NATIONAL COLLEGE OF COMPUTER STUDIES (NCCS)

**Tribhuvan University**

**Institute of Science and Technology**

**A PROJECT REPORT ON**

**English to Nepali Transliterator**

**Submitted to**

***Department of Computer Science and Information Technology***

***TU, IOST***

***In partial fulfillment of the requirement for the Bachelor Degree in Computer***

***Science and Information Technology***

**Submitted by**

August 2015

**National College of Computer Studies**

**Tribhuvan University**

**LETTER OF APPROVAL**

This is to certify that this project prepared by **[..]** entitled “English to Nepali Translitertor” in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology has been well studied. In our opinion it is satisfactory in the scope and quality as a project for the required degree.

|  |  |
| --- | --- |
| **SIGNATURE** | **SIGNATURE** |
| **SIGNATURE** | **SIGNATURE** |

**National College of Computer Studies**

**Tribhuvan University**

**Supervisor’s Recommendation**

I hereby recommend that this project prepared under my supervision by entitled “System” in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology be processed for the evaluation.

**SIGNATURE**

**SUPERVISOR**

**ABSTRACT**

The ***English to Nepali Transliterator*** deals with the use of information technology that helps a teacher to search for a home tuition and a student to search for a tutor. The project is entitled “”. SMT is a web based application supporting these facilities. The application is developed with simple user-interfaces but giving high priority to the application logic. The application logic uses the Artificial Neural Network- Back Propagation Algorithm (ANN BP) to search for perfect student-teacher pair. The system also provide the users with the details of who has viewed their information like email address, contact number, address etc.

***The front end is furnished with HTML, CSS and javascript. The application logic is realized with Python programming language. Abstract maa technology kura narkaahda raamro hola just describe on our project work*** The notification based security of user information, Machine Learning and two-way searching makes this application a useful entity.

**ACKNOWLEDGEMENT**

We would like to express our deepest appreciation to all those who provided us with the possibility to complete this report. We express our profound gratitude and deep regards to our project co-ordinator, Mr. Dilli Prasad Sharma ***(gold medalist)*** ***for this plzz concern with Dilli Sir*** for monitoring us, providing constant encouragement throughout the completion of this project and guiding towards right direction.

We are also thankful to our college administration for providing us with a work space where we could easily work in a group. We thank Mr. Dadhi Ram Ghimire for supporting us in understanding and implementing BP algorithm. We are obliged to our class friends for their full effort in guiding and supporting us in achieving the goal.

Finally, an honorable mention goes to our families for their understandings and supports on us in completing this project. Without the help of the particular that mentioned above, we would not have been able to complete this project.

***Acknowledgement maa hierarchy huncha***

1. ***Supervsior***
2. ***HOD***
3. ***College Teachers***
4. ***Friends***

Contents

[CHAPTER 1: INTRODUCTION 1](#_Toc490153907)

[1.1 Background 1](#_Toc490153908)

[1.2 Problem Definition 2](#_Toc490153909)

[1.3 Objectives 2](#_Toc490153910)

[1.4 Scope of Project 3](#_Toc490153911)

[CHAPTER 2: RESEARCH METHODOLOGY 3](#_Toc490153912)

[2.1 Literature Review 4](#_Toc490153913)

[2.2 Tools Used 5](#_Toc490153914)

[2.2.1 Software Tools 5](#_Toc490153915)

[2.2.2 Hardware Tools 6](#_Toc490153916)

[2.3 Algorithm 6](#_Toc490153917)

[CHAPTER 3: SYSTEM ANALYSIS 7](#_Toc490153918)

[3.1 Requirement Specification 8](#_Toc490153919)

[3.1.1 Functional Requirements 8](#_Toc490153920)

[3.1.2 Non- functional requirements 8](#_Toc490153921)

[3.2 Feasibility Study 9](#_Toc490153922)

[3.2.1 Technical Feasibility 9](#_Toc490153923)

[3.2.2 Economic Feasibility 9](#_Toc490153924)

[3.2.3 Operational Feasibility 10](#_Toc490153925)

[3.2.4 Schedule Feasibility 10](#_Toc490153926)

[CHAPTER 4: SYSTEM DESIGN 10](#_Toc490153927)

[4.1 Overview 11](#_Toc490153928)

[4.2 System Architecture 11](#_Toc490153929)

[4.3 Algorithm 15](#_Toc490153930)

[4.4 Program Flow Diagram 17](#_Toc490153931)

[4.5 Use CASE Diagram 18](#_Toc490153932)

[4.6 Class Diagram 19](#_Toc490153933)

[4.7 Sequence Diagram 20](#_Toc490153934)

[4.8 User Interface Design 21](#_Toc490153935)

[CHAPTER 5: SYSTEM IMPLEMENTATION 22](#_Toc490153936)

[5.1 Background 22](#_Toc490153937)

[5.2 Coding 23](#_Toc490153938)

[5.2.1 Training Codes: 23](#_Toc490153939)

[5.2.2 Deployment Codes: 23](#_Toc490153940)

[5.4 Execution Snapshots 24](#_Toc490153941)

[5.3 Testing Strategies 25](#_Toc490153942)

[5.3.1 Functional Testing 25](#_Toc490153943)

[5.3.2 Module Testing 26](#_Toc490153944)

[5.3.3 System Testing 26](#_Toc490153945)

[5.3.4 System Reliability Testing: 26](#_Toc490153946)

[5.4 Result Discussion and Analysis: 29](#_Toc490153947)

[CHAPTER 6: CONCLUSION AND FUTURE ENHANCEMENT 30](#_Toc490153948)

[6.1 Conclusion 30](#_Toc490153949)

[6.2 Limitation and Future Enhancement 30](#_Toc490153950)

[REFERENCE 31](#_Toc490153951)

[APPENDIX 32](#_Toc490153952)

# CHAPTER 1: INTRODUCTION

## 1.1 Background

Transliteration is the method of translating into another language by expressing the original foreign word using characters of the target language preserving the pronunciation in their source language [1]. They are phonetically similar to the actual Nepali word. This project entitled “English to Nepali Transliteration with Spell Checker” primarily focuses on converting Nepali words written in roman letters into Nepali Unicode and a spell checker, which checks for missed-spelled words with suggestion for correct or provide similar words. It mainly focuses on converting the Romanized Nepali words into Nepali Unicode and then the converted Nepali words will be checked from our dictionary for spelling mistakes and provide suggestions.

Most of those technologies uses English language as their main input language. This has led us to adapt to Romanized Nepali texts for communication in web. Therefore, this is our effort to help native Nepali speaker – like ourselves, who mostly use Roman alphabets, find the meaning of a Nepali word without knowing how it is written in Nepali and its corresponding Nepali Unicode text. They can just type a phonetically similar word in Roman alphabets and get the desired Nepali word with its Unicode conversion. This will help them learn Nepali words and their meanings quite easily.

**Typically, tags are short labels not more than a few words long, provided as metadata to software objects in software information sites. Users can attach tags to various software objects, effectively linking them and creating topic related structure. Tags are therefore useful for providing a soft categorization of the software objects and facilitating search for relevant information. To accommodate new content, most software information sites allow users to create tags freely.**

**However, this freedom comes at a cost, as tags can be idiosyncratic due to users’ personal terminology. As tagging is inherently a distributed and uncoordinated process, often similar objects are tagged differently.**

**In many community-based information web sites, such as StackOverﬂow, users contribute content, which are called software object, in the form of questions and answers, allowing others to learn through the collaboration and contributions of the community. These web sites often rely upon tags. Almost always, users are given the responsibility to choose tags which identify their own content. Furthermore, some software information sites (e.g., STACK OVERFLOW) require users to add tags at the time of posting a question, even if they are unfamiliar with the tags in circulation at that time. Due to differences in personal terminology and tagging purpose, it is often difﬁcult for users to select appropriate tags for their content. Having a tag recommendation system that can suggest tags to a new object (e.g., based on how other similar objects have been tagged in the past) could (i) help users select appropriate tags easily and quickly, and (ii) in time help homogenize the entire collection of tags such that similar objects are linked together by common tags more frequently.**

**A tag recommendation model can improve tag accuracy and effectiveness in a number of ways. For example, new users would not have to worry as much when choosing appropriate tags for their questions, and so the act of asking questions is an easier experience and results in better tags being chosen.**

**Furthermore, recommending tags decreases the possibility of introducing tag synonyms into the tagspace, as is commonly done through human error.**

## 1.2 Problem Definition

With increasing globalization and the rapid growth there should not be language barrier. It is roughly estimated that, a billion people are learning and using English around the world [2], most of which are second learners, which is same for Nepali language. Writing is probably the most difficult and profound among the four skills of language learning, even for native speaker. Most of the native Nepali speaker use Romanized Nepali words to communicate online. Therefore, they are more use to typing in Romanized Nepali than in Nepali Unicode. In current scenario, if they want to search for some specific word in Nepali they must know the exact spelling of that word in Nepali Unicode or in Romanized Nepali to get the meaning of the word. However, there are not many tools for Unicode conversion tools much less with a spell correction and suggestion feature.

**When a user of some information sites tries to post some questions, in the site, he/she may not be familiar with the tags in circulation at that time. Due to differences in personal terminology and tagging purpose, it is often difficult for users to select appropriate tags for their content.**

**Having a tag recommendation system that can suggest tags to a new object (e.g., based on how other similar objects have been tagged in the past) could**

1. **help users select appropriate tags easily and quickly, and**
2. **In time help homogenize the entire collection of tags such that similar objects are linked together by common tags more frequently.**

**By considering above facts an automatic tag recommendation system will be developed that uses the learning from historical software objects and their tags, and recommends appropriate tags for new objects.**

**Since it is field of huge scope and touches some part of AI and data mining, we have chosen this topic to in order to solve the problems like improper selection of key word(tags), improving efficiency of appropriate tagging, content categorization and selection by providing appropriate tags while posting the content in information sites by a user.**

## 1.3 Objectives

The main objectives for this project work are as follows:

• To convert the Romanized Nepali to Nepali Unicode.

• To check for the misspelled words in the search query.

• To provide suggestions for the misspelled words.

## 1.4 Scope of Project

With most people using English as base language, it becomes difficult to search Nepali based news, literature and other documents, even people who can use Nepali layout keyboard have difficulty in typing Nepali language. At most times, people have to use dictionary to find meanings of words that they are not familiar in Nepali because they are usually adapted to English language. With our system, users can write the Nepali words in English, which will then be converted into Nepali language thus, helping the users in searching Nepali content without much difficulty.

**Although there are several websites (such as StackOverﬂow, AskDifferent, AskUbuntu, etc.) which allow their users to tag their contents, there are still no recommendations system that assists users to tag their contents. This Tag Recommender System can easily fill this void to make the overall website system more effective. This application will be user friendly as users can easily interact with it and along with that it is also focused improving tag accuracy and effectiveness.**

**Moreover, the Tag Recommender System can also be very useful to the designers and developers of the website systems. The website system which allows their users to freely post contents, will have lots of data to maintain. It can be tedious to manage that much information manually. When managing such data, the foremost thing that needs to be done is categorizing the data according to their contents. This system can help to partition the contents to their appropriate category by tagging the contents by some keywords or tags that might fall under the category.**

**Hence, this system helps users, to tag their contents; developers, to organize their data; and increase the efficiency of searching the contents.**

# CHAPTER 2: RESEARCH METHODOLOGY

## 2.1 Literature Review

Nepali is technically from the same family as languages like Hindi and Bengali that falls under the category of Indo-Aryan language. Since English and Hindi language are very similar, extensive research materials were gathered from Indian research based papers.

Lots of research and project have been performed on transliteration process as well as spell checkers based on corpus measures, knowledge measures, hybrid measures and various other measures were found. For transliterating English to Hindi language were performed using dirty method (using hash map), phonetic lexicon based system, Machine Translation system, etc[3, 4]. After the conversion process, spell checking is performed to provide suggestions for incorrect words. In order to compare words (strings of words) for spell checking and suggestion, various methods have been employed by researchers to compare strings. These include methods such as Dice’s Coefficient, Matching Coefficient, Overlap Coefficient, Levenshtein distance[7], Needleman-Wunch distance or Sellers Algorithm, Longest Common Subsequence Ratio (LCSR)[4], Soundex distance metric, Jaro-Winkler metric and n-gram metric, phrase based statistical machine translation approach,etc [4].

A phonetic lexicon based system in [3] that performs transliteration process is used to minimize errors as Hindi has a series of characters which are phonetically same. Also, Statistical Machine Translation system in [4], transliterates English-Hindi language pair consisting of Indian names using different character encoding for target language by using phrase based statistical machine translation (SMT) technique. Another research paper published in 2009 uses Machine Transliteration Systems that generates the Hindi transliteration from English where the system learns the mappings of English-to-Hindi automatically from the bilingual NEWS training set being guided by linguistic features/knowledge[1] .

Huang et al. in [5] derives a transliteration model between the Romanized Hindi and the English letters. They apply this model to a parallel corpora and extract Hindi-English named entity pairs based on their similarities in written form.

As for spelling correction, reverse minimum edit distance by Chaudhuri in [6] uses a candidate set of words is produced by first generating every possible single error permutation of the misspelled string and then checking the dictionary if any make up valid word.

Lavenshtein edit distance algorithm in [4] along with other similarity metric algorithm such as Transliteration Similarity Metric, LCSR Metric, Jaro-Winkler Metric and N-gram(3) Metric. In order to compare it with other similarity metrics, the distance of two strings in the following equation to obtain the similarity between the source and the target string. If a word pair received at least two positive votes from these similarity metric algorithms, the pair was considered to be a correct alignment.

Another method proposed in [8] was Romanized text language identification system (RoLI) that uses an n-gram based approach and also exploits sound based similarity of words. In our system we are addressing identification of phonetic as well as spelling differences the variations using combination of Classic Soundex algorithm, and Levenshtein distance[7] was proposed for language identification of Romanized text. The similarity of document profile Di from the language profile Li was obtained using the following formula for similarity robust to multilingual text with an average accuracy of 98.3%, despite the spelling variations as well as sound variations in Indian languages(Tandon, 2010).

The main paper we have focused is “English-Hindi Transliteration using Multiple Similarity Metrics”[4] Based on various research papers and websites, we found that using dirty method for Transliteration engine can be used to convert English characters to corresponding Nepali characters as it is simple and straight forward. As for spell checking, Levenshtein Edit distance was chosen as because of its simplicity and effectiveness. In [4], it was used along with other similarity metrics to improve accuracy so we propose starting from Levenshtein Edit distance and then it can be extended with other similarity metrics.

**One of the most defining feature of tag recommendation system is efficiency which is not surprising that enables us to search for particular content with just few keywords. During research of this project, we have reviewed several websites, articles, journals about Tag Recommendation System Development. We used several such systems in the internet and reviewed about them.**

**After studying many such applications we have concluded that there are many Tag Recommendation Systems developed such as springer, grouplens which are unable to give the proper features of recommendation and communication. Thus, this brings us the concept of developing proper Tag Recommendation System.**

**The development of tag recommendation systems for user created content is a relatively new ﬁeld and most previous work has taken place within the last couple of years. Thus, tag recommendation is a ﬁeld in which the state of the art is still being actively developed, and the most accurate methods for recommending tags have yet to be established.**

**Numerous analyst have carried out enormous tedious groundwork rendering concern about Tag Recommender System. Some illustrious are:**

**Xia et al. propose an automatic tag recommendation algorithm TagCombine [1]. There are three components of TagCombine, each of which tries to assign the best tags to untagged objects:**

**(1) multi-label ranking component, which predicts tags using a multi-label learning algorithm,**

**(2) similarity based ranking component, which uses similar objects to recommend tags, and**

**(3) tag-term based ranking component, which analyzes the historical afﬁnity of tags to certain words in order to suggest tags.**

**Wang et al. propose a tag recommendation system dubbed EnTagRec. The proposed EnTagRec computes tag probability scores using two separate methods, Bayesian Inference and Frequentist Inference, and then takes a weighted sum of the probability scores [2].**

**They can be used to search, describe, identify, and bookmark various software objects. For software development, tags also helps to bridge the gap between social and technical aspects [3], [4].**

**In multi-label rankingcomponent, we consider the tag recommendation problem as a multi-label learning problem [5], where each tag maps to a label.**

**Averaging over StackOverﬂow and Freecode results, we improve TagRec proposed by Al-Kofahi et al. [6] by 22.65% and 14.95%, and the tag recommendation method proposed by Zangerle et al. [7] by 18.5% and 7.35% for recall@5 and recall@10 scores.**

**There is a huge diversity of algorithms and approaches that help creating personalized recommendations. Two of them became very popular: collaborative ﬁltering and content-based ﬁltering. They are used as a base of most modern recommender systems [8].**

**ENTAGREC borrows from two opposite yet complementary lines of thought in the statistics community, Bayesian and frequentist , each with its own advantages [9].**

**BIC is built on top of a state-of-the-art Bayesian inference technique (Labeled LDA [10]), creating a uniﬁed mixture model over the distribution of all tags to measure the likelihood of a tag to be assigned to a software object.**

## 2.2 Tools Used

### 2.2.1 Software Tools

1. Python
2. Django Framework
3. Web Browser (Chrome)
4. Microsoft Visio
5. Windows 7 and above

### 2.2.2 Hardware Tools

1. Windows7/8 running Computer

## 2.3 Algorithm

**The algorithm that we used on this Tag Recommender System is inherited EnTagRec Algorithm.**

**Tag Recommender algorithm :-**

**Tag Recommender contains two processing components:**

* **Preprocessing Component (PC),**
* **Engine**

1. **Input software objects are processed by PC to generate a common representation. Here the user’s input (specifically: question) is processed by the preprocessor to identify the specific tags (which are the nouns).These textual documents are then input to the main processing engines, namely BIC.**
2. **BIC models a software objects as a probability distribution of tags, and a tag as a probability distribution of words that appear in the software objects that are tagged by it.**

**The TAG RECOMMENDER works in two phases, a training phase and a deployment phase. In the training phase, TAG RECOMMENDER trains several of its components using training software objects and corresponding tags. In the deployment phase, the trained TAG RECOMMENDER is used to recommend tags for untagged software objects.**

**In the training phase, BIC is trained based on the training data. BIC takes the bag of words representation of the software objects and their corresponding tags to train itself. The result is a statistical model which takes in a bag of words representing a software object and produces a ranked**

**list of tags along with their probabilities of being related to the input software object.**

**After TAG RECOMMENDER is trained, it is used in the deployment phase to recommend tags for untagged objects. For each such object, we ﬁrst use PC to convert it to bags of words. Next, we feed this bag of words to the trained BIC. Each of them will produce a list of tags with their likelihood scores. The top few tags with the highest likelihood scores will be output as the predicted tags of the input untagged software object.**

# CHAPTER 3: SYSTEM ANALYSIS

## 3.1 Requirement Specification

The requirements are the major part in the system development. Once the requirements are collected they determine the structure, functionalities and operational constraints of the system. The requirements are hard to determine due to their dynamic and dependent nature. During system development the requirements may change by the system user. One requirement may depend on another requirement thus making changes to lower requirement leads to change of upper requirements and vice-versa.

The requirements of the proposed system ‘English to Nepali Transliterator has been determined by ourselves. We haven’t taken any interview with the users since it is a generic software product. This section contains all of the functional and non-functional requirements of the system. It gives a detailed description of the system and all its features.

### 3.1.1 Functional Requirements

The functional requirements specify the services that the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do.

This section includes the requirements that specify all the fundamental actions of the tag recommender system.

* **User Input**

Get the characters as input through textfield/textArea.

#### Processing of the input characters

##### Uses AJAX and javascript to asynchronously transfer characters

##### Each characters from input are processed and their combinations of nepali characters are obtained

##### Each obtained words of nepali characters are searched in the dictionary and matched

#### Compare each nepali words to dictionary words

#### Display matched nepali words as suggestions

### 3.1.2 Non- functional requirements

The requirements in this section provide a detailed specification of the user interaction with the system and measurements placed on the system performance.

#### Response time:

It is relative with the numbers of input character before white spaces. Output is generated within 1 or 2 seconds.

#### Hard drive space:

Relative to dictionary size and minimum space required to run is about 2mb.

#### Reliability:

Words matched in dictionary provides reliable transliteration to users.

#### Availability:

The availability of the system will be 100% except in hardware failure (not considering network).

* Maintainability:

##### System extendibility:

The system will be extensible .New features could be easily added in future reference.

* + System testability:

Test environments will be built for the system in order to test the system.

#### Portability:

System will be platform independent running on any browser supported operating systems.

## 3.2 Feasibility Study

### 3.2.1 Technical Feasibility

It is technically feasible since any browser that supports JavaScript can use our system. Most of the people have smartphones and computers with internet connectivity nowadays and thus this system can be easily used. Technically many users are aware of basic usage of computing devices as well as internet and they can freely use this system.

### 3.2.2 Economic Feasibility

**“English to Nepali Transliteration with Spell Checker”** application is an open source, platform independent application and freely accessible from internet so that project is economically feasible.

### 3.2.3 Operational Feasibility

A users with internet access can easily access the webpage with any browsers to access the system. It has user friendly interface to display output along with suggestions as soon as users types the characters. So, it is very easy to use by normal users so it is operationally feasible.

### 3.2.4 Schedule Feasibility

By considering complexity of requirements of system and availability of resources for system development the project is aimed to complete within specific schedule.

Figure 3.2.4: Gantt chart of project time interval

# CHAPTER 4: SYSTEM DESIGN

## 4.1 Overview

When users type the English characters on the textfield provided in the webpage, the system converts them to its respective characters. The system also provides suggestions of the nepali words if they are found in the dictionary.

The system transforms English to Nepali words by taking the software objects from users as input, processing them and transforming English words to Nepali as output with suggestions. The system processes the input by firstly performing the preprocessing action. During preprocessing, the system maps each English characters with its respective Nepali characters and create a combination of Nepali words. For example, English character “a” has multiple phonetic mapping as ("अ", "आ", "ए", "ा"),"b" has ("ब्", "ब", "भ्", "भ") and so on. Similarly, some characters such as "aa"”has mapping ("आ", "ा"). The preprocesser makes all the possible combinations to create nepali characters.

After the combinations of Nepali words are created from English words, the system checks each Nepali words with the dictionary words. Then, the matched words are displayed as suggestions in user output. When, user presses white spaces the input characters are converted to its respective Nepali words.

## 4.2 System Architecture

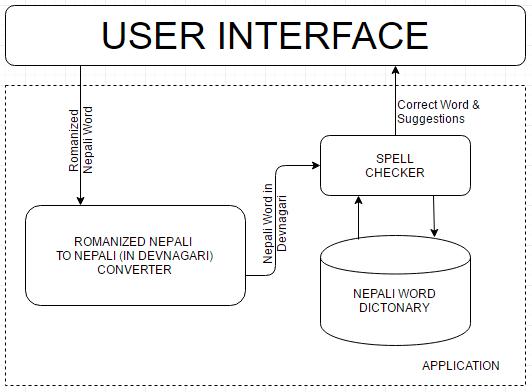
****

Fig. 4.2 Fig 1: System Architecture of English to Nepali Transliteration with

Spell Checker

The block diagram of the system architecture for the English to Nepali Transliterator is shown in fig 4.().The system gets input, software object, from the user, processes it using several of its components and finally converts it to Nepali words along with suggestions. The system consists of mainly five components:

1. Interface
2. Pre-Processor
3. File
4. Engine

1. Interface

The interface for the system is basically a web page. The interface for the system is shown in fig. 4.**()**. The user feeds the system with English characters and for each consecutive white spaces, the system provide output. While user is typing, each characters is asynchronously send to the server so if the characters match with the dictionary they are displayed as suggestions. However, if the words are not found in the dictionary, user have to explicity choose Nepali words from a list of suggestions that is created by the preprocessor and not from the dictionary.

1. Pre-Processor

The Pre-Processor performs various actions on the set of words obtained from the software object. The major actions performed are:

* Mapping of English characters with its respective Nepali characters that sound phonetically similar.
* Each English characters may have multiple mappings so for ‘n’ English characters, the preprocessor makes combination of Nepali characters by taking all the possible combinations of each English characters having multiple mappings.
* It also checks if the preceding characters may be present in the user input. For example: user types “aa” and the system has mapping to “a” as ("अ", "आ", "ए", "ा") and "aa" as ("आ", "ा"). Thus, the system takes the mapping of "aa“ ignoring the mapping of "a”.

Hence, by following this approach, the output of the pre-processor is a set of Nepali words that are used for checking the dictionary words.

1. File

A file is a resource for storing information, which is available to the computer program and is usually based on some kind of durable storage. All the valid Nepali words are stored in the file with “.txt” format. After the preprocessor has a output of set of Nepali words, each words are checked in the file.

1. Engine

(..)

## 4.3 Algorithm

**Algorithm 1 : Pre-Processor Component Algorithm**

1. Pre-Process the Software Object
2. Input: *swObj* (Software Object or characters)
3. Output: *wordlist* (Set of distinct words)
4. Method:
5. For each characters, if the preceding English characters in the dictionary list are not found
   1. Convert characters to Nepali words
   2. Combine characters with each mapping of Nepali words(English characters may have multiple mapping to Nepali characters)
   3. Use regular expression to filter some unnecessary characters
6. Send set of combinations to the Engine for comparing with words in the file.
7. Send set of combinations back to web interface

**Algorithm 2 : Engine Component Algorithm**

1. Recommend the words for the given *wordList* (of the *swObj*).
2. Input: *wordList* (Pre-Processed set of words)
3. Output: Dictionary words
4. Method:
5. Search the file for existence of words in *wordList*
6. IF the search is successful
   1. Retrieve words as suggestion
   2. When user types the white spaces, the first suggestion is automatically used to convert English to Nepali word.

## 4.4 Program Flow Diagram

The flow of the program is depicted by the flow chart provided below. This project mainly consist of 4 parts first one is user input fetch, second part consist of identifying common representation in the provided input using preprocessor, third part consists of using BIC to find probabilities of various tags given a bag of words representing a software object using Bayesian inference and last part consist of recommending the highest ranked tag.

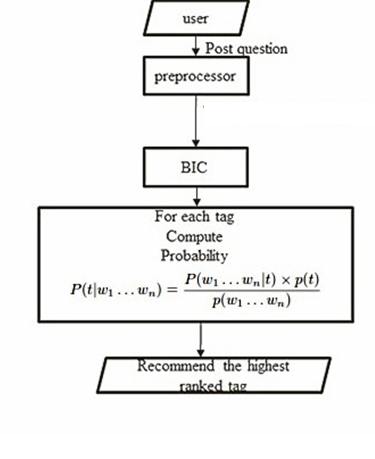


Fig 4.4 Program Flow Diagram

## 4.5 Use CASE Diagram

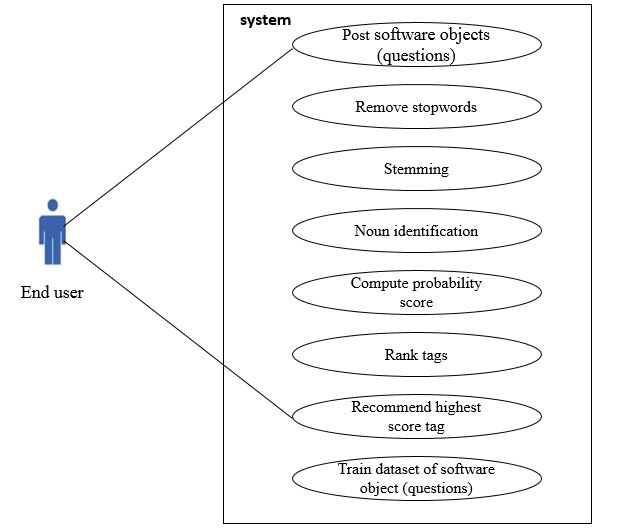


Fig. 4.5 use case diagram for Tag Recommender System

## 4.6 Class Diagram

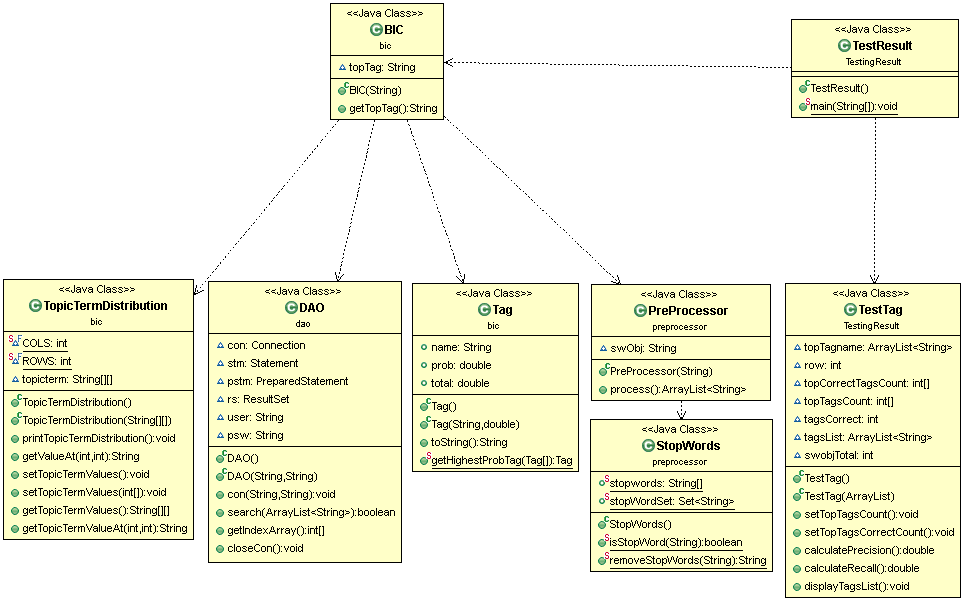
****

Fig. 4.6 Class Diagram for Tag Recommender System

## 4.7 Sequence Diagram

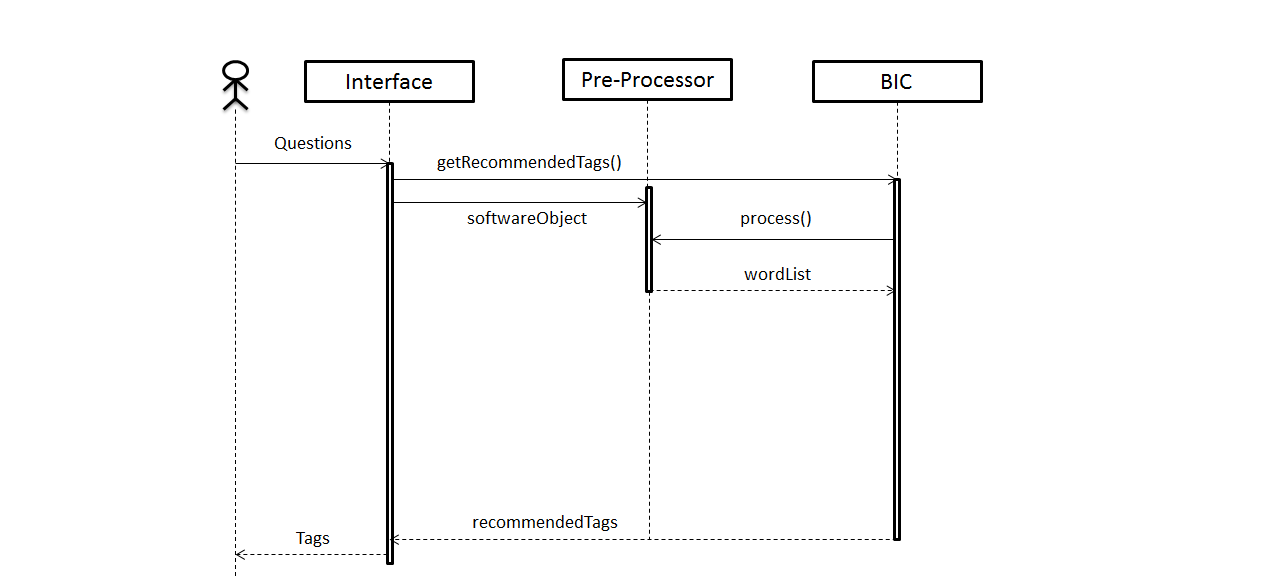


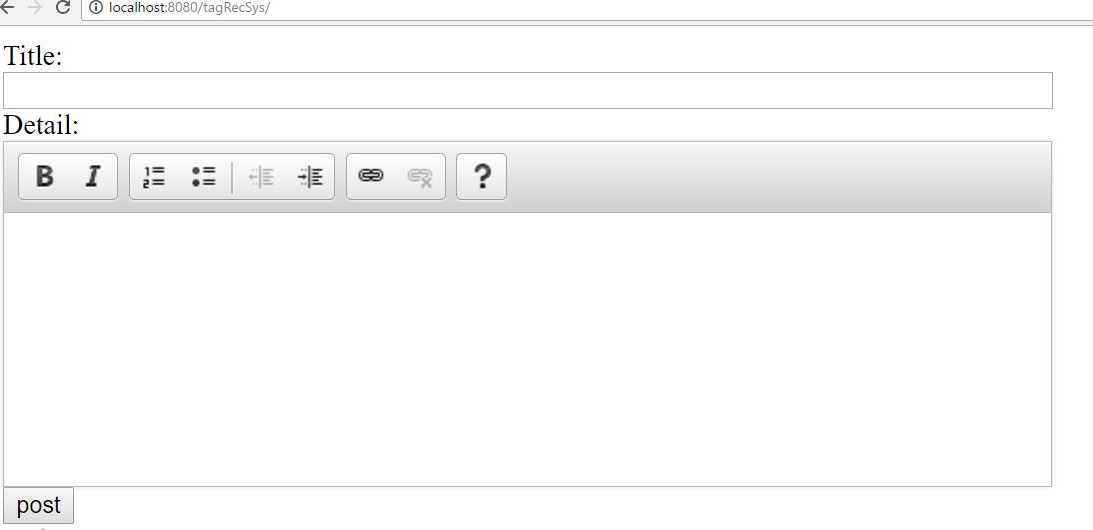
Fig 4.7 Sequence Diagram for Tag Recommender System

The given sequence diagram demonstrates diagrammatically how the system works. A user posts a question on the interface. Upon this event, the interface invokes getRecommnededTags() function of the BIC or engine component. The interface also passes the software object to the Pre-Processor component. When the BIC invokes process() function of the Pre-Processor Component, the Pre-Processor removes stopwords, performs stemming, POS Tagging and return the “wordlist” which the set of tokens or nouns. The BIC then performs further calculations as discussed previously, and return recommended tags to the interface which displays the tags to the user.

## 4.8 User Interface Design

The initial interface design for this project was to show the text area where the user inputs questions which is show in figures below.



* 
* Fig:4.8.1 User Interface

# CHAPTER 5: SYSTEM IMPLEMENTATION

## 5.1 Background

The system uses Python, HTML, JavaScript, Django Framework to provide a web based system. The backend server uses Python programming language and the web interface uses AJAX calls to asynchronously send characters typed by the user to the server.

**In training phase,10,000 training questions were used of each tag Java, C#,PHP,iOS, HTML.**

The system is implemented using Java programming language. Since it is web based so it is platform independent. The interface of the system is built using DHTML and JavaScript.

## Coding

### Training Codes:

Codes: LLDAlearn.scala(APPENDIX, page 44)

### 5.2.2 Deployment Codes:

**Using Preprocessor to identify common representations**

Codes:Preprocessor.java(APPENDIX, page 29)

**Identifying Term tag values**

Codes:DAO.java ,Tag.java(APPENDIX, page 33, 35)

**Identifying Stop Words in the user input**

Codes:StopWord.java (APPENDIX, page 30 )

**Using BIC to calculate probability and recommend tags**

Codes:BIC.java(APPENDIX,page36)

## 5.4 Execution Snapshots

When the program is run, user enter their question in title field and its description in detail field. The input screen shot shown below.

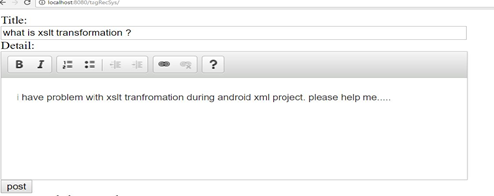


Figure 5.4.1: Input interface for user

After the user enters the input and clicks enter. PC(preprocessor) processes the input and identifies the common representation in it and then BIC executes to find conditional probability and finally recommend tag. This can be illustrated in figure below :

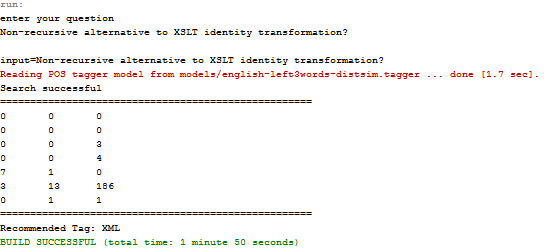


fig 5.4.2:execution inside server

This input is processed in the server where PC extracts the nouns, BIC computes the probabitity score

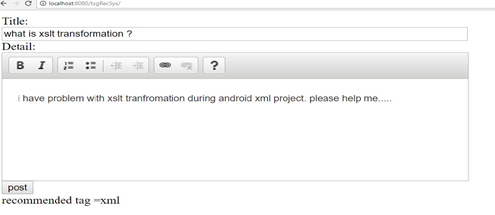


Fig:5.4.3 final output

And finally the tag is recommended by the system which is ‘xml’ for the question ‘Non-recursive alternative to XSLT identity transformation?’

## 5.3 Testing Strategies

The testing phase is done to verify and validate the Tag Recommender System. This application is tested to make sure that the developed system is free from programming and logical errors, and what we were expecting. This application is tested in each changes we made and checked whether all the system and user requirements are met or not.

### 5.3.1 Functional Testing

The system is treated as a black box whose behavior can only be determined by studying its inputs and the related outputs. We are only concerned with the functionality and not the implementation of the software. In this method we performed the testing of test cases by simply providing questions as inputs to the system then analyze the tags recommended by the system.. The outputs determine whether the test cases are met or not. Tag Recommender System has successfully passed the functional testing.

### 5.3.2 Module Testing

This application is built in Java language.JAVA language is object oriented language. Thus, we have defined classes for performing the application logic. In module testing the objects for each class are created and then the methods of the classes are tested with the test data. The testing is done to detect errors in the class specifications and operations. This application has successfully passed the module testing.

### 5.3.3 System Testing

System testing is focused on assessing the system’s reliability. It helps to determine the optimality of the internal structure and the outputs generated by the system meets the system requirements. Faults that are discovered during system testing are passed back to the development phase for repair. Then the faults are recovered and then the system is tested again as a whole. This process helps in validating the system by testing the system as a whole that covers each module of the application, database specifications and underlying configurations. This application passes the system testing and is ready for real world implementation.

### 5.3.4 System Reliability Testing:

The **recall** and **precision** are used for the evaluation purpose of Tag Recommender’s reliability.

Recall=

Precision=

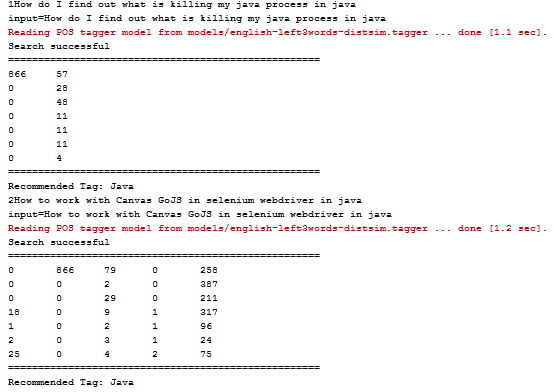
Where *n= total no of tags*

*topTagsCorrectCount[i]=total no of times tag[i] is recommended by valid testing s/w objects*

*topTagsCount[i]= total no of times tag[i] is recommended in whole testing set*

*correctTagsCount[i]= total no of valid s/w objects taken for tag[i] in testing set*

Around 140 random questions were tested for calculation of recall and precision.



.

.

.

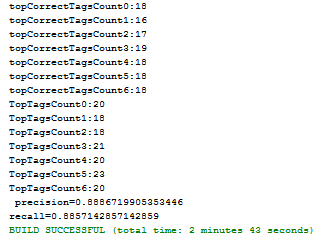


Fig: Snapshots representing the System’s testing process

Output:

Precision=0.8886719905353446

Recall= 0.8857142857142859

## 5.4 Result Discussion and Analysis:

The above result shows that, the Tag Recommender is reliable in terms of its output that it generates. Hence it is an elite solution for the tagging automation. The recommendation is very efficient and fast.

Tag Recommender System is very easy to maintain. The maintenance cost is very low and doesn’t require large storage. The information stored in database are light weight due to this data is easy to access and store. The database is secured so that user won’t be able to alter the information stored. Other maintenance includes interfaces changes as user requirements and add new information to database.

This project was built to provide appropriate answer (top rated) to question along with tags, but we were only able to recommend tags for the questions. After researching different research papers, documentation, and reports from different universities in internet of making a tag recommending system such as StackOverflow, the implementation of probability finding algorithms was our biggest challenge. It requires algorithm for identifying common representation which is done using Preprocessor Component and finding conditional probability of tags and then recommending the tags using BIC. Further this project was developed to recommend user with suitable answers along with tags.

But due to limitation of time and knowledge, we are able to recommend tags only. This project is possible to extend more if more time and resource is made available. So, this project can be developed further for economic, social, scientific purpose.

# CHAPTER 6: CONCLUSION AND FUTURE ENHANCEMENT

## 6.1 Conclusion

The main intention of the project is to develop a system such that user doesn’t have to face the problems regarding identification of the content they want to know. This project provides the way of finding appropriate tags for the user’s input.

Hence, this project shows the usefulness of the information related to content identification among several many contents over the internet.

## 6.2 Limitation and Future Enhancement

The limitations of this system are:

1. Ambiguity
2. The algorithms fails to consider the diversity of different users' interests as the top ranked tags are presented to the user as suggestions.
3. No multi label recommendations.
4. No learning.
5. May not cover whole programming terms

This application can be extended to further uses such as for recommendation of answers with respect to the recommended tags. This system recommends keywords as tags for the given questions.

For example: for question -‘Non-recursive alternative to XSLT identity transformation?’ the tag ‘xml’ is recommended.

With further enhancement, this system could be able to recommend the answer in brief rather than tags only.

Similarly,the learning can be done,to improve the reliability of tagging .In the same way the mulitlabel recommendations can added so that user can get close reference of tags.

# REFERENCE

1. X. Xia, D. Lo, X. Wang, B. Zhou. Tag Recommendation in Software Information Sites. MSR, 2013.
2. S. Wang, D. Lo, B. Vasilescu, A. Serebrenik. EnTagRec: An Enhanced Tag Recommendation System for Software Information Sites. ICSME.
3. C. Treude and M. Storey, “How tagging helps bridge the gap between social and technical aspects in software development,” in Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on. IEEE, 2009, pp. 12–22.
4. C. Treude and M.-A. Storey, “Work item tagging: Communicating concerns in collaborative software development,” Software Engineering, IEEE Transactions on, vol. 38, no. 1, pp. 19–34, 2012.
5. G. Tsoumakas and I. Katakis, “Multi-label classiﬁcation: An overview,” International Journal of Data Warehousing and Mining (IJDWM), vol. 3, no. 3, pp. 1–13, 2007.
6. J. Al-Kofahi, A. Tamrawi, T. Nguyen, H. Nguyen, and T. Nguyen, “Fuzzy set approach for automatic tagging in evolving software,” in Software Maintenance (ICSM), 2010 IEEE International Conference on. IEEE, 2010, pp. 1–10.
7. E. Zangerle, W. Gassler, and G. Specht, “Using tag recommendations to homogenize folksonomies in microblogging environments,” Social Informatics, pp. 113–126, 2011.
8. P. Resnick and H. R. Varian, “Recommender systems,” Commun. ACM, vol. 40, pp. 56–58, March 1997.
9. B. Efron. Why isn’t everyone a Bayesian? The American Statistician,40(1):1–5, Feb86
10. D. Ramage, D. Hall, R. Nallapati, and C. D. Manning. Labeled lda: a supervised topic model for credit attribution in multi-labeled corpora. In EMNLP ’09, pages 248–256, 2009.
11. M. F. Porter. An algorithm for sufﬁx stripping. In Readings in information retrieval, pages 313–316. Morgan Kaufmann, 1997.
12. G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. IEEE Trans. Softw. Eng., 28(10):970–983, Oct. 2002.

# APPENDIX

Preprocessor.java

package preprocessor;

import edu.stanford.nlp.tagger.maxent.MaxentTagger;

import java.util.ArrayList;

public class PreProcessor {

// declare software object

String swObj;

public PreProcessor(String swObj) {

this.swObj = swObj;

}

public ArrayList<String> process() {

// remove stopwords

swObj = StopWords.removeStopWords(swObj);

// Initialize the tagger

MaxentTagger tagger = new MaxentTagger("models/english-left3words-distsim.tagger");

// tag the swObj

String tagged = tagger.tagString(swObj);

ArrayList<String> wordList = new ArrayList<>();

String str[] = tagged.split(" ");

for (String x : str) {

if (x.substring(x.lastIndexOf("\_") + 1).startsWith("N")) {

wordList.add(x.split("\_")[0]);

}

}

return wordList;

}

}

StopWords.java

package preprocessor;

import java.util.HashSet;

import java.util.Set;

import java.util.Arrays;

import java.util.regex.Pattern;

public class StopWords {

// as used by the algorithm from http://www.textfixer.com/resources/common-english-words.txt

// public static String[] stopwords = {"a","able","about","across","after","all","almost","also","am","among","an","and","any","are","as","at","be","because","been","but","by","can","cannot","could","dear","did","do","does","either","else","ever","every","for","from","get","got","had","has","have","he","her","hers","him","his","how","however","i","if","in","into","is","it","its","just","least","let","like","likely","may","me","might","most","must","my","neither","no","nor","not","of","off","often","on","only","or","other","our","own","rather","said","say","says","she","should","since","so","some","than","that","the","their","them","then","there","these","they","this","tis","to","too","twas","us","wants","was","we","were","what","when","where","which","while","who","whom","why","will","with","would","yet","you","your"};

public static String[] stopwords = {"gonna","won\'t","don\'t","i\'m","a", "as", "able", "about", "above", "according", "accordingly", "across", "actually", "after", "afterwards", "again", "against", "aint", "all", "allow", "allows", "almost", "alone", "along", "already", "also", "although", "always", "am", "among", "amongst", "an", "and", "another", "any", "anybody", "anyhow", "anyone", "anything", "anyway", "anyways", "anywhere", "apart", "appear", "appreciate", "appropriate", "are", "arent", "around", "as", "aside", "ask", "asking", "associated", "at", "available", "away", "awfully", "be", "became", "because", "become", "becomes", "becoming", "been", "before", "beforehand", "behind", "being", "believe", "below", "beside", "besides", "best", "better", "between", "beyond", "both", "brief", "but", "by", "cmon", "cs", "came", "can", "cant", "cannot", "cant", "cause", "causes", "certain", "certainly", "changes", "clearly", "co", "com", "come", "comes", "concerning", "consequently", "consider", "considering", "contain", "containing", "contains", "corresponding", "could", "couldnt", "course", "currently", "definitely", "described", "despite", "did", "didnt", "different", "do", "does", "doesnt", "doing", "dont", "done", "down", "downwards", "during", "each", "edu", "eg", "eight", "either", "else", "elsewhere", "enough", "entirely", "especially", "et", "etc", "even", "ever", "every", "everybody", "everyone", "everything", "everywhere", "ex", "exactly", "example", "except", "far", "few", "ff", "fifth", "first", "five", "followed", "following", "follows", "for", "former", "formerly", "forth", "four", "from", "further", "furthermore", "get", "gets", "getting", "given", "gives", "go", "goes", "going", "gone", "got", "gotten", "greetings", "had", "hadnt", "happens", "hardly", "has", "hasnt", "have", "havent", "having", "he", "hes", "hello", "help", "hence", "her", "here", "heres", "hereafter", "hereby", "herein", "hereupon", "hers", "herself", "hi", "him", "himself", "his", "hither", "hopefully", "how", "howbeit", "however", "i", "id", "ill", "im", "ive", "ie", "if", "ignored", "immediate", "in", "inasmuch", "inc", "indeed", "indicate", "indicated", "indicates", "inner", "insofar", "instead", "into", "inward", "is", "isnt", "it", "itd", "itll", "its", "its", "itself", "just", "keep", "keeps", "kept", "know", "knows", "known", "last", "lately", "later", "latter", "latterly", "least", "less", "lest", "let", "lets", "like", "liked", "likely", "little", "look", "looking", "looks", "ltd", "mainly", "many", "may", "maybe", "me", "mean", "meanwhile", "merely", "might", "more", "moreover", "most", "mostly", "much", "must", "my", "myself", "name", "namely", "nd", "near", "nearly", "necessary", "need", "needs", "neither", "never", "nevertheless", "new", "next", "nine", "no", "nobody", "non", "none", "noone", "nor", "normally", "not", "nothing", "novel", "now", "nowhere", "obviously", "of", "off", "often", "oh", "ok", "okay", "old", "on", "once", "one", "ones", "only", "onto", "or", "other", "others", "otherwise", "ought", "our", "ours", "ourselves", "out", "outside", "over", "overall", "own", "particular", "particularly", "per", "perhaps", "placed", "please", "plus", "possible", "presumably", "probably", "provides", "que", "quite", "qv", "rather", "rd", "re", "really", "reasonably", "regarding", "regardless", "regards", "relatively", "respectively", "right", "said", "same", "saw", "say", "saying", "says", "second", "secondly", "see", "seeing", "seem", "seemed", "seeming", "seems", "seen", "self", "selves", "sensible", "sent", "serious", "seriously", "seven", "several", "shall", "she", "should", "shouldnt", "since", "six", "so", "some", "somebody", "somehow", "someone", "something", "sometime", "sometimes", "somewhat", "somewhere", "soon", "sorry", "specified", "specify", "specifying", "still", "sub", "such", "sup", "sure", "ts", "take", "taken", "tell", "tends", "th", "than", "thank", "thanks", "thanx", "that", "thats", "thats", "the", "their", "theirs", "them", "themselves", "then", "thence", "there", "theres", "thereafter", "thereby", "therefore", "therein", "theres", "thereupon", "these", "they", "theyd", "theyll", "theyre", "theyve", "think", "third", "this", "thorough", "thoroughly", "those", "though", "three", "through", "throughout", "thru", "thus", "to", "together", "too", "took", "toward", "towards", "tried", "tries", "truly", "try", "trying", "twice", "two", "un", "under", "unfortunately", "unless", "unlikely", "until", "unto", "up", "upon", "us", "use", "used", "useful", "uses", "using", "usually", "value", "various", "very", "via", "viz", "vs", "want", "wants", "was", "wasnt", "way", "we", "wed", "well", "were", "weve", "welcome", "well", "went", "were", "werent", "what", "whats", "whatever", "when", "whence", "whenever", "where", "wheres", "whereafter", "whereas", "whereby", "wherein", "whereupon", "wherever", "whether", "which", "while", "whither", "who", "whos", "whoever", "whole", "whom", "whose", "why", "will", "willing", "wish", "with", "within", "without", "wont", "wonder", "would", "would", "wouldnt", "yes", "yet", "you", "youd", "youll", "youre", "youve", "your", "yours", "yourself", "yourselves", "zero"};

public static Set<String> stopWordSet = new HashSet<>(Arrays.asList(stopwords));

public static boolean isStopWord(String word) {

if (word.length() < 2) {

return true;

}

if (word.charAt(0) >= '0' && word.charAt(0) <= '9') {

return true; //remove numbers, "25th", etc

}

if (stopWordSet.contains(word)) {

return true;

} else {

return false;

}

}

public static String removeStopWords(String string) {

String result = "";

Pattern ptrn = Pattern.compile("[ .!]");

String words[] = ptrn.split(string);

// String[] words = string.split("\\s+");

for (String word : words) {

if (word.isEmpty()) {

continue;

}

if (isStopWord(word)) {

continue; //remove stopwords

}

result += (word + " ");

}

return result;

}

}

DAO.java

package dao;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.ArrayList;

public class DAO {

Connection con = null;

Statement stm = null;

PreparedStatement pstm = null;

ResultSet rs = null;

String user = null;

String psw = null;

public DAO() {

this.user = "root";

this.psw = "";

}

public DAO(String user, String psw) {

this.user = user;

this.psw = psw;

}

public void con(String hostName, String dbName) throws ClassNotFoundException, SQLException {

Class.forName("com.mysql.jdbc.Driver");

con = DriverManager.getConnection("jdbc:mysql://" + hostName + "/" + dbName, user, psw);

stm = con.createStatement();

}

public boolean search(ArrayList<String> wordList) throws SQLException {

String querystmt = "";

// gernerating dynamic query

for (int i = 0; i < wordList.size() - 1; i++) {

querystmt = querystmt.concat("token LIKE '" + wordList.get(i) + "' OR ");

}

querystmt = querystmt.concat("token LIKE '" + wordList.get(wordList.size() - 1) + "'");

String query = "select token\_id from tokens where " + querystmt + "ORDER BY token";

rs = stm.executeQuery(query);

return rs.next();

}

public int[] getIndexArray() throws SQLException {

// to get the no. of items in resultset

rs.last();

int index[] = new int[rs.getRow()];

rs.beforeFirst();

int i = 0;

while (rs.next()) {

index[i++] = Integer.parseInt(rs.getString("token\_id"));

}

return index;

}

public void closeCon() throws SQLException {

rs.close();

stm.close();

con.close();

}

}

Tag.java

package bic;

class Tag implements Cloneable {

public String name;

public double prob;

public double total;

public Tag() {

name = "";

prob = 0.0;

total = 0.0;

}

public Tag(String name, double total) {

this.name = name;

this.total = total;

}

@Override

public String toString() {

return "(" + name + ", " + prob + ")";

}

public static Tag getHighestProbTag(Tag[] tags) {

for (int i = 0; i < 7; i++) {

for (int j = i; j < 7; j++) {

if (tags[i].prob < tags[j].prob) {

try {

Tag tempTag = (Tag) tags[i].clone();

tags[i] = (Tag) tags[j].clone();

tags[j] = (Tag) tempTag.clone();

} catch (CloneNotSupportedException ex) {

System.out.println("Error: " + ex);

}

}

}

}

return tags[0];

}

}

BIC.java

package bic;

import dao.DAO;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import preprocessor.PreProcessor;

import java.sql.SQLException;

import java.util.ArrayList;

import javax.swing.JOptionPane;

public class BIC {

public static void main(String[] args) throws IOException {

String swObj;

ArrayList<String> wordList = new ArrayList<>();

int index[] = null; //an array to store the index of words found in database

System.out.println("enter your question");

/// swObj = JOptionPane.showInputDialog(null, "Enter your question");

BufferedReader bfr= new BufferedReader(new InputStreamReader(System.in));

swObj=bfr.readLine().toString();

System.out.println("input="+swObj);

// perform pre-processing on the given software object and store in wordList

wordList = new PreProcessor(swObj).process();

// create Database Access Object

DAO dao = new DAO();

try {

// connect to "entagrec" database connection as "localhost"

dao.con("localhost", "entagrec");

// search database for the existence of words

if (dao.search(wordList)) {

System.out.println("Search successful");

// index[] stores the index of words found in database

index = dao.getIndexArray();

}else{

System.out.println("No Recommendations");

System.exit(0);

}

} catch (SQLException ex) {

System.out.println("Error: " + ex);

} catch (ClassNotFoundException ex) {

System.out.println("Error: " + ex);

} finally {

try {

dao.closeCon();

} catch (SQLException ex) {

System.out.println("Error: " + ex);

}

}

System.out.println("====================================================");

TopicTermDistribution ttd = new TopicTermDistribution();

ttd.setTopicTermValues(index);

ttd.printTopicTermDistribution();

// compute P(tags|wordList) using Bayesian Rule

String topictermvalues[][] = ttd.getTopicTermValues();

Tag tags[] = new Tag[7];

tags[0] = new Tag("Java", 37146.0);

tags[1] = new Tag("PHP", 30754.0);

tags[2] = new Tag("C#", 35429.0);

tags[3] = new Tag("HTML", 40596.0);

tags[4] = new Tag("IOS", 43594.0);

tags[5] = new Tag("XML", 18902.0);

tags[6] = new Tag("JavaScript", 21224.0);

for (int i = 0; i < 7; i++) {

for (int j = 0; j < index.length; j++) {

tags[i].prob += (Double.parseDouble(topictermvalues[i][j])) / tags[i].total;

}

tags[i].prob /= 7.0;

}

System.out.println("====================================================");

System.out.println("Recommended Tag: " + Tag.getHighestProbTag(tags).name);

}

}

TestResult.java

package TestingResult;

import bic.BIC;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.util.ArrayList;

import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import org.apache.poi.ss.usermodel.CreationHelper;

import org.apache.poi.ss.usermodel.Row;

import org.apache.poi.ss.usermodel.Sheet;

public class TestResult {

public static void main(String[] args) throws FileNotFoundException, IOException {

FileInputStream file = new FileInputStream(new File("test.xls"));

HSSFWorkbook wb = new HSSFWorkbook(file);

CreationHelper createHelper = wb.getCreationHelper();

Sheet sheet = wb.getSheet("sheet1");

ArrayList<String> topTagList=new ArrayList<>(); // for storing recommended tags for each testing s/w obj

for(int i=0;i<140;i++)

{

Row row = sheet.getRow(i);

String swObj=row.getCell(3).toString();

System.out.println(i+swObj);

//reading every testing sw object and generating corrensponding tag

BIC test= new BIC(swObj);

topTagList.add(test.getTopTag()); //getting recommended tag for each testing swobj

}

System.out.println("taglist size="+topTagList.size());

TestTag test=new TestTag(topTagList);

test.displayTagsList();

test.setTopTagsCorrectCount();

test.setTopTagsCount();

double precision=test.calculatePrecision();

double recall=test.calculateRecall();

System.out.println(" precision="+precision);

System.out.println("recall="+recall);

}

TestTag.java

package TestingResult;

import java.util.ArrayList;

public class TestTag {

ArrayList<String> topTagname=new ArrayList<>(); //storing recomended top tags

int row=7;

int [] topCorrectTagsCount=new int [row]; //stores |tagsicorrect|

int []topTagsCount=new int[row]; //stores |tagsitopK|

int tagsCorrect=20; //choosen 2000 test s/w object for each tag so tags correct is 2000; stores |tagsi correct| for each tag

ArrayList<String> tagsList=new ArrayList<>();//tags list

int swobjTotal=140;

public TestTag()

{

}

public TestTag(ArrayList str)

{

topTagname=str;

tagsList.add("Java");

tagsList.add("PHP");

tagsList.add("C#");

tagsList.add("HTML");

tagsList.add("IOS");

tagsList.add("XML");

tagsList.add("JavaScript");

}

public void setTopTagsCount()

{

for(int i=0;i<row;i++)

{

for(int j=0;j<swobjTotal;j++)

{

if(tagsList.get(i).equals(topTagname.get(j))){

topTagsCount[i]++; //counting TopTags for each tags

}

}

System.out.println("TopTagsCount"+i+":"+topTagsCount[i]);

}

}

public void setTopTagsCorrectCount()

{

for(int i=0;i<row;i++)

{

topCorrectTagsCount[i]=0;

for(int j=20\*i;j<20+(20\*i);j++)

{

if(tagsList.get(i).equals(topTagname.get(j))){

topCorrectTagsCount[i]++; //counting topCorrectTags for each tag

}

}

System.out.println("topCorrectTagsCount"+i+":"+topCorrectTagsCount[i]);

}

}

public double calculatePrecision()

{

double precision=0;

for(int i=0;i<row;i++)

{

precision+=topCorrectTagsCount[i]/(double)topTagsCount[i];

}

return (precision/row);

}

public double calculateRecall()

{

double recall=0;

for(int i=0;i<row;i++)

{

recall+=topCorrectTagsCount[i]/(double)tagsCorrect;

//System.out.println("recall="+recall);

}

return (recall/row);

}

public void displayTagsList()

{

for(int i=0;i<topTagname.size();i++)

{

System.out.println("taglist"+topTagname.get(i));

}

}

}

LLDA Learn.scala

import scalanlp.io.\_;

import scalanlp.stage.\_;

import scalanlp.stage.text.\_;

import scalanlp.text.tokenize.\_;

import scalanlp.pipes.Pipes.global.\_;

import edu.stanford.nlp.tmt.stage.\_;

import edu.stanford.nlp.tmt.model.lda.\_;

import edu.stanford.nlp.tmt.model.llda.\_;

val source = CSVFile("javamerge.csv") ~> IDColumn(1);

val tokenizer = {

SimpleEnglishTokenizer() ~> // tokenize on space and punctuation

CaseFolder() ~> // lowercase everything

WordsAndNumbersOnlyFilter() ~> // ignore non-words and non-numbers

MinimumLengthFilter(3) // take terms with >=3 characters

}

val text = {

source ~> // read from the source file

Column(4) ~> // select column containing text

TokenizeWith(tokenizer) ~> // tokenize with tokenizer above

TermCounter() ~> // collect counts (needed below)

TermMinimumDocumentCountFilter(4) ~> // filter terms in <4 docs

TermDynamicStopListFilter(10) ~> // filter out 30 most common terms

DocumentMinimumLengthFilter(2) // take only docs with >=5 terms

}

// define fields from the dataset we are going to slice against

val labels = {

source ~> // read from the source file

Column(2) ~> // take column two, the year

TokenizeWith(WhitespaceTokenizer()) ~> // turns label field into an array

TermCounter() ~> // collect label counts

TermMinimumDocumentCountFilter(2) // filter labels in < 10 docs

}

val dataset = LabeledLDADataset(text, labels);

// define the model parameters

val modelParams = LabeledLDAModelParams(dataset);

// Name of the output model folder to generate

val modelPath = file("llda-cvb0-"+dataset.signature+"-"+modelParams.signature);

// Trains the model, writing to the given output path

TrainCVB0LabeledLDA(modelParams, dataset, output = modelPath, maxIterations = 100);

// or could use TrainGibbsLabeledLDA(modelParams, dataset, output = modelPath, maxIterations = 1500);