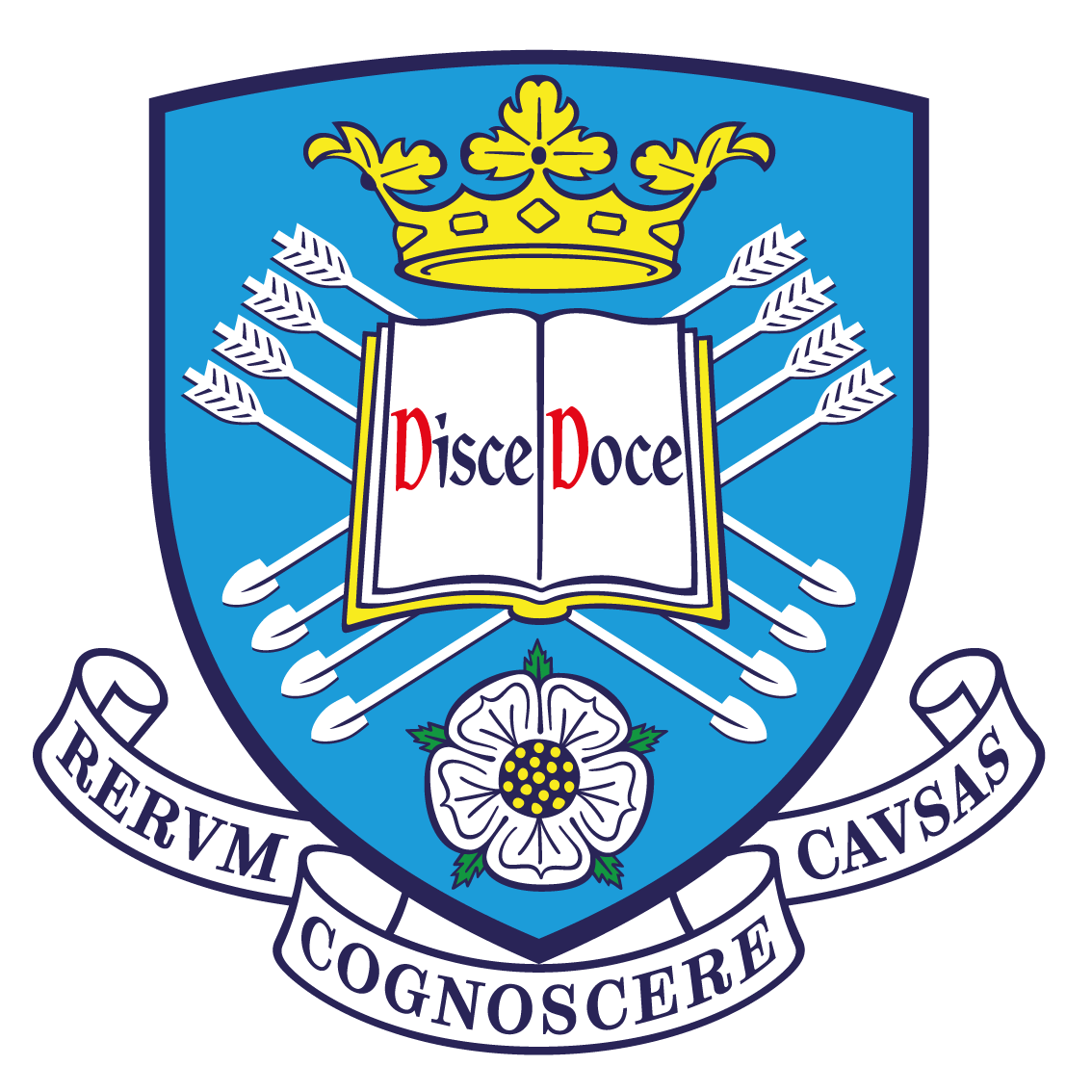
University of Sheffield

COM3610 Dissertation

## Unsupervised Discovery of Word Morphology



Danny Sint

Supervisor: Mark Hepple

This report is submitted in partial fulfilment of the requirement for the degree of

BSc in Computer Science

The Department of Computer Science

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Danny Sint

## Abstract

Words in all languages are made up of morphemes. These morphemes can consist of the affixes or stems of the word that change their meaning depending on which words are put together. For example, the word “unrequited” consists of these morphemes: a negation “un”, a prefix “re”, a stem “quit”, and a suffix to convey past tense in “ed”. In attempting to analyse a language be it a previously undiscovered one or in preparing material to educate learners of foreign languages, it can be useful to break up words into their morphemes.

The aim of this project is to create a software tool that can detect morphemes in words in passages of text and automatically places boundaries between the morphemes.

These kinds of projects have been ongoing since 2001. There have been numerous methods devised since then with the most successful being a software called Morfessor developed by Creutz and Lagus. This project will look to recreate some of these methods and experiment with combining them to produce higher accuracy results.

Implement, experiment with the Keshava method

As an introductory example, the minimum description length first theorised by Goldsmith (2001) is used as a baseline in testing subsequent morpheme boundary creators. As such a mock-up has been started prior to this report by creating the code but currently it only produces the most popular characters.

## COVID-19 Impact Statement

The lockdown imposed because of COVID-19 caused additional challenges for the completion of this project. In the second semester of the project, the university switched to online delivery of all teaching, and university buildings were closed. All project meetings were shifted to email correspondence and video meetings.

## Acknowledgements

I would like to thank my supervisor Professor Mark Hepple for all of his support and guidance throughout the project. His knowledge and experience in this field was critical to the progression of the project.

Contents

[Unsupervised Discovery of Word Morphology i](#_Toc39050555)

[Abstract iii](#_Toc39050556)

[Acknowledgements iv](#_Toc39050557)

[Chapter 1 – Relevant Background Information 1](#_Toc39050558)

[Chapter 2 – Experimentation and analysis 4](#_Toc39050559)

[Chapter 3 – Range of approaches/methods for morphological segmentation. 5](#_Toc39050560)

[Chapter 4 – Evaluation methods 8](#_Toc39050561)

[Chapter 5 – Future plan 9](#_Toc39050562)

[Bibliography 10](#_Toc39050563)

## Chapter 1 – Introduction

Morphology refers to the study of language word structure. Morphemes refer to the smallest parts of words that make up the meaningful words. These morphemes can refer to the stems, prefixes, suffixes or root words. Altering which morphemes attach to each other change the meaning of the word. For example, the stem word might be “watch” and the suffix might change from “ing” to “ed”. These suffix morphemes change the meaning from the present tense - what is currently happening to the past tense - what has previously happened.

Morphemes are units of words that may be conjugated in order to produce a range of different meanings, for example in English the word “unacceptable” has the following morphemes: “un”, “accept” “able”. The “un” denotes a negation statement. The “accept” is a root word that’s a primary lexical unit which carries the most significant aspects of the semantic content. Finally, the “able” is a suffix meaning capable of. Put together, this means the negative of the capability of something that is accept[ed].

An interesting development of how some words can be conjugated are examples like "baked". The root word is "bake" and the past participle is conveyed with "ed". These class of words that end in "e" have been merged together with their tense morpheme in order to smooth out the language to make it sound and appear more natural.

### 1.1 Project Aim ✔

This project aims to use some of the methods that have been outlined in previous academic publications documenting unsupervised morpheme segmentation such as those listed as entries to the morpho challenge project’s competition in order to achieve a good f-score similar to the results of the morpho challenge. Where possible, multiple methods might be used in conjunction with one another to obtain a better score. In addition, some of the methods have historically been very slow to process so it would be interesting to see if the time remains the same as documented or if they are faster now with more modern hardware.

### 1.2 Summary of the project

## Chapter 2 – Literature Review

### 2.1 Morphological Analysis (EMPTY)

### 2.2 Morphology as a linguistic phenomenon and difference between languages

Languages can have different subject verb object (SVO) order. For example, in English “I ate dinner” will be read as (“I [subject] dinner [object] ate [verb]”) in form: Subject Object Verb (SOV). Japanese uses this aforementioned form (私は (I) 晩御飯 (dinner) を食べました (ate)。). Other examples that most English speakers might be more familiar with is Object Subject Verb order “Dinner, I ate” as this is how in the popular film series Star Wars the character Yoda talks.

Additionally, some different dialects may come up for different types of speech. However, as this project is focusing on research papers and other kinds of text different dialects will be kept to a minimum and considered a negligible factor.

Different types of language can be divided by the morpheme per word ratio and how the fusion of morphemes occurs. Languages can be defined as more isolated the higher the morphemes per word ratio was. For example, English with words like “unacceptable” (morphemes: “un”, “accept”, “able”) has a >1 (3:1) morpheme ratio.

Fusional or inflected languages are built from morphemes but instead of using the morphemes to alter their meaning the morpheme itself can sometimes changes. For example, the past tense of “eat” changes to “ate” or “sang” as the past tense of “sing”.

Modern Hebrew is an example of a fusional language that uses different particles to denote additional meaning for words for example for verbs adding a “lamed” (ל) is a prefix that changes the verb from “verb” to “to verb” (future tense).

Agglutinative languages are made from morphemes but with an interesting effect. The morphemes are unaltered in order to fit in with other morphemes that makes up its full word.

Some languages that are agglutinative have morphemes that merge into one another causing some letters/sounds to be suppressed from the ending and beginning of the morphemes.

For example, in Japanese the character representing 8 (八 or ha-chi) might have an altered pronunciation depending on what words you’re using with it e.g. (八百 haa-pyaku meaning “8,000”) or (八つya-tsu meaning “8 [things]”) but when written, does not change its character order making morphemes less difficult to identify.

Languages that are truly isolated have a 1:1 ratio between words and morphemes with no alteration of words based on case, tense gender, etc. For languages that uses many different symbols for all the words in the language such as Mandarin, this would be a likely candidate. However, while the individual morphemes are easy to identify, the morpheme per word ratio is not 1:1 as individual words (ideas) can be built from multiple symbols. A better example of an isolating language is the Yoruba language spoken in West Africa where each morpheme is a word. For example, “n̄ ò lọ” (I didn't go) each morpheme refers to different words.

### 2.3 Task of morphological analysis and why interest in it

#### 2.3.1 The Morpho Challenge Project ✔

Taken directly from the Morpho Challenge’s website: <http://morpho.aalto.fi/events/morphochallenge/>

The objective of the Morpho Challenge is to design a statistical machine learning algorithm that discovers which morphemes (smallest individually meaningful units of language) words consist of. Ideally, these are basic vocabulary units suitable for different tasks, such as text understanding, machine translation, information retrieval, and statistical language modelling.

While their scientific goals for unsupervised segmentation is:

* To learn of the phenomena underlying word construction in natural languages
* To discover approaches suitable for a wide range of languages
* To advance machine learning methodology

The morpho challenge competitions ran from 2005 till 2010 and provided a wide range of resources of data from the word lists used as input, the gold standards used to evaluate the candidate segmentations the various programs submitted and the evaluation scripts that were used to measure each program’s efficacy to determine the winner of the competition.

### 2.4 Task of performing morphological analysis (EMPTY)

### 2.5 Unsupervised and supervised approaches

For the task of analysing a previously undiscovered or little studied language an aspiring researcher might consult experts in the language if possible such as a native or fellow analysers of the language.

However, this can be expensive as such experts might rare and a considerable amount of time would need to be used to devise the rules for understanding their language. Some languages might even have fellow natives disagree on some things. Experts on languages might not even exist in the worst case scenarios. Thus it is

(annotated, hand-coded rules)

Rule-based approaches

Rather than segment just use part of speech, this might convey meaning as well.

### 2.6 Range of approaches ✔

Goldsmith (2001) describes four categories of approaches to morpheme induction and divides them into four categories.

1. Identification of morpheme boundaries using transitional probabilities.
2. Identification of morpheme internal bigrams or trigrams.
3. Discovery of relationships between pairs of words.
4. Information theoretic approach to minimise number of letters in morphemes of languages.

The following methods make use some of these ideas in order to analyse and segment morphemes.

#### **2.6.1 Minimum Description Length**

Minimum Description Length is a method of morpheme segmentation that is at the origin of the idea of computational morpheme segmentation (Goldsmith, 2001, pp. 153-198). This method describes splitting each word at a point based on a probability to get stems and affixes. Then it classifies these words based on their similarity to the generated shared morphemes. This is the baseline used by the previous competitions hosted by the Morpho Challenge Project in comparison with other algorithms when experimenting.

Minimum description length is a method that simply cuts each word at a point based on a probability, creates a list of these cuts of words and counts the frequency. This is the baseline method used to compare other methods.

#### 2.6.2 Orthographic and Semantic similarity

From Baroni, Miatiasek, and Trost (2002)’s comes forth the academic idea of a method solving the problem of segmenting words using orthographic and semantic similarity of words.

String Similarity is a technique of measuring the orthographic similarity of words by quantifying the edit distance. Edit distance is the quantification of the number of edits needed to be done to a word in order to reach another word.

In some of these methods the edit distance is a useful statistic in determining information that aids detection of morphemes within words. Thus, there is a necessity to explore the range of edit distance measurers that are available.

In order to judge string similarity one such method is edit distance, there are many types of edit distance quantification such as Levenshtein distance [Levenshtein, 1966], longest common subsequence, hamming distance, Damerau-Levenshtein distance, and Jaro distance. These methods incorporate the different types of string manipulation to discover the amount of alterations of a string to match the other string it’s being compared to. As such these string manipulations can consist of deletion, insertion, substitution, and transposition of characters or variations of these string manipulation techniques. [Gonzalo, 2001]

|  |  |  |  |
| --- | --- | --- | --- |
| String Manipulation | Word 1 | Word 2 | Edit Quantity |
| Substitution | Rake | Bake | 1 |
| Addition | Bake | Baked | 1 |
| Deletion | Rooted | Root | 2 |
| Transposition | Nile | Line | 1 |

Additional thoughts about these methods:

It might be interesting to note that none of these methods have gone on to use weighting for their chosen methods (insertion or substitution) or weight based off the characters. For example, edits of a ‘d’ changing to a ‘t’ might be weighted differently from a ‘d’ changing to ‘e’ as the likelihoods of specific character manipulations are more likely than other character manipulations in terms of string similarity.

It might be interesting to see if experimenting on these methods with an annotated dataset with supervised learning would determine if the weightings are better or worse than these current methods provide statistically significant and useful information.

However, for this task the edit distance with the most functionality is the Damerau-Levenshtein distance as it allows for all methods of string manipulations (deletion, insertion, substitution, and transposition), however for this method it has been determined this method is not useful as

transposition is used mainly for spelling errors and is it unlikely that morphemes are transposable since morphemes are usually pieced together sometimes with conjugation by which letters are removed from the morphemes in order to be pronounceable or sound smoother. It is important to note that it is additionally unlikely that transposition will aid in determining morphemes as nearly all morphemes don’t overlap with consecutive morphemes. However, there are some exceptions such as “baked” to “bake” + “ed” or “sing” to “sang”. It is unlikely that transposition would aid these kinds of morphemes. These kinds of outliers are a difficulty in solving this problem for English. Finally, for this project it is unlikely that the corpus has spelling errors given that they have been edited prior to being published. In this case the Levenshtein distance that does not use transposition will suffice.

Semantic similarity refers to the methods employed that can be used to determine how similar in meaning different words are. Examples of such methods are pointwise mutual information and word embedding.

Pointwise Mutual Information is the idea that words that appear more frequently together suggests that they are related in meaning. In a given corpus, words that are semantically related appear near each other. For example, in a descriptive sentence, colours might be seen close as in “*the building was grey, but the wisterias were violet*”. As these words are more frequently seen together than apart over a large collection of documents, they will build up a co-occurrence. However, this technique is more useful for words that are directly together such as country names “Puerto” “Rica” or “Hong” “Kong”. The further the words are apart; the more computation power and corpus data is required to get a score on them meaning that for some language sources that do not have a high quantity of sources it might be difficult to use this technique.

Word embedding are an assortment of other technique that can be used to determine if words are semantically related. Word embedding techniques vectorise a word with its full meaning converting it to a vector of reals. Similarly related words will have a low distance from one another suggesting they is a higher chance they are semantically related. However, this method does employ the usage of neural networks and its level of complexity may restrict its usage. For a project of this size the complexity is beyond the scope. The similarity of vectors should be measured by the cosine angle as the cosine similarity is unaffected by magnitude (Bengio, Schwenk, Senécal, Morin, & Gauvain, 2006).

Orthographically related words are likely to be morphologically related as well. These related words can be calculated via the semantic similarity methods mentioned earlier.

The following are methods that employ the previously mentioned strategies.

#### 2.6.3 Morfessor

From the conference Unsupervised Morpheme Analysis - Morpho Challenge 2010 the highest rated F-measure scoring for the widest variety of languages was the Morfessor project while the highest Unsupervised learning was Base Inference by Lignos.

Morfessor is a program that in the 2010 conference achieved the best result with semi-supervised learning. It uses probabilistic maximum a posteriori framework to calculate morphemes. A list of morphemes is created from the data. Probabilistic maximum a posteriori framework (Creutz & Lagus, Unsupervised Models for Morpheme Segmentation and Morphology Learning, 2007)

Prior to this, the same authors worked on the same morpheme segmentation but with a different method called model costing and recursive segmentation.

#### 2.6.4 Model costing

Model costing created costings of the source text and the codebook (morph types). This algorithm worked by attempting to reduce the costs of these two sources by segmenting morphemes and selecting the minimum cost each time.

Once the morpheme list has been finalised recursive segmentation goes through each word in the text and checks if the word is a morpheme, then every split of the word is checked to be a morpheme and if so the original word is removed and replaced by the morphemes. Then these two parts are recursively checked again until no more morphemes have been found.

#### 2.6.5 Letter successor variety

Letter successor variety measures the amount of letters before or after a part of the word and compares it with the amount after or before respectively. At any rapid peaks of this comparison there is an increased chance of morpheme boundary occurring. However, since there are many different words with differing lengths the level of noise disrupts this kind of strategy. As expected, this method is regarded as having one of the lowest success rates for solving the original problem. (Bordag, 2005)

As a note: a small caveat is that it is not always clear programmatically where the word should be split. For example, “hoped” can be split into “hope” + “d” or hop” + “ed”.

#### 2.6.6 Base Inference

Base Inference is the method devised by the author that achieved the highest scoring in the final conference. Via creation of ruleset that the language follows it attempts to relate word pairings with the base (stem) and account for the other kind of words that are related to the base word. From there it calculates the differences using transforms as the measurer between the base word and its morpheme differentiated counterparts.

This does not currently account for fusional words that change (eat – ate)however, because there would generally be rules created that are similar to the words that have affixes but not stems there would be information to separate them regardless. As an example: the word “fatefully” has been found in the text but neither “fate” nor “fully” has been observed in the text. But, “lawfully”, “joyfully”, “artfully” have been observed and thus the morphemes “law”, “joy”, “art”, and “fully” can be separated as morphemes. This would mean “lawfully” can have “fully” removed from it meaning “law” must be a morpheme too. Where we encounter “fully” from now on for any and all words containing fully it can be cut out. (Lignos & Beck, 2011)

Similarly, there’s another method by Baroni, Matiasek and Trost that deals with creating pairs of words. More specifically for this method the words that appear 0.01% of the time are used as a wordlist and compared to the words in the input text. The nearest (in edit distance) that matches with a word in the 0.01% list are paired together. This method uses the string edit distance and the pointwise mutual information outline in the previous methods as techniques to determine morphemes. (Baroni, Matiasek, & Trost, 2002)

2.6.7 Keshava / Pitler

### 2.7 Evaluation ✔

For any software that performs a task with multiple methods it is useful to determine which is the best tool for the job. This process is the evaluation stage.

For natural language processing (NLP) tasks, a common method of evaluating is by measuring the precision and recall of the results against the gold standard document culminating in the harmonic mean otherwise known as the F-measure.

The recall denotes the amount of information that is obtained by the program while the precision is how relevant the information obtained is. The f-measure provides a harmonic mean of these two to obtain an average that punishes the final result (f-measure) if either of the two scores is greatly higher than the other.

The gold standard is a document that denotes what the program undergoing testing should be producing. The specific way in which the methods are evaluated is important for scoring how well the method have succeeded or in the case of multiple methods to compare which has achieved the best result. As the competitions at the Morpho Challenge resulted in a coalescence of methods it is logical to look there for the results and evaluation method.

The gold standard contains the following useful information: the word and its morphemes. This can be in any form such as “word morpheme1 morpheme2” or “morpheme1 morpheme2... etc” with a delimiter between morphemes in all cases. The gold standard available from the morpho challenge does have this information but

There have been competitions at conferences to decide which tool has the highest F-measure. These have taken place in Aalto University School of Science and Technology in Espoo, Finland, from 2005 to 2010 which is where this project is basing most of its data and gold standards from.

For the desired comparison from the morpho project they have provided an evaluation script in Perl. Unfortunately, there are some problems getting this to work. As this is from 2005 the binary that allows for this is 15 years out of date, this means there is a security risk, a compatibility issue, a depreciation issue and a usage issue. The issue with using the current version of Perl is that aspects of the Perl script are using depreciated methods and also requires a Unix shell to input data as the “>” is not allowed on the Windows command line. Since previous Perl installation binaries for Linux are not available without a license that costs £84 with no information about which Perl versions were available, no confirmation about whether they would be the right versions with the ability to run this Perl script without any side effects, inaccuracies or any other issues this became a path of many obstacles. While this Perl script could have been rewritten in python, a faster and better solution was sought and obtained.

#### 2.7.1 Pyports

Pyports is a program that has an evaluator for the task of comparing a gold standard and a diagnostic standard. It uses concepts provided by the Morpho Challenge to evaluate the output produced by the program that needs measuring and output the precision, recall, and f-score of the result compared to a gold standard that is fed in. As it was written in python it could be adapted to work within a program seamlessly without needing to run a separate file for evaluation.

The evaluation works by getting a large sample of word pairs from the gold standard and the diagnostic standard such that both words in the pair have at least one morpheme in common. A number of word pairs are obtained where at least one morpheme is shared by both word pairs. These pairs are compared to the gold standard. Points are given for word pairs that have a morpheme in common but taken away for morphemes that are not in common. The total of these scores is divided by the number of word pairs. An example taken from the morpho website for the 2005 evaluation is displayed.

“*For instance, assume that the proposed analysis of the English word "abyss" is: "abys +s". Two word pairs are formed: Say that "abyss" happens to share the morpheme "abys" with the word "abysses"; we thus obtain the word pair "abyss - abysses". Also assume that "abyss" shares the morpheme "+s" with the word "mountains"; this produces the pair "abyss - mountains". Now, according to the gold standard the correct analyses of these words are: "abyss\_N", "abyss\_N +PL", "mountain\_N +PL", respectively. The pair "abyss - abysses" is correct (common morpheme: "abyss\_N"), but the pair "abyss - mountain" is incorrect (no morpheme in common). Precision here is thus 1/2 = 50%.”*

There are multiple wordlists available from the Morpho Challenge’s website. Some of them such as the 2005 English wordlist require some editing as the character “à” causes an unintended split. As there are only 6 instances of words containing this letter in the English text it would be faster to simply remove these entries.

As an aside, a potential problem was foreseen and considered. Punctuation makes morphological analysis more difficult as it splits up two words into a single e.g. “don’t” really means “do not” they would ideally need to be split up into their proper words. While some of the more simple examples could be fixed, there could be issues with a solution such as a dictionary of words or a library such as pycontractions (Beaver, n.d.) containing apostrophes mapped to their that might interfere with the final results as there are nearly 87,000 apostrophes in one of the wordlist files provided by the morpho challenge. However, as acknowledged, the words that have been combined with an apostrophe would most likely be in the wordlist as well so the apostrophe’s have been opted to be ignored as they shouldn’t provide too much benefit unless there are multiple examples of apostrophe’s within the gold standard.

Similarly words with dashes or any other kind of punctuation within the word are ignored as they most likely are morphemes delimited by dashes.

Additionally, the pyports program came with a gold standard and datasets containing word lists and gold standards of its own for English, Russian and Japanese.

**TALK ABOUT THE MAIN IDEA OF PYPORTS AND HOW IT WORKS**

### 2.8 how well it performed / how the evaluation works? against gold standard (EMPTY)

### 2.9 datasets / challenges (EMPTY)

Write about pyports and morpho challenge data. Format of data, what segmentations it gives

### 2.10 give examples to pad space (EMPTY)

How words might be segmented.

“this approach will be used in the project work” – justify more time is going to be spent on it later

## Chapter 3 – Planned Experiment and Analysis (EMPTY)

### 3.1 Removing punctuation

thresholds modifying

modifying method

talk about the three conditions

Remove the first condition check if it's another word see if removing this helps/abets the score

Add Japanese/Russian experiments

words like "bake" wouldn't occur in the word existence check for words such as "baking" as "bak" doesn't exist as a real world.

experiment, the results and then which gives better results and then continue with that

### 3.1 Trie Structure and Nodes

### 3.2 Inputs and Outputs

### 3.3 Scoring affixes

### 3.4 Segmenting affixes

### 3.4.1 Multiple affix peeling

### 3.5 Evaluation

## Chapter 4 – Implementation / Testing

Pitler and Samarth’s “A segmentation approach to morpheme analysis” presents a method of morpheme segmentation that works on concatenative morphemes. It uses the approaches of discovered words that are substrings of other words (e.g. phone -> smartphone) and detecting changes in transitional probabilities (e.g. Detecting a “d” is more likely to appear after an “e” as opposed to an “x”.

Keshava / Samarth method background

### 4.1 Getting Morpho Challenge Data (Unfinished)

There are multiple wordlists and gold standards available on the Morph Challenge’s website ranging from 2005 to 2010. The 2010 version has the most amount of information, so it makes sense to use that version.

However, the gold standard file from pyports has some of the most common words that are used.

### 4.2 Trie Structure and Nodes

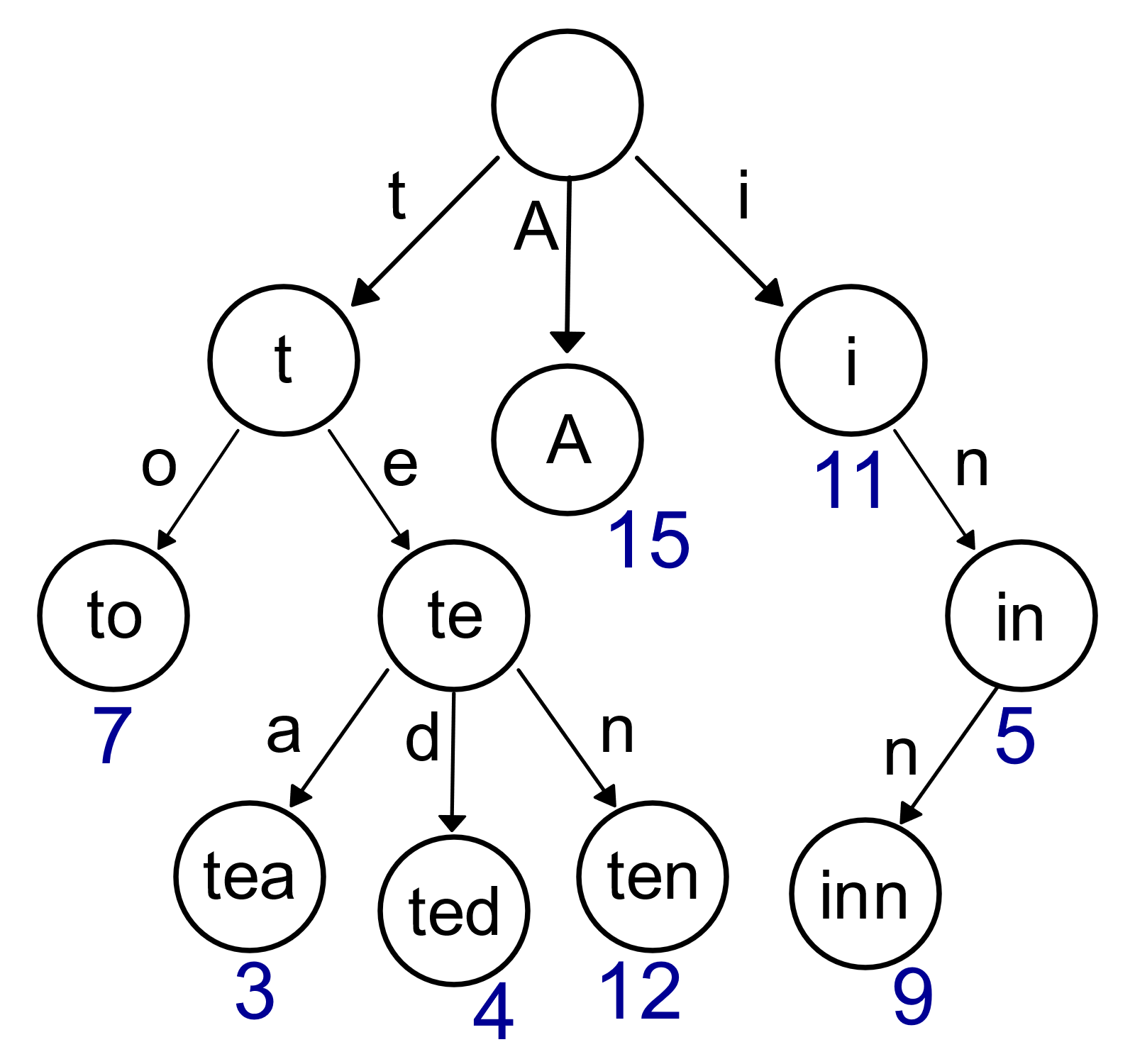
The first part of the method by Pitler and Keshava outlines the use of forward and backwards Tries. 

Figure 1 – CREDIT AUTHOR

As shown in Figure 1 an example of a Trie is shown. It is similar to a binary tree but instead of only 2 possible nodes there can be as many as wished for. Each possible node contains a single character and the path from the root to a leaf (ignoring the character in the root) evaluates to a full word.

Tries have a real life applicable usage where they are most useful in predicting words. For example, all the possible words that can be created given that the input starts with x are the nodes that descend from the current position of the node. In figure 1 given that input “te” is input there exists only 3 possibilities, an “a”, a “d” or an “n”.

Using these tries it would be possible to determine the probability of how many other words share the same starting letters as a word currently trying to determine the morphemes of. It is highly likely the peak corresponds to morpheme boundaries.

A normal equation for this calculates the probability of a letter given the current word. This is a usage of Bayes theorem where the probability is:

P (A|B) = ( P(B|A) ・P(A) ) / P(B)

Going through this with an example is the word “reports”. The morphemes for “reports” are “re”, “port” and “s”. In order to devise this the probability of (s|report) is needed to be obtained. Bayes theorem provides the answer to be the probability of “reports” divided by the probability of “report”. The probabilities will be an O(1) lookup of the forward and backwards tries.

#### 4.2.1 Forwards and Backwards Tries

Each node contains the value containing how many words start or end (depending on the type of trie it is) with the sequence of letters up to this node from the root. As an example, this means that the node for “a” that comes directly from the root node is the number for all the words in the lexicon that start with “a”.

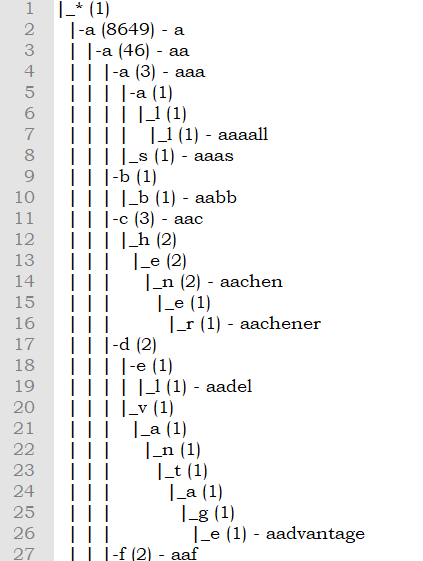


Figure 2 Example of forwards Trie

The backwards trie is a reverse kind of Trie. Instead of starting with the highest node being the first letter, the highest node is instead the last letter of the word. This means that the probability of a word ending in x can be compared to calculate the transitional probability. An example of a backwards Trie would

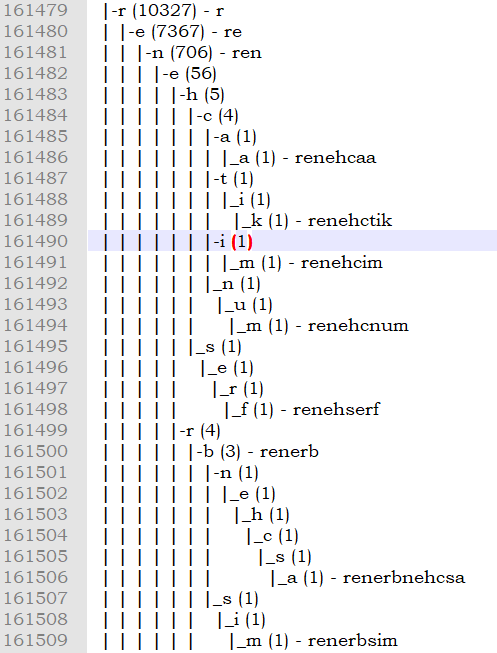


Figure 3 Backwards Trie example

All the words are in reversed order. For example, “renehserf” is “freshener”. This means that the probability of the end of a word can be calculated.

#### 4.2.2 Adding to tries

This algorithm starts at the root node and takes in a word and loops over the characters in it. For each character it checks whether that character exists within the descendants of the current node.

If it does it just increments the count of the node and sets the child node containing the character to the next node for the next loop.

If it does not find a descendant node with the character currently being looped over then it will create that node for the character with an initialised value of 1.

#### 4.2.3 Get the Trie node’s value (find\_prefix)

To get the value contained at each node (the amount of words that are descended from that node) a traversal algorithm is needed. This is the crux probability function mentioned in the Pitler & Samarth method.

This algorithm loops through the children nodes to find the next character then replaces the current node with the new node if found and a 0 with a flag if it hasn’t been found. The algorithm does not need to maintain the path in memory as it is only looking for the final node that contains the number of all words that descend from it.

### 4.3 Inputs and Outputs

#### 4.3.1 Inputs

The inputs are configured by the command line arguments. The input file by -i, the output file by -o and the gold standard by -g. The wordlist uptakes the words per line. If the frequency is available it stores that too, if the frequency is not present and there are only words per line it’ll just set the frequency to 0. If the frequency is a factor, for example experimentation on which empirical numbers work best then the program can opt to ignore any words that have lower than a specific value.

The gold standard is just used to compare with the output file. The program goes through all gold standard words, removes the segmentation marker and checks to see if the word is in the output. If it is, it starts evaluating the two words. If not, it goes to the next one.

#### 4.3.2 Outputs

The program then outputs the standard to be compared against the gold standard. The format matches the gold standard format in that each line contains a word separated by the segmentation marker.

It does this to prepare for evaluation and remain modular so that the evaluation code can be picked up and placed in another file and run on a file with the required format.

### 4.4 Scoring affixes

In order to devise these programmatically it is necessary to go through each word in the wordlist.

The function does a brute force method by going through each potential split in the word to split it into 2 parts to test whether the three conditions outlined in Pitler & Samarth’s method are true.

For prefixes it works with the backwards trie and tests second part of the word to score the first part while with suffixes the forward trie is used to test the first part of the word to score the second.

For example: the word “reports” has the suffix of “s”. Thus, the forward trie will be used as outlined previously to check the probabilities. It will iterate across the entire word from start to finish splitting up each point so that each consecutive part of the word has an opportunity to be tested. The function call looks like this:

|  |  |
| --- | --- |
| First word part | Second word part |
| r | eports |
| re | ports |
| rep | orts |
| repo | rts |
| repor | ts |
| report | s |

The first criterion addresses the observation that root words are attached by prefixes and suffixes.

The first criterion is formalised by checking that the word part being tested (for suffixes it’s the first part and vice versa) exists within the dictionary. As a aside it is important that the words are in hashable format that enables O(1) constant time lookup otherwise each word part will run a O(n) operation putting the morpheme analysis at a O(n2) runtime.

To implement this, two variables have been used, one to hash the words and another to iterate through in dictionary order.

The second and third condition check the root has more than one potential children implying that other affixes may be attached to the root but that the root’s parent node only has a single node identifying it as a true root.

They are formalised by checking the probability of the tested part of the word that has the last letter cut off is divided by the probability of the tested part of the word is near 1.

Finally, the third condition checks the tested part’s probability divided by tested part with an additional appended letter is less than 1.

Each word part that is put through the function that scores potential morphemes is added to a wordscore dictionary. This keeps track of how many times

The other part of the word is tracked in a potential morpheme dictionary. If all three conditions are passed. The word part has its score raised by 19. If it fails any test it’s score is lowered by 1. This is an arbitrary value to punish words that are deemed not probable enough to be **the** morpheme for that particular word. The conditions operate on a 5% acceptance rate in that they need to pass the tests at least 5% of the time in order to be considered to be a morpheme. Words can contain the same parts and are judged many times. For example, the word “mountains” and “rivers” both would at some point test “mountain” & “s” and “river” & “s” thus rendering judgement onto both “mountain” and “river” but “s” twice. Extend this idea across all words and the word part “s” will be judged very frequently to ensure it’s a morpheme.

This means word fragments such as “en” are heavily punished for being in words like “heaven” or “kindergarten” or similar words.

Being the most likely morpheme for that word is also important. There might be a high probability of a “d” being the suffix in words like “rooted”, “baked”, or “loved” but the actual morpheme here is “ed”. The conditions prevent affixes like “d” from rising to the top together with “ed” while they reward morphemes like “~s” since these are far more likely to appear together than the fragment “d” by itself.

The same idea applies the prefixes but instead using the backwards trie the conditions. For the first condition the second part is checked whether it exists within the dictionary. The second condition checks the probability of the reverse of the second part divided by the probability of the reverse of the second part with the last letter removed. Finally, the third condition checks the probability of the reverse of the second word part divided by the probability of the second word part with an added letter.

#### 4.4.1 Pruning

All entries in the word scoring dictionary with a negative score are removed from consideration. At the end of this sequence.

### 4.5 Segmenting affixes

Segmentation of affixes works with the same concept of transitional probabilities but with peeling off affixes from the root leaving the root itself untouched. It works with the score

### 4.5.1 Multiple affix peeling

### 4.6 Evaluation

## Chapter 5 – Experiment Results

Chapter 2 – Literature Survey

Do you need to include in this bit the connection with computing?

Objective (?)

This project will additionally require evaluating the success of the results produced by the code compared to a gold standard example. The contents of the gold standard will contain the original word as well as the segmentations that are to be expected. The Morpho Challenge Project provides an English gold standard wordlist with gold standard segmentations that will be used for this project.

The objective of this project is to identify and segment these morphemes in a given corpus without any prior training. As this does not use any annotated input or specifics knowledge of any language it could theoretically work on every language and not only be limited to English*.* However, in practise only English will be used because testing another language might be too wide in scope and ambitious.

With a tool capable of analysing words and deconstructing the morphemes that incorporate these words without any annotated information available this may have useful application for under resourced languages as a first step towards analysis by creating a list of morphemes that can be furthered clustered into their respective grammatical groups. For example, verbs would be grouped together.

This would reduce the need for experts in the language, reduce the time needed to analyse a language and additionally open doors that could not otherwise be opened since some languages might not have willing or existence experts available.

Additionally, words that have not been encountered before can be broken down into their morphemes and eventually into their stems. If meaning is attached to these morphemes (for example “in”, “un”, “de” denoting negation) and these morphemes grouped into classes then it can reduce the workload required to analyse a language. Furthermore, the word can be chopped into its stem to gather the actual semantic value of the word even if it’s never been seen before. For example “factoid” may not have always been a word but if people create and start using it as a natural recourse to explain a small fact then a morphological analyser will be able to discern the two morphemes “fact” and “oid” to determine that certain people are using this word to mean “fact” despite never having come across it previously.

In addition, if the software this project produces builds up lists of words encountered from each subsequent text entry it might be able to have a wider range of morphemes to use for unencountered words. However, this is only additional projection of usage beyond this current project.

## Chapter 2 – Experimentation and analysis

As this project will be written in python (much better when like here, you mention something specific) the Natural Language Toolkit (NLTK) might be useful for working with language data. The usage of stemming, tokenisation and semantic reasoning available via the NLTK would make some of the methods far quicker to implement worth explaining this a bit more.

# Bibliography

Baroni, M., Matiasek, J., & Trost, H. (2002). Unsupervised discovery of morphologically related words based on. *Morphological and Phonological Learning - Association for Computational Linguistics*, 48-57.

Bengio, Y., Schwenk, H., Senécal, J.-S., Morin, F., & Gauvain, J.-L. (2006). A Neural Probabilistic Language Model. Studies in Fuzziness and Soft Computing. 137-186.

Bird, S. E. (2009). *Natural Language Processing with Python.* O'Reilly Media Inc.

Bordag, S. (2005). Unsupervised Knowledge-Free Morpheme Boundary Detection.

Creutz, M., & Lagus, K. (2007). Unsupervised Models for Morpheme Segmentation and Morphology Learning. *ACM Transactions on Speech and Language Processing*.

Creutz, M., Stig-Arne Grönroos, Oskar Kohonen, Mikko Kurimo, Krista Lagus, Peter Smit, & Sami Virpioja. (2015, July 14). *Morpho Project*. Retrieved from Morpho Project: http://morpho.aalto.fi/projects/morpho/index.html

Goldsmith, J. (2001). *Unsupervised Learning of the Morphology of a Natural Language.* Chicago: Computational Linguistics archive.

Gonzalo, N. (2001). A guided tour to approximate string matching. *ACM Computing Surveys*, 31-88.

Levenshtein, V. I. (1966). *Binary codes capable of correcting deletions, insertions, and reversals.* Soviet Physics Doklady.

Lignos, C., & Beck, J. (2011). The$Power$of$Objects$in$Morphology$. *LSA Annual Meeting.* Pennsylvania: University of Pennsylvania.

Edward Sapir (1884–1939).  Language: An Introduction to the Study of Speech.  1921.

<https://www.omniglot.com/writing/yoruba.htm>