place and for that information to be stored directly in the database with its parent record, removing the need for re-computation of preprocessing every time the paper is called from the database.

The method for storing hyponyms and synonyms is slightly different to the NLP system. Rather than being part of a relations list the parent paper record holds, instead each relation holds the `key_phrase` references they are a relation of. The original `paper` can still be retrieved through retrieving the referenced `key_phrase` records. The `hyponym` record holds the two `key_phrase` records it is a relation of, as well as a relative ID, which is the ID of the hyponym relation relative to the individual paper it is from. The `synonym` relation is a little more complex. While the NLP system created didn't support more than two way synonym relation extraction and there was only one example of a three way synonym relation in the ScienceIE data set, to support the range of relations defined as part of ScienceIE these synonym entity has to support one synonym referencing a variable amount of `key_phrase` records. Therefore, to avoid a many-to-many relation between entities, each `synonym` record holds a reference to a `syn_link`, the concept being a set of synonyms, each referencing one `key_phrase`, all reference the same `syn_link` record, which makes the synonym relation between the set of referenced key phrases. Having `syn_link` also provides a convenient way to count the number of synonyms in the system.

6.4 Implementation

This section shall cover the infrastructure of the POC system and how the requirements are handled. The graphical user interface (GUI) information shall be described later in the *Web Interface* section.


```

<dependency>
  <groupId>xyz.tomclarke.fyp</groupId>
  <artifactId>fyp-nlp</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <!-- Stop logging dependency errors -->
  <exclusions>
    <exclusion>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
    </exclusion>
  </exclusions>
</dependency>

```

Figure 6.2: The Maven configuration for listing the NLP project as a dependency, as used in the POC system. Note, the exclusion (as discussed) are to remove logging dependency conflicts using Spring Boot causes; this may break logging if another system uses this configuration but doesn't have its own logger available.

Project Configuration

The creation of the project involved including various dependencies in the `pom.xml` file and setting up the launch configuration for Spring Boot.

To include Spring Boot, `pom.xml` was configured as described on the Spring Boot website, along with a connector package for MySQL⁹.

To use the NLP system built in this project, Maven also needed to be told to import this so it can be used to generate information about given papers when the full system has been developed. Given the NLP system has been compiled and installed to the local Maven repository (as it is not hosted anywhere), the configuration shown in figure 6.2 will include the NLP system in a compiled project. Doing this also sets any dependencies of the NLP system as dependencies of this system, so they do not need to be re-listed as dependencies. The unfortunate effect of this is where there are conflicts between dependencies, which is what happened in this project. The logging included in Spring Boot was an alternate version to that in the NLP project, and while it didn't damage any part of the system, on every boot it would choose one of the loggers and print out many error messages warning the developer against it. Therefore, the conflicting dependencies in the NLP system were excluded to remove this problem, and this was done to the NLP system rather than to Spring Boot as this problem may occur if the NLP system was used in another project including other dependencies, so it made sense to list how to fix it for the NLP system dependency.

For the system to launch, some basic configuration had to be set. There is an `application.properties` file in the Java resources with the following parameters (with explanations as to why those were chosen appended):

- **server.port=8080** 8080 is a standard port for testing web applications on. Furthermore, port 80 needs administrative privileges to be used every time the application is launched, which would not only be frustrating to a developer, but not supporting good security standards if there was a vulnerability in the web service. The router the developer system was sitting behind, through port forwarding, allowed incoming connections on port 80 to be directed to 8080 on the developer system, so when connecting via web browser, no port would be needed to be specified.
- **debug=false** If `true` was set, this would output all debug information to the console. Given the log4j configuration being used output all debug information to disk, there was no need to have it also in console, making it harder to see what was going on while running.
- **spring.jpa.hibernate.ddl-auto=none** This parameter has several different possible values. `none` means the connection to the database is standard, allowing the Java application to commit reads

⁹<https://spring.io/guides/gs/accessing-data-mysql/>

and writes. Other values were possible, which would support (if the database hasn't been setup) automatic configuration of the database based on the entity declarations. While this could have been useful, control of configuring the database being left to the entity relationship software included in the MySQL workbench seemed preferable.

- `spring.datasource.url=jdbc:mysql://tomclarke.xyz:3306/fyp?verifyServerCertificate=false&useSSL=true` This is the JDBC connection string, allowing the system to find and connect to the database. To increase security, encryption of the connection through Secure Sockets Layer (SSL) is enabled, but as the database server is not configured with any certificate, this is set to be ignored.
- `spring.datasource.username=fyp_user` This is simply the database user setup for the system to use. The permissions on this user were restricted to read and write of data in the `fyp` database, so if the application is compromised the database connection can't be used to read information from other databases on the same instance, or change the configuration of the database system.
- `spring.datasource.password=<password>` This is the password of the database user for the system to login with.

Several parts of the above are concerned with, as well as required configuration, security of the database and system as whole, and while this project is not concerned with security, it is seen as good practise to include these features.

Finally, the class `FypGuiApplication` needed to be created, as it is this class that contains the classic `public static void main(String[] args)` method that starts the whole program. Along side the opportunity to include any other pieces of configuration completed through Java code, this initiated the Spring MVC technology stack.

Implementing The Requirements

To complete the requirements, several internal services needed to be created. For the simple display of data (for the first requirement), simple database queries could be ran and the data immediately propagated to screen. However, the other requirements require more complex solutions.

Before continuing, the following classes are all annotated with `@Service`. This tells Spring to instantiate these classes (each one as a singleton) and allows them to be automatically injected into other objects. For example, `PaperUtil` is a class which provides utility methods for the system (including holding training paper information and utilities for calculating TF-IDF values and for loading serialised papers). It has a `@Service` tag, and if another class wishes to use it, `@Autowired private PaperUtil util;` can be added as a field in the class and Spring will automatically make the instantiated `PaperUtil` object available.

To handle search, two classes were created. `PaperSearch` handles the incoming `SearchQuery` object, which is all the information about a users query. Using this information, `PaperSearch` retrieves all the database information required to complete the suggested algorithm for calculating paper scores. Each paper, one by one, is then passed to `PaperScoreCalculator.calculatePaperScore` to have its score calculated. The reason for a second class being used here is it allows the user of asynchronous technology. The method called is annotated with `@Async`, and returns `CompletableFuture<Double>` (score is measured in the `Double` data type), which causes the processing calling the method to not wait for a return value and carry on, effectively making a new thread every time the method is called. The effect this has is that a loop can be made, calling this method for every paper, which immediately returns a list of `CompletableFuture<Double>`. This list, effectively made up of a set of threads, can be joined on later, meaning execution will only continue past the *join* once all threads have finished their calculations. The configuration for this pool of threads is completed in the `FypGuiApplication` class which starts the system. Given the test set, the data set that will be input to the system, is 100 items large, the thread pool was given a size of 100, so if a query covering all papers was requested the entire query could be parallelised. This should allow fast execution of this case, and should also cope with several users doing multiple queries simultaneously without much slow down. From testing, using this asynchronous method oppose to sequentially calculating a score for each paper saw some massive speed increases, which can be seen in table 6.1. Once the search is complete and the results ordered, the papers are converted to a `SearchResult` model which contains the key information for the user to see while searching.

Query	Time in seconds (sequential)	Time in seconds (asynchronous)
a	80	23
the	10	3.8
support vector machine	0.5	0.2

Table 6.1: Times for searches to complete, before and after asynchronous calculation of paper scores were added.

The next requirement to implement was the input and processing of all papers. Upon input of a new paper by a user, a new paper record was made. It simply held the location given of the paper and a status of 0. To pick up this new paper, the `PaperProcessor` class has a method `processWaitingPapers`. This method is annotated with `@Scheduled(fixedDelay = 60000)`, which tells Spring to run this method once every 60000 seconds (60 minutes), which is a long time but stops it regularly doing requests to the database making that connection busy, to a database that should rarely have new papers added given web resources are not supported. This method searches for all papers with a given status, one status at a time (starting at 0 and ending at 3). This covers all *in progress* papers. It then applies a *task* to it, which is the processing step from the NLP system. In this sense, it makes maintenance of this system fairly simple, as if there was a need to test another process for a subtask, the class using this process could be listed and `PaperProcess` would apply it automatically. If there was any issue with processing a paper, its status would be set to -1 so it would not be processed again, and would be easy to find as an error paper in the database.

Each *task* was implemented in a class which implemented the `NlpProcessor` interface. The classes created were, in order of processing, `PreProcessor`, `KpSvm`, `ClassW2V` and `RelSvm`. The interface defined three required methods which `PaperProcess` runs in order: `loadObjects()` which loads the required resources for that process to be run, `processPaper(Paper)` which is run on every paper to extract the required information using the components from the NLP system, and finally `unload()` which ensured the memory is cleared to allow for other classes to load in their required resources.

Finally, the information for the data visualisations needed to be prepared. The pie chart, listing the proportions of each classification of key phrase, were deemed simple to compute at run time, as it requires querying the database to find the count of each classification and simple division of that over the total number of key phrases. However, the key phrase word clouds need slightly more computation to display.

For each of the three word clouds, all of the key phrases for that classification need retrieving, tokenizing and their values (count or TF-IDF depending on the test) calculating. This takes some time, which is too long for a user to wait to view them, and will be the same for every query (unless the processing of papers is currently happening). Therefore, a cache was designed. This cache was built under the method `KeyPhraseCloudCache.updateCache()`, which has the special annotation `@PostConstruct`. This annotation tells Spring to run this method after the object has been created and any dependencies added through the `@AutoWired` made available, but before the web server is actually launched. This increases the system start up time, but means the word clouds are available to be send to an interface immediately. To ensure the clouds are up to date if papers are being processed, every time a set is processed under any task, this `.updateCache()` method is called again to regenerate the word clouds.

An important note on the architecture here is the use of TF-IDF, and that it requires a library of papers to be calculated. The training papers are loaded in the mentioned `PaperUtil` class, which also has a method with the `@PostConstruct` annotation which loads this information. These training papers can be retrieved by any other class (mainly for use in training the SVMs as part of the paper processing functionality), but are also used when a TF-IDF value is calculated. The potential problem if the word cloud builder tried to use `PaperUtil` to calculate TF-IDF before the training papers had been loaded in and prepared. However, as `PaperUtil` is a dependency of `KeyPhraseCloudCache`, Spring ensures the post construct method in `PaperUtil` is always ran before that of `KeyPhraseCloudCache` to ensure that any resources are fully loaded and ready to be used.

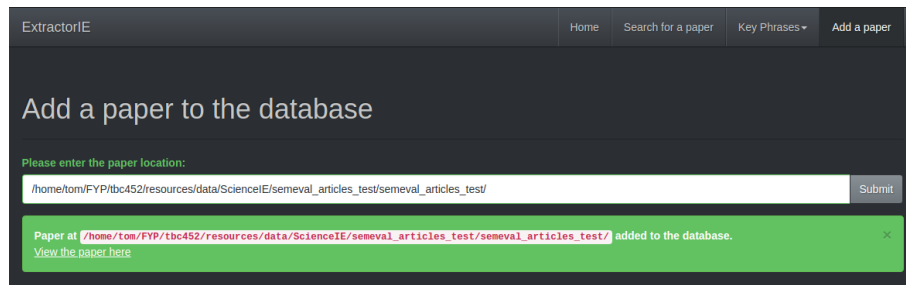


Figure 6.3: The add page of the website, with feedback to the user shown giving indication of a successful paper addition to the system.

6.5 Web Interface

Controller Infrastructure

The web interface covers a series of different pages. Spring, supporting the MVC design pattern, allows for controller classes (annotated with `@Controller`) to load views in the form of JSPs, and pass them model information acquired from other services. In the case of forms where the user has to input some data, a model can be bound directly to the input fields on the page. When submitted, automatic validation (including type checking) can be done by Spring, and the controller can receive the model object for processing. Specified in the controller class, URIs are bound to specific controllers. The controllers can also have different methods for interpreting GET and POST requests, for which different parameters can be specified.

All supported URIs and request methods, with controller, JSP and function provided are listed in table 6.2.

When data is added to a JSP, it can often be directly printed out, however, some items are in lists (such as search results) which cannot be printed straight out. There are also cases where we wish to display one thing or another (for example, a success message or an error message) so there is the need to do *if* statements in the JSP. The JavaServer Pages Standard Tag Library (JSTL) provides all the solutions needed here, as the `forEach` tag can be used to iterate down a list of data, and tags such as `if` and `choose` can be used to show or hide different parts of the page. The JSTL tags need to be imported on every page. An example of this, where it is used to give user feedback on the success of an action, is on the `/add` page, shown in figure 6.3

To ensure all pages have the necessary styling, scripts and navigation bar (which is essential for navigating through the website) are included on every page, the following tags are on every page:

```
<jsp:include page="header.jsp"></jsp:include>
<jsp:include page="navbar.jsp">
  <jsp:param name="active" value="" />
</jsp:include>
```

`header.jsp` and `navbar.jsp` include all of the imports and generic styling every page should have applied.

Styling

In terms of styling, to keep layout simple and text clear to read (as both of the example websites are), rather than building a consistent style from the ground up, Bootstrap¹⁰ was used. To give the website a slightly more unique feel, an on-line customizer was utilised to help quickly develop an aesthetically pleasing and easy to read style¹¹.

To support Bootstrap and custom scripting in the web pages produced, jQuery¹² was also included in the project.

Using Bootstrap allowed for neat and uniform layouts, with full support for both mobile and desktop browsers. This means text scaled correctly so should always be easy to read, along with dynamic resizing

¹⁰Bootstrap is made up of CSS files and JavaScript files, all of which are needed and can be retrieved from <https://getbootstrap.com/>. At the time of development, version 3 was available.

¹¹The *slate* theme was used as a basis on <https://www.bootstrap-live-customizer.com/>.

¹²<https://jquery.com/>

URI	Method	Controller	JSP	Functionality
/	GET	Home	home.jsp	Return the title of the project and displays the pie chart visualisation.
/add	GET	Add	add.jsp	Returns the page to input a paper location on.
/add	POST	Add	add.jsp	Accepts a papers location, and if valid adds it to the database for processing. Returns a message indicating success or failure of the action.
/search	GET	Search	search.jsp	Returns the search input fields.
/search	POST	Search	search.jsp	On submission, validates the search and uses the <code>PaperSearch</code> service to do a search. It then returns the same page as before, with a table listing all results. Clicking on a row loads <code>/view</code> for that paper.
/view	GET	View	view.jsp	It accepts a paper ID as an input. It then loads the paper from the database (including all extraction information) and adds it to the JSP to be returned to the user. It shows the title, main text, and a table with all key phrases in. Relations were not shown due to no good way being decided upon to represent the information, but given the NLP system gives poor results for relations this features was not focussed on too heavily.
/view/download /view/extractions	GET	View	N/A	Given a valid paper ID as an input, this returns a stream of a text file containing the original document or the extractions in BRAT format, respectively.
/kps	GET	KPView	kps.jsp	Returns the shell of the page to display the word clouds on.
/kps	POST	KPView	N/A	Accepting a type parameter, it returns all world cloud information (tokens and TF-IDF values) in a JSON format.
/kps/{type}	GET	KPView	kp.jsp	Returns a page containing a table of all key phrases with a classification of the given <i>{type}</i> . Each row is a link, and takes the user to the <code>/view</code> for its parents paper.
N/A	N/A	Error	error.jsp	This controller handles errors. The returned page includes the HTTP error code (for example 404: not found) and, if debug is enabled in <code>application.properties</code> , a stack trace of what went wrong.

Table 6.2: All available requests that are supported by the web site. The *controller* is the name of the Java class that provides the functionality.

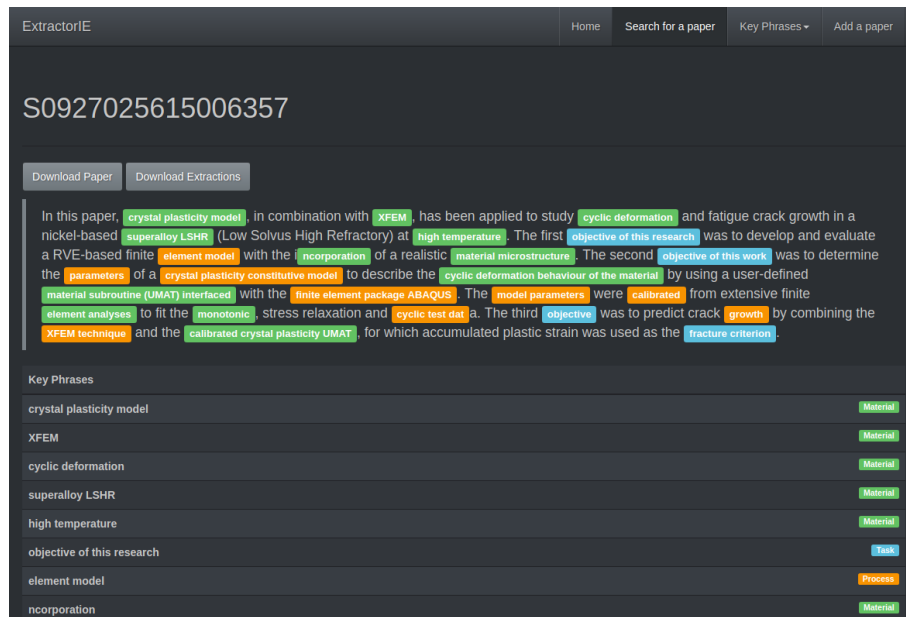


Figure 6.4: The view page, with a paper's content being shown.

of important elements such as tables. Furthermore, Bootstrap has label styling, which was utilised to help highlight the classifications of key words. Throughout the project, Bootstraps' *info* (blue), *warning* (orange) and *success* (green) colours were used for marking tasks, processes and materials respectively. Keeping this consistent styling helps make sure the user can quickly understand new information presented, without having to relearn what graphical components mean. The strongest use of the labels is on the view page, where the text (on a dark background) is greeted with bold highlighted areas where key phrases are, making the key information stand out. This can be seen in figure 6.4.

Data Visualisations

To present the proposed data visualisations, JavaScript solutions were searched for. This would allow the visualisations to be rendered in browser, which reduces the bandwidth required by a image, and can be made to scale depending upon the size of the screen (desktop or mobile).

For the pie chart, it was found the D3.js could be used to create interactive visualisations, and a third party source allowed simple creation of a donut chart¹³. The donut chart generated was set to have overall information in the centre of the chart (including key phrase count and paper count), with the segments being labelled as to the class they represent and the percentage of key phrases they represent. Furthermore, the centre of the chart was made clickable, taking the user to the word cloud page, and the segments were made clickable, taking the user to a list of all key phrases under the given classification. This all results in a lightweight chart that quickly shows some key pieces of information about the contained data, in an interactive way which engages the user. There is even a loading animation included which shows the segments growing, to draw user attention to the visualisation. This is an important chart as it is part of the home page of the system, so being fast to load and show off the information clearly is critical. The segments are also the correct colours to match the rest of the website. The final results can be seen in figure 6.5.

To achieve the word cloud visualisation, D3.js was again investigated, but a simpler solution using jQuery as a base was found. A third party library called jQCloud¹⁴ was found which supported the input of a JavaScript array of objects, where each object has a string field and a value field which is what determines the size of the string field when drawn. This is why the KPView controller has a method which returns JSON, as this called be called and its response piped into the jQCloud constructor function. AJAX was used to call the KPView controller three times, once for each classification, and passed the result, when it arrived, to the JQCloud drawing function which draws the cloud of the page. Links under each cloud to a full list of all key phrases with that classification are also provided. The final product

¹³<https://d3js.org/> is the base technology, with <http://d3pie.org/> being used to create a custom donut chart with ease.

¹⁴<http://mistic100.github.io/jQCloud/>



Figure 6.5: The home page of the website, with the pie chart visualisation.

can be seen in figure 6.6.

The Search Page

The search page consists of a few simple inputs, seen at the top of figure 6.7. It allows for any string to be input, along with a set of check boxes for the user to clearly indicate a focus.

Upon searching, the back end is called which returns a list of **SearchResult** objects, contained in a **SearchResultAndDetails** which holds some extra information. The extra information given includes the number of search results and the time the search took (inspired by the example websites' feedback to the user) which are both displayed just below the search bar (shown on the screen shot). The search results themselves are shown in a table. This has a main column which contains the title of the paper and a snippet. This snippet contains the piece of text in the document (plus or minus a few words) containing the related query term (with the specific words being in bold, again inspired by the example websites). The key phrase and relation counts for the papers listed are also shown on the right. Next to the title of the paper, is a *quick download* button which is a way for the user to skip to the document rather than load the websites `/view` page for that paper. A similar idea is presented for relations (downloading the annotation data) on the far right next to the counts.

While not shown in the screen shot, the list is capped at 10 items. After 10 items have been drawn (if there are more than 10 items to draw), a final row is appended with the text "Show X more", where X is the number of results hidden. Clicking on this row reveals the remainder of the results. This is so the user is not overloaded with a large amount of text and information as soon as the search returns.

6.6 Testing

Testing consists of checking the functionality of the website, as well as evaluating the useful or effectiveness of the implemented requirements.

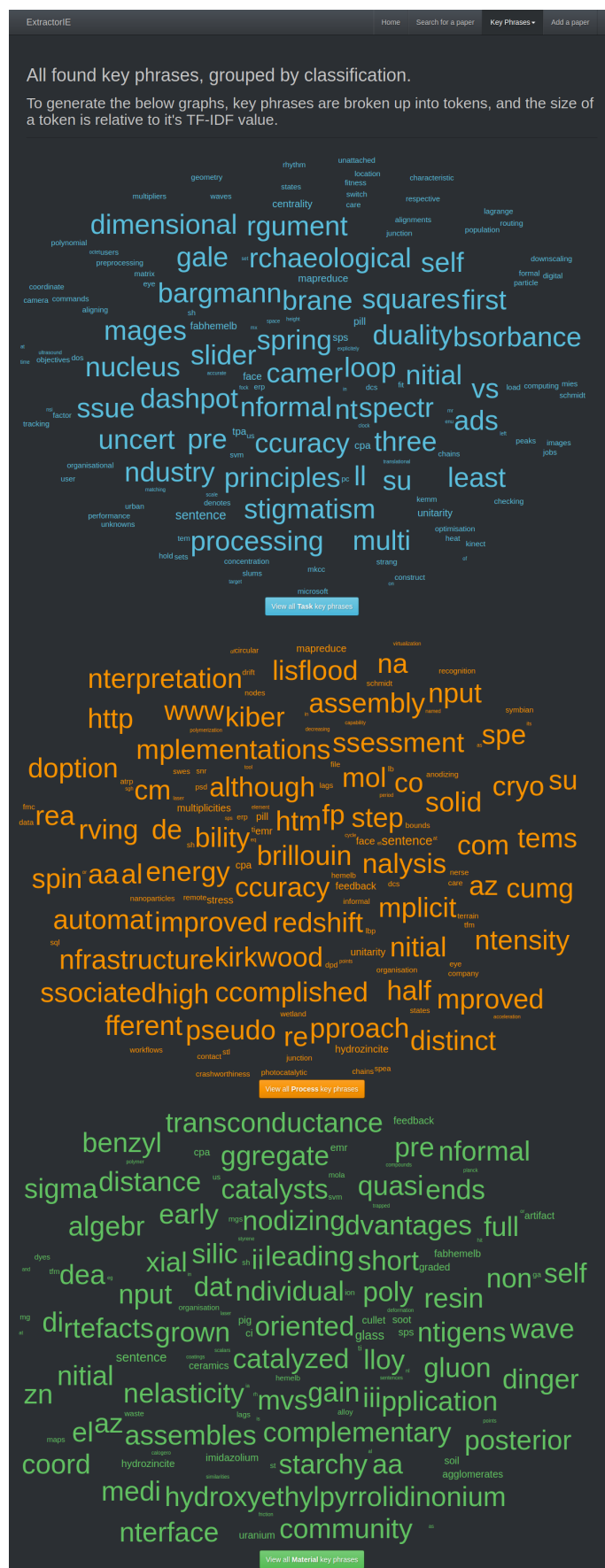


Figure 6.6: The page containing the three word clouds, one for each classification. The value used to determine a words size is the words TF-IDF.

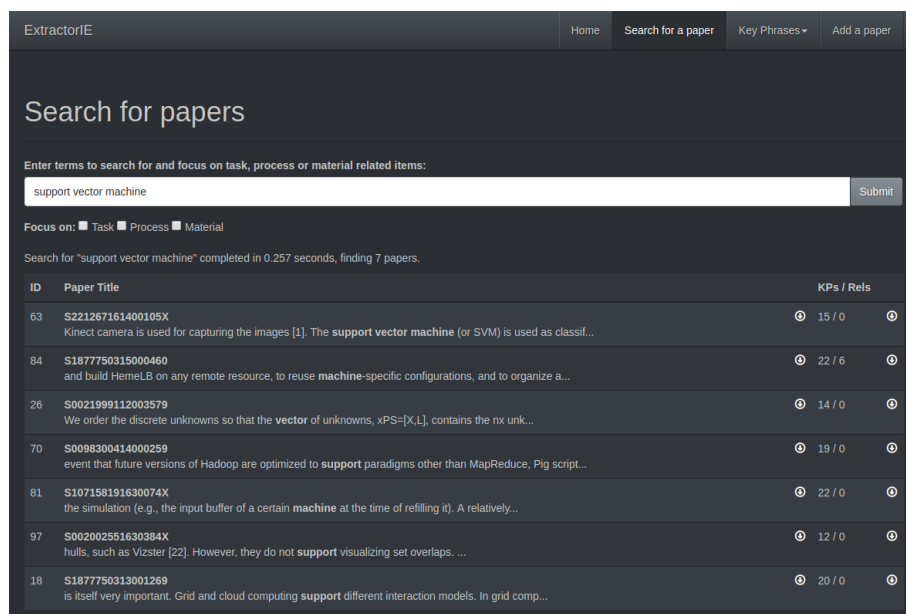


Figure 6.7: The search page of the website, with "support vector machine" queried.

Strict Testing

There are a set of strict ways the websites functionality can be tested. A test plan was constructed, which covered the following aspects:

- All links on all pages take the user to the correct destination, without an error.
- Typing in a random URL returns the correct error page with a 404 message.
- The two input fields accept all English letter and numbers without throwing an error (due to a badly formed string).
- If the targeted result of the input fields (finding papers or adding a record to the database fails), a reasonable error message is displayed.
- When more than 10 papers are displayed at once on the search page, all those after 10 are hidden behind the *view more* button.
- Downloading a `.txt` or `.ann` file always completes without error.
- With respect to the browser window shape/size at load time, both visualisations display readable text and the text does not leave the screen.

The website was brought up to a standard where all of the above were complied with, aside from when downloading `.ann` on macOS with Safari, as it automatically appends `.txt` to the end of it. Unfortunately, this could not be avoided as it was diagnosed as a default feature of Safari on macOS.

In terms of browser compatibility, the above tests were carried out on the following browsers on various platforms: Microsoft Edge (Windows 10, Android), Google Chrome (Windows 10, Android), Firefox (Windows 10, Android) and Safari (macOS).

Requirement Evaluation

In terms of the simple display requirement, this was achieved well. Taking several pieces of information from the example search engines, the list of papers available (as a result of a search) are displayed in a uniform way, with the title and relevant snippets shown. There are links to an internal page for the paper, as well as quick download links. These are very useful when trying to make a collection of papers or key pieces of information, without having to load individual page to view each paper. This is where one downside presents itself, in that each paper's page has to be loaded individually: unfortunately due to the method used to make the links clickable, middle-click cannot be used, so papers can not be opened

in a new tab or window for later viewing. This would be a useful feature to include, as it would allow a user to make a backlog of papers they are interested in reading while they continue to browse.

Furthermore, with regards to the paper display pages, the overall conveying of information is very clear, with boldly highlighted key phrases, and a listing for even more clarity is placed below. Unfortunately, hyponyms and synonyms were not annotated, and while it would have a positive point to have made them available, given the NLP system being used a library to generate this information struggles to make many or high quality relations, not showing this information in a graphical way has no major impact on the user. All of this information is still available in the annotation download. The solution implemented was appropriate for the length of papers being shown, however, this would not be suitable if using full papers instead. If full publications were being displayed, the generated web page would be extremely long, and the design of this may need to be reconsidered.

In terms of search, the overall outcome was positive but not definitive. If the user completed a query, and then changed the focus, given suitable selection of papers were originally found, the order would change. This was verified as being a suitable change by individually checking the key phrases of the papers listed and ensuring the correct classifications of key phrases (as searched for) were being promoted. However, it is hard to definitively call this a success, as to properly test this ideally many hundreds or thousands papers would have been in the database and processed. This would allow for a greater selection of examples to search for and experiment with, and when focussing the effect of different weightings of key phrases could have been investigated further. The use of asynchronous execution when conducting the search was a sensible addition, as it significantly decreased the search time for most queries, however, the time still needed to be displayed to the user as system feedback as some searches could still take some time.

Furthermore, the search results page was built with the size of the data set in mind. With a maximum of 100 results that could be returned, simply hiding those past the initially readable limit was reasonable for this project. However, that cause significantly problems in terms of data transfer and browser stability if the document pool was scaled up to thousands of records, which could overload the user and the browser. Ideally, pagination should be implemented to cope with many more documents, allowing the user to progress through small selections at a time.

The functionality to support adding papers to the system for automatic processing, and that processing being conducted, was a great success. While the feature didn't support web resources (which would not have been suitable as generally they are quite long), it effectively used the NLP system from earlier in the project to build the database with all of the predicted information. The pending documents would automatically be retrieved from the database and the changes committed, including the extra synonyms found from the sanitisation additions from subtask A's SVM. The only issue that kept occurring, especially when completing processing for subtask C, was that the program kept running out of allocated memory. This caused an exception to be throw, and left a paper at a time in an *error* state, while the error was not caused by the paper itself but the system encountering a problem. Furthermore, in the case of subtask C, if it ran out of memory while processing one paper, it would like do the same for the follow up papers in that instance. Therefore, the code was added to catch memory exceptions, and not run the final subtask again if it crashed once. Otherwise, the system performed as expected, and the data could immediately be viewed on the website. A small addition made to the paper view page was a progress bar being shown if a paper was still undergoing processing (which filled up as it progressed through each stage of processing).

Finally, the two visualisations created were effective. The first was placed on the home page of the website, which quickly gave new users an indication of the information contained. This linked through to the second page, giving the website a good flow. The cursor would also change to a *hand* (for selecting items) when it was moved across the visualisation, helping to indicate it could be pressed. The word cloud visualisations were less useful, although still interesting to see. As each key phrase was tokenized, no selection of words shown made any grammatical sense, which detracted from the concept of giving users ideas of the key pieces of information selected in this system, as few complete piece of information was shown as one string. Furthermore, due to the removal of odd symbols, some strings occasionally looked odd, for example some were missing letters, or were the combination of two words (where they would have been separated by some symbol, but that symbol was removed). The two values for the words proposed were both examined, and (as predicted) TF-IDF functioned better. The TF-IDF value is used in figure 6.6, which clearly has lots of large words at the centre of each cloud which are unusual words, with more common words being smaller and surrounding these big words. Basing the size of words on their occurrence count merely made most of the words small, with very common and less useful words like "the" and "of" very large at the centre, shown in the example in figure 6.8.

Chapter 7

Conclusion

Overall, the solutions presented to the ScienceIE problem do not succeed in beating the solutions proposed as part of the task, however, do explore other idea about how to achieve them. Word2Vec usage in particular was explored and utilised through the solutions provided to all three subtasks. A more suitable model for Word2Vec would have been useful and likely boosted the performance of the systems. The reason for creating these systems is to extract classified key pieces of information from scientific papers, and a use for this was proposed and demonstrated in the proof-of-concept website produced. This used key phrase information when completing search to try to prioritise relevant papers to a users query, which was reasonably fast and accurate, given the data available.

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>
- Aswani Kumar, C., Radvansky, M. & Annapurna, J. (2012), ‘Analysis of a vector space model, latent semantic indexing and formal concept analysis for information retrieval’, *Cybernetics and Information Technologies* **12**(1), 34–48.
- 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.
- Goldberg, Y. & Levy, O. (2014), ‘word2vec Explained: deriving Mikolov et al.’s negative-sampling word-embedding method’, (2), 1–5.
URL: <http://arxiv.org/abs/1402.3722>
- 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>
- Honnibal, M. & Johnson, M. (2015), ‘An Improved Non-monotonic Transition System for Dependency Parsing’, *Emnlp 2015* (September), 1373–1378.
URL: <http://aclweb.org/anthology/D15-1162>
- Landauer, T. K., Foltz, P. W. & Laham, D. (1998), ‘An introduction to latent semantic analysis’, *Discourse Processes* **25**(2-3), 259–284.
URL: <http://www.tandfonline.com/doi/abs/10.1080/01638539809545028>
- 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., 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. & Sætne, 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>

- Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013), ‘Efficient Estimation of Word Representations in Vector Space’, pp. 1–12.
URL: <http://arxiv.org/abs/1301.3781>
- Nayak, N. (2015), ‘Learning Hypernymy over Word Embeddings’, *CS224N Projects* pp. 1–8.
- Page, L., Brin, S., Motwani, R. & Winograd, T. (1998), ‘The PageRank Citation Ranking: Bringing Order to the Web’, *World Wide Web Internet And Web Information Systems* **54**(1999-66), 1–17.
URL: <http://ilpubs.stanford.edu:8090/422>
- Pennington, J., Socher, R. & Manning, C. (2014), ‘Glove: Global Vectors for Word Representation’, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* pp. 1532–1543.
URL: <http://aclweb.org/anthology/D14-1162>
- 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>
<http://dx.doi.org/10.1016/j.jbi.2015.02.004>
- 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

Appendix C

Stop Words List

Below is the list of stop words used in this project:

!!	...	does	in	she's	we're	dont
?!	'll	doesn't	into	should	we've	hadnt
??	's	doing	is	shouldn't	were	hasnt
!?	'm	don't	isn't	so	weren't	havent
'	a	down	it	some	what	hes
"	about	during	it's	such	what's	heres
"	above	each	its	than	when	hows
-lrb-	after	few	itself	that	when's	im
-rrb-	again	for	let's	that's	where	isnt
-lsb-	against	from	me	the	where's	its
-rsb-	all	further	more	their	which	lets
,	am	had	most	theirs	while	mustnt
.	an	hadn't	mustn't	them	who	shant
:	and	has	my	themselves	who's	shes
;	any	hasn't	myself	then	whom	shouldnt
"	are	have	no	there	why	thats
'	aren't	haven't	nor	there's	why's	theres
?	as	having	not	these	with	theyll
<	at	he	of	they	won't	theyre
>	be	he'd	off	they'd	would	theyve
{	because	he'll	on	they'll	wouldn't	wasnt
}	been	he's	once	they're	you	were
[before	her	only	they've	you'd	werent
]	being	here	or	this	you'll	whats
+	below	here's	other	those	you're	whens
-	between	hers	ought	through	you've	wheres
(both	herself	our	to	your	whos
)	but	him	ours	too	yours	whys
&	by	himself	ourselves	under	yourself	wont
%	can	his	out	until	yourselves	wouldnt
\$	can't	how	over	up	###	youd
@	cannot	how's	own	very	return	youll
!	could	i	same	was	arent	youre
^	couldn't	i'd	shan't	wasn't	cant	youve
#	did	i'llv i'm	she	we	couldnt	
*	didn't	i've	she'd	we'd	didnt	
..	do	if	she'll	we'll	doesnt	

Google Scholar and ScienceDirect Search Pages

The following are screen shots of the query "computer science" being queried in two popular search engines: Google Scholar (left) and ScienceDirect (right). The "... " denote some of the search results has been cropped out. As cropping has been applied, the results presented by Google Scholar and ScienceDirect are 10 and 25 (configurable to 50 or 100) respectively.

Google Scholar

Articles About 5,610,000 results (0.05 sec)

Any time
Since 2015
Since 2014
Custom range...

Sort by relevance
Sort by date

☒ Include patents
☒ Include citations

Create alert

computer science

[book] The design and analysis of spatial data structuresAddison-Wesley Series in Computer Science
H.Licet: 1990 - ignorenow updog co
By Heman Samet. The Design and Analysis of Spatial Data Structures (Addison-Wesley series in computer science) 0201502550 the design and analysis of spatial data structures addison wesley series in computer science by heman samet download and read the design ...
☆ 90 Cited by 3767 Related articles All 8 versions 30

[jrn] Concrete mathematics : a foundation for computer science
R.L.Gabhart, DE.Knuth, O.Patashnik, S.Liu - Computers in Physics, 1989 - also scitation.org
For \$79 MicroMath's GRAPHI program provides you with most of the plotting features found for many more expensive packages. Included are several built-in empirical curve generating routines (interpolating or least squares splines, polynomials, etc) and data may be...
☆ 90 Cited by 1717 Related articles All 17 versions 30

[book] Principles of Compiler Design (Addison-Wesley series in computer science and information processing)
BYAbdo Z.Ullman - 1977 - [alsovintage updog co]
By Alfred V.Aho, Jeffrey D.Ullman: Principles of Compiler Design (Addison-Wesley series in computer science and information processing) browse and read principles of compiler design addison wesley series in computer science and information processing principles of ...
☆ 90 Cited by 1942 Related articles All 7 versions 30


[scitation] The implementation of functional programming languages (prentice-hall international series in computer science)
SI.Peyton Jones - 1967 - dcm org
The Implementation of Functional Programming Languages (Prentice-Hall International Series in Computer Science). Author: Simon J. Peyton Jones, Publication: Book, The Implementation of Functional Programming Languages (Prentice-Hall International Series ...
☆ 90 Cited by 1051 Related articles All 3 versions 30

[scitation] Probability & statistics with reliability, queuing and computer science applications
KS.Triand - 2008 - John Wiley & Sons
☆ 90 Cited by 4133 Related articles All 16 versions 30

Related searches

- mathematics computer science lecture notes
- computer science curriculum
- computer science foundations
- computer science university of southampton
- software engineering computer science
- computer science software credits
- procedia computer science

- computer science lecture notes
- computer science seminar
- computer science artificial intelligence
- electrical engineering computer science
- introductory computer science
- computer science school of electronics
- computer science information and software
- computer science courses



Next

1 2 3 4 5 6 7 8 9 10

ELSEVIER | About ScienceDirect | Remote access | Shopping list | Contact and support | Terms and conditions | Privacy policy

Content not for sale. For more information, visit our website page.

Copyright © 2018 Elsevier B.V., its licensors or contributors. ScienceDirect® is a registered trademark of Elsevier B.V.

Appendix E

How to Run the Project from Source

To run the two applications, the following prerequisite steps must be taken:

1. Install Java 8, Maven 3 and MySQL.
 - Running `./install_*.sh` in `resources/scripts` will do this for a system with `apt-get` available, such as Ubuntu.
2. Run the `env_maven_opts.sh` script in `resources/scripts` to setup required system environment variables.
3. A Word2Vec model is needed. Based on this report, the pre-trained Google News model¹ is recommended for download.

Some files required by the system need pointing to. In terms of training and testing paper locations, checking the `paper.txt` and `test_papers.txt` in the resource folders of both Java projects should reveal the paths used on the development system. The paths listed in this file are all processed unless they begin with a `#`, and either point to a text file for processing, or a directory containing multiple text files for processing. Change these appropriately.

Other components are more embedded in the Java code. Ideally a better solution to this should have been included by the developer, but unfortunately other tasks used up this time. Change `log4j2.xml:8` and `NlpObjectStore.java:23` to any folder where logs and serialised files can be saved. Change `Word2VecPretrained.java:11` to point to a Word2Vec model (either the Google News model or a model renamed to the same file descriptor).

To setup the MySQL database, navigate to `resources/sql`. Run, in order, `./setup.sh` and `./build.sh`. If you wish to use the final data made in this project run `./recover.sh`. Depending upon your MySQL configuration, you may need to change the password to be stronger. Finally, update `application.properties` in the FYP-GUI project's resource folder with the correct connection string and password.

Navigate to the root of the two Java projects (both in `java/`), and a series of useful scripts for compiling and running are presented:

Java Project	Script	Description
FYP-NLP	<code>./build.sh</code>	Compiles the system, without running any tests.
FYP-NLP	<code>./install.sh</code>	Compiles the system, without running any tests, and installs the project to the local Maven repository (required for building the FYP-GUI project).
FYP-NLP	<code>./test.sh <test class></code>	Compiles the system and runs the JUnit test class given. Test classes can be found in <code>java/FYP-NLP/src/test/java/xyz/tomclarke/fyp/nlp</code> , and specifying the name of the class will run all test methods inside it without <code>@Ignore</code> above the <code>@Test</code> annotation.
FYP-GUI	<code>./build.sh</code>	Compiles the GUI and runs all JUnit tests.
FYP-GUI	<code>./build-and-run.sh</code>	Compiles the GUI, without running any tests, and launches the GUI.
FYP-GUI	<code>./run.sh</code>	Launches the GUI (assumes it is already built).

¹<https://drive.google.com/file/d/0B7XkCwpI5KDYNNUTTISS21pQmM/>