Pipilika N-gram Viewer: An Efficient Large Scale N-gram Model for Bengali

Adnan Ahmad*, Mahbubur Rub Talha*, Md. Ruhul Amin*, Farida Chowdhury*
*Search Engine Pipilika
*Department of Computer Science & Engineering
*Shahjalal University of Science & Technology
Sylhet, Bangladesh
{sust.adnan, talha13, shajib.sust, deeba.bd}@gmail.com

***Abstract*—In this paper, we introduce a large-scale Bengali *N*-gram model, trained on online newspaper corpus and present results and analysis of two different experiments done by using the model, namely Context-aware spell checker and Trending topic detection. We also present the process with emphasis on the problems that arise in working with data at this scale. One signicant aspect of our *N*-gram model is that the model contains information of *N*-gram occurrence per day over a period of eight years, from the year 2009-2017. This enables further applications of the model, for example, Trending topic detection. Our Bengali *N*-gram language model contains *N*-grams up to 5-gram with more than 2 million unique Unigrams and over 656 million duplicate Unigrams. We evaluate our model by calculating the perplexities of different years. We obtain F-score of 86.6% in an experiment of Context-aware spell checker. In another experiment, we successfully detected the trending topics of a given time frame. This paper also presents first Bengali *N*-gram viewer, where one can query by a particular *N*-gram and see the resulting graph of the frequency of that term occurred using different time frames.**

***Keywords*—** *Bengali N-gram Model, Bengali N-gram Viewer, Bengali Context-aware Spellchecker, Bengali Trending Topic Detection*

I. INTRODUCTION

For low resource languages like Bengali, where the amount of digital text or web content is not very large compared to other major languages, available amount of data in Newspaper corpora makes it attractive for many natural language processing tasks, such as language modeling. Web-scale language models have been shown to improve performance of many language processing tasks, such as Spell-checker, Machine translation, Automatic speech recognition and information retrieval etc. [1] [2] [3]

In this paper, we introduce the largest *N*-gram model for Bengali which is prepared using large-scale web content, ranging from 13 different online newspapers, including news articles from year 2009 to 2017. This large amount of data is crawled, preprocessed and used as train data for the *N*-gram model. The model contains frequency per day for every *N*-gram up to 5-gram, which required efficient large-scale computational solutions. Also, indexing technique is applied for fast and efficient retrieval of *N*-gram frequencies from the model. In an addition, our work also contains a *N*-gram viewer, where one can see the occurrence of a *N*-gram over a particular time period and a *N*-gram API[1] for querying in the model. Although the idea of *N*-gram viewer is much more similar like Google *N*-gram viewer[2], there are two major aspects where our *N*-gram viewer differs from it. Firstly, Google *N*-gram viewer is based on books, where our *N*-gram is based on daily newspapers. Secondly, Google *N*-gram viewer shows a normalized value of *N*-gram frequency in a per year time frame where our *N*-gram model contains frequency per day which requires more computations. But this feature enables us the opportunity to use this model in tasks like trending topic detection which requires *N*-gram frequency of much smaller time frame.

Later, to evaluate our model, we calculate the perplexity of the model which is a standard evaluation method for language models. We also use it to perform multiple NLP tasks, namely context-aware spell checker and Trending topic detection. We include the performance results of these tasks and show the efficiency of the model. This model can be used in many farther NLP tasks by researchers, so we decided to make our *N*-gram model publicly available.

The rest of the paper is arranged in following manner. Section II includes some background studies of large-scale *N*-gram models for both Bengali and English. Section III contains the details of data collection and preprocessing. Later, section IV contains the model generation and section V contains model evaluation. Finally, we conclude our work in section 6.

II. BACKGROUND

The first edition of the Google Books *N*-gram Corpus is introduced here [4] and they used it to quantitatively analyze a variety of topics ranging from language growth to public health. Google also released *N*-gram viewer, which has become a popular tool for examining language trends. The corpus contains both words and phrases and their frequency over time. Currently Google Books *N*-gram Corpus contains over 8-million books, or 6% of all books ever published [5]. The Google Books *N*-gram Corpus has been available since 2010, currently contains 8 languages, however it doesn't contain Bengali. On the other hand, there is no available large-scale Bengali *N*-gram model till date.

[1]https://developers.pipilika.com/ngram, Accessed on June 02, 2018
[2]https://books.google.com/ngrams, Accessed on June 02, 2018

978-1-5386-8207-4/18/$31.00

III. Data preparation

In the fields of computational linguistics and probability, *N*-gram is a contiguous sequence of n items from a given sequence of text or speech. An *N*-gram of size 1 is referred to as a Unigram; size 2 as a Bigram and size 3 as a Trigram. Larger sizes are sometimes referred to by the value of n in modern language, e.g., four-gram, five-gram, and so on. Using the statistical properties of *N*-grams, one can build *N*-gram language model. Our primary goal is to create a model that can be used to be queried for language model probability. To achieve that goal, we need to compute our model on large amount of data. We choose online newspaper contents as our primary source of data.

A. N-gram data collection

Bengali is a low resource language. Unlike major languages like English or Chinese, there isnt much data available for Bengali in online. The main source for data is Newspaper, Blogs, Online portals and Wikipedia. As the goal is to create an *N*-gram model as well as *N*-gram viewer to retrieve and show frequencies of *N*-grams per day, newspaper data is ideal for that. We collected articles from 13 different Bengali online newspapers, in a range of year 2009 to 2017. A chart representing number of words per news website is given in figure 1.

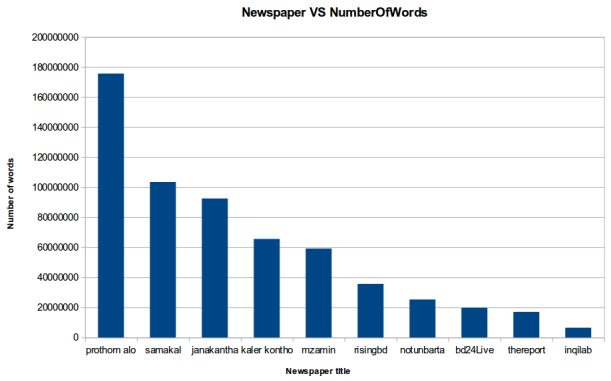

Fig. 1. Number of words per news website.

Our data contains duplicate sentences and that includes a risk of having a certain amount of redundant data, for example, copyright text, newspaper address etc. We do not remove duplicate sentences as that may result in removing valuable sentences from the original data. Also, we don't include blog or Wikipedia data as these data do not have the properties of daily frequency count. A graph of frequency spectrum is shown in figure 2.

We limit our graph to 50 spectrum as a spectrum is often characterized by very high values corresponding to the lowest frequency classes, and a very long tail of frequency classes with only one member. Thus, a full spectrum plot on a non-logarithmic scale will always have the rather uninformative L-shaped profile. So, we plot it on a log-log scale. The same distribution shows itself to be linear in figure 3. This is the characteristic signature of a power-law.

In order to develop an intuition about how rapidly vocabulary size is growing, Vocabulary Growth Curve (VGC) is given in figure 4. A Vocabulary Growth Curve reports vocabulary size (number of types, V) as a function of sample size (number of tokens,N).

Another graph shows the log-log relationship between the rank and frequency of unique words in the corpus in figure 5. This graph is generated in order to understand how terms are distributed across documents. Zipf's law is a commonly used model of the distribution of terms in a collection.The graph shows that the fit of the data to the law is good enough to serve as a corpus for language model.

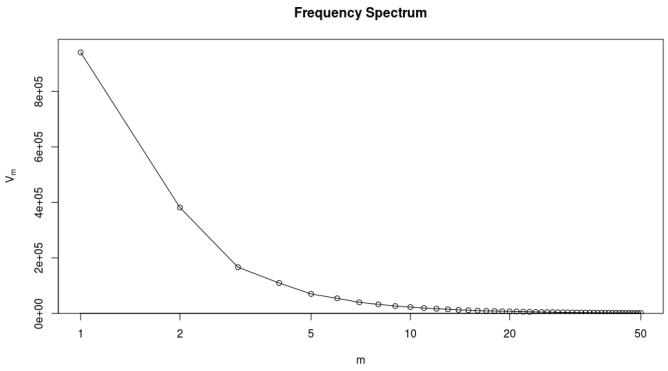

Fig. 2. Plot of the first 50 spectrum elements with the X axis on a logarithmic scale.

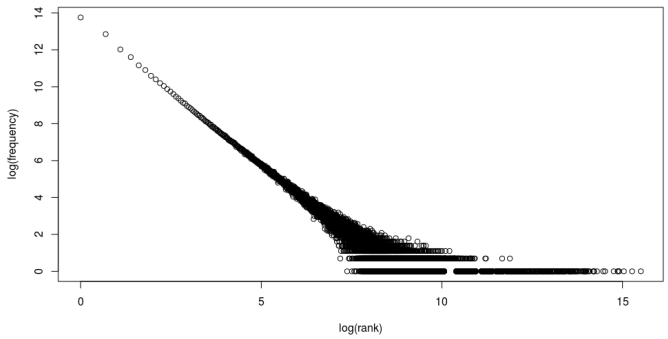

Fig. 3. Log-log scale plot of the distribution of frequency among the corpus.

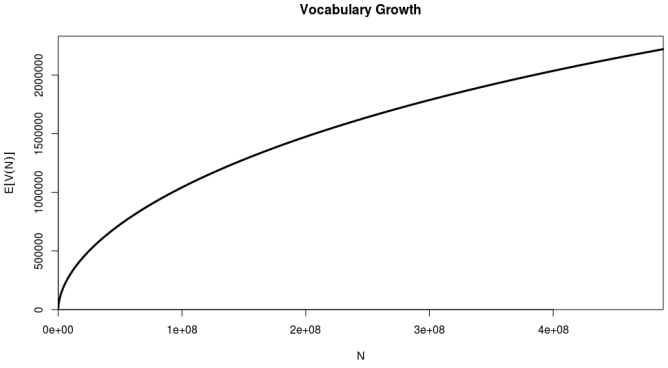

Fig. 4. Vocabulary Growth Curve(VGC) of the *N*-gram data.

IV. Language model and *N*-gram viewer

After collecting data and prepossessing it, we create our *N*-gram language model. We calculate frequency per day up to 5-gram. As the whole process is computationally expensive, we had to find and apply an elegant engineering solution. we use a comparatively fast approach using multithreading and NoSQL[3] rather than traditional approach using single thread and SQL. Later we index calculated

[3]https://www.mongodb.com/, Accessed on June 02, 2018

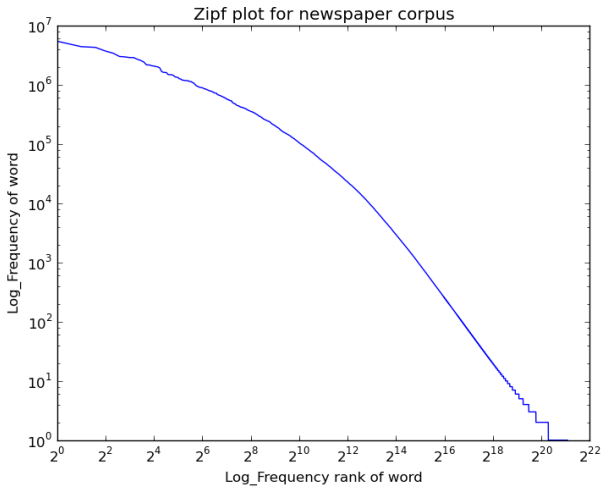

Fig. 5. Zipf's law between the rank and frequency of unique words in the corpus.

N-grams using solr[4] indexer for fast retrieval.

We also create a visualizer with an interactive graphical user interface(GUI)[5] where one can query and view the graph of N-gram frequencies of a particular N-gram across a given time frame. Also, one can compare the frequency distribution of multiple N-grams by simply querying comma separated N-grams. An example of such visualization is given in figure 6.

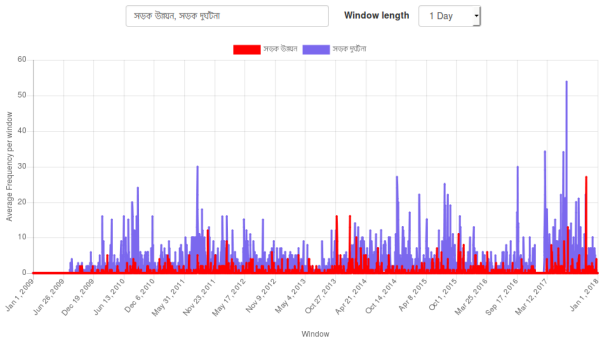

Fig. 6. N-gram viewer resulting graph of frequency per query.

V. Evaluation of the Model

There are two kinds of evaluation for language models, namely extrinsic and intrinsic evaluation. To evaluate Bengali N-gram language model, we show both kind of evaluation. For intrinsic evaluation, we calculate the perplexity of our model in a test set of previously unseen 36556 sentences containing 399271 words. For extrinsic evaluation, we perform two natural language processing tasks, namely context-aware spell checker and trending topic detection.

A. Perplexity

In information theory, perplexity is a measurement of how well a probability distribution or probability model predicts a sample. It is used to compare probability models. A low perplexity indicates the probability distribution is good at predicting the sample. Perplexity is,

$$Perplexity(C) = 2^{-\frac{1}{N}\sum_{i=1}^{m}\log p(s_i)} \quad (1)$$

where $p(s_i)$ is the N-gram probability of ith sentence, N is the number of words in test set, m is the number of sentences. Here, the log is base 2. We calculate perplexity for Unigram, Bigram and Trigram model for same test set with a corpus of 36556 sentences containing 399271 words. For Unigram probability calculation, we multiply the factor

$$\frac{1}{|vocabulary|}$$

with each probability as described here [6]. To deal with the probability of unknown words $p(unk)$, we used Laplace smoothing [7]. The results of perplexities from different year from year 2009 to 2017 as well as the overall perplexity of the model is given in Table I.

TABLE I
Perplexity of Unigram, Bigram and Trigram model accross different years.

Perplexity	Unigram	Bigram	Trigram
2009-2017	1425.30	180.53	11.01
2009	2093.63	128.49	5.30
2010	2079.77	133.99	5.94
2011	2069.72	134.95	5.93
2012	1505.37	144.37	6.60
2013	1202.32	146.34	6.67
2014	901.23	157.25	7.80
2015	896.60	153.40	7.72
2016	1473.90	113.63	5.38
2017	605.17*	173.07	8.92

[*This Unigram perplexity value is relatively low, because of 3 month's of data is missing in our dataset of year 2017]

There are many tasks in Natural language Processing where N-gram language model can be used either to perform a task or to improve any task. Here, we are giving two examples of such tasks, nemely context-aware spell checker and trending topic detection.

B. Context-aware Spell checker

N-gram model has been used to improve the result of spell checker. An introduction of N-gram based automatic spelling correction tool to improve information retrieval effectiveness can be found here [8]. In another approach [9] researchers applied memory-based learning techniques to the problem of correcting spelling mistakes in text using a very large database of token N-gram occurrences. They used web text as training data.

We use N-gram language model in order to develop a context-sensitive spell checking for Bengali. Most spell checkers only check words in isolation and determine whether they are spelled correctly. However, a large number of spelling errors does not involve non-words but rather valid words that are used in invalid places in sentences. Hence, spell checkers that only look at individual words are unable to detect many of the most common spelling mistakes. Such mistakes can only be detected by investigating word patterns, syntactic patterns, and collocations, making use of frequency information, and employing statistical models. In this experiment, we use our N-gram language model described in previous section in order to develop a context-sensitive spell checking for Bangali. This spell checker can be used as an independent unit to scan texts and detect errors as well as suggesting correct words.

To create a spell checker for Bengali, the first problem is to find a proper dictionary of the words that has been used in common texts. Creation of such dictionary may take a lot of manual labour as it

[4]lucene.apache.org/solr/, Accessed on June 02, 2018

[5]https://developers.pipilika.com/ngram, Accessed on June 02, 2018

requires digitalization of printed dictionary. We use an alternative method to create such dictionary automatically by a process, where we use unique Unigram of our *N*-gram model. To do so, we first retrieve the unique Unigram with total counts of one year. After that we remove any noise from the Unigram data.

As the Unigram are created using newspaper content, there are misspelled words. But we assume that, misspelled words frequency should be low. So, we use a frequency threshold and take Unigram appearing at least 30 times in the corpus. Thus, we get a list of 201749 words which we later use as a dictionary. One advantage of such dictionary is that, it contains contemporary words as well as popular names of persons, places and organizations which are not present in the traditional dictionaries.

There are two main steps for a successful spell checker. The first step is spell checking and the second step is spell suggestion. The spell checking process is straightforward; we consider a sentence and check each word of the sentence whether our dictionary contains that word or not. This kind of linear checking of errors is computationally costly, so we use an efficient algorithm combining Levenshtein distance and BKTree. Levenshtein distance is a string metric for measuring the difference between two sequences [11]. Informally, the Levenshtein distance between two words is the minimum number of single-character edits, insertions, deletions or substitutions required to change one word into the other. A BK-tree is a metric tree suggested in this paper [10]. BK-trees can be used for approximate string matching in a dictionary.

Levenshtein distance metric commonly used when building a BK-tree. Once the tree is created, we can efficiently search to see whether any word is in the dictionary or not. If the dictionary doesn't contain that word, it is considered as a wrong or misspelled word. After that, we proceed to the second step, which is spell suggestion.

For any misspelled word, we can retrieve word suggestions from BKTree within a certain edit distance. Once we get a list of words as suggestions, we use our *N*-gram model for context-awareness to narrow down the suggestion list and ranking final suggestions. For each word of the suggestion list we add the context words, Bigram or Trigram and query in our *N*-gram model for the frequency counts of co-occurrences. We remove the words having 0 count, which means the context and the word never occurred together in our model. After that, for the remaining words in suggestion list with frequency more than 0, we sort the suggestion list and use as final suggestion. For automatic spell correction, one can use the top word of the suggestion list.

We evaluate our spell checker on a dataset of 100 sentences containing a total of 923 words and among them 134 are misspelled. We manually prepare this dataset, where we also have the corresponding correct word for each misspelled word. Now we use that dataset to test our spell checker. We try to detect misspelled words for each sentence and predict the correct words and provide a list of short spell suggestions. We check whether our suggested words contains the corresponding correct word from the dataset. We calculate two separate F-score measures, one for only detection of the misspelled words and another is for correction. We only suggest words for the correctly detected misspelled words.

Following Table II represents the results of our context-aware spell checker.

TABLE II
CONTEXT-AWARE SPELL CHECKER EVALUATION RESULTS.

	Precision	Recall	F-score	Accuracy
Detection	.850	.805	.827	0.953
Detection+Correction	.836	.898	.866	0.968

C. Trending Topic Detection

We demonstrate another application using our Bengali *N*-gram model, which is is trending topic detection. Our *N*-gram model is based on newspaper data and it contains the frequency information of each *N*-grams per day. That enables the opportunity of the task trending topic detection. For simplicity, we will only demonstrate the Unigram and Bigram trending topic detection. But the process should be the same for any topic or *N*-gram.

To detect the trending topics, we use a statistical method called Chi-square test [12]. Chi-square test is a statistical hypothesis test assessing the goodness of fit between a set of observed values and those expected theoretically. The formula for the chi-square statistic used in the chi square test is:

$$\tilde{\chi}^2 = \sum_{k=1}^{n} \frac{(O_k - E_k)^2}{E_k}$$

Where O is observed value and E is expected value. Now consider a Unigram. In our case, the observed value for the Unigram is the average frequency of that Unigram in a window of 1 day for daily trending topic detection, 7 days for weekly and 30 days for monthly trending topic detection. The expected value is the overall average frequency of that particular *N*-gram in our model. In practice, we can limit the expected value window size for the past 7 days for daily trending topic detection, 30 for weekly and, 60 or 90 days for monthly trending topic detection.

According to the method described above, we calculate the chi-square values for all the *N*-grams up to 5 grams of a particular time frame of which we wish to detect the trending topics. We can ignore the summation part of the equation as its mathematically convenient. We sort the *N*-grams according to their chi-square values and return N number of entities with top chi-square values. To narrow down the results, also marge the topics describing same event by matching the strings.

As the evaluation of trending topic detection requires a list of valid trending topics, currently we do not have any authentic source of such list. So, we evaluate our trending topic detector against the national events like Victory day, Martyrs day and Independence day. We assume that, on these days, the trending topics of the newspaper should be related to these events. If our trending topic detector can successfully detect these topics as trending, we can conclude that our process is valid. Here are the results which represents the corresponding top 5 trending topics(weekly) of year 2015 in Table III.

Another graph shows the frequency spectrum of the top trending topics when queried in *N*-gram viewer in figure 7. This graph clearly indicates the bumping of the frequency that our model could detect as statistically significant.

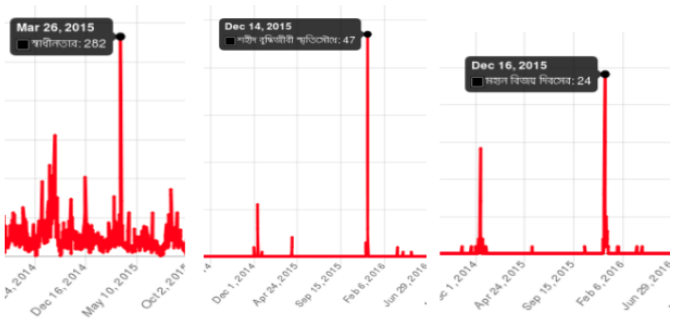

Fig. 7. *N*-gram viewer resulting graph of top trending topics.

TABLE III
TRENDING TOPIC DETECTION (TOP 5) RESULTS.

Independence day - 26 March, 2015
Victory day - 16 December, 2015
Martyrs day - 14 December, 2015

Independence day	স্বাধীনতার মহান স্বাধীনতা ও জাতীয় স্মৃতিসৌধে মহান স্বাধীনতা ও জাতীয় দিবস ঢাকা উত্তর সিটি করপোরেশনের ঢাকা দক্ষিণ সিটি করপোরেশনের
Victory day	মহান বিজয় দিবসের মহান বিজয় উৎসব মহান বিজয় র‍্যালি বিজয়ের এবারের বিজয় উৎসব
Martyrs day	শহীদ বুদ্ধিজীবী স্মৃতিসৌধে শহীদ বুদ্ধিজীবী দিবস শহীদ বুদ্ধিজীবীদের মনোনয়নপত্র প্রত্যাহার করেছেন ফাঁসির রায়

VI. CONCLUSION

In this paper, we present a large scale Bengali *N*-gram language model based on online newspaper data and show its application in Trending topic detection and Context-aware spell checker. We also present a *N*-gram viewer where one can query with a particular *N*-gram and and see the graphical presentation of its frequency over time. We evaluate our model in both intrinsic and extrinsic manner. An API is also released where one can query such data. We also provide a perplexity test of the model and evaluate our model in both intrinsic and extrinsic manner.

ACKNOWLEDGMENT

This work is partially funded by Access to Information (a2i) programme, which ran from the Prime Ministers Office of Bangladesh[5].

REFERENCES

[1] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. 2007. Large language models in machine translation. In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Language Learning, pages 858867

[2] C. Chelba and J. Schalkwyk, 2013. Empirical Exploration of Language Modeling for the google.com Query Stream as Applied to Mobile Voice Search, pages 197229. Springer, New York.

[3] D. Guthrie and M. Hepple. 2010. Storing the web in memory: Space efficient language models with constant time retrieval. In Proceedings of EMNLP 2010, Los Angeles, CA.

[4] Michel, J. B., Shen, Y. K., Aiden, A. P., Veres, A., Gray, M. K., Pickett, J. P., and Pinker, S. (2010). Quantitative analysis of culture using millions of digitized books. science, 1199644.

[5] Lin, Y., Michel, J. B., Aiden, E. L., Orwant, J., Brockman, W., and Petrov, S. (2012, July). Syntactic annotations for the google books ngram corpus. In Proceedings of the ACL 2012 system demonstrations (pp. 169-174). Association for Computational Linguistics.

[6] M. Federico, N. Bertoldi, and M. Cettolo. 2008. IRSTLM: an open source toolkit for handling large scale language models. In Proceedings of Inter-speech, Brisbane, Australia

[7] C.D. Manning, P. Raghavan and M. Schtze (2008). Introduction to Information Retrieval. Cambridge University Press, p. 260.

[8] Ahmed, F., Luca, E. W. D., and Nrnberger, A. (2009). Revised *N*-gram based automatic spelling correction tool to improve retrieval effectiveness. Polibits, (40), 39-48.

[9] Carlson, A., and Fette, I. (2007, December). Memory-based context-sensitive spelling correction at web scale. In Machine learning and applications, 2007. ICMLA 2007. sixth international conference on (pp. 166-171). IEEE.

[10] W. Burkhard and R. Keller. Some approaches to best-match file searching, CACM, 1973

[11] Levenshtein and Vladimir I. (February 1966). "Binary codes capable of correcting deletions, insertions, and reversals". Soviet Physics Doklady. 10 (8): 707710.

[12] Pearson, K. (1900). On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. Philosophical Magazine Series, 5, 50 (302): 157175.

[5]https://a2i.gov.bd/, Accessed on June 02, 2018