A Novel Rule-based Recursive Stemming Algorithm for Plagiarism Detection in Devanagari Scripts

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 such scripts caused by the lack of available datasets and the complicated grammatical rules and 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 introduces a rule-based recursive stemming algorithm that effectively handles the complexities in grammar and the language structure to pre-process Nepali texts. Furthermore, we use pre-processed texts to develop a Nepali Plagiarism Detection System using tf-idf feature vector construction with Cosine similarity measure. The ability to pre-process Nepali texts gives rise to opportunities to perform various NLP tasks in the Nepali language.**

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

I. Introduction

Plagiarism is one of the increasing crimes brought by increased use of the internet. It is the act of citing a part or whole document that has been copyrighted without mentioning the author(s) correctly. It is also described as a form of stealing other people's ideas by copying intrinsically or extrinsically, but still closely resembles the idea of the source document without mentioning its author correctly [1]. To identify and deal with this crime, we need to build a system that can detect plagiarism of any form in texts. A major component of such systems is generating good features from the texts using an appropriate and accurate stemming algorithm.

There have been many works to address this problem. In 1980, Porter presented a simple algorithm for stemming English language words. This algorithm is the basis for pre-processing texts until now. Willet [2] summarises the main features of the algorithm and highlights its role in modern information retrieval research and a range of related subject domains. It works well in practice to a range of languages and has spurred interest in stemming as a topic for research [2].

Over the years many methodologies have been developed to perform automatic detection of plagiarism, including tools for natural language text detection such as Turnitin (iParadigms, 2010) and CopyCatch (CFL software, 2010), and tools for computer programming source code detection such as MOSS (Aiken, 1994). Various approaches have been developed to deal with both external and intrinsic plagiarism in written texts (Lukashenko Graudina & Grundspenkis, 2007) [3] but mostly in the English language. For instance, the results of applying several NLP techniques to process a set of suspicious and original documents, by analyzing strings and the structure of the text, using resources to account for text relations showed that NLP techniques improve the accuracy of existing approaches [3]. Likewise, Z.Ceska [4] purposes plagiarism tools based on Singular Value Decomposition (SVD) technique like SVDPlag which solves associations among phrases contained in the examined documents to infer the mutual similarity of all pairs of the documents. Also, D. Gupta, V. K, and C. K. Singh [5] present a plagiarism detection method, which focuses on detecting intelligent plagiarism cases where semantics and linguistic variations play an important role. Although these works have contributed significantly to detecting various forms of plagiarism in English texts, they fail to perform effectively in scripts of other languages like Devanagari scripts for the Nepali language.

This paper aims to overcome the challenges in the task of pre-processing Devanagari scripts. Some various complicated grammatical rules and structures need to be considered during the pre-processing task. A simple stemming algorithm developed or used in the English language does not perform well for the Nepali language due to these structural and semantic differences. As a result, we aim to develop a novel rule-based recursive stemming algorithm to effectively pre-process the Devanagari scripts and ultimately use it to detect plagiarism between Nepali texts.

This paper defines several grammatical and syntactic rules to demonstrate the rule-based recursive stemming algorithm for Devanagari scripts, followed by the construction of tf-idf feature vectors on the tokens obtained. Finally, we compute similarity measures like Cosine and Jaccard between the resulting vectors to classify the pair of texts into Plagiarised and Non-Plagiarised classes using a threshold value. We have obtained an F1-score of 96.55%

on a manually annotated Devanagari scripts dataset containing 100 pairs of original and suspicious scripts.

II. Methodology

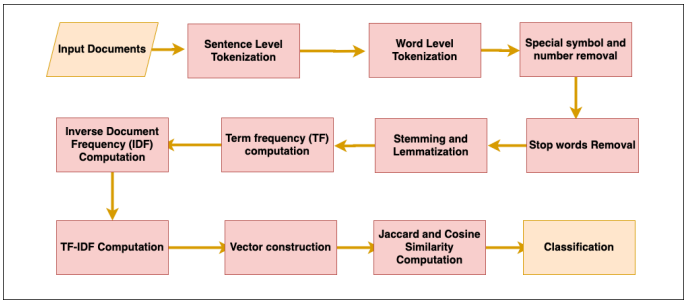

Figure 1. Flow diagram for Plagiarism detection showing the major steps

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. Preprocessing of documents

1) 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.

Example:

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

2) 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', 'गर्छौं', 'नि ?', 'नगरी', 'कहाँ', 'हुन्छ', 'त !']

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

4) 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.

5) Stemming and Lemmatization
Our major contribution is in the stemming and lemmatization algorithm for Devanagari scripts. Several stemming algorithms are available for the English language. However, such general stemming algorithms do not perform well for Devanagari scripts. Unfortunately, there is no algorithm available to stem Nepali texts which use Devanagari scripts. Hence, we have developed our algorithm with the help of the Language Technical Kendra.

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 from 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:

We have introduced a rule-based stemming approach, which agrees with all the grammatical restrictions for stemming the texts in the Nepali language. As a result, the output of the stemming is almost always a valid root word. It means that the features used to construct vectors are a better representation of the document, while the irrelevant features are removed. A few prefixes and suffixes are listed below.

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 and used to strip the affixes.

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.

The rules are written in the form:
[Rule no][POS][n(sub-rules)][Morpheme][Tag][Ignore]

The number at the beginning indicates 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 [6].

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 .

Example: खानुभयो खाए खाउ खान्छु to खा

Stemming and Lemmatization Algorithm:

Input: A list of pre-processed tokens

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

i 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.

ii 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.

iii 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.

iv 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.

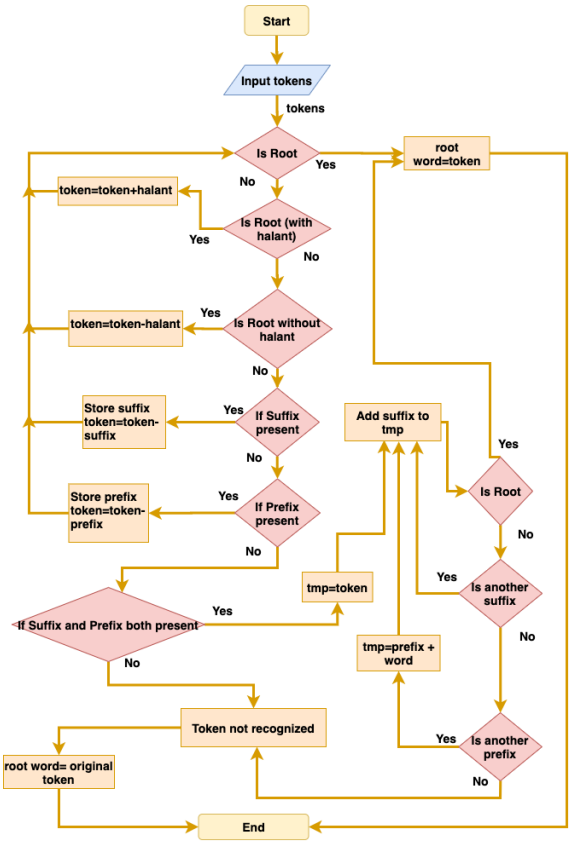

Figure 2. Detailed Flow diagram for Plagiarism detection including all the stemming and lemattization steps

v 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.

vi If both prefix and suffix are present, recombine the suffix one by one and check for lemma verification.

vii Repeat steps i to vii until root word is found or the token is unrecognized.

viii If the token is unrecognized, return the token without striping else return the striped token as the lemma for the token.

B. Feature Vector Construction

1) 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).

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

where,

t = a particular term,

d = a particular document,

D = corpus of documents,

$tf(t, d)$ = term frequency of term t in document d,

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

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

i 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)

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

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

ii 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)

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

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)

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

A good 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].

2) 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].

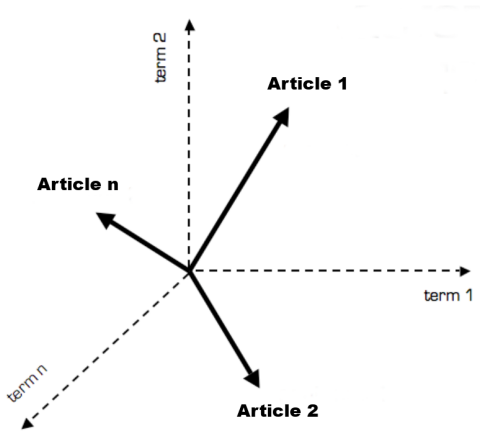

Figure 3. Vector space model.

C. Calculating similarity

1) 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.

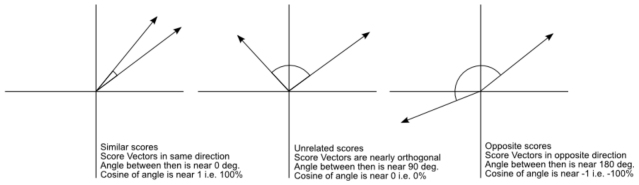

Figure 4. Cosine similarity visualization

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

where,

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

n is the length of the vector or vocabulary size

2) 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.

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

III. Results

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.

A. Corpora

1) News articles
We collected a pair of 100 Nepali news articles from different online news portals sites.

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

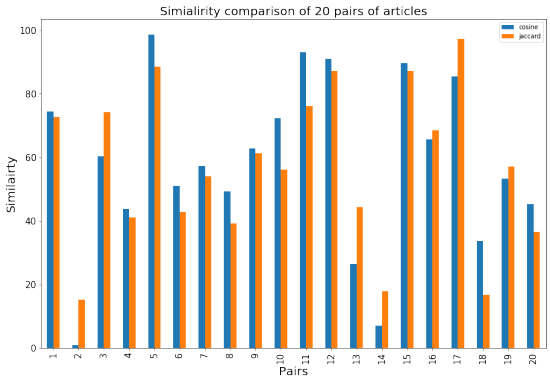

Figure 5. Cosine and Jaccard similarity scores comparison for 20 pairs of articles

For experimentation, we annotated the 100 pairs of news articles collected from different online news portal sites as plagiarised and non-plagiarised. Then using our rule-based recursive stemming algorithm, we performed pre-processing of these articles. The pre-processed texts were then fed into the feature selection process to construct the tf-idf feature vectors as per the process explained in the methodology section above. Both Cosine and Jaccard similarity scores were computed for each pair of articles.

The scores of a sample of a sample of 20 pairs of articles from the corpus are shown in Figure. 5

Comparing the results obtained from Cosine similarity and Jaccard similarity for all the articles' pairs, we observed that Cosine similarity always yields better results. Hence, after obtaining the similarity scores, a threshold of 25% cosine similarity score was used to classify a pair of the article as plagiarised or non-plagiarised. The confusion matrix obtained in the classification task on 100 pairs of manually annotated articles is shown in Table I. Also, the results of the evaluation metrics on the same task are shown in Table II.

Table I
Confusion matrix on classification of 100 pairs of manually annotated news articles

		Predicted	
		Plagiarised	Non-plagiarised
Actual	Plagiarised	70	0
	Non-plagiarised	5	25

Table II
Evaluation Metrics on classification of 100 pairs of manually annotated news articles

Accuracy	95%
Precision	93.33%
Recall	100%
F1-Score	96.55%

IV. Conclusion

This paper introduced a novel stemming algorithm using a set of custom-defined recursive rules to pre-process Devanagari scripts effectively for the Nepali language in order to detect plagiarism with more precision and recall in Nepali articles. Since a proper pre-processing method for Devanagari scripts has not been developed yet, this algorithm can be helpful in various NLP tasks, other than Plagiarism as well, that need to be performed in such scripts. 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. 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, the evaluations performed in this paper were done on a manually annotated dataset, limited in number. 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. One 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] R. Adam and S. Suharjito, "Plagiarism detection algorithm using natural language processing based on grammar analyzing," *Journal of Theoretical and Applied Information Technology*, vol. 10, pp. 168 – 180, 05 2014.

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

[3] M. Chong, L. Specia, and R. Mitkov, "Using natural language processing for automatic detection of plagiarism," 01 2010.

[4] Z. Ceska, "Plagiarism detection based on singular value decomposition," in *Advances in Natural Language Processing*, B. Nordström and A. Ranta, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 108–119.

[5] D. Gupta, V. K, and C. K. Singh, "Using natural language processing techniques and fuzzy-semantic similarity for automatic external plagiarism detection," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sep. 2014, pp. 2694–2699.

[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