CMPE 492

Developing a Rule-based Turkish Stemmer with Dictionary

Musa Şimşek

Meriç Keskin

Advisor:

Tunga Güngör

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Broad Impact

In recent years, the field of computational linguistics has witnessed significant advancements, particularly in the development of language-specific stemmers. The creation of a Turkish stemmer, the primary focus of this project, holds substantial implications both for the Turkish language and for the broader spectrum of natural language processing (NLP) and information retrieval (IR). Turkish is an agglutinative language which presents unique challenges due to its complex morphology. By considering these challenges, this project aims to contribute to the enhancement of text analysis, search algorithms, and data processing in Turkish, thereby impacting sectors ranging from academic research to industry applications.

The project aligns with global trends in linguistics and computer science, where the emphasis is the developing tools to address to a diverse array of languages. This practice ensures that lesser-researched languages, like Turkish, receive attention, leading to technological advancements that are equitable and universal. The development of a Turkish stemmer is expected to promote linguistic diversity in the digital realm.

1.2. Ethical Considerations

In the creation of the Turkish stemmer, considerations for the methods are paramount. The algorithm must be designed determined and inclusively as it must handle Turkish language as a whole with the least amount of exceptions. Therefore, we need detailed research for language processing and interpretation methods, existing language stemmer designs and analysis on attempts to process Turkish language. Furthermore, the project will adhere to open-source principles to promote transparency, collaboration, and accessibility in the field of computational linguistics.

2. PROJECT DEFINITION AND PLANNING

2.1. Project Definition

The project is dedicated to developing a Turkish stemmer, a computational tool that reduces Turkish words to their base or root forms. The stemmer will take into account the unique morphological features of the Turkish language, which is characterized by heavy affixation. The aim is to enhance the accuracy and efficiency of information retrieval and text processing in Turkish, thereby making better search algorithms, text analysis, and data processing possible.

2.2. Project Planning

2.2.1. Project Time and Resource Estimation

The project is planned to span several months, considering the complexity of Turkish morphology. It will involve stages of research, algorithm development, testing, and refinement. The necessary resources include access to comprehensive dictionary of Turkish roots and bases, computational tools for developing the stemmer algorithm, and the widely used benchmarks to be able to test the stemmer formally.

2.2.2. Success Criteria

There are different evaluation criteria in the literature that can be applied to a stemmer. As our project aims to develop a language-specific stemmer for Turkish, the universal IR benchmarks are not applicable here. However, several datasets are available for Turkish such as TTC-3600 [Kılınç et al., 2017] and the one used in the article *Information Retrieval on Turkish Texts* [Can et al., 2008]. From these datasets, we can obtain the information retrieval success of our stemmer from metrics like precision, recall, etc. While applying those benchmarks, computational resources also can be measured to compare our project to other widely-known stemmers. In addition to these, we can develop a small dataset on our own to evaluate our stemmer, like Giorgos Tsiftsis et al. did [Tsiftsis et al., 2014], in order to see the accuracy of our stemmer on separating suffixes from each other and from the stem.

2.2.3. Risk Analysis

Potential risks include the complexity of Turkish morphology leading to challenges in algorithm development, the possibility of unforeseen challenges in the algorithm, and the need to constantly update the stemmer in response to linguistic changes and advancements in the field. The process of developing the project will include continuous testing, real-time feedback, and iterative improvements.

2.2.4. Team Work

The project will be handled collaboratively, as we separately work on different aspects of computational linguistics, Turkish language studies and software development. We will use regular meetings for our communication process and a shared project management platform (GitHub) with a shared cloud space (Google Drive) to coordinate and track our efforts.

3. RELATED WORK

3.1. Inflection vs Derivation

The affixes in any language are divided primarily into two classes: inflection and derivation. However, the differences between inflections and derivations are not easy to confirm always. Therefore, we started with a book that suggests different ways of distinguishing an inflection from a derivation.

The seventh chapter of the book *Functional Structure in Morphology and the Case of Nonfinite Verbs* [Nielsen, 2016] is based on the discussions of inflections and derivations since it may be necessary to know for some applications whether the discarded part of the word has an inflectional or derivational impact on the stemmed word. For this purpose, the author gives several definitions of inflection and derivation. Here are some of them:

- Intuitively, inflection should affect the physical characteristics of an object, whereas derivation creates a different thing from the object.
- Oxford English Dictionary defines inflection as "the modification of the form of the word to express different grammatical relations into which it may enter" and derivation as "formation of a word from a more primitive word or root in the same or another language".

After the definitions, the author proposes some criteria to distinguish inflection from derivation:

- Inflection is a grammatical phenomenon, derivation is lexical. As an example, "spilled" is an inflectional change on the word "spill" but "spillage" is a derivational change.

- Inflection creates different forms from the same stem, while derivation creates new stems.
- Inflection does not change the category of the word; however, derivation does. The word "foolish" is not under the same category as "fool"; therefore, "ish" suffix is a derivation.
- Inflation is semantically regular: the meaning of the inflected word is fully predictable. Derivation is the opposite.
- Derivation is less restricted in terms of the set of morphemes and more open to inclusion of new derivatives.

Even though Turkish does not have such discussions since it is easy to find out which suffixes are inflections and which are derivations, this book helped us to see one of the general challenges of developing a stemmer.

3.2. Porter Stemmer

After we finished the discussion of differences between inflections and derivations, we investigated the methodologies applied to the most widely known stemmers. Basically, there are three types of stemmers in the literature: truncating, statistical, and mixed approaches. Truncating stemmers slice the words and convert them to fixed-length words. They rely on language-specific rules most generally. Statistical stemmers, on the other hand, are based on analytical approaches and apply mathematical approximations to get the stemmed words. Finally, the mixed approach section includes several techniques like corpus-based statistical stemmer, and inflectional/derivational stemmers, etc.

We started to learn the principles of different types of stemmers with rule-based approach. The first stemmer is one of the most widely known truncating(rule-based) stemmer: Porter's Stemmer.

In this article, the author describes the stemming process briefly and explains the algorithm of Porter's stemmer [Mawlood-Yunis, 2013].

The author states that a stemmer is usually expected to remove only inflectional morphemes. However, Porter's algorithm does not satisfy this requirement because it assumes that there is no stem dictionary. The reason behind that decision is that the most important purpose of stemming is to increase the information retrieval performance, and adding the dictionary would slow the process. The author states that the success rate for the proper suffix stripping will be significantly less than 100 percent for rule-based approaches because of the absence of a control mechanism that ensures the stemmed word is an actual word.

The article continues with the details of the algorithm of Porter's stemmer. A word is a combination of consonants and vowels(vowels are A, E, I, O, U, and Y, Y is a vowel if it is preceded by a consonant). Based on this definition and some derivations, Porter realized that any word can be denoted as

$$\mathbf{[C][VC]m[V]}$$

where C is a list of consonants and V is a list of vowels. The [C] and [V] parts at both ends are optional, and the (VC) part in the middle is repeated m times.

After this, Porter starts to define the rules for removing a suffix from a word in the following form:

$$\mathbf{(condition)\ S1 \rightarrow S2}$$

If a word ends with suffix S1 and the stem before S1 satisfies the *condition*, the algorithm replaces S1 with S2. The conditions can be given in terms of the value of m, a regex, or both together. It should be noted that in a set of rules that matches with the word, only one is obeyed at each step, and that will be the rule with the longest matching S1.

3.3. The Porter Stemming Algorithm: Then and Now

The article titled "The Porter stemming algorithm: then and now" [Willett, 2006] provides an extensive review of the Porter stemming algorithm, which has been a foundational method in information retrieval and linguistic research since its inception in 1980. The algorithm, developed by Martin F. Porter, is designed to reduce words to their base or root form, thereby improving the efficiency and effectiveness of information retrieval processes. The paper focuses on the algorithm's original features, its significant role in information retrieval research, and its extension to a variety of languages beyond English.

Willett emphasizes that the algorithm's original appeal lay in its simplicity and effectiveness, which contrasts with earlier, more complex stemming approaches like the one by Lovins, which had a large dictionary of suffixes and numerous context-sensitive rules. The Porter algorithm streamlined this process, reducing the complexity of the rules associated with suffix removal and employing a single, unified approach to handle context by using a measure based on the number of consonant-vowel-consonant sequences, which provides a computable representation of syllables.

The algorithm has been adapted for a wide range of languages, and this adaptability highlights its second major contribution: stimulating further research into stemming as an important topic in its own right. The algorithm's continuous relevance and utility are evidenced by the numerous citations it has received over the years, not just in the field of information science but also in a broader computer science context, showcasing its far-reaching impact on data and knowledge processing. The paper concludes by underscoring the enduring significance of the Porter algorithm in the field of information retrieval and its continued relevance in modern applications, a testament to its robust design and the ongoing interest it generates among researchers and practitioners alike.

3.4. A Hybrid Statistical Approach to Stemming In Turkish

After reviewing literature related to Porter's stemmer and learning the basics of the other rule-based approaches, we decided to move on to statistical stemmers. Since we can define the rules of suffix addition in Turkish, we started to review Turkish stemmers with the statistical approach..

The article "A Hybrid Statistical Approach to Stemming in Turkish: An Agglutinative Language" [Kışla and Karaoğlan, 2016] provides an in-depth analysis of stemming in Turkish, which is an agglutinative language. Stemming is crucial for such languages due to the potentially infinite number of surface forms that can be generated from a single lexeme. The authors propose a novel method that leverages the fact that nouns and verbs have different suffix patterns, and apply statistical methods to strip off these suffixes to determine the part of speech (PoS) and consequently the stem boundary. The paper reviews various stemming methods for Turkish and highlights that previous efforts either relied on incomplete lexicons or produced multiple stem candidates requiring disambiguation. The proposed methodology is lexicon-independent, which increases its reliability and avoids the issues of open vocabulary. This approach uses two finite state machines (FSMs) to strip off suffixes from nouns and verbs, then employs n-gram statistics to pinpoint the actual stem among the potential candidates.

Experimental results using the METU-Sabancı Turkish Treebank demonstrate a high performance rate of 93.83 percent for their method on a hand-tagged corpus. When integrated with automatic PoS tagging, the performance slightly decreases due to the PoS algorithm's error rate. Nevertheless, the proposed method's reliance on a closed and restricted vocabulary of suffix patterns rather than a full dictionary addresses the problem of words not found in the dictionary. The researchers suggest further testing on a larger trained corpus and aim to create a comprehensive stemmed, PoS tagged, and well organized Turkish corpus in future work.

3.5. A Detailed Analysis of English Stemming Algorithms

The article "A Detailed Analysis of English Stemming Algorithms" [Hull and Grefenstette, 1996] presents a comprehensive evaluation of different stemming techniques and their impact on information retrieval. It delves into the comparison between traditional suffix-stripping algorithms and more linguistically grounded morphological analysis methods. One of the core findings is that, while all stemming methods tend to perform better than no stemming, there's little difference in average performance among the various algorithms. However, individual query analysis reveals significant variances, suggesting that the choice of stemming method can be crucial in certain cases.

The study uses a large test collection from the TREC/TIPSTER project to measure performance. The authors discuss the limitations of suffix-stripping stemming algorithms like Lovins and Porter, which can lead to both under-stemming (failing to conflate related words) and over-stemming (conflating words that are not semantically related). They also analyze the Xerox morphological tools, which perform both inflectional and derivational analysis, considering that inflectional morphology should be more relevant for information retrieval. Tables within the article, such as Table 1, illustrate the comparative performance of different stemmers using average precision and recall measures.

These tables are critical as they provide empirical data supporting the text's arguments, showing the nuanced effects of different stemming approaches on retrieval performance. The authors highlight that statistical significance testing of the results is necessary to validate the observed differences, as the differences among stemmers are generally small.

The article concludes that no single stemming method is universally superior, and suggests that linguistic stemmers might be improved by adapting them specifically for information retrieval tasks. The detailed query analysis suggests that linguistic knowledge could be used to improve the rules-based suffix removal approach, which might be overly simplistic for complex retrieval needs. The findings indicate that while linguistic-based stemming is not significantly better than traditional algorithms, there are several ways in which it could be optimized for information retrieval purposes, potentially making it more successful in the future.

3.6. Development of a Stemming Algorithm

The article ”Development of a Stemming Algorithm” [Lovins, 1968] is a pioneering work on the subject of computational linguistics and information retrieval. Lovins presents a detailed process of creating a stemming algorithm for English, focusing on a context-sensitive, longest-match principle that is designed to find the stem of a word by removing its longest possible ending. This work, significant for its early contribution to the field, also discusses the issues of spelling variation of stems and provides programmed solutions to these.

In the article, Lovins elaborates on the form and meaning of stemming, explaining that stemming algorithms must deal with the inherent limitation of not understanding the grammatical and semantic relations between words. The assumption that words with the same root are close in meaning underlies the functionality of a stemming algorithm. This assumption is not always accurate but works as an approximation for the purposes of information retrieval systems, such as the ones used in library catalogs.

The article discusses different types of stemming algorithms, their complexities, advantages, and drawbacks. It also provides a review of previous attempts to construct stemming algorithms, comparing their approaches. The compilation of a list of endings is described, showing how these are categorized and ordered in storage for efficient processing.

One important aspect emphasized in the article is the handling of ”spelling exceptions,” which are variations in the stem spellings that occur before certain endings.

Lovins proposes methods like recoding and partial matching to address these exceptions. The article includes tables that exemplify these spelling exceptions and the outputs of the stemming process, which are crucial to understand the algorithm's effectiveness and the adjustments needed for its improvement.

Finally, the article describes the two-phase stemming routine used in the Project Intrex, its results, and the iterative improvements made to the algorithm. This work provides foundational knowledge for those interested in the development of stemming algorithms, like us.

3.7. N-gram Based Stemming

The article *"Generation Implementation and Appraisal of an N-gram based Stemming Algorithm"* [Pande et al., 2019] begins with a comparison of the best-known stemming approaches: raw words (no stemming), Snowball stems, pseudo-stems, and 4-grams. Out of these approaches, the 4-grams approach tends to be the best according to the authors. The authors propose approximating the frequencies of 4-grams, 5-grams, ... n-grams instead of finding frequencies of overlapping N characters.

The idea is based on taking 4-grams as the initial guess and finding frequencies of n-grams where $n = 4, 5, \ldots$ to select the best among them. With this approach, words with a length of 3 or less will be discarded, which might not be a problem since many of the stop words are of length 3 or less.

The N-gram of a word selected as the stem must have a relatively good frequency across the corpus. All N-grams shorter than it must be highly frequent, and longer ones should exhibit lower frequencies gradually. The authors argue that at the point where the suffix starts, the change in the frequencies of N-gram and (N+1)-gram is very likely to be dramatic. The algorithm is as follows:

Let:

$\lambda_i = |F_i - F_{i-1}|\ if\ i > 4,\ 0\ else$, where F_i is the frequency of the ith N-gram

$\Delta_i = \lambda_i - \lambda_{i-1}$

Ψ_N: $the\ variable\ corresponding\ to\ the\ N - gram\ of\ length\ N$

And the proposed algorithm is as follows:

Phase 1:

1. $\lambda_4 = M$, where M is a very high integer
   $\Psi_N = 4$

2. calculate $\lambda_i\ where\ 4 < i \leq |word|$
   if $\lambda_i > \Gamma$: $\Psi_N = argmax_{4<i\leq word|}[F_i, F_{i-1}\}$, else: $\Psi_N = i$

   where Γ is a threshold value which defines how much frequency change of two N-grams is acceptable to select an N-gram.

3. if N = |word|, go to *Phase 2*. Else if $\Delta_i > 0$, stop.

Phase 2:

$$\Psi_N = \Psi_{N-3},\ if\ \Psi_{N-3} > 3\ AND\ F_{N-2} = F_{N-1} = F_N$$

The authors also compare this approach with Porter's Stemmer. After applying a Wilcoxon test to the results, it fails to accept that this approach is inferior to Porter's Stemmer.

3.8. Joint PoS Tagging and Stemming for Agglutinative Languages

The article *Joint PoS Tagging and Stemming for Agglutinative Languages* [Bölücü and Can, 2018] addresses the challenges of processing agglutinative languages, such as Turkish, where the number of word forms can be vast due to the extensive use of affixes. The paper presents an unsupervised Bayesian model that employs Hidden Markov Models (HMMs) for simultaneously performing part-of-speech (PoS) tagging and stemming. The use of stemming in this context helps reduce the sparsity typically encountered in PoS tagging tasks by decreasing the number of distinct word forms to distinct stem types.

In their approach, the authors propose several iterations of the model, each differing in how they handle stem and suffix information. The performance of these models is evaluated on Turkish and Finnish, both agglutinative languages, as well as English, which is morphologically less complex. The results demonstrate that the joint approach to PoS tagging and stemming enhances PoS tagging performance. This is especially true when using stem emissions rather than whole words, indicating that focusing on stems can lead to more accurate linguistic processing in agglutinative languages.

The paper includes several tables, such as Table 2, which maps the Universal tagset to the Penn Treebank tagset and the FinnTreeBank tagset, and Table 6, which presents stemming results for Turkish and Finnish, providing a numerical comparison of the different model settings and their accuracies. The results show that the stem-based Bayesian HMM models outperform the word-based models, especially in the context of agglutinative languages. However, the authors acknowledge that while their results for Finnish are competitive with existing models, their results for Turkish are not as strong and suggest that incorporating additional features, such as semantic information, may enhance performance, indicating potential areas for future research.

The conclusion emphasizes the success of the proposed models in reducing sparsity and improving PoS tagging accuracy by using stems and suffixes in a fully unsupervised learning framework. The researchers aim to refine their models by integrating more features to capture semantic similarities and addressing irregular word forms in future work.

3.9. YASS: Yet Another Suffix Stripper

In the paper *YASS: Yet Another Suffix Stripper* [Majumder et al., 2007], authors propose a clustering-based approach for stemming. They define various string distance measures and use them as metrics to cluster different variants of the same root into a single class.

Majumder et al. define four different string distance metrics for clustering. They pay special attention to defining these metrics in a way that rewards long matches and penalizes mismatches early.

Given two strings: $X = x_0x_1....x_n$ and $Y = y_0y_1....y_m$,

- They define a function p_i for penalty:

$$p_i = \begin{cases} 0 & \text{if } x_i = y_i \text{for 0 } le \text{ i } le \text{ min(n, n')} \\ 1 & \text{otherwise} \end{cases}$$

- If length of X does not match with the length of Y, they add null characters at the end of the shorter string until lengths match. After that, both strings will have length n+1 and they define the string distance measures D_1, D_2, D_3, and D_4:

$$D_1(X,Y) = \sum_{i=0}^{n} 2^{-i} * p_i$$

$$D_2(X,Y) = \begin{cases} \frac{1}{m}\sum_{i=m}^{n} \frac{1}{2^(i-m)} & \text{if } m > 0 \\ \infty & \text{otherwise} \end{cases}$$

$$D_3(X,Y) = \begin{cases} \frac{n-m+1}{m} \sum_{i=m}^{n} \frac{1}{2^(i-m)} & \text{if } m > 0 \\ \infty & \text{otherwise} \end{cases}$$

$$D_4(X,Y) = \begin{cases} \frac{n-m+1}{n+1} \sum_{i=m}^{n} \frac{1}{2^(i-m)} & \text{if } m > 0 \\ \infty & \text{otherwise} \end{cases}$$

After defining these metrics, they conducted various experiments to determine the best distance metric among them, as well as the optimal threshold and number of clusters for different languages, such as English, French, and Bengali. Based on the results of these experiments, they concluded that the performance of a stemmer generated by a clustering-based approach, without any language-specific rules, is comparable to rule-based approaches like Porter's Stemmer.

3.10. Hybrid Inflectional Stemmer and Rule-based Derivational Stemmer for Gujarati

The paper *Hybrid Inflectional Stemmer and Rule-based Derivational Stemmer for Gujarati* [Suba et al., 2011] focuses on developing two types of stemmers for the Gujarati language. The authors introduce a hybrid inflectional stemmer that combines an unsupervised learning model with Part Of Speech (POS) tagging and substitution rules to effectively identify and strip inflectional suffixes. This innovative approach results in a high accuracy of 90.7 percent. For derivational stemming, which deals with word formations that often change the word's part of speech, the authors propose a rule-based system. This system employs a set of linguistic rules for suffix stripping and achieves a 70.7 percent accuracy rate.

The development process of these stemmers involves utilizing the EMILLE corpus, which aids in identifying common stems and suffixes in the language. The hybrid stemmer utilizes a step-by-step approach that includes verifying if a word is already in stem form, applying POS-based stemming, using linguistic rules to find the split point between stem and suffix, and finally choosing the best split based on the frequency of the resulting stems and suffixes. This method ensures that the stemmer is not overly aggressive, which can lead to losing meaningful linguistic information, nor too lenient, which would result in ineffective stemming.

The paper presents data tables that illustrate the performance impacts of various algorithm configurations. For instance, it shows that not imposing a minimum stem size yields the best results, and that giving equal weight to stems and suffix frequency during the split selection process enhances accuracy. The authors conclude by acknowledging the success of their hybrid approach to inflectional stemming and suggest future improvements could be made to the derivational stemmer by refining the rule set and incorporating additional NLP modules, such as Named Entity Recognition, to further enhance performance.

4. METHODOLOGY

After investigating different stemming approaches, we have concluded that:

- Statistical stemmers cannot keep any suffix information since those stemmers try to find a substring of an input that is the most probable substring to be the actual stem of the input. In addition, selecting the most probable substring results in a stem that usually is not even a word. Since we want to keep suffix information and the actual word, dividing "arabadakiler" into "ara"/"arab" as the stem and "badakiler"/"adakiler" as suffixes does not help.
- Even though it is possible to seperate each suffix correctly, rule-based stemmers also suffer from the same problem: it is not good at finding the correct word as the final stem, "araba" from "arabadakiler" as in the previous example. This feature will be an essential part of our project since the expected behavior from our stemmer is starting to strip derivations after the elimination of inflectional suffixes from the word. Due to the rules of suffix addition in Turkish, the proper elimination of derivational suffixed requires the remaing part from inflection-removing algorithm to work correctly.

Based on these inferences, we decided to choose a hybrid approach: a stemmer with rules and a word dictionary. We can expand our proposed solution into three sections: *Base*, *Algorithm*, and *Challenges*.

4.1. Base of the Solution

We decided to use a word dictionary in our stemmer. The dictionary will help the stemmer to find the correct stem as the final result. The dictionary will be held in a set data structure, and we estimated 250 mb memory allocation for one million words.

The algorithm might find more than one stem from the input; for example, "kale"(castle) and "kalem"(pen/pencil) are two seperate words in Turkish and the word "kalemin"(my castle's / my pen's) has both of them in it. We will give both matches as the output and expect from another algorithm or the user to select the most probable stem.

The suffixes [Adalı, 2021a] [Adalı, 2021b] of Turkish will be put in a TRIE. Each suffix will be placed letter by letter, starting from the last and every letter of each suffix will be a node in the TRIE. Nodes can have custom rules to apply in case of a match, and each node will have a boolean variable which denotes whether the path from that node to the root of the tree is a suffix in Turkish. As seen in the Figure 4.1, suffixes

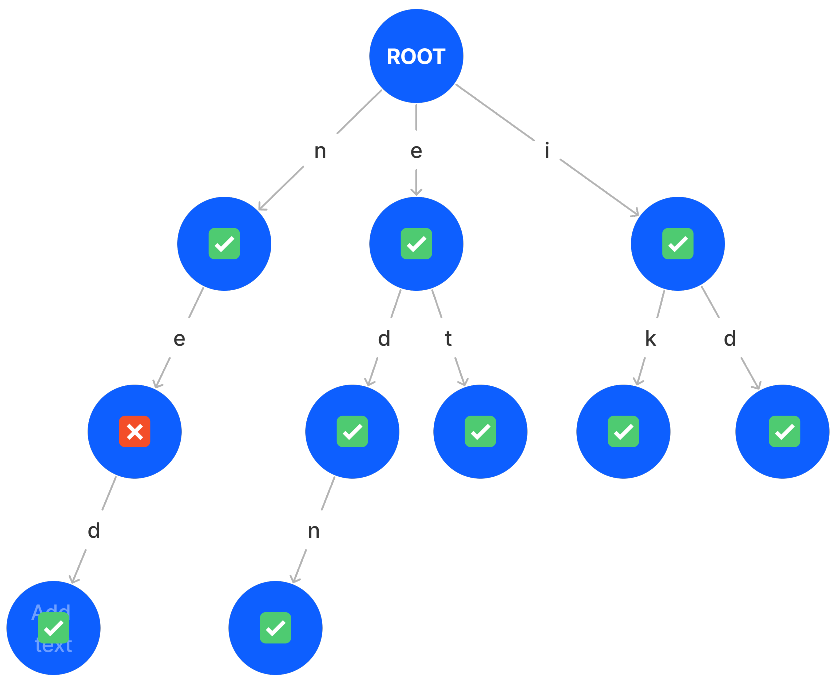


Figure 4.1. An example TRIE that will be used in the stemmer

are placed starting from the end: "-den" is a suffix in Turkish, and it is placed in the tree by "n" in the first level; then "e" and "d" finally. "-n" is also a proper suffix in Turkish; therefore, its boolean variable will be set to True. Oppositely, "-en" is not a proper *inflectional* suffix; as a result, its boolean variable will be False.

4.2. Algorithm

The stemmer will traverse the tree again and again, until the length of the remaining word is too short to be stripped, based on the word given as the input. During this traversal, it will consult the dictionary whenever required. Let x be the input word. Here is the pseudocode of the algorithm:

Algorithm 1 Stemming Algorithm

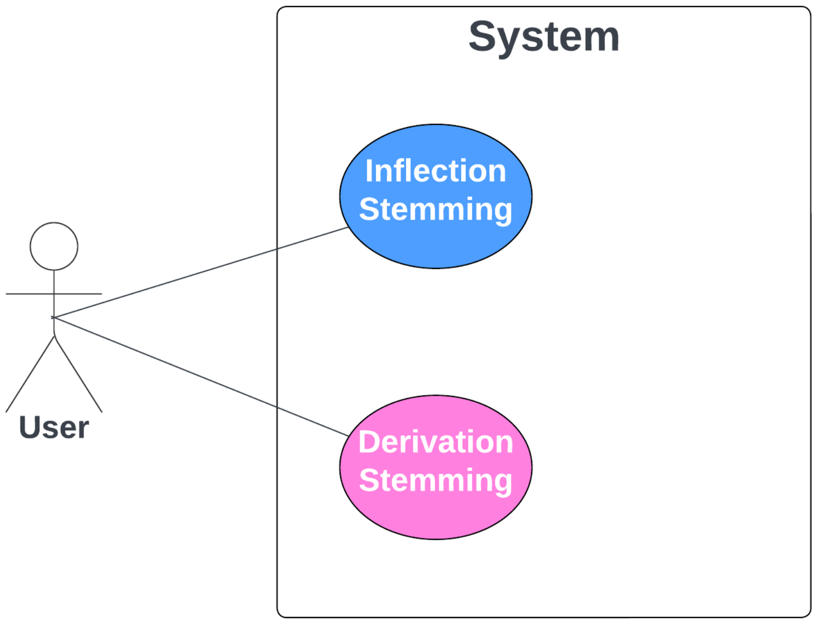
```
Require: $len(x) \geq 2$
1: $tree \leftarrow SuffixTree()$
2: $suffixList \leftarrow []$
3: $possibleStems \leftarrow []$
4: $dictionary \leftarrow TurkishDictionary()$
5: $stem \leftarrow x$
6: while $len(stem) \geq 2$ do
7:     if $dictionary.find(stem)$ then
8:         $possibleStems.add(stem)$
9:     end if
10:    $node, stem, match = tree.search(stem)$
11:    if $node \neq NULL$ then
12:        if $dictionary.find(stem)$ then
13:            $possibleStems.add(stem)$
14:        else
15:            $stem \leftarrow node.applyRules(stem)$
16:            if $dictionary.find(stem)$ then
17:                $possibleStems.add(stem)$
18:            end if
19:        end if
20:    end if
21: end while $len(stem) < 2$
```

4.3. Challenges

There are several cases in Turkish to keep in mind during the stripping suffixes:

- **Ünsüz Yumuşaması:** Since this case appears in the stem itself (”kalbim” should result in ”kalp” + ”i” + ”m”), we decided to add these cases as rules in the nodes.
- **Ünsüz Sertleşmesi:** This case appears in the suffix (”uçakta” should give ”uçak” + ”da/ta”). Therefore, this case will be in the TRIE as nodes.
- **Kaynaştırma:** This case creates a bridge between a suffix and a stem (”yakıyorum” is [”yak-” + ”-ı-” + ”yor” + ”um”]). We will handle this case by creating additional nodes in the tree, too.
- **Ünlü Düşmesi:** This case appears in the stem itself, too (”akla” should be ”akıl” + ”a”); therefore, this case will be considered by rules in the nodes.

5. REQUIREMENTS SPECIFICATION



6. DESIGN

6.1. Information Structure

In this project, we obtained a roots dictionary that we can use for the stems and gathered all the affixes in Turkish language, which are suffixes, classified as derivational and inflectional and further noun and verb below inflectional.

For storing, roots dictionary is added to our project as .pkl files which are used to serializing a tuple of two numpy arrays as we developed the stemmer in python. Suffixes, on the other hand, stored as .csv files and loaded to the tree structures when the stemmer starts working.

6.2. Information Flow

6.2.1. Activity Diagrams

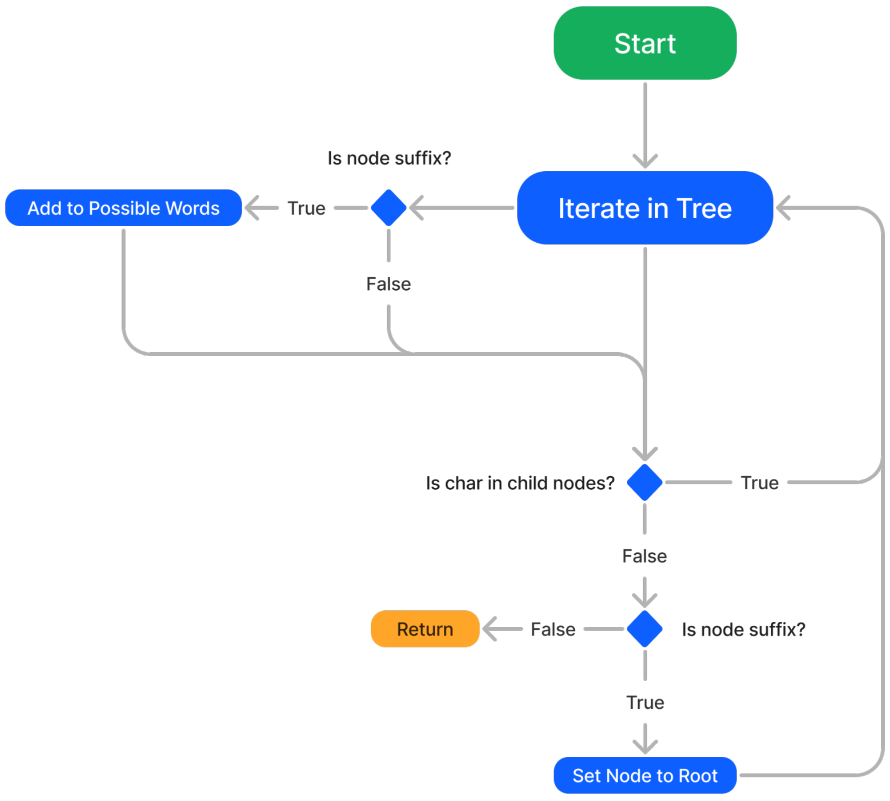


Figure 6.1. All Possible Words Search

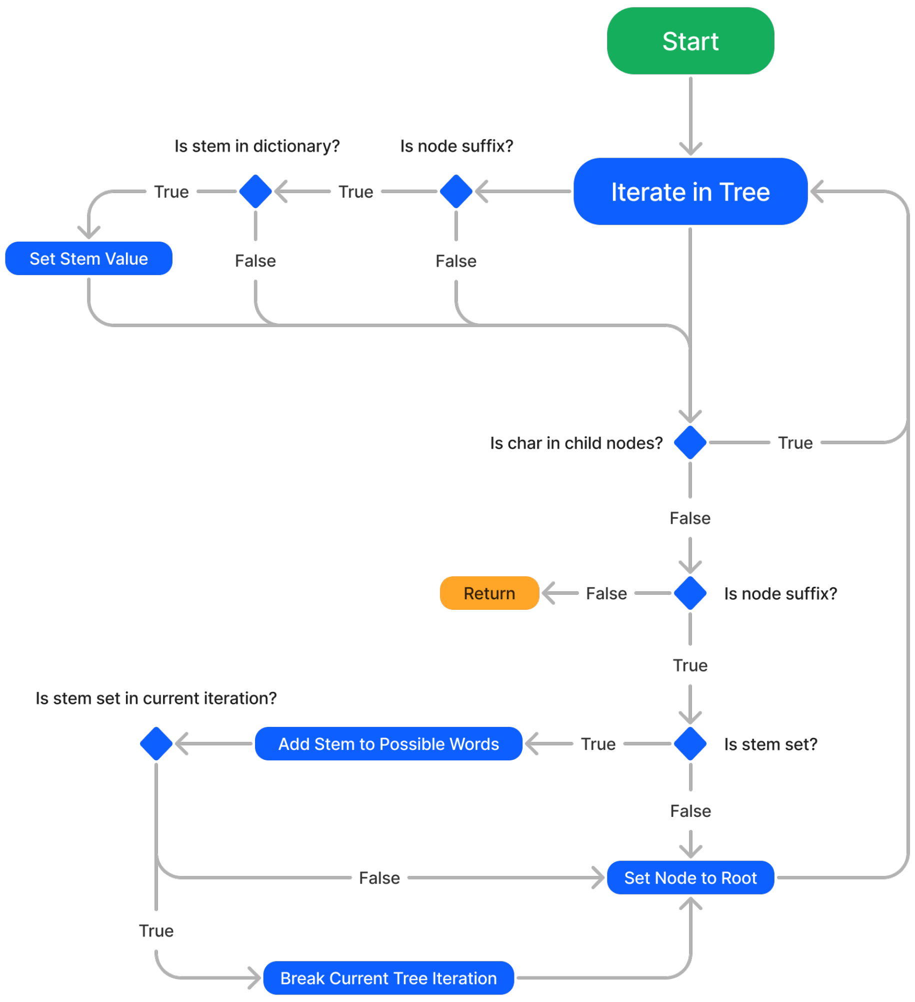
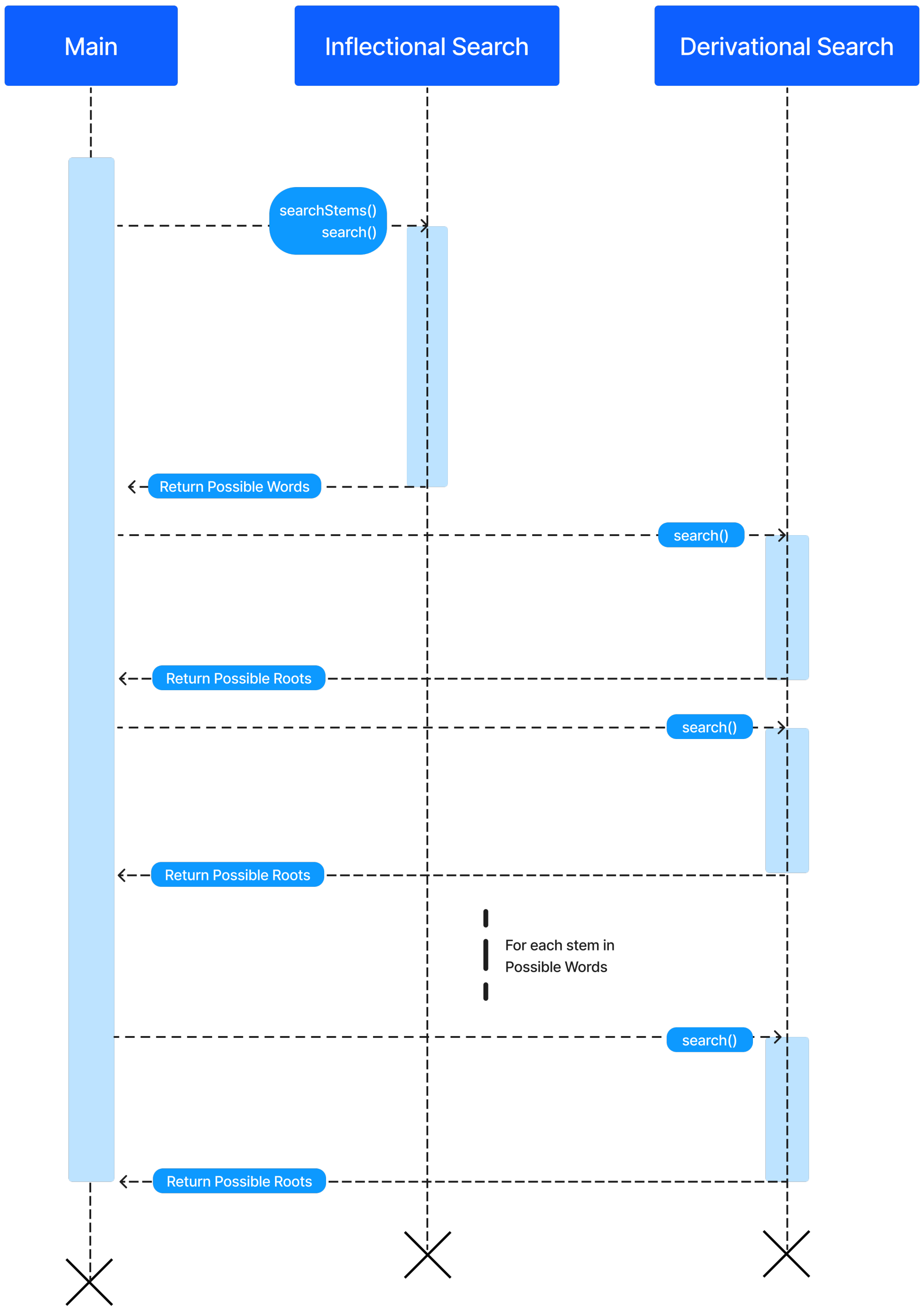


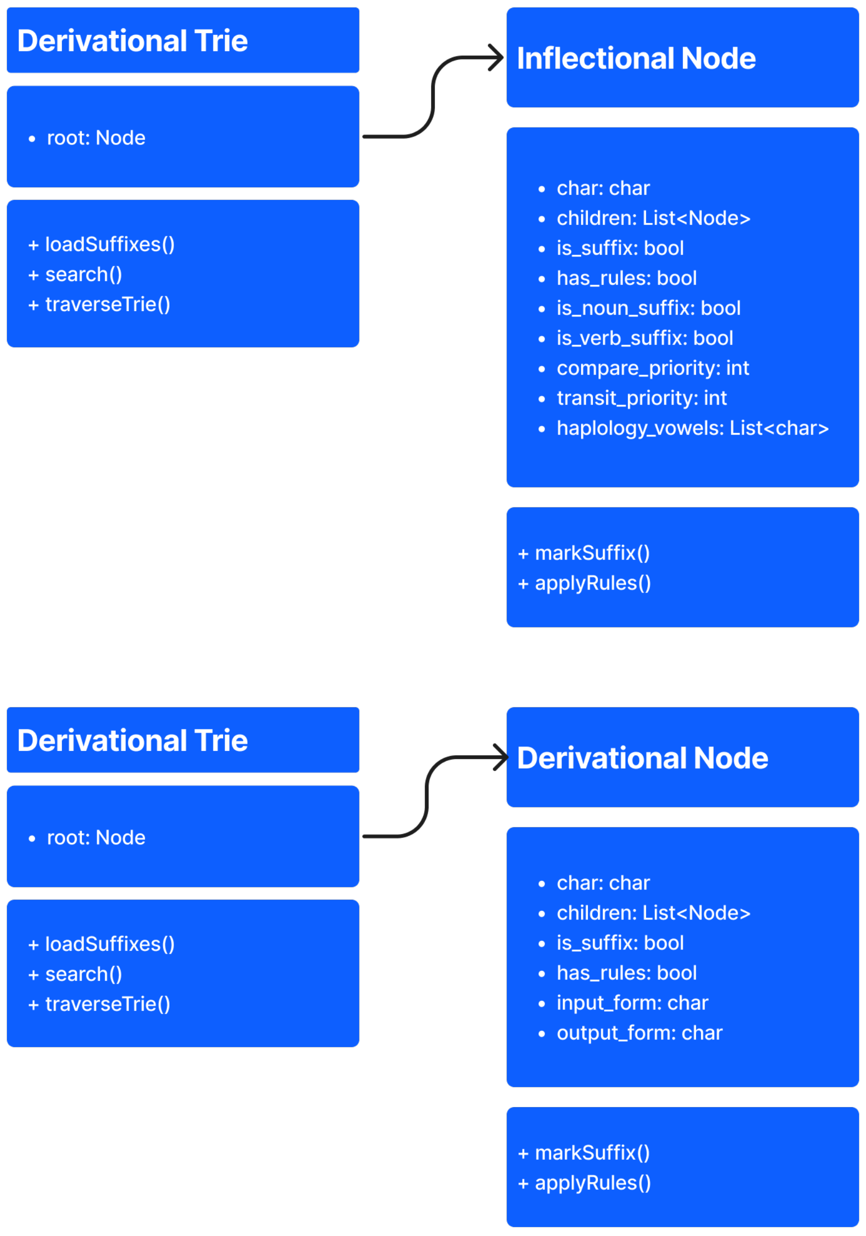
Figure 6.2. Longest Suffix Search

6.2.2. Sequence Diagram



6.3. System Design

6.3.1. Class Diagram



7. IMPLEMENTATION AND TESTING

7.1. Implementation

Before the steps of implementation, we obtained a roots dictionary that we can use for the stems and gathered all the affixes in Turkish language, which are suffixes, classified as derivational and inflectional and further noun and verb below inflectional.

7.1.1. Tree Structure for Suffixes

Suffixes are stored in trees separated as their classifications. Every node of a tree is holding a character of a suffix. Starting from the root of the tree, the suffixes are added backwards character by character as each possible previous character added as a child node. Each node is marked as if it forms to a suffix from the root.

7.1.2. Search Algorithm

The developed stemmer works by iterating through the input word starting from its last character, searching the current character in the children of the current node and analyzing remaining word according to roots dictionary at each iteration. There are two approaches to search algorithms that are used in this project.

- **All Possible Words Search** The algorithm iterates through the suffix tree recursively until the next character is not in the children or it reaches to a leaf. If the reached node marked as a suffix, it continues iterating through remaining word, resetting the current node to root. Unrelated to the existence of the remaining word in the roots dictionary, when recursion returns to the previous character due to finishing the word or reaching to a leaf, algorithm then tries to detach a suffix from the previous node, leading to find all possible suffix combinations along with all possible roots. This search algorithm is used to detach inflectional suffixes.

[göz - leri , göz - ler - i]

Figure 7.1. All Possible Words Search Outcome

- **Longest Suffix Search** The algorithm iterates through the suffix tree similar to All Possible Words Search, but this time, it returns to a shorter suffix only if there are no possible root found by going with the longest suffix. This search algorithm is used to detach derivational suffixes.

[göz - leri]

Figure 7.2. Longest Suffix Search Outcome

The two search algorithms are used in an order to find the inflectional suffixes first and derivational suffixes later as in Turkish, there should not be any derivational suffix comes after an inflectional suffix.

7.1.3. Optimization

As the algorithm is implemented to the project, it came to the surface that some optimizations are needed to enhance the correctness and effectiveness.

- **Priority** [Kartallıoglu, 2003] Finding all possible suffixes of a word can mean too much possibility in Turkish. This lead us to research about the order of the suffixes to gather information. As there are lack of documentation about the ordering of suffixes in Turkish, we analyzed and gave suffixes some priority values.
- **Third Tree for Inflectional** Some of the inflectional suffixes have more than one usage along with them to be common in noun and verb trees. This effected the accuracy of our solutions and some cases exceeded priority. Thus, we added

an additional tree that consists of 2 suffix types as they can only come to the end of words and appear to be problematic.

7.2. Testing

We decided to compare the possible morphological variants this algorithm gives with the results given by Morphological Analyzer project in TULAP(Turkish Language Processing Platform)[1] to the same word. We accepted Morphological Analyzer's output as ground truth at this stage.

Morphological Analyzer gives the output with a vowel harmony notation concern. As an example; 'sın', 'sin', 'sun', 'sün' are different forms of the same inflectional suffix, and they are represented as '+sHn' altogether. In this project, however, different trees have the same suffix, different formations of the same suffix are in seperate tress, and single suffixes stand for more than one suffix that have different roles. As a result, it is impossible to make this project give an output similar to Morphological Analyzer's. Therefore, we decided to assimilate the Morphological Analyzer's output to our stemmer's.

In addition, Morphological Analyzer gives additional information such as the type of the root(noun, adjective, etc.), the roles of the suffixes, etc. Detailing the analysis gives rise to duplicate results: 'kırmızıya' is 'kırmızı' + 'YA'; however, 'kırmızı' can be both adjective and noun. Event though Morphological Analyzer gives both versions in the result, we get list of unique morphological parses since neither the word's nor suffix' role is our concern.

To compare our stemmer with Morphological Analyzer, we need a list of tokens to stem in both approaches. For this purpose, we used TS Corpus Word List from Turkish Data Depository[2] , in which there is a list of Turkish tokens. After eliminating unhelpful

[1]Morphological Analyzer, TULAP: `https://tulap.cmpe.boun.edu.tr/entities/tool/a14b8c30-536e-41e2-ade7-27047c1d1222`

[2]Turkish Data Depository: `https://data.tdd.ai/`

entries like words shorter than three characters or words that contain whitespaces, we have left with near 1000000 tokens. After sampling a set with 200000 tokens, we started comparing the results of our project to the ground truth Morphological Analyzer for these tokens.

7.3. Deployment

The instructions in the README file of the project[3] are sufficient to deploy the project and start using it.

[3]Turkish Stemmer - CMPE492: `https://github.com/musasimsekdotdll/Turkish-Stemmer`

8. RESULTS

There are two metrics that can be measured during the test: root accuracy and variant accuracy. Root accuracy is calculated by comparing lists of unique roots, morphological variant accuracy is calculated by comparing lists of unique morphological parses. We measured precision and recall values of these metrics. Precision is the rate of correct entries to total number of entries in our project's output: number of common entries in the unique lists of these two models divided by number of unique entries. Recall, on the other hand, is the rate of correct entries returned to all correct entries that should be returned. Table 7.1 gives the final output of our tests. For a rule-based stemmer that gets help from a dictionary, finding proper morphological variants for half of all words is quite exciting. On top of it, this project could be compared with standard stemmers in information retrieval tasks in terms of both success and speed given the root accuracy results.

	Precision	**Recall**	**F-Score**
Morphological Variants	0.434	0.607	0.506
Root	0.629	0.711	0.668

Table 8.1. Root Accuracy & Morphological Variant Accuracy Results

9. CONCLUSION

This project aimed to enhance the diversity in linguistic field by adding a stemmer for Turkish language. We achieved this goal by developing our stemmer with a mixed approach, traversing with rules. This approach is based on the syntax of Turkish language having affixes only as suffixes and edge effects occur within the addition of these suffixes. Our algorithm traverse the word recursively which gives itself the ability to track the different possible suffix combinations. Along with the traversing, the edge effects are easily handled with the rules that we designed to apply when searching a stem in the dictionary.

The used approach gives this study an opportunity to be further used in morphological analysis of Turkish language as we not only stripped words to their common roots, but also tracked the suffix combinations in the outcome. Furthermore, it is included in a small number of studies that concern language processing in Turkish, which can create ideas for improvements to this project or new projects that use the knowledge shared within our process.

REFERENCES

Adalı, 2021a. Adalı, E. (2021a). Türkçenin yapım ekleri.

Adalı, 2021b. Adalı, E. (2021b). Türkçenin Çekim ekleri.

Bölücü and Can, 2018. Bölücü, N. and Can, B. (2018). Joint pos tagging and stemming for agglutinative languages. In *Computational Linguistics and Intelligent Text Processing: 18th International Conference.*

Can et al., 2008. Can, F., Kocberber, S., Balcik, E., Kaynak, C., Ocalan, H., and Vursavas, O. (2008). Information retrieval on turkish texts. *Journal of the American Society for Information Science and Technology.*

Hull and Grefenstette, 1996. Hull, D. and Grefenstette, G. (1996). A detailed analysis of english stemming algorithms. *Rank Xerox Research Centre 6.*

Kartallıoglu, 2003. Kartallıoglu, Y. (2003). The sequenct of the declension affixes in turkish and mongolian. *Selçuk Üniversitesi Türkiyat Araştırmaları Dergisi(13).*

Kılınç et al., 2017. Kılınç, D., Özçift, A., Bozyigit, F., Yıldırım, P., Yücalar, F., and E., B. (2017). Ttc-3600: A new benchmark dataset for turkish text categorization. *Journal of Information Science, 43(2), 174-185.*

Kışla and Karaoğlan, 2016. Kışla, T. and Karaoğlan, B. (2016). A hybrid statistical approach to stemming in turkish: An agglutinative language a. *Anadolu Üniversitesi Bilim ve Teknoloji Dergisi.*

Lovins, 1968. Lovins, J. (1968). Development of a stemming algorithm. *Mech. Transl. Comput. Linguistics.*

Majumder et al., 2007. Majumder, P., Mitra, M., Parui, S., Kole, G., Mitra, P., and

Datta, K. (2007). Yass: Yet another suffix stripper. *ACM transactions on information systems.*

Mawlood-Yunis, 2013. Mawlood-Yunis, A.-R. (2013). Porter stemmer.

Nielsen, 2016. Nielsen, P. J. (2016). *Functional Structure in Morphology and the Case of Nonfinite Verbs.* Brill Academic Pub.

Pande et al., 2019. Pande, P., Tamta, P., and Dhami, S. (2019). Generation, implementation and appraisal of an n-gram based stemming algorithm. *Digital Scholarship in the Humanities.*

Suba et al., 2011. Suba, K., Dipti, J., and Bhattacharyya, P. (2011). "hybrid inflectional stemmer and rule-based derivational stemmer for gujarati.". In *Proceedings of the 2nd workshop on south southeast Asian natural language processing (WSSANLP).*

Tsiftsis et al., 2014. Tsiftsis, G., Stathopoulos, T., Fertakis, N., Mantas, C., and Pagkalos, S. (2014). Turkish stemmer.

Willett, 2006. Willett, P. (2006). The porter stemming algorithm: then and now. *Program: electronic library and information systems, Vol. 40 No. 3, pp. 219-223.*