UNIVERSITY OF BIRMINGHAM

Final Year Project

Extracting Key Phrases and Relations from Scientific Publications

Dissertation for B.Sc in Computer Science

School of Computer Science, University of Birmingham

Author
Thomas Clarke (1443652)

Supervisor
Dr Mark Lee

April 2018

Declaration

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

Acknowledgements

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

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

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

Abstract

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

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

Scores of NLP go here

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

Keywords

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

Contents

List of Figures

List of Tables

Chapter 1

Introduction

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

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

1.1 Aims and Objectives

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

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

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

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

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

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

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

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

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

1.2 Report Outline

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

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

Chapter 2

Background and Literature Review

2.1 Definitions and Descriptions

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

F1 Score

Tokenization

Bag-Of-Words Representation

Term Frequency - Inverse Document Frequency (TF-IDF)

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

Support Vector Machine (SVM)

Neural Networks

Conditional Random Field (CRF)

Clustering

2.2 ScienceIE Proceedings

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

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

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

– Needs rewriting after this bit –

The best end-to-end ScienceIE team used a long short-term memory (LSTM) approach for phrase extraction, with labelling completed by a CRF based sequence tagging model (Ammar, Peters, Bhagavatula & Power 2017). Their sequence tagging model employed gazetteers built from scientific words extracted from the web. Another team (Marsi, Sikdar, Marco, Barik & Sætre 2017) also had similar ideas, using CRFs to complete some of the task using WordNet[1] as a data source for the classifier they created. Both teams here also used sensible rules to help improve their score, such as intuitively marking

[1] https://wordnet.princeton.edu/

all instances of a key phrase as a key phrase upon finding one instance (so if *carbon* is extracted and labelled as a *material*, then all other instances are labelled to match) and exploiting hypernym relationship's bidirectional property (so if word 1 is a hypernym of word 2, the reverse is also true and therefore recorded).

Unfortunately, while extraction and classification were generally well handled, relation extraction has very low accuracy across all teams taking part with the average F1 score only being 0.15, with the highest score being 0.28. (Ammar et al. 2017) achieved this using gazetteer built from Wikipedia[2] and freebase. This seems more appropriate than hand written rules, which may seem appealing as they can be somewhat tailored and provide high accuracy, but require much more effort from the developer and may not work well on unseen conditions providing low accuracy (Manning & Jurafsky 2012). As mentioned earlier, WordNet is also a potential source of information for building a classifier for relation extraction, and a study by (Snow, Jurafsky & Y. Ng 2013) compared building a classifier off of Wordnet and Wikipedia for hypernym-only extraction. The result of this shows that Wikipedia may be more suited to creating this type of classifier as it achieved an F1 score higher than using WordNet (the Wikipedia based classifier got 0.36 while the WordNet based classifier got 0.27). While an improvement, it is not ultimately a huge increase and there is no evidence either Wikipedia is better for the specific area of scientific papers (as the 2013 study was completed on a generic set of data).

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

2.3 Other Revelant Background Information

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

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

2.4 Word2Vec

2.5 The Supplied Data Set

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

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

[2]https://en.wikipedia.org

	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.

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

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

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

Chapter 3

Analysis and Specification

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

3.1 Project Architecture

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

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

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

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

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

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

log4j

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

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

[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/

to process data, which can be considered when going forward; for instance in terms of formulating efficiently timed tests plans.

- Labelling of logs into levels such as *debug*, *info*, *error* and *fatal* messages: This can be used when analysing the logs to catch where things went wrong (filtering for error messages) and then to try to debug the system by finding information logged prior to that (with debug). During development an excellent use of this feature is to output all levels aside from 'debug' to terminal, so monitoring progress isn't overloading the executor with information, but if something does go awry the steps leading up to the bad event can be analysed in the log saved to disk.

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

Maven

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

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

- Project compilation: Maven can be used to build a project and automatically run specified or all tests, with more detailed and well formatted output than compiling Java code by hand. Therefore, compilation and testing can more easily be scripted and output more clearly analysed. It also handles importing libraries used in a Java project when compiling (which can be very troublesome when completed by hand), which is discussed below.

- Library import: The `pom.xml` can specify dependencies of the Java project. While custom, third party repositories exist, Maven has a central repository[6] with many libraries available. This includes log4j described above, and all other libraries used in this project. Dependencies are downloaded to the systems local Maven repository at compile time.

- Library export: As discussed in the introduction, the NLP system shall be used in a POC system. Rather than combining these two systems into one large package, or doing a confusing copy of the required resources, Maven can be used to export the compiled NLP system to the local Maven repository. Then, the POC system can simply list the NLP system as a dependency, and Maven shall include it as a library when building the executable program.

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

JUnit

Word2Vec

[5] https://maven.apache.org/
[6] http://repo.maven.apache.org/maven2/
[7] https://logging.apache.org/log4j/2.x/maven-artifacts.html

Chapter 4

The ScienceIE Task: Design and Implementation

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

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

4.1 Data Preprocessing

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

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

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

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

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

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

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

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

4.2 Subtask A - Key Phrase Extraction

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

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

4.2.1 Method 1: Support Vector Machine

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

Processing Data

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

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

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

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

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

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

Defining Support Vectors

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

Thankfully, the idea behind using an SVM is to find what separates key phrases from just normal phrases. Therefore, I was able to create an initial range of support vectors, as defined in table 4.1. Here it is evident most support vectors 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 data points, each of which holds a set of support vectors. Each of these data points must be labelled. The label is what we are trying to predict on the test data, so here the label is where or not the token is a key phrase (`0` for *normal*, or `1` for key phrase).

Model Selection

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

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

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

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

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

Development

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

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

Cross Validation

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

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

4.2.2 Method 2: Clustering

Concept

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

Inspired by this, this paper proposes a similar process, where the similarity of terms is based around their Word2Vec similarities. Conceptually, it is possible that gathering similar terms will have the effect of creating clusters of important concepts, from which key phrases can be extracted. With the most similar words at the center, 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

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

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 clusteing algorihm selected was hierarchial clustering (Rai & Singh 2010). Bottom up (agglomerative) hierarchial 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 reltively straight foward clustering algorithm to implement, so should allow results to be found quickly.
- While the term similarities are based off of Word2Vec similarities, the distances between clusters could be evaluated in a number of ways. These include *single* (the shortest distance between terms in each cluster), *average* (the average distance between each term in one cluster to each term in another) and *complete* (the largest distance between terms n each cluster). These are called the *linkage criteria*.
- If successful, the benefits of using clustering could be two fold. Not only might this produce effective key phrases, but it may even aid in classification. Once a number of clusters have formed key phrases, the hierarchial 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 classifiaction, k-means clustering may be more appropriate with a $k = 3$ where each cluster would be a different classifiaction.

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

A large difference between this and SVM usage is that this method is unsupervised learning and does not require training data, while using an SVM is supervised learning. This means that, given this method works for this testing scenario, its application may scale better in the *real world* as is it not tied to the quality of the training data. Furthermore, given a suitable WOrd2Vec model, differences in key phrase output may be seen. This means that this algorithm may suffer here given the Word2Vec models currently available are not based on scientific publications, but running this algorithm on test data with similar context to the Word2Vec model may improve things, which means it may even cope with different languages.

Development

The process of turning this theory into a practical implementatin 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, all stop words and unimportant words are removed. Unimportant words are calculated based on thresholding TF-IDF scores. This threshold was manually set, by generating a list

of all tokens and their TF-IDF scores and evaluating where the cut off would be to remove all words not in any key phrase. For example, "WORD" has a low TF-IDF score of SCORE, but is in the key phrase "WORDS".

Then, the process of hierarchial 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 evalauted on a sample of the processed documents (as manually reviewing all 100 test documents cluster patterns accross 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: Word2Vec

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

4.4 Subtask C - Relation Extraction: Support Vector Machine

A section all about what I did for part 3

4.4.1 Many Support Vectors

Discuss the SVM I tried to do this with (including Word2Vec)

4.4.2 Few Support Vectors

Chapter 5

The ScienceIE Task: Evaluation

How each section went, including test results and maybe some info on other experiments.

5.1 Subtask A - Key Phrase Extraction

5.1.1 Method 1: Support Vector Machine

5.1.2 Method 2: Clustering

5.1.3 Conclusion

5.2 Subtask B - Key Phrase Classification

5.2.1 Other Experinmentation

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

5.2.2 Conclusion

5.3 Subtask C - Relation Extraction

5.3.1 Conclusion

Chapter 6

Creating a Proof of Concept Use for ScienceIE data

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

6.1 Further Research

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

6.2 Design and Implementation

How it was pulled off

6.3 Web Interface

Exactly what was achieved

6.4 Testing

(Get) user feedback

6.5 Conclusion

Overall impact of the GUI on the project

Chapter 7

Discussion

Talk about overall results

7.1 Improvements and Extensions

Bibliography

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

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

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

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

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

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

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

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

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

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

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

Appendix A

Example ScienceIE Training/Test Document

The following is ScienceIE test paper file S0010938X15301268.txt:

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

Appendix B

Example ScienceIE Training/Test Annotation Data

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

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