SPL-1 Project Report

Bangla Parts of Speech Tagger

Submitted by

Sharif Mohammad Abdullah

BSSE Roll No. : 1211

BSSE Session: 2019-2020

Submitted to

Dr. Ahmedul Kabir

Assistant Professor

Institute of Information Technology

University of Dhaka

[29-05-2021]

1. Introduction

The world of Computer Science and software engineering is huge, NLP(Natural Language Processing), being one of the many different fields. As with any domain of knowledge, this field of Computer Science also has many different aspects, one such is Tagging Parts of Speech. It is corpus linguistics, part-of-speech tagging (POS tagging or PoS tagging or POST), also called grammatical tagging, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context [1].

PoS Tagging is one of the first steps of NLP. Usually, this task is done with the help of high level languages, such as Python, using libraries. My project aims to perform the same task while using none of the libraries and with completely raw code.

First attempts at automating Pos tagging had been done for the English language and since then works and researches for English have developed significantly. Some other languages also have rich works in NLP. It is a matter of sorrow that Bangla, being the fifth most spoken language, little work has been done for the NLP of Bangla. It is my aim to contribute to the development of this field by using the software, "**Bangla Parts of Speech Tagger**", that I have developed for the Software Project Lab-1.

2. Background of the Project

2.1 Definitions

Part Of Speech : a part of speech or part-of-speech (abbreviated as POS or PoS) is a category of words (or, more generally, of lexical items) that have similar grammatical properties. Words that are assigned to the same part of speech generally display similar syntactic behavior (they play similar roles within the grammatical structure of sentences)[2].

NLP : Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.
NLP combines computational linguistics—rule-based modeling of human language—with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to 'understand' its full meaning, complete with the speaker or writer's intent and sentiment[3].

Tokenization : Tokenization is breaking the raw text into small chunks called tokens. These tokens help in understanding the context or developing the model for the NLP. The tokenization helps in interpreting the meaning of the text by analyzing the sequence of the words. Tokenization can be done differently according to different contexts, but in case of PoS tagging the most general format is used which is to tokenize according to punctuation marks, blank spaces and new lines[4].
For example, the text "আমি ভাত খাই।" can be tokenized into 'আমি', 'ভাত', 'খাই'. Of note is that, none of the tokenized words contain the dari(।).

Stemming : Stemming is the process of dividing a word into its root and suffix. For example, "আমগুলো" can be stemmed into its root word "আম" and suffix "গুলো". The rules of stemming may vary across different languages.

3. Description of the Project

The project is based on the algorithm provided in the article "Bangla Parts-of-Speech Tagging using Bangla Stemmer and Rule based Analyzer"[6]. According to the algorithm, the project has been divided into three different parts that work together to produce the final result. Their working procedures are as follows:

Tokenization : The input text must be tokenized in order to work further. This process has two different aspects. The first part is to separate words based on spaces and new lines.

These words contain letters as well as punctuation marks. These marks are removed . What remains are raw data that we will work with in the next steps.

Stemming: To apply the rule based algorithm, words need to be stemmed to find their roots and suffixes. This work is done using the algorithm provided in the article, "Designing a Bangla Stemmer using rule based approach"[7]. It uses the tokenized words obtained in the previous step.

PoS Tagging: The final part of the project which tags words according to the algorithm[6] which works by applying grammatical rules . Moreover it uses a dictionary to find tags. I have implemented a B+ tree to store and search the dictionary.

4. Implementation and Testing

4.1 Tokenization

The first task in tokenization is to store the strings from the input file. This is done by simply taking line-by-line input. These lines are then processed in an input string stream buffer. This buffer is used because getline takes a newline and attaches it with the last string of the line. Using a buffer solves this problem.

The next task is to find punctuation marks, especially the dari symbol. As dari is not a symbol recognized in the English language, it is marked as a unicode character. From the perspective of C++ it can be considered as a 3 length string which is found using KMP string matching algorithm, then it is replaced with "#".

After this process, all that is left is to check for punctuation mark and the # in a string , omit them if found and store it inside a new string.

4.2 Stemming

The stemming process is quite straightforward. According to the algorithm, some words have been marked as non-stemable. We check if the current encountered string is one of them using binary search, if it is, then we store the string in as is form and continue.

Otherwise, we start checking the string from backwards and look for potential suffixes noun,adverb,adjective and number suffixes and "bivokti". If one such is found, the execution stops and stores the word,its root and its suffix.
The aforementioned process is done using the functions shown below. The trimmer is a user defined function that utilizes string matching and replacing functions. and separates a word into its root and suffix.

4.3 PoS Tagging

According to the algorithm , the PoS Tagger function first checks if a suffix exists in the noun,adverb or adjective suffix list.. It is done by a simple linear search as the list is quite small. If it is, this function stores the tag accordingly and stops execution.

If this is not the case then this function tries checking if the word is an quantifier(such as : "টাকা","মিটার","কোটি"etc. The checking is done using a binary search as the list contains more than 30 words. If the string is in the list, the tag is stored as a noun.

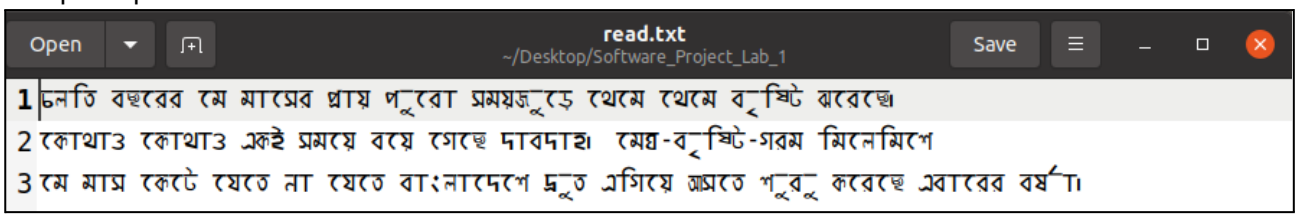
If the tag is a noun and its previous tag is an adjective, the function changes the tags of the previous tag to adjective.

The function next tries to look for a word's root in the quantifier list and stores the tag as a noun if found.

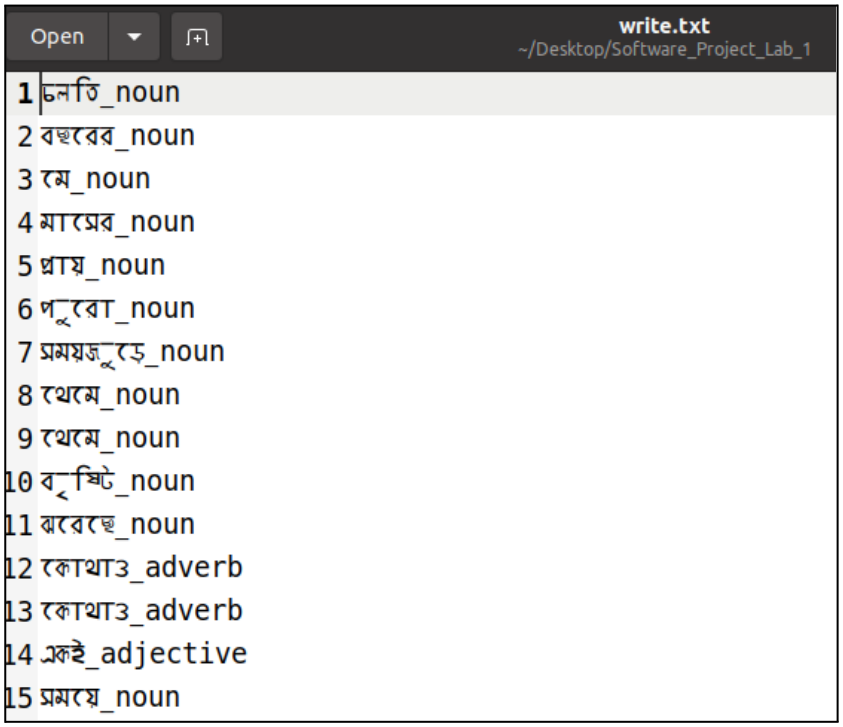
Again if the word and its root are different, the function looks for it in the B+ tree. If it is , the function assigns the tag based on context which is derived from bengali grammar rules.

5. User Interface

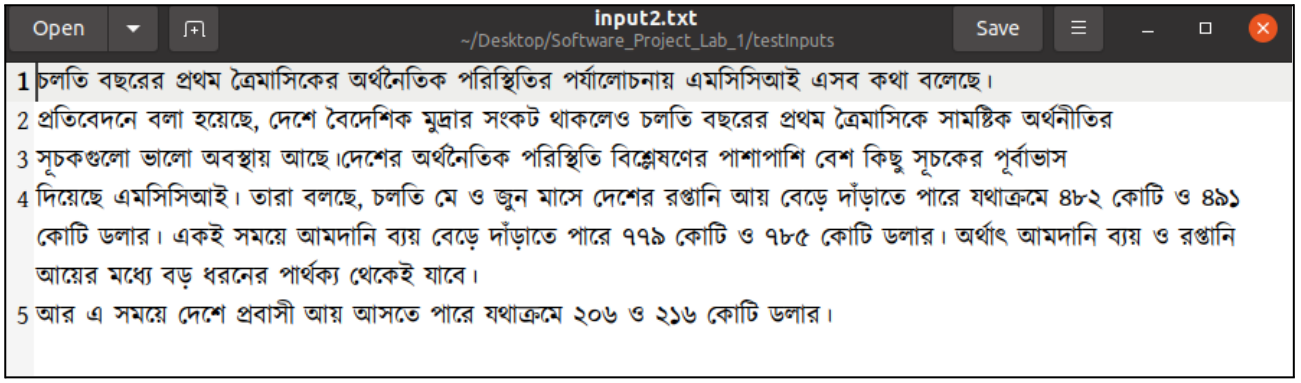
Sample input 1 :



Sample output 1 :



Sample input 2 :

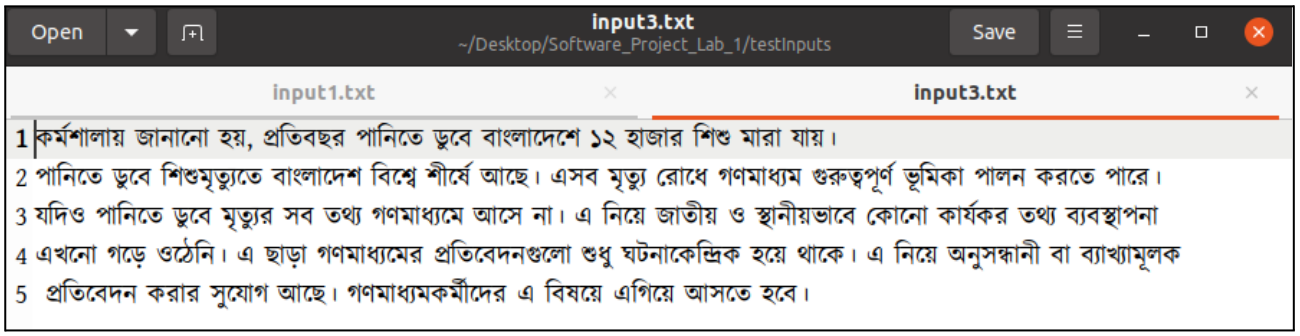


Sample output 2 :

write.txt

```
1  চলতি_noun
2  বছরের_noun
3  প্রথম_noun
4  ত্রৈমাসিকের_noun
5  অর্থনৈতিক_noun
6  পরিস্থিতির_noun
7  পর্যালোচনায়_noun
8  এমসিসিআই_noun
9  এসব_noun
10 কথা_noun
11 বলেছে_noun
12 প্রতিবেদনে_noun
13 বলা_noun
14 হয়েছে_noun
15 দেশে_noun
16 বৈদেশিক_noun
17 মুদ্রার_noun
18 সংকট_adjective
19 থাকলেও_noun
20 চলতি_noun
```

Sample input 3:



Sample output 3 :

write.txt

```
1  কর্মশালায়_noun
2  জানানো_noun
3  হয়_noun
4  প্রতিবছর_adverb
5  পানিতে_noun
6  ডুবে_verb
7  বাংলাদেশে_noun
8  ১২_noun
9  হাজার_noun
10 শিশু_noun
11 মারা_noun
12 যায়_noun
13 পানিতে_noun
14 ডুবে_noun
15 শিশুমৃত্যুতে_noun
```

6. Challenges Faced

The first challenge was to deal with Bangla. Till now we have dealt with ASCII characters which are of 1 byte in C/C++. But Bangla is an Unicode language, thus it takes 3 bytes to store each character. To address this issue, I considered all characters as strings of 3 bytes and operated on them accordingly. I have created a separate stringOperations header file that helps in the operations.

The second was to store a huge amount of data and search for it efficiently. To do this I had to implement B+ tree that was quite challenging.

The third was that projects of this type are usually implemented using high level languages such as Java,Python etc. To implement it in a mid level language , C++ was a challenging task.

Handling large amounts of code, implementing algorithms mentioned in research papers and lack of resources in Bangla NLP were some other challenges.

7. Conclusion

To develop a software was not an easy task but completing it has taught me about implementing algorithms efficiently and using data structure to handle large volumes of data. PoS tagging is the stepping stone for a vast field of research in Natural Language Processing which include but are not limited to various tasks such as information retrieval, parsing, Text to Speech (TTS) applications, information extraction, linguistic research for corpora. They are also used as an intermediate step for higher-level NLP tasks such as parsing, semantics analysis, translation, and many more, which makes POS tagging a necessary function for advanced NLP applications. Moreover, as Bangla NLP is a relatively new research field, there can be a lot of applications of this tool.

8. Reference

[1] Kumawat, Deepika, and Vinesh Jain. "POS tagging approaches: A comparison." International Journal of Computer Applications 118.6 (2015).
[2] Lyons, John. "Towards a 'notional'theory of the 'parts of speech'." Journal of linguistics 2.2 (1966): 209-236.,
[3] https://www.ibm.com/cloud/learn/natural-language-processing,"Natural Language Processing (NLP)", 29/05/2022
[4] Webster, Jonathan J., and Chunyu Kit. "Tokenization as the initial phase in NLP." COLING 1992 Volume 4: The 14th International Conference on Computational Linguistics. 1992.
[5] Shakib, Shahidul, et al. "Designing a Bangla Stemmer using rule based approach." *International Conference on Bangla Speech and Language Processing (ICBSLP)*.
[6] Hoque, Nesarul, and Hanif Seddiqui. "Bangla Parts-of-Speech Tagging using Bangla Stemmer and Rule based Analyzer." *18th International Conference on Computer and Information Technology (ICCIT), 2015*.

9. Appendix

```
while(getline(f,line,' '))
    {
        if(f.good())
        {
            iss.clear();
            iss.str(line);
            while(iss.good())
            {
                iss >> word;
                strings[i] = word;
                i++;
            }
        }
    }
```

Fig 1 : Storing inputs

```
    string DARI1 = "।";
    string DARI2 = "।";

    int find = stringFind(strings[i],DARI1);
    int find1 = stringFind(strings[i],DARI2);

    if(find != -1 || find1 != -1)
        {
            if(find!=-1)
                strings[i] = stringReplace(strings[i],DARI1,"#");
            else
                strings[i] = stringReplace(strings[i],DARI2,"#");
        }
```

Fig 2 : Replacing Dari

```
for (int j = 0; j < strings[i].size(); j++)
{
    if (!ispunct(strings[i][j]) && strings[i][j]!='#')
        updatedStr[id++] = strings[i][j];
}
```

Fig 3 : Storing string while ignoring punctuations

```
if(binary_search(NotStemmed_suffix, NotStemmed_suffix+380, alpha))
    {
        f << "found NotStemmed_suffix where alpha is " << alpha << endl;
        storeIntoWordsWithRoots(alpha,alpha,k);
        f<<wordWithRootAndSuffix[k].first << " = " << wordWithRootAndSuffix[k].second.f
        continue;
    }
```

FIg 4 : Searching for words that will be not stemmed

```
for (int len = n-1; len >=0; len--)
{
    string beta = "";
    beta +=  alpha.substr(len);
    f << " beta at " << i <<"th word at position " << len << " is "<<beta << " where alpha " <
    if(binary_search(nounSuffix, nounSuffix+36, beta))
    {
        f << "found noun suffix" << endl;
        storeIntoWordsWithRoots(alpha, beta, &is_noun, k);
        f<<wordWithRootAndSuffix[k].first << " = " << wordWithRootAndSuffix[k].second.first <<
        break;
    }
```

Fig 5 : Searching for suffixes

```
void storeIntoWordsWithRoots(string alpha, string beta, bool *is_true, int index)
{
    *is_true = true;
    wordWithRootAndSuffix[index].first = alpha;
    wordWithRootAndSuffix[index].second.first = trimmer(alpha, beta);
    wordWithRootAndSuffix[index].second.second = beta;
}

void storeIntoWordsWithRoots(string alpha, string beta, int index)
{
    wordWithRootAndSuffix[index].first = alpha;
    wordWithRootAndSuffix[index].second.first = beta;
    wordWithRootAndSuffix[index].second.second = "";
}
```

Fig 6: Storing words,roots and suffixes

```
tags[index] = getTagFromRule(suffix); //detects noun,adj,adv
if(tags[index] != "unknown")
    {
        cout << tags[index] << endl;
        return;
    }
```

Fig 7 : Storing noun,adverb and adjectives according to rule

```
bool isQuantifierMarker(string word, string *temp)
{
    if(binary_search(quantifierList, quantifierList+36, word))
        {
            *temp = "noun";
            return true;
        }

    else return false;
}
```

Fig 8 : Checking for quantifiers

```
if(tags[index] == "noun") //checks if a word is n
    {
        tags[index-1] = "adjective";
        if(index>1)
            {
                if(tags[index-2] == "adjective")
                tags[index-1] = "noun";
            }
        cout << tags[index] << endl;
        return;
    }
```

Fig 9 : tagging based on context

```
if(word!=root && isQuantifierMarker(root,&temp))  //if a word is not found i
    {
        tags[index] = temp;
        cout << tags[index] << endl;
        return;
    }
```

FIg 10 : Checking for quantifiers based on root

```
if(word!=root)
{
    temp = nodeNonVerb.search(root);
    if(temp != "unknown")
    {
        //temp = nodeNonVerb.search(root);
        tags[index] = temp;
        if(tags[index] == "Adjective")
            tags[index] = "noun";

        else if( temp == nodeVerb.search(root))
            tags[index] = "verb";
        else
            tags[index] = "noun";
        cout << tags[index] << endl;
    }
    else
    {
        temp = nodeVerb.search(root);
        if(temp != "unknown")
            tags[index] = "verb";
        else
            tags[index] = "noun";
        cout << tags[index] << endl;
    }
}
```

Fig 11 : Searching for tag in B+ tree and changing tags according to context