

**TRIBHUVAN UNIVERSITY**

**INSTITUTE OF ENGINEERING**

**THAPATHALI CAMPUS**

**Mid-term Report on**

**On**

**Plagiarism Detection in Nepali Thesis**

**Submitted By:**

Ankit B.K. (THA076BCT006)

Ayush Batala (THA076BCT011)

Mishan Thapa Kshetri (THA076BCT019)

Nishant Uprety (THA076BCT023)

**Submitted To:**

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

January, 2024



**TRIBHUVAN UNIVERSITY**

**INSTITUTE OF ENGINEERING**

**THAPATHALI CAMPUS**

**Mid-term Report on**

**On**

**Plagiarism Detection in Nepali Thesis**

**Submitted By:**

Ankit B.K. (THA076BCT006)

Ayush Batala (THA076BCT011)

Mishan Thapa Kshetri (THA076BCT019)

Nishant Uprety (THA076BCT023)

**Submitted To:**

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

In partial fulfillment for the award of the Bachelor’s Degree in

Computer Engineering.

**Under the Supervision of**

Er. Saroj Shakya

January, 2024

# ACKNOWLEDGEMENT

We would like to express our sincere gratitude towards the Institute of Engineering, Tribhuvan University for the inclusion of major project in the course of Bachelors in Computer Engineering. We are also thankful towards our Department of Electronics and Computer Engineering for the proper orientation and guidance during the project **“Plagiarism Detection in Nepali Thesis”**

We would like to acknowledge the authors of various research papers and developers of various programming libraries and frameworks that we have referenced for developing our project.

We would like to express our gratitude towards our Project Supervisor Er. Saroj Shakya for continuous suggestions throughout. We would also like to express our appreciation towards Ram Raj Lohani sir for his guidance during the project.

Finally, we would like to thank all the people who are directly or indirectly related during our study and preparation of this project.

Ankit B.K (THA076BCT006)

Ayush Batala (THA076BCT011)

Mishan Thapa Kshetri (THA076BCT019)

Nishant Uprety (THA076BCT023)

# ABSTRACT

Plagiarism is the way of copying someone’s works entirely or modifying it by using similar word and claiming it as their own. It has been a major problem in academics and literature field from the past. This practice in detecting plagiarism in Nepali Language hasn’t been common as most of Nepali literature cannot be found in digitalized forms. This project aims to develop a training model to detect the plagiarism in thesis through Natural Language Processing (NLP). The system preprocesses the theses thoroughly and parse it into paragraphs then into sentences and tokens and used different mathematical metrics like modified Jaccard similarity and cosine similarity to assign the certain value of similarity to them. These values are used as features in training SVM model to detect plagiarism. For computing the word similarity, a library with the synonyms set and POS tag for the Nepali words using LSTM model is developed. This application is supposed to generate a report showing the plagiarized text along with the source document from which it is copied and the respective plagiarized percentage from each source.

*Keywords: LSTM, Nepali, NLP, Plagiarism, POS, Similarity, SVM, Thesis*

**TABLE OF CONTENTS**

[ACKNOWLEDGEMENT i](#_Toc155617191)

[ABSTRACT ii](#_Toc155617192)

[List of Figures vii](#_Toc155617193)

[List of Tables viii](#_Toc155617194)

[List of Abbreviations ix](#_Toc155617195)

[1. INTRODUCTION 1](#_Toc155617196)

[1.1 Background 1](#_Toc155617197)

[1.1.1 Natural Language Processing 1](#_Toc155617198)

[1.1.2 Support Vector Machine 1](#_Toc155617199)

[1.1.3 LSTM 2](#_Toc155617200)

[1.2 Motivation 2](#_Toc155617201)

[1.3 Problem Definition 2](#_Toc155617202)

[1.4 Objectives 2](#_Toc155617203)

[1.5 Project Application and Scope 3](#_Toc155617204)

[2. LITERATURE REVIEW 4](#_Toc155617205)

[3. REQUIREMENT ANALYSIS 8](#_Toc155617206)

[3.1 Feasibility Analysis 8](#_Toc155617207)

[3.1.1 Technical Feasibility 8](#_Toc155617208)

[3.1.2 Economic Feasibility 8](#_Toc155617209)

[3.1.3 Operational Feasibility 8](#_Toc155617210)

[4. DATASET PREPARATION AND ANALYSIS 9](#_Toc155617211)

[4.1 Dataset for Plagiarism Detection 9](#_Toc155617212)

[4.2 Collection of Nepali Theses from TUCL 9](#_Toc155617213)

[4.3 Dataset Analysis 10](#_Toc155617214)

[4.3.1 Complete Plagiarism 10](#_Toc155617215)

[4.3.2 Direct Plagiarism 10](#_Toc155617216)

[4.3.3 Paraphrasing Plagiarism 11](#_Toc155617217)

[4.3.4 Non-Plagiarized texts 12](#_Toc155617218)

[4.4 Techniques for Addressing Spelling Mistakes 14](#_Toc155617219)

[4.5 Techniques for Paraphrasing 14](#_Toc155617220)

[4.5.1 Reformulating the Sentence 15](#_Toc155617221)

[4.5.2 Combining or Elaborating ideas 15](#_Toc155617222)

[4.5.3 Removing Irrelevant Information 15](#_Toc155617223)

[4.5.4 Use of Synonyms 16](#_Toc155617224)

[4.6 Dataset for Parts of Speech Tagger 16](#_Toc155617225)

[5. METHODOLOGY 20](#_Toc155617226)

[5.1 NLP Approach 20](#_Toc155617227)

[5.2 Procedural Sequence 20](#_Toc155617228)

[5.2.1 Text Preprocessing 20](#_Toc155617229)

[5.2.2 Parts of Speech Tagger 21](#_Toc155617230)

[5.2.3 Model Building for POS Tagger 23](#_Toc155617231)

[5.2.4 Model Building for Plagiarism Detection 24](#_Toc155617232)

[5.3 Use Case Diagram 27](#_Toc155617233)

[5.4 System Architecture 28](#_Toc155617234)

[5.4.1 Feature Extraction 29](#_Toc155617235)

[5.4.2 TF-IDF with Smoothing 30](#_Toc155617236)

[5.4.3 Calculation of Word Similarity Value 31](#_Toc155617237)

[5.4.4 Calculation of Fingerprint Similarity Value 32](#_Toc155617238)

[5.4.5 Creation of Nepali Synset 33](#_Toc155617239)

[5.5 Flowchart of Proposed System 35](#_Toc155617240)

[5.6 Instrumentation Tools. 36](#_Toc155617241)

[5.6.1 Programming Language, Libraries and Frameworks 36](#_Toc155617242)

[6. IMPLEMENTATION DETAILS 38](#_Toc155617243)

[6.1 Dataset Preparation 38](#_Toc155617244)

[6.2 Text Extraction from Nepali thesis 38](#_Toc155617245)

[6.3 Text Preprocessing 39](#_Toc155617246)

[6.4 POS tagger 41](#_Toc155617247)

[6.5 LSTM Architecture 41](#_Toc155617248)

[6.5.1 Embedding Layer 41](#_Toc155617249)

[6.5.2 Dropout Layer 42](#_Toc155617250)

[6.5.3 LSTM Layer 42](#_Toc155617251)

[6.5.4 Dense Layer 42](#_Toc155617252)

[6.6 Post-Position Separation for POS tagger 43](#_Toc155617253)

[6.7 Feature Extraction 44](#_Toc155617254)

[6.8 Calculation of Similarity Score of a Document 46](#_Toc155617255)

[6.9 Hyperparameter Tuning of SVM model 47](#_Toc155617256)

[6.10 Hyperparameter Tuning of LSTM model 49](#_Toc155617257)

[7. RESULTS AND ANALYSIS 51](#_Toc155617258)

[7.1 Hyperparameter Tuning Report of SVM model 51](#_Toc155617259)

[7.2 Accuracy with Features 52](#_Toc155617260)

[7.3 POS tagger 54](#_Toc155617261)

[7.3.1 Hyperparameter Tuning Report of POS Model 55](#_Toc155617262)

[7.3.2 Performance Metrics of POS tagging Model 55](#_Toc155617263)

[7.3.3 Loss Curve for LSTM Model 57](#_Toc155617264)

[7.4 Nepali Synset 58](#_Toc155617265)

[7.5 Plagiarism Report 59](#_Toc155617266)

[8. REMAINING TASK 61](#_Toc155617267)

[8.1 Web Application Deployment 61](#_Toc155617268)

[8.2 Database Optimization 61](#_Toc155617269)

[9. APPENDICES 62](#_Toc155617270)

[Appendix A: Project Timeline 62](#_Toc155617271)

[Appendix B: Nepali Dataset Snippets 63](#_Toc155617272)

[Appendix C: Code Snippets 64](#_Toc155617273)

[References 65](#_Toc155617274)

List of Figures

[Figure 4‑1: Collection of Nepali theses 9](#_Toc155617023)

[Figure 4‑2: Dataset in XML format 17](#_Toc155617024)

[Figure 4‑3: Refined dataset 17](#_Toc155617025)

[Figure 5‑1: Steps involved in Preprocessing 20](#_Toc155617026)

[Figure 5‑2: Block Diagram of LSTM Architecture 23](#_Toc155617027)

[Figure 5‑3: Detailed Diagram of LSTM Architecture 23](#_Toc155617028)

[Figure 5‑4: POS Tagger using LSTM 24](#_Toc155617029)

[Figure 5‑5: Hyperplane 26](#_Toc155617030)

[Figure 5‑6: SVM model 26](#_Toc155617031)

[Figure 5‑7: Use Case Diagram 27](#_Toc155617032)

[Figure 5‑8: System Architecture 28](#_Toc155617033)

[Figure 5‑9: Block Diagram showing TF-IDF Calculation 31](#_Toc155617034)

[Figure 5‑10: Calculation of Word Similarity Value 32](#_Toc155617035)

[Figure 5‑11: Calculation of Fingerprint Similarity 33](#_Toc155617036)

[Figure 5‑12: Synset Preparation Flowchart 34](#_Toc155617037)

[Figure 5‑13: Flowchart of Proposed System 35](#_Toc155617038)

[Figure 6‑1: Extraction of Text from digitalized form 39](#_Toc155617039)

[Figure 6‑2: Calculation of Similarity Score for a Documents 46](#_Toc155617040)

[Figure 7‑1: Accuracy vs Feature Graph 52](#_Toc155617041)

[Figure 7‑2: POS Tagger Output Interface 54](#_Toc155617042)

[Figure 7‑3: Performance Metrics 56](#_Toc155617043)

[Figure 7‑4: Training vs Validation Loss 57](#_Toc155617044)

[Figure 7‑5: User Interface of Nepali Synset 58](#_Toc155617045)

[Figure 7‑6: Color Coded Plagiarized text 59](#_Toc155617046)

[Figure 7‑7: Plagiarism Report 60](#_Toc155617047)

# List of Tables

[Table 4‑1: Number of Datasets 13](#_Toc155617048)

[Table 4‑2: Generalization of Tagset 18](#_Toc155617049)

[Table 7‑1: Parameter Tuning Report 51](#_Toc155617050)

[Table 7‑2: Hyperparameter of LSTM model 55](#_Toc155617051)

[Table 9‑1: Gantt Chart 62](#_Toc155617052)

# List of Abbreviations

|  |  |
| --- | --- |
| CSS | Cascading Style Sheet |
| CSV | Comma Separated Value |
| GRU | Gated Recurrent Unit |
| HTML | Hyper Text Markup Language |
| IOE | Institute of Engineering |
| JSON | JavaScript Object Notation |
| LSTM | Long Short-Term Memory |
| MCANN | Monte Carlo Based Artificial Neural Network |
| ML | Machine Learning |
| MNN | Multi Neural Network |
| NLP | Natural Language Processing |
| NLTK | Natural Language Toolkit |
| NNC | Nepali National Corpus |
| PDF | Portable Document Format |
| POS | Parts of Speech |
| RNN | Recurrent Neural Network |
| SVD | Singular Value Decomposition |
| SVM | Support Vector Machine |
| TF-IDF | Term Frequency-Inverse Document Frequency |
| TU | Tribhuvan University |
| TUCL | Tribhuvan University Central Library |
| XML | Extensible Markup Language |
|  |  |
|  |  |
|  |  |
|  |  |

# INTRODUCTION

## Background

The origin of the word plagiarism is supposed to be from a Latin word “plagiarius” (literally meaning kidnapping) which denotes someone stealing someone else’s creative work [1]. Plagiarism is like counterfeiting. In academics it is a serious ethical offense. Plagiarism is presenting someone else’s work without their consent. Plagiarism occurs in varieties of context including books, music, software, academic papers, journals etc. The ease of information sharing using internet has encouraged searching for literature as well as other ideas online to replicate other’s work as their own.

With more people using internet maintaining academic integrity in school and institution is becoming increasingly difficult. However, the success of these systems for detecting plagiarism based on their capacity to identify various frauds for modifying the texts without altering its meaning has been questionable ever since.

### Natural Language Processing

Natural Language Processing (NLP) is a component of Artificial Intelligence which enable computers comprehend natural (human) language. NLP includes computer science as well as computational linguistics to ensure communication between human and machine. [2] NLP involves ability to understand text and spoken words by the computers. Only few NLP works in Nepali language has been performed mostly due to the increased complexity and lack of available resources which isn’t sufficient.

### Support Vector Machine

Support Vector Machine is a machine learning algorithm used mostly for classification task. SVM helps to classify the inputs based on statistical analysis performed on training data. SVM is a supervised type of machine learning algorithm i.e., it needs human supervising the learning process. An SVM training algorithm builds a model that tries to predict the classification of new unknown data. SVM generalizes the problem and acts on it. The project uses this algorithm to classify texts into two classes either plagiarized or not.

### LSTM

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN). It is specifically designed to handle sequential data, such as time series, speech, and text. LSTM networks are capable of learning long-term dependencies in sequential data, which makes them well suited for Natural Language Processing. LSTM is specifically designed to address the exploding/vanishing gradient problem and capture long-term dependencies in sequential data. It introduces memory cells and gating mechanisms that allow it to retain and update information over longer sequences. The project uses this algorithm to classify Nepali words into two different parts of speech.

## Motivation

In universities and colleges, many students refer to many books and internet for their work, sometimes they just find easy to use someone’s work/idea without properly crediting them especially in Nepali language where this practice hasn’t been observed. This threatens the integrity of someone’s honest work and provide false credentials to an individual for actions they did not perform.

Plagiarism Detection techniques in English language are common due to the increased complexity and lack of available resources it’s not commonly practiced in Nepali language. So, It has been decided to work on a Plagiarism detection tool for Nepali language

## Problem Definition

The problem is the lack of Natural Language Processing resources in Nepali language. This project aims to develop a library for POS tagging and synonyms for Nepali words. A model which detects plagiarism effectively in Nepali theses. A model for POS tagging is needed to be trained as well as another model for checking the similarity between the texts.

## Objectives

The primary objectives of our project are

* To develop a model for POS tagging for Nepali words
* To build a plagiarism detection software.

## Project Application and Scope

Plagiarism is one of the major issues in academics and journalism. In academics, students’ and professors’ faces suspension and may losses their license if their documents are found to be plagiarized. While in journalism, reporters’ work needs to be trustful and if they are found to copy the work of others, they may face legal charges. So, plagiarism detection algorithm can be used in those cases to verify their works.

The applications of plagiarism detection are:

* Plagiarism detection in Nepali academics
* Verification of research work done by the students and professors in academics.
* Approval of honest work from reporter in journalism.

The scopes of plagiarism detection are:

* In verification of authenticity of research works
* Approval of honest work in Journalism
* Fails to identify common knowledge and observable facts
* Doesn’t consider diagrams, charts or other visual materials.

# LITERATURE REVIEW

TF-IDF is a well-established technique in the field of NLP and information retrieval. The higher the occurrence of a word in documents gives higher term frequency and the less occurrence of word in documents yields higher importance (IDF). It measures significance of a term in the respective document. Using original TF-IDF, some of the terms gets ignored i.e., value for some term becomes zero. However, the value of a sequence of words should not be zero at all. This is where various different smoothing techniques come into the picture. When the dataset is small and the documents are relatively long, a simpler smoothing method like Laplace or Jelinek-Mercer smoothing may be appropriate. [3] [4]

In 2004, A report on Nepali Grammar concluded a brief overview of the sentential structure of Nepali Language with illustration. The parts of speech of Nepali language were studied following it by a detailed analysis on the phrase structure of Nepali Grammar on the paper which could be the basis for POS tagging in Nepali language. [5]

In 2008, Zdenek Ceska proposed a new plagiarism tool: SVDPlag which employs Singular Value Decomposition. To examine the efficiency, the experiment used corpus of nine hundred and fifty text documents and indicated that this method improved the accuracy. [6]

In 2009, Alberto Barron-Cedeno and Paolo Rosso proposed a research paper On Automatic Plagiarism Detection Based on n-grams Comparison in which suspicious and reference document are taken and suspicious document are split into sentences and reference documents are divided into n-grams and checked into n-grams level in the range from 1 to 10. The result showed that using only n=2 and 3 i.e. bigram and trigram yielded the best result. [7]

In 2013, Research in lemmatization had been conducted which was able to obtain the total accuracy of 70.10%. This algorithm was used for Nepali Language and has given better result in comparison to rule-based system. The main reason for lower accuracy was due to lack of availability of corpus of Nepali data during the research. [8]

In 2018, Ram Bhakta Bahachan and Arun Kumar Timilsina proposed a Nepali Plagiarism Detection Framework using Monte Carlo Based Artificial Neural Network (MCANN). In this paper, authors collected different Nepali documents from different sources and passed it in the framework. They applied Cosine Similarity and Jaccard Similarity between each paragraph from the source and suspicious data. The mean accuracy for MCANN was found to be in the bracket of 98.657 and 99.864% in the comparison of the documents. [9]

In 2019, Sanjeev Subba and team implemented deep learning Recurrent Neural Network (RNN) for Nepali text classification and compared the performance with traditional approach Multi Neural Network (MNN). Data was collected from different online Nepali News portals. Twenty different classes data were collected but only business and interview class data were used as it had larger size than others. They used stemming (Usual & Gunal, 2014) in preprocessing. Ten experiments were performed, five for RNN and five for MNN. The experiment results showed the accuracy of RNN outperform the accuracy of MNN. The highest accuracy of MNN was 48% and the highest accuracy of RNN was 63%. They suggested for more accuracy in neural model data collection should be large. They also suggested lack of proper stemming and lemmatization in Nepali causes the neural model to fail to achieve high accuracy. [10]

In 2019, Nova Eka Dianna and Ikrima Hanana Ulfa proposed a paper Measuring Performance of N-Gram and Jaccard-Similarity Metrics in Document Plagiarism Application which employed N-grams and Jaccard to check the document similarity. In this research, they found out that the performance given by Jaccard along with N-gram method with the average similarity were 100% and 83.52% respectively. [11]

In 2019, Winda Kurnia Sari and team made a study on text classification using Long Short-Term Memory (LSTM) by tuning parameters and comparing the eight LSTM models. The hyper-parameters used are the Relu and Tanh activation functions, Adam and RMSProp optimizers with a learning rate of 0.001 and 0.0001. The output activation function, loss function and epochs were constant. AGNEWS dataset was used for taking the model. AGNEWS dataset contained classification of topics in four categories/classes: World, Entertainment, Sports, Business. It was concluded that the model with optimizer Adam, learning rate, 0.0001 hidden activation function tanh had the best accuracy of 95.17 among eight models. And they suggested that the LSTM model with GloVe feature can achieve good performance. [12]

A considerable amount of work has already been done in the field of POS tagging for English. Different approaches like the rule-based approach, the stochastic approach and the transformation-based learning approach along with modifications have been tried and implemented. However, if it is looked at the same scenario for South-Asian languages such as Bangla, Hindi, and Nepali, it is found that not much work has been done.

In 2019, Bal Krishna Bal presented a rule-based recursive stemming algorithm for Nepali language. In this paper, authors created a stemming and lemmatization algorithm to pre-process Devanagari scripts more effectively. They also made the use of Jaccard Similarity in order to calculate similarity between the token of the preprocessed documents. The obtained accuracy and precision of the document at lexical level was found to be 95% and 93.33%. [13]

In 2020, Sarbin Sayami and Subarna Shakya implemented and compared various deep learning approaches for POS tagging in Nepali Language. Recurrent Neural Network (RNN), Gated Recurrent Unit (GRU), Long Short-Term Memory (LSTM) and Bidirectional LSTM were the deep learning approaches that were implemented. They suggested to deep learning approach as Nepali language is morphologically rich, rule based and statistical techniques do not take care of context and sequence. Corpus consisting of 40 Tagset and 88000 words were used as dataset. Among these models, Bidirectional LSTM showed highest accuracy of 97.27% as they were able to understand context better due to its ability to approach a unit from both the directions. [14]

In 2020, Research titled Semantic Plagiarism Detection System for English Text proposed that they have created a system considers semantic similarity, paraphrase detection. The model was trained using Microsoft Research Paraphrase Corpus which give an accuracy of 69.33%. [15]

Plagiarism endangers the genuinely of educational process as students may receive credit for something they have not done. Some of the plagiarism detection tools available are Turnitin, Eve2, CopyCatGold, etc. [16]. However, there are no detection tools available to check plagiarism for Nepali Language documents.

Christopher Burges proposed extrinsic plagiarism detection using SVM in English Language. In this paper, they have used features like word pairs, word similarity, fingerprint similarity, LSA similarity. First the preprocessing of data then followed by feature extraction took place. Then, the training data is classified in the sub category It uses the input data the features’ values that are calculated in the feature extraction process. The algorithm performs classification using the model. The result of the classification is if the new input text is plagiarized or not. The accuracy created using Support Vector Machine was found to be 84.375%. [17]

# REQUIREMENT ANALYSIS

## Feasibility Analysis

After gathering of the required resources, whether the completion of the project with the gathered resource is feasible is checked using the following feasibility analysis.

### Technical Feasibility

Plagiarism Detection in Nepali Thesis is a web application that uses Django, a python- based framework. It uses HTML, Bootstrap CSS, JavaScript to design frontend and Python as the backend. It requires a server, client and Internet connection to function properly.

It supports both Windows and Linux platform for its operation. All of the technology required by the application can be accessed and available freely. This project uses Natural Language Processing, Machine Learning and Database Management System. The resources can be available freely with technical support. Hence, it is determined technically feasible.

### Economic Feasibility

In our project, no additional costs is expected to be spent on as in this program only the open-source resources are used which are easily available. The expenses incurred in the project are mostly indirect expenses. Personal devices are used to build and test the system and personal internet subscription to do the research online. However, some direct expenses is expected incur in the acquisition of data and resources for NLP tasks. Hence, considering the costs and benefits of completing the project, the project is deemed economically feasible.

### Operational Feasibility

Plagiarism Detection for Nepali Thesis has a simple design and is easy to use. It can be easily accessed by any user connected to internet, and can be used to check for Plagiarism in the input documents or to further use the application for various NLP applications. Hence, Plagiarism Detection in Nepali Thesis is determined to be operationally feasible

# DATASET PREPARATION AND ANALYSIS

## Dataset for Plagiarism Detection

It is mainly focused on detection of external plagiarism i.e., when both source text and suspicious text are present using Support Vector Machine. Thousands of annotated Nepali texts are needed to train the model. Due to complexity of task and lack of digitalized resources, plagiarism detection has not been performed in Nepali language [18]. The process of creating more dataset is taken parallelly along with other model development and training.

## Collection of Nepali Theses from TUCL

Due to lack of practice of plagiarism detection in Nepali literature no annotated dataset were found. The corpora of Nepali language thesis by student of Department of Nepali Education as requirement for fulfillment of master’s degree is used for this. The total of 452 different theses in Nepali language for annotation is used.

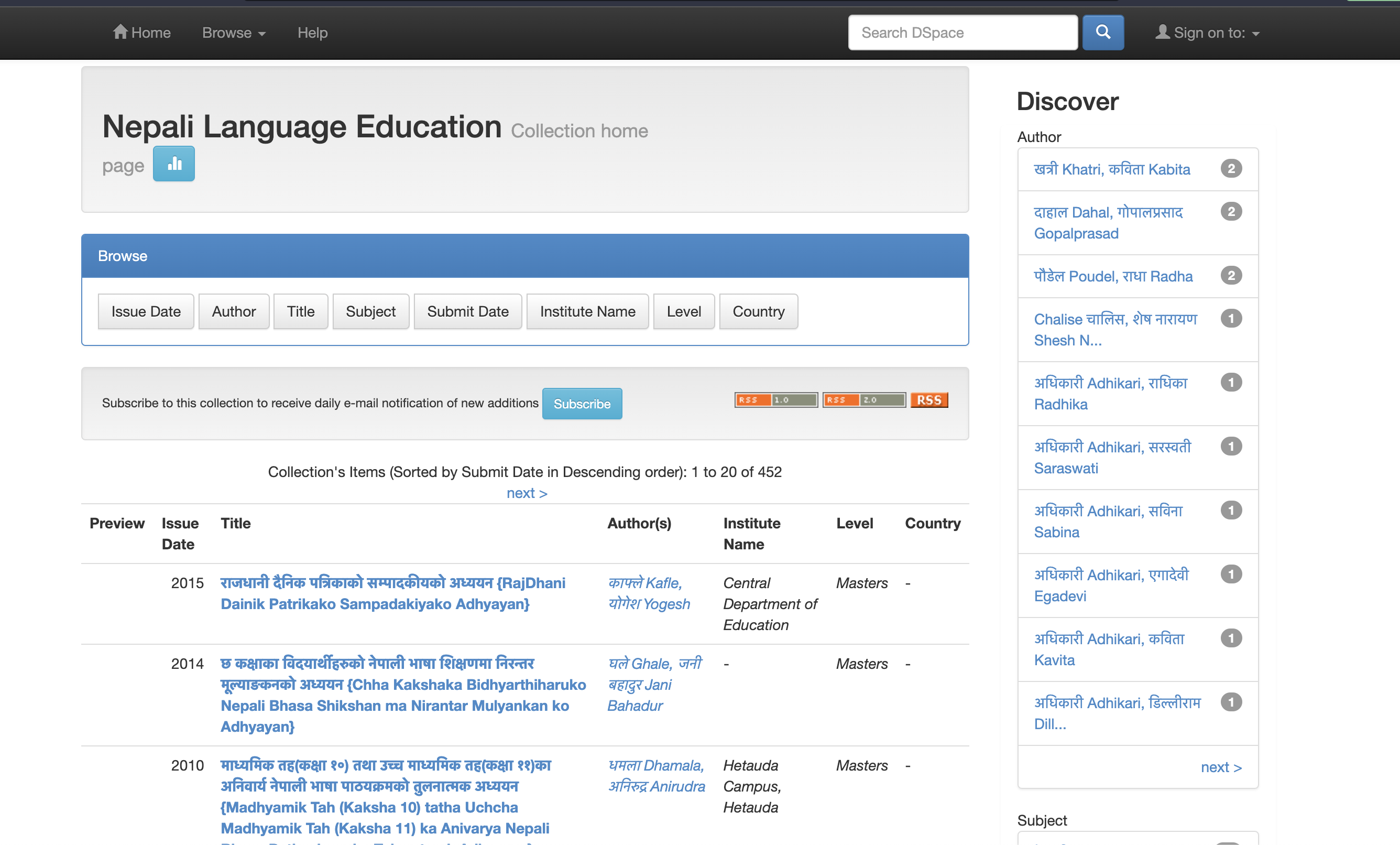


Figure 4‑1: Collection of Nepali theses

## Dataset Analysis

The dataset of following type of plagiarism were prepared for training the model using Support Vector Machine.

### Complete Plagiarism

It refers to copying the entirety of someone's work. "It is equivalent to intellectual theft and stealing." This is the most easily detectable type of plagiarism by the system as there is no changes from the original work. One thousand of such paragraphs with complete plagiarism is prepared.

Example:

Original Text:

फणीन्द्रराज खेतालाको जन्म वि.सं. १९७९ असोज १३ गते बडादसैंको महानवमीका दिन पिता भुवनराज भट्टराई र माता दिव्य कुमारीको कोखबाट काठमाडौंको पकनाजोल सल्लाघारीमा भएको हो । थर भट्टराई हटाएर खेताला' उपनाम लेख्ने गरेका फणीन्द्रराज खेतालाले साहित्य, शिक्षा, समाज सेवाका क्षेत्रमा उल्लेख्य योगदान दिएका छन्‌ ।

Plagiarized Text:

फणीन्द्रराज खेतालाको जन्म वि.सं. १९७९ असोज १३ गते बडादसैंको महानवमीका दिन पिता भुवनराज भट्टराई र माता दिव्य कुमारीको कोखबाट काठमाडौंको पकनाजोल सल्लाघारीमा भएको हो । थर भट्टराई हटाएर खेताला' उपनाम लेख्ने गरेका फणीन्द्रराज खेतालाले साहित्य, शिक्षा, समाज सेवाका क्षेत्रमा उल्लेख्य योगदान दिएका छन्‌ ।

### Direct Plagiarism

Direct or verbatim plagiarism occurs when someone replicates exact sentence word by word. It is similar to complete plagiarism in a way but only contains a portion of original text. Sixteen hundreds of paragraphs with direct plagiarism is prepared

Example:

Original text:

फणीन्द्रराज खेतालाको जन्म वि.सं. १९७९ असोज १३ गते बडादसैंको महानवमीका दिन पिता भुवनराज भट्टराई र माता दिव्य कुमारीको कोखबाट काठमाडौंको पकनाजोल सल्लाघारीमा भएको हो । थर भट्टराई हटाएर खेताला' उपनाम लेख्ने गरेका फणीन्द्रराज खेतालाले साहित्य, शिक्षा, समाज सेवाका क्षेत्रमा उल्लेख्य योगदान दिएका छन्‌ ।

Plagiarized text:

फणीन्द्रराज खेतालाको जन्म वि.सं. १९७९ असोज १३ गते बडादसैंको महानवमीका दिन पिता भुवनराज भट्टराई र माता दिव्य कुमारीको कोखबाट भएको हो | राणातन्त्रको विरोध गर्दै राजनीतिमा भाग लिन पुगेका खेतालाले काराबासको सजाय समेत झेलेको पाइन्छ । वि.सं. १९९४ को “गोरखापत्र मा पैसा' शीर्षकको कविता प्रकाशित गरी सार्वजनिक रूपमा साहित्यिक यात्राको थालनी गरेका खेतालाले खास गरी कविता, नाटक तथा एकाङ्की विधामा उल्लेख्य सफलता हासिल गरेका छन्‌ ।

### Paraphrasing Plagiarism

This is another most type of plagiarism. It is using other’s work with some changes in sentence structure and words. Even if the words look completely new, the original idea remains the same and plagiarism occurs. This type of plagiarism detection requires analysis of complete sentiment of the text this would be most difficult to recognize and complexity is high. One thousand nine hundred and eighteen different paragraphs with paraphrased texts are prepared to train the model.

Example:

Original text:

फणीन्द्रराज खेतालाको जन्म वि.सं. १९७९ असोज १३ गते बडादसैंको महानवमीका दिन पिता भुवनराज भट्टराई र माता दिव्य कुमारीको कोखबाट काठमाडौंको पकनाजोल सल्लाघारीमा भएको हो । थर भट्टराई हटाएर खेताला' उपनाम लेख्ने गरेका फणीन्द्रराज खेतालाले साहित्य, शिक्षा, समाज सेवाका क्षेत्रमा उल्लेख्य योगदान दिएका छन्‌ ।

Plagiarized text:

फनिन्द्रराज खेतलाको जन्म १३ अक्टोबर १९७९ मा काठमाडौंको सल्लाघारीमा भएको थियो । उनी भुवनराज भट्टराई र दिव्या कुमारीका छोरा हुन् । खेतलाले सन् १९९४ मा आफ्नो थर परिवर्तन गरेर खेताला राखे र त्यसपछि उनले थुप्रै कविता र उपन्यास प्रकाशित गरिसकेका छन्।

### Non-Plagiarized texts

SVM can still perform well with imbalanced class proportions but having balanced class proportions can lead to more reliable and accurate results so four thousand five hundreds and eighteen such non plagiarized texts are prepared to be fed into the model for training.

Example:

Original text:

फणीन्द्रराज खेतालाको जन्म वि.सं. १९७९ असोज १३ गते बडादसैंको महानवमीका दिन पिता भुवनराज भट्टराई र माता दिव्य कुमारीको कोखबाट काठमाडौंको पकनाजोल सल्लाघारीमा भएको हो ।

Plagiarism free text:

खेतालाका सबै कृतिका बारेमा त्यति समीक्षा र टिप्पणी भएको नदेखिए पनि केही रचना र कृतिहरूका बारेमा उल्लेख भएका टिप्पणी र तिनका समीक्षालाई यहाँ प्रस्तुत गरिएको छ |

Table 4‑1: Number of Datasets

|  |  |
| --- | --- |
| **Types of Plagiarism** | **Number of Paragraphs** |
| Complete Plagiarism | 1000 |
| Direct Plagiarism | 1600 |
| Paraphrasing Plagiarism | 1918 |
| Non-Plagiarized | 4518 |
|  | Total=8736 |

## Techniques for Addressing Spelling Mistakes

Certain paragraphs within a thesis may contain words with spelling errors. Therefore, it is essential to account for these types of mistakes when evaluating potential plagiarism in the dataset. Consequently, it is imperative to construct a dataset that accommodates the potential occurrence of such errors during plagiarism checks. These types of plagiarized paragraph will be identified by fingerprint feature.

To address these spelling mistakes within the dataset, two distinct approaches have been employed. Firstly, certain letters and punctuations have been substituted with alternative wrong ones. Secondly, certain letters have been omitted. Through the application of these methods, a total of 200 plagiarized texts and 200 non-plagiarized texts have been generated.

Example:

Original text:

यो एक उदाहरण पैराग्राफ हो। यसमा कई वाक्यहरू छन्। प्रत्येक वाक्यलाई केहि अक्षरहरूलाई यादृच्छिक रूपमा हटाइनेछ।

Text after introducing spelling error:

यो एक उदाहण पैराग्रफ हो। यसकमा कई वाक्यरू छन् प्रत्येक वाक्यलाईक केहि अक्षरहरूलाई यादृच्छिक रूपम हटाइनछ।

## Techniques for Paraphrasing

Paraphrasing is restating another author’s idea or word while keeping the meaning same. There are many techniques to paraphrase a text. This project focuses on three of the most common ways of paraphrasing namely:

### Reformulating the Sentence

The most common method of reformulating the sentence is by changing the voice of the sentence i.e., active to passive and vice versa. This method jumbles the words in the text without changing the words but just the forms of verb and noun phrase.

Original Text: रामले मलाई बोलायो ।

Paraphrased Text: रामद्वारा म बोलाइएँ ।

### Combining or Elaborating ideas

Two or more simple sentences can be combined to form compound and complex sentences giving the same idea. Likewise, a complex sentence can be broken down to simple sentences to try to avoid plagiarism. Both of these techniques for preparation of dataset are used.

Original Text:

म स्याउ खान्छु।म आँप खान्छु।

Paraphrased Text:

म स्याउ र आँप खान्छु।

### Removing Irrelevant Information

Another method of paraphrasing text is by only using relevant information and excluding words or even sentences which might seem irrelevant. The words which seem irrelevant to reader are excluded from the paraphrased text.

Original Text:

एउटा बिरालो सोफामा बसिरहेको छ।दिन घाम लागेको थियो।बिरालो अल्छी थियो।सोफा नरम थियो।

Paraphrased Text:

घमाइलो दिनमा एउटा अल्छी बिरालो नरम सोफामा बसिरहेको छ।

### Use of Synonyms

This is considered the most effective way of paraphrasing a text. The use of synonyms which gives similar idea without changing the meaning of text is a popular method used to avoid plagiarism.

Original Text:

मलाई स्याउ मन पर्छ।

Paraphrased Text:

मलाई स्याउ खान रमाइलो लाग्छ।

## Dataset for Parts of Speech Tagger

The Nepali Monolingual written corpus is part of the Nepali National Corpus. The Nepali National Corpus was produced in 2006 in the framework of the project Bhasha Sanchar also known as Nelralec.The written corpus is morphogically-annotated. A part-of-speech (POS) tagset has been produced within the project: the Nelralec Tagset. In this corpus, the tokens are appropriately-sized units for morphosyntactic analysis rather than orthographic tokens.

The written corpus is in XML format where a paragraph from the text is enclosed with <p> tag, the sentences in the paragraph are enclosed with <s> tag, and the words in the sentences are enclosed with <w> tag with its POS tag specified as the value of attribute ‘ctag’. There are a total of 112 POS tags in the corpus denoted by roman alphabetic symbols and an index is maintained for the symbols.



Figure 4‑2: Dataset in XML format

To create a POS tagging model for Nepali, the process begins by collecting the Nepali National Corpus (NNC), which consists of approximately 13 million tagged words. It is followed by refining this dataset by removing duplicate sentences and correcting any inaccuracies. Finally, it is organized in CSV file as shown in fig below. This curated dataset served as the foundation for training our LSTM model, enabling it to accurately assign POS tags to Nepali words.

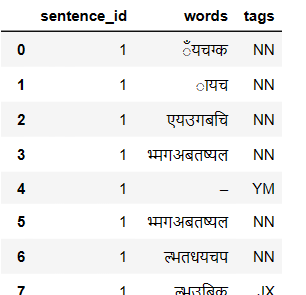


Figure 4‑3: Refined dataset

This dataset comprises approximately 112 parts-of-speech (POS) tags. However, for our application, such fine-grained specificity in POS tagging is not required for the project. Therefore, to simplify and generalize the 112 POS tags into common categories, as illustrated in the following table.

Table 4‑2: Generalization of Tagset

|  |  |
| --- | --- |
| **Generalized POS Tags** | **NELRALEC Tags** |
| Noun | NN, NP |
| Pronoun | PMX, PTN, PTM, PTH, PXH, PXR, PMXKM, PMXKF, PMXKO, PTNKM, PTNKF, PTNKO, PTMKM, PTMKF, PTMKO, PRFKM, PRFKF, PRFKO, PMXKX, PTNKX, PTMKX, PRFKX, PRF |
| Determiner | DDM, DDF, DKM, DKF, DJM, DJF, DGM, DGF, DDO, DKO, DJO, DGO, DDX, DKX, DJX, DGX, RD, RK, RJ |
| Verb | VVMX1, VVMX2, VVTN1, VVTX2, VVYN1, VVYX2, VVTN1F, VVTM1F, VVYN1F, VVYM1F, VOMX1, VOMX2, VOTN1, VOTX2, VOYN1, aVOYX2, VI, VN, VDM, VDF, VDO, VDX, VE, VQ, VCN, VCM, VCH, VS, VR |
| Adjective | JM, JF, JO, JX, JT |
| Adverb | RR |
| Postposition | II, IH, IE, IA, IKM, IKO, IKF, IKX |
| Numerals | MM, MOM, MOF, MOO, MOX |
| Classifier | MLM, MLF, MLO, MLX |
| Conjunction | CC, CSA, CSB |
| Interjection | UU |
| Question Marker | QQ |
| Particle | TT |
| Punctuation | YF, YM, YQ, YB |
| Foreign Word | FF, FS, FO, FZ |
| Unclassifiable | FU |
| Abbreviation | FB |
| NULL Tag | NULL |

# METHODOLOGY

## NLP Approach

Natural Languages Processing refers to processing of human understandable language by the machines. NLP involves ability to understand text and spoken words by the computers. Only few NLP works in Nepali language has been performed mostly due to the increased complexity and lack of available resources which isn’t sufficient for our work.

## Procedural Sequence

### Text Preprocessing



Figure 5‑1: Steps involved in Preprocessing

The general preprocessing techniques used on the text were

* Parsing Paragraphs into words: The text was initially segmented into discrete sentences, and then each sentence was further divided into individual words. This process, known as tokenization, transformed the flow of text into a series of tokens or words. These tokens were then treated as vector quantities, which allowed for mathematical operations to be performed on the textual data. This step is foundational in preparing the text for various NLP tasks, as it converts unstructured text into a structured form.
* Removal of Punctuation: During preprocessing, any unnecessary punctuation that did not contribute to the meaning of the text was removed. Punctuation can introduce noise in certain NLP tasks, and its removal simplifies the dataset by eliminating characters that are not needed for understanding the semantic content of the text.
* Parts of Speech tagging: Once the text was tokenized, each token was assigned a grammatical tag corresponding to its part of speech, such as noun, verb, adverb, etc. This process is known as parts of speech tagging and provides additional structure to the tokens, which is particularly useful for tasks that require understanding the grammatical structure of sentences.
* Stemming and Lemmatization: The text underwent two processes to reduce words to their base or root form. Stemming simplified words to their stems, enabling a more generalized comparison analysis by reducing different forms of a word to a common stem. For instance, the words "riding," "rides," and "ridden" were all reduced to the stem "ride." Lemmatization, a more nuanced process, involved normalizing words by considering their part of speech and context to ensure that the root word (lemma) reflects the underlying meaning. For example, comparative and superlative adjectives like "better" and "best" were normalized to their base form "good."
* Similarity Adjustment:The semantic analysis of the text involved measuring the similarity between words. This was done by looking for synonyms and considering the context provided by the parts of speech tags. This similarity adjustment is crucial for understanding the meaning conveyed by the text and for tasks such as detecting plagiarism, where paraphrasing can be used to mask copied content.

Each of these steps was critical in the preprocessing phase to prepare the text for further analysis, such as plagiarism detection. By breaking down the text into a structured format, removing unnecessary characters, providing grammatical context, simplifying words to their base forms, and analyzing semantic similarity, the text was made ready for sophisticated NLP applications.

### Parts of Speech Tagger

Part-of-speech (POS) tagging is a natural language processing (NLP) task that involves assigning grammatical labels or tags to words in a given text based on their syntactic roles and relationships within a sentence. These tags represent the part of speech or word category to which each word belongs, such as noun, verb, adjective, adverb, pronoun, preposition, conjunction, and so on. The POS tagger is used to find the word similarity and determine the paraphrased sentences.

To develop a robust POS tagging model for the Nepali language, the LSTM (Long Short-Term Memory) architecture is used for deep learning. However, in order to train such a model effectively, a properly tagged dataset is used. This dataset should comprise a comprehensive assortment of Nepali sentences, with each word annotated with its respective part-of-speech (POS) tag. These POS tags are useful in representing the grammatical category or syntactic role of each word within a given sentence. By using this annotated dataset, LSTM model is used to accurately assign POS tags to Nepali words, facilitating better understanding and analysis of the language's grammatical structures.

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is specifically designed to handle sequential data, such as time series, speech, and text. LSTM networks are capable of learning long-term dependencies in sequential data, which makes them well suited for Natural Language Processing. LSTM is specifically designed to address the exploding/vanishing gradient problem and capture long-term dependencies in sequential data.

It introduces memory cells and gating mechanisms that allow it to retain and update information over longer sequences. It consists of three main components: the input gate, the forget gate, and the output gate.

The main equations used are as follows

ft = σ (Wf. [ht-1, xt] + bf) ………………... (5-1)

it = σ (Wi. [ht-1, xt] + bi) ……………….… (5-2)

𝑐’t = tanh (Wc. [ht-1, xt] + bc) ……………. (5-3)

ct = ft \* ct-1 + it \* 𝑐’t ……………….……. (5-4)

ot = σ (Wo. [ht-1, xt] + bo) ………………... (5-5)

ht = ot \* tanh (ct) ………………………… (5-6)

where, ft is forget gate, it is input gate, ot is output gate, ct is cell state and ht is hidden gate

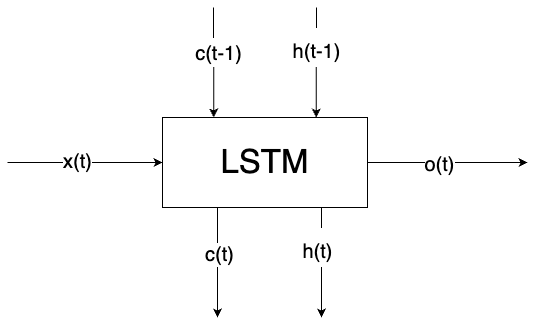


Figure 5‑2: Block Diagram of LSTM Architecture

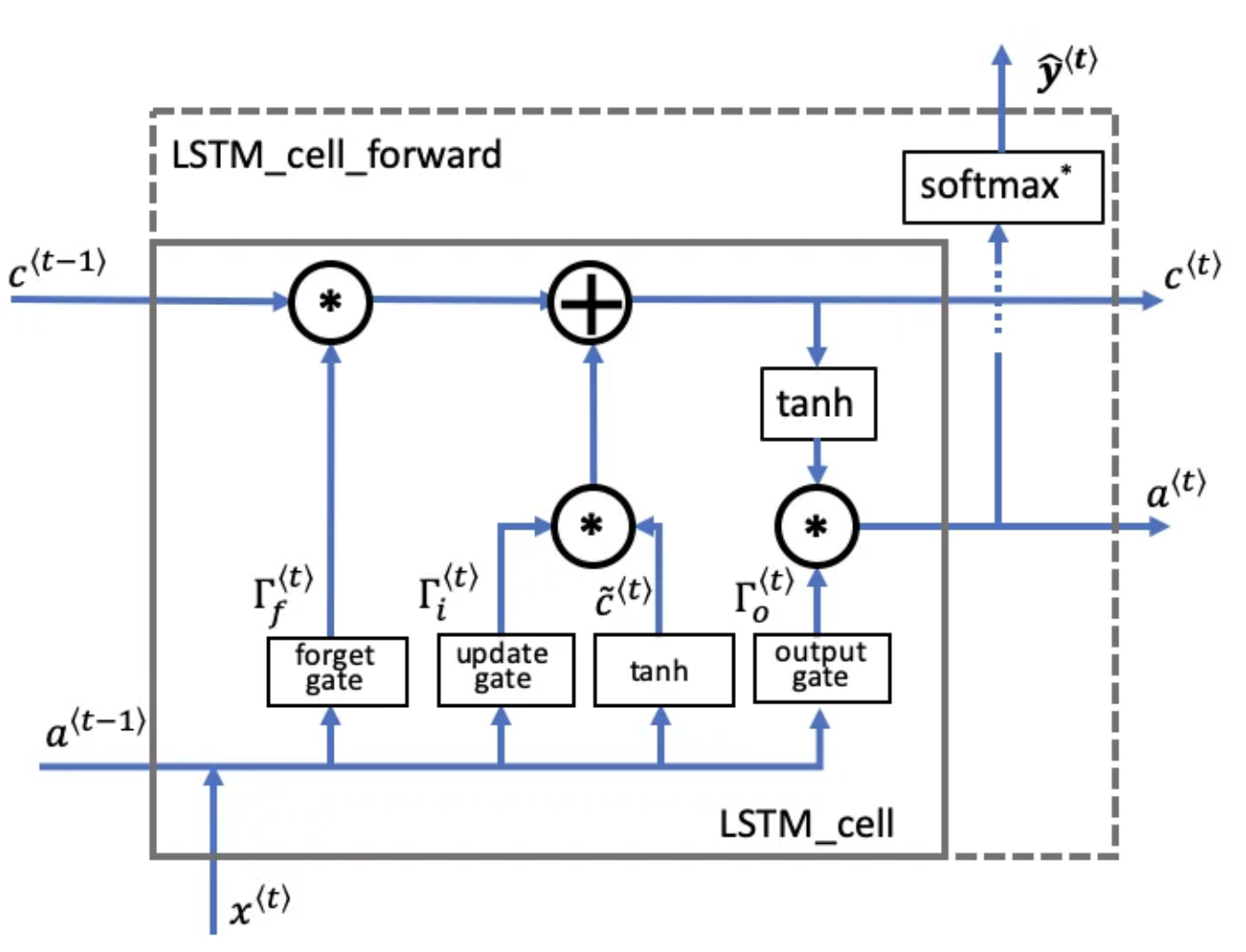


Figure 5‑3: Detailed Diagram of LSTM Architecture [19]

1. **Forget Gate**: The forget gate determines which information is discarded from the cell state. It takes the input x(t) and the previous hidden state a(t−1), applies a weight matrix, adds a bias, and passes them through a sigmoid function. The output is a vector with values between 0 and 1, which will be multiplied element-wise to the previous cell state c(t−1).
2. **Input (Update) Gate and Candidate Memory** : Simultaneously, the update gate decides what new information is stored in the cell state. It also processes x(t) and a(t−1) in a similar manner to the forget gate but uses a separate set of weights and bias. The candidate memory creates a vector of potential values to add to the cell state, which is derived by applying a tanh function to the weighted and biased combination of x(t) and a(t−1).
3. **Cell State Update** : The cell state is updated by multiplying the old cell state c(t−1) by the forget gate's output to discard information deemed unnecessary. Then, the result of the input gate is multiplied element-wise with the candidate memory to decide which new information is added. These two results are added together to form the new cell state c(t).
4. **Output Gate and Hidden State** : The output gate regulates the information that is output from the cell. It applies another sigmoid function to x(t) and a(t−1). The new cell state c(t) passes through a tanh function (to normalize the values between -1 and 1) and then is multiplied element-wise by the output gate. The result is the new hidden state a(t), which is also the output of the LSTM cell at this timestep.
5. **Output Prediction**: Finally, the hidden state a(t) can be used to generate predictions. It is typically passed through a fully connected layer followed by a softmax activation function (not shown in detail in this diagram) to obtain a probability distribution over possible outputs, denoted as y^​(t).

Each of these steps involves parameters such as weights and biases that are learned during the training process. The LSTM cell's ability to regulate information flow through gates allows it to maintain long-term dependencies and avoid issues like vanishing gradients, making it effective for tasks involving sequential data.

### Model Building for POS Tagger

Once the refined corpus is obtained each word and its corresponding POS tag are encoded using the one-hot representation. This encoding scheme assigns a unique binary vector to each word and POS tag pair. And to maintain uniformity in the training data, padding is applied. Padding involves adding a special token (usually represented as zeros) to sequences of varying lengths, ensuring they all have the same length. This step is crucial for training the model efficiently, as neural networks typically require fixed-length input sequences. Then a model having different layers i.e., word embedding layer, dropout layer, LSTM layer, dense layer etc. are created. Finally, the model is be trained using the encoded values. Through this training process, the model learns to discern the associations between the encoded words and their corresponding POS tags, enabling it to accurately predict the POS tags for new and unseen words in a given sentence.

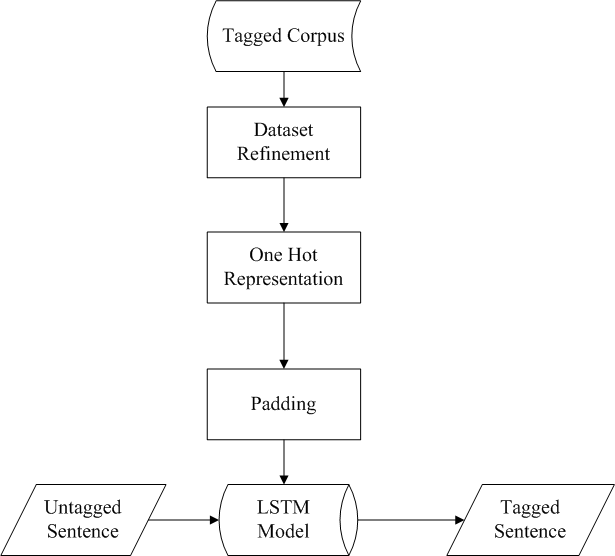


Figure 5‑4: POS Tagger using LSTM

1. **Tagged Corpus**: The workflow began with a tagged corpus, a structured dataset where each word had already been annotated with its corresponding POS tag. This corpus served as the training data, offering the LSTM model the necessary examples of words and their grammatical contexts.
2. **Dataset Refinement**: The tagged corpus underwent a refinement stage where the data was cleaned and standardized. This stage might have included removing noise such as irrelevant symbols, converting all text to a uniform case, and other preprocessing steps aimed at creating a clean dataset for the model to learn from.
3. **One-Hot Representation**: After refining the dataset, each word in the corpus was transformed into a one-hot encoded vector. In this representation, a vector consists of all zeros except for a single one that indicates the index of the word in the vocabulary. The POS tags associated with the words were encoded in the same manner.
4. **Padding**: Given the variable lengths of sentences, a padding step was implemented. Shorter sequences were extended with padding tokens to match the length of the longest sequence in the training dataset. This padding ensured that the LSTM received input sequences of uniform length, which is a requirement for training neural networks.
5. **LSTM Model**: The processed and padded one-hot encoded vectors were then fed into the LSTM model. This model, known for its ability to handle sequential data and remember information over long periods, learned to predict the POS tags based on the context provided by the input sequences.
6. **Untagged Sentence Input**: For prediction or inference, untagged sentences were introduced into the trained LSTM model. These sentences did not contain any POS annotations and represented new data for the model to tag.
7. **Tagged Sentence Output**: The LSTM model outputted the tagged sentence, providing a sequence of one-hot encoded vectors that represented the predicted POS tags for each word. These predictions were subsequently converted from the one-hot encoded format back into a readable sequence of POS tags, thus completing the tagging process for the untagged input sentence.

### SVM for Plagiarism Detection

Plagiarism detection is based on lexical analysis of the tokens made from parsing of the text. The tokens of the suspicious documents were compared with the tokens of original document. Overlap Coefficient was used to compare the tokens of the suspicious and the original documents. The matching process took the normalized word with the same POS. The value of each text was compared to the value of the original text and system are classified into the four categories. The greater the value of the coefficient defines the greater in similarity between the new document and the old document.

J (A, B) = …………………. (5-7)

The Jaccard Similarity has a major drawback when the two texts of document that are checked have different word length. Consider two texts, X and Y, where both texts contain 100 words. Now assume that 50 of those words are similar in the texts. The Similarity score is J (X, Y) = 50 / (100 + 100 - 50) = 0.33. Let us increase the word length of text X by 10 words and decrease the word length of text Y by the 10 words, keeping the total of 50 words the same, the Similarity score remains the same.it had no effect to the word length of the text. [20]

The Overlap Coefficient (Szymkiewicz–Simpson coefficient), is a derivation of Jaccard Similarity where the size of the sets is taken into account. It is the intersection of cardinality set A and set B over the size of the cardinality of smaller set between A and B.

……………… (5-8)

In Machine Learning, the index was able to quantify the similarities between computer’s identified texts and the training data sets. The corpus of the data was used as testing data for the machine training. The basic idea of SVM is the to construct an optimal hyper plane, which can be used for classification, The optimal hyper plane is a hyper plane selected from the set of hyper planes for classifying patterns that maximizes the margin of the hyper planes.

The equation for hyperplane can be given as:

aX + bY = C………………………… (5-9)

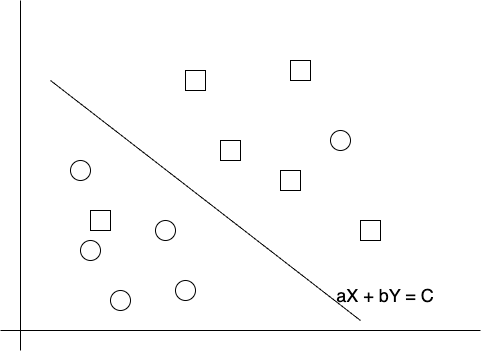


Figure 5‑5: Hyperplane

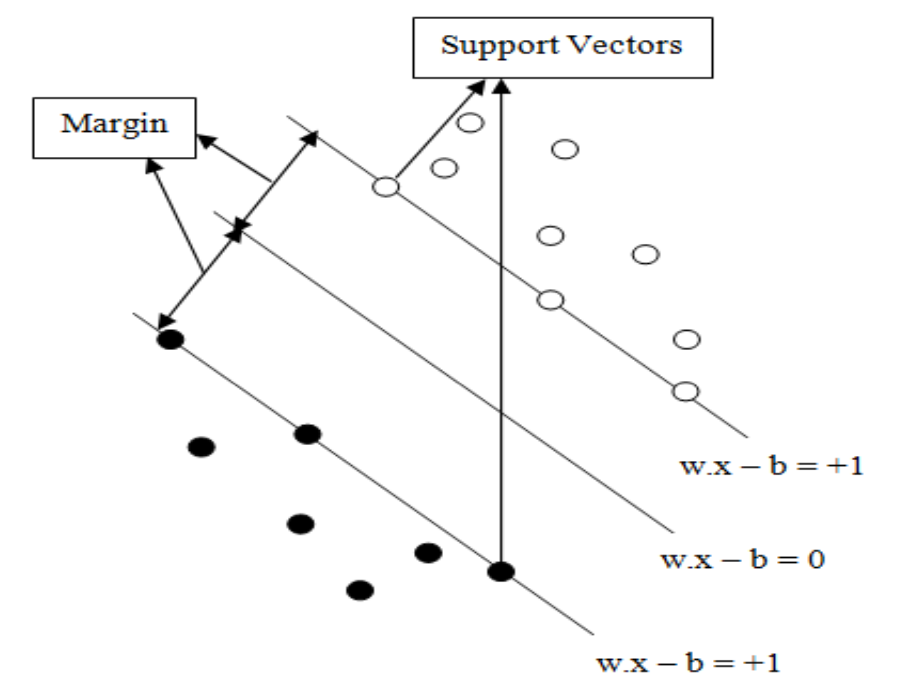


Figure 5‑6: SVM model [21]

The figure 5-7 shows that the model consists of three different lines. These three lines construct the hyper plane that separates the given patterns and the pattern that lies on the edges of the hyper plane is called support vectors. The perpendicular distance between the line of margin and the edges of hyper plane is known as margin. The objective of support vectors is to optimize the margin so that it can classify the given problem.

The general form of the hyperplane in SVM is expressed by the equation:

wTx - b=0

where x is the feature vector, b is the bias, and w is weight vector.The support vectors are the data points that are closest to the hyperplane and contribute to determining the position and orientation of the hyperplane.The hyperplanes that pass through the support vectors are:

wTx-b=1

wTx-b=-1

These equations define the boundaries of the margin.

In a binary classification setting, the training data consists of n pairs (x**i** ​,yi ​), where each xi is a n-dimensional feature vector, and y**i**​ is the class label which can be either 1 or -1. The SVM algorithm learns to predict new data points to one of these categories.

In contrast to Softmax regression, which is often used in logistic regression models and aims to maximize the log-likelihood, SVM focuses on maximizing the margin. This margin maximization leads to a model that is robust to outliers and has good generalization properties, often resulting in better performance on unseen data..

### Hinge Loss Function

The hinge loss function, integral to the learning mechanism of Support Vector Machines (SVM), is designed to both penalize classification mistakes and enforce a wide margin between classes. For a given set of data points, where each point is represented by a feature vector xi​ and an associated class label yi​, the hinge loss for a prediction can be expressed mathematically as

L(xi,yi) = max(0,1-yi\*(wTxi+b))

In this formula, w is the weight vector, b is the bias term, and yi is the actual class label of the data point, which should be either 1 or -1.

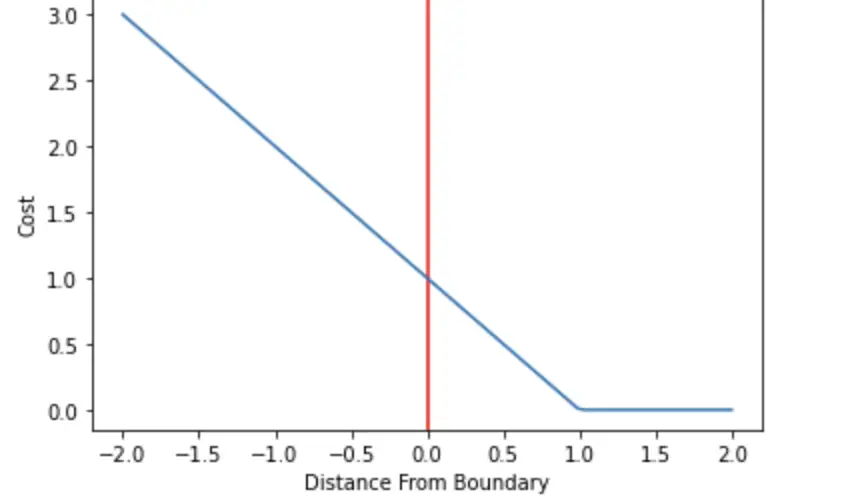


Figure 5‑7: Hinge loss Visualized

The essence of the hinge loss is captured by its piecewise linear graph, where the loss is zero for any data point that is classified correctly with a margin greater than 1. In other words, if the product yi\*(wTxi+b) is greater than 1, indicating that the point is on the correct side of the margin, no penalty is applied. However, as this product decreases towards zero, and especially as it becomes negative (implying a misclassification), the hinge loss increases linearly. This is visually represented by a graph with a hinge: a flat line at zero loss transitioning into an upward sloping line as the margin criterion is violated.

The graph of the hinge loss function is characterized by a 'hinge' at the point where yi\*(wTxi+b)) =1For values greater than this threshold, the graph remains at zero, indicating no loss. However, when the value is less than 1, the graph slopes upwards, with the slope determined by the extent of the violation of the margin. This slope represents increasing penalties for points that are either within the margin or misclassified. Consequently, during the training of an SVM, the optimization process aims to minimize this loss, effectively pushing the decision boundary to a position where it maximizes the margin between classes, resulting in a model that is generalizable and robust to outliers.

The graph and properties of the hinge loss function thus align closely with the SVM's objective of maximizing the classification margin, making it a key component in SVM's capacity to produce models with strong predictive accuracy.

## Use Case Diagram

The use case diagram for the plagiarism detection system represents interactions between actors and system through use cases. The use cases depict the different functionalities of the system such as paper submission by the student, review of the plagiarism report and database management by the professor; management of database and account information by the admin, comparison of sources to detect plagiarism, and generation of the plagiarism report. The "include" relationship was used to show that a use case depends on another use case and is included as a part of it. The compare Sources use case is initiated by the system as of the Submit Paper use case. The Generate Report use case is initiated by the system as a result of the Compare Sources use case.

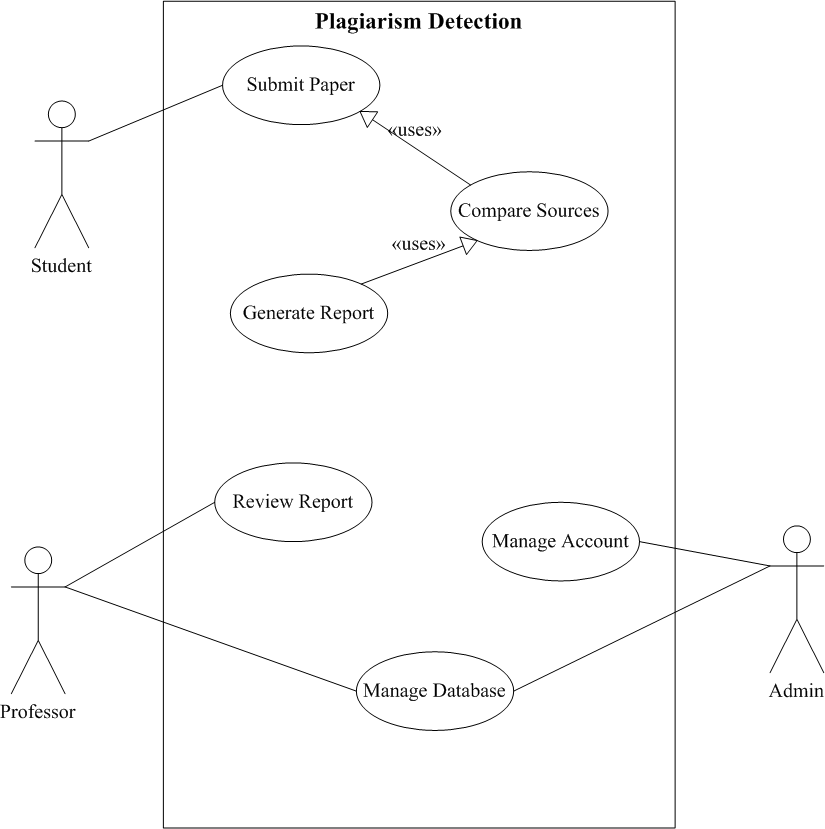


Figure 5‑8: Use Case Diagram

## System Architecture

The figure below describes the architecture of the proposed system. The application was provided to the user to insert the documents into the system one original and other suspicious ones then preprocesses both original and suspicious theses. Next, the application estimated the similarities between the documents provided and provide user with the plagiarism status through different feature extraction methods.

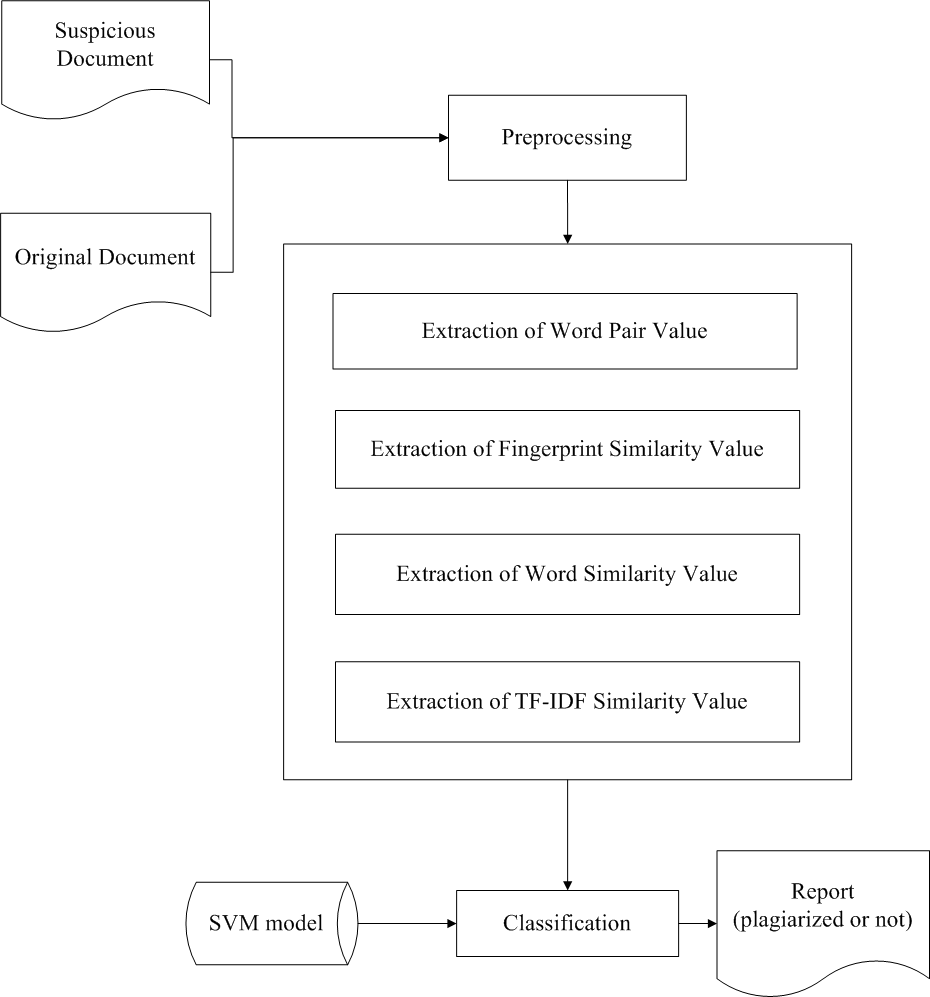


Figure 5‑9: System Architecture

### Feature Extraction

* **Word Pair**: The analysis involved a focus on identifying consecutive similar words within a text. If a high count of such word pairs was found, it indicated a greater likelihood of plagiarism. This process, known as word pair extraction, analyzed the similarity between consecutive word pairs to capture specific patterns and repetitions. The presence of such patterns was used to suggest similarities in the text that could point to plagiarism.
* **Word Similarity**: Beyond examining individual words, the semantic understanding of sentences was considered crucial for determining text similarity. The word similarity feature was employed to compare each word in the text under scrutiny against a set of synonyms to ensure that the text was not even remotely similar to any source document. This process helped detect instances where words might have been replaced with synonyms or otherwise altered to conceal plagiarism.
* **Fingerprint Similarity**: Fingerprint similarity was used to perform a more structural analysis of the text. It focused on the formation of n-grams, which are sequences of 'n' consecutive words found within the text. By evaluating these specific sequences of words, the analysis was able to identify patterns and similarities that might not be immediately apparent through individual word comparison. This approach was particularly effective at uncovering similar phrases or arrangements of words that could indicate potential plagiarism..
* **TF-IDF Similarity**:The likeness between documents was assessed using TF-IDF (Term Frequency-Inverse Document Frequency) similarity measures. The process calculated similarity scores by comparing the TF-IDF values of terms that appeared in both documents. Each document was represented as a vector in a multidimensional space, where each dimension corresponded to a unique term weighted by its TF-IDF score. The analysis then measured the cosine similarity between these vectors, which quantifies the cosine of the angle between them. A higher cosine similarity score signaled a greater degree of resemblance between the documents, suggesting a shared similarity in content, while a lower score indicated less similarity.

In summary, these features worked together to capture different aspects of similarity between texts. Word pair and word similarity focus on the individual words, while fingerprint similarity analyzes the structural patterns. TF-IDF similarity provided a contextual understanding of the text, considering its meaning and semantics. By utilizing these features collectively, accuracy of plagiarism detection algorithms is increased and instances of text similarity is detected more effectively.

Support Vector Machine (SVM) is machine learning algorithm that recognizes patterns. It can make a hyperplane that can separate two classes. SVM modeled the existing data into points in a space. The location of each point depended on the value of the features used.

### TF-IDF with Smoothing

Additive smoothing TF-IDF is used as a technique to improve the accuracy of our text classification. To implement this technique, first the term frequency for each term in the document is calculated.

TF = n/N................... (5-10)

Then the inverse document frequency for each term is calculated using smoothing, which is the logarithm of the total number of documents +1 divided by the number of documents containing that term +1.

IDF = log ((N + 1) / (n + 1)) + 1............... (5-11)

Finally, the term frequency is multiplied with the smoothed inverse document frequency to obtain the TF-IDF weight for each term.

TF-IDF = TF \* IDF...................... (5-12)

This weight was used to represent the importance of each term in the document. By using additive smoothing TF-IDF, the system was able to effectively adjust the weights of words in our documents, ensuring that rare terms were not given too much weight, while common terms were not ignored. This improved the accuracy of our text classification and allowed us to better extract relevant information from our text data.

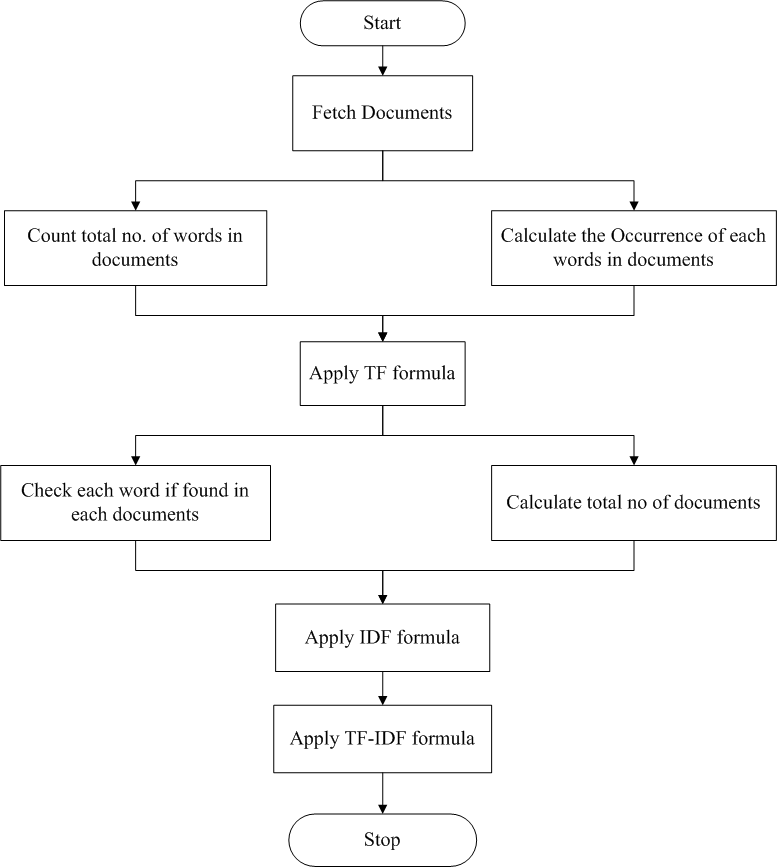


Figure 5‑10: Block Diagram showing TF-IDF Calculation

The process of text analysis using the TF-IDF methodology was initiated by fetching a collection of documents to be examined. Once the documents were retrieved, the total number of words in each document was counted. This step was crucial in preparing for the subsequent calculation of term frequencies. Concurrently, the occurrence of each word within these documents was determined, laying the groundwork for the application of the Term Frequency (TF) formula.

Following the preparation phase, the Term Frequency for each word was computed. This involved checking the presence of each word across all documents and applying the TF formula to quantify the number of times a word appeared in a document relative to the total number of words in that document. Simultaneously, the total number of documents in the collection was calculated. This count was instrumental for the next step, which involved applying the Inverse Document Frequency (IDF) formula. The IDF formula measured the rarity of each word across the entire set of documents; common words received a lower score, while rare words were assigned a higher score.

Finally, the individual TF and IDF scores were combined using the TF-IDF formula. This multiplication resulted in a set of values representing the importance of each word within the documents in relation to the entire document corpus. Words that were common across all documents had their significance downplayed, while unique or rare words in particular documents were highlighted as being of higher importance. With the TF-IDF scores calculated, the analysis process came to a close, concluding the computational phase and yielding data ready for further interpretation, such as in tasks of document similarity analysis or plagiarism detection.

### Calculation of Word Similarity Value

Word similarity value is the similarity of words in the sentences extracted from original and suspicious documents i.e., percentage of word similarity between those sentences. At first, both the suspicious and the original document are reduced to sentence level and the different preprocessing techniques are applied to create the metadata of words. Finally, the Overlap similarity score is used to measure the word similarity between the original and suspicious documents.

The original and suspicious documents are resolved into paragraph and then the paragraph into the sentence using sentence segmentation. The different preprocessing technique used are tokenization, lowercase conversion, stop-word removal, punctuation removal, Part-of-Speech tagging (POS). Part of speech tagging is done using our developed POS tagging model. To compare the words in both documents, the Nepali Synsets created from Nepali dictionary is used. Lists of the synonym’s words are collected using Nepali Synsets. The words in the suspicious document are compared with similar or same words in the original document taking the POS tag into consideration to find the similarity value. Then *Overlap Similarity Score* is calculated which gives the percentage of word similarity between two sentences or paragraph.

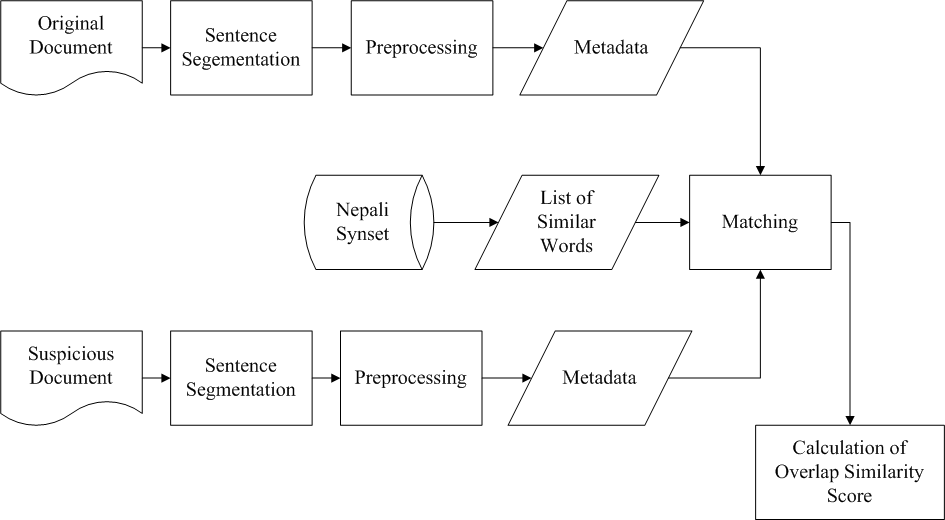


Figure 5‑11: Calculation of Word Similarity Value

### Calculation of Fingerprint Similarity Value

Fingerprint similarity is a measure of how similar two documents are based on their content. It entails creating distinctive identifiers for each document and contrasting them to assess how similar each document is. The fingerprint is a collection of integers that is produced by hashing a document's key contents from its subsets. A document is separated into a number of text "fingerprints" that may or may not overlap one another. In this manner, fingerprints assist us in assessing the degree of correspondence between papers. The calculation of the fingerprint similarity of given documents uses Simhash algorithm.

Using a method known as shingling, the Simhash algorithm takes the documents and extracts a set of features from documents. Fingerprints of two documents are compared to measure the similarity and compute similarity score using Overlap Similarity Coefficient.

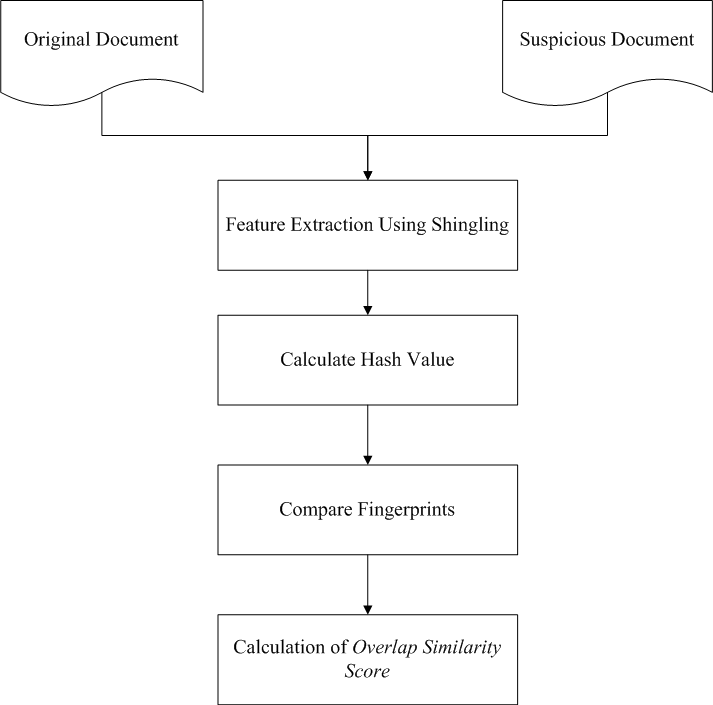


Figure 5‑12: Calculation of Fingerprint Similarity

1. **Feature Extraction Using Shingling**: The process starts with the extraction of features from both the original and suspicious documents using a technique called "shingling." Shingling involves creating a set of unique elements, called shingles, from the document. A shingle is typically a contiguous sequence of items, which can be characters, words, or other tokens, depending on the granularity chosen. For text documents, shingles are often sequences of words (e.g., 5-grams would be sequences of five consecutive words). This step transforms the text into a form that makes comparison computationally manageable and meaningful.
2. **Calculate Hash Value**: Each shingle is then hashed using a hash function. Hashing converts the shingles into a numerical value (hash value) that represents the original sequence in a more compact form. The hash function is designed to be fast and to minimize the chances of different shingles producing the same hash value (collisions). The result is a set of hash values for each document, effectively creating a "fingerprint" of the document's content.
3. **Compare Fingerprints**: With both documents represented by sets of hash values, the next step is to compare these fingerprints to identify overlaps. The comparison seeks to find shingles that are common to both documents, with the underlying assumption that shared shingles may indicate shared content, and therefore, potential plagiarism.
4. **Calculation of Overlap Similarity Score**: After comparing fingerprints, an overlap similarity score is calculated. This score quantifies the degree of similarity between the two documents. It is typically represented as a percentage, with a higher percentage indicating more overlap and, consequently, a higher likelihood of the suspicious document having content copied from the original document.

### Creation of Nepali Synset

The PDF document from the Nepal Academy website was first downloaded and then converted it to JSON so that it can be used to create Synonyms set for Nepali words or Nepali Synset. The information from the PDF file was extracted using Tesseract for this conversion, and the data was converted and stored on a JSON using the JSON package.

A trie data structure also known as prefix tree is a tree-based data structure commonly used to store and retrieve strings as keys. It can be adapted to store word meanings in a dictionary, where each word maps to a node in the trie. For example, if we have a dictionary entry for the word "apple," the trie would have a path from the root node to a leaf node corresponding to the letters ‘a'−> 'p' −> 'p' −>'l' −> 'e'. The leaf node would store the meaning and synonyms of the word.

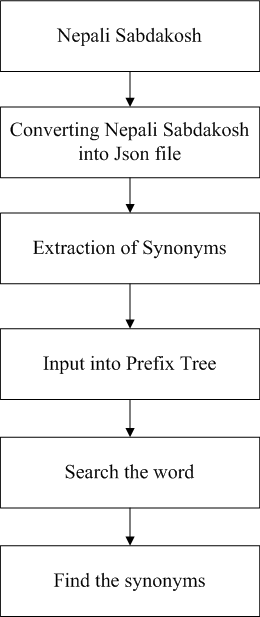


Figure 5‑13: Synset Preparation Flowchart

Once the Nepali Sabdakosh was available in JSON format, the next action involved the extraction of synonyms from the dataset. This process required parsing the JSON file to retrieve synonyms for each entry, creating a comprehensive list of words along with their associated synonyms.

Following the extraction, the list of words and their synonyms were input into a Prefix Tree, also known as a trie. A Prefix Tree is a data structure that allows for efficient retrieval of words or phrases. It's particularly well-suited for autocomplete or spell checking systems because it organizes the words in a way that common prefixes are shared, which makes the search operation very efficient.

The final stage in the process was to search for a specific word within the Prefix Tree. When a word was searched, the Prefix Tree structure allowed for quick and efficient retrieval of data associated with that word, including the list of synonyms that had been previously extracted and stored. Therefore, when a search query was entered, the system was capable of quickly finding and returning the synonyms for the given word from the Nepali Sabdakosh, making it a useful tool for anyone looking to expand their vocabulary or find alternative words in the Nepali language.

## Flowchart of Proposed System

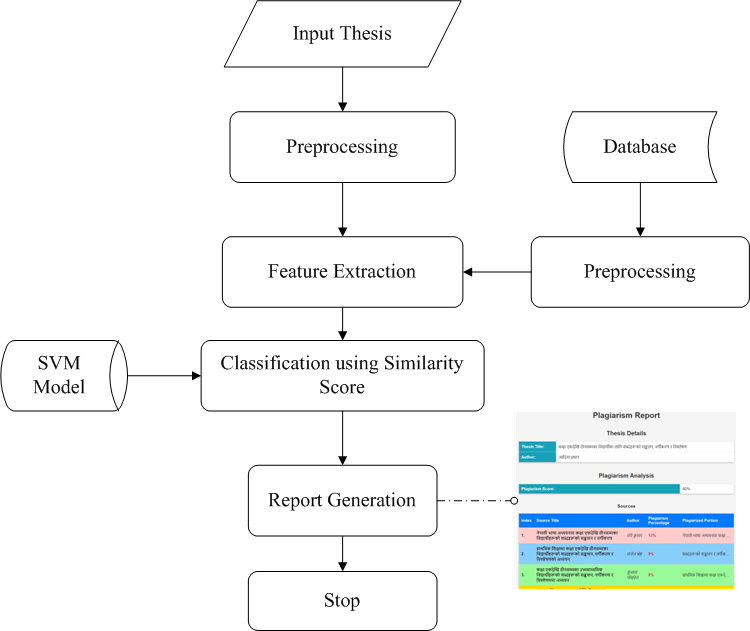


Figure 5‑14: Flowchart of Proposed System

Our application is designed to help users check if their documents contain any plagiarized content. Users can input the document they want to check, and our system then compares it with the documents available in our database.

The image provided appears to be a flowchart describing a process for detecting plagiarism in an input thesis using a Support Vector Machine (SVM) model. Here's an explanation in paragraph form:

Initially, a thesis was input into the system to undergo plagiarism detection. This thesis, as well as existing documents from a database, were subjected to preprocessing. Preprocessing is a crucial step where the text is cleaned and normalized; it often includes tasks such as removing special characters, converting all text to lower case, removing stop words, and possibly reducing words to their base or root forms through stemming or lemmatization.

Following the preprocessing, the system extracted features from the text. Feature extraction is a process of converting text into a numerical form that machine learning algorithms can understand. This typically involves selecting meaningful attributes or patterns from the text that can be used to assess similarity. These features might include the frequency of certain words or phrases, the presence of specific syntactic structures, or statistical measures like term frequency-inverse document frequency (TF-IDF) scores.

These features were then fed into an SVM model. Support Vector Machines are a type of supervised machine learning algorithm that can classify data into different categories. In this case, the SVM model used the extracted features to classify pieces of the thesis as either plagiarized or original based on a similarity score. This score likely represented the degree of likeness between the thesis and documents in the database.

Once the SVM model completed its classification, the system generated a report. This report likely detailed which parts of the thesis were original and which parts were potentially plagiarized, including the similarity percentages and possibly the sources from which the content might have been copied.

The process concluded with the completion of the report, marking the end of the plagiarism detection operation. The generated report would serve as a tool for educators or reviewers to understand the extent of originality within the thesis and take appropriate actions if necessary.

## Instrumentation Tools.

### Programming Language, Libraries and Frameworks

* **Python:** Python Programming Language is mainly used to implement Natural Language Processing for better analyzing and machine training for our model. It provides us with various packages to implement simple mathematical formulae like NumPy, matplotlib to complex data science packages like Scikit-learn and NLTK packages.
* **NLTK (Natural Language Toolkit):** NLTK provides various modules and functions for text processing, such as tokenization (breaking text into individual words or sentences), removing punctuation and stop words (commonly used words that do not carry significant meaning), stemming (reducing words to their root form), and more. NLTK simplifies complex NLP tasks and allows developers to preprocess and analyze text data efficiently.
* **Scikit-learn**: Scikit-learn provides a comprehensive set of tools for various machine learning tasks, including classification, regression, clustering, and dimensionality reduction. Scikit-learn offers efficient implementations of popular algorithms, such as Support Vector Machines (SVM), which are commonly used for model training. It also provides utilities for data preprocessing, feature extraction, model evaluation, and cross-validation. Scikit-learn simplifies the implementation of machine learning models and enables easy experimentation with different algorithms.
* **Pandas**: Pandas provides data structures, such as data frames, that allow for efficient handling and manipulation of structured data. Pandas offers functions for reading and writing data from various file formats, data cleaning and preprocessing, filtering and selecting data, grouping and aggregating data, and more.
* **Matplotlib**: Matplotlib is a widely used data visualization library in Python. It provides a comprehensive set of functions for creating various types of plots, charts, and graphs used for visualizing various dataset in process of feature extraction and dataset preparation. It is used in conjunction with Pandas to visualize data frames and gain a better understanding of the data.
* **Beautiful Soup**: Beautiful Soup is a Python library for parsing data from HTML and XML files. It provides idiomatic ways of navigating, searching, and modifying the parse tree. Beautiful Soup was used to scrape the digital thesis from TUCL library.
* **Django:** Our project leverages the combination of Django for backend processing and HTML, CSS for frontend development. Django serves as the backbone, handling model processing and predictions through Python code, ensuring efficient data manipulation and execution. On the frontend, HTML, CSS created dynamic and interactive user interfaces, seamlessly connecting to Django's backend via Django RestFramework's API endpoints, enabling smooth data exchange
* **MySQL**: MySQL provides a structured and efficient way to store, retrieve, and manage large amounts of data. In the project, MySQL is used as the database software to store and access the documents. By using MySQL, the system can handle the storage and retrieval of documents in an organized and efficient manner.

# IMPLEMENTATION DETAILS

## Dataset Preparation

Due to the limited resources available for Nepali language, there aren’t any data sets to work on Nepali language plagiarism detection. We had to prepare the dataset from scratch and label it ourself. The first step in this process was to create a paragraph-level dataset to assess plagiarism at the paragraph level. The dataset is designed to include four different types of plagiarism: complete plagiarized texts, direct plagiarized texts, non-plagiarized texts, and paraphrased plagiarized texts. To ensure the quality and consistency of the dataset, we decided to use the standards provided by poorvucenter Yale University. The goal of this dataset creation was to provide a comprehensive resource for Nepali language plagiarism detection and to fill the gap in the limited resources currently available for this language. This multi-type dataset will allow for a thorough evaluation of Nepali language plagiarism and help to increase the performance and accuracy of this application.

## Text Extraction from Nepali thesis

The collection of Nepali theses required for the text extraction were stored in online repository of Tribhuvan University Central Library (TUCL). Beautiful soup and requests library were used to scrape the pdf files from website of TUCL and downloaded the PDF file and passed further for text extraction from the PDF.

Firstly, the extracted pdfs were converted into word format document for ease and fast extraction. The conversion also helped to preserve the format of document to some extent. The document contained the text which was in incomprehensible form. A library called “npttf2utf” was used to convert the Preeti font to Unicode (python understandable form) with the help of font mapper class. The output was in the form of text file with encoding utf-8 which had the extracted text in a comprehensive format.

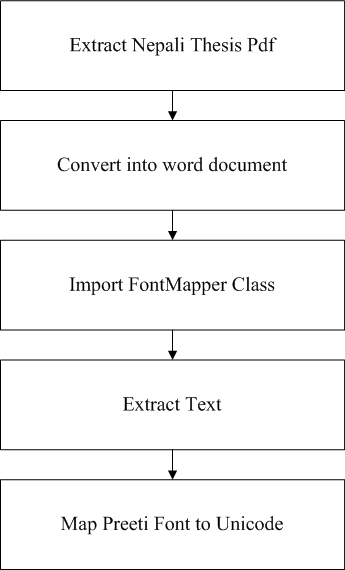


Figure 6‑1: Extraction of Text from digitalized form

Initially, a Nepali thesis presented in PDF form was extracted. PDFs, known for their fixed layout, do not allow direct text manipulation, which necessitated the next step of converting the document into a Word format. This conversion facilitated text manipulation and set the stage for the next critical steps.

With the thesis now in a Word document, the "FontMapper" class was employed. This class is a component often used in programming environments to resolve font-specific encoding challenges, which is particularly significant for scripts such as Devanagari used in Nepali. The FontMapper ensured that the characters were represented correctly in the system.

Once the Word document was ready, text extraction commenced. This was a pivotal step as it transformed the content from a format that is hard to edit into one that could be processed by text analysis tools. The text at this stage, however, was still in the Preeti font, which is a non-Unicode font commonly used for Nepali typing.

To make the text universally readable and processable, the "npttf2utf" library was utilized. This library, with the aid of the FontMapper class, converted the Preeti font into Unicode. Unicode is a computing industry standard for consistent encoding and representation of text, allowing for the text to be understood and manipulated within Python and other environments. This conversion was not just a transliteration of characters but also a necessary step for standardizing the text representation, ensuring that the Nepali text could be read and processed accurately by any system worldwide.

## Text Preprocessing

Different types of text pre-processing are applied to both the source and suspicious texts to clean and prepare it for the feature extraction.

* **Punctuation removal:** All the punctuation symbols present in the text were replaced with the space using punc\_removal(text) function from the preprocessing module. This function returned the string after replacing each punctuation with space. The NLTK library provided us with the list of English punctuation to be removed. For Nepali language, an additional punctuation ‘।’ needed to be removed.

Input text = “नेपाल धर्मनिरपेक्ष, बहुसांस्कृतिक, बहुभाषिक र बहुधार्मिक राष्ट्र हो ।”

Output text = “नेपाल धर्मनिरपेक्ष बहुसांस्कृतिक बहुभाषिक र बहुधार्मिक राष्ट्र हो ”

* **Stop-word removal:** Non-functional words were removed from the token list as they had no important explanation in the text. This was done using stopword\_removal(token list) function from the preprocessing module. It returns the cleaned token list.

Some examples of Nepali stopwords are: { गए, जस्तो, त्यहाँ, आदि, एकदम }etc

* **Tokenization:** Tokenization is the process of dividing the text into the tokens/words. This was done using tokenization(text) function from the preprocessing module. This function returned the list of tokens.

Input text= “नेपाल धर्मनिरपेक्ष बहुसांस्कृतिक बहुभाषिक र बहुधार्मिक राष्ट्र हो ।”

Output = ['नेपाल', 'धर्मनिरपेक्ष', 'बहुसांस्कृतिक', 'बहुभाषिक', 'र', 'बहुधार्मक', 'राष्ट्र', 'हो']

* **Stemming:** Stemming is the process of map words/tokens to a common base word.

Words= [ खायो, जानु]

Root word = [खा, जा]

* **POS tagging:** POS tagging is the processing of assigning the Part of Speech tag to each token.

Text = "यस लाई पदहरू मा विभाजन गरिन्छ ।"

Output: [('यस', 'DDX'), ('लाई', 'IA'), ('पदहरू', 'NN'), ('मा', 'II'), ('विभाजन', 'NN'), ('गरिन्छ', 'VVYN1'), ('।', 'YF')]

Output After Generalizing Tags: [('यस', 'Determiner'), ('लाई', 'Postposition'), ('पदहरू', 'Noun'), ('मा', 'Postposition'), ('विभाजन', 'Noun'), ('गरिन्छ', 'Verb'), ('।', 'Punctuation')]

These preprocessing steps helped to improve the accuracy of plagiarism detection by reducing noise in the text data and making it easier to compare documents. We had used tokenization, punctuation and stop-word removal for calculating bigram, TF-IDF and fingerprint similarity and added lemmatization and pos-tagging for calculating word similarity.

## POS tagger

POS tagging using LSTM is a common and effective approach for assigning grammatical categories (POS tags) to each word in a given sentence or text. LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) that can handle sequential data making it well-suited for sequence-to-sequence tasks like POS tagging. The steps involved in POS tagger model development are:

* **Data Preparation**: The initial step in developing the POS tagger was to prepare the dataset required for training the LSTM model. This dataset was composed of sentences, with each word in these sentences annotated with the appropriate POS tag. These annotations were necessary for the model to learn the context and the corresponding grammatical categories of the words within the sentences.
* **Vector Representation**: Once the dataset was ready, the sentences and their corresponding POS tags needed to be converted into a form that the LSTM could process. This was achieved by transforming the sentences into vector representations. Words and tags were assigned numerical values, creating unique identifiers for each distinct word and POS tag in the dataset. This step converted the text data into numerical data, which is essential for the mathematical operations performed by neural networks.
* **Padding and Sequences**:LSTMs, like many neural network architectures, require input data of a consistent shape and size. Given that sentences naturally vary in length, it was necessary to ensure that each input sequence was of fixed length before it could be fed into the LSTM. To achieve this uniformity, shorter sentences in the dataset were padded with a special token, often referred to as a padding token. This padding ensured that all input sequences matched the length of the longest sentence in the dataset, allowing the LSTM to process them effectively.

## LSTM Architecture

### Embedding Layer

The embedding layer is the first layer of LSTM model. It takes the numerical representation of words (vectors) as input and converts them into dense vectors of fixed size. These dense vectors are learned during the training process and capture semantic relationships between words, which allows the LSTM model to better understand word similarities and differences.

### Dropout Layer

Dropout is a regularization technique used to prevent overfitting in neural networks, including LSTM models. The dropout layer randomly drops a certain percentage of neurons during training. This helps to learn more robust and generalizable features. Dropout can be placed after the embedding layer to prevent overfitting and improve the model's performance on unseen data.

### LSTM Layer

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) designed to handle sequential data by effectively capturing and learning dependencies over long sequences, particularly adept at mitigating the vanishing or exploding gradient problem. Within each LSTM unit, three gates—input, forget, and output gates—control the flow of information, allowing the network to selectively retain or forget information over varying time spans. This enables LSTMs to maintain a memory state that can preserve important information over extended sequences.

### Dense Layer

The Dense layer is the final layer in your LSTM model. It takes the output this LSTM layer and transforms it into the desired output shape. For POS tagging, the output layer should have neurons equal to the number of unique POS tags. Each neuron corresponds to a specific POS tag, and the activation values in the output layer indicate the likelihood of each word being associated with a particular POS tag. The Dense layer uses softmax activation function to produce the final probabilities for each POS tag.

This LSTM model with the mentioned architecture processes the input sentences by first converting them into vectors using the embedding layer. The dropout layer helps prevent overfitting, and the bidirectional LSTM layer captures contextual information from both directions in the sequence. Finally, the Dense layer produces the probability distribution of POS tags for each word in the sentence.

The model predicts one of one hundred and ten POS tags for each word in a sentence. However, to reduce the complexity and streamline the POS tag set, a dictionary is utilized. This dictionary maps the hundred and ten tags to a reduced set of eighteen tags. After obtaining the initial tag from the LSTM model, it is then looked up in the dictionary to convert it into one of the 18 final tags. This simplification helps enhance the model's interpretability and facilitates downstream tasks that require a more compact and manageable set of POS tags.

## Post-Position Separation for POS tagger

For POS tag prediction using LSTM model, a crucial pre-processing step is applied to each word in the input sentences before passing them to the model for prediction. This pre-processing involves passing each word through a dedicated post position separation function, designed to detect and separate specific suffixes present within the word. The post position separation function is based on a fixed set of known suffixes, and its purpose is to identify instances where the word contains any of these suffixes. When a word is found to have a matching suffix, the function separates the suffix from the original word, creating two distinct components: the core base word and the extracted suffix.  
Once this pre-processing step is completed for all words, the sentences are then passed to the LSTM model for POS tagging. With this approach, the model is trained on data where suffixes are isolated from the original words, which helps it to effectively learn and predict the POS tags in a more context-aware and accurate manner.

Sentence Before Separation:

यसमा त्यस क्षेत्रमा रहेका मानिसहरूको बोलीभाषा समाज र संस्कृतिका साथै उनीहरूका सुख दुःख बेदनाका अलावा जीवनका विविध आयामहरूको लयात्मक प्रस्तुति गरिएको पाईन्छ ।

Sentence After Separation:

यस मा त्यस क्षेत्र मा रहे का मानिस हरू को बोलीभाषा समाज र संस्कृति का साथै उनी हरू का सुख दुःख बेदना का अलावा जीवन का विविध आयाम हरू को लयात्मक प्रस्तुति गरिए को पाईन्छ ।

## Feature Extraction

* **Word Pair or N-gram:** N-grams are contiguous sequences of N items (words, characters, or tokens) extracted from a given text. The most common case is word-based n-grams, where sequences of N words are extracted from the text. N-grams capture local context and word relationships within a text, which can be valuable for various NLP tasks. Here, the texts are divided into the bigrams. Bigrams are two token sequences. Thus, created list of bigrams of suspicious and source text were used to calculate the overlap similarity. The value of overlap similarity was used as one of the features in SVM model.
* **TFIDF with Smoothing:** TF-IDF is a numerical representation used to evaluate the importance of a word in a document relative to a collection of documents (corpus). Term Frequency (TF) measures the frequency of a term (word) within a document. It indicates how often a word appears in the document   
  Inverse Document Frequency (IDF) measures the significance of a term across the entire corpus. It helped down-weight common words and amplify the importance of rare words. Using TF-IDF as a feature helped highlight important words that were distinctive to each paragraph and contributed to the comparison of paragraphs based on their content.
* **Word Similarity:** Wordsimilarity value refers to the degree of likeness between words found within sentences extracted from original and suspicious documents. It reflects the percentage of word similarity present between these sentences. The process initiates by breaking down both documents into lists of paragraphs. Subsequently, each paragraph was compared with the corresponding one from the other document. In order to facilitate this comparison, an approach involving the association of words with their respective part-of-speech tags was adopted. Following this, the Nepali Synset comes into play, aiding in the identification of synonymic words for each individual word. This method unveils words that might have been used in a plagiarized manner. As a result, a metadata profile for each word is generated, encompassing its tags and synonymous meanings. The final step employed a modified Jaccard similarity approach to quantify the resemblance between two paragraphs. This entails considering the metadata-enriched words within the paragraphs to compute the similarity score accurately.
* **Fingerprint Similarity:** Fingerprint are the unique identifiers used to distinguish one document form another. Fingerprint Similarity was used to determine the similarity between two documents in more structural level unlike n-gram. For computation of Fingerprint Similarity, first text was preprocessed then shingling to create overlapping phrases, hashing the shingles into distinct numerical representations, minhashing was applied for efficiency, indexing the fingerprints, and comparing new document fingerprints to compare potential duplicates based on a similarity threshold are all steps in the fingerprint similarity for detecting duplicate documents process. Like TFIDF the algorithm required smoothing for its better functioning in Nepali language.

## Calculation of Similarity Score of a Document

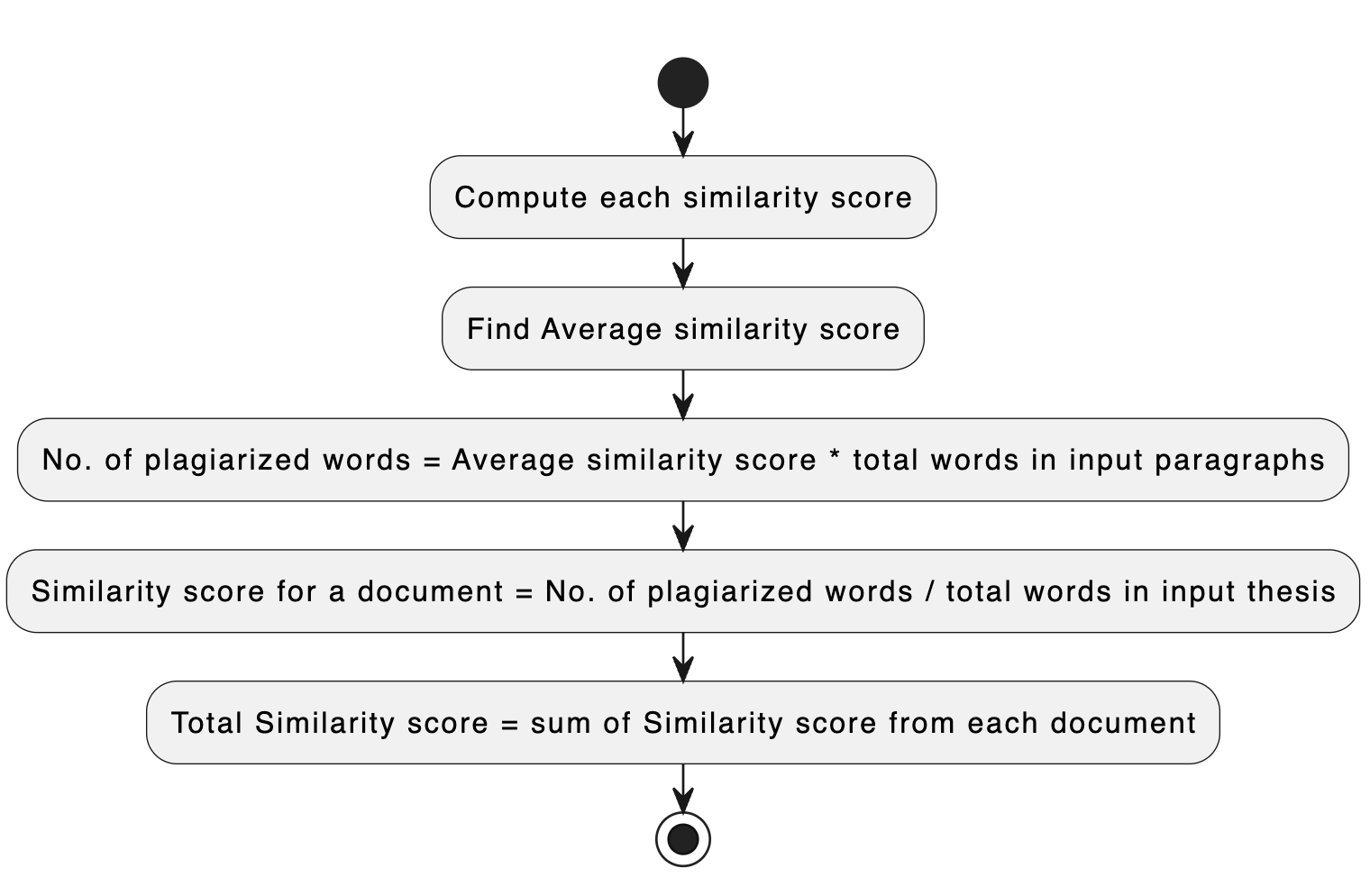


Figure 6‑2: Calculation of Similarity Score for a Documents

To quantify the degree of similarity between an input thesis and a set of reference documents, a systematic methodology was employed. This methodology involved several key steps:

1. **Computation of Each Similarity Score**: Initially, individual similarity scores were computed for the input thesis when compared to each reference document. These scores indicated the extent to which portions of the input thesis resembled or matched sections within each reference document.
2. **Calculation of Average Similarity Score**: After computing similarity scores for all reference documents, an average similarity score was determined. This average represented the typical similarity observed across the set of reference documents.
3. **Calculation of Number of Plagiarized Words**: To estimate the extent of plagiarism in the input thesis, the average similarity score was multiplied by the total number of words in the input paragraphs. This calculation provided the count of words within the input thesis that exhibited a level of similarity to the reference documents.
4. **Similarity Score for a Document**: The similarity score for an individual document was then computed by dividing the number of plagiarized words (determined in the previous step) by the total number of words in the input thesis. This score quantified the degree of similarity between the input thesis and each specific reference document.
5. **Computation of Total Similarity Score**: Finally, the total similarity score for the input thesis was determined by summing up the individual similarity scores obtained for each reference document. This score represented an aggregate measure of how closely the input thesis aligned with the entire set of reference documents.

By following this methodology, we were able to quantitatively assess the similarity between the input thesis and the reference documents, providing valuable insights into potential instances of plagiarism within the document.

## Hyperparameter Tuning of SVM model

A parameter grid was created defining the possible values for specific hyperparameters that were tuned during the hyperparameter optimization process.

* **Kernel**

The kernel hyperparameter determines the type of kernel function used in the Support Vector Machine (SVM) algorithm. Each kernel has its own characteristics and is suitable for different types of data. By exploring different kernel types, the best kernel for the dataset was found

Possible values: 'linear', 'rbf', 'poly', 'sigmoid'

* Linear kernel: It is the simplest kernel function and is used for linearly separable data. It defines the dot product between the input vectors in the original feature space.
* Polynomial kernel: It is used for non-linear data and maps the data into a higher-dimensional space. It is defined by the degree of the polynomial and a coefficient.
* Radial basis function (RBF) kernel: It is the most commonly used kernel function and is used for non-linear data. It is a nonlinear kernel function that maps the input data into a higher-dimensional feature space using a Gaussian function.
* Sigmoid kernel: It is used for non-linear data and is defined by two parameters, gamma and coefficient. It is suitable for capturing complex relationships in the data that may not be easily separable in input space.
* **C**

The C hyperparameter, also known as the regularization parameter, controls the trade-off between achieving a low training error and a low testing error. A smaller C encourages a simpler decision boundary (higher bias, lower variance), while a larger C allows for a more complex decision boundary (lower bias, higher variance). Tuning C helped to find the right balance for the model.

Possible values: [0.1, 1, 10,100]

* **Gamma**

The gamma hyperparameter defines the influence of a single training example. A low value of gamma implies a high influence, resulting in a more localized decision boundary, while a high value makes the influence more global. The options 'scale' and 'auto' are heuristic methods to set gamma based on the input data. Including specific values allows fine-tuning based on the dataset characteristics.

Possible values: ['scale', 'auto', 0.1, 0.01, 0.001]

* If gamma is set to a float value, it directly specifies the kernel coefficient.
* If gamma is set to 'scale' (default), the kernel coefficient is set to 1 / (n\_features \* X.var()),

where n\_features is the number of features in the input data X.

X.var() is the variance in the X data

* If gamma is set to 'auto', the kernel coefficient is set to 1 / n\_features.

## Hyperparameter Tuning of LSTM model

A parameter grid was created defining the possible values for specific hyperparameters that were tuned during the hyperparameter optimization process.s  
For the report, describing the tuning of an LSTM model's parameters within the context of a scikit-learn framework using the Keras wrapper, you might write the following in past tense:

In the conducted experiment, a deep learning model was defined and subsequently integrated into the scikit-learn workflow using the Keras library, which is an open-source neural network library written in Python. A custom function, which instantiated a Keras Sequential model, was developed to specify the architecture of the Long Short-Term Memory (LSTM) network. This architecture included a specified number of LSTM layers and units, an output layer, as well as the choice of activation functions and optimizer.

The model creation function was then passed to the KerasClassifier wrapper, a utility from Keras that allows deep learning models to be used as classifiers in scikit-learn. This wrapper made it possible to leverage scikit-learn's extensive capabilities in hyperparameter optimization and model evaluation. A comprehensive grid search was performed using GridSearchCV, a scikit-learn method, to systematically explore a range of hyperparameters and determine the most effective combinations

* **Epoch**

Epoch is the number of times the entire training dataset is passed forward and backward through the neural network. Training a neural network involves adjusting the weights of the model to minimize the error on the training dataset. One epoch corresponds to one complete cycle through the entire training dataset. Multiple epochs are often needed to allow the model to learn patterns and generalize from the training data.

* **Batch** **Size**

Batch size defines the number of training examples utilized in one iteration. During training, the dataset is divided into batches, and each batch is processed through the network. Batch training helps in parallelizing the training process, making it more computationally efficient. It also introduces a form of regularization by updating the model's weights based on a subset of the data rather than the entire dataset.

* **Optimizer**

The optimization algorithm used to update the weights of the neural network. Optimizers play a crucial role in determining how quickly the model learns and converges. Popular optimizers include:

* + Adam: Adam is an adaptive learning rate optimization algorithm that combines ideas from both momentum and RMSprop. It maintains two moving averages for each weight: the first moment (mean) and the second moment (uncentered variance). The algorithm adapts the learning rates for each parameter based on the magnitude of these moving averages.
  + RMSprop: RMSprop is an adaptive learning rate optimization algorithm that adjusts the learning rates for each parameter based on the root mean square of recent gradients. It helps handle issues like vanishing or exploding gradients.
  + SGD (Stochastic Gradient Descent): SGD is a basic optimization algorithm that updates weights based on the gradient of the loss function. It is a stochastic version of gradient descent where each iteration involves a randomly selected subset (mini-batch) of the training data
* **Learning** **Rate**

Learning rate is a hyperparameter that controls the step size during optimization. Learning rate determines the size of the steps taken during optimization. Too high a learning rate may cause the model to overshoot the optimal weights, while too low a learning rate may result in slow convergence or getting stuck in local minima. It is a critical hyperparameter that influences the training process.

* **Loss** **function**

Loss function is a function that quantifies how well the model is performing by measuring the difference between predicted and actual values. During training, the goal is to minimize this function. For classification tasks with multiple classes, common choices include:

* + Categorical Crossentropy: Categorical Crossentropy is used when the target variable is one-hot encoded, meaning that each example in the dataset belongs to exactly one class, and the classes are mutually exclusive
  + Sparse Categorical Crossentropy: Sparse Categorical Crossentropy is used when the target variable is integer-encoded, meaning that each example in the dataset is represented by an integer corresponding to the class label. This loss function is more suitable when you have integer labels and don't want to one-hot encode them explicitly.

# RESULTS AND ANALYSIS

## Hyperparameter Tuning Report of SVM model

The parameter tuning report outlines the top three and bottom three combinations of hyperparameters along with their corresponding accuracies.

Table 7‑1: Parameter Tuning Report

Top Three Combinations:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rank | Kernel | C | Gamma | Accuracy |
| 1 | rbf | 10 | scale | 0.9838 |
| 2 | rbf | 10 | auto | 0.9838 |
| 3 | rbf | 10 | 0.1 | 0.9826 |

Bottom Three Combinations:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rank | Kernel | C | Gamma | Accuracy |
| 1 | poly | 0.1 | 0.001 | 0.4959 |
| 2 | polyy | 1 | 0.001 | 0.4959 |
| 3 | poly | 10 | 0.001 | 0.4959 |

In our quest to fine-tune the hyperparameters of our Support Vector Machine (SVM) model, diverse configurations were explored, revealing notable trends in performance. The Radial Basis Function (RBF) kernel emerged as the star performer, consistently achieving top accuracies around 98.38%. Particularly noteworthy were configurations featuring a high regularization parameter (C=10) paired with either 'scale' or 'auto' for the gamma parameter, affirming the RBF kernel's adaptability to intricate decision boundaries. In contrast, the Polynomial (Poly) kernel, known for capturing non-linear relationships, faltered with a configuration of C=0.1 and gamma=0.001, yielding an accuracy of 49.59%. These results suggest a nuanced interplay between kernel types and regularization, with the RBF kernel exhibiting a robust preference for higher regularization values in capturing the dataset's underlying patterns.

The findings underscore the importance of careful hyperparameter selection in SVM modeling, emphasizing the superiority of the RBF kernel with elevated regularization. The notable accuracy discrepancies between kernel types highlight the need for a tailored approach, steering future investigations towards a deeper understanding of dataset-specific characteristics. Insights gained from this hyperparameter tuning exercise will play a pivotal role in finalizing our SVM model, ensuring it leverages the optimal combination of kernel functions and regularization parameters for superior predictive performance on the given dataset.

## Accuracy with Features

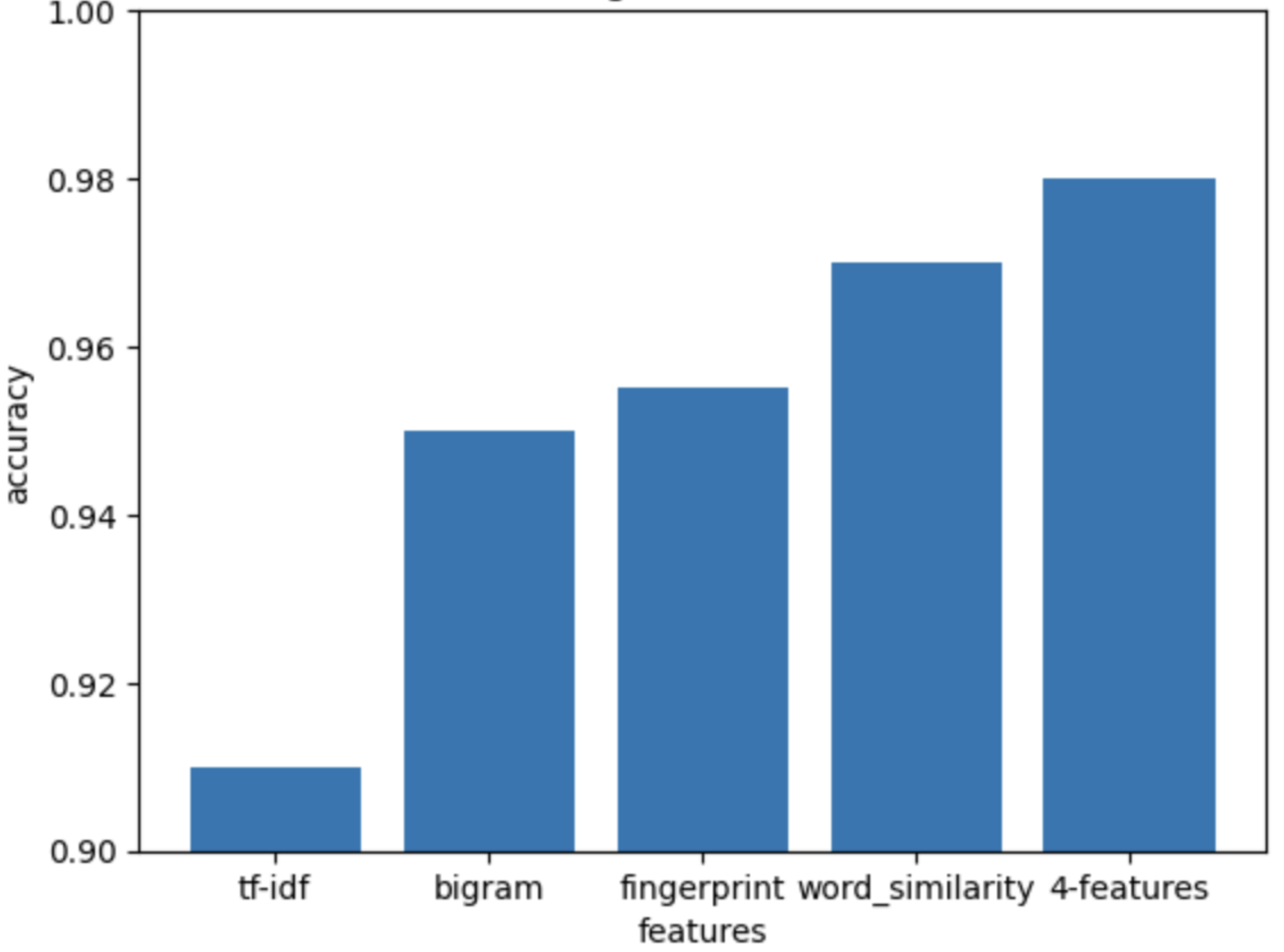


Figure 7‑1: Accuracy vs Feature Graph

The SVM (Support Vector Machine) model's performance in plagiarism detection was evaluated using various feature sets. When individual feature sets were considered, it was observed that using TF-IDF features alone resulted in an accuracy of 91%. These features are based on term frequency-inverse document frequency and provide insights into term frequency distributions in the documents. Subsequently, using only bigram features improved accuracy to 95%, highlighting the significance of capturing word relationships within documents. Additionally, we assessed the model using fingerprint features alone, achieving an accuracy of 95.5%. Fingerprint features are distinctive representations of document content and excel at identifying similarities. Notably, the use of word similarity features exclusively yielded an impressive accuracy of 97%. These features evaluate word-level similarity, offering insights into the semantic relationships between words.

However, the most substantial enhancement in accuracy was observed when all feature types were combined. In this holistic approach, the model achieved an accuracy of 98%. This result underscores the synergistic nature of these feature sets, collectively capturing diverse aspects of document similarity. The SVM model, with this comprehensive feature set, demonstrates robust capabilities in plagiarism detection, making it a highly effective tool for identifying instances of plagiarism across various domains.

## Confusion Matrix

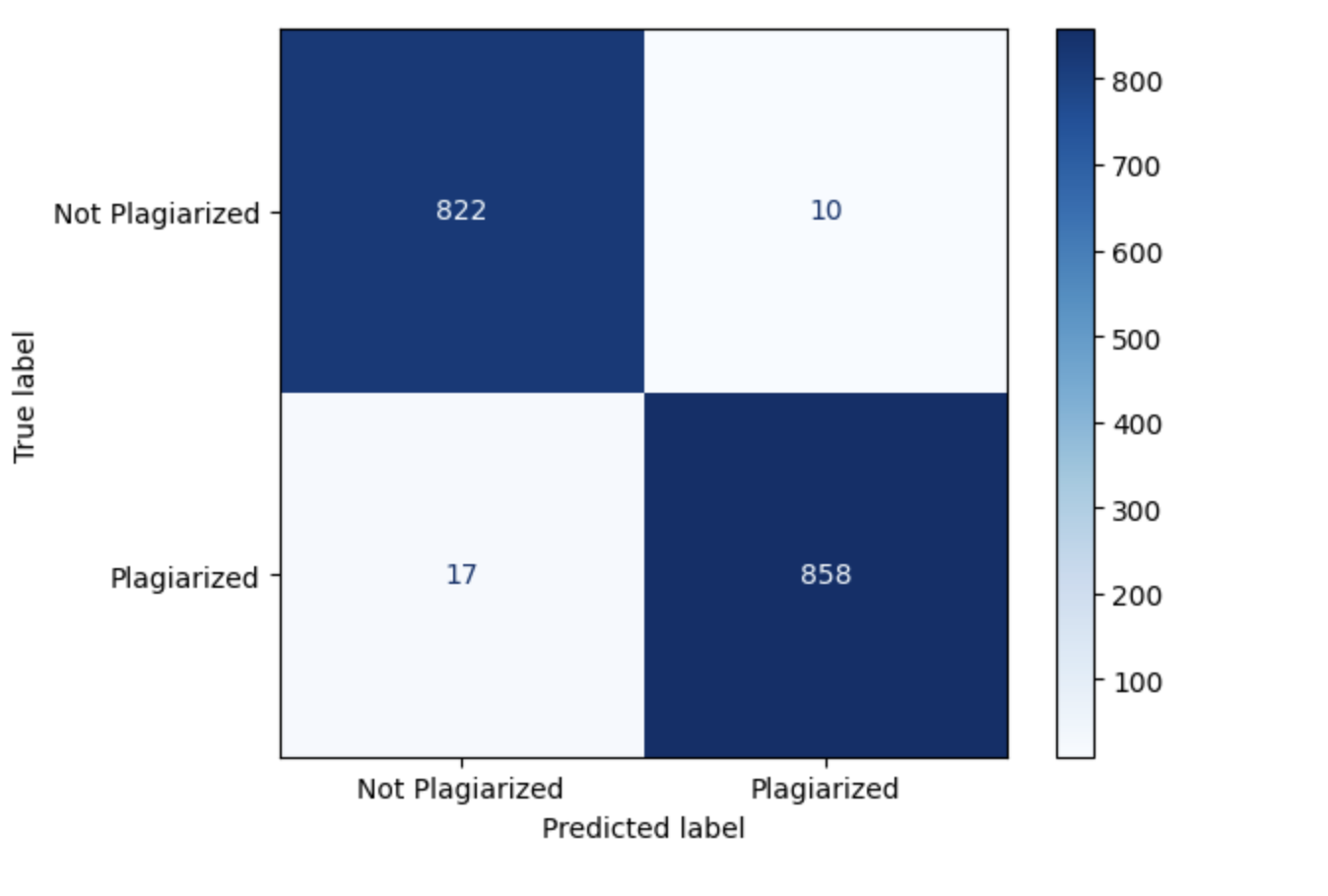


Figure 7‑2: Confusion Matrix

The classification model's performance was assessed using a confusion matrix, which provided a visual and quantitative depiction of the model's predictive accuracy. The matrix displayed two classes: 'Not Plagiarized' and 'Plagiarized.' The top-left quadrant of the matrix contained the number 822, indicating that the model correctly identified 822 instances as 'Not Plagiarized.' The bottom-right quadrant showed the number 858, signifying that 858 instances were accurately classified as 'Plagiarized.'

However, the model was not entirely infallible. The top-right quadrant revealed that 10 instances were incorrectly predicted as 'Plagiarized' when they were not, representing false positives. Conversely, the bottom-left quadrant indicated 17 false negatives, where instances were wrongly classified as 'Not Plagiarized' despite being 'Plagiarized.'

The overall accuracy of the model was derived from the sum of the true positives and true negatives, divided by the total number of instances. The false positives and false negatives were critical in understanding the model's precision and recall, respectively. Precision measures the accuracy of the positive predictions, while recall, also known as sensitivity, measures the ability of the classifier to find all the positive instances.

* **Accuracy**: The model's accuracy was determined to be approximately 98.38%. This metric signified that, in general, the model was highly capable of correctly classifying instances as either plagiarized or not. Accuracy is a measure of the model's overall correctness across all classes and is calculated by dividing the sum of true positives and true negatives by the total number of instances.
* **Recall**: The recall achieved by the model stood at around 97.97%. This is a measure of the model's ability to correctly identify positive instances—in this case, instances of plagiarism. A high recall indicates that the model was able to detect the majority of plagiarized cases. It is calculated specifically by dividing the number of true positives by the sum of true positives and false negatives.
* **Precision**: With a precision of approximately 98.80%, the model demonstrated a high likelihood that an instance classified as plagiarized was indeed plagiarized. Precision measures the accuracy of the positive predictions, quantified by dividing the number of true positives by the sum of true positives and false positives.
* **F1 Score**: The F1 score was found to be approximately 98.38%. As a metric that balances precision and recall, the F1 score is especially useful when the cost of false positives and false negatives is high or when there is an imbalance in the class distribution. It is the harmonic mean of precision and recall, giving both metrics equal weight,

It was observed that while the classification model demonstrated high overall performance, it exhibited specific limitations, particularly in the detection of heavily paraphrased instances. The subtleties involved in such cases, where the original text was extensively reworded or restructured, were not effectively recognized by the model.

## Precision Recall Curve

Precision is the measure of the correctness achieved in positive prediction — that is, the ratio of true positives to the sum of true positives and false positives. Recall, or sensitivity, measures the ability of the classifier to find all the positive samples, calculated as the ratio of true positives to the sum of true positives and false negatives.

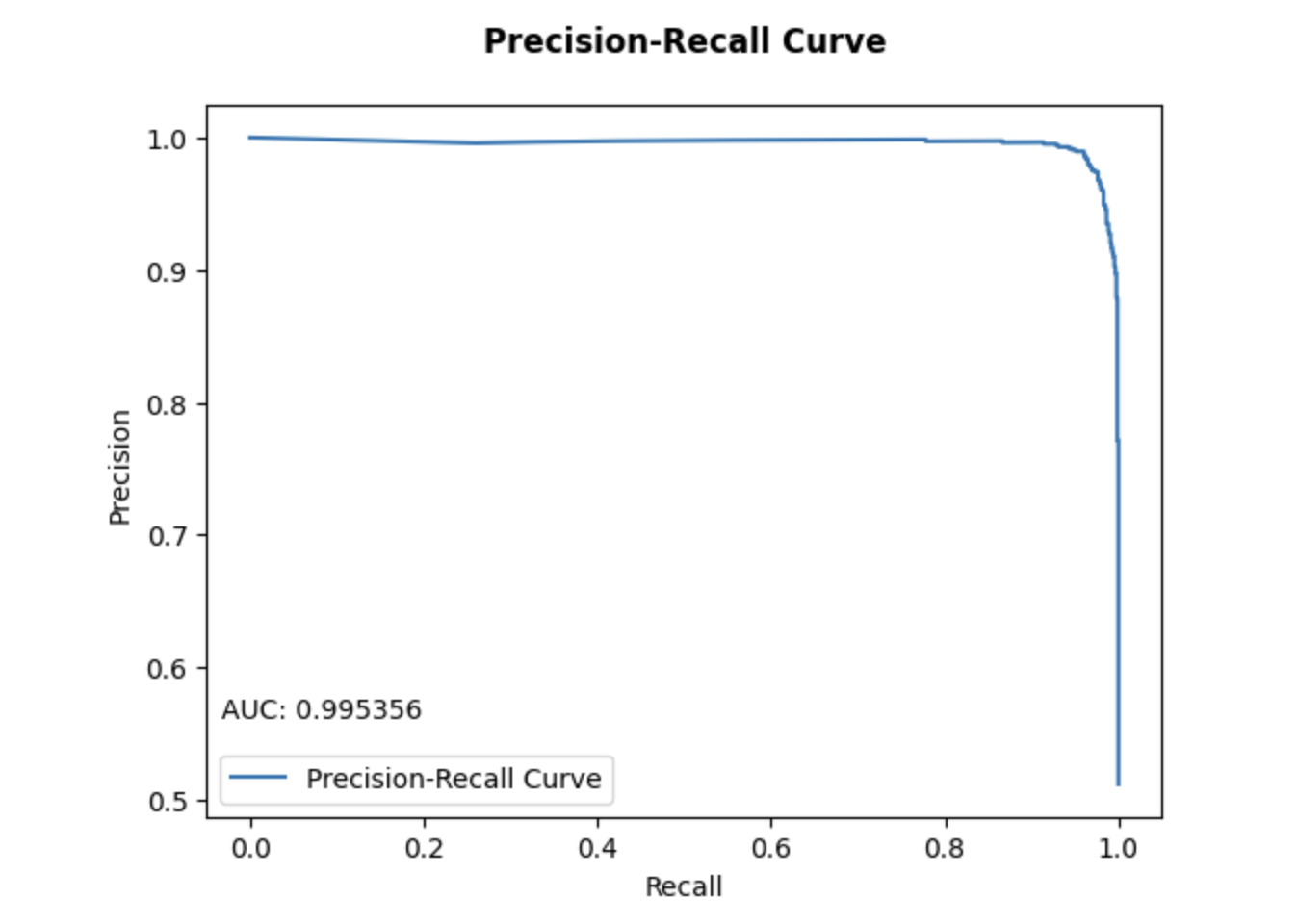


Figure 7‑3: Precision Recall Curve

The curve started from the right, indicating that at lower threshold levels (where the model is more lenient in predicting a positive class), both precision and recall were high. This suggested that the model was able to achieve high precision without compromising recall, which is an ideal scenario in classification tasks. As the curve progressed to the left, indicating an increase in the threshold (where the model becomes stricter in predicting a positive class), precision slightly decreased while recall remained high. This slight decrease in precision while maintaining a high recall suggested that the model began to misclassify a few more negative instances as positive (false positives) but continued to identify most of the positive instances correctly.

The area under the curve (AUC) was noted to be approximately 0.9954, which was a near-perfect score. This high AUC value indicated that the SVM classifier performed exceptionally well in distinguishing between the classes across different thresholds. A high AUC value is typically indicative of a model with both high precision and high recall across various threshold settings, which implies that the classifier is effective at yielding a low number of false positives and false negatives.

In summary, the Precision-Recall Curve provided evidence that the SVM model had exhibited a remarkable capacity to maintain a balance between precision and recall, ensuring that the positive predictions were accurate and that the majority of actual positive instances were correctly identified across different decision thresholds. This analysis was crucial in understanding the model's ability to handle the trade-off between precision and recall, especially in scenarios where the cost of false positives and false negatives could have significant consequences.

## Precision Recall Curve

Precision is the measure of the correctness achieved in positive prediction — that is, the ratio of true positives to the sum of true positives and false positives. Recall, or sensitivity, measures the ability of the classifier to find all the positive samples, calculated as the ratio of true positives to the sum of true positives and false negatives.

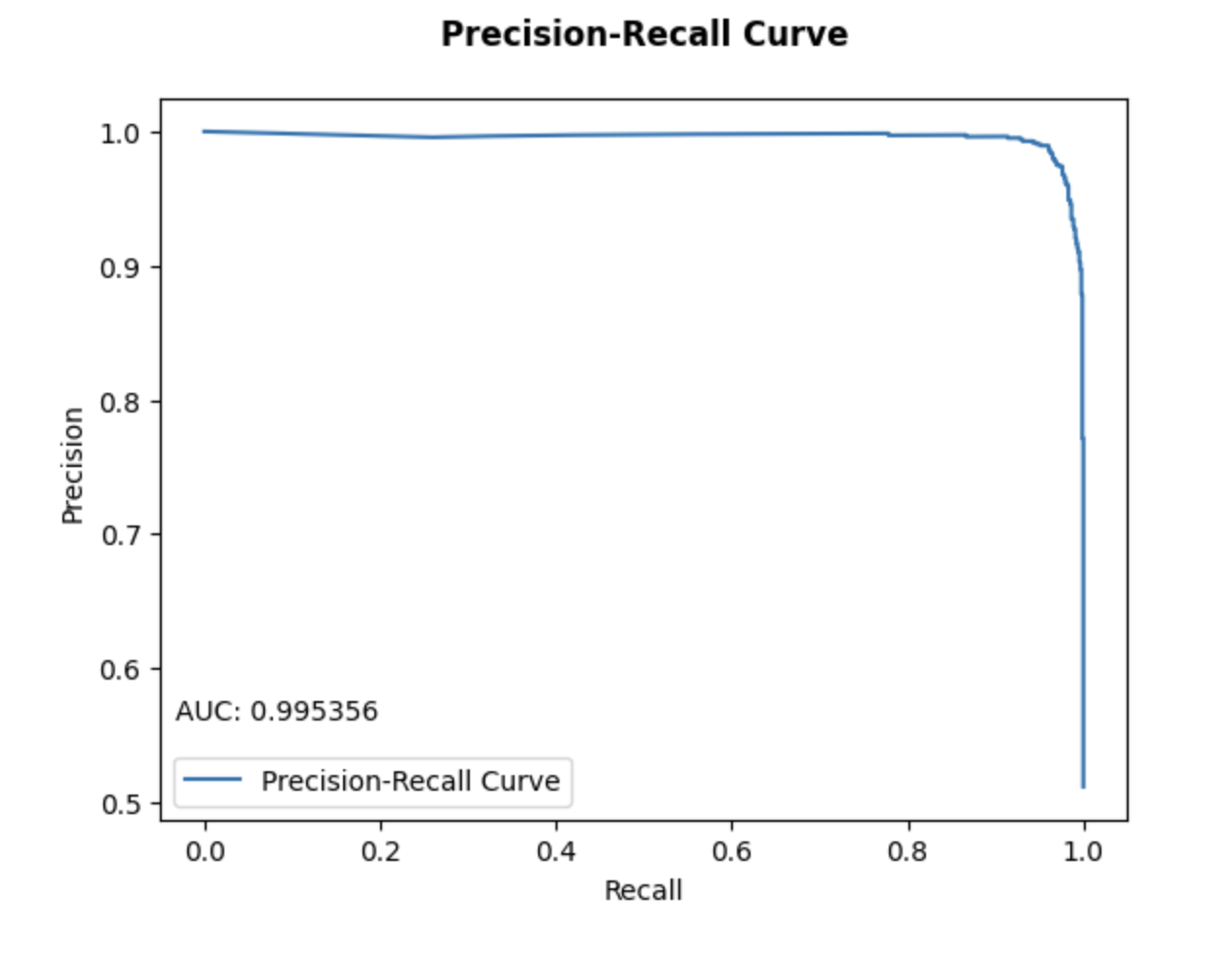


Figure 7‑4: Precison Recall Curve

The curve started from the right, indicating that at lower threshold levels (where the model is more lenient in predicting a positive class), both precision and recall were high. This suggested that the model was able to achieve high precision without compromising recall, which is an ideal scenario in classification tasks. As the curve progressed to the left, indicating an increase in the threshold (where the model becomes stricter in predicting a positive class), precision slightly decreased while recall remained high. This slight decrease in precision while maintaining a high recall suggested that the model began to misclassify a few more negative instances as positive (false positives) but continued to identify most of the positive instances correctly.

The area under the curve (AUC) was noted to be approximately 0.9954, which was a near-perfect score. This high AUC value indicated that the SVM classifier performed exceptionally well in distinguishing between the classes across different thresholds. A high AUC value is typically indicative of a model with both high precision and high recall across various threshold settings, which implies that the classifier is effective at yielding a low number of false positives and false negatives.

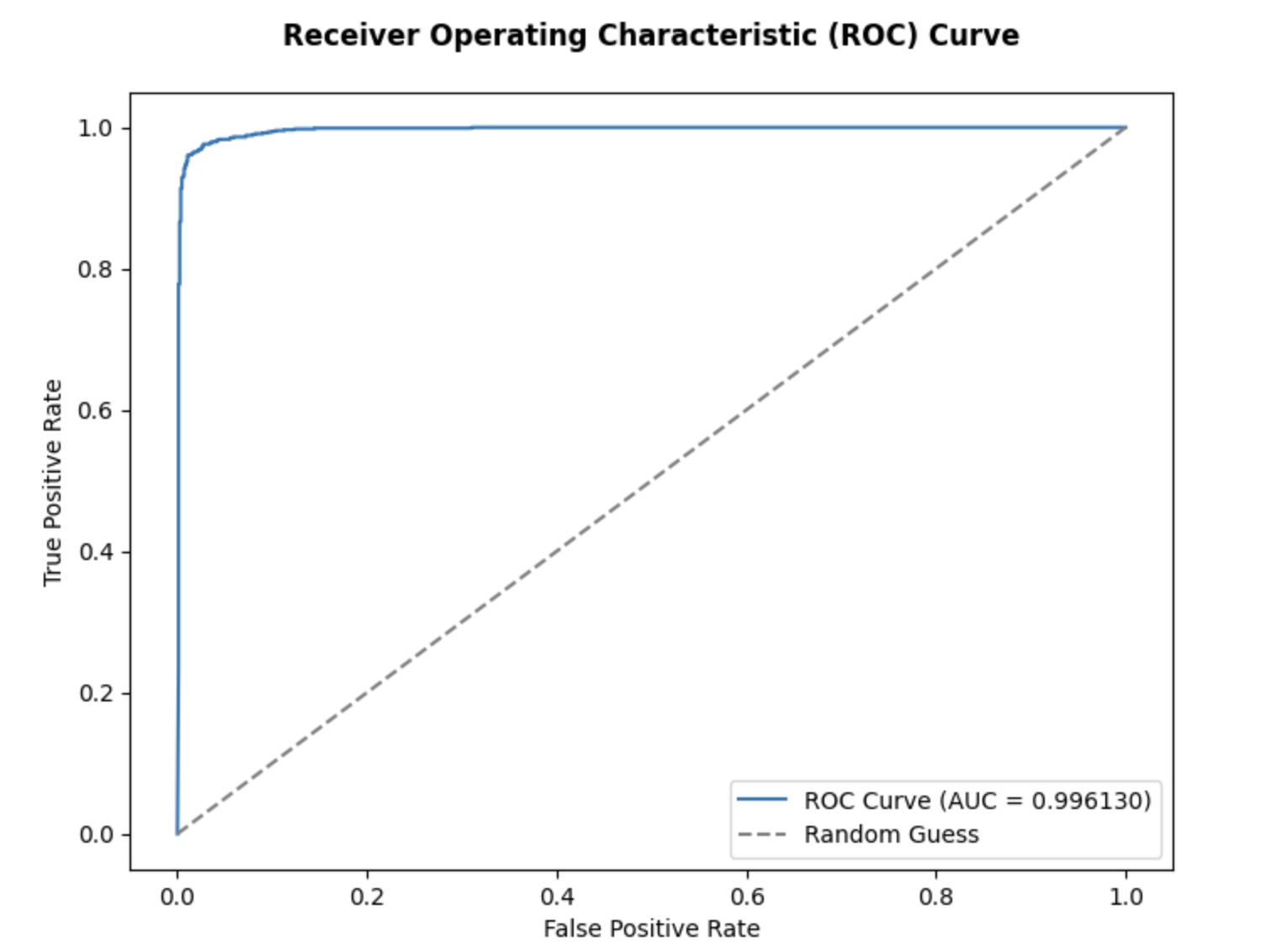
In summary, the Precision-Recall Curve provided evidence that the SVM model had exhibited a remarkable capacity to maintain a balance between precision and recall, ensuring that the positive predictions were accurate and that the majority of actual positive instances were correctly identified across different decision thresholds. This analysis was crucial in understanding the model's ability to handle the trade-off between precision and recall.

## Receiver Operating Curve

The Receiver Operating Characteristic (ROC) curve was used to evaluate the performance of the plagiarism detection model. This curve plotted the True Positive Rate (TPR) on the y-axis against the False Positive Rate (FPR) on the x-axis at various threshold settings.

In the context of the ROC curve for the plagiarism detection model:

1. The **True Positive Rate (TPR)**, also known as recall or sensitivity, measured the proportion of actual plagiarized instances that were correctly identified by the model. It was defined as the ratio of true positives to the sum of true positives and false negatives.
2. The **False Positive Rate (FPR)** measured the proportion of non-plagiarized instances that were incorrectly identified as plagiarized. It was defined as the ratio of false positives to the sum of false positives and true negatives.



The curve started near the top-left corner of the plot, indicating a high TPR and a low FPR for the model at lower threshold levels. This suggested that the model was capable of correctly identifying a high number of plagiarized instances while keeping the number of false alarms (non-plagiarized instances incorrectly labeled as plagiarized) very low.

As the threshold increased, both the TPR and FPR showed little to no increase, which was indicative of the model's robustness. The model maintained a high level of sensitivity to plagiarism without being misled by non-plagiarized content, even as the criteria for classifying an instance as plagiarism became stricter.

The area under the ROC curve (AUC) was a quantitative measure of the model's overall ability to discriminate between the plagiarized and non-plagiarized classes. The AUC was noted to be approximately 0.996130, which was very close to the ideal score of 1. This high AUC value underscored the excellent performance of the model, showing that it had a high true positive rate across all thresholds while maintaining a low false positive rate.

In comparison, the dashed line represented the ROC curve of a random classifier (equivalent to random guessing), which would produce a diagonal line from the bottom left to the top right of the plot. The significant distance between the model's ROC curve and the line of randomness further emphasized the model's discriminative power.

In conclusion, the ROC curve from the historical data demonstrated that the plagiarism detection model was highly effective, with an impressive ability to distinguish between plagiarized and non-plagiarized instances across various decision thresholds. This capability was critical for a plagiarism detection tool, where the cost of false positives and false negatives could be significant.

## Cross Validation Result

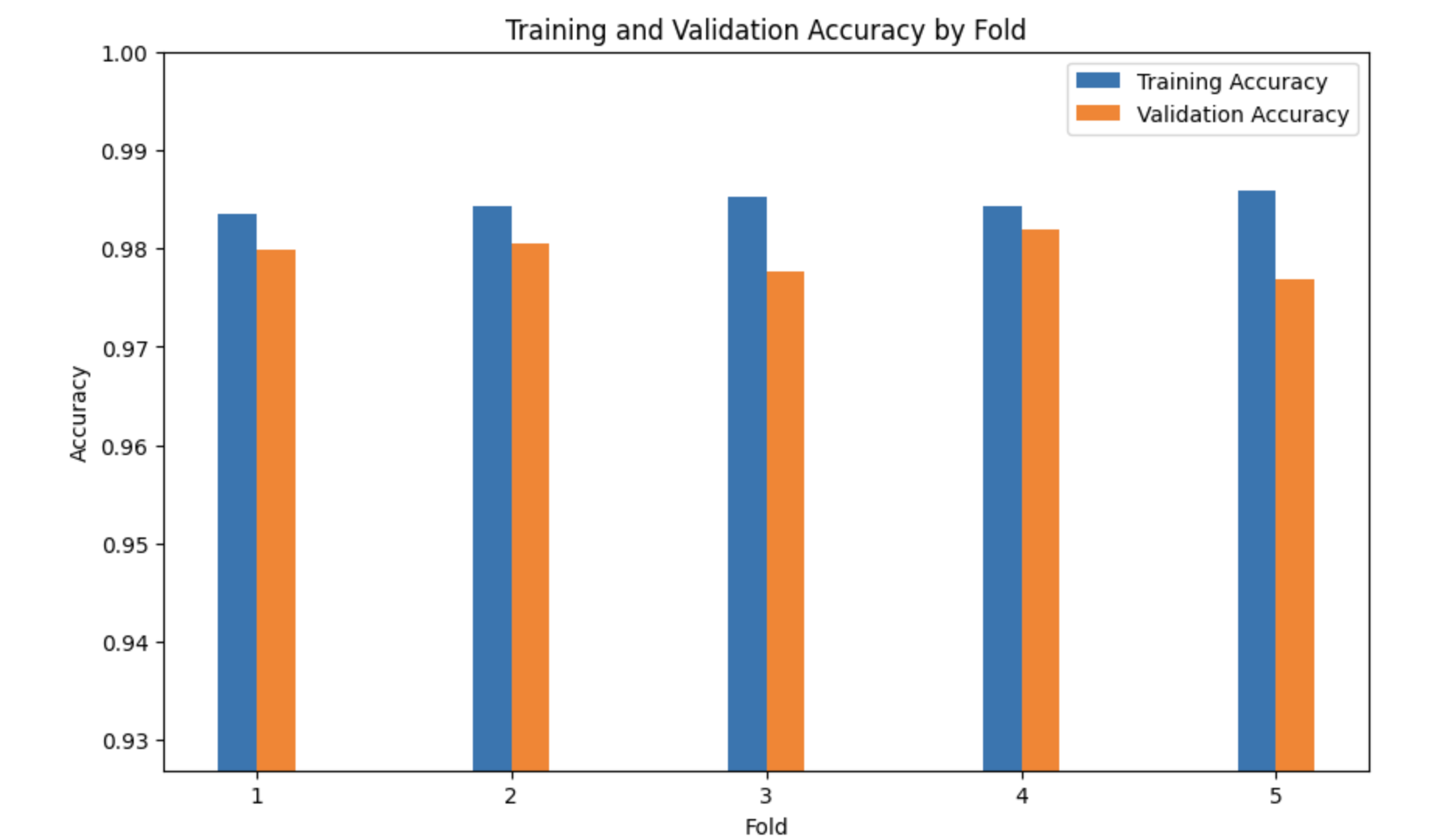


Figure 7‑5: Cross Validation Results

In the evaluation of the model using 5-fold cross-validation, the data was partitioned into five distinct subsets. The model underwent training and validation cycles five times, each with a different fold serving as the validation set while the remaining folds constituted the training set. This methodical approach provided insights into the model's performance and its ability to generalize to unseen data.

The bar chart, which depicted the results of this cross-validation, revealed high training accuracy across all five folds. This high level of accuracy indicated that the model learned the patterns in the training data well. Correspondingly, the validation accuracy for each fold was also high, albeit slightly lower than the training accuracy, a common and expected outcome due to the model's familiarity with the training data as opposed to unseen validation data.

An essential observation from the cross-validation was the consistency in the model's performance across all folds. The minor discrepancies between training and validation accuracy suggested that overfitting was not a significant issue for this model. Such consistency pointed to the model's stable and reliable performance, regardless of the specific data subset it was trained or validated on.

Furthermore, the validation accuracies achieved were indicative of the model's robust generalizability. The fact that each fold produced a high validation accuracy implied that the model was well-tuned and capable of maintaining its predictive quality across various data samples.

Conclusively, the 5-fold cross-validation process demonstrated that the model was not only adept at learning from the training data but also possessed a commendable ability to generalize its learning to new, unseen data. The consistent and high accuracy across training and validation phases underscored the model's potential for reliable deployment in practical settings.

## POS tagger

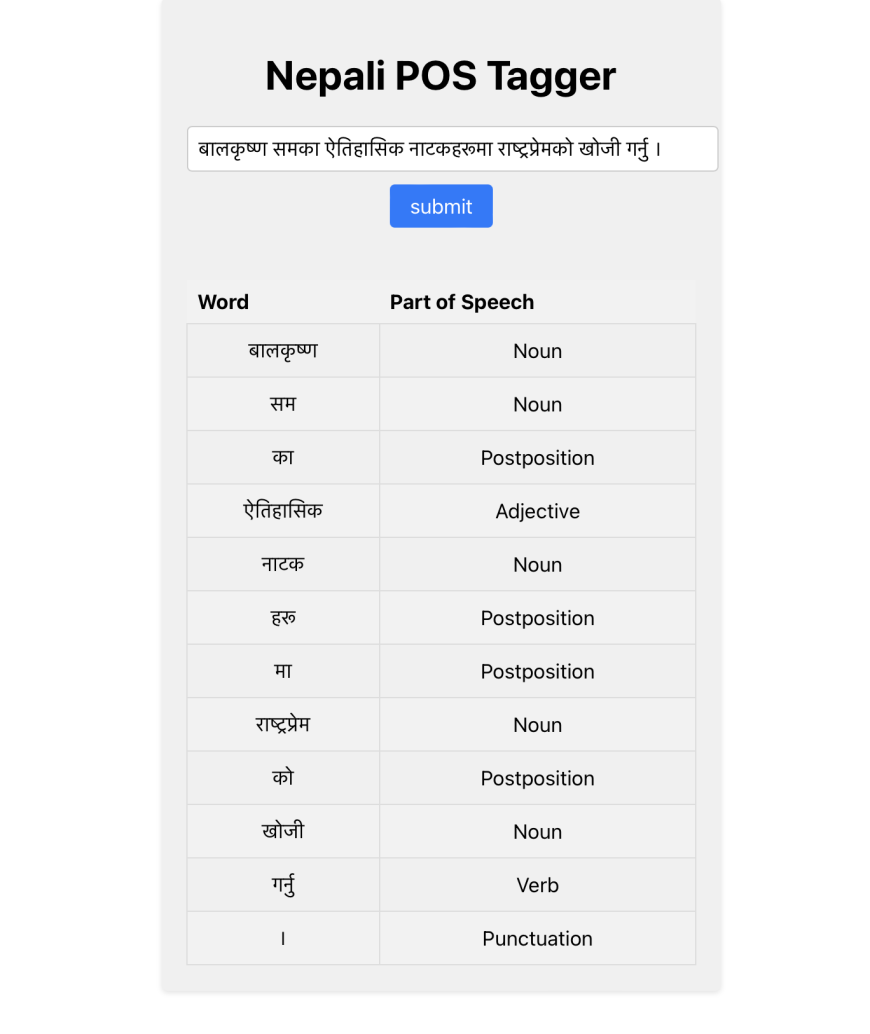


Figure 7‑6: POS Tagger Output Interface

### Hyperparameter Tuning Report of POS Model

In the hyperparameter tuning experiment, the different configurations of optimizers and learning rates were tested corresponding to different batch size and number of epochs and it was found that the best result was achieved with the RMSprop optimizer with a learning rate of 0.001 when using a batch size of 128. The key findings can be summarized as

The RMSprop optimizer, with a learning rate of 0.001, achieved the highest test accuracy of 98.12% with loss of 0.0110. A lower learning rate (0.001) appears to be more effective than higher learning rates (0.01 and 0.1) for the chosen optimizers. The choice of RMSprop over Adam seems beneficial for this specific task and dataset. The selected batch size of 128 resulted in the best overall performance, indicating that it strikes a good balance between computational efficiency and model accuracy.

Categorical Crossentropy was used as the target variable were one-hot encoded, meaning that each example in the dataset belongs to exactly one class, and the classes are mutually exclusive.

Table 7‑2: Hyperparameter of LSTM model

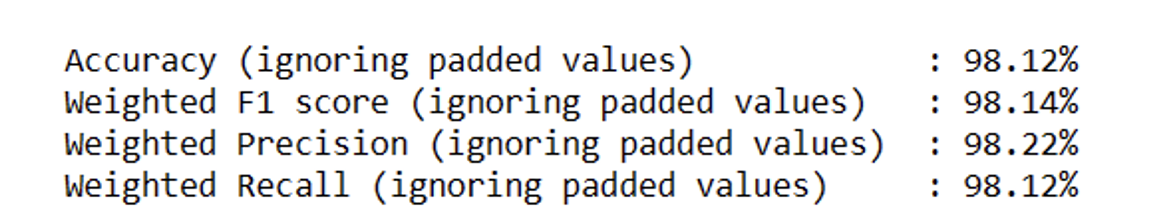
|  |  |
| --- | --- |
| **Attribute** | **Value** |
| Epochs | 5 |
| Batch Size | 64 |
| Optimizer | RMSprop |
| Learning Rate | 0.001 |
| Loss function | Categorical Crossentropy |

### Performance Metrics of POS tagging Model

To accurately measure the performance metrics of the POS model, we excluded predictions for padded tokens, ensuring accurate evaluation for actual words. The metrics were calculated on a test set representing 30% of the total dataset, providing unbiased assessments of the model's performance on unseen data.

* **Accuracy**: The model achieved an accuracy of 98.12%, indicating that it correctly tagged POS labels for approximately 98 out of 100 words.
* **F1** **Score**: The weighted average F1 score of 98.14% implies a balanced performance, considering both precision and recall. This indicates that the model effectively minimized false positives and false negatives, resulting in a robust performance.
* **Precision**: The precision score of 98.22% means that the model correctly predicted the POS tags for around 98.22 % of the predicted instances. It has a low false positive rate, minimizing incorrect predictions for specific POS tags.
* **Recall**: With a recall score of 98.12%, the model effectively captured approximately 98.12% of the actual positive instances (ground truth). This suggests that the model is proficient at identifying instances of each POS tag, minimizing false negatives.

Table 7‑3: Performance metric for LSTM



Overall, the LSTM model exhibited exceptional performance in POS tagging, delivering impressive accuracy, F1 score, precision, and recall. It proficiently assigned POS labels to the majority of words in the dataset, striking a balance between precision and recall. This highlights its reliability and effectiveness in accurately tagging words with their corresponding POS labels. As a result, the model shows great promise for natural language processing tasks and the calculation of word similarity values in our plagiarism detection project for Nepali theses.

### Loss Curve for LSTM Model

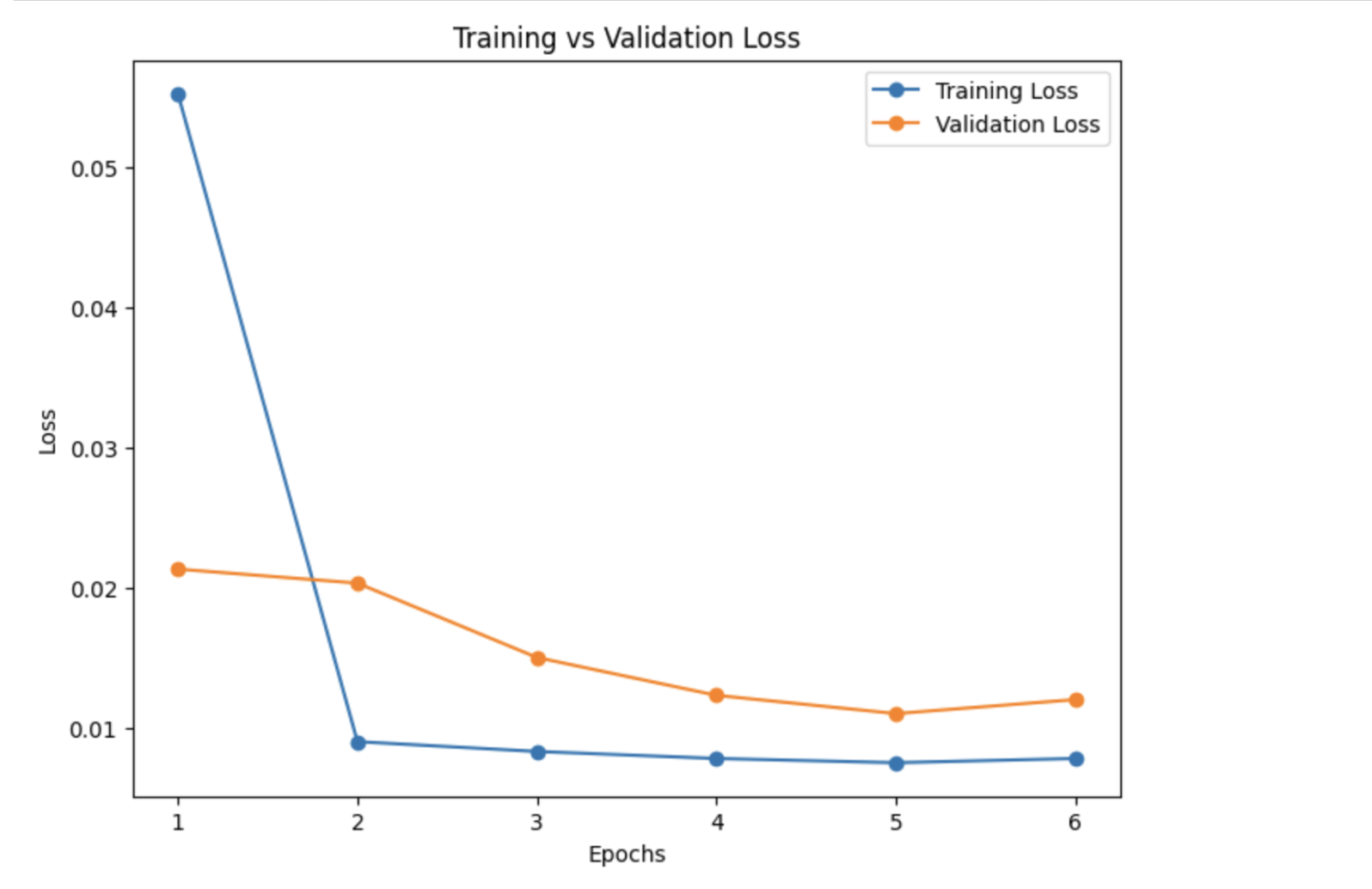


Figure 7‑8: Training vs Validation Loss

It was observed that both the training and validation loss were found to be decreasing steadily. Initially, at epoch 1, the training loss stood at approximately 0.0552, while the validation loss was around 0.0213. As the training progressed, both losses exhibited a consistent downward trend. By epoch 4, the training loss had reduced to 0.0078, while the validation loss reached a minimum value of 0.0110 at epoch 5 before slightly increasing to 0.0120 at epoch 6. This pattern indicated that the model was learning effectively from the training data and initially demonstrating good generalization to the validation data.

However, as the model's performance was continuously monitored, it was decided that the validation loss started to exhibit a slight increase, suggesting that the model might be beginning to overfit the training data. In light of this observation and to prevent potential overfitting, the training process was halted at epoch 6. Further analysis and evaluation of the model's overall performance were conducted to make informed decisions about its training and potential overfitting.

## Nepali Synset

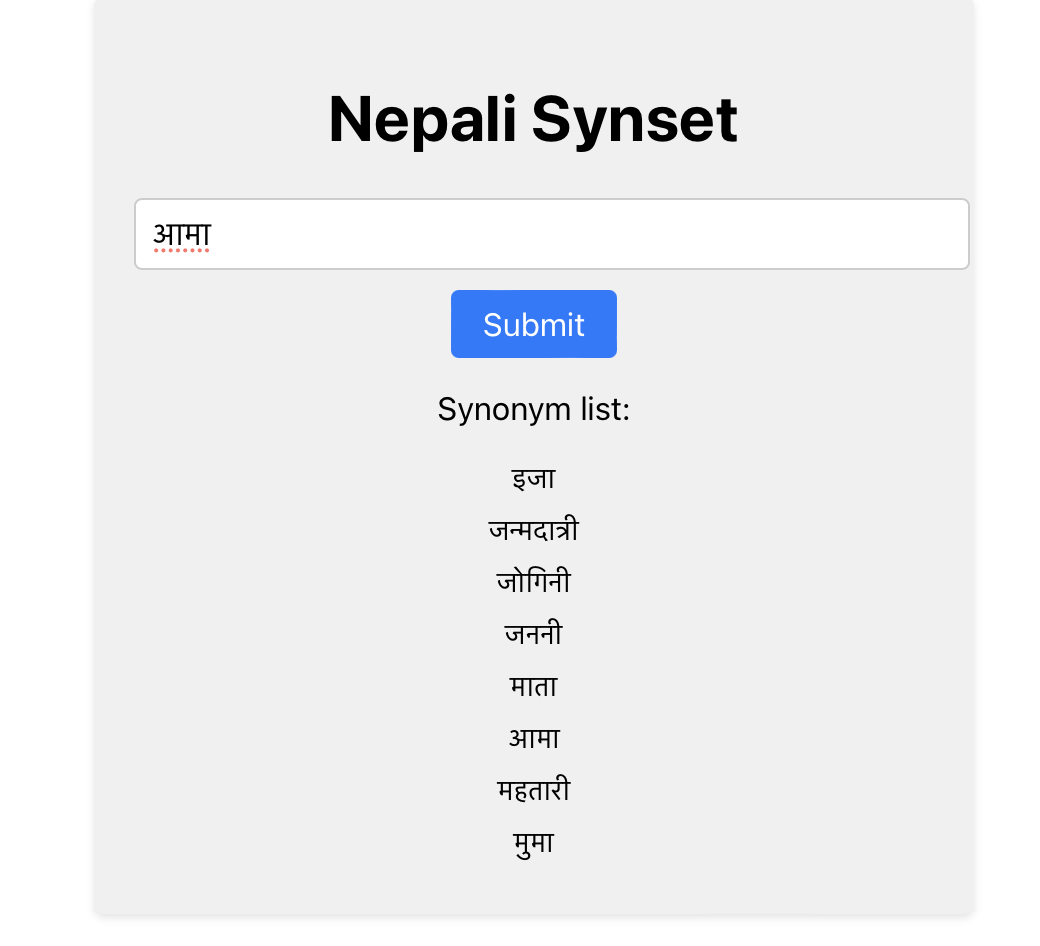


Figure 7‑9: User Interface of Nepali Synset

## Plagiarism Report

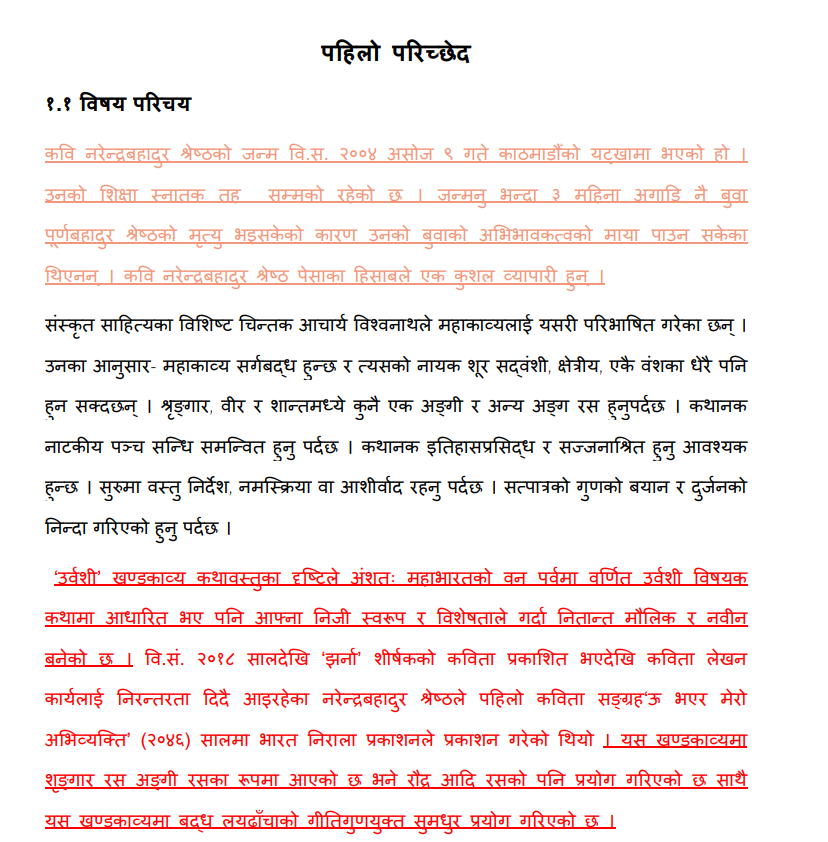


Figure 7‑10: Color Coded Plagiarized text

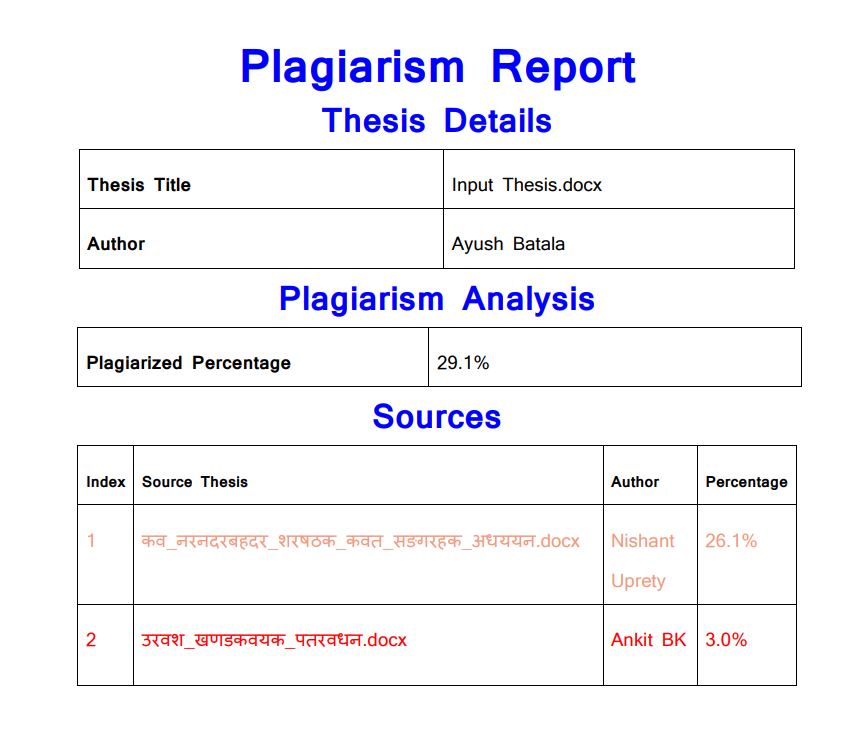


Figure 7‑11: Plagiarism Report

# REMAINING TASK

## Web Application Deployment

The plagiarism detection model will be integrated into a web application created using Django. This integration aims to simplify the process of content verification and originality checks across various domains, including academic institutions.

The online program will feature a user-friendly interface that will enable users to submit their Nepali Theses for plagiarism analysis effortlessly. Upon document submission, the program will preprocess the text and utilize the integrated model to conduct a comprehensive plagiarism evaluation. The input thesis will undergo comparison with other theses within the database, resulting in the generation of an overall similarity score for the input thesis. This deployment will enhance the accessibility and convenience of plagiarism analysis, facilitating users in efficiently assessing the originality of their documents.

## Database Optimization

In the course of the project, the development of the plagiarism detection model has been completed. However, a significant challenge has arisen regarding processing speed, particularly when analyzing theses within the database. Currently, the time required to process these theses exceeds desired benchmarks, resulting in delays in providing results.

To address this issue and enhance the efficiency of the plagiarism detection system, focus will be on developing a more robust and optimized database infrastructure. This improved database will be designed to streamline the retrieval and processing of theses, significantly reducing processing times. The aim is to create a database that not only accelerates document retrieval but also ensures that the overall system operates more swiftly. By developing this enhanced database infrastructure, the anticipation is to deliver substantially faster results to users, ultimately improving the usability and responsiveness of the plagiarism detection system. This optimization aligns with the overarching goal of providing a reliable and efficient solution to plagiarism detection.

# APPENDICES

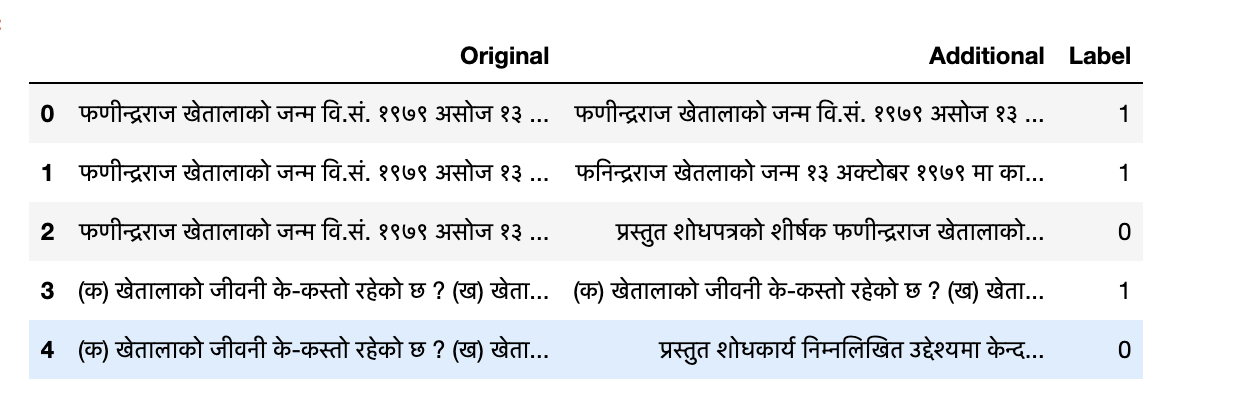
## Appendix A: Project Timeline

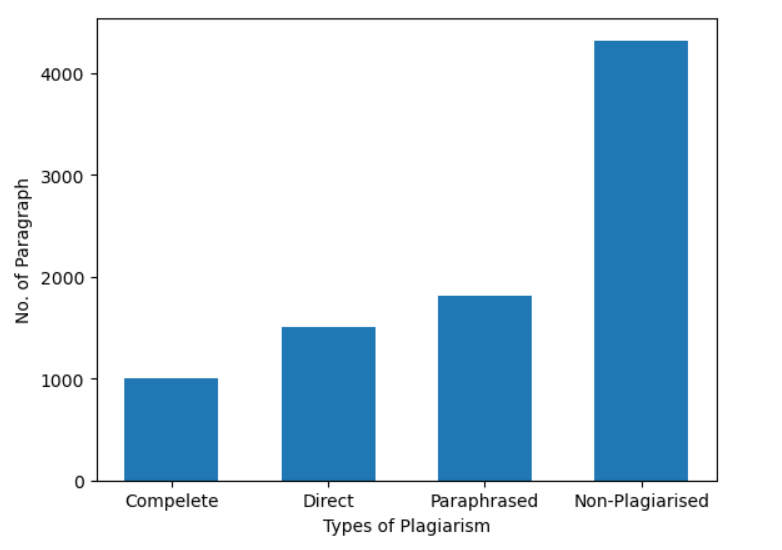
Table 9‑1: Gantt Chart

May 15 July 15 Sep 15 Nov 15 Jan 1 Mar 15 Apr 30

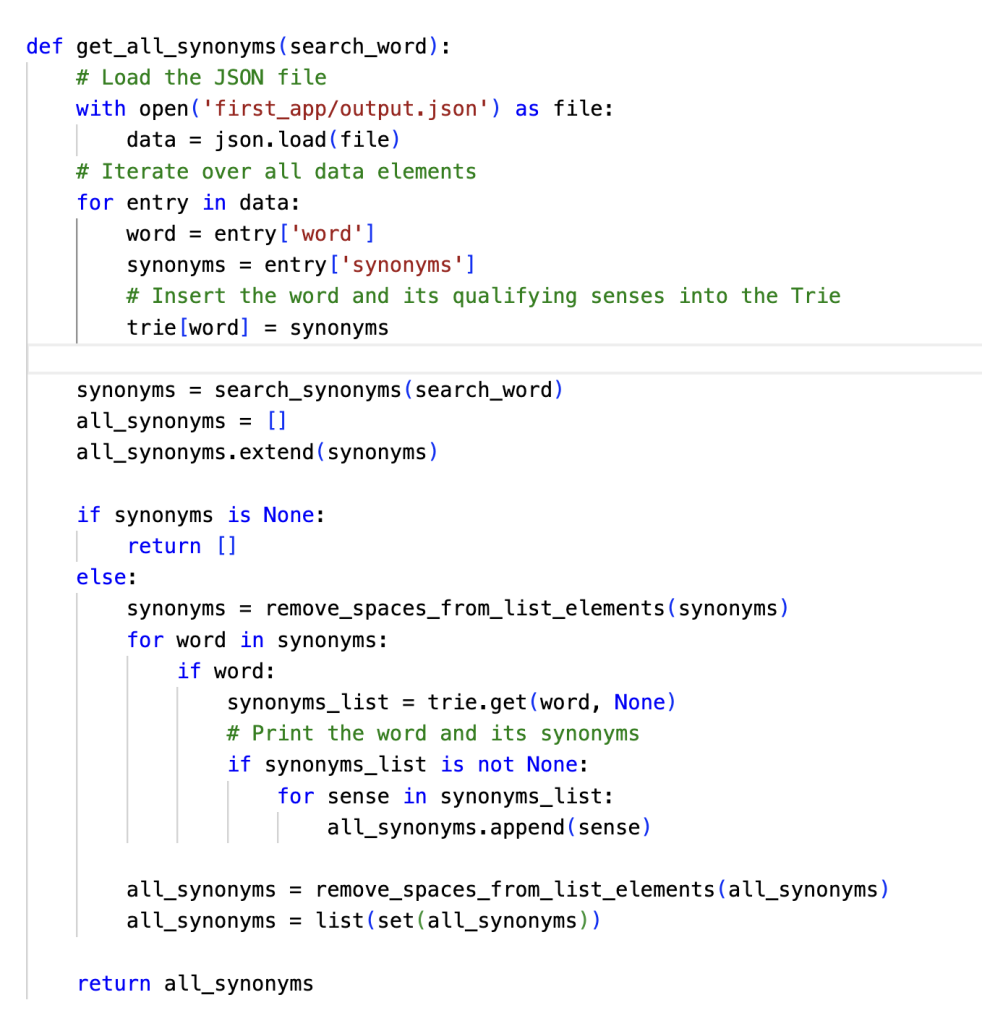
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Research |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Data Collection |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Dataset Preparation |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Model Implementation |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| UI Development |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Prototype Development |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Testing and debugging |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Documentation |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

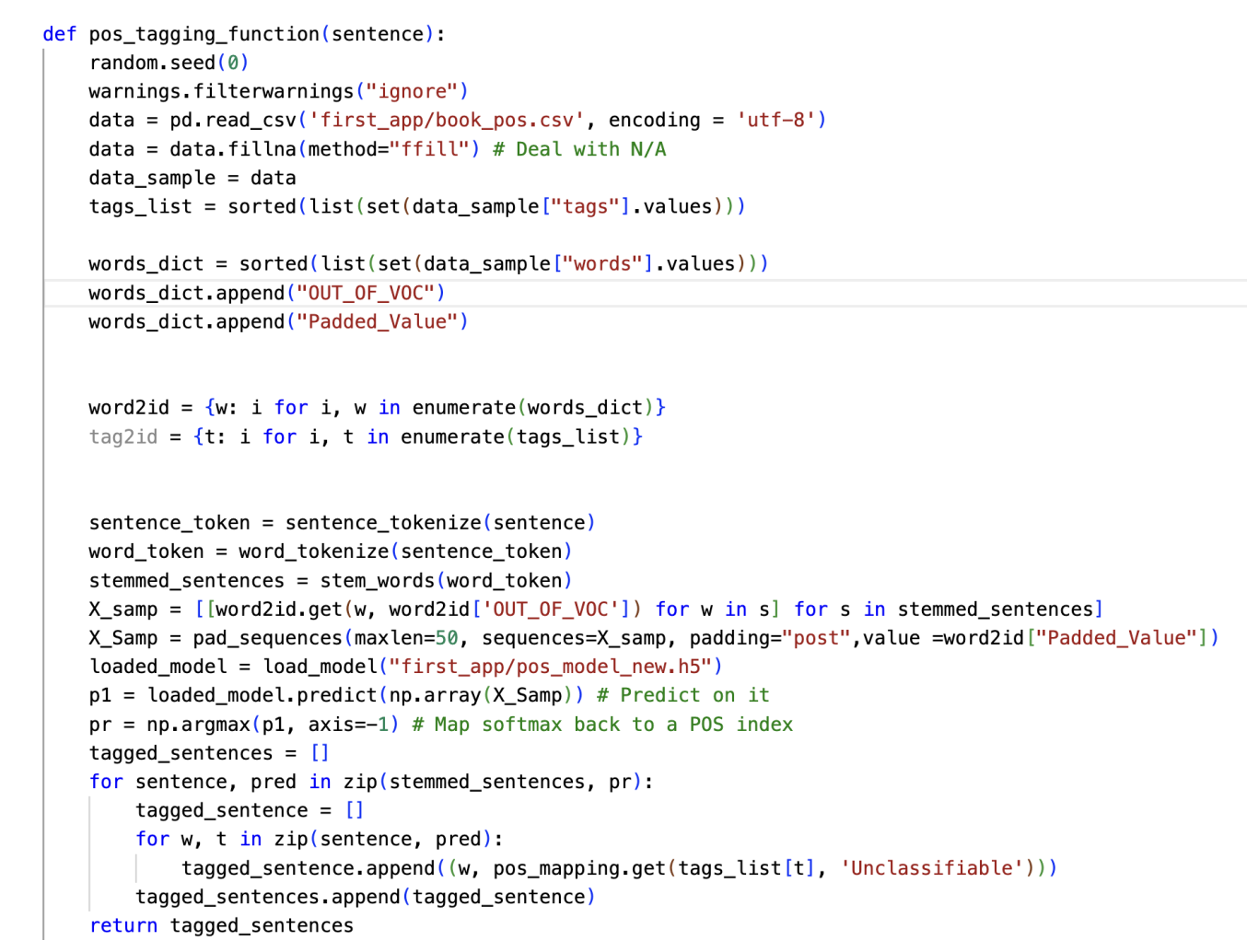
## Appendix B: Nepali Dataset Snippets





## Appendix C: Code Snippets





# References

|  |  |
| --- | --- |
| [1] | D. Harper, "Online Etymology Website," 2 august 2020. [Online]. Available: www.etymonline.com. [Accessed January 2023]. |
| [2] | SAS Institute Inc., "Natural Language Processing (NLP): What it is and why it matters," Analytics Software & Solutions(SAS), [Online]. Available: https://www.sas.com/en\_us/insights/analytics/what-is-natural-language-processing-nlp.html. [Accessed January 2023]. |
| [3] | A. Kumar, "Data Analytics," 31 Jan 2018. [Online]. Available: https://vitalflux.com/quick-introduction-smoothing-techniques-language- models/. [Accessed February 2023]. |
| [4] | S. F. C. a. J. Goodman, "An Empirical Study for Smoothing Techniques for Language Modeling," 1998. |
| [5] | B. K. Bal, "Structure of Nepali Grammar," Kathmandu University, 2004. |
| [6] | Z. Ceska, "Plagiarism detection based on singular value decomposition," *Advances in Natural Language Processing,* pp. 108-119, 2008. |
| [7] | A. B.-C. a. P. Rosso, "On Automatic Plagiarism Detection Based on n-Grams Comparision," Natural Language Engineering Lab, Valencia. |
| [8] | C. Sitaula, "A Hybrid Algorithm for Stemming of Nepali Text," Kathmandu, 2013. |
| [9] | R. K. B. a. A. K. Timalsina, "Plagiarism detection framework using Monte Carlo based Artificial Neural Network for Nepali language," in *IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*, 2018. |
| [10] | N. P. T. B. S. Sanjeev Subba, *Tribhuvan University Journal,* vol. 33, no. 1, pp. 11-22, 2019. |
| [11] | N. E. D. a. I. H. Ulfa, "Measuring Performance of N-Gram and Jaccard-Similarity Metrics in Document Plagiarism Application," *Journal of Physics: Conference Series,* 2019. |
| [12] | D. P. R. R. F. M. Winda Kurnia Sari, "Text Classification Using Long Short Term Memory with Glove Features," *JITEKI,* vol. 5, no. 1, pp. 85-100, 2019. |
| [13] | B. K. Bal, "A novel rule-based recursive stemming algorithm for Nepali Plagiarism Detection". |
| [14] | S. S. a. S. Shakya, "Nepali POS Tagging Using Deep Learning Approaches," *International Journal of Science,* vol. 17, no. 2, pp. 69-84, 2020. |
| [15] | A. N. G. N. P. P. a. P. S. K. Anupama Nair, "Semantic Plagiarism Detection System for English Texts," *IRJET,* vol. 07, no. 05, 2020. |
| [16] | D. D. A. A.-N. S. R. J. K. L. K. J. G.-D. Ö. Ç. a. D. W.-W. T. Foltýnek, "Testing of support tools for plagiarism detection," *International Journal of Educational Technology in Higher Education,* vol. 17, no. 1, 2020. |
| [17] | "Detailed Analysis of Extrinsic Plagiarism Detection System Using Machine Learning Approach". |
| [18] | O. i. c. o. R. C. f. E. i. a. development, Interviewee, *Plagiarism Detection in Nepali Journals.* [Interview]. 27 January 2023. |
| [19] | A. M. S. a. M. M. Almustafa, "ANALYSIS AND CLASSIFICATION OF CUSTOMER REVIEWS IN ARABIC USING MACHINE LEARNING AND DEEP LEARNING," October 2023. [Online]. |
| [20] | B. Rees, "Similarity in graphs: Jaccard versus the Overlap Coefficient," 3 May 2019. [Online]. Available: https://medium.com/rapids-ai/similarity-in-graphs-jaccard-versus-the-overlap-coefficient-610e083b877d. [Accessed January 2023]. |
| [21] | C. J. BURGES, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery,* vol. 2, pp. 121-167, June 1998. |
| [22] | "When you must Cite," [Online]. Available: https://poorvucenter.yale.edu/undergraduates/using-sources/understanding-and-avoiding-plagiarism/warning-when-you-must-cite. [Accessed 2023]. |
| [23] | "Plagiarism Overview," Purdeu University, [Online]. Available: https://owl.purdue.edu/owl/avoiding\_plagiarism/index.html. [Accessed January 2023]. |
| [24] | K. Paudyal, *Nepali Stopwords.* |
| [25] | G. Ansal, "Plagiarism Detection Corpus". |
| [26] | B. K. B. E. Al., "Nepali Spellchecker".*Centre for Research in Urdu Language Proc.* |