A Rule-based Recursive Lemmatization Algorithm with POS tagging for Plagiarism Detection in Nepali texts

Ayush Kumar Shah
Department of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14623, USA
as1211@rit.edu

***Abstract*—There have been many notable works for detecting plagiarism in the English texts, but none in the Nepali texts, which uses Devanagari scripts. It is mostly due to the involved challenges in the pre-processing of Nepali texts caused by the lack of available datasets and the complicated grammatical rules and syntactic structure of the Nepali language. Since pre-processing texts are vital in performing any Natural Language Processing (NLP) task, we aim to accomplish this task for Nepali texts. The paper (1) introduces a set of 140 custom suffix and prefix rules to extract the root word, that deals with the grammatical complexities and syntactic structure specific to the Nepali language, (2) presents a rule-based recursive lemmatization algorithm, which considers the effect of parts of speech (POS) to pre-process Nepali texts using these rules, (3) presents a NEP-PLAG2019v1 dataset[1] (4) implements a Nepali Plagiarism Detection System, which uses tf-idf feature vector obtained from the pre-processed texts as inputs to compute the Cosine similarity to detect plagiarism. For the NEP-PLAG2019v1 dataset (10,000 pairs), the proposed method was able to obtain an F1-score of 96.05%. The ability to extract root words with higher accuracy from Nepali texts, with the complex grammar and structure handled by the introduced rules, gives rise to opportunities to perform various NLP tasks in the Nepali language.**

***Index Terms*—Plagiarism, POS, Tokens, tf-idf, Stemming, Lemmatization**

I. Introduction

To identify and deal with plagiarism, we need to build a system that can detect plagiarism of any form in texts. A significant component of such systems is generating useful features from the texts using appropriate and accurate stemming or lemmatization algorithm. There have been many works to address this problem. In 1980, Porter presented a simple algorithm [1] for stemming English language words. This algorithm is the basis for pre-processing texts until now.

Pre-processing of texts written in Devanagari scripts, specifically in the Nepali language is more challenging than those in the English language. The general stemmers cannot extract the root word accurately due to the complex and unique syntactic structure of the Nepali language. Similarly, the grammatical rules are more vast than in the English language. For instance, the verbs in the Nepali sentences require gender agreement, in addition to the number agreement (in the English language), with the subject of the sentence. E.g. the verb ”eat” is खान्छ and खान्छे for male and female subjects, respectively. Likewise, there are different forms of the same pronoun and verb for different levels of respect for the person. E.g., the sentence ”You eat.” becomes तँ खान्छस् । तिमि खान्छौ । तपाई खानुहुन्छ । for three different levels of respect for the person.

S.D. Makhija [2] has described different types of stemming algorithms and summarized stemmers designed for different languages based on Devanagari scripts. D. Monika et.al. [3] have presented the development of Hindi stemmer based on Devanagari script for stripping both prefixes and suffixes to obtain the root word using the combination of lookup algorithm, suffix stripping algorithm, and prefix removal algorithm. The lookup table approach makes the system inefficient to use. [4] used a supervised approach using n-gram models to design a common stemmer for languages in Devanagari scripts. They tested four approaches based on frequency, statistics, length, and iterations to split the words into suffix and stem. This method does not consider prefix, and the results are worse in the Nepali language, since the complex grammar and structure specific to the Nepali language are not considered. [5] used a unique approach of romanization (one to one mapping of each vowel and consonant of Devanagari script into an English roman code to exploit the English coded corpus. The common limitations in the previous works are not considering the dependency of stemming on the parts of speech (POS) since the same word occurring as different POS results, into a different root word, and the different complex grammatical structures specific to the Nepali language.

This paper aims to overcome the limitations in the

[1]https://github.com/ayushkumarshah/Nepali_Plagiarism_Detection/tree/master/datasets/NEP-PLAG2019v1

Note: The dataset and results of this work have not been generated yet and are simply meant to be an exercise in research writing. However, the methods and setup are real.

previous works by introducing 140 different prefix and suffix rules, which consider the POS and complex grammar and syntactic structure of the Nepali language to extract the root words from Nepali texts with higher accuracy. A rule-based recursive lemmatization algorithm is implemented to pre-process the Nepali texts effectively using those rules. The paper also contributes a new dataset NEP-PLAG2019v1 for plagiarism of Nepali texts. Finally, we convert the pre-processed tokens into tf-idf feature vectors and compute Cosine similarity measures between the resulting vectors to classify the pair of texts into Plagiarised and Non-Plagiarised classes using a threshold value (25%). We have obtained an F1-score of 96.05% on the NEP-PLAG2019v1 dataset. The code[2] used in our experiments is publicly available.

II. Preprocessing of documents

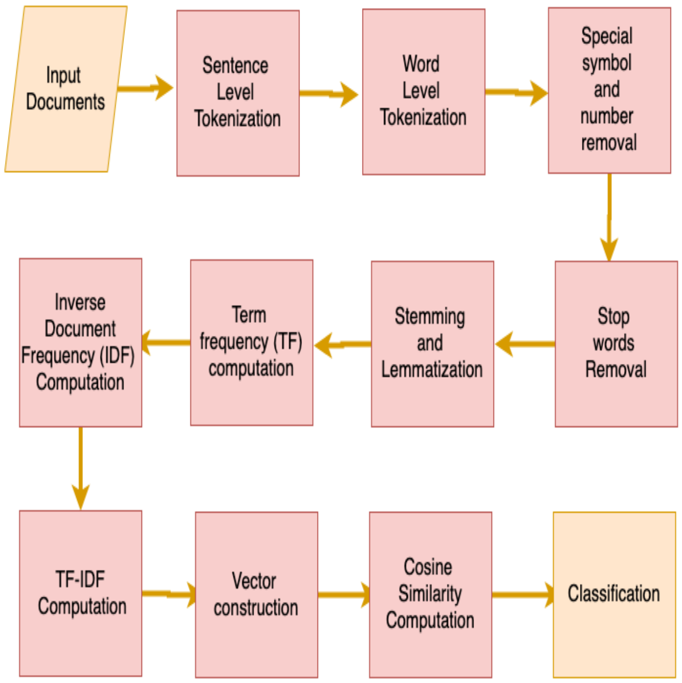
Input Documents
Sentence Level Tokenization
Word Level Tokenization
Special symbol and number removal
Stop words Removal
Stemming and Lemmatization
Term frequency (TF) computation
Inverse Document Frequency (IDF) Computation
TF-IDF Computation
Vector construction
Cosine Similarity Computation
Classification

Figure 1. Flow diagram for Plagiarism detection showing the major steps

As shown in Figure. 1, there are various operations involved in the methodology including pre-processing of Devanagari texts using the introduced rule-based stemming algorithm, feature vector construction using tf-idf and finally, similarity computations for the final classifications.

A. Sentence tokenization

The text in the document is split into sentences, thereby allowing line-by-line processing in the subsequent sentences. Vertical bars, question marks, and exclamation marks are used to break down the text into individual sentences.

[2]https://github.com/ayushkumarshah/Nepali_Plagiarism_Detection

Example:

"परिश्रम नगरी हुन्छ ? परिश्रम सफलताको एक-मात्र बाटो हो। अक्सर जो परिश्रम गर्छ, उही सफल हुन्छ। अब त परिश्रम गर्छौ नि ? नगरी कहाँ हुन्छ त !"

The following collection of sentences is tokenized into:

'परिश्रम नगरी हुन्छ ?', 'परिश्रम सफलताको एक-मात्र बाटो हो।', 'अक्सर जो परिश्रम गर्छ, उही सफल हुन्छ।', 'अब त परिश्रम गर्छौ नि ?', 'नगरी कहाँ हुन्छ त !'

B. Word tokenization

The input sentences are broken down into individual tokens. White space and comma are used to break down the words.

Example:

['परिश्रम', 'नगरी', 'हुन्छ ?', 'परिश्रम', 'सफलताको', 'एक-मात्र', 'बाटो', 'हो।', 'अक्सर', 'जो', 'परिश्रम', 'गर्छ', 'उही', 'सफल', 'हुन्छ।', 'अब', 'त', 'परिश्रम', 'गर्छौ', 'नि ?', 'नगरी', 'कहाँ', 'हुन्छ', 'त !']

C. Special symbol and number removal

Special symbols like ,)([]?! " "":– and numbers both in English and Nepali are removed from the tokens using regular expressions.

D. Stop words removal

Stop words are high-frequency words that do not have much influence in the text and are removed to increase the performance. A list of 592 stop-words like त्यही, जब, अथवा, were collected, and these words were removed from the list of tokens.

III. Stemming and Lemmatization

Our primary contribution constitutes the stemming and lemmatization algorithm for the Nepali language. We have taken POS and complex grammatical and syntactic structures into account to design the algorithm. Lemmatization is more complex than stemming since the result of lemmatization is a meaningful word present in the dictionary as opposed to stemming. We have implemented lemmatization using a corpus of words in the Nepali dictionary to ensure we almost always get a valid root word. Hence, the features used to construct vectors are a better representation of the document, while the irrelevant features are removed. We have developed our algorithm with the help of the Language Technical Kendra[3]. Note

[3]http://ltk.org.np/

that the term 'stemmer' used throughout the paper corresponds to the lemmatization algorithm although the name suggests a stemming algorithm.

A. Prerequisites of the Stemmer

The prerequisites of the Stemmer Module would be the following [6]:

1) POS Tagset
2) Tokenizer
3) Free morpheme-based lexicon
4) Two sets of affixes for the suffix and prefix
5) A database of word breaking grammatical rules

B. POS Tagset

The NLP team at Madan Puraskar Pustakalaya, Nepal has developed a relatively simplified POS Tagset of 91 tags. The tagset has been developed targeting the Grammar Checker for Nepali. The tagset development guidelines and experiences of Hindi, Urdu and some others like the British National Corpus have been consulted in developing the current POS Tagset. The POS tag coverage test of the developed POS TagSet is currently being tested. For the testing purpose, the free morpheme list entries and the affixes list are being POS tagged [6].

C. Set of Affixes

Stemming is a crude heuristic process that chops off the ends of words in the hope of achieving root words (morphemes) and often includes the removal of derivational affixes. There are two sets of affixes: suffixes and prefixes. The prefixes and suffixes are stemmed for obtaining a root word.

For example, In a word 'सफलताको' , 'सफल' is a root and it is combined with the suffix 'ता' and 'को' and the resulting form becomes 'सफलताको'. Some of the prefix and suffix in the dataset is as follows:

Prefix set:	Suffix set:
न|1	यो|17
उप|2	दा|18
महा|3	दै|19
अ|4	◌ँदै|19
सु|5	पालिका|20

Here, the number present after the suffix and prefix, delimited by the '|' pipe sign, points to the suffix and prefix rules present in the word breaking grammatical rules to strip the affixes.

D. Grammatical rules for inflection

Stemming often results in a word that is only close to the root word, which may not be found in the dictionary. Lemmatization is a more proper process with the use of vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. These lemmas are obtained by applying the suffix and prefix rules repeatedly and using the root word collection for verification.

During the breaking of words, insertion, and deletion of one or more free vowels and vowel symbols or dependent vowels is a common phenomenon.

Examples:

सु+आगत=वागत

In the above example, सु and the character आ combine to form स्वा and in doing so, the vowel sign ◌ु is deleted, the consonant स is reduced to स् and the vowel आ is substituted by the cluster वा.

These patterns are taken into consideration while breaking the words into the morphemes. Also, the POS is introduced into the rule to decide on how to break the word according to its POS.

To formulate the rules and apply them, the irregular affixes will have to be studied in more detail. For regular affixes, we have formulated the rule with a header line, which includes the rule number for indexing, the type of affix, the number of sub rules, the morpheme as an affix, and the respective grammatical category it represents. All of these fields are space delimited. After the header line, sub rules are present. The rules are written in the form:
[Rule no][POS][n(sub-rules)][Morpheme][Tag][Ignore]

Some of the rules are:

Prefix rules:

```
3 PFX 1 महा महा
महा .

4 PFX 1 अ NEG
अ .

5 PFX 1 सु PTV
सु .
```

Suffix rules:

```
13 SFX 3 छु CHU N
 ◌ँछ .
 न्छु .
  छु .

14 SFX 2 छे प N
◌ँछ .
 छे .

15 SFX 2 छौ CHAU N
◌ँछौ .
छौ .
```

Abbreviations:
NN – Common Noun
VV – Verb Base Form
ADQ – Adjective Qualitative
PFS– Pronoun First Person

SFX - Suffix
PFX - Prefix

PLE – Ergative
HRU – Plural Marker

Example: खानुभयो खाए खाउ खान्छु to खा

The number at the beginning indicates the rule number, followed by the POS tag, number of sub-rules, the morpheme structure, tag, and an N or Y to indicate whether to ignore or not to skip the rule for smaller suffixes to be handled. Below the main rule, are the sub-rules in which the first part is the substring to delete from the word and the second part is the substring to insert to the word. The dot indicates not to insert any substring. The sub rules are simple. The first field indicates what is to be deleted and the second field indicates what is to be appended. The two fields are delimited by space [6].

E. Stemming and Lemmatization Algorithm

The summary of the algorithm has been illustrated in Figure. 2.

Input: A list of pre-processed tokens

Output: A list of lemmas (root words) obtained by striping the affixes using the rules

1) Check whether the token is lemma or not using the root words and alternate root words collections. If root word, return success with the token unchanged. If not, go to the next step.
2) Check whether the token when added to halanta or virama (◌्) gives a root word or not. If yes, then return the corresponding root word as the new token. If not, go to the next step.
3) Check if the token end with halanta or virama (◌्) and the token without virama gives a root word or not. If yes, set the token to the corresponding root word. If not, go to the next step.
4) Check whether any of the suffixes are present in the token. If yes, then strip the suffix using the corresponding rules and set the token as the corresponding root word. If not, go to the next step.
5) Check whether any of the prefixes are present in the token. If yes, then strip the prefix using the corresponding rules and set the token as the corresponding root word. If not, go to the next step.
6) If both prefix and suffix are present, recombine the suffix one by one and check for lemma verification. This case may arise when the suffix and prefix cannot be found individually but may be found after recombination with the suffix. In this case, the root word may contain a suffix, and hence the suffix are recombined instead of the prefix.
7) Repeat steps 1 to 6 until root word is found or the token is unrecognized.
8) If the token is unrecognized, return the token without striping else return the striped token as the lemma for the token.

IV. Feature Vector Construction

A. Calculating tf-idf (Term frequency-inverse document frequency)

Term frequency-inverse document frequency is a statistical measure that measures how important a term is relative to a document and a corpus, a collection of documents. It is used to construct the vector representation of the text documents. Mathematically, the tf-idf of a term is given by the equation (1).

$$\textit{tf-idf}(t,d,D) = \textit{tf}(t,d) \times \textit{idf}(t,D) \tag{1}$$

where,

t = a particular term,

d = a particular document,

D = corpus of documents,

$\textit{tf}(t,d)$ = term frequency of term t in document d,

$\textit{idf}(t,D)$ = inverse document frequency of term t in a corpus of documents D,

So, the tf-idf weight is composed of two terms:

1) Normalized Term Frequency (tf)
The term frequency (tf) is the measure of how frequently a term occurs in a single document or article. The frequency is normalized by dividing it by the total number of terms.

A term appearing relatively higher is an important term and has a higher tf value. Mathematically, the term frequency is given by (2)

$$\textit{tf}(t,d) = \frac{f_d(t)}{\sum_{w\in d} f_d(w)} \tag{2}$$

where,

$f_d(t)$ = number of term t in document d,

$\sum_{w\in d} f_d(w)$ = Total number of terms in document d

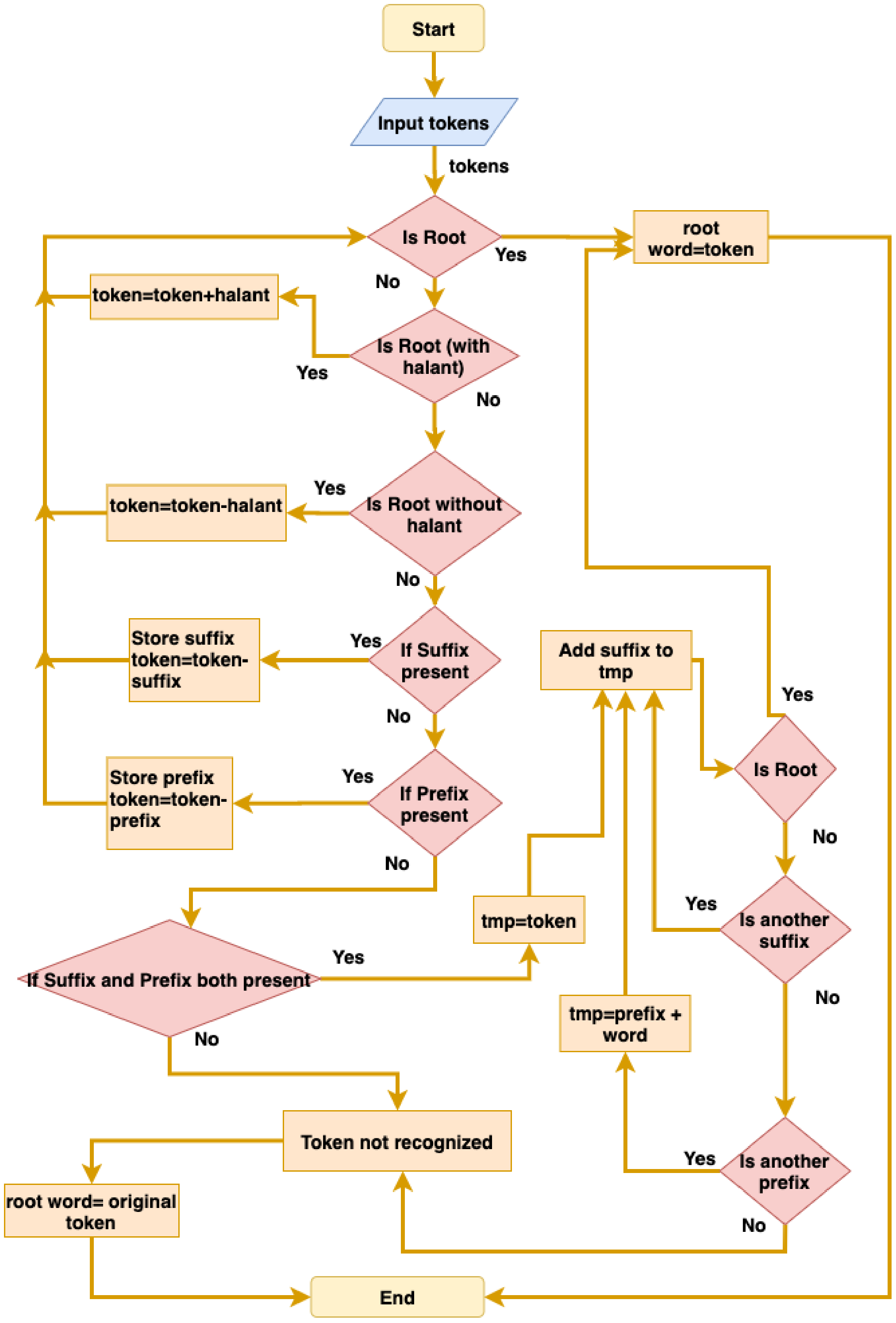

Figure 2. Detailed Flow diagram for stemming and lemmatization algorithm

2) Inverse Document Frequency (idf)
The inverse document frequency (idf) is the measure of how important a term is to the entire collection of documents in the corpus. It weighs down the effects of terms that occur too frequently while weighs up the effects of less frequently occurring terms by inverting the number of documents containing the particular term [7]. We calculate idf for each term which is mathematically given by (3)

$$idf(t, D) = \ln\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right) \tag{3}$$

where,

$|D|$ = Total number of documents in the corpus,

$|\{d \in D : t \in d\}|$ = Number of documents containing term t

So, from (1), (2), and (3), we can write the tf-idf as defined in (4)

$$tf\text{-}idf = \frac{f_d(t)}{\sum_{w \in d} f_d(w)} \times \ln\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right) \tag{4}$$

An excellent example of how idf comes into play is for the word ”the.” We know that just about every document contains ”the,” so the term is not significant, thereby producing a very low IDF [8].

B. *Construction of vector (Vector Space Model)*

Using the relation in (4), we construct tf-idf dictionaries for each term-document pair. Thereupon, we create a matrix that consists of an array of vectors, where each vector represents a document in the corpus. The vector will be a list of frequencies for each unique word in the corpus – the tf-idf value if the word is in the document, or 0.0 otherwise. The set of documents in the corpus is then viewed as a set of vectors in a vector space, as shown in Figure. 3. Each term in the corpus will have its axis [9].

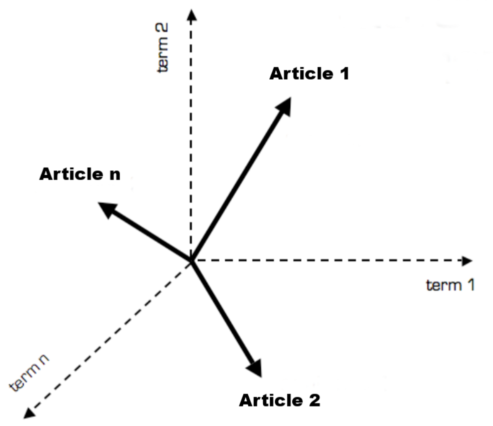

Figure 3. Vector space model.

V. Calculating similarity

A. *Cosine similarity*

It is a metric used to measure the degree of similarity between two vectors. The vectors’ size doesn’t affect the similarity value since the magnitude of the vectors automatically normalizes the value. We use this metric to compute the similarity between pairs of news articles using the equivalent tf-idf feature vector representation of the articles. Mathematically, it measures the cosine of the angle between two vectors projected into a multi-dimensional space. In this context, the two vectors are arrays containing each term’s tf-idf value in the two documents.

The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, the higher, will be the cosine similarity. When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude [10], as shown in Figure. 4. Using the formula given in (5), we can find out the cosine similarity between any two documents.

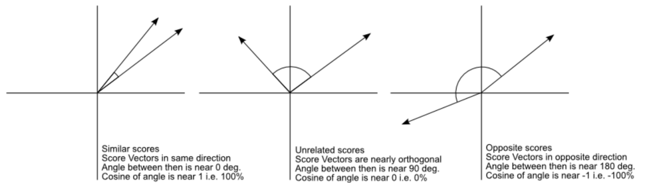
Similar scores
Score Vectors in same direction
Angle between then is near 0 deg.
Cosine of angle is near 1 i.e. 100%
Unrelated scores
Score Vectors are nearly orthogonal
Angle between then is near 90 deg.
Cosine of angle is near 0 i.e. 0%
Opposite scores
Score Vectors in opposite direction
Angle between then is near 180 deg.
Cosine of angle is near -1 i.e. -100%

Figure 4. Cosine similarity visualization

$$\cos(\vec{a}, \vec{b}) = \frac{\vec{a}.\vec{b}}{\|\vec{a}\|\|\vec{b}\|} = \frac{\sum_{i=1}^{n} \vec{a}_i \vec{b}_i}{\sqrt{\sum_{i=1}^{n} (\vec{a}_i)^2}\sqrt{\sum_{i=1}^{n} (\vec{b}_i)^2}} \tag{5}$$

where,

$\vec{a}$ and $\vec{b}$ represents tf-idf vectors of two documents,

n is the length of the vector or vocabulary size

B. *Jaccard similarity*

It is a measure to calculate the similarity between any two documents or terms or sentences. To calculate the Jaccard similarity, the number of common attributes is divided by the number of attributes in at least one of the two objects. Jaccard Coefficient can also be used as a dissimilarity or distance measure, unlike cosine similarity. If the Jaccard index between two sets is 1, then the two sets have the same number of elements in the intersection as the union, and we can show that $A \cap B = A \cup B$. So every element in A and B is in A or B, so $A = B$. The

Jaccard index can also be used on strings. We define a set containing the characters in a string for each string, so the string "cat" becomes c, a,t. If we have two strings, "catbird." and "cat," then the numerator is 3, and the denominator is 7, which gives us a Jaccard index of 3/7.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \qquad (6)$$

VI. Results and Discussions

One of the major challenges in performing experiments for plagiarism detection in Nepali texts was the lack of labeled datasets. We collected the following data from different sources. These data have been made publicly available[4].

A. Corpora

1) NEP-PLAG2019v1
We collected 10000 pairs of Nepali news articles from different online news portals sites and manually annotated them into plagiarised and non-plagiarised pairs. This dataset is published as NEP-PLAG2019v1. It contains 4553 pairs of plagiarised and 5447 pairs of non-plagiarised articles. Hence, the classes are almost balanced.

2) Nepali root words
A total of 20,641 Nepali root words were collected from Language Technical Kendra.

3) Suffixes
A total of 135 Nepali suffixes were collected from Language Technical Kendra.

4) Prefixes
A total of 5 Nepali prefixes were collected from Language Technical Kendra.

5) Suffix and prefix rules
We wrote the rules for striping the corresponding suffixes and prefixes, respectively, to obtain root words.

6) Stop words
A total of 592 stop words were collected from multiple online sources.

B. Experiment

For experimentation, we used the 10,000 pairs of annotated news articles in the NEP-PLAG2019v1 dataset. We performed a rule-based recursive stemming algorithm on the articles in the dataset using our rules to produce tokens consisting of the root words. We then fed the pre-processed tokens into the feature selection process to construct the tf-idf feature vectors as per the process

[4]https://github.com/ayushkumarshah/Nepali_Plagiarism_Detection/tree/master/datasets

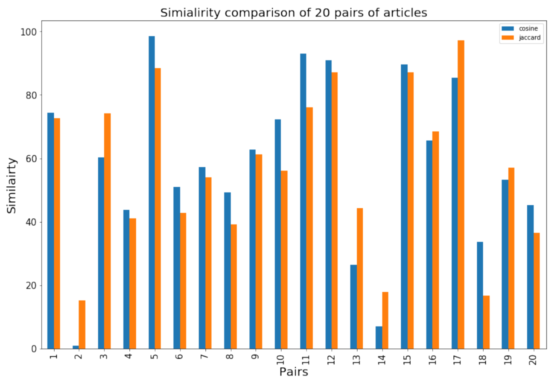

Figure 5. Cosine and Jaccard similarity scores comparison for a random sample of 20 pairs of articles

explained in the methodology section above. Both Cosine and Jaccard similarity scores were computed for each pair of articles. The comparison between the two score metrics in a random sample of 20 pairs of articles from the corpus are shown in Figure. 5.

Unlike the Cosine similarity score, the Jaccard similarity does not take the frequency of input tokens into account. Also, the comparison of results obtained from Cosine similarity and Jaccard similarity for all the articles' pairs confirm that Cosine similarity always yields better results. Hence, we use only the Cosine similarity score for classification of article pairs.

Table I
Confusion matrix on classification of 10,000 pairs of news articles in the NEP-PLAG2019v1 dataset

		Predicted	
		Plagiarised	**Non-plagiarised**
Actual	**Plagiarised**	4397	156
	Non-plagiarised	205	5242

Table II
Evaluation Metrics on classification of 10,000 pairs of news articles in the NEP-PLAG2019v1 dataset

Evaluation Metrics	**Value**
Accuracy	96.39%
Precision	95.54%
Recall	96.57%
F1-Score	96.05%

A threshold of 25% cosine similarity score was used to classify each pair of articles as plagiarised or non-plagiarised. The confusion matrix obtained in the classification task on 10,000 pairs of articles in the NEP-PLAG2019v1 dataset is shown in Table I. Also, the results of the evaluation metrics on the same task are shown in Table II.

C. Discussions and Comparisons

Rule-based segmentation approaches have been used in Devanagari scripts in previous works [3] as well. However, most of the rule-based and other previous approaches [4], [5] have not considered the complex grammatical rules specific to Nepali language and the dependency of the stemming and lemmatization behaviour on the POS of the derived word in the sentence. Moreover, [4] does not consider prefixes for extracting the root word, and hence returns invalid root words for words containing prefixes. The use of lookup tables is not efficient in terms of computation and time as well. Also, the approaches generalized for many languages based on the Devanagari script [4], [5] fails to capture the unique changes in the vowels and consonants of a specific language, here Nepal texts, during the inflection of derived words into the root words. These factors contribute to a decrease in the accuracy of obtaining a valid root word, ultimately affecting the plagiarism classification.

The proposed method has tackled with these limitations by defining both prefix and suffix rules, which are applied recursively until the root word is obtained. The approach can handle any unique cases of changes in the vowel or consonant, specific to Nepali texts as well. Likewise, including the POS tags in the rule adds the ability to apply different rules to the words with consideration to the POS of the word in the sentence. Improvements have been made to ensure that the root word is almost always valid by using a dictionary to perform lemmatization.

For feature selection, tf-idf feature vectors are used, since they are easy to compute and represent the texts well. They give fair results for plagiarism tasks since the dataset used was relatively simple. However, tf-idf feature vector is likely to fail in capturing the features of a complex dataset since it cannot capture the semantic context in the sentence and document level. Hence, for future works, tf-idf should be replaced by other complex embedding models like word2vec or BERT.

The results obtained from the task of plagiarism detection using the proposed rule-based recursive segmentation and tf-idf feature vector construction with Cosine similarity have been compared with approaches using previous methods of stemming and pre-processing but with the same post-processing steps using tf-idf and Cosine similarity on the NEP-PLAG2019v1 dataset in the Table III.

Table III
Comparison of Plagiarism classification F1-scores on 10,000 pairs of articles in the NEP-PLAG2019v1 dataset

Method	**F1-score**
Rule-based recursive lemmatization*	96.05% *
Effective Devanagari Hindi Stemmer [3]	86.05%
Lightweight general Devanagari Stemmer using n-gram [4]	87.05%
Romanization based stemmer [5]	92.05%

The comparison results clearly show that the proposed method outperforms the previous stemmers designed for Devanagari scripts.

VII. Conclusion

The paper introduced a stemming algorithm using a set of custom-defined recursive rules to pre-process Devanagari scripts specifically for the Nepali language, considering the complex syntactic and grammatical structures, and dependency on POS of the word in order to detect plagiarism with higher F1-score in Nepali articles. Since the proposed algorithm is able to outperform previous Devanagri script-based stemmers, it can be a helpful step in various NLP tasks in Nepali texts, other than Plagiarism as well. Most of the NLP tasks include the pre-processing step before employing the main algorithm or the required task. Moreover, this algorithm can be extended to other languages as well by developing similar recursive rules as per the grammatical rules of the corresponding language and incorporating the variation in the rules to represent POS dependency. Hence, this paper has a wide range of applications in the field of NLP.

The obvious limitation to this approach is the algorithm's non-robust nature since the grammatical and syntactic rules required in this algorithm need to be defined manually, which is dependent on the language completely. Likewise, the semantic and contextual information in the texts is not considered, which may reduce the performance in a large dataset. Due to insufficient dataset of Nepali texts, the evaluations performed in this paper were done on our own NEP-PLAG2019v1 dataset. Hence, the evaluation may not be a standard benchmark for comparing the performance.

In the future, we plan to overcome the current limitation by incorporating semantic and contextual information during the computation of the similarity measures by using deep learning approaches to capture the context of the words instead of relying on the frequency entirely. Likewise, another approach may be comparing the similarities of the obtained tokens' meanings by constructing and using a dictionary instead of the similarities between the tokens. Thereby, we need to construct a corpus containing Nepali root words and their corresponding meanings. When similarity is measured at a semantic level, sentences having different words but giving the same meaning will get a higher similarity score.

Acknowledgment

Our thanks to our supervisor, Associate Professor Bal Krishna Bal, for assisting and guiding us throughout the project and sincere thanks to Language Technical Kendra for providing us the required datasets and resources.

References

[1] P. Willett, "The porter stemming algorithm: Then and now," *Program electronic library and information systems*, vol. 40, 07 2006.

[2] S. D. Makhija, "A study of different stemmer for Sindhi language based on Devanagari script," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, Mar. 2016, pp. 2326–2329.

[3] D. MONIKA, T. ABHISHEK, and M. UPENDRA, “An effective stemmer in Devanagari script,” Hamirpur, India, Apr. 2013, pp. 22–25. [Online]. Available: https://www.seekdl.org/conferences/paper/details/1309.html

[4] S. R. Dangui and N. Naik, “A Lightweight Stemmer for Devanagari Script,” in *Proceedings of the 8th Annual ACM India Conference*, ser. Compute ’15. New York, NY, USA: Association for Computing Machinery, Oct. 2015, pp. 55–62. [Online]. Available: http://doi.org/10.1145/2835043.2835061

[5] B. P. Pande, P. Tamta, H. S. Dhami, and S. Campus, “A Devanagari Script based Stemmer,” p. 12.

[6] B. K. Bal, P. Shrestha, and M. P. Pustakalaya, “A morphological analyzer and a stemmer for nepali.”

[7] J. Vembunarayanan, “Tf-idf and cosine similarity,” 10 2013. [Online]. Available: https://janav.wordpress.com/2013/10/27/tf-idf-and-cosine-similarity/

[8] Triton, “Triton subscription engine for publishers,” 01 2012.

[9] I. C. Mogotsi, “Christopher d. manning, prabhakar raghavan, and hinrich schütze: Introduction to information retrieval,” *Information Retrieval*, vol. 13, no. 2, pp. 192–195, Apr 2010. [Online]. Available: https://doi.org/10.1007/s10791-009-9115-y

[10] S. Prabhakaran, “Cosine similarity – understanding the math and how it works (with python codes),” 10 2018. [Online]. Available: https://www.machinelearningplus.com/nlp/cosine-similarity