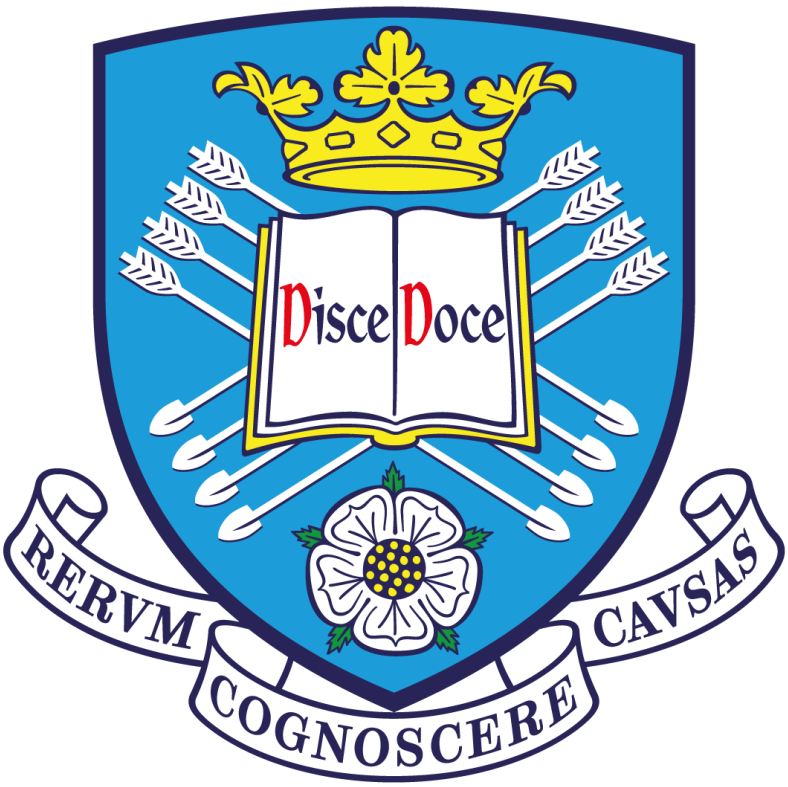
University of Sheffield

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

## Signed Declaration

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

Date: 13th May 2020

## Abstract

Words in all languages are made up of morphemes. These morphemes can consist of the affixes or stems of the word that changes their meaning depending on which morphemes are joined together. For example, the word “unrequited” consists of three of these morphemes: a prefix negation “un”, a prefix “re” denoting the idea of ‘returning’ or ‘again’, the 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 preparation for materials to educate learners of second languages, it may prove useful to break words up 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 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 precision and more accurate results.

## 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 his support and guidance throughout the project. His knowledge and experience in this field was critical to the progression of the project.

**Contents**

# Chapter 1 – Introduction

Morphology refers to the study of language's word structures. 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 in how some words can be conjugated are examples like "baked". The root word is "bake" and the past participle is conveyed with "ed". This 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. Therefore, "bake" becomes "baked" rather than "bakeed".

## 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 (splitting words into their morpheme parts) such as those listed as entries to the morpho challenge project's competition which was an annual competition between the years of 2005-2010 held in Finland to determine which program worked best at segmenting morphemes. This is in order to achieve a good 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

This project contains six chapters that convey the research of various literature related to the project aim, the methods available to tackle the problem, experiments to be conducted to increase efficacy, the implementation and testing of the chosen methods, the results of the experiments and finally future ideas that might benefit this research.

Chapter 2 explains the problem of morpheme induction into segmentation and explains the methods available within the various literature designed to solve this problem. It talks about how other approaches to the problem were the project using supervised learning instead of unsupervised learning and how this can help. The evaluation metrics are explained to score any methods of solving the morpheme segmentation against other methods and for an understanding of how complete and comprehensive the given solution to the problem is.

Chapter 3 describes the designs of experiments meant to make tweaks to the method chosen, in order to ascertain whether the tweaks aid or hinders the solution.

Chapter 4 explains the implementation and testing of how the data from the dataset is obtained and fed into the program, the data structures used in the outlined method, the scoring of morphemes, the segmentation of words based on those morpheme scores and finally the evaluation to score which method and tweaks works better.

Chapter 5 displays the results of each of the experiments outlined in chapter 3 while chapter 6 analyses and concludes the findings of the project, confirms whether the project aim was achieved and discusses future ideas that may improve the project further.

# Chapter 2 – Literature Review

## 2.1 Morphological Analysis

Morphology is the study of words and their formations along with relating words to other words. Morphology aims to study parts of words that act as the structure with each part carrying semantic information. These are known as morphemes and while they contain meaning, not all of them are standalone words. They might convey past tense for verbs e.g. “ed” or a morpheme denoting plurality e.g. “~**s**”. Words might even be joined to convey semantic reasoning for new words to be added to the dictionary. For example, in recent years “smart” has been added to “phone” to mean a multi-purpose phone that can do multiple tasks such as connect to the internet. Previously this word did not exist.

At a more basic level for the parts of words, roots are lemmatised word forms and as such, are the most basic units of words - roots lie at the centre of words and can be derived by removing all affixes and reducing the stem to its root via lemmatisation - whereas stems change inflection based on the additional information conveyed such as tense. Prefixes are morphemes that can be prepended to a root word that change the word’s meaning while suffixes are morphemes that can be appended to the word and similarly alter the word’s meaning.

Inflection are changes in the form of words to show a grammatical attribute such as tense, mood, person, number, case, or gender.

These rules might be different in different languages and as such it can be useful to find the differences between languages.

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

There are many types of differences between languages. Some languages 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 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 arise 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, in English the word “unacceptable” (morphemes: “un”, “accept”, “able”) has a >1 (3:1) morpheme ratio thus making it a fusional language.

Fusional or inflected languages are built from morphemes but instead of using the morphemes to alter their meaning, the morpheme itself can sometimes change. For example, the past tense of “eat” changes to “ate” or “sang” is 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 are agglutinative which means they 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 use many different symbols for all the words in the language such as Mandarin, this suggests its morpheme to word ratio is 1:1. 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, in the sentence "ṅ ò lọ" (I didn't go) each morpheme refers to different words.

### 2.3 Task of morphological analysis and its potential

The objective of this project is to identify and segment morphemes in words for a given corpus without any prior training. As this identification and segmentation process does not use any annotated input or specific knowledge of any language the kind of solutions created for this problem could theoretically work on every language and not only be limited to English. However, this is limited by practicality as some of the algorithms or methods used for the task might be more effective on specific types of language. For this project English will be the main use case because testing another language is too wide in scope and ambition and the author only has fluency in English with basic knowledge of Hebrew and Japanese.

There may be further potential for a tool capable of analysing words and deconstructing the morphemes and which incorporate these words without any annotated information available. It may have a useful application for under-resourced languages as a first step of analysis by creating a list of morphemes that can be further clustered into their respective grammatical groups. For example, verbs would be grouped together. In the case of English: tenses, ("ing", "ed") genders, ("his", "her") plurality, ("~s) or singularity (~) and more could also be grouped together. This would reduce the need for experts in the language, reduce the amount of time needed to analyse a language and additionally create opportunities for researchers that could not otherwise be opened since some languages might not even have willing experts or such experts might not even exist.

Unencountered words can be broken down into their morphemes that will usually contain the root. If meaning is attached to these morphemes (for example "in", "un", "de" denoting negation) and the morphemes grouped into classes, it can reduce the workload required to analyse a language. Furthermore, the word can be reduced to its root 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 it as such 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, a root and a suffix ("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 the suggested additional projection of usage beyond this current project.

This leads us to the Morpho Challenge which is the first step towards understanding the problem and the solutions there are thus far.

#### 2.3.1 The Morpho Challenge Project

The Morpho Challenge Competition is a website hosted by the Aalto University in Espoo, Finland.

*"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."* from the Morpho Challenge's website: (Mathias Creutz, et al., 2010)

Their scientific goals for unsupervised segmentation are:

- 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 providing a wide range of data resources from the word lists used as input, to the gold standards used to evaluate the candidate proposed segmentations of the various programs submitted, to the evaluation scripts used to measure each program's efficacy to determine the winner of the competition.

### 2.4 Task of performing morphological analysis

For this project Python 3.7.2 has been chosen. Python has an extensive number of libraries that could prove useful for this particular task whether they be utility features or scientific libraries and methods. It's also an interpreted language with less strict syntax, dynamic variables, not requiring compilation, and weak typing that could reduce the amount of overhead work required for fast iterations of testing, debugging, and experimentation.

Thereby the task of morphological analysis and segmentation denotes reading literature on how to solve the problem of analysing morphemes to further on segment the morphemes in words. It requires an understanding of what unsupervised learning is and how it differs from supervised.

### 2.5 Unsupervised vs supervised approaches

"Unsupervised segmentations suffer from problems such as oversegmentation of roots and erroneous segmentation of affixes." (Kilic & Bozsahin, 2012)

Unsupervised segmentation methods suffer from creating certain rules that do not actually exist within the language they are studying. For example, in English the suffix "en" occurs frequently which might be mistaken for a true suffix and a morpheme which is incorrect. Therefore, it is important to take these problems into account to devise solutions for them. It can be useful to look at supervised segmentation in order to generate ideas and methods that might be useful in dealing with these issues.

Supervised learning is a machine learning technique of completing a task by providing training, input and output data. The training data contains annotated information that allows the program to learn whether the output it's produced is correct and punishes itself on incorrect output while rewarding and emphasising good behaviour that produces the labelled correct outputs.

A flaw about supervised learning is that it requires data that's doctored for good behaviour whereas unsupervised doesn't require any such doctored data.

It can prove useful to compare and contrast supervised methods vs unsupervised methods in order to make connections between both methods to generate analysis of the various algorithms or pitfalls that might prove useful in creating unsupervised methods. Therefore, it is important to understand some ideas of supervised learning.

In an ideal world, there would be an abundance of information related to a language (even if undiscovered) by the natives of the language. Unfortunately, the real world shows that the kind of information that would prove helpful in analysis for linguistics experts is not forthcoming and so

techniques need to be created and adapted to overcome these shortcomings. In the search for unsupervised learning it can prove useful to display supervised learning as an alternative to show its strengths and weaknesses. In the future it might be possible to use these supervised approaches on the data outputted by unsupervised methods.

These techniques could come under using annotated word forms with part of speech (POS) to learn what kind of words conjugate to create similar semantically related meaning within the rules of the language itself. There might be hand-coded rules such as in English the word "create"'s option to prepend "re" to create "recreate". You'd have the root word and a list of common rules that would be applicable to that root in a kind of mapping table. The POS would be useful information in determining what kind of words they are. As an example, the words: "root", "cling" and "kick" are all verbs, but "cling" is an irregular verb while "root" and "kick" are classified as more regular verbs. This means some rules may apply to all of them, but some may apply to the regular class containing "root" and "kick" that do not apply to the irregular verb class "lie" and vice versa.

### 2.5.1 Minimally Supervised Morphological Analysis by Multimodal Alignment

One method of minimally supervised morphological is a method of supervised learning of word morphology that uses prior knowledge and annotated dataset was used by Yarowsky and Wicentowski to create a lookup table of words and their affixes along with any potential stem changes prior to affix appendage. It uses these regular and irregular words forms to learn new unseen words and forms to add to the table.

In English many (but not all) words are regular and appending suffixes like "ed" or "ing" generally work as in the case of "punch" can be changed to past tense by adding "ed" in the case of "punched" or the present tense such as "punching". Therefore, each root word can be sourced to find out which canonical suffixes are suitable for adding to the lookup table. This works if the words are regular. However, there are many words that are not so easily defined, thus called irregular.
These kinds of words might be regular in English but for the purpose of segmentation they have different rules and are regarded as a different class from 'normal' verbs.
These words change stems in order to accommodate their new form. For example, "take" needs to be changed to "took" to denote past tense and "taking" for its present tense suggesting that not all verbs fit in perfectly with the aforementioned model. However most irregular verbs could be summed up in form {root (+ or -) stem-change + suffix} such as "got" would be reduced to "get" as its stem. Or "kick" + "ed" for an example of a more regular verb for their inflection with the inflection mapping table.
This distinction is however not as useful for agglutinative languages or highly inflected languages since their morphemes do not change or change too much when morphemes are glued together to form words.

Some of the methods outlined in using the prior information in order to add new words to the mapping table are outlined here to serve as an introductory explanation of supervised learning of word morphology.

#### 2.5.1.1 Lemma Alignment by frequency similarity:

For words that are regular, the simple appendage of a morpheme can work. For example, "root" -> "rooting", "rooted", "roots". However, there exists a pitfall that can easily be fallen for. These are verbs that are irregular such as "take". "Take"'s past tense is "took" rather than what would seem like the normal choice for a regular verb ending which would be "taked". Additionally, the morpheme "ed" has not been appended to "take" for this hypothetical example but it has been conjugated instead. This kind of irregularity forms with many verbs that end in "e".
There exists a pitfall within this pitfall for words that appear to have a regular verb appendage but actually are a different word entirely. The word "sing" transforms to "sang" when properly transformed but the word "singed" also exists so a program might find "singed" and try to assign it to the past version of "sing". Therefore, a technique called corpus-based frequency (number of occurrences of a word per

corpus) can determine if words are related as well as a number of methods outlined. Two of them have been highlighted and discussed.

#### 2.5.1.2 Lemma Alignment by Context similarity

Getting the semantic relation between words is a useful way of determining whether words are related such that they could be related morphologically meaning they are a few morphemes away (e.g. “Fantastic” -> “Fantastically”). Measuring the cosine similarity between vectors that resemble the semantic meaning of words proves a definitive method of clustering groups of morphemes. Word vectors are talked about briefly in the Range of approaches section.

However, this project uses unsupervised learning by not relying on annotated datasets so a range of methods for working with unsupervised learning are needed. But some ideas such as word embedding, and cosine similarity might prove useful in tackling unsupervised learning.

## 2.6 Range of approaches for unsupervised learning

For the task of analysing or studying a previously undiscovered language where very few resources exist an aspiring researcher might consult experts in the language if possible, such as natives or fellow analysts of the language. However, this can be expensive as such experts might be 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. In worst-case scenarios, experts on the language might not even exist. Thus, an automated tool that doesn’t require expert information would alleviate these problems.

Researchers in linguistics have written about a number of different methods useful for morpheme induction to alleviate this problem.

Goldsmith (2000/2001) describes four categories of approaches to unsupervised 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 of 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.

### 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 Techniques | 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 on 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 also many exceptions to this rule such as "baked" to "bake" + "ed" or "sing" to "sang". Despite this, it is unlikely that transposition would aid these kinds of morphemes. These kinds of outliers are a difficulty in solving the overall problem for the English language.

Finally, for this project it is unlikely that the corpus has spelling errors given that they are an edited word list 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 value. However, this technique is more useful for words that are always 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 not be feasible to use this technique.

Word embeddings are an assortment of other techniques 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 have a higher chance they're semantically related. The similarity of vectors should be measured by the cosine angle as the cosine similarity is unaffected by magnitude. However, it's important to note 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. (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 a probabilistic maximum a posteriori framework (Creutz & Lagus, Unsupervised Models for Morpheme Segmentation and Morphology Learning, 2007) to calculate morphemes. A list of morphemes is created from the dataset that builds upon itself.

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 creates costings of 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 number 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 boundaries 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 other kinds of words 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.

However, this does not currently account for fusional words that change stems (eat – ate), 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 removed ideally generating new words that can be tested on. (Lignos & Beck, 2011)

### 2.6.7 Word pairing

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 match with a word in the 0.01% list are paired together. This method uses the string edit distance and the pointwise mutual information outlined in the previous methods as techniques to determine morphemes. (Baroni, Matiasek, & Trost, 2002)

### 2.6.8 Keshava / Pitler

This method uses detection of substrings in strings and transitional probability as a basis for solving the problem of morpheme induction. While it attempts to follow methods such as word segmentation by letter successor varieties (A.Hafer & F.Weiss, 1974) it also tries to remove any empirical observations to make the algorithm more robust to different kinds of languages. Despite this it has fallen into the trap for a punishment rewarding mechanism that uses a >5% for scoring word parts that are more likely to be morphemes.

This method builds two types of trie structures which it uses to build fast, efficient transitional probability checks that are necessary as they make up the bulk of the operations of analysis and segmentation. Beyond that they are checked to determine whether the root of the word being tested has multiple children and that the root’s parent has only one child. These kinds of tests refer back to the problem of unsupervised learning of oversegmentation mentioned in 2.5 Unsupervised vs supervised approaches. These conditions actively attempt to confirm whether potential morphemes are true morphemes rather than mistaken assumptions.

If the word fragment passes all these tests, its counterpart gets rewarded (consider the word “reports” is tested, if “report” passes all tests, then “s” gets rewarded). This is continued for all words in the training dataset. Then all words that have failed (have been punished more than 95% of the time more than they’ve been rewarded) get removed.

Potential morphemes that are made from other smaller morphemes are pruned by testing the scores between the smaller morphemes, if the two smaller morphemes are higher than the bigger one the bigger is removed from the list.

Using the positive morphemes, affixes can be peeled off from the words by using a similar transitional probability check of all possible morpheme pairs. Then the highest scoring one (lowest value) is taken as long as it's within the boundary of less than the value of 1.

This approach will be used in the project work so more detail will go into it in the implementation and testing stage.

### 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 or F-score.

The recall denotes the amount of information that is obtained by the program that is correct 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 methods 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.

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.

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 another source has proven useful in providing a more accessible gold standard. This source is a program called pyports as will be talked about in 2.7.1 Pyports.

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 for Perl that runs this program 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 and doesn't have any information about which Perl versions are 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.

The 2007 version of the evaluation works by getting a large sample of word pairs from the gold standard and the diagnostic standard output by the program 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 2007 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%.*"

While this might be a more developed method of evaluating the options available were limited as pyports evaluation was available but not much else. Pyports uses the 2005 method of evaluation instead.

### 2.7.1 Pyports

From the morpho's challenge there is a 2005 method of evaluation.

*The evaluation is based on the placement of morpheme boundaries.*

***Example.*** *Suppose that the proposed segmentation of two English words are: boule vard and cup bearer s'. The corresponding desired (gold standard) segmentations are boulevard and cup bear er s '. Taken together, the proposed segmentations contain 2 hits (correctly placed boundaries between cup and bear, as well as between er and s). There is 1 insertion (the incorrect boundary between boule and vard) and 2 deletions (the missed boundaries between bear and er, and between the plural s and the apostrophe ' marking the possessive).*

Pyports is a program that has an evaluator implementing the 2005 version of the morpho challenge's evaluation method 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 python program seamlessly without needing to run a separate file for evaluation.

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's 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 word (e.g. "don't" really means "do not"). They would ideally need to be split up into their proper words to analyse but for segmentation the correct answer is not clear. If a word containing an apostrophe is in the gold standard should the program check it against its proper form or leave them both as they are and attempt to segment based on the apostrope. While some of the more simple examples of words containing apostrophes 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 proper split word forms 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 apostrophes 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.

### 2.8 Evaluation inner working

Pyports is a program containing an evaluation class that implements the 2005 version of the morpho challenge's evaluation that enables evaluation of the previously mentioned methods. This is because the Morpho Challenge's Perl script that ran their evaluation has depreciated features as it was written about 15 years ago and Perl has since broken some methods it relies on.

Regardless, the concept of the evaluation algorithm can be recreated as shown by pyports. The evaluation works by going through each word in the gold standard, checking it exists in the outputted word standard list. Checking the segmentations in the gold standard with the outputted word standard are correct or incorrect and summing up the number of true positives, false positives, and false negatives. Then outputting these as precision, recall and finally the f-score. Where the f-scores for two or more methods are the same the precision will be the secondary score.

The specifics of this evaluation method are explained in the implementation section in chapter 4.

### 2.9 Datasets / Challenges

There are two kinds of datasets that are intended for use. The morpho challenge's data and the project pyports data. While the morpho challenge ran from 2005 until 2010 it skipped out 2006 so it has 5 sets of wordlists while some of them are very similar with only a few changes made. 2005 is by far the smallest with 167,000 words, with 2007, 2008 and 2009 having roughly 384,000. 2010 is the largest set as it contains 878,000 words. However, the morpho challenge's data carries lots of junk data such as words like "öresund" which while it might be used in some obscure setting is still pretty harmful towards the morpheme analyser as its origin is not English and morphemes of words like that might be dissimilar to English. The methods intended to solve this problem might get tripped up.

Pyport's data is from a corpus and as such contains many repeated words meaning the words must be tokenised so words do not repeat their reward/punishment more than once. They are using practical data rather than a theoretical word list so roots for certain words might be missing meaning the proposed method might find it more difficult to work with as it requires those roots.

### 2.10 Examples of word segmentation

Here are some examples of word segmentations that might be helpful for understanding the project thus far. Each word will be translated to it's gold standard segmentation with its proposed segmentation form (morphemes) delimited by a "+".

| Word | Ideal segmentation |
|---|---|
| Root | Root |
| Rooted | Root+ed |
| Rooting | Root+ing |
| Payment | Pay+ment |
| Concerns | Concern+s |
| Weekend | Week+end |
| Loved | *Love+ed |
| Lover | *Love+er |
| Ticker | Tick+er |
| Redistribute | Re+distribute |
| Unrequited | Un+re+quit+ed |

| | |
|---|---|
| Intergrated | *Inter+grate+ed |
| Disorganised | *Dis+organise+ed |
| Unrelenting | Un+relent+ing |

*Words that conjugate the “ed” or “er” or other similar morphemes should theoretically be split into two. Unfortunately, none of the current methods researched have been able to perform this task.

# Chapter 3 – Planned Experiment and Analysis

## 3.1 Removing punctuation

Words containing any punctuation in the wordlist are problematic as they usually merge two words together into a contraction, compressing or truncating part of the word resulting in the morpheme boundaries being less clearly defined. For example, “don’t” is the contraction of “do” and “not” but unless a system properly separates these two words, the analyser nor segmenter algorithms will understand what to do. But if it does work correctly, it might return “do” and “n’t” which wouldn’t be helpful. Usually apostrophe’s take the place of single letters, but some dialects use words like “I’d” which would be collapsed to mean either “I had” or “I would”. “ain’t” is also a word used by various users which might translate to “{is/am} not” depending on whether the speaker is talking about a creature or object. These words are not easily transformed or segmented.

While an expert creating a rule-based system would replace these variants with their proper terms a program might find this difficult to do without a dictionary which for unsupervised learning wouldn’t be possible.

Hence, testing if removing words containing punctuation from being read from the wordlist would improve the performance and scoring of the program.

## 3.2 Thresholds Modification

One of the arbitrary numbers in the Pitler & Keshava method is the $P(x) \approx 1$. This means that for the second of the conditions needed to be passed in order for the word fragment to be rewarded, the probability needs to be near 1. However, Pitler/Keshava do not specify how near exactly. It would be of interest to confirm whether lowering or raising the threshold increases, decreases or doesn’t affect the f-measure after evaluation.

## 3.3 Removing the first condition (word existence check)

The first condition checks whether the word being tested is in the dictionary. While these words do serve the purpose of observation that prefixes and suffixes are usually added on to a base word it would be interesting to see the effects of ignoring this condition.

## 3.4 Word scoring reward / punishment adjustment

Specified in the method by Pitler & Keshava the word-part reward and punish mechanism is designed to create a list of viable morphemes. These values are specified as +19 for rewarding and -1 for punishment or in other words the reward is 19 times the magnitude of the punishment. This means viable morphemes need to be rewarded >5% more of the time than punished. As this is an observed variable changing the magnitude of the reward could affect the results.

## 3.5 Wordlist test

With each year of the morpho challenge competition, more data is accumulated. If this is the case it makes sense that the original word list contains more frequently used words whereas the later years accumulated a lot of junk data that aren’t as useful for analysing nor segmenting the words that normal corpus might actually use.

## 3.6 Frequency Check

Getting rid of words with less than x number of appearances might remove junk data that occur irregularly. This is because there are a lot of words in the wordlists that have been added that someone might use very rarely but have only been added because they have technically been used at some point.

### 3.7 Multiplying frequency by reward mechanism

Testing whether the frequency multiplied with the reward weighting would give off better results to take the frequency of a word appearing into account might prove useful as the frequency will have a proper weight rather than just scoring based on if it exists.

A final note about the different languages Pyports has supplied. Pyports has come with multiple language datasets for English, Japanese and Russian including wordlists, testing sets and gold standards. Unfortunately, trying to test the Japanese and Russian resources have not worked well and the results have been disappointing as the evaluation returned a score of 0 for all results. Due to the inability to confirm morpheme segmentation accuracy for either language this has been considered a dead-end.

# Chapter 4 – Implementation / Testing

For this project Pitler and Keshava's "A segmentation approach to morpheme analysis" has been chosen as it's the only method where sufficient detail has been provided with examples in order to be processed and written. It 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").

## 4.1 Getting Morpho Challenge Data

There are multiple wordlists and gold standards available on the Morpho Challenge's website ranging from 2005 to 2010. The 2010 version has the most amount of information, so normally it would make sense to use that version. However, the 2010 version also contains a lot of junk data - far more than 2007-2009 - of words that wouldn't really occur in normal English corpus.

The English gold standard file from pyports has some of the most common words that are used which would be the best at determining the best scores for evaluating.

## 4.2 Trie Structure and Nodes

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

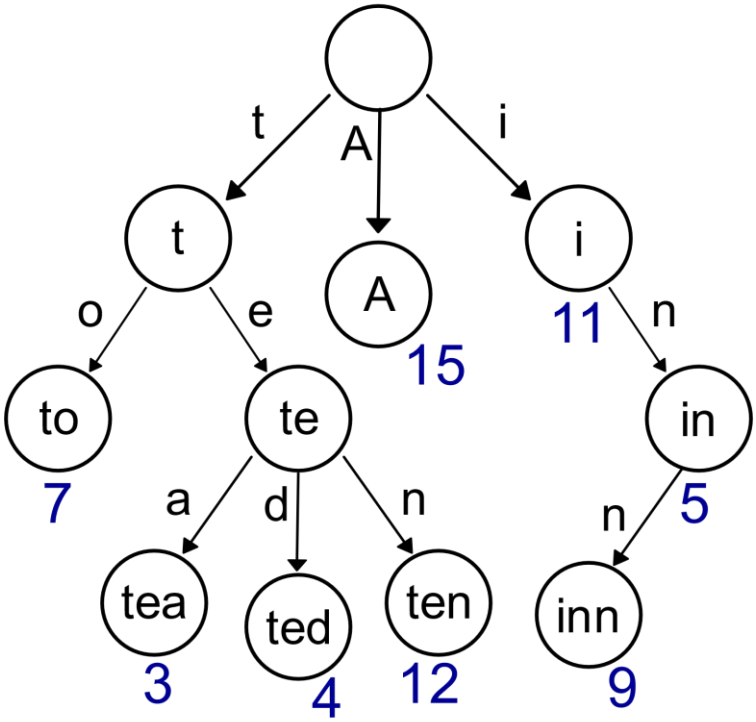



*Figure 1 public domain* [https://upload.wikimedia.org/wikipedia/commons/thumb/b/be/Trie_example.svg/1024px-Trie_example.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/b/be/Trie_example.svg/1024px-Trie_example.svg.png)

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 needed. Each possible node contains a single character and so 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” to form “tea”, “ted” or “ten”.

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

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:

$$P(A|B) = (P(B|A) \cdot P(A)) / P(B)$$

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”.

```
|_* (1)
  |-a (8649) - a
  | |-a (46) - aa
  | | |-a (3) - aaa
  | | | |-a (1)
  | | | | |_l (1)
  | | | |   |_l (1) - aaaall
  | | | |_s (1) - aaas
  | | |-b (1)
  | | | |_b (1) - aabb
  | | |-c (3) - aac
  | | | |_h (2)
  | | |   |_e (2)
  | | |     |_n (2) - aachen
  | | |       |_e (1)
  | | |         |_r (1) - aachener
  | | |-d (2)
  | | | |-e (1)
  | | | | |_l (1) - aadel
  | | | |_v (1)
  | | |   |_a (1)
  | | |     |_n (1)
  | | |       |_t (1)
  | | |         |_a (1)
  | | |           |_g (1)
  | | |             |_e (1) - aadvantage
  | | |-f (2) - aaf
```

*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 can be seen in this Figure 3.

```
161479  |-r (10327) - r
161480  | |-e (7367) - re
161481  | | |-n (706) - ren
161482  | | | |-e (56)
161483  | | | | |-h (5)
161484  | | | | | |-c (4)
161485  | | | | | | |-a (1)
161486  | | | | | | | |_a (1) - renehcaa
161487  | | | | | | |-t (1)
161488  | | | | | | | |_i (1)
161489  | | | | | | |   |_k (1) - renehctik
161490  | | | | | | |-i (1)
161491  | | | | | | | |_m (1) - renehcim
161492  | | | | | | |_n (1)
161493  | | | | | |   |_u (1)
161494  | | | | | |     |_m (1) - renehcnum
161495  | | | | | |_s (1)
161496  | | | | |   |_e (1)
161497  | | | | |     |_r (1)
161498  | | | | |       |_f (1) - renehserf
161499  | | | | |-r (4)
161500  | | | | | |-b (3) - renerb
161501  | | | | | | |-n (1)
161502  | | | | | | | |_e (1)
161503  | | | | | | |   |_h (1)
161504  | | | | | | |     |_c (1)
161505  | | | | | | |       |_s (1)
161506  | | | | | | |         |_a (1) - renerbnehcsa
161507  | | | | | | |_s (1)
161508  | | | | | |   |_i (1)
161509  | | | | | |     |_m (1) - renerbsim
```

*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 in O(1) time.

### 4.2.2 Adding to tries

This algorithm starts at the root node, 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 & Keshava’s 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.

### Tests for Trie

#### Adding to the Structure

```
root = TrieNode('*')
root.add("hackathon")
root.add('hack')
root.add('rep');
root.add('repo');
root.add('repor');
root.add('report');
root.add('reports');
root.add('root')
```

With this code the structure contains the root node along with a sample of other words chains. Printing this out with the function:

```
root.pprint()
```

Results in the following:

```
└* (1)
  ├h (2)
  | └a (2)
  |   └c (2)
  |     └k (2) - hack
  |       └a (1)
  |         └t (1)
  |           └h (1)
  |             └o (1)
  |               └n (1) - hackathon
  └r (6)
    ├e (5)
    | └p (5) - rep
    |   └o (4) - repo
    |     └r (3) - repor
    |       └t (2) - report
    |         └s (1) - reports
    └o (1)
      └o (1)
        └t (1) - root
```

#### Finding

Finding has 3 results applicable.

1. A word is found, and the value returned
2. The prefix is found, and the value returned
3. The prefix doesn't exist

Results 1 and 2 are practically the same, the only difference is the algorithm has not found an actual word that was in the wordlist and instead has found a fragment of other words.

Result 3 is the error condition. If this happens in the main program it will crash. There is intentionally no exception made for the finding function as these errors are helpful in determining bugs made by changes in the program. The probability function that simply divides the results obtained from the finding

algorithm will crash if it receives a 0 as in the word doesn't exist within the Trie structure. When the main program is running correctly this third condition should never happen.

This is the structure of the trie at this point

```
└* (1)
  ├h (2)
  | └a (2)
  |   └c (2)
  |     └k (2) - hack
  |       └a (1)
  |         └t (1)
  |           └h (1)
  |             └o (1)
  |               └n (1) - hackathon
  ├r (6)
  | ├e (5)
  | | └p (5) - rep
  | |   └o (4) - repo
  | |     └r (3) - repor
  | |       └t (2) - report
  | |         └s (1) - reports
  | └o (1)
  |   └o (1)
  |     └t (1) - root
  └c (1)
    └a (1)
      └t (1) - cat
```

*Figure 4*

While these are the inputs noting all 3 error conditions.

```
print(find_prefix(root, 'hackathon'))
print(find_prefix(root, 'hac'))
print(find_prefix(root, 'hack'))
print(find_prefix(root, 'hackathon'))
print(find_prefix(root, 'ha'))
print(find_prefix(root, 'hammer'))
print(find_prefix(root, "rep"))
print(find_prefix(root, "report"))
print(find_prefix(root, "reports"))
```

*Figure 5*

These are the results:

```
hackathon (True, 1)
hac (True, 2)
hack (True, 2)
hackathon (True, 1)
ha (True, 2)
(False, 0)
rep (True, 5)
report (True, 2)
reports (True, 1)
```

*Figure 6*

Once the probability function reaches the 0, as the word “hammer” doesn’t exist it will attempt to divide by 0 crashing the program. As the program works symmetrically it doesn’t matter if a numerator is 0 because at some point the same string will be passed to the denominator and the proceeding attempt to divide by 0 will crash the program.

For the rest of the inputs, as can be seen from Figure 4 they match up perfectly with the trie structure.

### 4.3 Inputs and Outputs

#### 4.3.1 Inputs

The inputs are configured by the command line arguments. The input file by the argument -i, the output file by -o, the gold standard by -g, and the testing file by -t. 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. If there’s no testing file with -t then the testing file is set to the input file.

The input receiver does use a trick, as it ascertains whether the frequency exists in the wordlist through the use of a space check it can work with both files containing just words or “frequency word”. This does mean it requires either one of two specific formats.

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 & Keshava’s method are true.

For prefixes it works with the backwards trie and tests the 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 an aside it is important that the words are in hashable format that enables O(1) constant time lookup otherwise each word part will run an O(n) operation putting the morpheme analysis at an $O(n^2)$ 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 that the root has more than one potential child 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 the 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 word score dictionary. This keeps track of how many times a word fragment has been punished or rewarded.

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 to the prefixes but instead using the backwards trie for its 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

Pruning performs the task of checking if morphemes consist of two other morphemes. If the two smaller morphemes have a greater score than the first bigger morpheme then the bigger morpheme is removed. The program does this by looping through the morpheme in the word score, iterating across the words to see whether they exist within the word score. If both the first part and second part are in the word score, they are then tested against the original morpheme and removed if both the first part and the second part of the morpheme have a higher score than the original morpheme.

Prefixes and suffixes are separated for the pruning as morphemes in prefixes do not consist of morphemes in suffixes and vice versa.

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

| Word fragment | Score |
|---|---|
| “re” | 1800 |
| “s” | 2000 |
| “en” | -500 |
| “un” | 800 |

All but “en” remain in the wordlist.

Tests for scoring and pruning:

```
if __name__ == '__main__':
    ma = MorphemeAnalysis("data/wordlist-2007.eng")
    ma.is_suffix("report", "reports")
    ma.is_suffix("mountain", "mountains")
    ma.is_prefix("re", "reports")
    ma.is_prefix("re", "report")
    ma.is_prefix("re", "retaking")
    ma.is_prefix("re", "resupply")
    pruned_suffixes = ma.prune_affixes(ma.word_score_suffix)
    pruned_prefixes = ma.prune_affixes(ma.word_score_prefix)
    print(pruned_suffixes)
    print(pruned_prefixes)
```

*Figure 7*

Testing this on the words for: “reports”, “report”, “mountains”, “retaking”, and “resupply”.

“Report” and “reports” correctly segments the “s” from the end of “reports” as it does from mountains too. However, the scorer only picks up the “re” from resupply and not from “retaking” or “report” suggesting its sensitivity is not as strong as it could be. According to the gold standard from pyports, the word “report” doesn’t actually segment the “re” in “report”, but the word “retaking” should trigger a segmentation showing that the algorithm or at least this implementation is not perfect.

```
{'s': 38}
{'re': 16}
```

*Figure 8*

As can be seen from Figure 8 the score for “s” is +19 for both “reports” and “mountain” while the prefix for “re” has been both rewarded and punished. It’s been rewarded for “resupply” but punished for “reports, “report” and “retaking” adding 19 but subtracting 1 three times.

Pruning is necessary here because the dictionaries containing the prefixes and suffixes are initialised with every word in the wordlist so navigating to the words that are not 0 would be troublesome as there’s nearly 400,000 for this wordlist.

### 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 pruned morpheme list to only

keep morphemes that have been rewarded 5% of the time more than punished. The segmenter will go through the word again and get all possible combinations of word parts.

Then with those potential prefixes and suffixes the probability of the part of the word specific to the prefix or suffix is tested and the combinations of the 1st and 2nd part of the words are scored. Whichever is less than 1 and is the lowest is the deciding segmentation. If the transitional probability segmentations are not less than 1 then the word is left as is. If there are two with the same value it will just pick the first one.

For example, for suffixes the forward trie is used to get the probability of the prefix word part divided by the probability of the prefix word with the first letter of the suffix word added on.

For prefixes, the backwards trie is used to get the probability of the reverse of the last letter of the prefix word appended to the suffix word divided by the probability of the suffix word.

Unfortunately, the prefix segmentor is too sensitive and picks up a lot more errors than picking up correct segmentations. Due to this, disabling the prefix segmentor actually increases the precision and f-score of the program. However, from observing the affected prefixes it's been possible to optimise it a bit by removing word parts that are of length one. Even so the scoring is worse with the prefix segmentation working by about 4% better (63% □ 67%) with that optimisation (otherwise it would be roughly 51%).

Tests:

```
print(ma.segment_suffix("results"))
print(ma.segment_suffix("staying"))
print(ma.segment_suffix("falling"))
print(ma.segment_suffix("purchased"))
print(ma.segment_suffix("root"))
print(ma.segment_suffix("beneficiaries"))
print(ma.segment_prefix("disillusion"))
print(ma.segment_prefix("unhappy"))
print(ma.segment_prefix("undoing"))
print(ma.segment_prefix("unfold"))
print(ma.segment_prefix("disability"))
```

```
re_sult_s
stay_ing
fal_l_ing
purchas_ed
root
beneficiaries
di_sillusion
un_happy
un_doing
un_fold
di_sability
```

*Figure 10*

As can be seen from Figure 9 and Figure 10 the results from testing these words. For the "re" it works well as well as all of the suffixes. However, there is too much sensitivity for the prefixes as it's picking up "fal" and "di" instead of "fall" and "dis". As the "dis" has the same probability as "di" a consistent but random one gets chosen depending on the ordering of the word fragments. Attempts to solve such as allowing the longer word first would render the solution language specific and allow it to work on some words but reduce the efficacy of other words. Similarly, with pruning this would be avoided if the prefixes and suffixes would be tested together. However, in this case the "s" from "dis" is not the same as one denoting plurality. "s" is not a prefix which is why it is not pruned from the prefixes.

### 4.6 Evaluation

The method of evaluation iterates through the gold standard list of words, checking if the word is in the output proposed word segmentations which will be called "word standard" for brevity. The gold standard word's morphemes are compared with the word standards.

True positives are recorded where there is a correctly placed segmentation marker between morphemes in both the gold standard and the word standard. For example, if the gold standard reads *"cup bearer"* and the word standard also reads *"cup bearer"*.

False positives are recorded for situations where the segmentation marker is placed incorrectly in the word standard.

False negatives are recorded for situations where the segmentation marker has been placed in the gold standard but not within the word standard.

These three values are used to calculate the precision and recall. The precision is calculated by dividing the true positives by the true positives + the false positives. The recall calculated by dividing the true positives by the true positives + false negatives. Finally, the f-score is calculated by calculating the harmonic mean between the precision and the recall.

Tests:

```
golds   = ['re+port+s', 'un+dy+ing', 'hypno+tic', 'reticient']
results = ['re+port+s', 'undy+ing', 'hypno+tic', 'retic+ient']
```

*Figure 11*

```
INIT-GOLD-STD: ['re+port+s', 'un+dy+ing', 'hypno+tic', 'reticient']
INIT-RESULTS: ['re+port+s', 'undy+ing', 'hypno+tic', 'retic+ient']

GOLD-STD: [('re', 'port', 's'), ('un', 'dy', 'ing'), ('hypno', 'tic'),
('reticient',)]
RESULTS: [('re', 'port', 's'), ('undy', 'ing'), ('hypno', 'tic'), ('retic',
'ient')]

True positives: 4
False positives: 1
False negatives: 1

Precision: 0.8
Recall: 0.8
F-measure>: 0.8000000000000002
```

*Figure 12*

As can be seen from Figure 11 and Figure 12 the evaluation tests the segmentation marker's position to test if they're in the right position. As 4 markers are in the correct position ("re+port+s", "+ing", "+tic") they are awarded 4 true positive results. However, there is one in the incorrect position in "retic+ient" and a position where a market should have been but wasn't ("un+").

## Chapter 5 – Experiment Results

These experiments were designed in mind to discover tweaks or quirks to increase the f-score or to discover interesting quirks that might allow the discovery of what is causing the issues of programmatic morpheme segmentation that are difficult to solve.

As with all tests the control must be defined. This is known as the baseline score. The baseline score uses suffix segmentation only with the 2007 wordlist, threshold at 0.05, rewarding at 19. Results have been truncated to 5 significant figures. The results are in decimal form:

| Precision | Recall | F-Score |
|---|---|---|
| **0.56923** | **0.71154** | **0.67766** |

### 5.1 Removing Punctuation from wordlist

| Precision | Recall | F-Score |
|---|---|---|
| **0.56923** | **0.71154** | **0.67766** |

As this result is the same score as the baseline it can be surmised that the program was unable to deal with words containing punctuation regardless and the same pitfalls it fell into the without punctuation it fell into this time too as there has been no ability to deal with punctuation.

### 5.2 Threshold Modification

| Threshold | Precision | Recall | F-Score |
|---|---|---|---|
| **0.05** | **0.56923** | **0.71154** | **0.67766** |
| 0.01 | 0.58659 | 0.67308 | 0.6538 |
| 0.003 | 0.58659 | 0.67308 | 0.6538 |
| 0.002 | 0.58659 | 0.67308 | 0.6538 |
| 0.001 | 0.58659 | 0.67308 | 0.6538 |

When the threshold is 0 the recall obtains practically nothing showing that most of the words do not fit neatly within the equation which is why Pitler & Keshava have determined the threshold to be necessary. Increasing the threshold reduces the score while having it at the same magnitude type values (0.00x) doesn't change it at all until later values. Interestingly enough the interpretation originally given is the highest scoring value at 67.8%. This is most probably language and dataset driven and different but undeterminable thresholds might work best for different languages or datasets.

### 5.3 Removing the first condition (word existence check)

| Precision | Recall | F-Score |
|---|---|---|
| **0.27273** | **0.67308** | **0.52032** |

Without the first condition checking the currently testing word is in the dictionary the precision drops heavily dropping the F-score. This is most likely because the number of false positives the evaluator picks up is increased since there are a higher number of morphemes that haven't been pruned. The segmentor function is adding segmentations where they shouldn't exist.

### 5.4 Arbitrary Scoring

| Reward | Precision | Recall | F-Score |
|---|---|---|---|
| 21 | 0.52174 | 0.69231 | 0.64982 |
| 20 | 0.53846 | 0.67308 | 0.64103 |

| | | | |
|---|---|---|---|
| 19 | 0.56923 | 0.71154 | 0.67766 |
| 18 | 0.55897 | 0.69872 | 0.66545 |
| 17 | 0.63690 | 0.68590 | 0.67551 |
| 16 | 0.64848 | 0.68590 | 0.67807 |
| 15 | 0.66061 | 0.69872 | 0.69075 |
| 14 | 0.67901 | 0.70513 | 0.69975 |
| 13 | 0.68790 | 0.69231 | 0.69142 |
| **12** | **0.70779** | **0.69872** | **0.70051** |
| 11 | 0.71053 | 0.69231 | 0.6958 |
| 10 | 0.71053 | 0.69231 | 0.69588 |
| 9 | 0.73913 | 0.65385 | 0.66929 |
| 8 | 0.75000 | 0.65385 | 0.67105 |
| 7 | 0.75000 | 0.63462 | 0.65476 |

According to these results it is in fact the lower value scorings that are better with an f-measure score. More than 3% better than the value of 19 offered by Pitler & Keshava. However, it is important to note that these are empirical observations and might perform better or worse depending on the data the program has been given.

### 5.5 Wordlist Testing

| Year | Precision | Recall | F-Score |
|---|---|---|---|
| Morpho 2010* | 0.60119 | 0.63522 | 0.62811 |
| Morpho 2009 | 0.56923 | 0.71154 | 0.67766 |
| Morpho 2008 | 0.56923 | 0.67766 | 0.66871 |
| Morpho 2007 | 0.56923 | 0.71154 | 0.67766 |
| Morpho 2005 | 0.51256 | 0.68000 | 0.63830 |
| Pyports en_conll2000_train | 0.6875 | 0.64706 | 0.65476 |

*Morpho 2010 requires the format to be in latin-1. This variable is at the start of morpheme_analysis.py

2007 till 2009 are relatively stable. 2005 has 1 MB of words while 2007-2009 have 4 MB. However, 2010's wordlist shoots up to 11 MB suggesting that a lot of junk data not suitable for morpheme segmentation was provided. This and 2005 where not as much data is available lowered the performance of the segmenter quite a lot whereas the others didn't make a huge difference.

### 5.6 Frequency Check

| Frequency | Precision | Recall | F-Score |
|---|---|---|---|
| 1 | 0.6397 | 0.64780 | 0.64617 |
| **2** | **0.56923** | **0.71154** | **0.67766** |
| 3 | 0.53093 | 0.66452' | 0.63268 |
| 5 | 0.50262 | 0.61935 | 0.59186 |
| 10 | 0.52239 | 0.68182 | 0.64259 |
| 15 | 0.53140 | 0.72368 | 0.67485 |
| 20 | 0.52427 | 0.72 | 0.66998 |
| 25 | 0.53202 | 0.71053 | 0.66584 |
| 50 | 0.54229 | 0.72667 | 0.68040 |

Increasing the frequency required to add a word to be tested and segmented doesn't help as much as one might assume. As the frequency goes up the sample size goes down which explains the fluctuating past 5-10. It seems that requiring 2 is the magic number to reduce junk data's influence in this dataset.

### 5.7 Multiplying frequency by reward mechanism

| Precision | Recall | F-Score |
|---|---|---|
| 0.18327 | 0.58974 | 0.40853 |

This kind of idea appeared to perform very badly with a score of 40.9%.

# Chapter 6 – Conclusion and Future Thoughts

## 6.1 Conclusion

After performing an analysis of the methods and techniques along with their tweaks to adjust for performance it appears the very best results that can be produced in English are an F-score of ~70% which is rated as good at solving the problem. In the final morpho challenge of 2010 this would rate among the best of the methods competing.

However, this is partly due to not using the 2010 method of evaluation. For the 2005 competition and using the 2005 evaluation it would also rate near the top (top 2) but this is because tweaks have been made to get the best possible results based on the results such as the reward/punishment. A blind test might not be as effective.

The parameters that best support this score changes the +19 to a +12 in the reward / punish mechanism. The algorithm for deciding the morpheme boundaries is similar to what humans intrinsically do in order to ascertain new words when learning their first language. While they don't perform the calculations, they are sensitive to previously observed morphemes (the morpheme analysis stage) and when come across a new word they segment it based on the morphemes they do know. Humans are able to comprehend the meaning of new words when they are able to break it down into its morphemes.

It is disappointing these results weren't reflected in the Russian or Japanese wordlists but this algorithm was developed from observing Indo-European languages and so the results are not too surprising.

Additionally, it was disappointing to see the prefix segmentation enabled the score dropping. If multiple methods were used to double check whether an affix was correct it would have been beneficial but unfortunately this project has only used one method and despite a few other ideas having attributed to improve the score the prefix remains non-functional.

The program has been sped up as the original code was written in Perl in 2005 while python, a more modern programming language has been used in this program. The analysis time which remains the slowest part takes about 11 seconds for nearly 400,000 words while Pitler & Keshava took 34 minutes to calculate 167,000 words suggesting there were some inefficiencies in their work. An earlier iteration of the implementation of Pitler & Keshava's work took 24 minutes to analyse a trimmed version of the wordlist that held only 100,000 words which was later sped up.

Despite this, a good effort has been made to implement a solution with other kind of ideas, techniques and tweaks taken from other methods. While this tool won't be too helpful at solving outside Indo-European languages it can act as a starting anchor towards providing a basis of information for researchers or second language learners.

## 6.2 Future work

If the timeline for this project was longer there would be additional avenues to explore that might help solve this problem of morpheme segmentation:

1. The Pitler & Keshava's method is a reasonable model of how the human mind works with analysing morphemes and segmenting via use of transitional probability. However, humans make use of prosodic information which is not available in any dataset currently. If there was a method of storing this information, there might be more opportunities for the learning method of morphemes to work even more effectively. This information might be gained from speech synthesis of natives speaking the language and recording the sounds uttered along with the prosodic information.

2. One of the tasks that would have proved useful was multiple affix peeling so words like “unrequited” would have been peeled off for 2 prefixes, 1 root and a suffix rather than just the highest scoring prefix.
3. Rework the prefix segmentation. The prefix segmentation actually reduced the f-measure meaning its sensitivity is too high and needs to be dialled back to correctly segment words while avoiding other less likely prefix candidates.
4. Figure out an algorithm to further determine which morpheme is best when two morphemes have the same score in the condition for segmentation.
5. Changing different stem forms to root words by measuring the semantic similarity between similar words (like “take” “took”) and segment them successfully.
6. Apostrophes in the gold standard are still bringing down the score for gold standard words containing them. Therefore, while it still might be useful to ignore apostrophes in the morpheme analysis stage it might be useful to split words containing apostrophes into their original form to segment them and then retain their apostrophe position to replace with the letter added back after. For example, “don’t” would translate to “do not” so the segmentation marker would be added between “do+not” and when translated back to apostrophe form it would become “do+n’t”.
7. Bring in methods from Morfessor to see if the performance can be improved if combined with the Pitler & Keshava method. Currently the Morfessor concept uses a lot of optimisations that are not trivial to understand and program but in future it could be possible to use what has been learnt to comprehend and implement the ideas.

8. For performance during morpheme analysis, when looking up probabilities it's always comparing two probabilities against each other. Some lookups are redundant as only words that are similar are being compared. e.g. "report" and "reports" or "r" and "report". This means that instead of performing O(2k) calculations where k is the number of letters in each word it could perform O(k) as it'll just reuse the calculation. This can be done by tabulating the data in a dictionary instead of the trie structure. This kind of idea could potentially reduce the number of computations by up to half.

9. There are certain rules of words in English that are consistent but less regular than normal. An example of these are known as verbs ending in “e” such as “bake” or “rake”. In future finding a method for dealing with these words where the morpheme they are adding is an “ed” but they are also retaining their “e” to give “bake” + “ed”.

10. Adding multicore processing would make the process of analysing morphemes and segmentation incredibly fast to enable iterations to be much faster. This could mean using a program in real-time on a smartphone as it needs that extra performance. This might make it feasible for researchers to explore and research undiscovered languages with a smartphone while analysing them in real-time where accessing a computer would be difficult. As the morpheme analysis stage can be divided into tasks for each letter of the alphabet that is being used. For example, core 1 can take care of all words starting “a” and core 2 can take care of all words starting with “b” as there is no overlap. The only proposed problem with this is they are writing to the same dictionary, but this can be alleviated by each having a separate dictionary and only merging them sequentially once they’ve finished. This could mean multi core processing could increase the speed and could be achieved if the dataset was large enough where it needs that speed to be run in reasonable times. However, this did not make sense for this project as the sample size was low and computation power available was high. However, if this kind of project would be run on a smaller device with less performance it could be beneficial to speed up times to run in reasonable time for their purpose.

All these ideas could bring us closer to analysing languages providing more opportunities for researchers.

## References

A.Hafer, M., & F.Weiss, S. (1974). Word segmentation by letter successor varieties. *Information Storage and Retrieval*, 371-385.

Ager, S. (n.d.). *Yoruba language*. Retrieved from Omniglot: https://www.omniglot.com/writing/yoruba.htm

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

Beaver, I. (n.d.). *Python Package Index*. Retrieved from https://pypi.org/project/pycontractions/

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.

Booyabazooka. (2006, July 15).

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.

Kilic, O., & Bozsahin, C. (2012). Semi-supervised morpheme segmentation without morphological analysis. *LREC 2012.* Istanbul: LREC.

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.

Mathias Creutz, Stig-Arne Grönroos, Oskar Kohonen, Mikko Kurimo, Krista Lagus, Peter Smit, & Sami Virpioja. (2010). *Morpho project*. Retrieved from The Morpho Challenge: http://morpho.aalto.fi/projects/morpho/

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