SIMILARITY SEARCH FOR BANGLA

A Thesis

Submitted to the Department of Computer Science and Engineering of

BRAC University

by

Mahbub Morshed

Student ID: 09201023

Shahid Md. Shahed

Student ID: 07101007

In Partial Fulfillment of the

Requirements for the Degree of

Bachelor of Science in Computer Science and Engineering

April 2011

BRAC University, Dhaka, Bangladesh

Declaration

I hereby declare that this thesis is based on the results found by myself. Materials of work found by other researcher are mentioned by reference. This Thesis, neither in whole nor in part, has been previously submitted for any degree.

Signature of Supervisor

Signature of Author

Signature of Author

Acknowledgments

We would like to thank my thesis supervisor, Mr. Matin Saad Abdullah for his guidance and ever helpful comments on my work. We also thank our teachers at BRAC University, our families and friends.

Abstract

Due to typos and misspelling search engines cannot provide users with proper information. Large search engines like Google provides suggestion tab "did you mean". But such options are not included in most of the open source search engines. Our goal was to find a way to implement an exhaustive similarity search in an efficient way and develop such option for Bangla search engine. We used Solr for that. And configured Solr with Lavenstine distance and Jaro Winkler algorithm to provide “Did you mean” for English. But to implement this for Bangla we needed a Stemmer for Bangla and that was not present in Solr. In order to build a efficient stemmer we need to tag the tokens properly according to their parts of speech as the stemming process for different parts of speech is different.

There are different approaches to the problem of assigning a part of speech (POS) tag to each word of a natural language sentence. We have used NLTK toolkit to develop a Regular expression tagger for Bangla verbs using the common suffixes(ক্রিয়া বিভক্তি) found in Bangla grammar. Then we analyzed its performance on main verbs extracted from a 100K token

tagged-corpus. In this thesis we also compare the performance of a few POS tagging techniques for Bangla language, e.g. statistical approach (n-gram) and transformation based approach (Brill's tagger). A supervised POS tagging approach requires a large amount of annotated training corpus to tag properly. We used the 100K token hand tagged corpus developed by Microsoft India to implement these techniques.

Table of Contents

Introduction

Similarity search has become a very important tool for search engines. Nowadays, we depend a great deal on this feature while searching. Google and other search engines have “Did you mean?” where they give us suggestions if our searched word has no good matches. But, these search engines only support English language. Complex languages like Bangla have greater need of this feature as the grammar is very complex compare to English and there is more possibility of spelling mistakes.

This thesis discusses some open source search engine implementation for similarity search as well as comparison between different taggers for 100k corpus for Bangla language. It concludes with an implementation of a custom tagger which can tag out words, especially verb so that analyzing the query gets easier and a better result can be obtained.

Similarity search for Bangla

All modern search engines attempt to detect and correct spelling errors in users' search queries. Google, for example, was one of the first to offer such a facility, and today we barely notice when we are asked "Did you

mean x?" after a slip on the keyboard. But these search engines do not support any other languages except English. For a more complex language like Bangla, this feature is a mandatory as the possibility of spelling mistakes is much more. If we search 'আকাশি' instead of 'আকাশী' we will not get any results although these two words sound the same. So, to make a Bangla search engine fly, we need to implement the "Did You Mean?" feature.

We started out with Apache Nutch and than moved to Solr. We were able to implement similarity search in Solr for English. But to implement this for Bangla we needed a Stemmer for Bangla and that was not present in Solr.In order to build a efficient stemmer we need to tag the tokens properly according to their parts of speech as the stemming process for different parts of speech is different.

Apache Nutch & Apache Solr:

Nutch is open source web-search software. It builds on Lucene and Solr, adding web-specifics, such as a crawler, a link-graph database, parsers for HTML and other document formats, etc. Nutch can run on a single machine, but gains a lot of its strength from running in a Hadoop cluster. Using Nutch, we implemented a full scale search engine. It can be configured to give search results for Bangla words as well as English words. But, to implement "Did You Mean?" even for English is very inefficient as Nutch uses Lucene under it's belt and the spell check suggestion for Lucene gives poor result.

Solr is the popular, blazing fast open source enterprise search platform from the Apache Lucene project. Its major features include powerful full-text search, hit highlighting, faceted search, dynamic clustering, database integration, rich document (e.g., Word, PDF) handling, and geospatial search. Solr is highly scalable, providing distributed search and index replication, and it powers the search and navigation features of many of the world's largest internet sites.

Solr is written in Java and runs as a standalone full-text search server within a servlet container such as Tomcat. Solr uses the Lucene Java

search library at its core for full-text indexing and search, and has REST-like HTTP/XML and JSON APIs that make it easy to use from virtually any programming language. Solr's powerful external configuration allows it to be tailored to almost any type of application without Java coding, and it has an extensive plugin architecture when more advanced customization is required. With Solr we were able to implement spell check suggestion for English words. But, for Bangla words, we need a proper analyzer so that Solr can analyze the queried word properly. For that, we need a stemmer. In the next page, there is a screenshot of our implementation of "Did You Mean?" for English word in Solr. Here, we searched with "sol" and it gave us the suggestion of "solr" which was indexed.

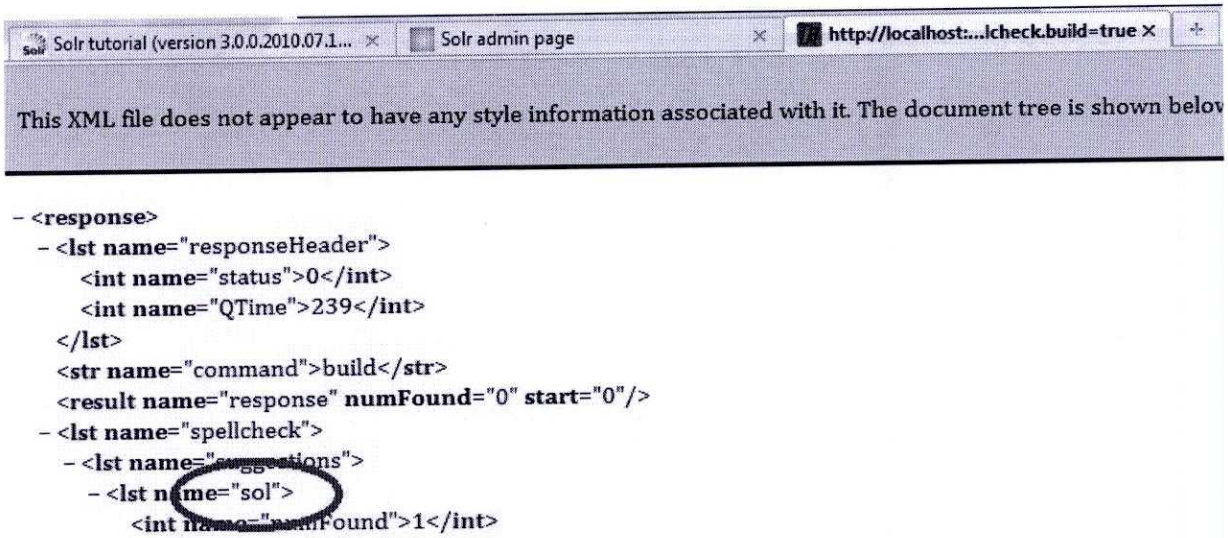

Figure - Similarity Search For English in Solr

Stemming and Lemmatization:

For grammatical reasons, documents are going to use different forms of a word, such as organize, organizes, and organizing. Additionally, there are families of derivationally related words with similar meanings, such as democracy, democratic, and democratization. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set.

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

am, are, is be

car, cars, car's, cars' car

The result of this mapping of text will be something like:

the boy's cars are different colors

the boy car be differ color

However, the two words differ in their flavor. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma . If confronted with the token saw, stemming might return just s, whereas lemmatization would attempt to return either see or saw depending on whether the use of the token was as a verb or a noun. The two may also differ in that

stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma. Linguistic processing for stemming or lemmatization is often done by an additional plug-in component to the indexing process, and a number of such components exist, both commercial and open-source.

The most common algorithm for stemming English, and one that has repeatedly been shown to be empirically very effective, is Porter's algorithm. The entire algorithm is too long and intricate to present here, but we will indicate its general nature. Porter's algorithm consists of 5 phases of word reductions, applied sequentially. Within each phase there are various conventions to select rules, such as selecting the rule from each rule group that applies to the longest suffix. In the first phase, this convention is used with the following rule group:

For Bangla words, specially verbs we need to stem properly to get a better search result. If an user searchs with the word "করতেছিলাম" and that word is not indexed then the search engine should give a suggestion. Here, if the indexed word is "কর" then it should suggest this word as the main root for the word "করতেছিলাম" is "কর". So we need to stem the input correctly to decrease the edit distance, otherwise it may give us some other suggestion. That is why we need a good stemmer. In order to do so, we need to tag different parts of speech. Because stemming process for different parts of speech is not the same. For example, if we extract "তে" from "বাড়িতে", the word will be properly stemmed. But for a verb " খেতে", if we extract "তে" then we will get "খে" which is not the root word. So, we need to tag the words properly so that we can stem properly.

Parts of speech Tagging

In corpus linguistics, part-of-speech tagging (POS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up the words in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context —i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc.

Once performed by hand, POS tagging is now done in the context of computational linguistics, using algorithms which associate discrete terms, as well as hidden parts of speech, in accordance with a set of descriptive tags. [1]

Parts of speech (POS) tagging means assigning grammatical classes i.e. appropriate parts of speech tags to each word in a natural language sentence. Assigning a POS tag to each word of an unannotated text by

hand is very time consuming, which results in the existence of various approaches to automate the job. So automated POS tagging is a technique to automate the annotation process of lexical categories. The process takes a word or a sentence as input, assigns a POS tag to the word or to each word in the sentence, and
produces the tagged text as output.[2]

In the following sections, we start by giving a overview of some of the widely used POS tagging models.

Classification

There are different approaches for POS tagging. The following figure demonstrates different POS tagging models.

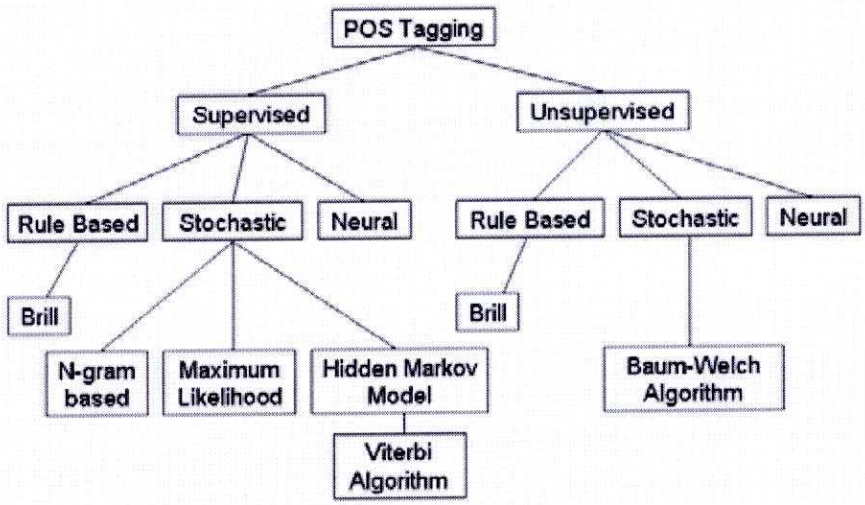

Figure 1: Classification of POS tagging models

Methodology

We implemented and tested the following methods using NLTK tagger.

- Unigram Tagger
- Bigram Tagger
- Brill's tagger
- Regex tagger

Unigram Tagger

The Unigram tagger (n-gram, n = 1) is a simple statistical tagging algorithm. For each token, it assigns the tag that is most likely for that token. For example, it will assign the tag 'adj' to any occurrence of the word 'frequent', since 'frequent' is used as an adjective (e.g. a frequent word) more often than it is used as a verb (e.g. I frequent this cafe).

To use a unigram tagger it must be trained using a corpus. The default taggers assigns 'NC' to unknown words.

Bigram Tagger

The Bigram tagger works in exactly the same way as the Unigram Tagger, the only difference is that it considers the context when assigning a tag to the current word. When training, it creates a frequency distribution describing the frequencies with which, each word is tagged in different contexts. The context consists of the word to be tagged and the tag of the previous word. When tagging, the tagger uses the frequency distribution to tag words by assigning each word, the tag with the maximum frequency given the context. For our case, when a context is encountered for which no data has been learnt, the tagger backs off to the Unigram tagger.

Brill's Tagger

The general idea of the tagger is very simple. It uses a set of rules to tag data. Then it checks the tagged data for potential errors and corrects those. In the same time it may learn some new rules. Then it uses these new rules to again tag the corrected data. This process continues until a threshold in improvement in each pass has been reached.

The Brill tagging model works in two phases. In the first phase, the tagger tags the input tokens with their most likely tag. This is usually done using a Unigram tagging model.

Then in the second phase, a set of transformation rules are applied to the tagged data

Regex Tagger

We also implemented a regex tagger that uses Regular expression to find verbs. In first pass the tagger finds the big suffixes like "-ইতেছিলেন" and

directly assigns it as a verb.

In the second pass the tagger finds the small suffixes and compares it with a verb root. Example of verb roots-

দেখ===>দেখে

পাঠা===>পাঠান

চল===>চলে

কর===>করে

কর===>করে

অতীত কাল

কাল	নাম পুরুষ (সাধারণ)		নাম ও মধ্যম পুরুষ (সম্ভ্রমাত্মক)		মধ্যম পুরুষ (সাধারণ)		মধ্যম পুরুষ (তুচ্ছ)		উত্তম পুরুষ	
	সে		তিনি আপনি		তুমি		তুই		আমি	
	সাধু	চলিত	সাধু	চলিত	সাধু	চলিত	সাধু	চলিত	সাধু	চলিত
৫। সাধারণ	-ইল	-ল	-ইলেন	-লেন	-ইলে	-লে	-ইলি	-লি	-ইলাম	-লাম -লুম
৬। নিত্যবৃত্ত	-ইত Second Pass- তো	-তে	-ইতেন	-তেন	-ইতে	-তে	-ইতিস্	-তিস্	-ইতাম	-তাম -তুম
৭। ঘটমান	-ইতেছিল	-ছিল	-ইতেছিলেন First Pass	-ছিলেন	-ইতেছিলে	-ছিলে -চ্ছিলে	-ইতেছিলি	-ছিলি -চ্ছিলি	-ইতেছিলাম	-ছিলাম
৮। পুরাঘটিত	-ইয়াছিল	-এছিল	-ইয়াছিলেন First Pass	-এছিলেন	-ইয়াছিলে First Pass	-এছিলে	-ইয়াছিলি	-এছিলি	-ইয়াছিলাম -ইয়াছি	-এছিলুম -এছিলাম

Corpora

Corpus size
Bangla – Manually annotated 7168 sentences (102933 words)

Tag Example

Example:

রপ্তানি\JJ.n.n দ্রব্য\NC.0.0.n.n -\PU তাজা\JJ.n.n ও\CCD.n শুকনা\JJ.n.n

ফল\NC.0.0.n.n ,\PU

The tag follows the word separated by a '\' (back slash) immediately after the word. There are no blank
spaces in between. After the whole POS tag there should be at least one blank (white space) before the
next word or a sentinel. In the above example, the first string of 2 to 4 uppercase characters denotes the
Category and Type. For example, in the above sentence the word দগর *is marked* as **NC** which *stands for*
*Noun Common (***N** *denotes Category Noun and* **C** *denotes type Common).*

The attributes are denoted as numbers or letters, as the case may be, after the tag for the lexical category

separated by '.' (dot). The order of the attributes is fixed and cannot be arbitrarily swapped. To illustrate

this, consider the category *proper noun* (NC) whose attribute set is {Number, Case-marker, Definiteness,

and Emphatic}. *Number* can take values from the set {Singular (sg), Plural (pl), Not-applicable (0)};

Case-marker can take values from the set {Accusative (acc), Genitive (gen), Locative (loc), Notapplicable

(0)}; *Definiteness* can take values from {yes(y) and no(n)} and *Emphatic* can take values from

{ yes(y) and no(n)}. *Therefore, for the Common Noun* দগর , *in the above example sentence, which is*

singular, not-applicable, non-definite and non-emphatic, the comple tag should be:

Untagged Data

Example Sentence. রপ্তানি দ্রব্য তাজা শুকনা ফল, আফিম পশুচর্ম ও পশম এবং কার্পেট ।

Tagged Data

রপ্তানি\JJ.n.n দ্রব্য\NC.0.0.n.n -\PU তাজা\JJ.n.n ও\CCD.n শুকনা\JJ.n.n

ফল\NC.0.0.n.n ,\PU

আফিম\NC.0.0.n.n ,\PU পশুচর্ম\NC.0.0.n.n ও\CCD.n পশম\NC.0.0.n.n

এবং\CCD.n কার্পেট\NC.0.0.n.n ।\PU

Previous Work

CRBLP has done some previous work on a small scale. Fahim Mohammad Hasan has worked with 4484 tokens and the results of his comparison is shown below.

Tokens	Unigram Accuracy	Brill Accuracy
0	0	0
60	51.2	50.4
104	51.1	44.6
503	60.7	56.3
1011	64.2	62.6
2023	69.1	67.8
3016	70.1	70.9
4484	71.2	71.3

Table 1: Performance of POS Taggers for Bangla [Test data: 85 sentences, 1000 tokens from the (Prothom-Alo) corpus; Tagset: Level 1 Tagset (14 Tags)]

Tokens	Unigram Accuracy	Brill Accuracy
0	0	0
60	17.2	38.7
104	17.4	26.2
503	26.1	46.1
1011	30	51.1
2023	36.7	49.4
3016	39.1	51.9
4484	42.2	54.9

Table 1: Performance of POS Taggers for Bangla [Test data: 85 sentences, 1000 tokens from the (Prothom-Alo) corpus; Tagset: Level 2 Tagset (41 Tags)]

Test data: 340 sentences, 5029 tokens					
Sentences	**Tokens**	**HMM Accuracy**	**Unigram Accuracy**	**Bigram Accuracy**	**Brill Accuracy**
1785	**25426**	**92.9**	**74.4**	73.2	83

Table 3: Performance of POS Taggers for Bangla on merged training and testing data [Test data and Tagset source: [41]]

According to Fahim Muhammad Hasan, "For Bangla, we did not have any annotated corpus available, and the reason of very low performance of Bangla on our cases is mostly due to the small corpus size"[2]

So in our research we tried out with a large corpus to see that whether performance actually improves or not.

Result

We compared Unigram, Bigram, Trigram and Brill's Tagger using 47,000 token as training data and another 47,000 token as test data. And the result was-

Tokens	Unigram Accuracy	Bigram Accuracy	Trigram Accuracy	Brill's Tagger Accuracy
47,000	83.2%	84.2%	83.8%	83.9%

Result of Regex Tagger -

Total Verbs	First Pass	Second Pass	Total Verbs Found	Accuracy	Verb Root
10518	4492	4986	9478	91.10%	377

Future Work

The tagger that we have built can be further used as a proper stemmer for Bangla language. We need some more efficiency on the stemmer so that our search result can give better output. Then we can implement that into our search engine and make that a good search engine producing similarity search.

Reference

[1] http://en.wikipedia.org/wiki/Part-of-speech_tagging

[2] COMPARISON OF DIFFERENT POS TAGGING TECHNIQUES FOR SOME SOUTH ASIAN LANGUAGES BY FAHIM MOHAMMAD HASAN

[3]http://streamhacker.com/2008/11/10/part-of-speech-tagging-with-nltk-part-2/

[4] Himanshu Agrawal and Anirudh Mani, "Part of Speech Tagging and Chunking with Conditional Random Fields", In Proceedings of the NLPAI Machine Learning 2006 Competition.

[5] http://nutch.apache.org/

[6] Atro Voutilainen, "Does tagging help parsing? A Case Study On Finite State Parsing", University of Helsinki, Finland.

[7] Linda Van Guilder, "Automated Part of Speech Tagging: A Brief Overview", Handout for LING361, Fall 1995, Georgetown University.

[8] http://lucene.apache.org/solr/

[9] http://lucene.apache.org/java/docs/index.html

List of tags

CATEGORY	Attributes
NOUN	Common
	Proper
	Verbal
	Spatio-temporal
VERB	Main
	Auxiliary
PRONOUN	Pronominal
	Reflexive
	Reciprocal
	Relative
	Wh
NOMINAL MODIFIER	Adjective
	Quantifier
DEMONSTRATIVE	Absolute
	Relative
	Wh
ADVERB	Manner
	Location
PARTICIPLE	Verbal (Adverbial)
	Conditional
PARTICLE	Coordinating
	Subordinating
	Classifier
	Interjection
	Others
Punctuation	
RESIDUAL	Foreign word
	Symbol
	Others