Nepali Plagiarism Detection

Araju Nepal
CE 4th year
araju7nepal@gmail.com

Manasi Kattel
CE 4th year
manasikattel1@gmail.com

Ayush Kumar Shah
CE 4th year
ayush.kumar.shah@gmail.com

Abstract

Natural Language Processing (NLP) analyses the structure of the text, using resources to account for text relations. There have been many notable works related to Plagiarism Detection techniques in English language but very few in Nepali language mostly due to the increased complexity and lack of available resources. So, in this paper we aim to work in this area by attempting to build a Nepali Plagiarism Detection System using various NLP techniques and similarity measures. As a part of this project, an application has also been developed where files can be uploaded and similarity measure is displayed.

1 Introduction

Plagiarism is one of the crimes brought by increased use of internet which is citing a part or whole document that have been copyrighted without mentioning the author(s) in the correct way. 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]

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). Plagiarism has become one of the most concerned problems since there are several kinds of plagiarism that are hard to detect. Current plagiarism detection tools aren't accurate. By incorporating Natural Language Processing (NLP) techniques into existing tools and approaches plagiarism detection can be made more accurate. Various approaches have been developed to deal with both external and intrinsic plagiarism on written texts (Lukashenko Graudina & Grundspenkis, 2007) but mostly in English language.[2]

There are many existing plagiarism detectors in various languages but not a proper one in Nepali language. Therefore,our aim here is to use the existing approach of plagiarism detection using Natural Language Processing (NLP) in Nepali language.

Our project simply aims to apply NLP tasks like stemming, lemmatization and calculating similarity measures including cosine similarity, Jaccard similarity to indicate the percentage of plagiarism between various Nepali documents uploaded through a simple user interface.

2 Related Works

2.1 Plagiarism detector

It is an intelligent free and most accurate software.The content entered is analyzed on the basis of its lexical frequencies,word-choice, matching phrases and many other important factors. The given text is mapped into their internal network, and then it is compared against different databases and the entire internet. The algorithm has been designed to especially ignore statistically common phrases to provide a better and more valuable search for potential plagiarism in the text. Once the plagiarism test completes, the results will appear along with the match percentage that their best plagiarism tool has found.

2.2 noplag.com

Noplag develops and market an online plagiarism checker tool. It allows users to manage task and collaborate with employees. The platform supports multiple file formats. The individual can create their customized library. The platform analyses the file and provide a useful statistics to the user. It also allows users to download and share plagiarism report. This platform is offered to writers, students, educators, companies, and websites.

2.3 Plagiarisma

Plagiarisma is a simple and free to use website that tests a given text for plagiarism. It does so by breaking down the text into various pieces and checking if those pieces can be found online on different websites. The search can be executed on Google, Bing, and Yahoo! search engines – it depends on your choice. The results are then displayed in a comprehensive and understandable manner.You can enter text in three ways: by simply pasting/typing it, by entering the text’s URL, or by uploading a document file. The yellow highlighting in the results, you will immediately find out if the text is unique. If the text is not unique and matching results have been found online, then the URL of the matching results is displayed. The result stats are also displayed; these include the character and word count of the text, the number of unique sentences, and the percentage of text that has been plagiarized.

3 Methods

3.1 Software Specification

Front End Tools:

Programming language - Python 3.6.8
Operating System - Ubuntu linux,Windows

Back End Tools : Jupyter Notebook, Anaconda, Atom Text Editor, PyCharm, Git

Libraries used:

1. **Numpy** : It is the fundamental package for scientific computing with Python and can be used as an efficient multi-dimensional container of generic data. It was used for calculation of dot product and norms required for computing cosine similarity.

2. **Tkinter:** Tkinter is Python's de-facto standard GUI (Graphical User Interface) package which was used to create an interface to upload files and calculate similarity.

3.2 Methodology

A. Corpora:

1. **News articles (for detection)**
   We collected few Nepali articles from different online news sites.

2. **Nepali root words**
   A total of 20,641 Nepali root words were collected from Language Technical Kendra. http://ltk.org.np

3. **Suffixes**
   A total of 135 Nepali suffixes were collected from Language Technical Kendra. http://ltk.org.np

4. **Prefixes**
   A total of 5 Nepali prefixes were collected from Language Technical Kendra. http://ltk.org.np

5. **Suffix and prefix rules**
   The rules for striping the corresponding suffixes and prefixes respectively to obtain root words

6. **Stop words**
   A total of 592 stop words collected from multiple sources online like Kushal Poudyal's github prepared for nlp.icodejava.com

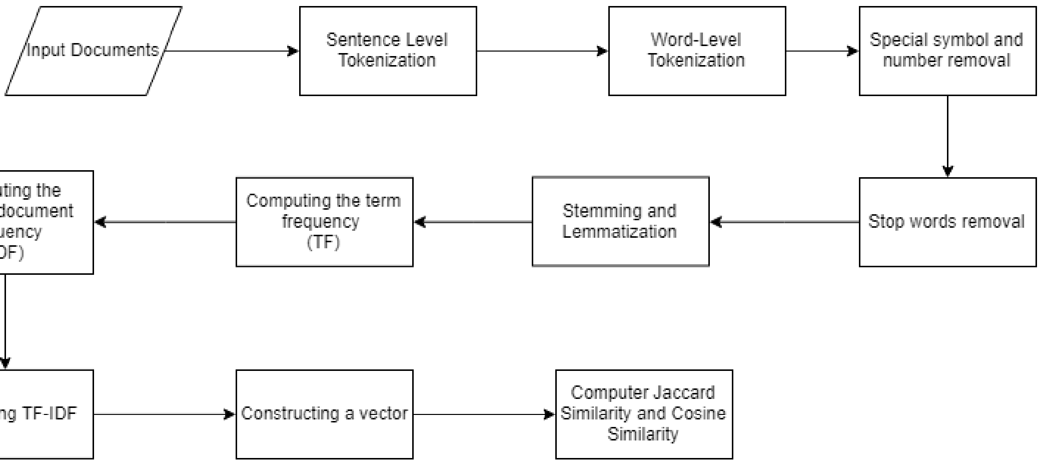

Input Documents
Sentence Level Tokenization
Word-Level Tokenization
Special symbol and number removal
Stop words removal
Stemming and Lemmatization
Computing the term frequency (TF)
Computing the inverse document frequency (IDF)
Computing TF-IDF
Constructing a vector
Computer Jaccard Similarity and Cosine Similarity

Fig 1: Flow diagram for Plagiarism detection

B. Preprocessing of documents

1. Sentence tokenization

The text in the document is split into sentences and thereby allowing line-by-line processing in the subsequent sentences.Vertical bar, 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!']

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 has not much influence in the text and are removed to increase the performance.

A list of 592 stop-words like त्यही, जब, अथवा, etc was collected and these words were removed from the list of tokens.

5. Stemming and lemmatization

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.

Several stemming algorithms are available for the English language. Unfortunately, there is no algorithm available to stem a Nepali word. Hence, we have developed our own algorithm with the help of reference from research done by Language Technical Kendra.

There are two sets of affixes: suffixes and prefixes. The prefixes and suffixes are stemmed to obtain 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:
न|1
उप|2
महा|3
अ|4
सु|5

Suffix set:
यो|17
दा|18
दै|19
ँदै|19
पालिका|20

We have used rule based stemming approach in this project. So, the number that is present after the suffix and prefix, delimited by the '|' pipe sign points to the **suffix and prefix rules** which are sued for the striping off the affixes.

Stemming often results in a word which is only close to the root word which may not be found in the dictionary. **Lemmatization** is more proper process with the use of a 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 lemma are obtained by applying the suffix and prefix rules repeatedly and using the root words collection for verification.

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 .

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.

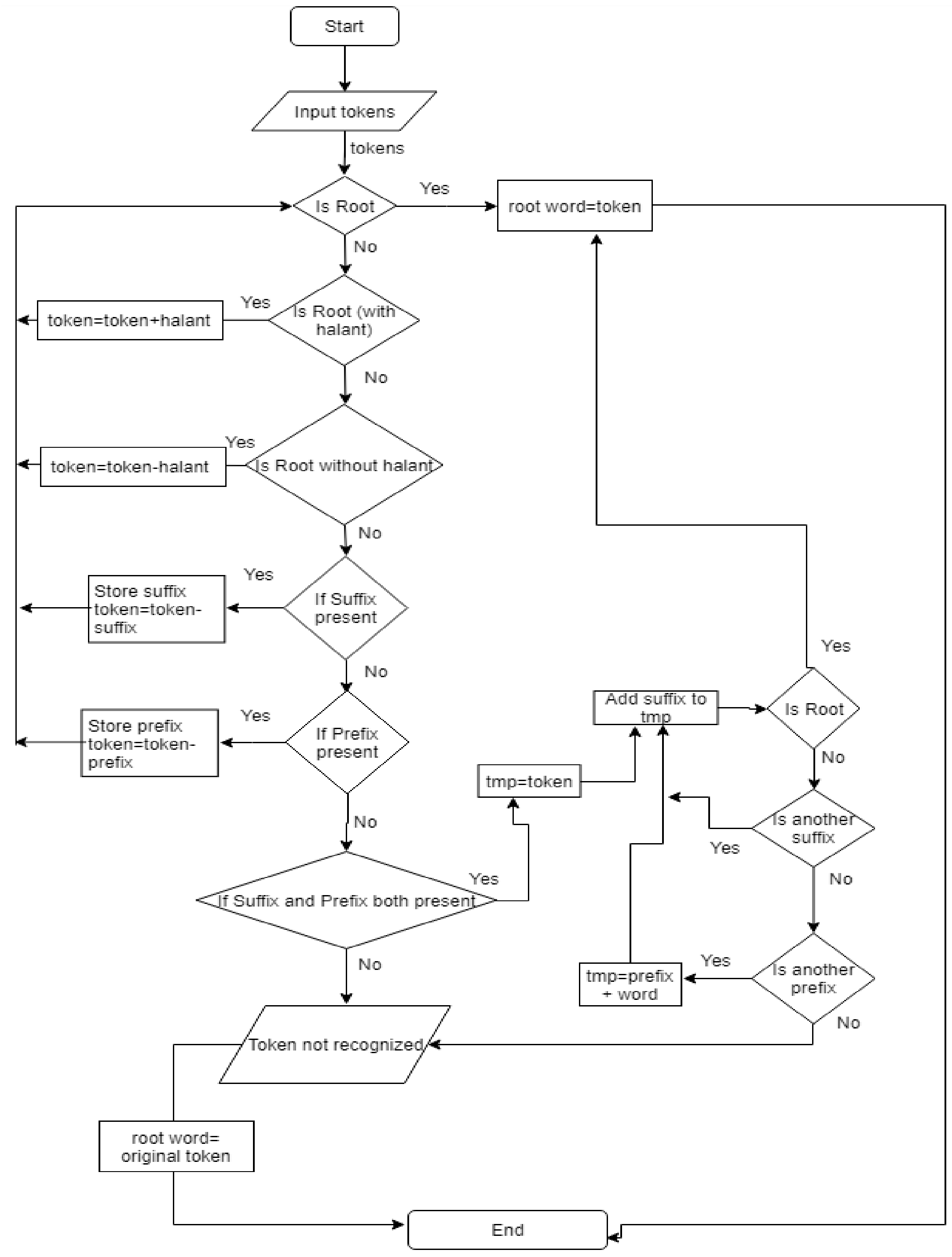
Example: खानुभयो खाए खाउ खान्छु to खा

Stemming and Lemmatization Algorithm:
Input: A list of preprocessed 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, goto step 2
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, goto step 3.

3. Check if 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, goto step 4.
4. Check whether any of the suffix is 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, goto step 5.
5. Check whether any of the prefix is 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, goto step 6.
6. If prefix and suffix both present, recombine the suffix one by one and check for lemma verification.
7. Repeat steps 1 to 7 until root word found or the token is unrecognized.
8. If the token is unrecognized, return the token without sriping else return the striped token as the lemma for the token.

Flow diagram:



Stemming and lemmatization algorithm

C. Feature Vector Construction

- **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 to a corpus, a collection of documents. The TF-IDF of a term is given by the equation:

  TF-IDF(term,document) = TF(term, document) * IDF(term)

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

  1. Normalized Term Frequency (tf)
  2. Inverse Document Frequency (idf)

  1. Normalized Term Frequency (tf)

  The term frequency (TF) measures how often a term shows up in a single document. On a large document the frequency of the terms will be much higher than the smaller ones. Hence we need to normalize the document based on its size. A simple trick is to divide the term frequency by the total number of terms.

  If a term appears relatively frequently in a document, it is most likely an important term.

  TF(term, document) = No. of times the term appears in document / total no. of terms in document

  2. Inverse Document Frequency (idf)

  The inverse document frequency (IDF) tells us how important a term is to a collection of documents. Certain terms that occur too frequently have little power in determining the relevance. We need a way to weigh down the effects of too frequently occurring terms. Also the terms that occur less in the document can be more relevant. We need a way to weigh up the effects of less frequently occurring terms. So, we calculate IDF for each term by ,

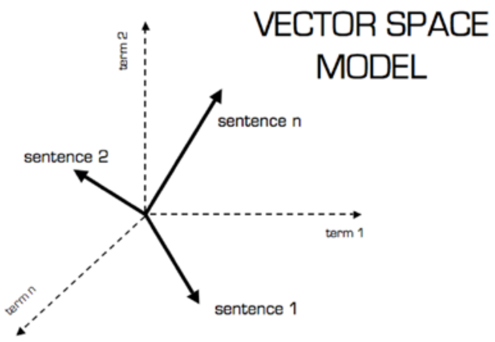
  IDF(term) = 1+log(total no. of documents / No. of documents with term in it)

  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 isn’t really special anymore, thereby producing a very low IDF.

  Now that we have our TF-IDF dictionaries, we can create our matrix. Our matrix will be an array of vectors, where each vector represents a review. The vector will be a list of frequencies for each unique word in the dataset—the TF-IDF value if the word is in the review, or 0.0 otherwise.

- **Construction of vector (vector space model)**

  For each document we derive a vector. The set of documents in a collection then is viewed as a set of vectors in a vector space. Each term will have its own axis. Now that we have our TF-IDF dictionaries, we can create our matrix. Our matrix will be an array of vectors, where each vector represents a document. The vector will be a list of frequencies for each unique word in the dataset—the TF-IDF value if the word is in the document, or 0.0 otherwise.



Vector Space Model

D. Calculating similarity

1. **Cosine similarity:**

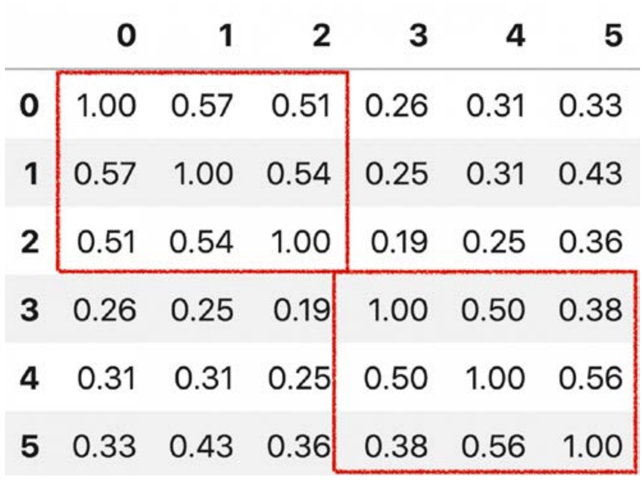
   It is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected into a multi-dimensional space. In this context, the two vectors are arrays containing the tf-idf value of each word in the two documents. 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 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. Using the formula given below we can find out the cosine similarity between any two documents.

$$Cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \, \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \sqrt{\sum_1^n b_i^2}}$$

where, $\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ is the dot product of the two vectors.

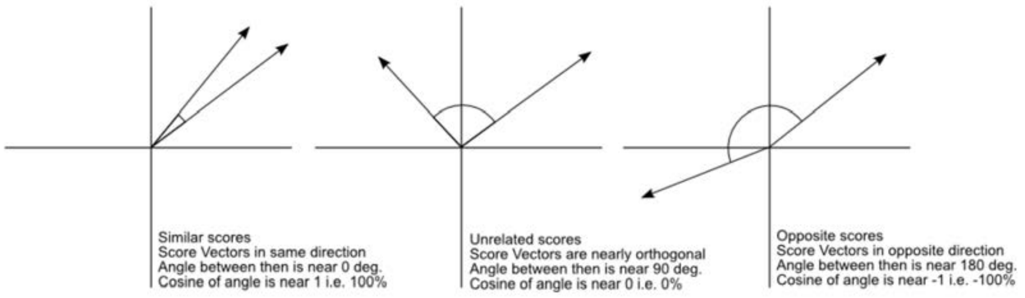
```
Cosine Similarity (d1, d2) =  Dot product(d1, d2) / ||d1|| * ||d2||

Dot product (d1,d2) = d1[0] * d2[0] + d1[1] * d2[1] * … * d1[n] * d2[n]
||d1|| = square root(d1[0]^2 + d1[1]^2 + ... + d1[n]^2)
||d2|| = square root(d2[0]^2 + d2[1]^2 + ... + d2[n]^2)
```



	0	1	2	3	4	5
0	1.00	0.57	0.51	0.26	0.31	0.33
1	0.57	1.00	0.54	0.25	0.31	0.43
2	0.51	0.54	1.00	0.19	0.25	0.36
3	0.26	0.25	0.19	1.00	0.50	0.38
4	0.31	0.31	0.25	0.50	1.00	0.56
5	0.33	0.43	0.36	0.38	0.56	1.00

As one might expect, the similarity scores amongst similar documents are higher (see the red boxes).


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%

The Cosine Similarity values for different documents, 1 (same direction), 0 (90 deg.), -1 (opposite directions).

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 that exists 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∩B=A∪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. For each string, define a set contains the characters in a 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 37.

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

E. Interface

We used Tkinter to design a simple interface. The interface allows to choose a folder containing a collection of Nepali text documents to be analysed for plagiarism by finding the cosine and Jaccard similarity metrics between them. The files are first read and preprocessed as per the steps discussed above and similarity is calculated. The resulting pair of documents are sorted as per the obtained cosine similarity and the pairs of files are displayed along with their Cosine similarity and Jaccard similarity.

4 Results and Limitations

Comparing the results obtained from Cosine similarity and Jaccard similarity for any pair of texts or documents we observed that Cosine similarity always yields better results. The obtained results are not perfect because each step was conducted in a lexical level. Due to lack of datasets it was difficult to detect plagiarism in semantic level.

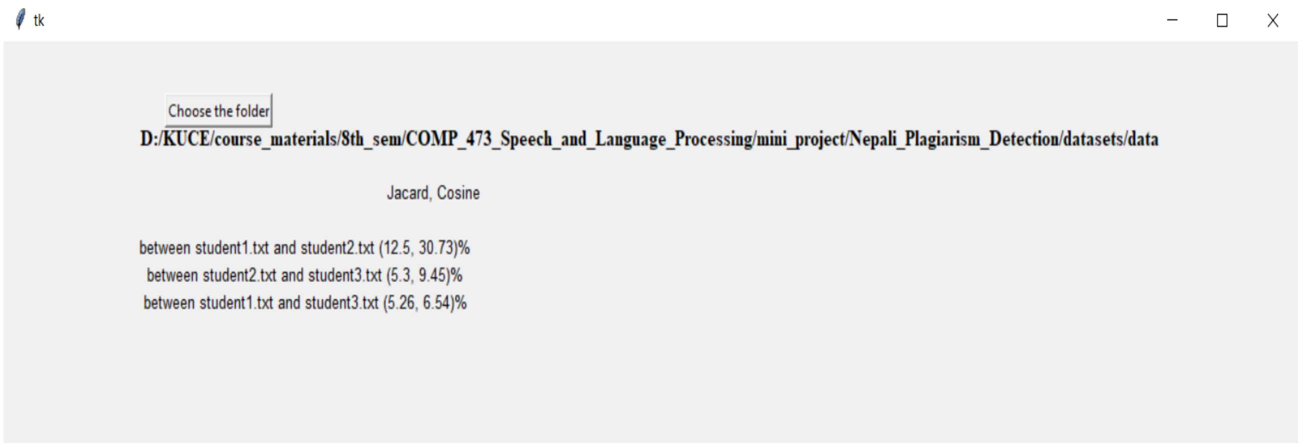
5 Future Plans

We plan to perform the similarity measures on a semantic level by comparing the similarities of the meanings of the obtained tokens instead of the similarities between the tokens. It is calculated using information from a dictionary. Hence, we need to find a corpus containing Nepali root words and their corresponding meanings. When similarity is measured in a semantic level, sentences having same/similar words give different meaning according to the order of the words used. Semantic similarity is also measured by counting the number of nodes of shortest path between two concepts.

6 Conclusion

Hence, in this paper we developed an approach to detect plagiarism in Nepali articles.With the help of cosine similarity we found the degree of similarity and dissimilarity between documents to know whether they are plagiarised or not. We used cosine similarity with tf-idf that is we first found the term frequency and inverse document frequency for each word and then did further calculation.

Appendix


tk

Choose the folder

D:/KUCE/course_materials/8th_sem/COMP_473_Speech_and_Language_Processing/mini_project/Nepali_Plagiarism_Detection/datasets/data

```
Jacard, Cosine

between student1.txt and student2.txt (12.5, 30.73)%
between student2.txt and student3.txt (5.3, 9.45)%
between student1.txt and student3.txt (5.26, 6.54)%
```

Fig 2: Interface for plagiarism detection

References:

1. Jatit.org. (2019). [online] Available at: http://www.jatit.org/volumes/Vol63No1/19Vol63No1.pdf [Accessed 11 Jun. 2019].

2. Citeseerx.ist.psu.edu. (2019). [online] Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.458.9440&rep=rep1&type=pdf [Accessed 11 Jun. 2019].

3. Machine Learning Plus. (2019). *Cosine Similarity - Understanding the math and how it works? (with python)*. [online] Available at: https://www.machinelearningplus.com/nlp/cosine-similarity/ [Accessed 11 Jun. 2019].

4. Vembunarayanan, J. (2019). *Tf-Idf and Cosine similarity*. [online] Seeking Wisdom. Available at: https://janav.wordpress.com/2013/10/27/tf-idf-and-cosine-similarity/ [Accessed 11 Jun. 2019].

5. Anon, (2019). [Blog].

6. Ww.panl10n.net. (2019). [online] Available at: http://ww.panl10n.net/english/final%20reports/pdf%20files/Nepal/NEP06.pdf [Accessed 11 Jun. 2019].

7. Ww.panl10n.net. (2019). [online] Available at: http://ww.panl10n.net/english/outputs/Working%20Papers/Nepal/Microsoft%20Word%20-%202_E_N_310.pdf [Accessed 11 Jun. 2019].

8. Nltk.org. (2019). *5. Categorizing and Tagging Words*. [online] Available at: https://www.nltk.org/book/ch05.html [Accessed 11 Jun. 2019].

9. Mylanguages.org. (2019). *Nepali Adverbs*. [online] Available at: http://mylanguages.org/nepali_adverbs.php [Accessed 11 Jun. 2019].

10. "Jaccard Coefficient - an overview | ScienceDirect Topics", *Sciencedirect.com*, 2019. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/jaccard-coefficient. [Accessed: 14- Jun- 2019]