Semantic Word Association:

Creating the Semantic Bi-Gram

Jordan Aitken

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BEng (Hons) Software Engineering

School of Computing

April, 2017

Authorship Declaration

I, Jordan Aitken, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:

Date: 23/04/2017

Matriculation no: 40136554

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

Natural Language Processing is a field in computer science that deals with how well computers can “understand” natural languages such as English. Computers use language models to look for patterns in languages, which it can use to evaluate sentences that are given to it.

Currently, the most common language model is the n-gram. It looks at sequences of words and uses the information it learns to assign a probability or perplexity value to a sentence it is given.

Unfortunately, there are many issues with the n-gram model, specifically in regards to how it deals with word sequences it has not seen before, or long sentences. The n-gram model is fairly weak when evaluating sentences it has not seen before.

This project introduces an alternative language model that looks at semantics of words, and the association between them and other words. This solves some of the shortcomings of n-grams.

The overall goal of this project is to create a language model that can evaluate the perplexity of a sentence it has never seen before, this can then be used for many purposes, such as identifying the best suitable match for an unfinished sentence.

Contents

List of Figures

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. John Owens, for his constant support and guidance throughout my Honour's project, as well as his weekly feedback and advice of what to do to progress in the project. Thanks to John, the project was always moving forward, and I was able to achieve very significant progress towards my goal in this project.

John also introduced me to Natural Language Processing, which I found very interesting and a very engaging topic to research.

I would also like to thank Malcolm Rutter, my secondary marker, for providing me with constructive feedback at the interim review meeting. It highlighted that at the time I was behind, and it motivated me to catch up and get back on track.

Finally, I would also like to thank my family and friends for their support and begrudgingly listening to my progress throughout the project, and offering advice where possible. They kept me focussed on the project and offered help where they could.

1 Introduction

1.1 Project Background

Natural Language Processing (NLP) is a field of computer science that is concerned with the interaction of computers and human (natural) languages. It deals with being able to “understand” the languages by training itself using a large body of text known as a corpus.

One of the most common ways of training software is to use the N-gram language model. This model looks at the frequency of sequences of words, and uses this frequency to evaluate a sentence and grant it a probability rating (based on how often it has seen the sequence in the past).

However, there are many limitations of the N-Gram language model. Namely, the model has difficulties with range, and it can only understand sequences it has seen before.

The drawbacks of the N-gram language model leave the possibility of creating a new language model that can have a deeper “understanding” of words at a semantic level, without the need for relying on the model having already seen a sequence of words to evaluate it.

1.2 Project Goal

In response to the limitation of the N-gram language model, I have opted to create a new language model that looks at the semantics of words, rather than sequences of them.

The goal of this project is to create the Semantic Bi-Gram (SBigram) language model. The SBigram looks at the permutation of words in a sentence, rather than sequences. Unlike the n-gram model, the SBigram is not concerned with the order that words appear in, but how often they occur with words in the same sentence.

For example, using the n-gram language model, the software would recognise these two sentences as completely different.

The big man.

The man was big.

Figure 1: Example Sentences

However, the SBigram would recognise that both of these sentences mean the same thing, and would therefore evaluate them as being very similar.

A major goal for the project is to create a language model that can evaluate a sentence that has never seen before.

1.3 Project Objectives

The key objectives of the project are as follows:

1. Read in a large text file (the corpus), ready to be parsed
2. Split the corpus into sentences
3. Remove unwanted words from the sentences (these are words that have little semantic meaning. For example: as, a, but, between, the, where, was)
4. Calculate the unigram counts of the corpus (how often each word occurs) and store this information in a TreeMap, as well as storing this information as a file.
5. Calculate the semantic bi-gram counts, by storing the occurrences of each permutation of words, and store this information in a TreeMap, as well as storing this information as a file.
6. Use the values of the semantic bi-gram counts to calculate the probability of a sentence the language model has not seen before.

1.4 Project Outline

The project outline is as follows:

2. Literature Review

The literature review aims to look at information and knowledge that must be researched before work is begun on the project in the field of Natural Language Processing.

3. Requirements Analysis

The requirements analysis sets a foundation for the project to be started with, it looks at the requirements of the project, both functional and non-functional as well as programming environment and corpora requirements.

4. System Design

The system design shows the intended infrastructure of the software, as well as the flow and design. This builds on the foundation as the project prepares for the implementation stage

5. Methodology

The methodology describes how the software has been implemented, it looks at the functions of the program, and how the entire system works.

6. Testing

The testing stage reviews how well the software has been tested, to provide information and tests the reliability and robustness of the software.

7. Results and Evaluation

After testing, the language model will be tested using a large set of test data to identify how well it performs with different types of input.

8. Self-Evaluation

The self-evaluation stage looks at how I, as the student, feel I performed and what I could have done better.

9. Conclusion

The conclusion looks at the project as a whole, and how well it meets the objectives.

10. Future Work

Although the language model is completed, there are many areas that can be continued to be worked upon to further increase the performance of the software.

11. References

All citations and references will be listed here.

12. Appendices

The appendices contains all reference material that are relevant to the project.

2 Literature Review

2.1 What is Natural Language Processing?

Natural Language Processing (NLP) is investigation into how computers "understand" the written or spoken language used by humans. In this project, we will be using the English language.

2.2 Natural Language Generation

Natural Language Generation (NLG) is a subfield of artificial intelligence and computational linguistics that is used in producing understandable text in English or other human languages. (Dale & Reiter)

2.3 Language Models

Language models play a key part in NLP; computers do not have a grammatical understanding of the English (or other) language like humans do. Therefore, a language model is used to evaluate the probability of a sentence, and assign a value as to how probable it is that that sentence would occur. E.g. does it make sense?

2.3.1 What are Language Models?

A language model is a way to evaluate the probability of a sentence, or sequence of words. It assigns a value based on the relative likelihood of the sentence or phrase occurring. This is very useful for many language constructs such as speech recognition, machine translation, part-of-speech tagging, parsing, handwriting recognition and information retrieval. (Language Models, 2016)

2.3.1.1 N-Grams

An n-gram is a language model that counts the occurrence of words in a sentence, or sequence of words. It is represented by a value of n. For example, unigrams, bigrams and trigrams.

(Fletcher, 2011)

2.4 N-Grams

2.4.1 What are N-Grams?

N-gram based techniques are predominant in modern natural language processing (NLP) and its applications. Traditional n-grams are sequences of elements as they appear in texts. These elements can be words, characters, POS tags, or any other elements as they encounter one after another in texts. Common convention is that “n” corresponds to the number of elements in a sequence. (Sidorov, Velasquez, Stamatatos, Gelbukh, & Chanona-Hernández)

2.4.2 How are N-Grams used?

An n-gram counts the occurrence, and calculates the probability of a set of n elements, where n is a positive integer.

2.4.3 Unigrams, Bigrams, Trigrams….

Some n-grams have names that they are commonly referred to as:

Unigram:	n =1,
Bigram:	n = 2
Trigram:	n = 3,
Four-gram:	n = 4

…etc.

<s> represents the start of a sentence

</s> represents the end of a sentence.

“<s> The man walked to the supermarket </s>”

Figure 2: Example Sentence

1-gram (Unigram)	<s>, The, man, walked, to, the, supermarket, </s>
2-gram (Bigram)	<s> The, The man, man walked, walked to, to the, the supermarket, supermarket </s>
3-gram (Trigram)	<s> The man, The man walked, man walked to, walked to the, to the supermarket, the supermarket </s>

Figure 3: N-gram model

(Chambers, Tetreault, & Allen)

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

Figure 4: N-gram equation

Example Corpus

“I would not like them
here or there.
I would not like them
anywhere.
I do not like
green eggs and ham.
I do not like them,
Sam-I-am.”

Figure 5: Example Corpus - Dr. Seuss

(Seuss, 1960)

2.4.3.1 Unigrams

A unigram is an n-gram model where n is equal to one. This is useful for counting the occurrence of a single element in a corpus. For example, a word, punctuation mark or a part of a word.

$$\boxed{P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)}$$

Figure 6: Unigram model equation

2.4.3.2 Bigrams

A bigram is an n-gram model where n is equal to two. This can be used to count how often two words appear next to each other. For example, the number of times the words “man walked” appear next to each other can be counted.

$$\boxed{P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})}$$

Figure 7: Bigram model equation

2.4.4 Advantages and Disadvantages

2.4.4.1 Advantages

Effective

The n-gram model is effective, as it can be used on a corpus of any size, and the accuracy of the n-gram model will increase the larger the corpus is.

Low weight

The n-gram model isn’t very intensive, all data is stored in text with low levels of complexity. However, a lot of text is stored, but it can be organised to be readable by humans.

2.4.4.2 Disadvantages

Long-Range Dependencies

Language often has long-range dependencies, which means that an n-gram only acknowledges words that are within its range. For example:

"The cup that was on top of the cupboard in the bedroom fell."

The language model is unable to link the words "cup" and "fell" in any n-gram where n < 12, which is an unusually high value. (Jurafsky)

Word Order

Using n-grams, word order is important, and the model relies heavily on the order of the words. This can lead to inaccuracies and missed information. For example, "the man and the boy" and "the boy and the man" may return different probability values.

Corpus Reliance

N-grams only work when used with a corpus. The language model must have seen an occurrence of a certain sequence of words in the corpus, or it will be assigned a probability of zero, no matter how much sense the sentence might make.

2.5 Semantic N-Grams

2.5.1 What is the Semantic N-Gram Model?

The semantic n-gram (sngram) model will be a language model that looks at how often words occur together in a sentence, and will

2.5.2 Why use SN-Grams?

Although n-grams are useful and we can often get away with using them, they are flawed, especially with long-range dependencies. For example, using a bigram model, given the text "The big orange cat fell" the model would fail to recognise that "big" and "cat" are related.

2.6 Regular Expressions

2.6.1 What are Regular Expressions?

Regular Expressions, or RegEx, are used in text parsing. A regular expression is a special text string that is used in identifying certain patterns such as a postcode or e-mail address. A list is shown below which shows the capabilities of regular expressions.

<table>
<tr><th>expression</th><th>matches...</th></tr>
<tr><td>abc</td><td>abc (that exact character sequence, but anywhere in the string)</td></tr>
<tr><td>^abc</td><td>abc at the beginning of the string</td></tr>
<tr><td>abc$</td><td>abc at the end of the string</td></tr>
<tr><td>a|b</td><td>either of a and b</td></tr>
<tr><td>^abc|abc$</td><td>the string abc at the beginning or at the end of the string</td></tr>
<tr><td>ab{2,4}c</td><td>an a followed by two, three or four b's followed by a c</td></tr>
<tr><td>ab{2,}c</td><td>an a followed by at least two b's followed by a c</td></tr>
<tr><td>ab*c</td><td>an a followed by any number (zero or more) of b's followed by a c</td></tr>
<tr><td>ab+c</td><td>an a followed by one or more b's followed by a c</td></tr>
<tr><td>ab?c</td><td>an a followed by an optional b followed by a c; that is, either abc or ac</td></tr>
<tr><td>a.c</td><td>an a followed by any single character (not newline) followed by a c</td></tr>
<tr><td>a\.c</td><td>a.c exactly</td></tr>
<tr><td>[abc]</td><td>any one of a, b and c</td></tr>
<tr><td>[Aa]bc</td><td>either of Abc and abc</td></tr>
<tr><td>[abc]+</td><td>any (nonempty) string of a's, b's and c's (such as a, abba, acbabcacaa)</td></tr>
<tr><td>[^abc]+</td><td>any (nonempty) string which does not contain any of a, b and c (such as defg)</td></tr>
<tr><td>\d\d</td><td>any two decimal digits, such as 42; same as \d{2}</td></tr>
<tr><td>\w+</td><td>a "word": a nonempty sequence of alphanumeric characters and low lines (underscores), such as foo and 12bar8 and foo_1</td></tr>
<tr><td>100\s*mk</td><td>the strings 100 and mk optionally separated by any amount of white space (spaces, tabs, newlines)</td></tr>
<tr><td>abc\b</td><td>abc when followed by a word boundary (e.g. in abc! but not in abcd)</td></tr>
<tr><td>perl\B</td><td>perl when not followed by a word boundary (e.g. in perlert but not in perl stuff)</td></tr>
</table>

Figure 8: Regular Expressions

2.7 Data Protection Act 1998

The Data Protection Act 1998 is an act that was introduced to preserve a user's identity and confidentiality. (gov.uk, 2017) The Data Protection Act states that user data is:

- used fairly and lawfully
- used for limited, specifically stated purposes
- used in a way that is adequate, relevant and not excessive
- accurate
- kept for no longer than is absolutely necessary
- handled according to people's data protection rights
- kept safe and secure
- not transferred outside the European Economic Area without adequate protection

2.8 Software Development Approaches

2.8.1 Agile

The Agile methodology is a non-linear software development approach. Instead of going through the stages of the software development lifecycle, and always moving forward through the stages, the Agile methodology introduces the idea of iteration. Earlier stages can be revisited in light of new information, or to fix errors than may have become apparent. The Agile methodology is more fluid than the traditional waterfall model and allows team members from different departments (such as design vs testing) to co-operate and increase the cohesion between members. (Waters, 2007)

3 Project Requirements Analysis

3.1 Introduction

All projects have requirements that must be looked at, and analysed, before the actual implementation of the project begins, and this project is no exception. By analysing the project requirements, hiccups can be avoided at later stages due to unforeseen requirements not being fully explored.

This stage attempts to identify the project's functional and non-functional requirements of the project that can later be used to build a strong foundation, and increase the project's odds of success.

3.1.1 Functional Requirements

The functional requirements are steps that must be implemented for the software to function. These can represented as a checklist of goals that must be fulfilled before the software can be considered complete.

The functional requirements are listed below. The software must be able to:

1. Read and parse a large body of text (the corpus)
2. Split the corpus into sentences
3. Remove unwanted words from the sentences (these are words that have little semantic meaning. For example: as, a, but, between, the, where, was)
4. Calculate the unigram counts of the corpus (how often each word occurs) and store this information in a TreeMap, as well as storing this information as a file.

5. Calculate the semantic bi-gram counts, by storing the occurrences of each permutation of words, and store this information in a TreeMap, as well as storing this information as a file.

6. Use the values of the semantic bi-gram counts to calculate the probability of a sentence the language model has not seen before.

7. Use a list of words to evaluate which word has the greatest score.

3.1.2 Non-Functional Requirements

Non-functional requirements are requirements that must be kept in mind while designing and implementing the solution for this project. These requirements are not required by themselves for the software to function, but they are still important to the software's Quality of Life (QoL). The significant non-functional requirements that must be kept in mind include:

3.1.2.1 Security and Confidentiality

The software is likely to be implemented as a stand-alone application. If this is the case, and the platform for the software is a desktop application. There is little concern for security. The language model will not make use of distributed services or applications, and no personal information is being transmitted.

However, the corpora used by the language model may contain information that may be sensitive. The corpora that will be used for the purpose of this project will be from a reputable source that contains public information from fiction, interviews and transcriptions.

In the future, there may be an option for user's to upload or use their own custom corpus. It is a very arduous task to verify the confidentiality of this content, and the corpus will be stored in the project's directory in plain text.

Upon the introduction of the feature of user corpus use, steps will need to be taken to prevent the corpora being read or stored longer than it needs to be. There should be a feature that allows a user to remove a corpus they uploaded as well as its counts and any information extracted from it.

The Data Protection Act should be considered, and enforced, if corpora is found to contain information which breaks this act. For example, a corpus of usernames, e-mails and passwords.

3.1.2.2 Usability

User interaction is one of the most important features to be considered in the development of the language model. During the implementation of the software, it is acceptable to use techniques such as breakpoints and using the console to query the language model.

However, it can be advantageous to implement an easy-to-use method of querying the language model, so as a Graphical User Interface (GUI).

Towards the end of the project, a GUI will be necessary to provide a non-technical method of interacting with the language model. A user should not have to have programming knowledge to operate a program.

3.1.2.3 Performance

Performance is important because it measures how efficient the program is. The tasks that the program performs should be done as quickly and as high a quality as possible.

Loading times should be carefully monitored, and code should optimised where possible. Metrics are available which can used to evaluate the software's performance. These metrics consist of things such as lines of code, levels of inheritance, coupling and cohesion.

3.2 Software Development Methodology

Methodology Chosen: Agile

The approach that will be taken is the Agile methodology. This means that stages in the software development lifecycle may be iterated over, and refined.

Agile is ideal because it allows for a more fluid path between stages. Rather than the process being linear, Agile allows for earlier stages to be revisited in light of new information, error fixing or to introduce new ideas.

3.3 Programming Environment

The main things that were kept in mind while selecting the programming language(s) is efficiency, familiarity, platform and ability to deal with several data structures.

The intended platform is a desktop application. A mobile application is unsuitable due to the nature of the project – the software will need to have access to a corpus that may take a long time to load, especially if it is on a smartphone or tablet.

A web application was briefly considered, but it was rejected due to the merits of having an online application not being very useful for this particular project. However, it may be a good idea to have the SBigram language model online with corpora preloaded, allowing users to interact with it.

A programming language that deals with data structures effectively is ideal. The program should also have a Graphical User Interface (GUI).

Programming languages that may be considered are Java, C#, C and Python.

3.4 Corpora

The corpus used for evaluating the language model is very important – a poor corpus has a significant impact on the quality of sentence evaluation. The corpus used should contain at least a million words, but the more words, the better.

Corpora can exist in both text and spoken words (audio).

3.4.1 Corpus Requirements

For the goal of being able evaluate the semantic meaning of bi-grams, it is important that the corpus has many common words and phrases, so that the software has a large source of everyday words.

The corpus must also be written text, rather than spoken, as the language model is unable to parse audio.

A technical corpus is unsuitable, because it contains many esoteric words or jargon that is uncommon in day-to-day language. For example, a corpus compiled of chemistry journals is unsuitable, as a lot of the content will comprise of chemical names and equations, which is unhelpful for the goal of this language model.

Due to the nature of the project, the budget is minimal, and therefore corpora that is not openly accessible is unable to be used.

3.5 Removal of Unnecessary or Unwanted Words

When evaluating the perplexity of sentences or word pairings, it may be beneficial to remove words with little semantic meaning. Words such as - the, as, but, where, become, were, over – contribute little to the semantic meaning of a sentence, and therefore there must be an option to filter these words from the corpus (and sentence).

4 System Design

4.1 Programming Environment

Language(s) Chosen: Java
IDE Chosen: Eclipse

The programming language that was selected is Java due its strength in dealing with strings and data structures. It also allows an Object-Oriented approach which can improve the efficiency of the software by using techniques such as polymorphism and inheritance.

Java is also a language that incorporates garbage collection. This simplifies the implementation of the software, as it automates the mundane task of memory allocation. Although garbage collection does have an impact on the code's efficiency, it increases the reliability and prevents the possibility of mistakes being made during memory allocation.

Java was also selected due to my personal familiarity. It meets the requirements of the software, and it made the process of implementing the project much easier.

C# and Python were also considered, due to their strengths with data structures. Python was rejected due to unfamiliarity of the programming language, and the choice between Java and C# was personal preference.

C was briefly considered due to creating the opportunity to learn about what was going on at a memory level, but it was ultimately rejected due the increased complexity it would add to the project.

The IDE used was Eclipse due to its wide popularity, ease of use and familiarity.

4.2 Corpus Selection

The corpus that was selected is a selection of works from the Open American National Corpus (OANC). The OANC was chosen due to its accessibility and no financial cost.

The OANC is comprised of many corpora from multiple categories, such as fiction, technical and journalism. A customised corpus can be created by combining different corpora from the OANC.

4.3 Reading and Parsing the Corpus

The software must be able to read in a corpus from a text file, with the option of selecting from a list of available corpora.

4.4 Removal of Unnecessary or Unwanted Words

Many words have little semantic meaning, and do not contribute much when trying to evaluate a sentence. Words such as “the, “as”, “but”, and “between” should have the option of being ignored when extracting information from the corpus. This will prevent the language model from producing inflated scores, and removal of these words will increase the accuracy of the language model.

4.5 System Flow

The system flow is represented in the form of an activity diagram. This looks at the possible paths that can be undertaken when a user interacts with the language model. It shows the process of the system and the order in which it accomplishes tasks.

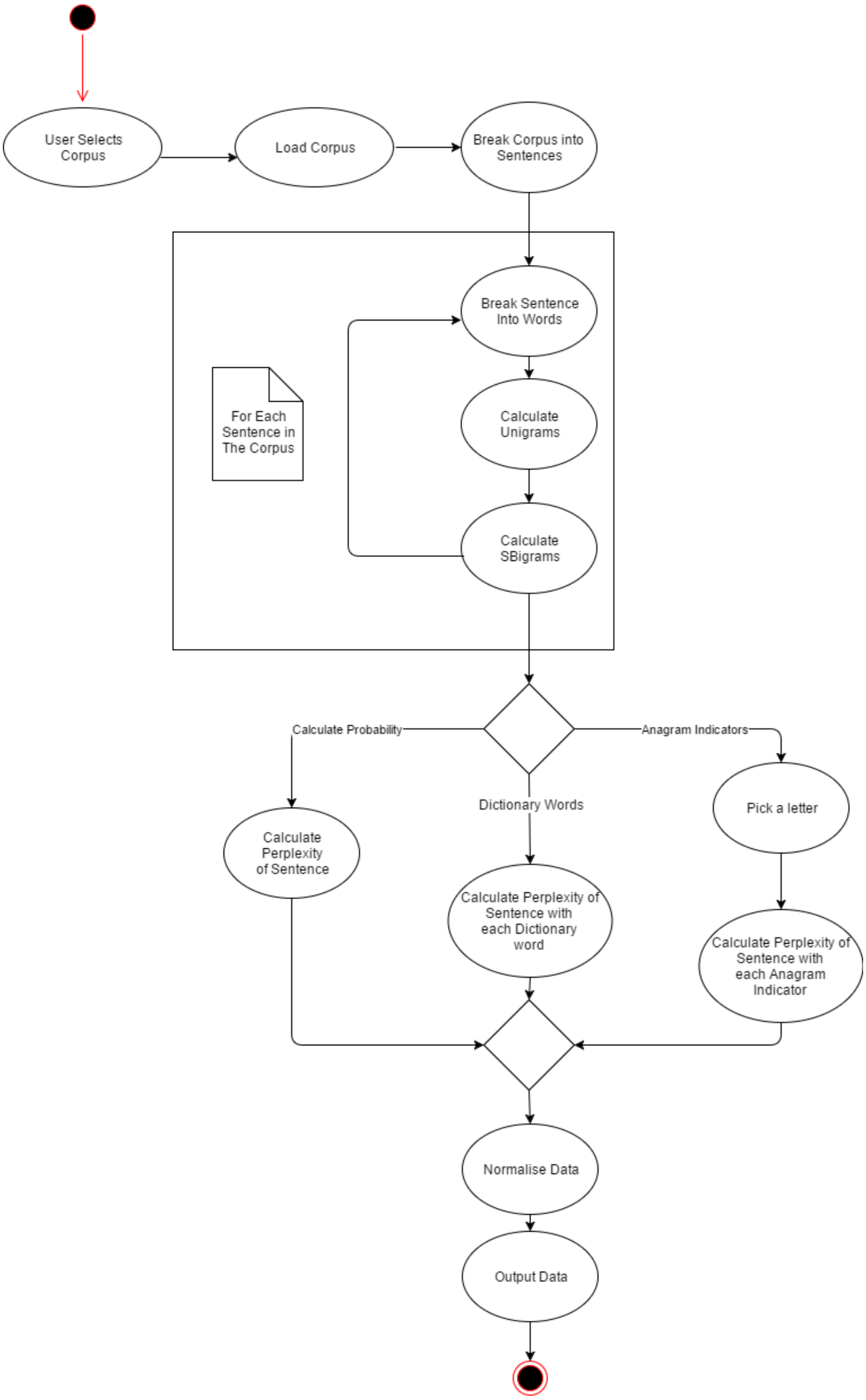


Figure 9: Activity Diagram

4.6 System Structure

The intended structure of the system is shown below. Most methods are contained within the Sngrams class – this is where most operations take place, such as corpus loading, parsing and sentence evaluating.

The AnagramIndicators and Dictionary classes will be used for evaluating the language model upon completion – they will iterate through a large list of words, and append each word to a given sentence and return what the model believes has the strongest semantic connection.

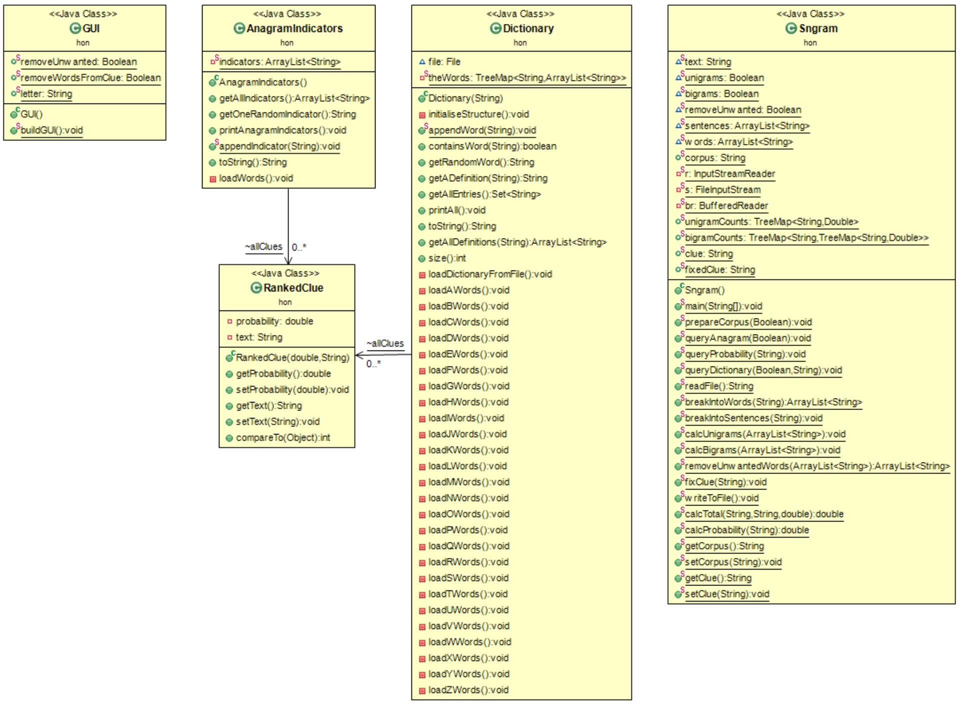


Figure 10: Class Diagram

4.7 Graphical User Interface

A Graphical User Interface (GUI) has a significant impact on the usability of the software. It is unreasonable to expect users to have to interact with the software by changing variable values and using the console as an input/output system.

The GUI allows the user to interact with the software easily, as they can simply check boxes and enter values into text boxes. The GUI should be intuitive and easy to use.

A GUI that does not fulfil these conditions have a significant impact on the user's experience with the software – the user does not want to have to deal with a badly designed or buggy GUI.

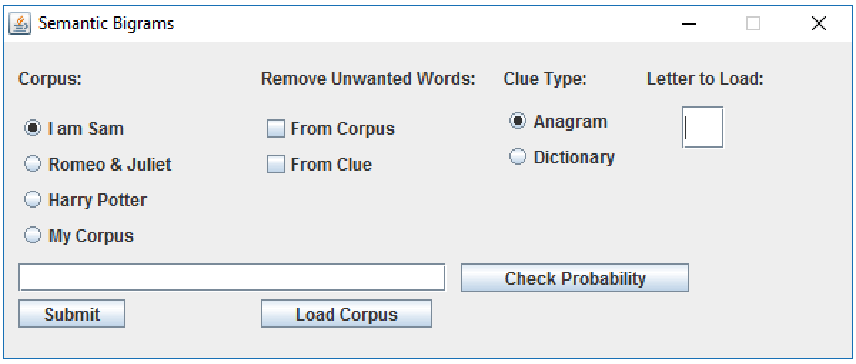


Figure 11: GUI

5 Methodology

5.1 Introduction

5.1.1 Plan

1. Read in the corpus
2. Split the corpus into sentences
3. Remove unwanted words from the sentences (these are words that have little semantic meaning. For example: as, a, but, between, the, where, was)
4. Calculate the unigram counts of the corpus (how often each word occurs) and store this information in a TreeMap, as well as storing this information as a file.
5. Calculate the semantic bi-gram counts, by storing the occurrences of each permutation of words, and store this information in a TreeMap, as well as storing this information as a file.
6. Use the values of the semantic bi-gram counts to calculate the probability of a sentence the language model has not seen before.

5.2 Corpora

5.2.1 The Practice Corpus

The corpus “I am Sam” was selected as a practice corpus, due to its very short nature. During the creation of the language model, this corpus was used to ensure that the language model is working properly – Any exceptions caused by the language are easily spottable and can be noticed easily.

I am Sam.

Sam I am.

I do not like green eggs and ham.

Figure 12: I am Sam corpus

Some errors that occurred during the creation of the language model were spotted and fixed because of this corpus. For example, during development, the language model initially did not properly update values, but instead overwrote them.

The “I am Sam” corpus was used throughout development of the language model, for several areas of testing – ensuring sentences were being broken up correctly, unigrams were counted correctly, and SBigrams were counted correctly.

This corpus was only used for practice, as it is far too short to gain any meaningful advantage from training the language model on.

5.2.2 Corpus Variety

After being satisfied that the language model was working correctly on the “I am Sam” corpus, it was time to move onto larger corpora so that the corpus could be used to evaluate sentences the language model has not seen before.

5.2.2.1 Harry Potter – 78,792 Words

Harry Potter and the Sorcerer’s Stone (US) was selected as the first test corpora. It is the entirety of the first Harry Potter book.

5.2.2.2 Harry Potter in different languages – Approximately 500 words

Excerpts from Harry Potter were also tested in French, German, Polish and Russian. This was used to test the portability of the language model. It was found that the language model could deal with any languages that use the same characters as English (such as French and Spanish). But the language model is unable to deal with languages that use non-English characters, such as Polish or Russian (which uses characters such as Я and Ł.

5.2.2.3 Romeo and Juliet – 26,249 Words

William Shakespeare's Romeo and Juliet was chosen due to the vast difference in language between Shakespeare and modern English. This corpus was chosen to verify that the language model can deal with the entire corpora being very different. It is expected that sentence evaluations will produce much different results, depending on which corpus is used.

5.2.2.4 Bohemian Rhapsody – 370 Words

Queen's Bohemian Rhapsody was chosen to test how well the language model deals with songs. Ultimately it was discovered that it deals with songs very poorly – songs often tend to be very short, and is not split up into many sentences (so the language model reads the entire song as one sentence) unless the song happens to have question marks (?) or exclamation points (!).

5.2.2.5 OANC excerpts – 831,034 Words

The Open American National Corpus (OANC) was the first large corpus used to evaluate the language model. Its large size creates a very large SBigram tree, and ensures that there is a variance in results depending on the sentence – the goal is that the result is based on the sentence being evaluated, not the corpus.

The OANC is a general word corpus, and includes text from mostly conversations, fiction and travel guides. This is deliberate, to avoid esoteric words or jargon that is not used in everyday conversation. For example, a corpus of published chemistry papers is unsuitable, as it is likely to contain technical words such as chemicals and equations.

5.2.3 Final Corpus Used

The OANC was the final corpus used to evaluate the language model, due to its large size and general language. It contains 831,034 words. The more words, the better, as it increases the accuracy and variance of SBigrams.

5.2.4 Preparing the Large Corpus

Unfortunately, making use of a corpus is not as simple as one might think – it isn't a case of simply downloading the corpus and opening one big text file.

The OANC is made up of multiple categories, such as fiction, technical and letters.

Name	Date modified	Type
fiction	06/10/2010 16:12	File folder
journal	04/11/2010 21:12	File folder
letters	06/10/2010 16:13	File folder
non-fiction	15/04/2017 11:37	File folder
technical	15/04/2017 11:41	File folder
travel_guides	15/04/2017 11:42	File folder

Figure 13: OANC Categories

HandRHawaii.anc	04/11/2010 20:38	ANC File	2 KB
HandRHawaii	12/10/2010 15:04	Text Document	16 KB
HandRHawaii-hepple	12/10/2010 14:59	XML Document	1,099 KB
HandRHawaii-logical	12/10/2010 14:50	XML Document	12 KB
HandRHawaii-np	12/10/2010 15:12	XML Document	222 KB
HandRHawaii-s	12/10/2010 14:51	XML Document	38 KB
HandRHawaii-vp	12/10/2010 15:21	XML Document	46 KB
HandRHongKong.anc	04/11/2010 20:38	ANC File	2 KB
HandRHongKong	12/10/2010 15:04	Text Document	2 KB
HandRHongKong-hepple	12/10/2010 14:59	XML Document	63 KB
HandRHongKong-logical	12/10/2010 14:50	XML Document	4 KB
HandRHongKong-np	12/10/2010 15:12	XML Document	12 KB
HandRHongKong-s	12/10/2010 14:51	XML Document	4 KB
HandRHongKong-vp	12/10/2010 15:21	XML Document	7 KB
HandRIbiza.anc	04/11/2010 20:38	ANC File	2 KB

Figure 14: OANC Format

The corpus contains many .XML files that must be parsed before being made readable. As shown below, the text is unreadable due to the XML tags, and it must be removed.

```
<region xml:id="penn-r32" anchors="249 253"/>
<node xml:id="penn-n32">
    <link targets="penn-r32"/>
</node>
<a label="tok" ref="penn-n32" as="xces">
    <fs>
        <f name="base" value="star"/>
        <f name="msd" value="NN"/>
    </fs>
</a>
<region xml:id="penn-r33" anchors="254 260"/>
<node xml:id="penn-n33">
    <link targets="penn-r33"/>
</node>
<a label="tok" ref="penn-n33" as="xces">
    <fs>
        <f name="base" value="rating"/>
        <f name="msd" value="NN"/>
    </fs>
</a>
<region xml:id="penn-r34" anchors="261 263"/>
<node xml:id="penn-n34">
    <link targets="penn-r34"/>
</node>
<a label="tok" ref="penn-n34" as="xces">
    <fs>
        <f name="base" value="in"/>
        <f name="msd" value="IN"/>
    </fs>
</a>
```

Figure 15: OANC .XML File content

This is done using the search and replace function inside Eclipse, and searching for a tag such as: *<fs>* or *<f name="base" value="*

All text in the document that matches that tag can be then be replaced with a blank character.

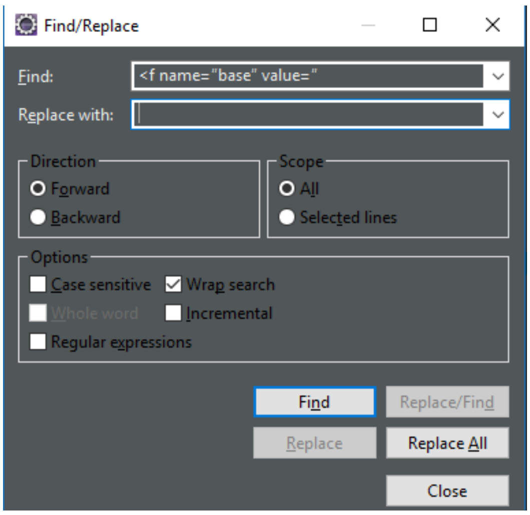


Figure 16: Find/Replace Function

Regular expressions (RegEx) can also be used to parse the corpus. Any lines that have tags with a string can be removed using the regular expressions option, and using a suitable RegEx line.

Fortunately, in the case of the OANC, there were text files that were usable, and a corpus could be built up using these text files. It is a fairly slow process, as there are many files, and many of them are short. The corpus was built up by organising the text files by size, selecting files from different categories and copying the content to the collection of extracts to create one big corpus.

5.3 Storing Corpora

A directory was created inside the program's source folder that contains all of the corpora. Corpora stored here can be removed and added to at will.

The corpora are stored as .TXT files.

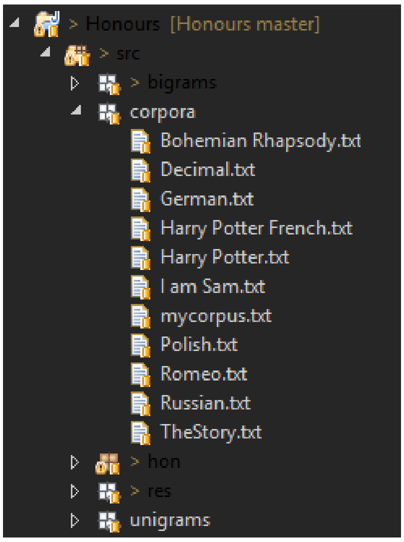


Figure 17: Corpora Storage

5.4 Reading in the Corpus

The corpora were accessed using a *FileInputStream()* and an *InputStreamReader().* These are fairly basic methods that can be used to read in a text file, and store the content as a string.

The software utilises these functions to read the file one character at a time, and adds the character to the string. This process repeats until the end of the file has been reached.

```java
// Read in the text file of the corpus
public static String readFile() throws IOException{


    // getCorpus() finds the desired corpus, selected by the user
    s = new FileInputStream("src\\corpora\\" + getCorpus() + ".txt");
    r = new InputStreamReader(s);

    int data = r.read();

    // Iterate through the file until the entire thing is read
    while(data != -1){
        char c = (char) data;
        text += c;
        data = r.read();
    }

    // Return the corpus content
    return text;
}
```

Figure 18: Reading the corpus - code extract

It should be noted that the corpus in its entirety is stored in a single string. A string has a storage capacity of 10 bytes (Microsoft, n.d.), which means that it can store approximately two billion characters. This causes problems if the corpus is especially large and exceeds two billion characters.

This is an issue that should be resolved for future use, in the case of larger corpora being used. A possible solution may be to read in the corpus one paragraph at a time.

5.5 Splitting the Corpus into Sentences

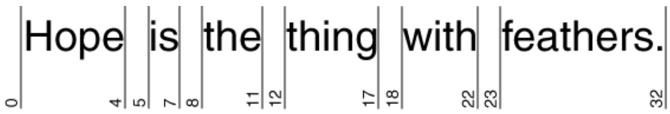
Breaking a body of text into sentences is a deceivingly difficult task. One might think that it's simply a case of splitting the text every time it encounters a full stop (.). However, a full stop is also present in text that does not mean the end of a sentence. For example:

Mr.   Mrs.   2.3   etc.   …   i.e.   myfile.jpg

Figure 19: Example of full stop not ending sentences

Sentences can also end in non-full stop characters, such as question marks (?) and exclamation points (!).

Sentence splitting is, in reality, a mammoth task. Fortunately, libraries exist which tackles this problem. The BreakIterator class was added, which has the capability to identify the instance of a sentence according to the UK English language (many languages have a different sentence structure to British English).



(Oracle, 2015)

Figure 20: BreakIterator

5.5.1 BreakIterator

The BreakIterator class makes use of a *getSentenceInstance()* method. The getSentenceInstance() method makes use of the markBoundaries function to identify the ending of sentences. It then prints a caret (^) under each boundary. It identifies sentences using patterns that occur in the UK English language.

```
She stopped.  She said, "Hello there," and then went on.
^             ^                                         ^

He's vanished!  What will we do?  It's up to us.
^               ^                 ^             ^

Please add 1.5 liters to the tank.
^                                 ^
```

(Oracle, 2015)

Figure 21: markBoundaries functionality

The software makes use the BreakIterator to identity a sentence instance, extract the sentence, converts the sentence to words, calculate the desired counts and then moves onto the next sentence.

For each sentence the BreakIterator identifies, the software:

1. Replaces all newlines with a space
2. Removes all non-alphanumeric characters, such as dashes and commas
3. Replaces all multiple spaces with a single space

The model accomplishes these tasks using regular expressions (RegEx) to identify the required text pattern in the sentence.

The necessary operations can then be applied to the identified sentence (such as calculate counts) and then the function moves onto the next sentence. It no longer stores the sentence, as it is unnecessary and takes up memory.

Initially, every time the software identified a sentence, it would store the sentence in an ArrayList – thereby having an ArrayList of all of the sentences in the corpus. After collecting all of the sentences, the software would then apply the operations. This was changed due to its inefficiency, and it was decided that it was better to discard sentences after the information has been extracted from them.

Another small issue is that the BreakIterator recognises titles (such as Mr. and Dr.) as the end of a sentence. However, this is very insignificant due to the overwhelming size of the corpus. But it should be noted that the error does exist.

```java
    // Break a text into sentences
    public static void breakIntoSentences(String text) throws IOException{

        String theSentence = "";
        ArrayList<String> words = new ArrayList<String>();

        BreakIterator iterator = BreakIterator.getSentenceInstance(Locale.UK);

        iterator.setText(text);
        int start = iterator.first();

        // Split the corpus into sentences
        // Extract the information from each sentence, add it to the tree
        // Then move onto the next sentence
        // Sentences are not held after they are used to save memory
        for(int end = iterator.next(); end != BreakIterator.DONE; start = end, end = iterator.next()){
            theSentence = text.substring(start, end);
            theSentence = theSentence.replaceAll("\n", " ");
            theSentence = theSentence.replaceAll("[^A-Za-z0-9 ]", "");
            theSentence = theSentence.replaceAll("\\s+", " ");

            // Break the sentence into words
            words = breakIntoWords(theSentence);  //separate into words

            // if removeUnwanted is true, remove unwanted words
            if(removeUnwanted){
                words = removeUnwantedWords(words);
            }

            // if unigrams is true, calculate the unigrams
            if(unigrams){
                calcUnigrams(words);
            }

            // if bigrams is true, calculate the SBigrams
            if(bigrams){
                calcBigrams(words);
            }
        }

        // Write the results to a file
        writeToFile();
    }
```

Figure 22: Splitting the corpus into sentences - code extract

5.5.2 Splitting a Sentence into Words

The process of breaking a sentence into words is actually fairly simple. The *breakIntoSentences()* function will send a string (the sentence) to the *breakIntoWords()* function.

The function will then identify the words by looking for spaces, and then split them and add the words to an ArrayList.

The function will then iterate through each word in the sentence, and remove any non-alphanumeric characters and convert them to lowercase (for consistency).

After each word has been stripped of non-alphanumeric characters, it is added to the ArrayList. Once all words have been added, the function will then return the ArrayList of words.

It should be noticed that removing punctuation affects some words, such as “didn’t” and “never-the-less”. These words simply become “didnt” and “nevertheless”.

```java
    // Break a sentence into words
    public static ArrayList<String> breakIntoWords(String theSentence){

        String [] words;
        ArrayList <String> theWords;
        ArrayList <String> theWordsTidiedUp;

        // Separate into words
        words = theSentence.split(" ");
        theWords = new ArrayList <String>(Arrays.asList(words));
        theWordsTidiedUp = new ArrayList<String> ();

        // Remove all punctuation and multiple spaces
        for(String word : theWords) {
                    word = word.replaceAll("[^A-Za-z0-9]", "").toLowerCase();
                    theWordsTidiedUp.add(word);
        }
        return theWordsTidiedUp;
    }
```

Figure 23: Splitting a sentence into words - code extract

5.6 Removing Unwanted Words from the Corpus

Many words have little semantic meaning, and therefore it is advantageous to have the option of removing certain words from the corpus when calculating SBigrams. The full list of unwanted words is shown below.

```
a
about
after
although
am
an
and
are
as
at
be
because
before
been
between
but
even
for
had
if
in
into
is
it
nor
of
on
once
or
since
so
that
the
though
till
to
too
unless
until
was
what
when
whenever
wherever
whether
which
while
yet
you
```

Figure 24: Unwanted Words list

This list is by no means exhaustive, and can (and should) be expanded on in the future. This is simply a matter of adding a new line to the “Unwanted Words.txt” file with the desired word.

Note that the words are in alphabetical order – this isn’t strictly necessary, but it makes it more readable when viewing the file.

```java
// Remove unwanted words using Unwanted words.txt
public static ArrayList<String> removeUnwantedWords(ArrayList<String> words) throws IOException{

    // Load the list of unwanted
    br = new BufferedReader(new FileReader("src\\res\\Unwanted words.txt"));

    String line;

    // For each word in the list
    while ((line = br.readLine()) != null) {

        // For each word in the sentence
        for(int i = 0; i < words.size(); i++){

            // If the words match, remove the word from the sentence
            if(words.get(i).equals(line)){
                words.remove(i);
            }
        }
    }

    // return the sentence with words removed
    return words;
}
```

Figure 25: Removing unwanted words - code extract

For each sentence in the corpus, the software passes an ArrayList of strings, representing the words in that sentence, into the function. Then, for each word, it checks to see if it matches a word on the unwanted words list.

The software then removes the unwanted words, and returns the ArrayList with the unwanted words removed.

For example, “The cat sat on the mat” would be sent as “[The, cat, sat, on, the, mat]”. The software would then check each word in the sentence against the list of unwanted words, to check if it must be removed.

In this case of this sentence, the returned value would be “[cat, sat, mat]”.

5.7 Calculating the Unigram Counts

The unigram counts represent how often each unique word appears in the corpus The *calcUnigrams()* function receives a sentence from the *breakIntoSentences()*

function (See 5.5.2 Splitting a Sentence into Words), extracts the information and updates a TreeMap containing all unique words, and their corresponding values.

```
wishing 4.0
wit 1.0
witch   12.0
witchcraft  5.0
witches 8.0
with    416.0
within  3.0
without 41.0
wizard  41.0
wizardin    1.0
wizarding   6.0
wizardry    6.0
wizards 27.0
wizened 1.0
wohsi   1.0
woke    10.0
woken   4.0
wolfing 1.0
wolfsbane   2.0
woman   18.0
women   3.0
```

Figure 26: Excerpt of Unigram counts in Harry Potter and the Sorcerer's Stone

For each sentence identified by the *breakIntoSentences()* function, it will pass an ArrayList of strings representing each word in the sentence.

The calcUnigrams() function will then iterate through each word in the ArrayList, and will update the TreeMap of unigram counts, with the current value + 1. If the word does not currently exist in the unigram counts TreeMap, then a new entry is created for that word with a value of 1 (since it is the first occurrence).

```
    // Calculate the unigrams (Occurrences of each word)
    public static void calcUnigrams(ArrayList<String> words){

        for (String s : words) {
            // Unigram exists - update count
            if (unigramCounts.containsKey(s)) {
                unigramCounts.put(s, unigramCounts.get(s) + 1);
            } else {
                // Unigram doesn't exist - create new branch with value of 1
                unigramCounts.put(s, 1.0);
            }
        }
    }
```

Figure 25: Calculating Unigram counts - code extract

All of the unigram counts are stored in a TreeMap that is created as an instance variable, this prevents entries from being overwritten or duplicated. The TreeMap consists of a string (the word) and a double (the number of occurrences).

```
    public static TreeMap <String, Double> unigramCounts = new TreeMap <String, Double> ();
```

Figure 27: unigramCounts Instance Variable

5.8 Calculating the Semantic Bigram Counts

The SBigram counts are quite similar to the unigram counts – each unique word paring is stored in a TreeMap within a TreeMap.

The TreeMaps are nested so that the three linked values can be stored – the first (outer) word, the second (inner) word and the number of occurrences.

TreeMap(String, TreeMap(String, Double))

(The,(Man 5.0))

Figure 28: bigramCounts Representation

The SBigram counts are then written to a file, which contains how often a pair of words has appeared together in the same sentence. The number in parenthesis is the total number of occurrences between the pair of words, while the number outside of the parenthesis is how many occurrences of that order of words.

For example, in the excerpt below, “escape” + “the” appears three times, while the sum of “escape the” and “the escape” is equal to five. By looking elsewhere in the file, one can expect “the escape” to equal two.

```
the escape 2.0 (5.0)
```

Figure 29: "the escape" bigram

```
escape of 1.0 (1.0)
escape punishment 1.0 (1.0)
escape straps 1.0 (1.0)
escape the 3.0 (5.0)
escape them 1.0 (1.0)

escaping day 1.0 (1.0)
escaping dudleys 1.0 (1.0)
escaping every 1.0 (1.0)
escaping gang 1.0 (1.0)
escaping helicopters 1.0 (1.0)
escaping house 1.0 (2.0)
escaping in 1.0 (1.0)
escaping muggles 1.0 (1.0)
escaping single 1.0 (1.0)
escaping the 1.0 (2.0)
escaping visited 1.0 (1.0)
escaping who 1.0 (1.0)

especially and 1.0 (1.0)
especially be 1.0 (1.0)
especially classroom 1.0 (1.0)
especially dangerous 1.0 (1.0)
especially days 1.0 (1.0)
```

Figure 30: Excerpt of SBigram counts in Harry Potter and the Sorcerer's Stone

For each sentence identified by the *breakIntoSentences()* function, it will pass an ArrayList of strings representing each word in the sentence to the *calcBigrams()* function.

For each word in the sentence, the function will compare it with each following word in the sentence. This means that every possible permutation is compared exactly once – each unique word pair is evaluated once.

For each word pair, the function will look at the bigramCounts TreeMap using the outer word, and depending on what it finds, it will complete different tasks. There are three possibilities:

1. Outer word does not exist

A new entry is made to the bigramCounts TreeMap. With the outer word, inner word and the number of occurrences is set to one (as it is the first instance of the pair being seen).

2. Outer word exists, inner word does not exist

A new branch is added to the TreeMap where the key equals the outer word. The inner word is set to whatever the second word in the word pair is. The value is set to one, as it is the first instance of that word being seen.

3. Both words exist already

In the case that the bigram already exists in the TreeMap, the value is incremented by one, to signify that another instance has been seen.

```
    // Calculate bigrams
    public static void calcBigrams(ArrayList<String> words){

        String word1;
        String otherWord;
        TreeMap <String, Double> newEntry;
        TreeMap <String, Double> oldEntry;

        for(int i = 0; i < words.size(); i++) {
            word1 = words.get(i);
            for(int j = i + 1; j < words.size(); j++) {
                otherWord = words.get(j);
                //System.out.println(word1 + " " +otherWord);
                if(bigramCounts.containsKey(word1)) {
                    // Key found
                    // Get inner map
                    oldEntry = bigramCounts.get(word1);
                    if(oldEntry.containsKey(otherWord)) {
                        // Bigram already exists
                        // Update count
                        oldEntry.put(otherWord, oldEntry.get(otherWord) + 1);
                        //System.out.println("Bigram exists " +word1 + " " +otherWord +" updating c
                    } else {
                        // Bigram doesn't exist
                        oldEntry.put(otherWord, 1.0);
                        //System.out.println("Key exists - new branch " +word1 + " " +otherWord);
                    }

                } else {
                    // Key not found - make new entry
                    newEntry = new TreeMap <String, Double> ();
                    newEntry.put(otherWord, 1.0);
                    bigramCounts.put(word1, newEntry);
                    //System.out.println("New key - adding " +word1 + " " +otherWord);

                }
            }
        }
    }
```

Figure 31: Calculating Bigrams - code extract

All of the bigram counts are stored in a TreeMap that is created as an instance variable, this prevents entries from being overwritten or duplicated. The TreeMap consists of a string (the word) and a TreeMap (containing a string (second word) and a double (the number of occurrences).

```
    public static TreeMap <String, TreeMap <String, Double>> bigramCounts = new TreeMap <String, TreeMap <String, Double>> ();
```

Figure 32: bigramCounts Instance Variable

5.9 Evaluating the Perplexity of a Sentence

One of the main goals of this project is to be able to evaluate a sentence that the language model has never seen before. *The calcProbability()* function achieves this goal by taking a sentence and then extracting information from it.

The function receives a string representing the sentence. This sentence is then broken up into words (see 5.5.2 Splitting a sentence into words). The function now has an ArrayList of the words in the sentence.

The function then starts at the first word in the ArrayList, then compares its score with each following word in the list. The function then iterates through this process until all possible word pairings are calculated.

The orange:	4.0
The cat:	6.0
The jumped:	2.0
orange cat	1.0
orange jumped:	0.0
cat jumped:	2.0
Total:	15.0

Figure 33: Perplexity Calculation Visual Representation

Each word pair needs to be evaluated, so for each pair of words that the calcProbabilities() function finds, it will pass the two words and a double representing the occurrences into the function. It will also inverse the words and send them again, this time the occurrences will usually be greater than zero.

The words have to be inverted because it needs the words in both orders. For example, if the word pair was “orange cat” it would also require “cat orange” due to how the values are stored in the TreeMap.

```java
// Evaluate the probability of the sentence and assign a value
public static double calcProbability(String theSentence){

    ArrayList<String> words = new ArrayList<String>();

    // Break sentence into words
    words = breakIntoWords(theSentence);

    double occ = 0;

    for(int i = 0; i < words.size(); i++){

        for(int j = i+1; j < words.size(); j++){
            occ += (calcTotal(words.get(i),words.get(j),0) + (calcTotal(words.get(j),words.get(i),0)));

            //System.out.println(words.get(i) + " " + words.get(j) + " " + occ);
        }
    }

    occ = occ / (words.size()*(words.size()-1)/2);

    return occ;
}
```

Figure 34: calcProbabilities function - code extract

5.9.1 Calculating the Word Totals

For each word pairing identified in the *calcProbability()* function, the number of occurrences needs to be looked up in the bigramCounts TreeMap.

The calcTotal() function accomplishes this by receiving three values – a string (the outer word), another string (the inner word) and a double (the number of occurrences).

orange cat

Outer word Inner Word

Figure 35: Inner and Outer Words

The function will then use the outer word to identify the key in the bigramCounts outer list. Once found, the inner word will be used to identify the key in the inner TreeMap. Once that is found, the the number of occurrences is looked at and the function returns the occurrences.

In the event that either the outer or inner words are not found in the bigramCounts TreeMap, then the function will simply return zero, as this means there are no existing counts.

```java
    // Add the inverse of a bigram to get an accurate number ("the man" is added to "man the")
    public static double calcTotal(String inWord, String outWord, double occurrences){

        if(!inWord.equals(outWord)){
            for(Entry<String, TreeMap<String, Double>> outer : bigramCounts.entrySet()){

                if(outer.getKey().equals(outWord)){
                    for(Entry<String,Double> inner : outer.getValue().entrySet()){

                        if(inner.getKey().equals(inWord)){
                            occurrences += inner.getValue();
                        }
                    }
                }
            }
        }

        return occurrences;
    }
```

Figure 36: calcTotal function – code extract

5.9.2 Normalising the Data

The data is normalised before outputting the sentence's score. The reason for this is that it prevents a bias towards longer sentences. The data is normalised by taking the sum of the occurrences for each word pair, then divide it by the number of permutations of words.

```java
        occ = occ /  (words.size()*(words.size()-1)/2);
```

Figure 37: Data Normalisation

For example, if a sentence had a total score of 150, and had 4 words. The equation would be *150 / ((4 * (4-1)) / 2)*. Once solved, this sentence now has a score of 25.

This equation is applied to each sentence that is being evaluated, and provides a fair score for each one.

5.10 Appending Word to a Sentence

Now that the language model is able to evaluate a sentence it has not seen before. One possible use is its ability to have a word with a blank space (e.g. The ____ cat jumped) and identify which word out of a list is best suited.

It should be noted that, unlike the n-gram model, the SBigram model does not rely on word sequence at all, so a word can simply be appended to the end of the sentence. If the case was that the word had to be in a certain spot, then this could be done so using a tag that the word could be inserted into.

Cryptic crossword clues were selected as the format for evaluating the SBigram model. Crossword clues come in multiple formats, but this project will only be looking at two – anagrams and dictionary words.

5.10.1 Anagram Indicators

Clues that make use of anagrams have three components in the clue – a word hint, a scrambled word, and a word that indicates motion or change (this is the anagram indicator, and gives a hint that there is an anagram directly after, or before, the indicator).

Example clue: Health resorts to quickly pass.

The hint is “health resorts”, the anagram is “pass” and the anagram indicator is “quickly”.

Therefore, to solve the clue, one would look at the word “pass” and identify an anagram of that word that means health resorts. The answer is “spas”.

However, one could see that the word “quickly” can be replaced with any word that indicates motion or change. It is ideal for this word to make sense in the context of the clue, and the language model is a viable way of evaluating which word is most suitable.

```
indicators.add("fabricated");
indicators.add("failing");
indicators.add("false");
indicators.add("fanciful");
indicators.add("fancy");
indicators.add("fantastic");
indicators.add("fashion");
indicators.add("fashioned");
indicators.add("fashoning");
indicators.add("faulty");
indicators.add("fermented");
indicators.add("feverishly");
indicators.add("fiddle");
indicators.add("fiddled");
indicators.add("figthing");
indicators.add("find");
indicators.add("fixed");
indicators.add("flapping");
```

Figure 38: Anagram Indicators excerpt

5.10.2 Dictionary Words

Clues that make use of dictionary words have three components in the clue – two words hint, a word the gives the first letter of the answer, and a word that implies being the first (first, leader, head, primary etc.).

Example clue: Start chopping wood for money.

The hints are “money” and “wood”, the word that gives the first letter is “chopping” and the word that implies first is “start”

Therefore, to solve the clue, one must look at the word following “start”, which is “chopping”. This means the answer begins with a C. Then one must consider another word for wood or money that can added to the C to mean the other. The answer is “Cash”. (C + ash).

However, one can see that rather than the word “chopping”, any word that begins with the letter C can be used. It is ideal for this word to make sense in the context of the clue, and the SBigram language model can be used to identify which word is most suitable from a list of words beginning with that letter.

The Dictionary class contains 26 methods for loading words – one for each letter of the alphabet. Upon being called, the function will append each word in the desired letter to the sentence to compare their scores.

It should be noted that the definitions in the Dictionary class are ignored, as only the word itself is required.

```
            key = "ta";
            meanings = new ArrayList <String>();
            meanings.add("brief thanks");
            meanings.add("thanks");
            meanings.add("thank you");
            this.theWords.put(key, meanings);

            key = "table";
            meanings = new ArrayList <String>();
            meanings.add("present formally");
            this.theWords.put(key, meanings);

            key = "tables";
            meanings = new ArrayList <String>();
            meanings.add("presents formally");
            this.theWords.put(key, meanings);

            key = "tablet";
            meanings = new ArrayList <String>();
            meanings.add("small computer");
            meanings.add("pill");
            this.theWords.put(key, meanings);
```

Figure 39: Dictionary Words excerpt

5.10.3 Appending Words

Depending on which clue type is selected (anagram indicators or dictionary words) the language model will create an ArrayList of strings that will store each word. If dictionary words are selected, the software will only load the words for which letter is selected. (I.e. it will only load T-words if the users selects T).

```
    public static void appendIndicator(String clue){

        String clueWithIndicator = "";
        double prob;
        for(String ti : indicators){

            clueWithIndicator = clue + " " + ti;
            prob = Sngram.calcProbability(clueWithIndicator);
            RankedClue newClue = new RankedClue(prob,clueWithIndicator);

            allClues.add(newClue);
        }

        for(RankedClue rc : allClues){
            System.out.println(rc.getProbability() + "\t" + rc.getText());
        }

        allClues.clear();

        DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
        Date date = new Date();
        System.out.println("Finished at " + dateFormat.format(date));
```

Figure 40: Appending Words

The function will iterate through each word in the ArrayList, and append it to the sentence that is being evaluated. Each time a sentence is evaluated, the full sentence and its score is stored as a RankedClue object.

A RankedClue object has two attributes – the sentence, and its score. The RankedClues are then added to an ArraySet, which allows them to be sorted by their score. This allows for the clues to be displayed to the user, who can then see which words have the greatest scores.

```
public class RankedClue implements Comparable {

    private double probability;
    private String text;

    public RankedClue(double probability, String text) {
        setProbability(probability);
        setText(text);
    }

    // Rank clues by score
    public int compareTo(Object o) {
        RankedClue theClue = (RankedClue) o;


        if(this.probability < theClue.getProbability()) {
            return 1;
        }
        if(this.probability == theClue.getProbability()) {
            return -1;
        }
        return -1;
    }

```

Figure 41: RankedClue class

5.11 Implementing Graphical User Interface

A Graphical User Interface (GUI) was implemented to make interaction with the language model easier. Up until this point, the console, and text files, were used as an input and output system. The software still uses text files to output the information (any displayed output via GUI or console is too large, and is not permanently stored).

Settings were also manually set before running the program, such as corpus selection and clue type. The GUI makes this process much easier and intuitive.

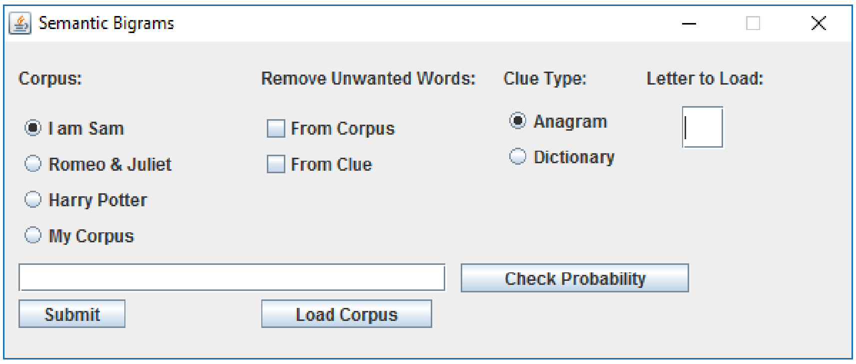


Figure 42: GUI

The user can select a corpus from a pre-set list, and then load the corpus. Once the corpus has been loaded, the user is free to query any sentence they want.

The user has the option of calculating the perplexity of the sentence, which will output the sentence plus its score to the console.

The user can also query clue types by selecting the appropriate radio button (and picking a letter if the clue type is a dictionary word). This will output the sentence, which each appended word. This list of sentences are sorted by score, and it will show the highest ranking clues first.

The GUI was implemented manually using Java.swing. All window components such as buttons and checkboxes were manually added, and properties set.

Figure 43: GUI Button Functionality

6 Testing

6.1 Corpora Variety

During the corpus implementation stage, many different varieties of corpora were tested. For example, novels, plays, songs and excerpts in non-English languages. The language was suitable for some of the mediums that were tested.

It was found that the language model worked well with novels and plays. The corpus could also be a non-English language, as long as it used English characters (i.e. French worked, but Russian did not). The language model was unable to deal with non-English characters, such as Cyrillic text or Korean characters.

The language model was also unable to deal with songs and poems, due to these forms of media using little punctuation – the language model would usually read the entire text as one large sentence.

6.2 Cryptic Crossword Clues

Upon completion of the language model, a feature was added that allowed a word to be appended to the end of a sentence before it is evaluated. This means that using a large list of words, the language model would be able to identify which of the words has the highest score, and therefore the greatest semantic connection.

To test how well the language model evaluates sentences, several sentences were used to identify which words the language model returned as having the strongest semantic connection. A total of 24 Dictionary Word clues and 26 Anagram Indicator clues were used to test and evaluate the program.

6.3 Benchmarking

Informal benchmarking was applied to the software, to identify how well the software works, and to evaluate how efficient it is. However, the benchmarking was not exhaustive or tested thoroughly.

The software runtimes were found to be exponential – it would take approximately 5 minutes to load a corpus containing 78,792 words, but it would take over 12 hours to load a corpus of 831,034 words.

Unfortunately, the benchmarking was not formally tested, and the results are purely from observing the language model creating the SBigram counts. In the future, the language model should be tested more formally, and test logs should be written.

7 Results and Evaluation

7.1 Introduction

The language model was tested with several cryptic crossword clues – 24 Dictionary words and 26 Anagram Indicators. The clues contained both challenging and simple clues in order to fully test the software. The full results are listed under Appendix D and Appendix E.

7.2 Dictionary Words

See Appendix D: Dictionary Words Evaluation for full results.

In general, the language model dealt with dictionary words well. However, some of the clues proved to be very challenging as they intentionally didn't have much for the language model to work with. Nevertheless, the language model still returned strong results.

However, a large issue with what results the model returned was that it often granted uncommon or obscure words such as "askew" low scores, and would award common words such as "good" or "great" high scores.

I feel that, in general, the language model almost always returned a suitable word within the top 10 results, but it was often unable to pick out a single best word. I feel that human supervision is required if the language model was to be used in its current state.

7.3 Anagram Indicator Words

See Appendix E: Anagram Indicator Words Evaluation for full results.

When dealing with anagram indicator words, the model did have a tendency to return the same words. For example, almost all clues had “new”, “some” and “about” ranked very highly. Although these words are often good, they shouldn’t appear in every list of words.

However, looking beyond the words that appeared in most clues. The language model still returned strong results that could be usable in a real cryptic crossword clue.

7.4 Overall Evaluation

Overall, the language model can consistently return a strong list of possible clues, depending on which dictionary word or anagram indicator scored the highest.

However, the language model often failed to select a single best word, and instead, the best suitable word was often lurking in the top eight or so results.

I feel that in its current state. The language model provides good quality results, but it requires human supervision to pick the best suitable match out of the words that the language model selects.

7.5 Google N-Grams

Google has its own N-Gram Viewer (Google, n.d.). It uses extremely large corpora, and can produce graphs based on how often n-grams appear in literature, as well as throughout time.

The results of the SBigram model were often compared to Google’s n-gram results, and it was found that the matches were often similar.

8 Self-Evaluation

8.1 Introduction

The goal of this section is to identify what I, as the student, could have done differently during the course of this project. These potential goals were discovered during the project, and were learned in light of new information, or due to restrictions later on in the project.

8.2 GitHub Repository

A private GitHub repository was used during the project, for the purpose of version and control and having a reliable backup of all work. However, the repository was not shared with the supervisor, who could have been invited as a collaborator to view the work, or possibly comment or suggest changes.

8.3 Over-Preparation

I feel that I spent too much time preparing at the beginning of the project. This time was mainly spent familiarising myself with regular expressions (RegEx), manipulating strings and inputting and outputting to a file.

Although this preparation proved helpful during the creation of the software, the time could have been better spent optimising some areas near the end of the project.

8.4 Testing

I feel that by allowing more time for the end of the project, I would have the opportunity to test the language model more by using multiple corpora – perhaps using content such as political interviews or technical published articles or journals. It would have been interesting to observe the differences between corpora, and it would further prove the strength of the language model.

8.5 Run-Time Issue

Towards the end of the project, I discovered that the language model takes a significantly long time to build the bigram counts (over 12 hours). This run time is unusually long, and I was unable to find the cause.

I feel that I may have been able to look into this issue, and possibly resolve it. However, I feel that this task may be rather time-intensive as it is likely that it would require an overhaul of the entire system.

8.6 Independence

I feel that I could have been more independent throughout the project. I feel that the path of which tasks were to be completed were laid out by the supervisor, and I simply followed it.

However, I did accomplish the tasks on my own, such as calculating the counts, building a corpus, removing unwanted words and evaluating the software. I feel that I could have taken more of a lead on where the project was going during its development.

9 Conclusion

Overall, I feel that the project was a success, over the course of the project, I managed to meet the objectives which I set, and I have created a working language model that can extract information from a large corpus, and use that information to evaluate a sentence that it has not seen before.

The semantic bigram model proved to be more complex than it appears, and it took some time for me to fully understand it. However, I now feel that I fully understand the model, and I would look forward to continue working on the language model in the future.

Unfortunately, there was a large issue that I faced late in the project's development, in which the language model reads large corpora extremely slowly. I was unable to find the cause of this, and I would have liked to work out the reason why it takes so long, and I would like to see the model working without this issue.

I feel that there are a lot of areas that can be improved upon in the future. These changes may have been tasks which were out of the scope of the project, or may be issues that I faced during the development of the language model. The potential for future work will be outlined in the next section, which shows where the language model can go in the future.

I feel that, in its current state, the language model works correctly, and it consistently picks strong words as the best suitable match for a sentence. However, it still requires human supervision, as it is unable to pick the single best match on its own.

I feel that, upon introducing larger corpora, the language model is only going to get more refined and more accurate.

10 Future Work

10.1 Introduction

Although the project has been completed, there are several areas that can be improved upon in the future, either by myself or a future developer. These changes may simply be Quality of Life (QoL) which makes the user's experience with the software more pleasant, or the changes can also be to functionally improve the software.

10.2 Graphical User Interface

The Graphical User Interface (GUI) has potential to be more intricate and allow more user customisation. The final GUI was fairly restrictive in regards to corpus selection, and did not allow multiple letters to be selected.

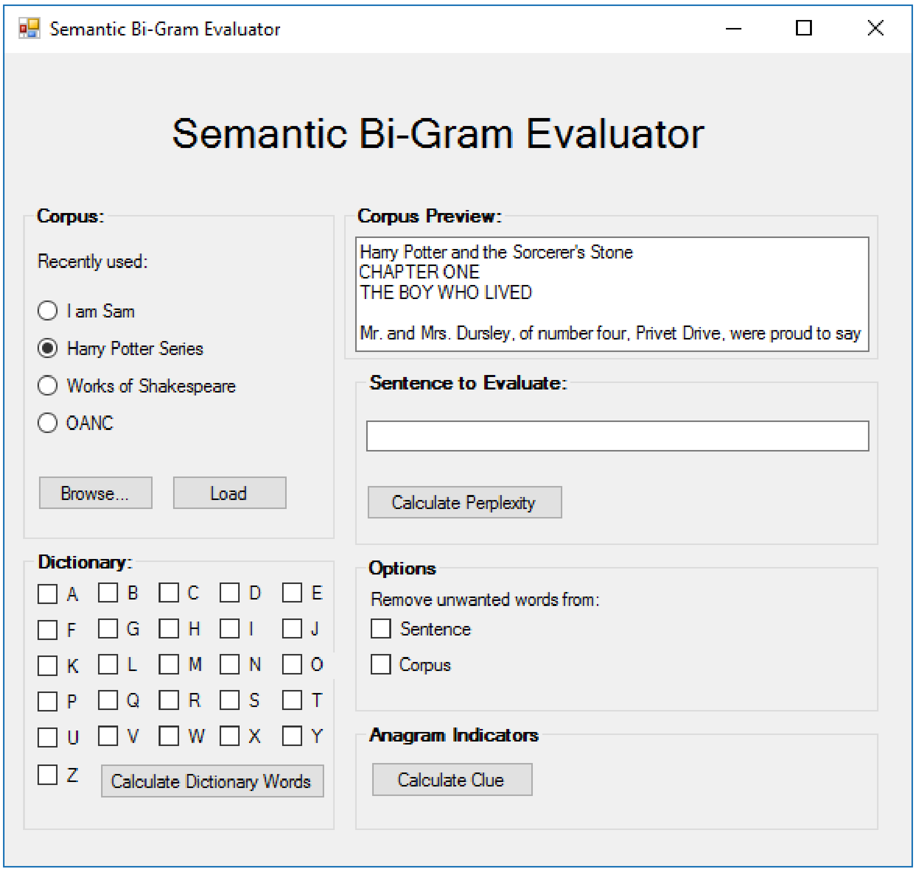


Figure 44: Ideal GUI

Shown above is a prototype that could potentially be implemented. It introduces more features and user customisability. It also solves issues such as the limited corpus selection in the final GUI. It also has a corpus preview that can bolster the software QoL (Quality of Life) and make the program more intuitive to use.

Note that the prototype is created as a C# Windows Form. This does not necessarily mean that the GUI would be made in C#, but a similar GUI could be created in Java.

10.3 Run-Time Optimisation

The process of loading a corpus, and calculating all of the unigrams and SBigrams was extremely time consuming. As shown below, this process can take over 12 hours for an 830,000 word corpus. This process should not take longer than a few minutes.

```
Started at 13:50:53
mycorpus Loaded at 02:12:05

25.5    leader getting cheeky again and
16.6    leader getting cheeky again a
8.4     leader getting cheeky again all
6.1     leader getting cheeky again as
4.6     leader getting cheeky again at
3.8     leader getting cheeky again are
```

Figure 45: Time taken to load the corpus

This run-time is very concerning, as it means that if larger corpora were used in the future, it would take an even longer time to load the corpus.

The time it takes to load a corpus appears to be exponential, but this is unconfirmed. A corpus of around 80,000 words takes approximately five minutes, while a corpus 830,000 words takes over 12 hours.

The time discrepancy is something that would need to be looked at before attempting to optimise the time taken to load a corpus.

10.4 Increase to Corpus Size

The final corpus used contained 831,034 words. However, the language model will produce more accurate results the more words there are in the corpus (provided the corpus is of a reasonable quality). Therefore it is advantageous to use the SBigram language model with larger corpora.

Corpora that contains millions of words are likely to produce different results from what the current corpus does, and it would be interesting to see what the results are.

10.5 Expand on Unwanted Words List

The language model has the option of removing unwanted words from the corpus or submitted sentence. These words have little semantic meaning, and do not provide a strong context for the words that are being evaluated.

Shown below is the list of words that were used in project. These words could be eliminated from the corpus or sentence if the user chose to.

```
a
about
after
although
am
an
and
are
as
at
be
because
before
been
between
but
even
for
had
if
in
into
is
it
nor
of
on
once
or
since
so
that
the
though
till
to
too
unless
until
was
what
when
whenever
wherever
whether
which
while
yet
you
```

Figure 46: Unwanted Words list

This last is far from exhaustive, and there are many words that could be added to this list. For example, “there”, “your” and “where”.

Adding these words to the list is a simple process – all that is required is that a new line is added to the “UnwantedWords.txt” file. Although the words are in alphabetical order, this is not necessary, but is simply to increase the readability of the file’s content.

It is fairly subjective deciding if a word has enough semantic meaning to be excluded from the list. But this is a not a big issue due to the easiness of editing this file.

An increase in words to this list would ensure that only words with semantic meaning are evaluated by the language model, and the language model is likely to return more accurate results when evaluating a sentence.

10.6 Cryptic Crossword Clue Output

Currently, the results of the cryptic crossword clues are outputted to the console, which allows the user to view the scores of each appended word, and which has the greatest score.

However, it may be beneficial to write this output to a text file, so that the results are stored permanently rather than immediately being deleted upon clearing the console or running another sentence.

This change is rather easy to implement, and prevents the loss of data.

10.7 Obscure Words

The language model rarely gave good scores to obscure words that are not very common in the English language. The model would always award a word such as “good” a much higher score than it would award a word such as “askew” even if the latter was a better fit for that particularly sentence. The reason for this, is that the model is based on frequency and occurrences. However, this creates a bias towards common words.

10.8 Other Natural Languages

Currently, the language model only works with languages that use English alpha-numeric characters. Any words that contain accented letters are simply evaluated without the accents or punctuation removed. Words such as “J’achète” (French: I am buying) become “jachete”. Although this isn’t very accurate, minor errors such as this are negligible within the size of the corpus.

The language model is unable to deal with languages that use non-alpha-numeric characters, such as Russian or Korean. This is due to a regular expressions function that removes these characters.

```
word = word.replaceAll("[^A-Za-z0-9]", "").toLowerCase();
```

Figure 47: RegEx non-alpha-numeric character removal

In the future, it may ideal to introduce a language setting that allows languages such as Russian or Korean to be evaluated. Upon selecting a language sentence, the language model could treat the content differently, and use a different regular expression to filter it.

10.9 Corpus Storage

Currently, the content of the corpus being read is stored as a string variable. This is not ideal, as a string has 10 bytes (Microsoft, n.d.), which allows it to store up to approximately two billion characters.

This can cause issues in the future, when dealing with particularly large corpora. If a corpus contains over two billion characters, then the entire content would not be able to be read.

In the future, the corpus could possibly be read one paragraph at a time. Upon loading a paragraph, the information could be extracted (for unigram and bigram counts) and then the model moves on to the next paragraph. This ensures that the entire corpus can be read, rather than being cut off because it is too large.

10.10 Corpora Management

In the current state, the corpora are stored in a directory as text files. This makes the corpora very easy to access, and it is very easy to load a desired corpus.

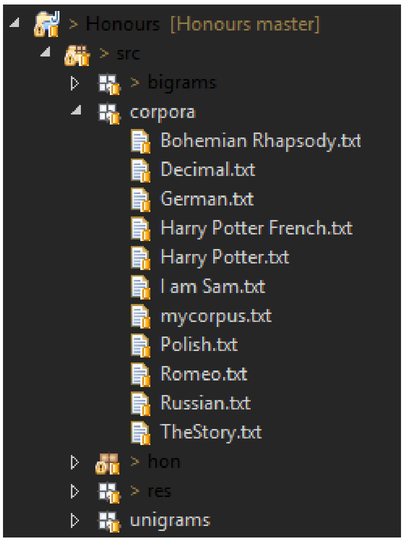


Figure 48: Corpora Storage

However, this introduces some issues in regards of security and confidentiality. In the future, a feature may be introduced that allows users to upload their own corpora. Since the corpora is so freely accessible, it may be dangerous to have the corpora stored like this.

If a user uploads a corpus which contains personal or confidential information, it is a risk to have the corpora to be so easily accessible. The submitted corpora may be able to be accessed by an unauthorised user (either deliberately or accidentally) which would reveal this confidential information.

The corpora should be stored more securely in the future, especially if user's have the opportunity to use their own corpora.

10.11 Documentation

The language model currently lacks any form of documentation, in terms of user and technical guides. This documentation is useful to have so that users have documentation to refer to if they do not understand the software, or need something to refer to in order to work out how to use the software.

A technical guide would also be useful, in the event that another developer chooses to continue working on the language model. The technical guide contains information that would allow a future developer to work on the language model, as it would contain information on the inner workings of the software that may not be easy to tell at a glance. The technical guide can also be used as a reference by the developer, and it would save a lot of time, by not requiring them to research and inspect the current model.

11 References

Chambers, N., Tetreault, J., & Allen, J. (n.d.). *Approaches for Automatically Tagging Affect.* Retrieved from University of Rochester: https://www.cs.rochester.edu/

Dale, R., & Reiter, E. (n.d.). *Building Natural Language Generation Systems.* Retrieved from http://assets.cambridge.org/97805216/20369/sample/9780521620369wsn01.pdf

Fletcher, W. H. (2011, June 27). *Phrases in English*. Retrieved from British National Corpus: http://phrasesinenglish.org/

Google. (n.d.). *Google N-Gram Viewer*. Retrieved from https://books.google.com/ngrams

gov.uk. (2017). *Data protection*. Retrieved from https://www.gov.uk/data-protection/the-data-protection-act

Jurafsky, D. (n.d.). *Language Modelling.* Retrieved from http://spark-public.s3.amazonaws.com/nlp/slides/languagemodeling.pdf

Kuhn, R., & De Mori, R. (1990, June 6). *A Cache-Based Natural Language Model for Speech Recognition.* Retrieved from http://visgraph.cs.ust.hk/biometrics/Papers/Voice/pami1990-06-01.pdf

Language Models. (2016, 10 27). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Language_model

Microsoft. (n.d.). *Data Type Summary*. Retrieved from https://msdn.microsoft.com/en-us/library/aa263420(v=vs.60).aspx

Oracle. (2015). Retrieved from https://docs.oracle.com/javase/tutorial/figures/i18n/i18n-4.gif

Oracle. (2015). Retrieved from https://docs.oracle.com/javase/tutorial/i18n/text/sentence.html

Seuss, D. (1960). In D. Seuss, *Green Eggs and Ham* (p. 8).

Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., & Chanona-Hernández, L. (n.d.). *Syntactic N-grams as Machine Learning Features.* Retrieved from http://www.g-sidorov.org: http://www.g-sidorov.org/Synt_n_grams_ESWA_FINAL.pdf

Waters, K. (2007). *What Is Agile? (10 Key Principles of Agile)*. Retrieved from All About Agile: http://www.allaboutagile.com/what-is-agile-10-key-principles/

12 List of Appendices

Appendix A: Project Poster

Student: Jordan Aitken
Supervisor: Dr. John Owens
Second Marker: Dr. Malcolm Rutter

Semantic Word Association: Creating the Semantic Bi-Gram

Introduction

Natural Language Processing (NLP) is an open problem that deals with how computers "understand" human languages.

One method of NLP is using n-grams. N-grams work by counting how often a sequence of words appear, with n representing the number of words in the sequence.

The	man	sat	down

The man
man sat
sat down

This is an example of an n-gram where n is equal to 2. This is also known as a bi-gram, and it counts the number of occurrences of word pairs.

However, using the n-gram language model does have its limitations - specifically in regards to its inability to deal with words out of its range, and with sentences in another order - an n-gram can only use sequences it has seen before.

Aims

The goal of this project was to create a new language model, that looks at the semantics of words and sentences, rather than sequences of words. The goals of the new language model were to:-

- Read in a large corpus (around 800,000 words) one sentence at a time.

- Have the option of removing words with little semantic meaning (the, and, but, etc)

- Calculate the semantic bigram counts by counting how often words appear together in the same sentence.

- Evaluate a sentence the software has never seen before, and assign a probability value.

- Take an unfinished cryptic crossword clue, and find the most suitable word out of a possible 773 words, to append to the end of the clue.

Summary

The semantic bigram language model was created with the ability to read in a large corpus, and calculate the semantic bigram counts. The software can then query sentences it has not seen before against the bigram counts to evaluate the probability or likelihood of the sentence.

This means that the software can take an unfinished sentence, and query a large list of possible words to find the most suitable word to complete the sentence.

Methodology

Design:

- One of the key parts of using an n-gram model is the corpus. A corpus is simply a huge collection of written texts.

- Initially, a very small corpus was used for testing, which was a poem written by Dr. Seuss.

- The corpus used in this project is a collection of texts from the OANC (Open American National Corpus) and contains approximately 820,000 words. This contains mostly general language, to prevent overuse of jargon or esoteric words.

- The language model reads in the corpus sentence-by-sentence, and then counts how often a word appears with each other word in the sentence.

- The software then stores the counts inside of a very large tree.

- The user can submit a sentence, and the language model will evaluate the sentence, by comparing each word permutation with the corresponding value in the tree and then normalising the data.

- The user also has the option to remove words with little meaning (the, but, was, etc).

- Cryptic crossword clues were chosen as a use for the language model as semantics are important, but word flow is not necessarily required.

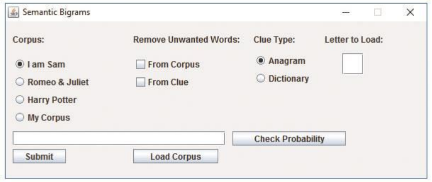
- There is a built in list of 773 words that can be used as anagram indicators in a cryptic crossword clue.

-The Language Model runs through this list, and compares each one to each other to find the most suitable word with the highest score.

- There is also a dictionary option which allows the user to select a letter and find the most suitable word beginning with that letter.

Implementation:

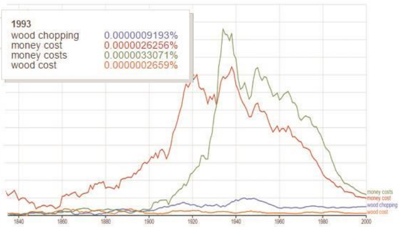
- Created in Eclipse
- Programmed in Java



Results

The software consistently returned words that matched with Google's n-gram viewer, which uses corpora with billions of words.

Start wood for money + C-word	
57.3	Start wood for money can
43.4	Start wood for money cost
35.5	Start wood for money costs
24.7	Start wood for money city
24.0	Start wood for money con
23.2	Start wood for money chief
21.0	Start wood for money certain
19.7	Start wood for money case
18.9	Start wood for money come
18.4	Start wood for money care
17.7	Start wood for money came
17.4	Start wood for money cash



Conclusion & Future Work

In conclusion, the language works as an alternative to the n-gram language model. The software can consistently select suitable words from a given list and, within the top ten results, there is often a word that is appropriate to the clue. The software is also very portable, and it can be used with any corpus.

- Often fails to select the best word by itself, and typically ranks the best word between 5th and 10th place - requires human supervision.

- Run-time is very long - could be optimised.

- Has a bias towards certain words or indicators - this can be improved by a larger corpus.

- Uncommon words are rarely selected - words such as "askew" and "deviously" are often granted poor scores.

- Only tested on one large corpus - the corpus has an heavy impact on what the model returns, it would have been ideal to have used multiple large corpora.

Appendix B: Second Formal Review Output

Shown below are the results of the interim review meeting with Dr. John Owens, myself and Malcolm Rutter.

SOC10101 Honours Project (40 Credits)

Report on the Interim Review Meeting

Student Name: Jordan Aitken
Matriculation Number: 4013 6554
Supervisor: John Owens
Second Marker: Malcolm Rutter
Date of Meeting: 16.11.16

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? **yes** **no***

If not, please comment on any reasons presented

Student has a handful of notes from meetings. = Notes ~~[illegible]~~ prepared by Supervisor.

Please comment on the progress made so far

Student agrees that he is behind schedule at the moment.
Has instead been prioritising other modules.
Has (last year) done string manipulation and regular expression.
Has created 4-5 pp of literature review, involving 3 references.
Student expects to catch up over Christmas Holiday.

Is the progress satisfactory? **yes** **no*** (1)

Can the student articulate their aims and objectives? **yes** **no*** Probably not

If yes then please comment on them, otherwise write down your suggestions.

(1) Student is behind schedule.

* Aim is to build a semantic ~~[illegible]~~ N-gram and evaluate it.
(a bigram is the probability of 2 particular words being adjacent)
First step: Calculate bigrams. Check accuracy manually.
Second step: bigrams on larger corpuses.
Third step: Calculate N-grams.

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

Interim Review Meeting: Page 1 of 2

Does the student have a plan of work? **yes** / **no***

If yes then please comment on that plan otherwise write down your suggestions.

1. Draw up a Gantt chart.
2. Finish literature review by end of Christmas holiday.
3. Finish any software activity by March 1st
4. March 1st ... 14th Evaluate
5. March 14th .... April 14th Write up.

will need consideration and refinement.

Does the student know how they are going to evaluate their work? **yes** / **no***

If yes then please comment otherwise write down your suggestions.

Compare the results of a bigram.

Any other recommendations as to the future direction of the project

(We ran out of time)

Signatures: Supervisor [illegible] Second Marker [illegible]

Student [illegible]

16/11/16

Please give the student a photocopy of this form immediately after the review meeting; the original should be lodged in the School Office with Leanne Clyde

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

Interim Review Meeting: Page 2 of 2

Appendix C Diary Sheets

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jordan Aitken **Supervisor: John Owens**

Date: 13/1/17 **Last diary date: --**

Objectives:

Read in text file
Remove unwanted words
Break into sentences
Break into words
Remove punctuation/general tidy up of words (Removing multiple spaces etc.)
Calculate unigrams
Calculate SBigrams (Extra)

Progress:

All tasks completed, except from SBigrams

Some minor bugs

- Mr. and Mrs. Breaks a sentence
- Punctuation is left in
- Removing a word leaves an empty space, which means multiple spaces
- Unigrams count removed words

Supervisor's Comments:

You seem to be getting on well tackling some of the programming tasks. Progress is fine.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jordan Aitken **Supervisor: John Owens**

Date: 20/1/17 **Last diary date: 13/1/17**

Objectives:

Fix the following errors:

- Mr. and Mrs. Breaks a sentence
- Punctuation is left in
- Removing a word leaves an empty space, which means multiple spaces
- Unigrams count removed words

Change the methods so that the program takes one sentence at a time, extracts what it needs, then moves on. Change unigram and bigram treemaps to instance variables so they are not constantly rewritten.

Calculate the bigram counts

Progress:

All outstanding errors fixed except from Mr. and Mrs. Breaking sentences

Program now reads in one sentence at a time, removes the unwanted words, and puts the words into an ArrayList that can used for unigram and bigram counts. (Unigram counts work)

Began work on bigram counts using nested treemaps

Supervisor's Comments:

The coding is going well. The bigram model is difficult but you now seem to fully understand it. Progress is good.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jordan Aitken **Supervisor: John Owens**

Date: 27/1/17 **Last diary date: 20/1/17**

Objectives:

Understand the SBigram model
- Test the SBigram thoroughly
- Calculate total occurrences of each SBigram (bidirectional)
- Identify suitable corpora

Progress:

Tested the SBigram model with the following test cases:
- Extract from book (Harry Potter)
- A full novel (Harry Potter)
- Shakespeare (Romeo & Juliet)
- French, Polish, Russian and German
- Song (Bohemian Rhapsody)

The SBigram accurately counts the SBigrams under all test cases where there are no non-English characters (such as “wyraźniej” or “появляется”)

The SBigram model counts page numbers and often counts blank characters (“”)

Songs often do not work completely, because there tends to be no punctuation in songs (It’s one long sentence)

Added a way to search for the number of occurrences given two values (currently doesn’t work if both values are the same)

Supervisor’s Comments:

Progress is very good and it is clear that you fully understand and have fully tested the SBigram – well done.

Aim to implement the “normalised count” method and keep looking for an appropriate large corpus

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jordan Aitken **Supervisor: John Owens**

Date: 3/2/17 **Last diary date: 27/1/17**

Objectives:

Take a sentence and break it into each possible sbigram
Find the number of occurrences for each sbigram and calculate the total of each sbigram
Normalise the occurrences by dividing by number of possible permutations

Research corpora and find a (or multiple) suitable large-scale corpus.

Progress:

Any given sentence can be evaluated and given a probability according to the corpus chosen (eg. "The wizard was at Hogwarts" has a much higher probability in the Harry Potter corpus than the Romeo and Juliet corpus)

Normalisation formula used is:

$$p = o \,/\, \left(\frac{n(n-1)}{2} \right)$$

Where:

n = number of words in the sentence

o = total number of occurrences of the sbigrams

p = probability

Using "The wizard was at Hogwarts":

Harry Potter: 215.8

Romeo and Juliet: 14.7

The function appears to be working correctly, but I would like to test it a bit more.

Looked into the Stanford corpora library, but haven't decided on one yet. (Must be a text corpus – not speech or any audio)

Supervisor's Comments:

Progress is fine, keep going in the same direction.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jordan Aitken **Supervisor: John Owens**

Date: 8/3/17 **Last diary date: 3/2/17**

Objectives:

Add the functionality of removing words from clues Research corpora

Progress:

Unwanted words can now be removed from clues Corpus still has to be found.

Supervisor's Comments:

Removing unwanted words is essential and so that is good progress

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jordan Aitken **Supervisor: John Owens**

Date: 10/3/17 **Last diary date: 3/2/17**

Objectives:

Create clues using a string with an appended anagram indicator to evaluate the probability of each.
Create an object for each anagram indicator, and sort them by probability.

Progress:

Clues can now be created.

Implemented a RankedClue object, with attributes of clue and probability.

Software now sorts the clues by probability

Supervisor's Comments:

You are getting closer to being able to test your model

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jordan Aitken **Supervisor: John Owens**

Date: 8/3/17 **Last diary date: 10/3/17**

Objectives:

Add a GUI to the program
Build a corpus of one to two million words

Progress:

Working GUI has been added, now it is possible to query clues without loading the corpus every time. Would like to test this a bit more.

A corpus of 831,034 words has been created.

Supervisor's Comments:

A GUI and a corpus in one week – good work

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jordan Aitken **Supervisor: John Owens**

Date: 17/3/17 **Last diary date: 13/3/17**

Objectives:

Implement the Dictionary class, which can be used to query dictionary words. Complete the evaluation sheet using the large corpus.

Progress:

Dictionary class has been implemented and can be used to query dictionary words. Unable to fulfil evaluation sheet due to very large runtime (>9 hours)

Supervisor's Comments:

The evaluation results will form an important part of your discussion in your report.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jordan Aitken **Supervisor: John Owens**

Date: 25/3/17 **Last diary date: 17/3/17**

Objectives:

Complete Evaluation Sheet

Progress:

Evaluation Sheet has now been completed using the Open National American Corpus.

Supervisor's Comments:

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jordan Aitken **Supervisor: John Owens**

Date: 01/4/17 **Last diary date: 25/3/17**

Objectives:

Redo Evaluation Sheet, in light of the new information gathered (Removal of unwanted words, allow a bit more lenience)

Progress:

Evaluation Sheet has now been completed.

Supervisor's Comments:

Appendix D: Dictionary Words Evaluation

Dictionary Words

Average Rating	/10
Thoughts	Fairly hit-or-miss. The software returned many good dictionary words, but was not very good at selecting the best one. If the software were to be used to generate crossword clues, I feel that it is unable to be fully automated, and would need human supervision to assist it in select the best clues.

Clue: American leader getting cheeky again
Test: leader getting cheeky again
Letter: A
Answer: Afresh

25.5	leader getting cheeky again and
16.6	leader getting cheeky again a
8.4	leader getting cheeky again all
6.1	leader getting cheeky again as
4.6	leader getting cheeky again at
3.8	leader getting cheeky again are
3.0	leader getting cheeky again an
1.2	leader getting cheeky again afraid
1.0	leader getting cheeky again another
0.8	leader getting cheeky again also
0.7	leader getting cheeky again any
0.7	leader getting cheeky again about
0.6	leader getting cheeky again away
0.5	leader getting cheeky again act
0.4	leader getting cheeky again along

0.0 leader getting cheeky again American

Best word found by software	And leader getting cheeky again
Best word found in list (Subjective)	Another leader getting cheeky again
Overall rating	7/10
Comments	Some dictionary indicators make sense due to the wording of the clue, but none of them really stand out.

Clue: Splendid bottle opener splendid
Test: Splendid opener splendid
Letter: B
Answer: Superb

3.0	Splendid opener splendid by
1.6666666666666667	Splendid opener splendid but
0.6666666666666666	Splendid opener splendid bay
0.6666666666666666	Splendid opener splendid back

0.3333333333333333	Splendid opener splendid bowl
0.3333333333333333	Splendid opener splendid bombay
0.3333333333333333	Splendid opener splendid body
0.3333333333333333	Splendid opener splendid boat
0.3333333333333333	Splendid opener splendid bird
0.3333333333333333	Splendid opener splendid beyond
0.3333333333333333	Splendid opener splendid bears

0.0 **Splendid opener splendid bottle**

Best word found by software	Splendid by splendid opener
Best word found in list (Subjective)	Splendid bowl splendid opener
Overall rating	4/10
Comments	Not very good. The word "Splendid" didn't get many hits, and so the words don't make much sense.

Clue: Cobbler initially records footwear
Test: initially records footwear
Letter: C
Answer: Clogs

1.1666666666666667	initially records footwear cable
0.6666666666666666	initially records footwear city
0.5	initially records footwear can
0.3333333333333333	initially records footwear cut
0.3333333333333333	initially records footwear copy
0.16666666666666666	initially records footwear cross
0.16666666666666666	initially records footwear create
0.16666666666666666	initially records footwear counts
0.16666666666666666	initially records footwear cost
0.16666666666666666	initially records footwear content
0.16666666666666666	initially records footwear close
0.16666666666666666	initially records footwear claim
0.16666666666666666	initially records footwear church
0.16666666666666666	initially records footwear certain
0.16666666666666666	initially records footwear cash
0.16666666666666666	initially records footwear case
0.16666666666666666	initially records footwear car

Best word found by software	Cable initially records footwear
Best word found in list (Subjective)	City initially records footwear
Overall rating	7/10
Comments	Some of the dictionary indicators make sense, but I feel this is more coincidence rather a suitable word being found.

Clue: Start chopping wood for money
Test: Start wood for money
Letter: C
Answer: Cash

57.3	Start wood for money can
43.4	Start wood for money cost
35.5	Start wood for money costs
24.7	Start wood for money city
24.0	Start wood for money con
23.2	Start wood for money chief
21.0	Start wood for money certain
19.7	Start wood for money case
18.9	Start wood for money come
18.4	Start wood for money care
17.7	Start wood for money came

17.4 Start wood for money cash
17.2 Start wood for money cut
17.1 Start wood for money car
16.4 Start wood for money church
16.1 Start wood for money class
16.0 Start wood for money customers
15.7 Start wood for money cannot
15.3 Start wood for money count
15.3 Start wood for money cable

Best word found by software	Start can wood for money
Best word found in list (Subjective)	Start cost wood for money
Overall rating	9/10
Comments	Cost and costs are suitable words (due to money) although it doesn't make much sense in the context of the clue's wording.

Clue: Poets start drinking in pubs
Test: Poets start in pubs
Letter: D
Answer: Bards

44.6 Poets start in pubs do
32.4 Poets start in pubs data
30.2 Poets start in pubs did
28.4 Poets start in pubs day
20.1 Poets start in pubs direct
19.8 Poets start in pubs diamonds
17.2 Poets start in pubs debt
16.7 Poets start in pubs due
16.4 Poets start in pubs decline
14.7 Poets start in pubs dear
13.8 Poets start in pubs desert
12.7 Poets start in pubs domain
12.4 Poets start in pubs done
12.3 Poets start in pubs date

Best word found by software	Poets start do in pubs
Best word found in list (Subjective)	Poets start date in pubs
Overall rating	7/10
Comments	Date is a suitable word if it is tweaked a little.

Clue: Might death start to appear unnatural
Test: Might start to appear unnatural
Letter: D
Answer: Forced

79.06666666666666 Might start to appear unnatural do
57.6 Might start to appear unnatural direct
55.86666666666667 Might start to appear unnatural data
54.333333333333336 Might start to appear unnatural did
53.333333333333336 Might start to appear unnatural day
48.8 Might start to appear unnatural debt
46.733333333333334 Might start to appear unnatural due
41.86666666666667 Might start to appear unnatural dog
40.86666666666667 Might start to appear unnatural date
40.333333333333336 Might start to appear unnatural desire
40.266666666666666 Might start to appear unnatural dear
39.6 Might start to appear unnatural decide
39.53333333333333 Might start to appear unnatural deal
39.46666666666667 Might start to appear unnatural decline

Best word found by software	Might do start to appear unnatural
Best word found in list (Subjective)	Might data start to appear unnatural
Overall rating	8/10
Comments	The wording of the clue is a bit tricky as it is fairly irregular.

Clue: Cut out tax before start of December
Test: Cut out tax before start of
Letter: D
Answer: Excised

137.38095238095238	Cut out tax before start of direct
130.04761904761904	Cut out tax before start of data
126.33333333333333	Cut out tax before start of do
119.28571428571429	Cut out tax before start of did
117.52380952380952	Cut out tax before start of day
117.14285714285714	Cut out tax before start of debt
110.85714285714286	Cut out tax before start of due
108.66666666666667	Cut out tax before start of date
107.42857142857143	Cut out tax before start of dead
106.95238095238095	Cut out tax before start of decline
106.71428571428571	Cut out tax before start of done
106.52380952380952	Cut out tax before start of desert
106.19047619047619	Cut out tax before start of domain
106.0952380952381	Cut out tax before start of deposit
106.04761904761905	Cut out tax before start of director
105.80952380952381	Cut out tax before start of deep
105.23809523809524	Cut out tax before start of dear
105.19047619047619	Cut out tax before start of deal
105.04761904761905	Cut out tax before start of dog

Best word found by software	Cut out tax before start of direct
Best word found in list (Subjective)	Cut out tax before start of day
Overall rating	8/10
Comments	Some of the clues make sense, but I feel this is just the more common D-words.

Clue: Thin boxes empty initially
Test: Thin boxes initially
Letter: E
Answer: Sparse

1.6666666666666667	Thin boxes initially equal
0.6666666666666666	Thin boxes initially even
0.3333333333333333	Thin boxes initially extra
0.3333333333333333	Thin boxes initially environment
0.3333333333333333	Thin boxes initially entire
0.3333333333333333	Thin boxes initially end
0.3333333333333333	Thin boxes initially else
0.3333333333333333	Thin boxes initially elaborate
0.3333333333333333	Thin boxes initially eat
0.3333333333333333	Thin boxes initially east

Best word found by software	Thin boxes equal initially
Best word found in list (Subjective)	Thin boxes equal initially
Overall rating	7/10
Comments	The word the software picked as the highest rated word works quite well, but

	some of the other ones in the list aren't very good.

Clue: Row about head of ginger cat
Test: Row about head of cat
Letter: G
Answer: Tiger

114.4	Row about head of cat general
108.06666666666666	Row about head of cat great
106.13333333333334	Row about head of cat good
92.4	Row about head of cat given
89.53333333333333	Row about head of cat get
85.46666666666667	Row about head of cat go
81.0	Row about head of cat god
80.06666666666666	Row about head of cat going
79.66666666666667	Row about head of cat green
79.0	Row about head of cat girls
77.33333333333333	Row about head of cat gallery
77.13333333333334	Row about head of cat grounds
76.93333333333334	Row about head of cat gave
76.06666666666666	Row about head of cat grasp
76.0	Row about head of cat gas
76.0	Row about head of cat game
75.66666666666667	Row about head of cat girl
75.6	Row about head of cat guard
75.4	Row about head of cat grow

Best word found by software	Row about head of general cat
Best word found in list (Subjective)	Row about head of great cat
Overall rating	8/10
Comments	The word the software picked works well, and there are some words in the list that also make sense. However, I feel that this is coincidence, due to how often the G-words appear.

Clue: Geoff's beginning to wander in a small wood
Test: beginning to wander in a small wood
Letter: G
Answer: Grove

1267.4642857142858	beginning to wander in a small wood good
1250.4642857142858	beginning to wander in a small wood great
1248.357142857143	beginning to wander in a small wood general
1245.5357142857142	beginning to wander in a small wood given
1244.4285714285713	beginning to wander in a small wood get
1243.7142857142858	beginning to wander in a small wood go
1230.5714285714287	beginning to wander in a small wood going
1230.0357142857142	beginning to wander in a small wood got
1228.7142857142858	beginning to wander in a small wood green
1226.892857142857	beginning to wander in a small wood god
1226.8214285714287	beginning to wander in a small wood girl
1225.9642857142858	beginning to wander in a small wood gave
1225.642857142857	beginning to wander in a small wood girls
1224.7857142857142	beginning to wander in a small wood grounds
1224.75	beginning to wander in a small wood game

Best word found by software	Good beginning to wander in a small wood
Best word found in list (Subjective)	Green beginning to wander in a small wood
Overall rating	6/10
Comments	The word the software picked works decently, and there are some words in

	the list that also make sense. However, I feel that this is coincidence, due to how often the G-words appear.

Clue: Sovereign the family start grabbing
Test: Sovereign the family start
Letter: G
Answer: King

176.1	Sovereign the family start great
162.9	Sovereign the family start good
158.6	Sovereign the family start general
128.8	Sovereign the family start get
125.8	Sovereign the family start given
117.3	Sovereign the family start go
95.8	Sovereign the family start green
93.8	Sovereign the family start god
92.1	Sovereign the family start going
89.1	Sovereign the family start girls
87.6	Sovereign the family start got
86.6	Sovereign the family start gave

Best word found by software	Sovereign the family start great
Best word found in list (Subjective)	Sovereign the family start going
Overall rating	7/10
Comments	Similar to the previous clues, the words picked make sense, but I feel this is more coincidence rather than good words being selected.

Clue: A labour leader taking many shares out
Test: A leader taking many shares out
Letter: L
Answer: Allots

93.04761904761905	A leader taking many shares out like
79.33333333333333	A leader taking many shares out long
75.66666666666667	A leader taking many shares out large
72.9047619047619	A leader taking many shares out less
72.47619047619048	A leader taking many shares out legal
72.33333333333333	A leader taking many shares out later
72.19047619047619	A leader taking many shares out land
72.0	A leader taking many shares out language
71.85714285714286	A leader taking many shares out left
71.52380952380952	A leader taking many shares out local
70.19047619047619	A leader taking many shares out last
69.95238095238095	A leader taking many shares out law
69.76190476190476	A leader taking many shares out love
69.23809523809524	A leader taking many shares out least
69.04761904761905	A leader taking many shares out light

Best word found by software	A like leader taking many shares out
Best word found in list (Subjective)	A large leader taking many shares out
Overall rating	8/10
Comments	A lot of the words chosen make sense. The word that the software picked as the best is also decent.

Clue: Clothing provided initially by social workers
Test: Clothing initially by social workers
Letter: P
Answer: Pants

29.4	Clothing initially by social workers people
28.733333333333334	Clothing initially by social workers personal
27.466666666666665	Clothing initially by social workers part
26.6	Clothing initially by social workers place
25.2	Clothing initially by social workers policy
24.933333333333334	Clothing initially by social workers period
24.933333333333334	Clothing initially by social workers pay
24.666666666666668	Clothing initially by social workers per
24.266666666666666	Clothing initially by social workers park
24.266666666666666	Clothing initially by social workers pan
24.133333333333333	Clothing initially by social workers plan
23.666666666666668	Clothing initially by social workers press
23.6	Clothing initially by social workers present
23.466666666666665	Clothing initially by social workers past
23.266666666666666	Clothing initially by social workers page
23.133333333333333	Clothing initially by social workers popular
23.066666666666666	Clothing initially by social workers paris
22.8	Clothing initially by social workers port
22.8	Clothing initially by social workers party

Best word found by software	Clothing people initially by social workers
Best word found in list (Subjective)	Clothing personal initially by social workers
Overall rating	8/10
Comments	Both ‘people’ and ‘personal’ are suitable words (due to social workers) and there are more suitable words in the list.

Clue: Party leader is not pleased with gifts
Test: leader is not pleased with gifts
Letter: P
Answer: Presents

219.95238095238096	leader is not pleased with gifts pan
183.04761904761904	leader is not pleased with gifts people
176.9047619047619	leader is not pleased with gifts place
176.57142857142858	leader is not pleased with gifts part
173.76190476190476	leader is not pleased with gifts personal
171.76190476190476	leader is not pleased with gifts park
171.0	leader is not pleased with gifts popular
170.04761904761904	leader is not pleased with gifts present
168.76190476190476	leader is not pleased with gifts person
167.71428571428572	leader is not pleased with gifts policy
167.71428571428572	leader is not pleased with gifts past
167.42857142857142	leader is not pleased with gifts plan
167.42857142857142	leader is not pleased with gifts period
167.38095238095238	leader is not pleased with gifts port
167.28571428571428	leader is not pleased with gifts put
166.8095238095238	leader is not pleased with gifts pay
166.33333333333334	leader is not pleased with gifts page
166.1904761904762	leader is not pleased with gifts palace
166.0	leader is not pleased with gifts paris
165.8095238095238	leader is not pleased with gifts president
165.52380952380952	**leader is not pleased with gifts party**

Best word found by software	Pan leader is not pleased by gifts
Best word found in list (Subjective)	People leader is not pleased by gifts (Honourable mention to ‘present’)
Overall rating	8/10
Comments	Many words make sense in the context of the clue. Both “party” and “present” ranked fairly highly on the list. But I feel this is more coincidence rather than a suitable clue being found.

Clue: Singer producing sick note before start of recital
Test: Singer producing sick note before start of
Letter: R
Answer: Tenor

42.861111111111114	Singer producing sick note before start of retirement
39.27777777777778	Singer producing sick note before start of rate
36.52777777777778	Singer producing sick note before start of river
36.22222222222222	Singer producing sick note before start of right
35.888888888888886	Singer producing sick note before start of result
35.44444444444444	Singer producing sick note before start of related
34.77777777777778	Singer producing sick note before start of royal
34.72222222222222	Singer producing sick note before start of range
34.583333333333336	Singer producing sick note before start of real
34.22222222222222	Singer producing sick note before start of required
33.97222222222222	Singer producing sick note before start of rates
33.72222222222222	Singer producing sick note before start of recent
33.69444444444444	Singer producing sick note before start of role
33.333333333333336	Singer producing sick note before start of remains
33.083333333333336	Singer producing sick note before start of rue
27.583333333333332	**Singer producing sick note before start of recital**

Best word found by software	Singer producing sick note before start of retirement
Best word found in list (Subjective)	Singer producing sick note before start of range
Overall rating	8/10
Comments	The word chosen by the software makes sense, and so do some of the words in the list.

Clue: Family with first of loaves in oven
Test: Family with first of in oven
Letter: L
Answer: Kiln

1392.047619047619	Family with first of in oven like
1378.904761904762	Family with first of in oven land
1374.7619047619048	Family with first of in oven less
1374.3333333333333	Family with first of in oven legal
1373.2857142857142	Family with first of in oven long
1370.095238095238	Family with first of in oven language
1369.095238095238	Family with first of in oven last
1369.047619047619	Family with first of in oven large
1368.904761904762	Family with first of in oven loan
1367.047619047619	Family with first of in oven later
1365.6666666666667	Family with first of in oven local
1365.6666666666667	Family with first of in oven law
1363.0	Family with first of in oven left
1360.6190476190477	Family with first of in oven least
1359.3809523809523	Family with first of in oven light

Best word found by software	Family with first of like in oven
Best word found in list (Subjective)	Family with first of land in oven
Overall rating	7/10
Comments	Some of the words make sense, but there isn't a strong semantic connection between the words.

Clue: Artificial people start running with kinky boots
Test: Artificial people start with kinky boots
Letter: R
Answer: Robots

13.238095238095237	Artificial people start with kinky boots right
12.571428571428571	Artificial people start with kinky boots retirement
11.714285714285714	Artificial people start with kinky boots rate
11.523809523809524	Artificial people start with kinky boots real
11.19047619047619	Artificial people start with kinky boots result
11.142857142857142	Artificial people start with kinky boots river
10.904761904761905	Artificial people start with kinky boots royal
10.857142857142858	Artificial people start with kinky boots room
10.714285714285714	Artificial people start with kinky boots region
10.666666666666666	Artificial people start with kinky boots red
10.619047619047619	Artificial people start with kinky boots rue
10.619047619047619	Artificial people start with kinky boots range
10.523809523809524	Artificial people start with kinky boots role
10.476190476190476	Artificial people start with kinky boots rooms
10.476190476190476	Artificial people start with kinky boots road
10.380952380952381	Artificial people start with kinky boots rat
10.285714285714286	Artificial people start with kinky boots related
10.285714285714286	Artificial people start with kinky boots recent
10.238095238095237	Artificial people start with kinky boots remains
10.19047619047619	Artificial people start with kinky boots rich

Best word found by software	Artificial people start right with kinky boots
Best word found in list (Subjective)	Artificial people start retirement with kinky boots
Overall rating	7/10
Comments	Some of the words make sense, but there isn’t a very strong semantic connection.

Clue: Scottish leader with horse problem
Test: leader with horse problem
Letter: S
Answer: Snag

26.2	leader with horse problem some
24.6	leader with horse problem see
19.5	leader with horse problem so
18.2	leader with horse problem she
15.2	leader with horse problem set
14.3	leader with horse problem same
14.0	leader with horse problem state
12.2	leader with horse problem since
11.9	leader with horse problem street
11.7	leader with horse problem side
11.6	leader with horse problem still
9.0	leader with horse problem support

Best word found by software	Some leader with horse problem
Best word found in list (Subjective)	State leader with horse problem
Overall rating	7/10
Comments	A lot of the words make sense in context, and a couple have a semantic connection.

Clue: Start pulling strings for situations
Test: Start strings for situations
Letter: P
Answer: Places

48.6	Start strings for situations pan
29.7	Start strings for situations people
19.0	Start strings for situations personal
17.0	Start strings for situations place
15.5	Start strings for situations port
15.4	Start strings for situations part
12.7	Start strings for situations period
12.7	Start strings for situations park
12.1	Start strings for situations pay
12.0	Start strings for situations present
11.9	Start strings for situations plan
11.4	Start strings for situations policy
11.2	Start strings for situations pairs
10.7	Start strings for situations popular
10.1	Start strings for situations per
9.9	Start strings for situations put

Best word found by software	Start pan strings for situations
Best word found in list (Subjective)	Start personal strings for situations
Overall rating	6/10
Comments	No strong words were found, and the words that were found have little semantic connection to each other. It would have been ideal if a verb was found.

Clue: Talk with Spanish leader at summit
Test: Talk with leader at summit
Letter: S
Answer: Speak

70.4	Talk with leader at summit see
69.46666666666667	Talk with leader at summit some
61.666666666666664	Talk with leader at summit so
59.266666666666666	Talk with leader at summit same
58.13333333333333	Talk with leader at summit she
55.6	Talk with leader at summit street
55.13333333333333	Talk with leader at summit set
55.0	Talk with leader at summit state
53.46666666666667	Talk with leader at summit still
53.13333333333333	Talk with leader at summit since
52.6	Talk with leader at summit side
50.86666666666667	Talk with leader at summit sea
50.06666666666667	Talk with leader at summit start
49.8	Talk with leader at summit say
49.733333333333334	Talk with leader at summit support
49.46666666666667	Talk with leader at summit study

47.333333333333336 **Talk with leader at summit Spanish**

Best word found by software	Talk with see leader at summit
Best word found in list (Subjective)	Talk with some leader at summit
Overall rating	7/10
Comments	A lot of words make sense, and there are some semantic connections such as ‘state’ and ‘leader’

Clue: Airbourne youth leader gets cast out
Test: Airbourne leader gets cast out
Letter: Y
Answer: Flying

7.933333333333334	Airbourne leader gets cast out you
0.8	Airbourne leader gets cast out yet
0.7333333333333333	Airbourne leader gets cast out year
0.6	Airbourne leader gets cast out yard
0.4666666666666667	Airbourne leader gets cast out yorkshire

Best word found by software	Airbourne you leader gets cast out
Best word found in list (Subjective)	Airbourne Yorkshire leader gets cast out
Overall rating	2/10
Comments	The words found were quite weak, and there is little semantic connections

Clue: Check the accounts of German car trader first
Test: Check the accounts of German car first
Letter: T
Answer: Audit

5293.571428571428	Check the accounts of German car first the
4782.607142857143	Check the accounts of German car first to
2941.6071428571427	Check the accounts of German car first this
2886.8928571428573	Check the accounts of German car first their
2797.9285714285716	Check the accounts of German car first than
2771.0	Check the accounts of German car first time
2757.5714285714284	Check the accounts of German car first there
2737.5714285714284	Check the accounts of German car first those
2727.6071428571427	Check the accounts of German car first take
2712.6071428571427	Check the accounts of German car first trust
2707.535714285714	Check the accounts of German car first total
2701.964285714286	Check the accounts of German car first times
2699.4285714285716	Check the accounts of German car first too
2698.4285714285716	Check the accounts of German car first top
2694.5714285714284	Check the accounts of German car first though
2694.3571428571427	Check the accounts of German car first table
2693.5714285714284	Check the accounts of German car first turn
2692.6785714285716	Check the accounts of German car first tower
2691.6071428571427	Check the accounts of German car first test
2691.1071428571427	Check the accounts of German car first thus
2689.964285714286	Check the accounts of German car first think
2689.5714285714284	Check the accounts of German car first thought
2689.3214285714284	Check the accounts of German car first trade
2689.0714285714284	Check the accounts of German car first tax

2674.535714285714 **Check the accounts of German car first traders**

Best word found by software	Check the accounts of German car the first
Best word found in list (Subjective)	Check the accounts of German car there first
Overall rating	5/10
Comments	A lot of the words found are unlikely to be useful in any clue, and the semantic connection of the other words isn't particularly strong.

Clue: Concealed lid on top of tin
Test: Concealed lid on top of
Letter: T
Answer: Covert

6012.266666666666	Concealed lid on top of the
1937.8	Concealed lid on top of to
649.8	Concealed lid on top of this
609.0	Concealed lid on top of their
528.9333333333333	Concealed lid on top of than
500.46666666666664	Concealed lid on top of time
495.06666666666666	Concealed lid on top of there
480.26666666666665	Concealed lid on top of those
477.6666666666667	Concealed lid on top of trust
465.4	Concealed lid on top of take
458.46666666666664	Concealed lid on top of total
451.4	Concealed lid on top of times
451.2	Concealed lid on top of table
448.1333333333333	Concealed lid on top of top
448.06666666666666	Concealed lid on top of too
445.6666666666667	Concealed lid on top of though
445.4	Concealed lid on top of turn
445.4	Concealed lid on top of tax

430.8 **Concealed lid on top of tin**

Best word found by software	Concealed lid on top of the
Best word found in list (Subjective)	Concealed lid on top of the table
Overall rating	7/10
Comments	A lot of the words found are unlikely to be useful in any clue, and the semantic connection of the other words isn't particularly strong. However, the word 'table' was selected, but I feel this is more coincidence rather than finding a suitable word.

Clue: Tin containing first of yellow colour
Test: Tin containing first of colour
Letter: Y
Answer: Cyan

154.4	Tin containing first of colour you
93.93333333333334	Tin containing first of colour year
67.53333333333333	Tin containing first of colour yet
60.6	Tin containing first of colour yeast
59.733333333333334	Tin containing first of colour yield
59.53333333333333	Tin containing first of colour yes
58.666666666666664	Tin containing first of colour yard
58.333333333333336	Tin containing first of colour yorkshire
58.266666666666666	Tin containing first of colour ye

Best word found by software	Tin containing first of you colour
Best word found in list (Subjective)	Tin containing first of yorkshire colour.
Overall rating	2/10
Comments	Overall, a very weak list of words

Appendix E: Anagram Indicator Words Evaluation

Anagram Indicators

Average Rating	/10
Thoughts	Overall, not as good as hoped for. The software had a tendency to return the same 10-15 words, in a different order. The software did occasionally return ideal words, but not very often.

Clue: Torn veils are bad things
Test: veils are bad things
Indicator: Torn
Answer: Evils

32.9 veils are bad things some
21.5 veils are bad things about
21.2 veils are bad things out
19.2 veils are bad things new
18.6 veils are bad things used
17.0 veils are bad things over
16.1 veils are bad things made
12.6 veils are bad things around
12.1 veils are bad things order
12.1 veils are bad things make
10.5 veils are bad things different
10.5 veils are bad things another
10.2 veils are bad things changes
9.4 veils are bad things away
9.0 veils are bad things off
8.5 veils are bad things find
8.3 veils are bad things become

4.5 veils are bad things torn

Best word found by software	Some veils are bad things
Best word found in list (Subjective)	Different veils are bad things
Overall rating	6/10
Comments	Most words don’t make sense, but there a couple of words that do.

Clue: Additional pay for a sincere sort
Test: Additional pay for a sincere
Indicator: sort
Answer: Increase

466.06666666666666	Additional pay for a sincere new
456.0	Additional pay for a sincere some
451.06666666666666	Additional pay for a sincere out
447.0	Additional pay for a sincere over
444.93333333333334	Additional pay for a sincere about
443.6	Additional pay for a sincere used
429.26666666666665	Additional pay for a sincere made
424.1333333333333	Additional pay for a sincere around
422.3333333333333	Additional pay for a sincere make
420.2	Additional pay for a sincere another
413.3333333333333	Additional pay for a sincere order
412.93333333333334	Additional pay for a sincere changes
410.0	Additional pay for a sincere makes
410.0	Additional pay for a sincere built
409.8666666666667	Additional pay for a sincere find
408.26666666666665	Additional pay for a sincere off
407.6666666666667	Additional pay for a sincere change
407.6	Additional pay for a sincere different
406.93333333333334	Additional pay for a sincere variety
406.93333333333334	Additional pay for a sincere building
406.73333333333335	Additional pay for a sincere form

393.2 **Additional pay for a sincere sort**

Best word found by software	Additional pay for a sincere new
Best word found in list (Subjective)	Additional pay for a sincere variety
Overall rating	6/10
Comments	Weak semantic connection between words. However, some of them do make sense.

Clue: Actors playing for Cuban leader
Test: Actors for Cuban leader
Indicator: Playing
Answer: Castro

45.6	Actors for Cuban leader new
37.6	Actors for Cuban leader some
35.3	Actors for Cuban leader used
30.1	Actors for Cuban leader about
28.3	Actors for Cuban leader out
26.2	Actors for Cuban leader over
20.4	Actors for Cuban leader made
17.8	Actors for Cuban leader make
17.8	Actors for Cuban leader another
15.2	Actors for Cuban leader order
14.4	Actors for Cuban leader around
13.6	Actors for Cuban leader changes
11.6	Actors for Cuban leader training
11.0	Actors for Cuban leader possible
10.7	Actors for Cuban leader makes
10.4	Actors for Cuban leader find
10.1	Actors for Cuban leader different
10.0	Actors for Cuban leader change
9.7	Actors for Cuban leader built

3.6 Actors for Cuban leader playing

Best word found by software	Actors new for Cuban leader
Best word found in list (Subjective)	Actors training for Cuban leader
Overall rating	8/10
Comments	The word ‘training’ appeared which is quite interesting. Multiple words make sense, and there are some hints of semantic connections.

Clue: Opera in new setting pout of doors
Test: Opera in setting pout of doors
Indicator: new
Answer: Open air

949.7142857142857	**Opera in setting pout of doors new**
946.2857142857143	Opera in setting pout of doors some
933.5714285714286	Opera in setting pout of doors out
927.047619047619	Opera in setting pout of doors over
919.4761904761905	Opera in setting pout of doors about
901.1904761904761	Opera in setting pout of doors used
892.047619047619	Opera in setting pout of doors around
891.2857142857143	Opera in setting pout of doors made
883.7142857142857	Opera in setting pout of doors another
882.6666666666666	Opera in setting pout of doors order
881.4761904761905	Opera in setting pout of doors changes
879.6190476190476	Opera in setting pout of doors built
878.1904761904761	Opera in setting pout of doors make
875.7142857142857	Opera in setting pout of doors form
872.5714285714286	Opera in setting pout of doors different
870.8095238095239	Opera in setting pout of doors find
870.7619047619048	Opera in setting pout of doors change
868.9047619047619	Opera in setting pout of doors building

Best word found by software	Opera in new setting pout of doors
Best word found in list (Subjective)	Opera in new setting pout of doors
Overall rating	9/10
Comments	The fact that the software chose ‘new’ as the most suitable word is coincidental. However, there are a lot of suitable words in the list.

Clue: Never failing courage
Test: Never courage
Indicator: failing
Answer: Nerve

8.0	Never courage made
6.333333333333333	Never courage some
4.0	Never courage out
4.0	Never courage away
3.6666666666666665	Never courage makes
3.6666666666666665	Never courage about
3.3333333333333335	Never courage poor
3.3333333333333335	Never courage fresh
3.0	Never courage new
2.3333333333333335	Never courage variety
2.0	Never courage turning
2.0	Never courage over
2.0	Never courage order
2.0	Never courage gets
2.0	Never courage another

0.0 **Never courage failing**

Best word found by software	Never made courage
Best word found in list (Subjective)	Never gets courage
Overall rating	5/10
Comments	The software didn't return many strong hits (Due to the short clue length)

Clue: Cilla changed colour
Test: Cilla colour
Indicator: changed
Answer: Lilac

1.3333333333333333	Cilla color false
1.0	Cilla color some
1.0	Cilla color replaced
1.0	Cilla color perhaps
0.6666666666666666	Cilla color out
0.6666666666666666	Cilla color made
0.6666666666666666	Cilla color incorrectly
0.6666666666666666	Cilla color correct
0.6666666666666666	Cilla color about

0.0 **Cilla color changed**

Best word found by software	Cilla false color
Best word found in list (Subjective)	Cilla replaced color
Overall rating	4/10
Comments	Changed 'colour' to 'color' as most of the corpus material uses American-English. The software returned very few matches, and most matches it did return weren't very good.

Clue: Repair broken sword
Test: Repair sword
Indicator: broken
Answer: Rapier

0.6666666666666666	Repair sword move
0.6666666666666666	Repair sword form
0.6666666666666666	Repair sword dance
0.3333333333333333	Repair sword used
0.3333333333333333	Repair sword turn
0.3333333333333333	Repair sword treatment
0.3333333333333333	Repair sword rocky
0.3333333333333333	Repair sword original
0.3333333333333333	Repair sword order
0.3333333333333333	Repair sword new
0.3333333333333333	Repair sword manufactured
0.3333333333333333	Repair sword hectic
0.3333333333333333	Repair sword damaged

0.0 **Repair sword broken**

Best word found by software	Repair move sword
Best word found in list (Subjective)	Repair damaged sword
Overall rating	9/10
Comments	The software returned a surprisingly good word within the list that makes perfect sense, and would make a very suitable clue. However, I feel that this is a one-off.

Clue: Hounds struggling in river
Test: Hounds in river
Indicator: struggling
Answer: Hudson

Clue: Rough terrain for coach

157.66666666666666	Hounds in river new
139.83333333333334	Hounds in river some
124.16666666666667	Hounds in river over
121.0	Hounds in river about
118.16666666666667	Hounds in river out
99.66666666666667	Hounds in river used
78.5	Hounds in river around
76.83333333333333	Hounds in river made
76.83333333333333	Hounds in river changes
72.33333333333333	Hounds in river order
68.0	Hounds in river built
63.333333333333336	Hounds in river another
57.166666666666664	Hounds in river form
54.833333333333336	Hounds in river change
53.333333333333336	Hounds in river make
50.0	Hounds in river find
48.166666666666664	Hounds in river building
47.0	Hounds in river off
46.0	Hounds in river different

21.333333333333332 **Hounds in river struggling**

Best word found by software	Hounds new in river
Best word found in list (Subjective)	Hounds out in river
Overall rating	5/10
Comments	The returned list of words don't have a very strong semantic connection.

Clue: Rough terrain for coach
Test: Rough for coach
Indicator: terrain
Answer: Trainer

72.0	Rough for coach new
59.0	Rough for coach some
55.333333333333336	Rough for coach used
46.166666666666664	Rough for coach about
43.666666666666664	Rough for coach out
39.666666666666664	Rough for coach over
30.5	Rough for coach made
26.333333333333332	Rough for coach make
26.166666666666668	Rough for coach another
21.5	Rough for coach order
20.333333333333332	Rough for coach around
19.166666666666668	Rough for coach changes
15.5	Rough for coach training
14.833333333333334	Rough for coach possible
14.333333333333334	Rough for coach makes

Best word found by software	Rough new for coach
Best word found in list (Subjective)	Rough training for coach
Overall rating	7/10
Comments	“Training” is a good word that appeared in the list, but I feel this is more coincidence rather than a strong word being selected.

Clue: Like a mad dog worrying a bird
Test: Like a mad dog a bird
Indicator: worrying
Answer: Rabid

132.66666666666666	Like a mad dog a bird new
129.57142857142858	Like a mad dog a bird out
125.47619047619048	Like a mad dog a bird some
123.9047619047619	Like a mad dog a bird over
117.14285714285714	Like a mad dog a bird about
107.71428571428571	Like a mad dog a bird used
105.14285714285714	Like a mad dog a bird made
103.23809523809524	Like a mad dog a bird around
96.42857142857143	Like a mad dog a bird make
93.0	Like a mad dog a bird another

Best word found by software	Like a mad dog new a bird
Best word found in list (Subjective)	Like a mad dog over a bird
Overall rating	5/10
Comments	The returned list of words don’t have a very strong semantic connection.

Clue: Playing on organ in Burmese city
Test: on organ in Burmese city
Indicator: playing
Answer: Rangoon

292.73333333333335	on organ in Burmese city new
285.3333333333333	on organ in Burmese city some
277.3333333333333	on organ in Burmese city over
275.8	on organ in Burmese city out
273.73333333333335	on organ in Burmese city about
261.3333333333333	on organ in Burmese city used
254.13333333333333	on organ in Burmese city around
249.53333333333333	on organ in Burmese city made
247.46666666666667	on organ in Burmese city built
246.46666666666667	on organ in Burmese city changes
246.26666666666668	on organ in Burmese city order
245.06666666666666	on organ in Burmese city another
240.66666666666666	on organ in Burmese city make

223.8 **on organ in Burmese city playing**

Best word found by software	New an organ in Burmese city
Best word found in list (Subjective)	Changes on organ in Burmese city
Overall rating	3/10
Comments	Very weak list of words that make little semantic sense.

Clue: Eminent performer in some art form
Test: Eminent performer in some art
Indicator: form
Answer: Maestro

84.85714285714286	Eminent performer in some art new
77.80952380952381	Eminent performer in some art some
74.52380952380952	Eminent performer in some art about
74.28571428571429	Eminent performer in some art over
73.52380952380952	Eminent performer in some art out
67.71428571428571	Eminent performer in some art used
61.42857142857143	Eminent performer in some art around
61.095238095238095	Eminent performer in some art made
60.57142857142857	Eminent performer in some art changes
59.142857142857146	Eminent performer in some art order
57.23809523809524	Eminent performer in some art built
56.23809523809524	Eminent performer in some art another
55.333333333333336	**Eminent performer in some art form**
54.38095238095238	Eminent performer in some art change
53.714285714285715	Eminent performer in some art make
52.904761904761905	Eminent performer in some art find
52.0	Eminent performer in some art building
51.61904761904762	Eminent performer in some art off
51.23809523809524	Eminent performer in some art different

Best word found by software	Eminent performer in some art new
Best word found in list (Subjective)	Eminent performer in some art form
Overall rating	8/10
Comments	Most of the returned words are not very strong, but it did find “form” within the list.

Clue: Gongs for damsel in distress
Test: Gongs for damsel in
Indicator: distress
Answer: Medals

571.4	Gongs for damsel in new
552.7	Gongs for damsel in some
534.4	Gongs for damsel in about
531.0	Gongs for damsel in over
530.3	Gongs for damsel in out
527.0	Gongs for damsel in used
498.5	Gongs for damsel in made
492.7	Gongs for damsel in around
491.8	Gongs for damsel in changes
490.6	Gongs for damsel in order
487.7	Gongs for damsel in another
481.9	Gongs for damsel in make
481.6	Gongs for damsel in built
475.2	Gongs for damsel in change
474.9	Gongs for damsel in form
472.5	Gongs for damsel in find
470.3	Gongs for damsel in building
470.0	Gongs for damsel in different
468.3	Gongs for damsel in possible
467.9	Gongs for damsel in free
467.7	Gongs for damsel in off
466.3	Gongs for damsel in makes
466.1	Gongs for damsel in become
465.7	Gongs for damsel in away
465.4	Gongs for damsel in turn
465.1	Gongs for damsel in training

Best word found by software	Gongs for damsel in new
Best word found in list (Subjective)	Gongs for damsel in training
Overall rating	5/10
Comments	Most of the words returned are very weak with little semantic meaning.

Clue: A vegetable Eric and Alec cooked
Test: A vegetable Eric and Alec
Indicator: cooked
Answer: Celeriac

1045.0	A vegetable Eric and Alec new
1032.9333333333334	A vegetable Eric and Alec out
1029.5333333333333	A vegetable Eric and Alec some
1022.6	A vegetable Eric and Alec about
1021.5333333333333	A vegetable Eric and Alec over
997.4666666666667	A vegetable Eric and Alec used
987.3333333333334	A vegetable Eric and Alec around
987.0	A vegetable Eric and Alec made
975.6	A vegetable Eric and Alec make
966.1333333333333	A vegetable Eric and Alec another
958.2666666666667	A vegetable Eric and Alec order
958.0666666666667	A vegetable Eric and Alec changes
957.2	A vegetable Eric and Alec different
957.0666666666667	A vegetable Eric and Alec off
956.4	A vegetable Eric and Alec find

Best word found by software	A vegetable Eric and Alec new
Best word found in list (Subjective)	A vegetable Eric and Alec made
Overall rating	6/10
Comments	A couple of the returned words make sense, and have a semantic connection.

Clue: Get a poor number of spectators
Test: Get a number of spectators
Indicator: poor
Answer: Gate

1459.3333333333333	Get a number of spectators new
1458.6666666666667	Get a number of spectators some
1451.8666666666666	Get a number of spectators out
1437.1333333333334	Get a number of spectators over
1421.2666666666667	Get a number of spectators about
1397.4	Get a number of spectators used
1390.5333333333333	Get a number of spectators around
1390.2	Get a number of spectators made
1377.2666666666667	Get a number of spectators another
1375.6666666666667	Get a number of spectators make
1367.0	Get a number of spectators order
1365.9333333333334	Get a number of spectators built
1362.6666666666667	Get a number of spectators changes
1361.4666666666667	Get a number of spectators different
1361.2666666666667	Get a number of spectators form
1359.2666666666667	Get a number of spectators find
1357.1333333333334	Get a number of spectators off
1355.9333333333334	Get a number of spectators variety
1354.9333333333334	Get a number of spectators change
1354.7333333333333	Get a number of spectators building
1351.8666666666666	Get a number of spectators away
1351.2666666666667	Get a number of spectators makes
1349.2	Get a number of spectators possible
1348.6	Get a number of spectators become
1348.0666666666666	Get a number of spectators turn
1347.4666666666667	Get a number of spectators free
1344.8666666666666	Get a number of spectators original

1342.0666666666666 **Get a number of spectators poor**

Best word found by software	Get a new number of spectators
Best word found in list (Subjective)	Get a different number of spectators
Overall rating	9/10
Comments	A lot of the returned words make sense and have a semantic connection.

Clue: Aintree fixture for apprentice
Test: Aintree for apprentice
Indicator: fixture
Answer: Trainee

71.66666666666667	Aintree for apprentice new
58.5	Aintree for apprentice some
54.833333333333336	Aintree for apprentice used
45.666666666666664	Aintree for apprentice about
43.166666666666664	Aintree for apprentice out
39.166666666666664	Aintree for apprentice over
29.833333333333332	Aintree for apprentice made
25.666666666666668	Aintree for apprentice make
25.666666666666668	Aintree for apprentice another
21.0	Aintree for apprentice order
19.833333333333332	Aintree for apprentice around
18.666666666666668	Aintree for apprentice changes
15.0	Aintree for apprentice training
14.333333333333334	Aintree for apprentice possible
13.833333333333334	Aintree for apprentice makes
13.166666666666666	Aintree for apprentice find
12.833333333333334	Aintree for apprentice different
12.666666666666666	Aintree for apprentice change
12.166666666666666	Aintree for apprentice built
11.666666666666666	Aintree for apprentice building

Best word found by software	Aintree new for apprentice
Best word found in list (Subjective)	Aintree used for apprentice
Overall rating	7/10
Comments	Some of the returned words are suitable, and have a semantic connection.

Clue: Pleasant Ealing production
Test: Pleasant Ealing
Indicator: production
Answer: Genial

2.0	Pleasant Ealing some
1.3333333333333333	Pleasant Ealing makes
1.3333333333333333	Pleasant Ealing changes
1.1666666666666667	Pleasant Ealing used
1.1666666666666667	Pleasant Ealing over
1.0	Pleasant Ealing letters
1.0	Pleasant Ealing about
0.8333333333333334	Pleasant Ealing out
0.8333333333333334	Pleasant Ealing new
0.8333333333333334	Pleasant Ealing making
0.8333333333333334	Pleasant Ealing change
0.6666666666666666	Pleasant Ealing training
0.6666666666666666	Pleasant Ealing make
0.5	Pleasant Ealing find

Best word found by software	Pleasant Ealing some
Best word found in list (Subjective)	Pleasant Ealing changes
Overall rating	8/10
Comments	Some of the returned words are good, but the software did not record many hits.

Clue: Pardon set many loose
Test: Pardon set many
Indicator: loose
Answer: Amnesty

10.333333333333334	Pardon set many some
10.166666666666666	Pardon set many new
8.333333333333334	Pardon set many about
7.333333333333333	Pardon set many out
7.0	Pardon set many over
6.5	Pardon set many made
6.333333333333333	Pardon set many used
6.0	Pardon set many around
4.166666666666667	Pardon set many find
4.0	Pardon set many off
4.0	Pardon set many changes
4.0	Pardon set many another

Best word found by software	Pardon set many some
Best word found in list (Subjective)	Pardon set many off
Overall rating	5/10
Comments	The returned list of words aren't particularly strong, and only have a little semantic connection.

Clue: The man composing a song
Test: The man a song
Indicator: composing
Answer: Anthem

3342.6	The man a song new
3303.5	The man a song over
3288.5	The man a song out
3287.5	The man a song some
3228.8	The man a song about
3201.2	The man a song used
3199.6	The man a song around
3165.7	The man a song made
3135.3	The man a song make
3124.9	The man a song built
3124.8	The man a song another
3116.7	The man a song order
3098.8	The man a song changes
3096.5	The man a song off
3096.0	The man a song find
3095.6	The man a song building
3088.0	The man a song away
3086.9	The man a song different
3084.9	The man a song form
3083.2	The man a song change
3071.7	The man a song makes
3071.6	The man a song free

3025.3	**The man a song composed**
3019.0	**The man a song composing**

Best word found by software	The man new a song
Best word found in list (Subjective)	The man made a song
Overall rating	7/10
Comments	Some of the words make sense, with a semantic connection. "Composed" and "Composing" also ranked decently (middle of the pack)

Clue: Inexorable seller sent out
Test: Inexorable seller sent
Indicator: out
Answer: Relentless

1.8333333333333333	Inexorable seller sent letters
1.5	Inexorable seller sent used
1.3333333333333333	Inexorable seller sent about
1.0	**Inexorable seller sent out**
0.6666666666666666	Inexorable seller sent made
0.5	Inexorable seller sent travel
0.5	Inexorable seller sent order
0.5	Inexorable seller sent away
0.3333333333333333	Inexorable seller sent transported
0.3333333333333333	Inexorable seller sent some
0.3333333333333333	Inexorable seller sent run
0.3333333333333333	Inexorable seller sent over
0.3333333333333333	Inexorable seller sent find
0.3333333333333333	Inexorable seller sent built

Best word found by software	Inexorable seller sent letters
Best word found in list (Subjective)	Inexorable seller sent letters
Overall rating	9/10
Comments	The software returned “letters” as the strongest word, which I agree with. It makes sense and has a strong semantic connection.

Clue: Cutter is dreadfully chesty
Test: Cutter is chesty
Indicator: dreadfully
Answer: Scythe

63.5	Cutter is chesty some
48.666666666666664	Cutter is chesty about
47.0	Cutter is chesty new
45.5	Cutter is chesty used
42.166666666666664	Cutter is chesty out
39.833333333333336	Cutter is chesty over
35.166666666666664	Cutter is chesty another
33.833333333333336	Cutter is chesty made
28.166666666666668	Cutter is chesty around
21.333333333333332	Cutter is chesty possible
20.5	Cutter is chesty different
19.0	Cutter is chesty make
19.0	Cutter is chesty built
0.0	**Cutter is chesty dreadfully**

Best word found by software	Cutter is some chesty
Best word found in list (Subjective)	Cutter is built chesty
Overall rating	4/10
Comments	The returned list of words were very weak – they did not make sense and little semantic connection.

Clue: The girl is different having lost weight
Test: The girl is having lost weight
Indicator: different
Answer: Lighter

1050.4285714285713	The girl is having lost weight new
1032.857142857143	The girl is having lost weight over
1032.3809523809523	The girl is having lost weight some
1025.142857142857	The girl is having lost weight out
1004.7142857142857	The girl is having lost weight about
994.7142857142857	The girl is having lost weight used
991.0	The girl is having lost weight around
975.3809523809524	The girl is having lost weight made
962.1428571428571	The girl is having lost weight another
961.3333333333334	The girl is having lost weight make
960.952380952381	The girl is having lost weight built
956.4761904761905	The girl is having lost weight order
947.4285714285714	The girl is having lost weight changes
947.047619047619	The girl is having lost weight building
946.9047619047619	The girl is having lost weight off
945.7619047619048	The girl is having lost weight find
945.0	**The girl is having lost weight different**

Best word found by software	The girl is new having lost weight
Best word found in list (Subjective)	The girl is different having list weight
Overall rating	7/10
Comments	Most returned words make little sense, but “different” did rank decently (middle of the pack)

Clue: Perhaps I reveal a girl’s name
Test: I reveal a girl’s name
Indicator: perhaps
Answer: Valerie

135.8	I reveal a girl’s name out
135.66666666666666	I reveal a girl’s name new
132.33333333333334	I reveal a girl’s name some
128.6	I reveal a girl’s name over
128.6	I reveal a girl’s name about
117.0	I reveal a girl’s name made
115.66666666666667	I reveal a girl’s name used
113.93333333333334	I reveal a girl’s name around
109.33333333333333	I reveal a girl’s name make
106.93333333333334	I reveal a girl’s name another
93.13333333333334	**I reveal a girl’s name perhaps**

Best word found by software	Out I reveal a girl’s name
Best word found in list (Subjective)	Perhaps I reveal a girl’s name
Overall rating	5/10
Comments	Most returned words make little sense. “perhaps” ranked fairly lowly.

Clue: Rodney wanders over there
Test: Rodney over there
Indicator: wanders
Answer: Yonder

27.333333333333332	Rodney over there some
26.0	Rodney over there out
18.833333333333332	Rodney over there about
18.166666666666668	Rodney over there new
16.833333333333332	Rodney over there used
15.666666666666666	Rodney over there over
15.0	Rodney over there another
14.833333333333334	Rodney over there around
13.166666666666666	Rodney over there made
12.5	Rodney over there order
12.333333333333334	Rodney over there off
12.333333333333334	Rodney over there changes
12.166666666666666	Rodney over there make
11.833333333333334	Rodney over there find

7.833333333333333 **Rodney over there wandering**

Best word found by software	Rodney some over there
Best word found in list (Subjective)	Rodney changes over there
Overall rating	5/10
Comments	Most returned words make little sense and have weak semantic connections.

Clue: Darkness is an odd thing
Test: Darkness is an thing
Indicator: odd
Answer: Night

180.4	Darkness is an thing some
173.4	Darkness is an thing about
173.0	Darkness is an thing new
170.4	Darkness is an thing used
168.4	Darkness is an thing out
168.0	Darkness is an thing over
159.9	Darkness is an thing made
159.3	Darkness is an thing another
154.4	Darkness is an thing around
150.5	Darkness is an thing make
149.0	Darkness is an thing different
148.3	Darkness is an thing possible
148.0	Darkness is an thing built
145.7	Darkness is an thing order
144.7	Darkness is an thing form
144.5	Darkness is an thing find
143.9	Darkness is an thing makes
143.9	Darkness is an thing building
143.5	Darkness is an thing off
143.4	Darkness is an thing perhaps
143.2	Darkness is an thing changes
143.2	Darkness is an thing bad
142.9	Darkness is an thing change

133.6 Darkness is an thing odd

Best word found by software	Darkness is an some thing
Best word found in list (Subjective)	Darkness is an new thing
Overall rating	7/10
Comments	“an” should be replaced by “a” in some cases, but this is expected. Most returned words make little sense, but some of them are decent.

Clue: A friend much altered
Test: A friend much
Indicator: altered
Answer: Chum

178.33333333333334	A friend much out
172.16666666666666	A friend much some
170.66666666666666	A friend much over
162.66666666666666	A friend much about
141.66666666666666	A friend much used
136.83333333333334	A friend much made
132.66666666666666	A friend much around
121.66666666666667	A friend much make
116.83333333333333	A friend much another
104.33333333333333	A friend much built
103.0	A friend much find
102.66666666666667	A friend much order
102.66666666666667	A friend much makes

70.33333333333333 **A friend much altered**

Best word found by software	A friend much out
Best word found in list (Subjective)	A friend much used
Overall rating	6/10
Comments	Most returned words make little sense and have little semantic connection.