Indian Institute of Science Education and Research Kolkata

Department of Computational and Data Sciences

Summer Project Report, May-July 2020

Designing a Context-Informed Stemming Algorithm for Bangla

Submitted by

Rudra Mukhopadhyay

on

15.07.2021

under the supervision of

Prof. Kripabandhu Ghosh

Department of Computational and Data Sciences

IISER Kolkata

Declaration

I declare here that the report included in this project entitled “Designing a Context-Informed Stemming Algorithm for Bangla” is the summer internship carried out by me in the Department of Department of Computational and Data Sciences, Indian Institute of Science Education and Research Kolkata, India from May 16 to July 15, 2021 under the supervision of Prof. Kripabandhu Ghosh.

In keeping with general practice of reporting scientific observations, due acknowledgements have been made wherever the work described is based on the findings of other investigators.

Rudra Mukhopadhyay.		
Rudra Mukhopadhyay	Lokepur, Bankura	July 15, 2021
Student’s e-Signature	**Place**	**Date**

Certificate

It is certified that the summer research work included in the project report entitled "Designing a Context-Informed Stemming Algorithm for Bangla" has been carried out by Mr. Rudra Mukhopadhyay under my supervision and guidance. The content of this project report has not been submitted elsewhere for the award of any academic and professional degree.

Signature of supervisor
Prof kripabandhu Ghosh
Project supervisor

July 15, 2021
IISER Kolkata

List of Abbreviations and Symbols

Symbol	Connotation
IR	Information Retrieval
AP	Average Precision
MAP	Mean Average Precision
P@n	Precision after n retrievals
BM25	Okapi BM25 scoring
co-occ	co-occurrence
d_i	i$^{\text{th}}$ document
t_i	i$^{\text{th}}$ term in some terms set
f_{t_i}	Frequency of a term t_i in the corpus

Contents

Acknowledgement

I sincerely thank IISER Kolkata for giving me the opportunity to carry out my summer internship. Also, I am grateful to my project supervisor Prof Kripabandhu Ghosh for his time, patience and guidance.

Rudra Mukhopadhyay.

Rudra Mukhopadhyay	Lokepur, Bankura	July 15, 2021
Student's e-Signature	**Place**	**Date**

Chapter 1

Designing a Context-Informed Stemming Algorithm for Bangla

1.1 Introduction

In the rapidly expanding horizon of digital communication, multiple regional language communities are becoming more and more prominent on the web platforms through articles, blogs, social media posts, etcetera. Accurate implementation of ad-hoc retrieval models on such non-mainstream languages documents has started demanding some attention. One such language- Bangla, is the focus of this study.

1.1.1 Brief background

To enhance retrieval efficacy and reduce search time (time to retrieve documents for one query), reducing the available vocabulary to an operable set of morphological term units is a crucial step in any ad-hoc retrieval task. Stemming has long been a crude but effective and reliable way to achieve the same. The early stemming algorithms (including one of the earliest Lovins stemmer (1968), the widely used Porter stemmer (1980), Krovetz stemmer (1993), etcetera) were mostly rule-governed. Those were required to be fed with a pre-compiled list of affixes and rules directing how to strip them off to find the stem. Though these algorithms performed satisfactorily [1] for low-inflectional languages, statistical stemming techniques have proved themselves to be more efficient for more inflectional languages [2]. Also, it is laborious and inefficient to acquire and maintain language-specific data for numerous regional languages in practice.
On the other end, a statistical stemmer develops a sense of the language grammar mostly in an unsupervised way. The accuracy of retrieving semantically meaningful variants can be highly improved by employing unsupervised morphology learning via minimum description length principle [3], co-occurrence statistics [4], graph-based prefix recovery [5], N-gram methods [6], etcetera. Due to its flexibility to capture uncompiled morphological rules in a language-independent way, statistical stemmers have witnessed increasing attention in recent years.

1.1.2 Current pursuit

In a broad sense, retrieving morphological variants from a vocabulary is bipartile: namely (i) reductive, (ii) generative [5]. In a reductive technique, possible variants are *reduced* to one root (stem or lemma) while constructing the inverted list and tokenizing the query. Whereas, in a generative technique, the variants are *mapped* to a probable root. Whenever the algorithm encounters one variant from the equivalence class of a particular root while processing a query, it expands that variant to the equivalence class the variant belongs to. The generative model demands more computational memory to execute the search process. However, it reduces the time complexity of constructing the inverted matrix, as each term need not be stemmed before being stored. The following study attempts to implement a basic prefix-stripping stemmer associated with co-occurrence based clustering in a generative way. After stop-word removal (the only language dependent step in the entire system) and monogram-bigram tokenization of the queries, a pre-stemming step is performed.

In this stage, the algorithm finds out the best match for a particular term that (the match) is always a corpus vocabulary member. Thus the prefix-match clustering is performed, followed by a simple co-occurrence based refinement. As a final development, a modified co-occurrence clustering is applied on the prefix-based bag of words. Okapi BM25 score is used to rank the retrieved documents. The proposed system is tested on a corpus containing 98,148 Bangla news articles for 50 pre-compiled query items.

1.2 Aim and Objective

Designing a stemming algorithm using prefix-match and co-occurrence statistics and comparing its efficiency to a no-stemming scenario for a Bangla news article corpus. A step-wise description goes as follows:

1. Document retrieval using the already provided query list with no stemming.
2. Performing a prefix match clustering for individual queries, expanding the query to the collection of its term specific clusters.
3. Refining the prefix match clusters based on certain co-occurrence restrictions.
4. Implementing the proposed stemming system with extended tokenizer and best-match finder.
5. MAP score will be reported for each of the steps to gain a comparative insight.

1.3 Discussion

Mentioned below are the critical components of the proposed system. A brief description is provided for each, along with the motivation behind it. Note that, before implementing any of the following algorithms, the inverted matrix is constructed. The term-specific posting lists contain the document ID's where the term has occurred as well as the frequency of its occurrence in that document. Each of the following steps will use the inverted matrix at least once.

1.3.1 Word monogram and bigram tokenizer

Any query q_i can be represented as $\{t_1, t_2, ..., t_n\}$, i.e. a particular sequence of terms. Any two terms, t_i, t_j, are assumed to be separated by any one or a combination of *space( ), hyphen (-), comma (,), single-quote (').* A query is tokenized using these symbols (or their combinations) as the splitting points.
The most natural way is to use a monogram tokenizer which would yield a set $\{t_i | t_i \in q\}$ from some query q.
Motivation: query item- 'মুম্বইয়ে ডান্স বার নিষিদ্ধ'
Observation suggests that the word 'ডান্সবার' is present in the corpus and might be a potential component of the given query. Similarly, terms like 'লালুপ্রসাদ', 'রামবিলাস', etcetera are meaningful and important morphological variants of 'লালু প্রসাদ', 'রাম বিলাস'.
The first component of the stemming system is designed to capture such variants. Following a monogram tokenization, a bigram tokenization (i.e. $\{t_i t_{i+1} | t_i, t_{i+1} \in q\}$) is performed. After the aforementioned steps, the algorithm checks whether the tokens, thus formed, are present in the corpus (using the inverted matrix). If so, that particular token will be considered as a fit to be stemmed. The process is briefly illustrated below.
Query: 'মুম্বইয়ে ডান্স বার নিষিদ্ধ'
After monogram tokenization: { 'মুম্বইয়ে', 'ডান্স', 'বার', 'নিষিদ্ধ'}
After bigram tokenization: { 'মুম্বইয়েডান্স','ডান্সবার', 'বারনিষিদ্ধ'}
Removing tokens absent from the corpus: { 'মুম্বইয়ে', 'ডান্স', 'বার', 'নিষিদ্ধ','ডান্সবার'}
The basic tokenizing process is given the following algorithm. Its detailed version which is actually used in the system can be found as algorithm 3.
Algorithm 1:

```
1 $q_{init} = \{t_i | t_i \notin \text{ stop-words}\}$
2 tokens = []
3 for $\in \{1, 2, \ldots, n-1\}$
4     target = $t_i$, neighbour = $t_{i+1}$
5     if $i = n - 1$
```

```
        tokens ← neighbour
        end for
    tokens ← target
    tokens ← target + neighbour
    end for
```

1.3.2 Best-match finder

This particular algorithm first generates a *match score* after taking two strings as inputs. It scans two strings from the front ends and the back ends, to find the longest common sub-string starting from both the end. The scanning stops if (i) a mismatch is encountered or (ii) the scanning reaches the middle point of the shorter string. The match value is given by $2\frac{l_f+l_b}{l_1+l2}$, where l_f, l_b denote the length of the common sub-strings from the front and back end of the shorter string, l_1, l_2 denote the length of the two strings.
For example, for the strings are 'abc**x**de' and 'abc**yz**de' the match score is $2 \cdot {}^{l_{abc}+l_{de}}/_{l_{abcxde}+l_{abcyzde}} = 2 \cdot {}^{3+2}/_{6+7} \approx 0.77$.
To enhance the matching efficiency, a booster score is introduced, given by $\exp\left(-0.1 \cdot |l_1 - l_2|\right)$. The above-mentioned best match algorithm would work best for a mismatch of one character (or a few, if they are clustered together). For example, it would not work satisfactorily for two strings- a**m**bcd**n**ef and abcde**p**f. The intuition is this: for one character mismatch (or two-three *occurring together*), the length of the target string, i.e. the *best match* one, would not be changed significantly. Therefore, strings with sufficiently different lengths cannot be a good match to each other. The same is illustrated below:
Case - i) 'abc**wx**de', 'abc**yz**de':
match $= 2 \cdot {}^{3+2}/_{7+7} \approx 0.71$
booster $= \exp\left(-0.1 \cdot 0\right) = 1$
booster $\cdot$ match ≈ 0.71
Case- ii) 'abc**wxy**de', 'abc**z**de':
match $= 2 \cdot {}^{3+2}/_{8+6} \approx 0.71$
booster $= \exp\left(-0.1 \cdot 2\right) \approx 0.82$
booster $\cdot$ match ≈ 0.58
This algorithm hence would claim that the first string-pair is a better match-pair. It is to be noted that 0.1 with the exponential is just a scaling function that prevents a drastic reduction in the match score.
The match finder function takes the string input (a term t_0, say) and it measures the boost-ed match score (say, γ) of that term with each word in the corpus (stored in the inverted matrix), say t_j and picks up only those variants having match scores ≥ 0.7. Ultimately, it returns $t_i \in \{t_j | t_j \in \text{corpus}\}$ such that $\gamma_{t_i} = \max\left(\{\gamma(t_0, t_j) | t_j \in \text{corpus}\}\right)$
Motivation: query item- 'বৃহত্তর নাগাল্যন্ড'
The term 'নাগাল্যন্ড' is not present in the corpus. Using the discussed match-finder, the best match for the given term is found from the corpus as 'নাগাল্যান্ড', with $\gamma \approx 0.78$.
The algorithm to calculate the match score is given below.
Algorithm 2:

```
s1 = string1, s2 = string2
front = [], back = []
f = True, b = True
index = min(len(s1), len(s2))
for $i \in \{1, \ldots$ index$\}$
    f = s1[i] == s2[i], b = s1[index - i -1] == s2[index - i -1]
    if f = False & b = True
        back ← s1[index - i - 1]
    if f = True & b = False
        front ← s1[index]
    if f = b = True
        front ← s1[index]
        back ← s1[index - i - 1]
    if f = b = False
```

```
15         end for
16     if len(front) + len(back) $\geq$ index
17         end for
18     end for
19 return $[2 \cdot {}^{\text{len(front) + len(back)}}/_{\text{len(s1) + len(s2)}}]_1$
```

Note: $[x]_1 = 1$ if $x \geq 1$, x otherwise.

1.3.3 More about bigram and best-match

The scope of the applicability of the best-match-finder algorithm can now be expanded to bigram tokens.
Motivation: query item- 'বাগলিহার জল-বিদ্যুৎ প্রকল্প নিয়ে দ্বিপাক্ষিক সমস্যা'
With bigram tokenization, a token of 'জলবিদ্যুৎ' is recovered. Surprisingly, this term is absent from the corpus. Its nearest match from the corpus is 'জলবিদ্যুত্'. Implementing a best-match algorithm might recover this variant.
This method can lead to another problem, nonetheless. In order to find such spelling mismatched compound variants, it is necessary to apply the match-finder on all bigram tokens. Doing so results in words that are not at all relevant in the context of the query. The same problem does arise even when bigram tokenization with no matching is implemented (Case- iii below). Three examples are illustrated.
Case- i) token:'সড়কপ্রকল্প' $\xrightarrow{\text{best-match}}$ 'সেচপ্রকল্প'
Case- ii) token:'প্রকাশঘিরে' $\xrightarrow{\text{best-match}}$ 'প্রকাশনগরে'
Case -iii) token: '[মন]মোহনসিংহ' $\xrightarrow{\text{present in corpus}}$ 'মোহনসিংহ'
To tackle this problem, co-occurrence of the doubtful term is measured with the certain query terms. In order to explicitly keep a record of the confidence level on a word, a value of 1 is assigned to the words which exist in the query as well as the corpus or are found as best-matches to monogram tokens and 0 otherwise. Say, t_d is a term found via tokenization and/or best-matching. $\{t_c^1, \ldots, t_c^n\}$ be the terms recovered with a confidence value of 1. The co-occurrence vector $u_{t_d} = \{\text{co-occ}(t_d, t_c^1), \ldots, \text{co-occ}(t_d, t_c^n)\}$ is calculated for t_d and thus normalized by f_{t_d}. The empirical condition to reject the term t_d is given by: $\sum_{i=1}^{n} {}^{\text{co-occ}(t_d, t_c^i)}/_{f_{t_d}} < 1$. Otherwise, t_d is accepted as a potential token.
Algorithm 3:

```
1 $q_{init} = \{t_i | t_i \notin$ stop-words$\}$
2 tokens = [], vocab = corpus-vocabulary
3 for $\in \{1, 2, \ldots, n-1\}$
4     target = $t_i$, neighbour = $t_{i+1}$
5     if $i = n - 1$
6         if neighbour $\in$ vocab
7             tokens $\leftarrow$ [1, neighbour]
8         if best-match(neighbour) $\in$ vocab
9             tokens $\leftarrow$ [1, best-match(neighbour)]
10        end for
11    if target $\in$ vocab
12        tokens $\leftarrow$ [1, target]
13        if best-match(target) $\in$ vocab
14            tokens $\leftarrow$ [1, best-match(target)]
15    if target + neighbour $\in$ vocab
16        tokens $\leftarrow$ [0, target + neighbour]
17    if best-match(target + neighbour) in vocab
18        tokens $\leftarrow$ [0, best-match(target + neighbour)]
19    end for
```

Note that best-match() is a function based on algorithm 2. The proposed co-occurrence analysis is performed on the thus found list of tokens.

1.3.4 Prefix match clustering

Assumption: Morphological variants of a word will definitely share a certain common prefix amongst them. An algorithm is used to find the maximum common prefix in this component, given two input strings. It linearly scans both the strings from the front ends and terminates when a mismatch occurs. A common prefix is considered to be a *valid* stem only if its length $l_p \geq {}^2\!/_3 \max(l_1, l_2)$, where l_1, l_2 denote the length of two input strings. To be noted that the scaling factor of ${}^2\!/_3$ is completely heuristic. The process is illustrated below:
Case- i) stem ('লালে', 'লালু') $\rightarrow$ 'লাল'
length(stem) $= 3 \geq {}^2\!/_3 \cdot 4 = 2.67\cdot$ length('লালে')
Therefore, the stemmed value is accepted as a valid one.
Case- ii) stem(('লালে', 'লাশ') $\rightarrow$ 'লা'
However, length(stem) $= 2 < 2.67 \cdot 5$
Hence, 'লা' is not included in the prefix cluster of 'লালে'.
Quite clearly, the stemming process, if considered alone, is not at all an effective way to find morphological variants. Hence, co-occurrence filtering has been used on thus formed prefix-match cluster.
The algorithm used to find the prefix intersection and perform clustering based on the above mentioned restriction is provided below. Algorithm 4.a deals with the prefix-matching process itself, whereas 4.b depicts the process of clustering.
Algorithm 4.a:

```
1 w1 = word1, w2 = word2
2 prefix = []
3 condition = floor(2^{max(len(w1), len(w2))}/3)
4 index = 1
5 while index $\leq$ min(len(w1), len(w2))
6     if w1[index] $\neq$ w2[index]
7         end while
8     prefix $\leftarrow$ w1[index]
9     end while
10 if len(prefix) $\geq$ condition
11     return [prefix, True]
12 else
13     return [prefix, False]
```

Algorithm 4.b:

```
1 bag-of-words = $\{t_1, \ldots, t_n\}$
2 vocab = corpus-vocabulary
3 for $t_i \in$ bag-of-words
5     equivalence-class-$t_i$ = []
6     for word $\in$ vocab
7         if prefix-match($t_i$, word)[2]
8             equivalence-class-$t_i$ $\leftarrow$ word
9         end for
10    end for
```

Note: prefix-match() is the function defined in algorithm-4.a.

1.3.5 Co-occurrence refinement

Assumption: For a term t_0, suppose, the prefix match algorithm has produced a cluster of two words $\{t_i, t_j\}$. Probability that $\{t_0, t_i\}$ are morphological variants is higher than the probability of $\{t_0, t_j\}$ being morphological variants if and only if $\{t_0, t_i\}$ occur together (i.e. co-occur) more number of times.
After prefix-match clustering, a cluster (say, $v = \{t_1, t_2, \ldots, t_n\}$) is generated corresponding to a query term (say, t_0). The matter of concern here ought to be $\mathbb{P}(t_i|t_0)$ (where $t_i \in v$), if we are interested in finding the co-occurring variants of the query word t_0. However, in practice there arises a bias. If $f_{t_i} >> f_{t_j}$ $(t_i, t_j \in v)$, there is always a chance that $\mathbb{P}(t_i|t_0) > \mathbb{P}(t_j|t_0)$, not because of their co-occurrence but for t_i is in general abundant in the corpus. One example is noted below.

co-occc('লাল', 'লালু') $= 36$
Whereas, co-occ('লালুও', 'লালু') $= 32$, co-occ('লালুরা', 'লালু') $= 7$
To counter this problem, $\mathbb{P}(t_0|t_i)$ is also taken under consideration. The refinement is done based on the restriction that $\max\left(\mathbb{P}(t_i|t_0), \mathbb{P}(t_0|t_i)\right) \geq 0.8$ which is again a heuristic condition. One immediate problem arises in this process. The bias gets shifted towards rarer words. I.e. a rare word that by chance occurs with t_0 would not get filtered out. It specifically perturbed the co-occurrence cluster quality for shorter words, as their prefix-match cluster already contains many garbage words. As an example, co-occurrence filtered cluster of 'বার' still contains words like 'বাড', 'বাত', 'বাচ', etcetera.
Nevertheless, a much-refined cluster is achieved for most of the words after carrying out this step. Provided below just one illustration:
Query term: 'গুর্জর'
After prefix-match clustering: { 'গুর্বো', 'গুর্জর', 'গুর্জরের', 'গুর্জরকে', 'গুর্জরি', 'গুর্জরী', 'গুর্তু', 'গুর্জররা', 'গুর্খা', 'গুর্জং', 'গুর্জরদের', 'গুর্জররাও', 'গুর্জরেরা', 'গুর্জরও' }
After co-occurrence refinement: { 'গুর্জর', 'গুর্জরের', 'গুর্জরকে', 'গুর্জররা', 'গুর্জরদের', 'গুর্জররাও', 'গুর্জরেরা', 'গুর্জরও' }

1.3.6 Further refinement: tan hyperbolic filter

As discussed, the last refinement method has been quite *loose*. Neither $\mathbb{P}(t_i|t_0)$ nor $\mathbb{P}(t_0|t_i)$ (where $t_i \in v$, the prefix-match cluster of t_0) is a good measure, if taken separately. This particular development deals with this problem. In this section, a method of combining these two values, and forming a cluster based on that, is discussed.
Define $\rho(t_0, t_i) := \dfrac{(\text{co-occ}(t_i, t_0))^2}{f_{t_i} \cdot f_{t_0}} = \mathbb{P}(t_i|t_0) \cdot \mathbb{P}(t_0|t_i)$
Assume t_i is an extraordinarily common word which is the sole reason behind its significant co-occurrence value with t_0. In that case $f_{t_i} >> \text{co-occ}(t_i, t_0) \implies \rho(t_i, t_0) \to 0$. On the other hand, if t_i is a rare word and occurs by chance with t_0, $f_{t_0} >> \text{co-occ}(t_i, t_0) \implies \rho(t_i, t_0) \to 0$. Hence, ρ measure is free from these two types of bias.
The next problem is its threshold value selection. Along the line of the last development, it is indeed possible to introduce a heuristic threshold cut-off. Before that, a *clustering* function is applied on the ρ statistics with an intention to *exaggerate* the ρ distribution. Provided below is a rigorous description of the algorithm.
As defined earlier, $v = \{t_i | i \in \{1, 2, \ldots, n\}\}$ denotes the prefix-match cluster of a particular query term t_0. ρ measurement is carried out on this array v, resulting in $v_\rho = \{\rho_{t_i} | i \in \{1, 2, \ldots, n\}\}$.
The *clustering* function of interest is a tan hyperbolic. Say, μ, σ denote the mean and standard deviation of the elements in v_ρ respectively. The clustering is defined as $(F)_{v_\rho} := \tanh\left({}^{x-\mu}/_{\sigma}\right)$, $x \in v_\rho$.
Motivation: Intuitively, a natural way to use such a non-linear *clustering* function is the following. Say the function $\mathcal{F}$ saturates towards 1, i.e. $\lim_{x\to\infty} \mathcal{F}(x) \to 1$. A threshold is set, based on empirical observation, at some value $\alpha < 1$, and that all the data points which are *significantly* far away from the population mean must be clustered together in the range $[\alpha, 1]$. The modified tan hyperbolic does the same job in this scenario.
As represented in Fig-1.1, for the same threshold value (0.7), clustering has improved significantly (top 13.3% of the data-set available in the cluster, compared with the 3.3% in no filtering scenario).
This technique, once implemented on the prefix-match clusters already made, many junk words are gotten rid of. However, to further improve the accuracy (for words that are hardly used with their morphological variants), a booster score is introduced. In this part, the focus is primarily on the nature of the stem (or the prefix-intersect). Say, stem of t_i with t_0 is a string $s_{0,i}$. Defined booster score $= 1 + \tanh\left({}^{f_{s_{i,j}}}/_{f_{t_0}}\right)$ and 1 otherwise. If the stem is present in the corpus and its frequency is way too low compared with that of the root word t_0, the booster drags down their ρ score. Finally, the top five variants are selected based on the boosted ρ scores from the words in a particular equivalent class. Noted below is an instance where using this parameter has improved the clustering.
Query word: 'লাল'
Tanh clustering without booster: {'লাভ', 'লাইন', 'লালের', 'লাগল', 'লাগে'}
Clustering with booster: {'লালকে', 'লাগে', 'লালম', 'লালে', 'লালের'}
Depicted in Fig-1.2 are the clustering-s based on the scores given by booster along with the ρ value. Just to mention, the final cluster for 'মিনা' is found to be {'মিনাও', 'মিনার', 'মিনারা', 'মিনাদের'}.

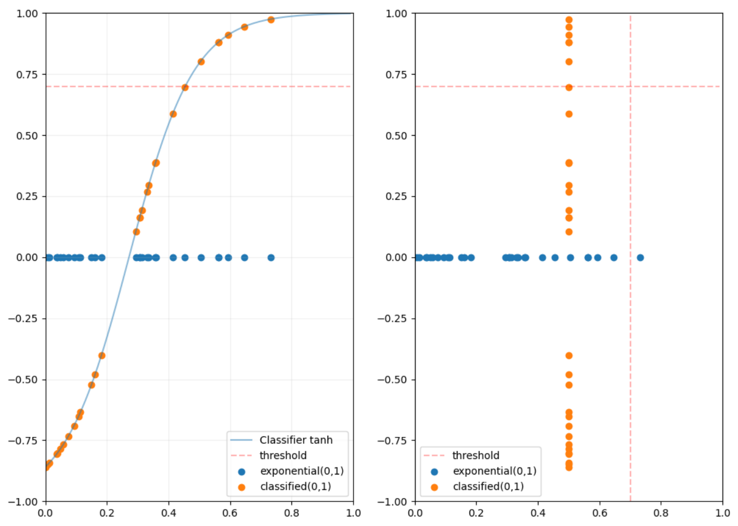

Figure 1.1: A random exponential distribution with 30 points and its tanh-clustering

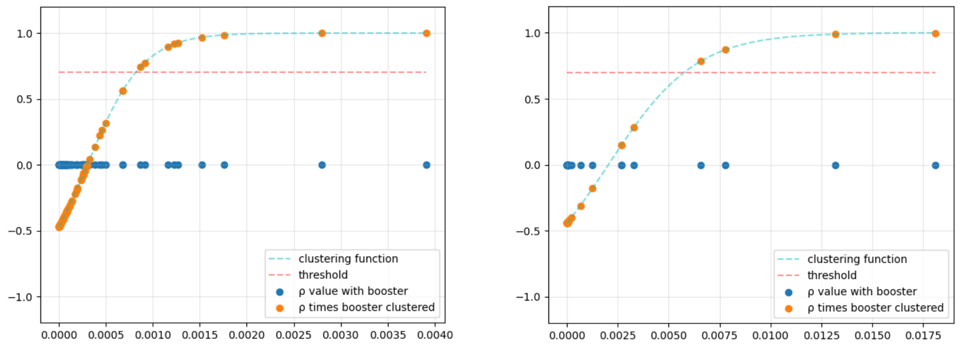

Figure 1.2: Tanh clustering for terms 'লাল' (left) and 'মিনা' (right)

1.4 Experimental section

1.4.1 Preliminaries

As mentioned in section 1.2, the proposed system is tested in five stages. A corpus with 98,148 documents (Bangla news articles) and 50 pre-compiled queries are used for this purpose. BM25 scoring is used in order to rank the retrieved documents. For the sake of completion, the formulation of the BM model is described below.

$$\text{BM25 weight} = \sum_{t \in q} \text{tf-idf}_{t,d} \frac{k_1 + 1}{k_1((1-b) + b \cdot (L_d/L_{avg})) + f_{t,d}}$$

Where tf-idf weight $= f_{t,d} \cdot \log(N/\text{df}_t)$
q is some query, N = number of documents in the corpus, $f_{t,d}$ = term frequency of t in document d, df_t = number of document in which t occurs, L_d, L_{avg} = length of document d and average document length respectively, b, k_1 are real parameters (values taken as 0.75, 1.2 respectively).

To be noted here that to enhance the run-time efficiency, an algorithm is implemented that seeks the *probably relevant* documents. BM25 will return a BM-score of zero if none of the words from the query (or the query-word specific clusters) is present in the document (as $f_{t,d}$ becomes zero). Therefore, the retrieval operation is performed only on those documents d_i for which $\exists t \in q_{\text{extended}}$ such that $t \in d_i$ ($q_{\text{extended}} = \cup_{t_i \in q} v_{t_i}$, v_{t_i} being the equivalence class of term t_i).
Also, a pre-compiled stop word list is used to remove the redundant stop words from the queries and the inverted matrix. Every experiment is carried out in a stop word free scenario.
At each of the following steps, MAP, P@10, P@5 are calculated. These will be referred to in the Discussion section.

1.4.2 Retrieval using mere query tokens (Control or baseline)

It is taken to be the baseline scenario for the proposed system. None of the aforementioned algorithms is used, except for monogram tokenization.

1.4.3 Prefix-match clustering and retrieval (Setup 1)

$\forall t_i \in q$ we find a v_{t_i} such that $\forall t_{i,j} \in v_{t_i}, \text{prefix-intersect}(t_i, t_{i,j}) \geq {}^{2\cdot\max(l_{t_i}, l_{t_{i,j}})}/_{3}$. The query q is thus represented as $Q = \cup_{t_i \in q} v_{t_i}$ and the BM score based ranked retrieval method is performed on Q.

1.4.4 Co-occurrence filtering (Setup 2)

The algorithm discussed in subsection 1.3.5 is implemented on the expanded query found via prefix-matching. We find $Q_{\text{co-occ}} = \cup_{t_i \in q} v_{t_i}^{\text{co-occ}}$, where $v_{t_i}^{\text{co-occ}} \subset v_{t_i}$ with the already discussed heuristic restriction. Retrieval is again performed on $Q_{\text{co-occ}}$.

1.4.5 With the proposed system (Setup 3)

The following developments are implemented in this step.

- Monogram as well as bigram tokenization, along with recovering best-matches
- Chopping off irrelevant matches found, using the method discussed in 1.3.3.
- Tanh based clustering (using the boost score) of Q

BM25 based retrieval is performed on the resulting cluster. The performance statistics are noted and discussed in the next section.

1.4.6 Proposed system along with a scaled BM scoring (Setup 4, 5)

Motivation: Consider the query- 'রাম মন্দির নিয়ে আডবাণী-সিঙ্ঘল সংঘাত'
The AP value decreases for prefix-match based retrieval and co-occurrence based retrieval (0.04 and 0.07 respectively), if compared with raw query tokens based retrieval (AP = 0.83). Observation suggested the following.
Say, $\cup v_{t_i}$ is the extended form of the query containing the variants found for each term (either prefix-matched terms or those after co-occurrence filtering) t_i. It might so happen that a certain term t_0, with a significant number of variants, can alone cause a high BM score for a document d if $|\cap(v_{t_0}, d)| > |\cap(v_{t_i}, d)|, t_i \in q$. This is exactly what happened in the illustration above. The retrieved document with top BM scores contain a high number of morphological variants of 'আডবাণী' (e.g. {'আডবাণীর', 'আডবাণীকে', 'আডবাণী', আডবাণীও'} etcetera), and hardly any other term from the same query. Similar has been the scenario for the following queries:
Q: 'পশুখাদ্য কেলেঙ্কারিতে লালু প্রসাদ যাদব', ('লালু' as the *dominat* variants supplier.)
Q: 'প্রমোদ মহাজনের বাংলোয় মদের আসর', ('প্রমোদ', 'বাংলোয়' being the dominant ones.)
Assumption: Terms in the query (except the stop words) are relevant and essential to the intention of the query.
With this assumption being made, we first define a factor β such that $\beta_{t_i} = 1$ if $\cap(v_{t_i}, d) \neq \phi$ and 0 otherwise. Now define a scaling factor $n := \sum_{t_i \in q} \beta_{t_i}$. Say a query with three original terms is expanded to

$q = \{v_{t_0} = \{t_{01}, \dots\}, \dots, v_{t_3} = \{t_{31}, \dots\}\}$. If at least one representative from each cluster is present in the document d, $n = 3$. If at least one term from any two clusters is present in d, $n = 2$. The ranking is done based on the value of $(n \cdot \text{BM score})$ (Setup 4).
As this factor is a linear one, for long queries the altered score value would be abruptly high. To exclude this possibility, a scaling factor $n_{\text{exp}} := 2 - \exp(-n)$ can be used (Setup 5). As n grows, $n_{\text{exp}} \to 1$. Using n_{exp} actually punishes documents with lesser number of *useful* representatives from the query, whereas using n rewards the documents with higher number of *useful* representatives.

1.5 Conclusions

1.5.1 Average precision and Precision at 10

Average precision for a query is defined as $\text{AP} = \frac{\sum_{i=1}^{n} p(i) \cdot \mathcal{I}(i)}{\sum_{i=1}^{k} \mathcal{I}(i)}$, where $\mathcal{I}(i) = 1$ if d_i is a relevant document and 0 otherwise, $p(i)$ denotes the precision value after i documents are retrieved. Mean average precision, therefore, is given by $\text{MAP} = \frac{\sum_{q_i} \text{AP}\, q_i}{N}$, N being the total number of queries. In the experiment section, for every query top 50 retrieved results are considered ($n = 50$). Total number of queries is 50 as well (hence, $N = 50$).
P@10 ($p(10)$) is another query-specific measure of the accuracy of the IR model. Given below are two plots representing the change of AP and P@10 behaviour at different stages of testing.

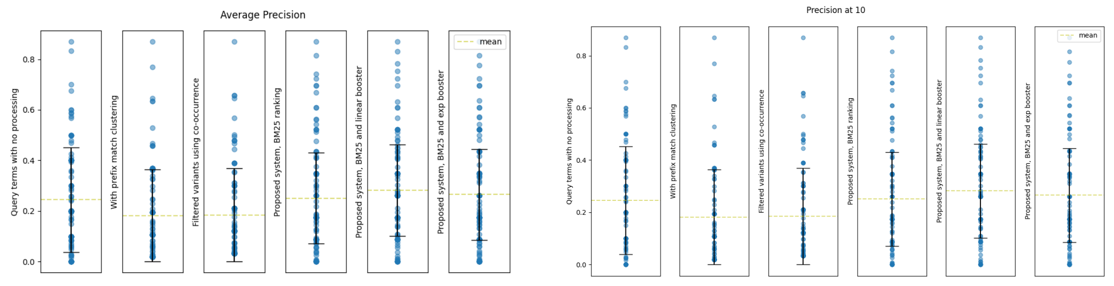

Figure 1.3: AP (left), P@10 (right) at different testing phases: The error bar corresponds to the standard deviation error

1.5.2 Mean Average Precision comparison

Using the formula for MAP, as mentioned above, the following values are calculated.

IR model	MAP
Monogram tokens from query with no processing (Control)	0.255
Queries expanded to prefix-match clusters (Setup 1)	0.194
Using prefix-clusters, refined using co-occurrence (Setup 2)	0.174
Proposed technique, with regular BM rank (Setup 3)	0.309
Proposed technique, ranked on linearly scaled BM (Setup 4)	0.350
Proposed technique, ranked on exponentially scaled BM (Setup 5)	0.326

Table 1.1: MAP value comparison

The proposed stem-based clustering technique has resulted in an improved MAP score. In the next section, a T-test analysis is performed to infer whether the improved MAP score is statistically significant.

1.5.3 T-test on AP values and percentage improvement

Null hypothesis (H_0): There is no statistically significant difference between the MAP values in Control versus that in Setup 3, 4 or 5.
Alternate hypothesis (H_a): The MAP value for Control is significantly different from that in Setup 3, 4 or 5.
As the subjects remain the same across various setups (a fixed set of 50 queries), paired T-test is used in this context.

Sample pair	t statistic	p value
Control - Setup 3	-2.12	0.03662
Control - Setup 4	-3.89	0.00018
Control - Setup 5	-2.98	0.00364

Table 1.2: Paired T test on AP values

A significance threshold score of 0.05 is considered here. As the p value in each of these cases is < 0.05, the null hypothesis is rejected. Each of Setup 3, 4, 5 MAP values is significantly different from the control MAP score. The improvement is therefore meaningful.
In order to quantify the degree improvement, the percentage increase is MAP values are recorded below.
% improvement in Setup 3: $100 \cdot {}^{(0.309-0.255)}/_{0.255} \approx 21.18\%$
% improvement in Setup 4: $100 \cdot {}^{(0.350-0.255)}/_{0.255} \approx 37.25\%$
% improvement in Setup 5: $100 \cdot {}^{(0.326-0.255)}/_{0.255} \approx 27.84\%$

1.5.4 Query specific analysis

It is worthy of discussing a few queries and how the proposed stemming technique has improved the retrieval accuracy even when no scaling is done on the BM score (Setup 3).

Query: 'মুম্বইয়ে ডান্স বার নিষিদ্ধ'

While using plain query terms or Setup 2, only one relevant document is found amongst the top twenty results. This inefficiency is attributed to the fact that most of the relevant documents contain the term 'ডান্সবার' and not 'ডান্স', 'বার' separately.
With bigram tokenization and relevant processing, it has been possible to retrieve seven relevant documents (in Setup 3) and eight relevant documents (Setup 4, 5).

Query: 'প্রতিরক্ষা বিভাগে অস্ত্র কেলেঙ্কারির তদন্ত'

This particular case depicts the usefulness of the best-match finding component. The term 'প্রতিরক্ষা' is absent from the corpus. The term 'প্রতীরক্ষা' is retrieved sing the match finder algorithm. Neither raw query term search (i.e. Control) nor Setup 1, 2 had been able to find one relevant document (in the respective top 20 retrieved document sets). However, Setup 3 has found two, whereas Setup 4, 5 have found three relevant documents after the top 20 retrievals.

Query: 'পশুখাদ্য কেলেঙ্কারিতে লালু প্রসাদ যাদব'

This query illustrates very well the reason behind the poor performance of Setup 1 and 2. As mentioned earlier, even after co-occurrence based filtering, the term 'লালু' has an exceptionally high number of variants if compared with the other terms in the same query (e.g. 'পশুখাদ্য' has one element in its filtered cluster, 'কেলেঙ্কারিতে' has two, while 'লালু' has sixteen).
The stricter tanh clustering eliminates this bias and yields quite a *uniform* result. The clusters mentioned earlier now look like the following.
'লালু': {'লালুকে', 'লালুর', 'লালু'}
'পশুখাদ্য': {'পশুখাদ্যে', 'পশুখাদ্য', 'পশুখাদ্যের', 'পশুখাদ্যেও'}
'কেলেঙ্কারিতে': {'কেলেঙ্কারি', 'কেলেঙ্কারির', 'কেলেঙ্কারিতে'}
More homogeneous clustering is reflected in the search result as well. Out of the top twenty results, Setup 1 and 2 brought two relevant documents. However, Setup 3 has retrieved four and Setup 5 has retrieved six relevant documents in their top twenty searches.

Query: 'গুর্জর ও মিনা সম্প্রদায়ের মধ্যে সংঘর্ষ'

It must be noted that even though the average performance of Setup 1 and 2 is unsatisfactory if compared with mere query term retrieval, there have been occasions when Setup 2 has notably outperformed the control setup. The retrieval task of the query mentioned above holds one such situation. The primary reasons behind Setup 2 performing well are the following.

- Frequencies of the terms like 'গুর্জর', 'মিনা' are *optimal*, i.e. these are neither exceptionally rare nor highly abundant. This makes the co-occurrence filtering a success.
- Most of the relevant documents have a proportionate representation from all the four clusters (clusters corresponding to 'গুর্জর', 'মিনা', 'সম্প্রদায়ের', 'সংঘর্ষ'. Therefore documents have rarely received a high BM score for over-representation of one cluster.
- Each of the terms being significantly long, the prefix match has not allowed loads of junk matches.
- Most importantly, absence of spelling mismatch or broken parts of a compound word.

For this particular query, a figure containing the precision-recall curves of Control and Setup 2 is provided below.

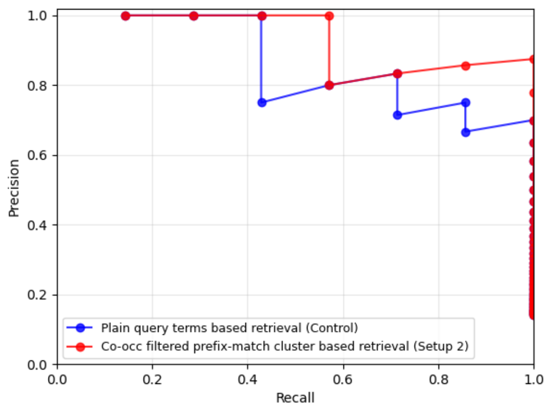

Figure 1.4: Precision-recall curve for the above mentioned query

1.6 Future Directions

1. The immediate step is to test the proposed stemmer on an extended set of queries and to analyze whether the improvement in MAP score remains consistent throughout.

2. In this study, the baseline is considered to be a no-stemming scenario. The proposed system has been shown to work well if compared with the baseline (referred to as Control in the earlier section). In practice, however, the baseline is not a no-stemming situation, for there exist a multitude of effective rule-governed and statistical stemmers. Some stemmers (including YASS [2] which uses a string distance function to measure morphological similarity between two terms, GRAS [5] that implements a graph-based stemming method) used for Bangla documents have previously shown a significant improvement in retrieval efficacy over most of the rule-based stemming methods (e.g. Lovin's, Porter's). One necessary step, therefore, is to compare the performance of the current system with the previously mentioned stemmers over the same set of queries in order to gauge its *practical* efficiency.

3. Except for the component concerning stop-word removal, the system is language-independent. Therefore, the same algorithm is expected to perform satisfactorily in other languages once fed with an up-to-date stop-word list. Any future work is expected to report its overall performance (compared with plain-word retrieval and retrieval using common stemmers) on other inflectional languages.

4. A primitive prefix-match clustering is used to capture the morphological variants, and a co-occurrence based function is employed to filter out possible semantical variants. A modern way to achieve this

filtering is via using the skip-gram model. That is to say, given a target word and its prefix-match cluster, a skip-gram model (e.g. word2vec) can be applied to find the terms off the cluster which are closest to the target word in a contextual sense (occurring in a similar context). Once implemented, further performance analysis needs to be done.

5. The proposed system also needs to be contrasted against state of the art stemming or lemmatizing techniques with supervised or semi-supervised learning support. Incorporating modern machine learning, ANN based algorithms in the current system, replacing the primitive prefix-match method, is a possible scope of improvement.

1.7 References and Notes

[1] Porter, M.F. 1980. *An algorithm for suffix stripping.* Program 14, 3, 130–137.

[2] Majumder, P., Mitra, M., Parui, S. K., Kole, G., Mitra, P. & Datta, K. 2007. *YASS: Yet another suffix stripper.* ACM Trans. Inform. Syst. 25, 4, Article 18 (October 2007), 20 pages. *DOI = 10.1145/1281485.1281489 http://doi.acm.org/10.1145/1281485.1281489*

[3] Goldsmith, J. 2001. *Unsupervised learning of the morphology of a natural language.* Computational Linguistics. 27, 2 (June 2001), 153–198. *DOI:https://doi.org/10.1162/089120101750300490*

[4] Paik, J.H., Pal, D. & Parui, S.K. 2011. *A novel corpus-based stemming algorithm using co-occurrence statistics.* In Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval (SIGIR '11). Association for Computing Machinery, New York, NY, USA, 863–872. *DOI:https://doi.org/10.1145/2009916.2010031*

[5] Paik, J.H., Mitra, M., Parui, S.K., & J¨arvelin, K. 2011. *GRAS: An effective and efficient stemming algorithm for information retrieval.* ACM Trans. Inf. Syst. 29, 4, Article 19 (November 2011), 24 pages. *DOI = 10.1145/2037661.2037664 http://doi.acm.org/10.1145/2037661.2037664*

[6] James Mayfield & Paul McNamee. 2003. *Single n-gram stemming.* In Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '03). Association for Computing Machinery, New York, NY, USA, 415–416. *DOI:https://doi.org/10.1145/860435.860528*