Content

CHAPTER ONE: INTRODUCTION **1**
1.1 Computational Linguistics and Natural Language Processing 2
1.2 Stemming 3
1.2.1 Porter Stemmer 5
1.2.2 Lancaster Stemmer 6
1.2.3 Snowball/Porter2 Stemmer 8
1.3 Problem 9
1.4 Motivation 9
1.5 Aims and Objectives 10
1.6 Research Questions 10
1.7 Methodology 10
1.7.1 Data 11
1.7.2 Method of Analysis 12
1.8 Limitations 13

CHAPTER TWO: LITERATURE REVIEW **15**

CHAPTER THREE: ANALYSIS **19**
3.1 Porter Stemmer 21
3.2 Snowball (Porter2) 25
3.3 Lancaster Stemmer 28

CHAPTER FOUR: CONCLUSION **33**

References **34**
Appendix A: Data Analysis Notebook **37**
Appendix B: Dataset and Processed Data **43**

CHAPTER 1

INTRODUCTION

CHAPTER ONE: INTRODUCTION

1.1 Computational Linguistics and Natural Language Processing

The goal of understanding the various mysteries of human 'Language' is at the core of the field of linguistics. These mysteries include various questions, such as the questions of language acquisition, language origin, cognitive and neurological basis of language, etc. There are many approaches and methodologies, taken and applied, to find the answers to a number of these questions. These approaches and their end goals define and segregate the various fields of linguistics.

One such field is the field of Computational Linguistics. Computational Linguistics is a dynamic and interdisciplinary field - a convergence of the fields of linguistics, computer science, and artificial intelligence. The field of Computational Linguistics is dedicated to unraveling the mysteries of Human Language by employing various computational methodologies and tools. It aims to develop models of human language, to understand its workings and intricacies, as well as instill in machines the capabilities to mimic human linguistic behavior.

Natural Language Processing (NLP) is a sub-discipline of Computational Linguistics, which focuses on the application of the understanding and knowledge of the human language, which is derived from the field of Computational Linguistics, to develop machines capable of mimicking various aspects of human linguistic behavior, such as comprehension, interpretation, and generation. The primary goal of this field is the development of practically applicable language technologies and tools to solve various real-world problems. The applications include - text summarization, text classification, chat-bots, virtual assistants, machine translation, information retrieval, sentiment analysis, and many more.

The field of Natural Language Processing presently is majorly focused on textual data. The applications thus are also text-based, such as chat-bots, sentiment analysis, information retrieval, text summarization, etc., but these are the more complex and advanced applications derived from various other simpler applications of Natural Language Processing.

As discussed, to develop advanced applications, there is an acute need for elementary-level processing. Thus any sophisticated NLP application can be presented or described in the form of a pipeline[1], at the starting point is the raw textual data, and at the end is the final expected response or output. Thus, they are a collection of many smaller NLP processes. One such highly influential application is the Search Engine, which in its essence is a highly sophisticated information retrieval system. This system works in a sequential manner starting from raw textual data, which is then normalized, standardized, annotated, and indexed, after that other statistical searching/extraction methods are applied to find the relevant information.

Normalization is an extremely important step as it fulfills a lot of functions, such as improving the efficiency of subsequent processes and accuracy of search queries, as well as reducing anomalies in results and redundancy of data. There are many text normalization processes such as tokenization, spelling corrections, lemmatization, stemming, etc. Among them stemming and lemmatization are the most crucial ones (most of the others are a given in any NLP application) for applications involving information retrieval or information extraction.

The focus of this study is one of these processes - **Stemming**.

1.2 Stemming

Language is a combination of elements of different natures at different levels adhering to different underlying rules. Sounds combine to form morphemes, which then combine to form words, a combination of which under given syntactic and semantic conditions form clauses, phrases, and sentences, which ultimately form a discourse. Each level of combination adds to complexity and adds further information to the system.

One such level is the level of a word; this level may be called the morphological level, but the morphosyntactic level presents a more accurate outlook. Words are generally a combination of one or more morphemes. These morphemes combined to be in the form of roots, bases, affixes, or stems. Roots are the basic unit of meaning that cannot be further

[1] A software pipeline is a sequence of interconnected software tools or components that process data or perform specific tasks in a linear fashion, often used for automating complex workflows or data transformations.

divided into smaller meaningful units, they serve as the foundation for forming words and are often the core element in a word's structure (Cao 2022, November). Bases are somewhat similar to roots, as they are also at the core of words and accept inflectional as well as derivational affixes. Stems are special kinds of bases that only accept inflectional suffixes. Affixes are special morphemes, that add extra morphosyntactic information to the word, and depending on where they combine with base/root/stem, can be called suffix (at the end), prefix (at the beginning), infix (in middle), circumfix (around).

Now as we can see, affixes provide extra morphosyntactic information depending on the word class, such as tense, number, etc., thus we can assume that the core meaning of a word lies in its root/base/stem. This very assumption is the basis for all stemming algorithms. Let's look at the words *happy*, *happily,* and *happiness*. They are of different word classes, expressing different notions of the idea 'happy', but at the core, they are derived from that very common idea - the idea of happiness or joy. All of the said words are different forms of the root word *happy*. This is in a way similar to lemmas.

Stemming is a text normalization process, which tries to emulate this very idea. It is a procedure that reduces words with the same stem to a common form (Lovins, 1968). It is in a way different from lemmatization, lemmatization yields accurate root words, whereas stemming simply strips off the suffixes from the words, which may not always result in very accurate spellings. This normalized form of words is used for improving the efficiency and accuracy of several information processing-related tasks, such as information retrieval, indexing, etc., it reduces the text in the documents to be searched from, into stems and uses various statistical methods for indexing, similarly, it reduces the search query to their relative stems, and uses them to find corresponding information or documents.

Let's take an example of a document retrieval system to understand better how this may work. For that, we need to understand a foundational statistical measure used in information retrieval - TF-IDF. TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure used in various research fields. It quantifies the importance of a term in a document or a collection of documents. TF-IDF takes into account both the frequency of a term in a document (TF) and the rarity of the term in the entire document collection (IDF). It is commonly used in natural language processing tasks such as text classification, information retrieval, and sentiment analysis. TF-IDF has been applied in different contexts,

including scene perception and eye movement prediction (Laubrock & Schlangen, 2023). Thus, it is a measure that essentially relies on word frequency to quantify information present in various documents, which is then used to fetch documents that match specific queries, and so, if one were to search for, say, *computation*, without normalization of terms using stemming, the measure would only fetch documents with a high frequency of this very specific term.

But it is very rare for people to be looking for something this specific, their query is less focused on specific terms and more on the idea. Thus, the said query may be intended for documents also containing the terms, *computers*, *computer*, *computationally*, *computerized* etc. In such situations, the impact of stemming comes into the picture.

Let's look at it from another perspective, if a document retrieval system were to index unstemmed text using the TF-IDF statistics or any statistic per se, it would have to account for every individual term, it will treat, *computer*, *computers,* etc. as individual terms, and thus calculate the statistic for every such term, and the number of such term without being reduced by the application of a stemming algorithms, would be much larger than if it were applied, resulting in a very inefficient system, as well as a very inaccurate one, as we could derive from the previous perspective.

Thus stemming is an essential process from various higher NLP applications. This study focuses on three of the most prominent stemming algorithms - The Porter Stemmer, The Lancaster Stemmer, and Snowball (aka Porter2).

1.2.1 Porter Stemmer

Porter Stemmer is one of the first stemming algorithms and was designed by Martin Porter, a computer scientist, and linguist, in 1980. It is one of the most widely used stemming algorithms to date (in 2023, it is still a go-to choice for most NLP applications). Let's take a look at the algorithm’s design.

Consonants (denoted by *c*) are any letters other than A, E, I, O, U and Y (when preceded by a consonant), letters other than consonants are vowels (denoted by *v*). An

uninterrupted sequence of vowels (*vvv...*) and consonants (*ccc...*) are to be denoted by *V* and *C* respectively, and any word or part of a word can be represented in four ways -

CVCV...C
CVCV...V
VCVC...C
VCVC...V

Which can be generalized into a standard form -

[C]VCVC...[V] ([] denotes the arbitrary presence of consonants and vowels).

It can be further reduced to the form -

[C](VC)^m[V]

Here *m* denotes the number of times *CV* is repeated and is called '*measure*' of a word, this measure along with other context-dependent rules and conditions is used to select the part that is to be stripped off from the stem. The general structure of these rules is as follows -

(condition) S1 -> S2

Which implies that if a word ends with the suffix *S1* and it satisfies the given *condition*, *S1* will be replaced by *S2*.

The algorithm works iteratively going through the whole set of rules until the *longest match* is found, i.e. the longest suffix that could be removed/replaced. If there is no match, the word is left as it is.

1.2.2 Lancaster Stemmer

The Lancaster stemming algorithm created by Chris D. Paice in 1990, was presented as an alternative to the Porter stemming algorithm. It takes a very aggressive approach for

stemming, this along with its design to form shorter stems results in a rise in the cases of overstemming, which we will discuss as we go further.

It works in a similar iterative manner as the Porter Stemmer, consisting of a set of rules that specify either the operation of replacement or deletion.

The rules are designed to be as a group of sections corresponding to the final letters of a suffix, the ending letters of words are significant for finding the corresponding rules. The rules are designed to be applicable to *intact* (words which have not been stemmed even once) words and otherwise. There are also provisions for *acceptability conditions* to prevent a certain degree of overstemming.

The rules are divided into five components:

a) an ending of one or more characters, held in reverse order

b) an optional intact flag "*"

c) a digit specifying the remove total (may be zero);

d) an optional append string of one or more characters;

e) a continuation symbol, ">" or "."

A sample rule would be:

deec2ss. { -ceed > -cess }

If we were to break it down into the said four components it would be as follows:

a) **deec** - the suffix in reverse

b) There is no need for an intact flag as this rule is applicable on words like ‘succeeded’

c) **2** - number of letters to be replaced towards the end

d) **ss** - letters that will replace those letters

e) > - '>' represent continuation of algorithm and application of more rules '.' marks the termination of the algorithm without further stemming

1.2.3 Snowball/Porter2 Stemmer

Snowball is another major contribution of Martin F. Porter in the domain of stemming algorithms. It is a framework for the development of stemming algorithms.

He proposed this framework for a couple of reasons, one of them being the difficulty of extraction of proper algorithmic explanations from source code for various stemming algorithms developed for languages other than English, and the other was also the issue of improper implementations of his own Porter stemming algorithms by others. The framework allows for programmers to write algorithms for stemming in the form specially designed for that very purpose, which can then be compiled into its equivalent program in another programming language such as ANSI C or Java.

Porter has provided on his website[2] a number of foreign language stemmers with structure (Porter 2001):

a) Algorithms implementation in Snowball
b) Algorithm described in a less formal English-language
c) Equivalent program in ANSI C or Java
d) Standard vocabularies of words and their stemmed equivalents are provided for each stemmer

All of which are used to pin down the definition of that given stemmer exactly.

He also presents an improved form of the Porter stemming algorithm on this website, named Porter2, which is implemented in Snowball. The changes are not very extensive, the reason for which we will see in the literature review section.

[2] See https://snowballstem.org/texts/introduction.html and https://snowballstem.org/algorithms/

1.3 Problem

Since the seminal paper "Development of a Stemming Algorithm" by JB. Lovins in 1968, there has been a lot of progress in both - the complexity and accuracy of stemming algorithms, and yet, none of the algorithms to date are a hundred percent accurate. Maybe they never can be.

Stemming algorithms, due to their design (which is dependent on the application or task at hand), can often be quite aggressive in the culling of suffixes, or at times quite lax in dealing with them, either of which can lead to inaccurate outcomes, overstemming in case of the former and understemming for the later.

In English, this problem essentially 'stems' due to the disparity between the written and spoken forms of the language. Any non-native English speaker can attest to this, as most of them would have struggled between 'their' and 'there', 'buy' and 'bye', etc. There are also spellings like 'translate' and 'translated', 'include' and 'including', where the stem essentially merges with the suffix when spelled. These kinds of situations in written language make the task of designing rules quite difficult (we will look at it in detail in the literature review chapter).

There have been well-thought-out and well-designed algorithms, which perform really well in many tasks. However, there are still issues like overstemming, understemming, and imperfect spelling, which arise essentially due to the nature of the written text. This is the problem we will be exploring in this paper. We would be looking at the output of the algorithms we discussed previously and try to identify situations in which these kinds of issues and inaccuracies arise.

1.4 Motivation

A few months ago, I was having a conversation with a peer who was studying stemming algorithms for Hindi. I had an opportunity to look at his work, and resultantly explore this domain. I understood the importance of stemming algorithms for a plethora of NLP applications, which are quite essential for the development of Language Technologies to

empower people and make a lot of technology accessible for them, and yet there are so many languages that are impoverished in this domain.

Thus, by performing this study, I wish to understand the ins and outs of stemming algorithms, how they are designed and the way they perform, the things that one needs to keep in mind when designing an algorithm, etc.; Such that, if I were to have an opportunity to address this problem at a relevant scale, I would have some degree of ability to do so.

1.5 Aims and Objectives

Similarly, as per the questions presented in section 1.5, we have decided on two broader objectives for this study.

- ***To compare the performance of the selected algorithms over a common dataset.***

- ***To identify a common pattern in the incorrect as well as seemingly acceptable outputs of these algorithms.***

1.6 Research Questions

As per the problem defined in section 1.3, we have identified two research questions for this study.

- ***What are the similarities and differences in the output of the selected stemming algorithms?***

- ***What are some of the common issues in the output of the selected algorithms with reference to the spelling of the initial words and their corresponding stems?***

1.7 Methodology

The study aims to analyze patterns in the output produced by the selected stemming algorithms when applied over a common dataset. The dataset we would be using for the study is the Brown Corpus, which has over 1 million tokens, 56057 types when considering case

(upper or lower case), and 42690 when ignoring it. Thus, performing an exhaustive analysis over the whole dataset - though possible, is a highly resource-consuming endeavor. Especially so for this study.

For this study, we are taking up the qualitative research methodology. We are going to apply the stemming algorithms over our selected sample and then analyze the output produced manually and subjectively report our findings.

For this study, we will be using the following tools:

- Python
- Jupyter Notebook
- NLTK (Natural Language Toolkit)
- Pandas

1.7.1 Data

We need to analyze what kind of output we receive from each of the stemming algorithms, stemming algorithms are applied to individual words. The dataset we would be using for this study as previously mentioned is the Brown Corpus. It was created in 1961 and has a collection of over a million words, across various genres.

We would remove the stop words and other functional words, retaining the remaining content words, this would be our primary dataset. As described in the previous section the Brown Corpus contains 56057 unique tokens when considering case and 42690 when ignoring it (including the functional words and stop words). Thus, we will be selecting a sample out of this primary dataset to apply analysis on.

Sampling

We will be applying systematic sampling for this study. The total size of the primary dataset after the removal of functional and stop words would be around 42000, we will be selecting

10 percent of this dataset. The words would be sorted based on word endings. We would then select every 10th word from this sorted dataset, which would give us around 4200 words, on which we will perform the analysis.

1.7.2 Method of Analysis

We would be using the Natural Language Toolkit (NLTK) for analysis. It will provide us with the Brown Corpus, and we will also be using its implementation of the selected algorithms (Porter, Lancaster, and Porter2 (snowball)) for this study.

As mentioned in the previous section, we would start by reducing the corpus to unique tokens, while ignoring the case differences - by reducing everything to lowercase, which will then be followed by the removal of most of the stop words and functional words. In the end, we'd be left with most of the content words, on which we apply the sampling discussed previously (section 3.1.1).

On the sampled datasets, we would apply all three algorithms. We would also apply the algorithms to the primary dataset. Following this we will analyze them from various perspectives, we will:

- Compare every output of each algorithm with each other, and observe where they produce common stems
- Observe which stems are closer to the real roots in the way they are spelled and which algorithms produce more accurate stems z their spellings
- Observe the efficiency of each algorithm in reducing the size of the whole dataset
- Compare each algorithm with each other individually
- Spot over-stemming and under-stemming issues

The aforementioned analysis would be performed mostly manually, with the use of the Pandas package, to tabulate the data.

The term manually here implies that the tables would be observed one after another, with every entry and every output compared visually.

1.8 Limitations

There are several limitations of this study, which mainly stem from the lack of resources and time, which is also a result of the fact that this study is part of a mini-thesis that is to be presented as a part of the PG curriculum.

Here are a few of the major limitations that we have identified:

- The dataset isn't exhaustive, i.e. it does not contain all the possible words in the English language.
- However, it is still a large dataset, and thus observations may contain only a partial set of issues.
- Subjectivity of identified issues.
- The issues are not structurally presented in a manner that could be systematically dealt with in the future.
- The approach studied in this paper is just the rule-based approach, there are many other approaches stemming out there
- We have only selected three of the most prominently used algorithms, there are others that have similarly been impactful to this field, e.g. XEROX stemmer.

CHAPTER 2

LITERATURE REVIEW

CHAPTER TWO: LITERATURE REVIEW

Stemming is the process by which words are reduced to their base form, this definition is a bit similar to that of *lemmatization*, in a way both of these have an intersection in certain aspects, but there is quite a subtle difference between these two. Jivani (2011) highlights their distinctions quite clearly - *Stemming* is the process of obtaining a 'stem' after applying a set of rules, which mostly include the removal of morphological affixes, without taking into account the POS, semantics, or context in which the word occurs. *Lemmatization* on the other hand involves obtaining a 'lemma' for corresponding words, by reducing different word forms to their root forms while taking into account the things that are ignored in stemming, i.e. POS, semantics, and context. It also highlights the core assumption in the design of any stemming algorithm - the words that are reduced from their morphologically complex forms to a common stem are assumed to be semantically related, i.e. having a core semantic meaning which is the same for all word forms when stripped of their morphological variations.

The development of stemming algorithms started with the goal of improving the accuracy and efficiency of information retrieval systems, and the foundational approach for the development of rule-based stemmers was first described by Lovins along with a first-ever published stemming algorithm in 1968 (Lovins 1968). It is an essential pre-processing step for any information retrieval application, and fulfills two major tasks - Firstly, it can improve information retrieval accuracy, especially in highly *inflective* languages and with short documents (Moral et al. 2014), and secondly, it improves the information retrieval efficiency by conflating variant forms of words (Larkey et al. 2007), as well as by reducing the storage capacity needed and increasing the computational load of the system (Moral et al. 2014). However, it may not be that effective overall in some instances in English information retrieval, and at times it can improve recall but at the cost of precision (Manning 2009).

As discussed in the previous section, Julia Beth Lovins, in her paper titled -"Development of a stemming algorithm", presented an approach that could be taken for the development of most rule-based stemming algorithms. This paper also contains the first published English Stemmer. Porter (2005) discusses the importance of this stemmer by Lovins, he highlights its significance by stating that she didn't receive the appropriate credit she deserved for her work.

However, today most of the works that refer to stemming can only go with the mention of this work by Lovins. Porter (2005) also discusses the relevance of her work at that time, especially with the consideration of the "*information explosion*", which refers to the exponential growth of publications and the challenges it posed for libraries and information centers in providing relevant information quickly (Voos 1971).

Lovins (1968) talks about the development of a stemming algorithm as a part of Project Intrex, which tried to redefine the library and associated technologies. She has a two-phased stemming system - The first phase involves retrieval of the stem by identifying and removing the *longest possible ending*, and the second phase involves the handling of *spelling exceptions* such as *absorbing* and *absorption* being producing stems *absorb* and *absorpt* respectively in the first phase, but resulting in under-stemming. Which then, in the second phase is corrected through the process she describes as *recoding,* which is simply the changing of the ending after the first phase to match the appropriate expected ending.

Lovins uses the term "*ending*" rather than using "*suffixes*" to describe the word endings that are to be dealt with.

Martin F. Porter (1980) presents a simple yet efficient stemming algorithm. When compared to the Lovins-type stemmers, the approach employed by Porter is significantly simpler, reducing the required number of rules by a huge margin. For a perspective, Lovins Stemmer uses around 294 endings, each associated with 29 context-sensitive rules determining whether or not to remove the specific word ending, along with 35 other *recoding* rules (Lovins 1968). In contrast, Porter uses around 60 suffixes, using two *recoding* rules along with a single type of context-sensitive rule, dependent on the "*measure*" of the word (which we have discussed in section 1.2.1). This provides a comparatively simpler and efficient method for stemming (Willet 2006). If we were to continue this comparison, Lancaster stemmer has a set of 129 stemming rules (Radu et al. 2020, May).

Porter Stemming algorithm is one of the most widely used stemming algorithms for Information Retrieval to date, which is the consequence of its previously discussed simplicity and efficiency (Singh & Gupta 2017). We have discussed the general structure of the algorithm in section 1.2.1, thus, we will not discuss that in detail here. The stemmer

Paice (1990) discusses various algorithms that were developed during the time, along with it, he describes the Paice/Husk stemmer which was developed and used at Lancaster University for several years, thus eventually getting the name - Lancaster Stemmer. He also expresses that the performance of stemming algorithms is measured on the basis of the improvements in performance for an Information Retrieval test collection, but the viability of rules in an algorithm cannot be measured in this manner. He ties this to the fact that most of the literature treated "Porter Stemmer" and the "Lovins Stemmer" as fixed immutable entities, and no attempt being made to optimize them, which could now either be optimized on the basis of performance measures based solely on their corresponding improvements in IR tasks.

Paice (1990) presents a single iterative rule table, each rule specifying the deletion or substitution of an ending. This stemming approach, in contrast to the Lovins approach, skips over the *recoding* stage, as the same is already merged in the substitution part. Also, when compared to the Porter (1980) approach which goes through five stages, with a different lookup table in each stage, the approach used here is simpler in the sense that it uses one iterative lookup table, with around 129 rules. It uses restrictions like "*intact*" word restrictions and a blanket *acceptability check*, before applying the rules (the general structure of the rules is discussed in section 1.2.2).

Years after the publishing of the paper titled "An algorithm for suffix stripping", and presenting the now-known Porter Stemmer, Martin F. Porter came up with the Snowball framework - A framework for stemming algorithms, in 2001 (Porter 2001).

(Porter 2001) discusses the necessities that led to the development of such a framework. Firstly, there was the lack of dis-ambiguous algorithms for languages other than English, along with the extreme difficulty in the extraction of algorithmic descriptions for the stemmers from the source provided. Secondly, Porter (2001) acknowledges the failure to promote the exact implementation of the original Porter stemming algorithm, leading to compounded problems in interpretation, leading to misunderstandings regarding the meaning and structure of the original algorithm. Thus, he presents the Snowball Framework. The Snowball Framework is discussed briefly in section 1.2.3, so we will skip over this part.

The Porter2 stemming algorithm is the implementation of the improved version of the original Porter stemming algorithm, using the Snowball Framework. The changes weren't too extensive as further changes were leading to rather high levels of complexity with a very minor performance improvement (Porter 2002).

Singh & Gupta (2018) discussed several other rule-based stemmers developed between the development of the above-mentioned stemming algorithms.

Dawson (1974) proposed the Dawson Stemmer in 1974 is a fast single-pass suffix removal algorithm, with a similar *longest match* principle and an exhaustive list of around 1200 suffixes.

Other than that there are a couple of other notable stemming algorithms in literature such as the Krovetz Stemmer (Krovetz 1993) and the XEROX stemmer (Hull 1996).

CHAPTER 3

ANALYSIS

CHAPTER THREE: ANALYSIS

Some of the stats we observed concerning the size of the dataset at different stages of processing:

Total number of tokens	1161192
Unique tokens	56057
Tokens in Primary Dataset	42551
Tokens in Sample Dataset	4256
Primary Dataset after Porter	26202
Primary Dataset after Lancaster	21361
Primary Dataset after Snowball	25797

Percentage reduction in the size of the primary dataset with the application of each algorithm:

Algorithm	% reduced
Porter	38.42 %
Lancaster	49.80 %
Snowball	39.37 %

It is observable that the Lancaster stemming algorithm provides the most reduction in the list of terms to be indexed, but at what cost we will see with some examples in the coming sections.

As expected during observation of the stemmed words, we encountered the most commonly occurring issues with stemming - over-stemming and under-stemming.

3.1 Porter Stemmer

Porter stemmer faces the same issues that are faced by any rule-based stemming algorithms - the imperfect roots obtained after processing. This is an outcome of the very nature of stemming, which is simply a process of stripping down inflectional word endings and conflating similar words under a common stem. Thus, we receive results such as:-

Stem	Words
agreeabl	['agreeable', 'agreeableness', 'agreeably']
agricultur	['agriculture', 'agricultural', 'agricultures', 'agriculturally']
authent	['authentic', 'authenticated', 'authenticate', 'authentication', 'authenticator', 'authentications', 'authentically', 'authenticity']
biblic	['biblically']
cathol	['catholics']
celebr	['celebrates']
decis	['decisive', 'decisions', 'decisiveness']
effectu	['effectual']
fritzi	['fritzie']
gantri	['gantry']

Though we have tried to use selective examples to show the varied spelling issues we receive in output stems, the most commonly observed issue was the deletion of the 'e' or missing in at the end of stems when they should be there and the presence of 'i' at the end of the stem where there should be a 'y'.

However, neither of them is that major of an issue when we take into account the primary purpose of a stemming algorithm, even from a readability perspective this isn't that serious of an issue. Let's look at a few examples.

Stems	Words

galleri	['gallery']
gantri	['gantry']
futhermor	['futhermore']
hegemoni	['hegemony']
imbecil	['imbecile']

The whole processed dataset is filled with such examples.

Still, even though several spelling issues were observed, most of the spelling observed were acceptable and even surprisingly accurate. This was the case for shorter, simpler, and most commonly used word endings like 'ing' and 's'.

Over-stemming

Over-stemming is one of the major issues that come up while developing a stemming algorithm. It happens when the algorithm puts words that were supposed to come under distinct stems under a common stem. Let's have a look at some of the observations.

Stem	Words
access	['accessible', 'accesses', '***accessions***', 'access', 'accessibility']
activ	['activated', 'activate', '**active**', 'activating', '***activism***', 'activation', '**activities**', 'actives', '**actively**', '**activity**']
bar	['barred', 'barring', '***bar***', '***bars***']
book	['**booked**', '**booking**', '*book*', '**bookings**', 'books']
cast	['**caste**', 'casting', 'casts', 'cast']
commun	['communicated', '**commune**', 'communicate', 'communicative', '**communize**', 'communicating', 'communicational', '**communal**', '***communism***', 'communication', 'communicator', 'communities', '**communes**', '***communisms***', 'communications', 'communicators', '**communitys**', '**community**']

expedit	['**expediting**', 'expeditions']
gener	['**generate**', '***general***', 'generations']
univers	['***universe***', 'universalize', 'universal', '**universities**', 'universals', '**universitys**', 'universally', 'universality', '**university**']
wit	['witnessed', '**witted**', 'witnessing', 'witnesses', 'witness', '**wits**', '**wit**']

As we can observe, stemming doesn't really take into consideration the context of the word, and simply acts on the basis of pre-fed spellings and word endings, thus giving us results as shown in the table above.

'*activism'*, '*activation*', and '*activity*' all should come under different stems as they are semantically different, but due to their similarity in spelling, they are reduced to the same stem '***activ***'. Similarly, we can look at the most commonly used example for over-stemming where the words like '*universe*', '*universality*', and '*university*', which should all come under different stems semantically are reduced to a common stem '***univers***'.

Under-stemming

Another major issue faced during the development of stemming algorithms is Under-stemming. This is when the stemming algorithm fails to concord words that are supposed to come under the same stem, and rather assigns different stems to them

Stem	**Word**
absorb	['absorb', 'absorbed', 'absorbing', 'absorber', 'absorbs', 'absorbent', 'absorbency']
absorpt	['absorptive', 'absorption', 'absorptions']
absurd	['absurd', 'absurdities', 'absurdity']
absurdli	['absurdly']
ambit	['ambition', 'ambitions']

ambiti	['ambitious', 'ambitiously']
bind	['bind', 'binding', 'binds']
binder	['binder', 'binders']
concur	['concurred', 'concur', 'concurs']
concurr	['concurrence', 'concurrent', 'concurrently']
creat	['created', 'create', 'creating', 'creates']
creation	['creation', 'creations']
creativ	['creative', 'creativeness', 'creatively', 'creativity']
creator	['creator', 'creators']
dynast	['dynastic', 'dynasts']
dynasti	['dynasties', 'dynasty']
withdraw	['withdrawing', 'withdrawal', 'withdraw']
withdrawn	['withdrawn']
withdrew	['withdrew']

Under-stemming can be for various reasons, one of which is that the stemming algorithms are designed while keeping in mind primarily the inflectional suffixes, thus we can see quite an acceptable level of accuracy in the output of stemming algorithms, but the same cannot be said for the derivational word endings.

We have selected a very small set of observations to show what kind of stems are produced due to under-stemming. A common type would be the words ending with ‘y’ being reduced simply to stems with an ending ‘i’ (except if it ends in ‘ently’, ‘ously’ and other few such cases). Examples of both can be seen in the observations, the case of ‘***dynast***’ / ‘***dynasti***’ and ‘***absurd***’ / ‘***absurdli***’ for the mentioned case, and ‘***ambit***’/ ‘***ambiti***’ and ‘***concur***’ / ‘***concurr***’ for the exception.

The Porter stemmer also in certain cases doesn’t address the double consonants in stem endings leading to under-stemming, ‘***concur***’ and ‘***concur***’ would be an example of the said situation.

3.2 Snowball (Porter2)

Porter2 implemented in the snowball framework is supposed to be a slightly improved version of the Porter stemming algorithm. So, as one would expect the outputs it produced are observed to be identical to that of the Porter Stemming algorithm for the majority of the cases, of course with a few exceptions, which resolve some of the issues of the Porter stemming algorithm we discussed in the previous section.

Since it is an improved version of the Porter stemmer, we'll first compare our observations of both the Porter algorithms them we'll compare them we'll look at the Lancaster stemmer and compare it with the Porter algorithms.

In the example given below, we can see the improvement Porter2 has over the Porter stemmer. This was a stem that we had highlighted in the previous section where we gave examples of over-stemming for Porter stemmer.

Words like *communicate*, *communicated*, *communications*, etc. should come under one stem, and words like *communism*, *communisms*, etc. should come under another, finally words like *community*, *communities*, *communities*, etc. should come under different stems, which have been resolved to some extent in Porter2.

Porter	Original	Lancaster	Snowball
	['communicated',	['commun',	['communic',
	'commune',	'commun',	'commune',
	'communicate',	'commun',	'communic',
	'communicative',	'commun',	'communic',
	'communize',	'commun',	'communiz',
	'communicating',	'commun',	'communic',
	'communicational',	'commun',	'communic',
	'communal',	'commun',	'communal',
	'communism',	'commun',	'communism',
	'communication',	'commun',	'communic',
	'communicator',	'commun',	'communic',
	'communities',	'commun',	'communiti',
	'communes',	'commun',	'commune',
	'communisms',	'commun',	'communism',
	'communications',	'commun',	'communic',
	'communicators',	'commun',	'communic',
	'communitys',	'commun',	'communiti',
commun	'community']	'commun']	'communiti']

We can take a look at one more example where the Porter2 resolves an over-stemming issue.

Original	Porter	Snowball
generator	gener	generat
generators	gener	generat
generous	gener	generous
generously	gener	generous

Now, let's take a look at an observation showing a resolution of an instance of under-stemming, where the words '*hesitantly*' and '*hesitatingly*' which should have been put under a common stem, were processed to different stems by the Porter stemmer, which on the

other hand were given a common stem '***hesit***' by the Porter2 (snowball) stemmer, that even though is not an accurate word in English but for information retrieval, indexing and other such purposes is more than sufficient.

Original	Porter	Snowball
hesitantly	hesitantli	hesit
hesitatingly	hesitatingli	hesit

Another issue that was resolved in the development of the Porter2 stemmer was the stem endings with 'i' if the word ended with 'y' and other instances where the stem ended with 'i' when it shouldn't have. Let's look at a few examples.

Original	Porter	Snowball
heartedly	heartedli	heart
heatedly	heatedli	heat
heavenly	heavenli	heaven
inwardly	inwardli	inward
palliative	palli	palliat
secretly	secretli	secret

Partly this could be attributed to proper removal of the 'ly' word-ending.

Another issue resolved is the improper removal of the 's' suffix. We can better understand it by looking at a few examples.

Original	Porter	Snowball
fractious	fractiou	fractious
impetus	impetu	impetus
lassus	lassu	lassus

modus	modu	modus
tortuous	tortuou	tortuous
vacuous	vacuou	vacuous

The Porter stemmer aggressively removes the ‘s’ at word endings, even when they aren’t supposed to be removed, leading to the production of improper stems.

3.3 Lancaster Stemmer

The very first observation is the shorter stem lengths. Lancaster stemmer uses a more aggressive approach thus producing comparatively smaller stems, but at the cost of over-stemming.

Algorithm	Stem Length
Original	7.754858875
Porter	6.345867312
Lancaster	5.534323518
Snowball	6.321825574

Let’s have a look at some over-stemming examples (the chart is on the next page), we can see that Lancaster stemmer produces smaller stems, this could be helpful in certain situations producing more accurate stems than the Porter stemmer, which was the most influential stemmer when Lancaster stemmer was introduced. However, more often than not this leads to highly inaccurate results, which are often hilarious. Just look at the stem ‘***cult***’ mentioned in the table below and you will understand.

Lancaster	Original	Porter	Snowball
ab	['ab', 'abed', 'abated', 'abe', 'abbe', 'abaringe', 'aber', 'abberations']	['ab', 'abe', 'abat', 'abe', 'abb', 'abaring', 'aber', 'abber']	['ab', 'abe', 'abat', 'abe', 'abb', 'abaring', 'aber', 'abber']

Lancaster	Original	Porter	Snowball
act	['activated', 'acted', 'activate', 'active', 'activating', 'acting', 'actual', 'activism', 'activation', 'action', 'actor', 'actualities', 'activities', 'actives', 'actions', 'actors', 'acts', 'act', 'actively', 'actually', 'actuality', 'activity']	['activ', 'act', 'activ', 'activ', 'activ', 'act', 'actual', 'activ', 'activ', 'action', 'actor', 'actual', 'activ', 'activ', 'action', 'actor', 'act', 'act', 'activ', 'actual', 'actual', 'activ']	['activ', 'act', 'activ', 'activ', 'activ', 'act', 'actual', 'activ', 'activ', 'action', 'actor', 'actual', 'activ', 'activ', 'action', 'actor', 'act', 'act', 'activ', 'actual', 'actual', 'activ']
ad	['ada', 'ad', 'add', 'added', 'adored', 'ade', 'adage', 'adorable', 'adore', 'adding', 'adulation', 'ads', 'adds', 'adores']	['ada', 'ad', 'add', 'ad', 'ador', 'ade', 'adag', 'ador', 'ador', 'ad', 'adul', 'ad', 'add', 'ador']	['ada', 'ad', 'add', 'ad', 'ador', 'ade', 'adag', 'ador', 'ador', 'ad', 'adul', 'ad', 'add', 'ador']
cult	['cultured', 'cultivated', 'culture', 'cultivate', 'culte', 'cultivating', 'cultural', 'cultivation', 'cultures', 'cultivates', 'cults', 'cult', 'cultist', 'culturally']	['cultur', 'cultiv', 'cultur', 'cultiv', 'cult', 'cultiv', 'cultur', 'cultiv', 'cultur', 'cultiv', 'cult', 'cult', 'cultist', 'cultur']	['cultur', 'cultiv', 'cultur', 'cultiv', 'cult', 'cultiv', 'cultur', 'cultiv', 'cultur', 'cultiv', 'cult', 'cult', 'cultist', 'cultur']
cur	['curia', 'cured', 'cure', 'curative', 'curing', 'curator', 'cur', 'currencies', 'cures', 'currants', 'currents', 'currant', 'current', 'currency', 'curly', 'currently']	['curia', 'cure', 'cure', 'cur', 'cure', 'curat', 'cur', 'currenc', 'cure', 'currant', 'current', 'currant', 'current', 'currenc', 'curli', 'current']	['curia', 'cure', 'cure', 'curat', 'cure', 'curat', 'cur', 'currenc', 'cure', 'currant', 'current', 'currant', 'current', 'currenc', 'cur', 'current']
den	['denatured', 'dene', 'denial', 'den', 'dennis', 'denials', 'dens']	['denatur', 'dene', 'denial', 'den', 'denni', 'denial', 'den']	['denatur', 'dene', 'denial', 'den', 'denni', 'denial', 'den']

Lancaster	Original	Porter	Snowball
depart	['departed', 'departure', 'departing', 'departmental', 'departures', 'departments', 'departs', 'department', 'depart']	['depart', 'departur', 'depart', 'department', 'departur', 'depart', 'depart', 'depart', 'depart']	['depart', 'departur', 'depart', 'department', 'departur', 'depart', 'depart', 'depart', 'depart']
wint	['winced', 'wintered', 'wincing', 'wintering', 'winter', 'winters']	['winc', 'winter', 'winc', 'winter', 'winter', 'winter']	['winc', 'winter', 'winc', 'winter', 'winter', 'winter']

Though, this more aggressive approach leads to a high-degree of over-stemming in several cases. In some cases, it produces better results than the Porter stemmer, as we can see in the examples below.

Lancaster	Original	Porter	Snowball
absorb	['absorb', 'absorbed', 'absorptive', 'absorbing', 'absorption', 'absorber', 'absorbs', 'absorptions', 'absorbent', 'absorbency']	['absorb', 'absorb', 'absorpt', 'absorb', 'absorpt', 'absorb', 'absorb', 'absorpt', 'absorb', 'absorb']	['absorb', 'absorb', 'absorpt', 'absorb', 'absorpt', 'absorb', 'absorb', 'absorpt', 'absorb', 'absorb']

abstract	['abstracted', 'abstractive', 'abstracting', 'abstractionism', 'abstraction', 'abstractions', 'abstractors', 'abstractedness', 'abstracts', 'abstractionists', 'abstract', 'abstractly']	['abstract', 'abstract', 'abstract', 'abstraction', 'abstract', 'abstract', 'abstractor', 'abstracted', 'abstract', 'abstractionist', 'abstract', 'abstractli']	['abstract', 'abstract', 'abstract', 'abstraction', 'abstract', 'abstract', 'abstractor', 'abstracted', 'abstract', 'abstractionist', 'abstract', 'abstract']

Both the Porrter stemmers are producing multiple stems for words which should be conflated to a single stem, which is not the case for the Lancaster stemmer.

Another case where the Lancaster stemmer seems to shine over the original Porter stemmer is the case with words having ‘ly’ as word endings.

Lancaster	**Original**	**Porter**	**Snowball**
abund	abundantly	abundantli	abund
accus	accusingly	accusingly	accus
yearn	yearningly	yearningli	yearn

In this aspect the Lancaster stemmer has comparable performance to the Porter2 stemmer, which was developed almost a decade later.

CHAPTER 4

CONCLUSION

CHAPTER FOUR: CONCLUSION

A clear difference in accuracy can be observed between both the Porter Stemmers and the Lancaster Stemmer. The aggressive approach of Lancaster Stemmer does indeed in some instances show better performance than the Porter Stemmer, however, at the cost of falling in short in various other areas.

Snowball aka Porter2 stemmer resolves some of the most commonly occurring problems of the Porter Stemming algorithm, but as observed the outputs are still far from perfect, but then again, one does not use the stemming algorithms while having the goal of receiving perfect outputs in mind.

We have identified issues of over-stemming and under-stemming, but most of the remaining issues are the ones that would require taking context into mind, and others are the ones that are the outcome of spelling exceptions in English. To address those issues one might need to write rules while focusing on every one of them individually.

Whether one would take such an endeavor or not would completely depend on the utility of such endeavor, the question - “How much more work would be required to improve the performance by a fraction of a percent?”, would be at the core of such an endeavor and the one’s taking up this work must consider this very question.

References

Cao, P. (2022, November). Identification and Distinction of Root, Stem and Base in English Linguistics Teaching. In *2nd International Conference on Education: Current Issues and Digital Technologies (ICECIDT 2022)* (pp. 698-706). Atlantis Press.

Çelikkol, P., Laubrock, J., & Schlangen, D. (2023, May). TF-IDF based Scene-Object Relations Correlate With Visual Attention. In *Proceedings of the 2023 Symposium on Eye Tracking Research and Applications* (pp. 1-6).

Dawson, J. (1974). Suffix removal and word conflation. *ALLC bulletin*, *2*(3), 33-46.

Hull, D. A. (1996). Stemming algorithms: A case study for detailed evaluation. *Journal of the American Society for Information Science*, *47*(1), 70-84.

Jivani, A. G. (2011). A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, *2*(6), 1930-1938.

Krovetz, R. (1993, July). Viewing morphology as an inference process. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 191-202).

Larkey, L. S., Ballesteros, L., & Connell, M. E. (2007). Light stemming for Arabic information retrieval. In *Arabic computational morphology: knowledge-based and empirical methods* (pp. 221-243). Dordrecht: Springer Netherlands.

Lovins, J. B. (1968). Development of a stemming algorithm. *Mech. Transl. Comput. Linguistics*, *11*(1-2), 22-31.

Manning, C. D. (2009). *An introduction to information retrieval*. Cambridge university press.

Moral, C., de Antonio, A., Imbert, R., & Ramírez, J. (2014). A survey of stemming algorithms in information retrieval. *Information Research: An International Electronic Journal*, *19*(1), n1.

Paice, C. D. (1990, November). Another stemmer. In *ACM Sigir Forum* (Vol. 24, No. 3, pp. 56-61). New York, NY, USA: ACM.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, *14*(3), 130-137.

Porter, M. F. (2001). Snowball: A language for stemming algorithms.

Porter, M. F. (2002). The English (Porter2) stemming algorithm. Snowball. http://snowball.tartarus.org/algorithms/english/stemmer.html

Porter, M. F. (2005). Lovins revisited. In *Charting a New Course: Natural Language Processing and Information Retrieval: Essays in Honour of Karen Spärck Jones* (pp. 39-68). Dordrecht: Springer Netherlands.

Radu, R. G., Rădulescu, I. M., Truică, C. O., Apostol, E. S., & Mocanu, M. (2020, May). Clustering documents using the document to vector model for dimensionality reduction. In *2020 ieee international conference on automation, quality and testing, robotics (aqtr)* (pp. 1-6). IEEE.

Singh, J., & Gupta, V. (2017). A systematic review of text stemming techniques. *Artificial Intelligence Review*, *48*, 157-217.

Voos, H. (1971). The Information Explosion; or, Redundancy Reduces the Charge!. *College & Research Libraries*, *32*(1), 7-14.

Willett, P. (2006). The Porter stemming algorithm: then and now. *Program*, *40*(3), 219-223.

APPENDIX

Appendix A: Data Analysis Notebook

Python Notebook used for processing data.

Here is a Google Colaboratory link for the notebook:
https://drive.google.com/file/d/1hd6Rdk3VBEThi-kJy5KdiRbfRnBGTHUZ/view?usp=sharing

Data Analysis for Masters Thesis 1

Title: Stemming algorithms for English

Algorithms being studied

- Porter
- Lancaster
- Snowball (Porter2)

Importing Required Packages

```
from nltk.stem import PorterStemmer, LancasterStemmer, snowball
from nltk.corpus import brown
import pandas as pd
```

Extracting words from the corpora and preparing them for further processing

```
words = brown.words()
corpus = ' '.join(words)

print("Total number of words in the corpus: ", len(words))
print("Total number of unique tokens: ", len(set(words)))
```

```
Total number of words in the corpus:  1161192
Total number of unique tokens:  56057
```

```
corpus = corpus.lower()
corpus = corpus.replace("\n"," ")
corpus = corpus.replace("-", " ")
cleaned_corpus = [i for i in corpus if i.isalpha() or i==' ']
cleaned_corpus = ''.join(cleaned_corpus)
rev_corpus = [i[::-1] for i in cleaned_corpus]
```

Removing functional words and sorting according to the reverse of the spelling so as it order it according to the word endings.

```
functional_words = [
    "the", "a", "an",
    "in", "on", "at", "by", "for", "with", "to", "from", "of", "about",
"through", "between", "among", "under", "over",
    "and", "but", "or", "nor", "for", "so", "yet",
    "although", "because", "if", "unless", "since", "while", "when",
```

```
"after", "before", "as", "though",
    "I", "you", "he", "she", "it", "we", "they", "me", "you", "him",
"her", "us", "them",
    "my", "your", "his", "her", "its", "our", "their", "mine", "yours",
"hers", "ours", "theirs",
    "myself", "yourself", "himself", "herself", "itself", "ourselves",
"yourselves", "themselves",
    "who", "whom", "whose", "which", "that",
    "this", "that", "these", "those",
    "who", "whom", "whose", "which", "what",
    "all", "another", "any", "anybody", "anyone", "anything", "both",
"each", "either", "everybody", "everyone", "everything", "neither",
"nobody", "no one", "nothing", "several", "some", "somebody", "someone",
"something",
    "can", "could", "may", "might", "must", "shall", "should", "will",
"would",
    "am", "is", "are", "was", "were", "be", "being", "been", "have",
"has", "had", "do", "does", "did",
    "also", "not", "never", "always", "very", "too", "so", "such", "here",
"there", "now", "then", "when", "where",
    "today", "yesterday", "tomorrow", "soon", "now", "then", "already",
"lately",
    "always", "usually", "often", "sometimes", "seldom", "never"
]

words = cleaned_corpus.split()

primary_dataset = [i for i in words if not i in functional_words]
primary_dataset = list(set(primary_dataset))
primary_dataset = [i[::-1] for i in primary_dataset]
primary_dataset.sort()
primary_dataset = [i[::-1] for i in primary_dataset]

print("Number of words in the primary dataset: ", len(primary_dataset))

Number of words in the primary dataset:  42551
```

Applying systematic sampling (selecting every 10th word) total of 10% of the dataset.

```
sample_systematic = []

for i in range(0, len(primary_dataset), 10):
    sample_systematic.append(primary_dataset[i])

print("Sample size: ", len(sample_systematic))
print("Sample preview: ", sample_systematic[0:30])

Sample size:  4256
Sample preview:  ['aa', 'elba', 'tuba', 'jamaica', 'veronica',
'atlantica', 'dellarca', 'ywca', 'hedda', 'salida', 'veranda', 'tenda',
'soda', 'medea', 'anthea', 'andrea', 'hoffa', 'bottega', 'ticonderoga',
'mischa', 'pasha', 'bertha', 'suburbia', 'acadia', 'pharmacopoeia',
'bahia', 'malia', 'anglia', 'julia', 'lumia']
```

Creating objects for each algorithm

```
stemmer_porter = PorterStemmer()
stemmer_lancaster = LancasterStemmer()
stemmer_snowball = snowball.EnglishStemmer()
```

Applying stemming over the primary dataset and the sample dataset

```
def stemAll(dataset):
    stemmed_lancaster = [stemmer_lancaster.stem(i) for i in dataset]
    stemmed_porter = [stemmer_porter.stem(i) for i in dataset]
    stemmed_snowball = [stemmer_snowball.stem(i) for i in dataset]

    df_processed = pd.DataFrame({
    "Original" : dataset,
    "Porter" : stemmed_porter,
    "Lancaster" : stemmed_lancaster,
    "Snowball" : stemmed_snowball,
    }, index=range(1,len(dataset)+1))

    return df_processed

df_primary = stemAll(primary_dataset)
df_sample_systematic = stemAll(sample_systematic)
print(df_primary.head())
print()
print(df_sample_systematic.head())
```

```
  Original   Porter Lancaster Snowball
1       aa       aa        aa       aa
2      aaa      aaa        aa      aaa
3       ba       ba        ba       ba
4  barnaba  barnaba    barnab  barnaba
5     paba     paba       pab     paba

   Original    Porter Lancaster  Snowball
1        aa        aa        aa        aa
2      elba      elba       elb      elba
3      tuba      tuba       tub      tuba
4   jamaica   jamaica    jamaic   jamaica
5  veronica  veronica   veronic  veronica
```

Average lenght of Original words and stems produced

```
len_original = len_porter = len_lancaster = len_snowball = 0

len_primary = len(df_primary)

for i in range(1, len_primary+1):
    len_original += len(df_primary['Original'][i])
    len_porter += len(df_primary['Porter'][i])
    len_lancaster += len(df_primary['Lancaster'][i])
```

```
    len_snowball += len(df_primary['Snowball'][i])

print(f"Original: {len_original/len_primary}\nPorter:
{len_porter/len_primary}\nLancaster:
{len_lancaster/len_primary}\nSnowball: {len_snowball/len_primary}")
```

```
Original: 7.754858875232075
Porter: 6.345867312166576
Lancaster: 5.53432351766116
Snowball: 6.321825574017062
```

Grouping outputs on the basis of stem produced by each algorithm

```
# Grouping by porter
group_porter_primary = df_primary.groupby("Porter").agg(lambda x:
x.tolist())
group_porter_sample_systematic =
df_sample_systematic.groupby("Porter").agg(lambda x: x.tolist())

# Grouping by lancaster
group_lancaster_primary = df_primary.groupby("Lancaster").agg(lambda x:
x.tolist())
group_lancaster_sample_systematic =
df_sample_systematic.groupby("Lancaster").agg(lambda x: x.tolist())

# Grouping by snowball
group_snowball_primary = df_primary.groupby("Snowball").agg(lambda x:
x.tolist())
group_snowball_sample_systematic =
df_sample_systematic.groupby("Snowball").agg(lambda x: x.tolist())
```

Words for which all algorithms produce a common stem

```
common_stems_primary = df_primary[(df_primary["Lancaster"] ==
df_primary["Porter"]) & (df_primary["Porter"] ==
df_primary["Snowball"])].reset_index()
common_stems_sample_sys =
df_sample_systematic[(df_sample_systematic["Lancaster"] ==
df_sample_systematic["Porter"]) & (df_sample_systematic["Porter"] ==
df_sample_systematic["Snowball"])].reset_index()
common_stems_sample_sys
```

	index	Original	Porter	Lancaster	Snowball
0	1	aa	aa	aa	aa
1	73	spa	spa	spa	spa
2	104	cab	cab	cab	cab
3	105	grab	grab	grab	grab
4	107	caleb	caleb	caleb	caleb
...	...	...	...	...	...
2381	4252	merz	merz	merz	merz
2382	4253	livshitz	livshitz	livshitz	livshitz
2383	4254	markovitz	markovitz	markovitz	markovitz
2384	4255	schwartz	schwartz	schwartz	schwartz

```
2385   4256       fuzz       fuzz       fuzz       fuzz

[2386 rows x 5 columns]
```

Eliminating words which are not stemmed

```
common_stems_primary_stemmed =
common_stems_primary[common_stems_primary["Original"] !=
common_stems_primary["Porter"]].reset_index()
common_stems_sys_stemmed =
common_stems_primary[common_stems_primary["Original"] !=
common_stems_primary["Porter"]].reset_index()
common_stems_primary_stemmed =
common_stems_primary_stemmed.drop(["level_0", "index"], axis = 1)
common_stems_sys_stemmed = common_stems_sys_stemmed.drop(["level_0",
"index"], axis = 1)
```

Reduced size of the dataset after application of algorithms

```
print(f"Original: {len(primary_dataset)}\nPorter:
{len(group_porter_primary)}\nLancaster:
{len(group_lancaster_primary)}\nSnowball: {len(group_snowball_primary)}")
```

```
Original: 42551
Porter: 26202
Lancaster: 21361
Snowball: 25797
```

Where output from snowball varies from Porter

```
porter_vs_snowball = df_primary[df_primary["Porter"] !=
df_primary["Snowball"]].reset_index()
porter_vs_snowball = porter_vs_snowball.drop("Lancaster", axis=1)
```

Writing all of this to the output files

```
df_primary.to_csv("./outputs/primary_dataset.csv")
df_sample_systematic.to_csv("./outputs/sample_dataset.csv")
group_porter_primary.to_csv("./outputs/group_porter_primary.csv")
group_porter_sample_systematic.to_csv("./outputs/group_porter_sample.csv")
group_lancaster_primary.to_csv("./outputs/group_lancaster_primary.csv")
group_lancaster_sample_systematic.to_csv("./outputs/group_lancaster_sample
.csv")
group_snowball_primary.to_csv("./outputs/group_snowball_primary.csv")
group_snowball_sample_systematic.to_csv("./outputs/group_snowball_sample.c
sv")
common_stems_primary.to_csv("./outputs/common_stems_primary.csv")
common_stems_sample_sys.to_csv("./outputs/common_stems_sample.csv")
common_stems_primary_stemmed.to_csv("./outputs/common_stems_primary_stemme
d.csv")
common_stems_sys_stemmed.to_csv("./outputs/common_stems_sample_stemmed.csv
```

```
")
porter_vs_snowball.to_csv("./outputs/porter_vs_snowball.csv")
```

Appendix B: Dataset and Processed Data

The Dataset Used for this Study is the Brown Corpus, which was accessed from NLTKs collection of Corpora.

The size of the Dataset as well as the Processed Data is a little bit too large to put in the appendix, so I have uploaded it to Google Drive and added the link to all the processed data here.

The link:
https://drive.google.com/drive/folders/1pQxQjYYUZB9XQWWSDiGFcwOp2i09MVyS?usp=sharing