International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 09 | Sep 2021 www.irjet.net p-ISSN: 2395-0072

SPELL CORRECTION AND SUGGESTION USING LEVENSHTEIN DISTANCE

Ansh Mehta[1], Vishal Salgond[2], Darshan Satra[3], Nikhil Sharma[4]

[1-4]*Department of Information Technology, K. J. Somaiya College of Engineering, Mumbai*

Abstract - *In today's Data Science and AI community, NLP has become one of the most important topics for research as we move to a world where communication is becoming increasingly important between humans and machines as well. Spell Correction and suggestion have become an essential part of many applications. The applications are supposed to give us the right suggestions and answers even if we make a mistake in spelling or we use different sentences for the same meaning. Spell Correction is to find a word close to the word typed. There are many ways to achieve this but in this paper, we have demonstrated it using the Levenshtein distance. Levenshtein distance is a similarity measure used for finding similarity between 2 words. We have implemented movie suggestions by finding the words (movie names) that have a close distance from the word we typed.*

Key Words*: *Levenshtein Distance, Edit Distance, Data Science, AI, NLP, Spell Correction

1. INTRODUCTION

NLP has become the forefront of today's AI and ML research as the applications that utilize NLP become increasingly popular. The most common use case you will come across is spelling correction and auto-suggestions based on keywords. Almost every application utilizes this feature, the most prominent being the search engines.

The search engines use spell correction and recommendation to give you relevant search results even if your choice of words is different or the spelling is incorrect. Before NLP came along it was very difficult for any search engine to show relevant results as people often used different words and some spelled words incorrectly and hence the results were not relevant. It was clear that this technology was needed.

Levenshtein distance is an algorithm that calculates the similarity between two words based on the number of edits required to convert one word to another. The operation Levenshtein distance covers insertion, deletions, and substitutions. Levenshtein distance is a special case of the Edit distance where the weights of all operations are 1 unit.

Levantine distance algorithm is one of the most widely used algorithms for spell correction and word recommendation. In this paper, we use the Levenshtein distance on the movie dataset. We find the relevant movies based on the words the user has entered. The algorithm is implemented from scratch using python. As we will see in the results section the results produced by the Levenshtein distance are optimal and similar words are being detected. We have also validated some of our cut-offs and assumptions using Wikipedia spell correction dataset.

2. SIMILARITY MEASURES AND THEIR APPLICATIONS IN DIFFERENT FIELDS

NLP is one of the blooming and fastest-growing sectors of the IT industry. NLP involves processes such as voice-to-text conversion or text conversion from a language to another or word recommendation, etc. For all these operations, we need algorithms or methods to compare and verify the results on the basis of inputs. For finding the similarity between text words or phrases. Predominantly, similarity measures are used as a metric to find whether there is any relation between two words or to what extent are the two words similar. A few similarity measures are mentioned below:

a. Edit Distance
b. N-Gram Distance
c. Jaro Similarity
d. Cosine Similarity
e. Jaro- Wrinkler Similarity

The various application of these Similarity measures are:

a. Automatic Suggestion
b. Text Recommendation Systems
c. Plagiarism Checking
d. Gene Matching in Computational Biology
e. Automatic Answer Correction System (for MCQ question based on already known correct answers)
f. Automatic Spelling Correction System
g. Sequence Matching in Data Mining and Cyber Security

3. METHODOLOGY

For creating a spelling corrector or spelling checker, first it was important to restrict the domain of the input words to a particular field, else, a dictionary would be required for implementing the system in general English language. Hence, the domain of the system is restricted to Hollywood, English movie names. Further due to memory and hardware restrictions, the system is trained on names of movies

released in or after 2000. After the user inputs the movie name, the system checks the edit distance of the input and all the movie names.

3.1 Edit Distance Algorithm

Edit distance between 2 words is calculated by calculating the number of Edit operations (Insertion, Deletion and Replacement) carried out on one word to transform it into the second. Each operation is given some cost. Consider all the operations are of cost y.

Eg: Edit Distance between "FOOD" and "FODDER" is 3y as if we want to convert FOOD to FODDER, we replace the second 'O' with D and insert 'ER' which will cost y for replacement and 2y for 2 insertions making it 3y.

3.2 Levenshtein Distance Algorithm

In 1965, Soviet mathematician Vladimir Levenshtein formulated and explained Levenshtein Distance after which the distance metric was introduced and named after him. Many fields like spell checking, speech recognition, DNA analysis, and plagiarism detection use the Levenshtein Distance (LD) algorithm. The distance is the number of deletions, insertions, or substitutions required to transform into. Levenshtein Distance is a special version or case of Edit distance where y (cost) = 1

To restrict the output or results, we have used a measure of Distance limit (all words having Levenshtein Distance with the input less than or equal to Distance Limit will be shown).

These systems can be used in many video streaming or entertainment platforms and integrated with different search buttons and results in APIs.

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j)+1 \\ \operatorname{lev}_{a,b}(i,j-1)+1 \\ \operatorname{lev}_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Fig.-1: Levenshtein Distance Formula

3.3 Workflow and Steps involved

To Create a Word-Bank of Movies:

1) Import Required libraries
2) Read the "IMDb movies.csv" dataset. (Link provided in the reference section)
3) Filter all the movie names of the movies which are released after 2000 and are in English and store them in a list

For Word Correction System

1) Accept Input word from the user
2) Validate the input
3) Calculate the Levenshtein Distance between the input word and all the words available in the Word bank
4) Check if the word is already present in the Word Bank:
    a. If yes, the input word is correct
    b. Else, continue to step 3
5) Display all the words which have a Levenshtein Distance less than or equal to a threshold value (in our case 2)

4. IMPLEMENTATION

The system was implemented in Python and the editor used was Google Colab. Along with python, we used the dataset, "IMDb movies.csv", which was derived from Kaggle and contained the information of all the movies released after the 1860s.

4.1 Dataset Description and Preprocessing:

The Kaggle dataset (link in references) was populated with a huge number of entries. With these many rows and such a huge dataset, it was important to prioritize time taken and space complexities as basic Python data types were used to accomplish the task. We decided to reduce the number of movie names by only selecting the movies released after 2000, which reduced the size considerably.

Since the movies present in the database or dataset also included the movies in other languages, the data was cleaned by only considering the movies which were released post-2000 and were in English. This helped in restricting the data or input from the user and restricted our domain to some extent. The system only accepted movie names that had alphabets and spaces. We were finally left with around 13400 movies which were enough to train and implement our model.

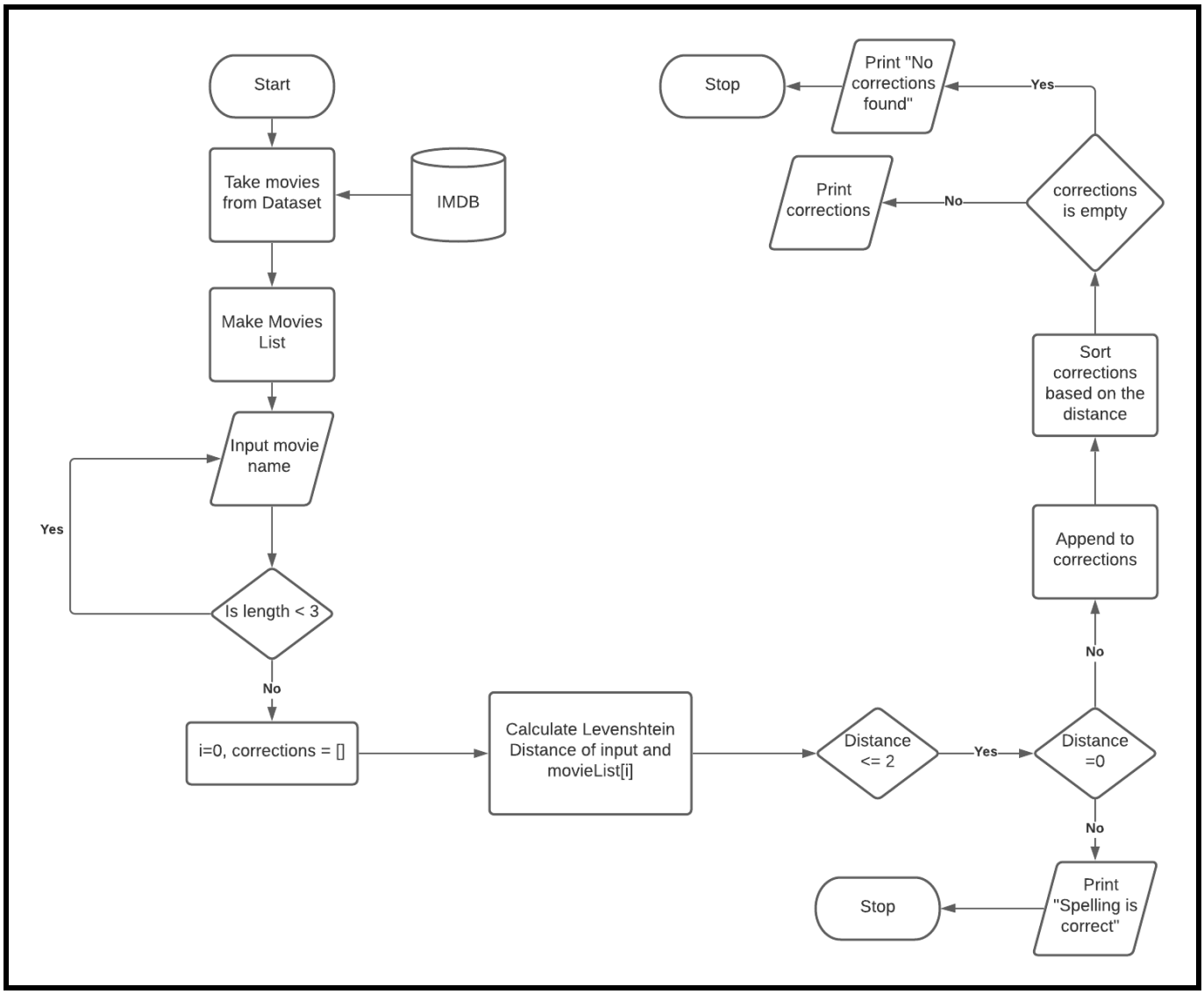

Fig.-2: System Workflow and Architecture

The user first inputs the movie name which is validated and accepted only if its length is greater than 3. Following this, the system calls the Autocorrection class which finds and returns all the movie names which can be derived or found from the input string by replacing some letters. In the Auto-Correction class, there are 2 functions namely, autocorrect and _LevenshteinDistance. The autocorrect validates input and finds the words which can be suggested to the user, while on the other hand, _LevenstheinDistance calculates the Levenshtein Distance and returns the distance.

Levenshtein Distance between all the 13400 words in the name-bank or word bank and input word is calculated and then the selected words or movie names that have Levenshtein Distance less than or equal to the Distance limit (threshold value) are printed or shown as the probable words.

For finding the Distance limit, we first loaded a dataset (txt file) of misspelled words in Wikipedia and created a dictionary that contains possible Levenshtein Distances as keys and the number of words with that distance from its Correct Spelling as the values.

While comparing the values, we saw that most of the misspelled words had a distance less than or equal to 2 (From their Correct Spelling). Keeping a higher threshold value (or distance limit) would result in incorrect suggestions or corrections. Hence, the distance limit was set to 2 and the words having distance less than or equal to 2 are printed. A special case appears when the distance is equal to 0, this means that there exists a movie with that text or word as the movie name, and hence no suggestions or corrections are recommended. The words are arranged in increasing order of the Levenshtein distance

5. OUTPUT OF THE SYSTEM

```
[116] autocorrect = AutoCorrection()
      autocorrect.correct("thro")

      ['troy', 'tre', 'thor', 'tko', 'torn', 'turbo', 'ehero', 'tri']
```

Fig.-3: Sample Output of the system

Understanding the output given in Fig.-3:

	-	T	H	R	O
-	0	1	2	3	4
T	1	0	1	2	3
R	2	1	1	1	2
O	3	2	2	2	1
Y	4	3	3	3	2

Fig.-4: Levenshtein Distance between "THRO" and "TROY"

Using the Levenshtein distance formula, Fig.-4 illustrates how the distance is calculated by the computer using a matrix or tabular notation.

The distance between Thro and Troy is 2.

Considering the example of "TRI":

	-	T	H	R	O
-	0	1	2	3	4
T	1	0	1	2	3
R	2	1	1	1	2
I	3	2	2	2	2

Fig.-5: Levenshtein Distance between "TRI" and "THRO"

The Levenshtein distance between "TRI" and "THRO" is 2, calculations for which can be referred from Fig.-5.

Here, both the words have Levenshtein distance equal to two and hence, the words are displayed according to the order in which they were stored in the dictionary. However, if the distances were different, the output would be arranged in the ascending order of the distance. For example: if the input is Spector

Levenshtein Distance (Spectre, Spector) = 2

Levenshtein Distance (Species, Spector) = 3

So, the output of the system, when input = "Spector" will be Spectre followed by Species as the distance between Spectre and Spector is less (2 < 3) and the user might mean to find or watch Spectre movie.

6. FUTURE SCOPE

The algorithm discussed and implemented in the paper was a naïve and basic algorithm and could lead to many more complex and advanced architectures. This system is currently designed only to correct or suggest corrections in Movie Names entered by a user on basis of a dataset of movies released after the year 2000. However, if this were to be implemented in a search engine, the dataset or database needs to be generalized and the barrier of movies released after 2000 needs to be removed. Also, for this system to be deployed on an entertainment platform, it needs to be made multi-lingual, where the user can get the freedom of searching his favorite movies in different languages, which is a complex task. A great efficiency is not the only objective, sometimes famous movies having superstars and international actors or singers or individuals of different domains are more searched than lesser known, low-budget films or documentaries. For an instance, when a user searches inter, Interstellar is shown first as a recommendation and the next movie shown is The Intern. This is because the former was a bigger, famous and acclaimed movie than the latter. To provide this, we can add weights to a movie according to its ratings, cast and popularity. Results can be arranged on the basis of this for better and satisfactory outcomes.

7. CONCLUSION

The paper discussed the importance of similarity measures and various types of similarity measures. Additionally, it also suggested different applications of similarity measures in various fields of human interests. The paper specified the importance of Spelling checking and its use in NLP. Along with this, the paper presented an efficient implementation of spell correction with Levenshtein distance with a unique algorithm which resulted in the Levenshtein distance correctly identifying the relevant words or movie names for any given input. It was also proved that using the cut off or threshold value equal to two is most optimal by validating it using Wikipedia spell correction dataset. Further, the paper mentions the future scopes that can be implemented in different projects or different methods which can make the system more efficient and useful.

REFERENCES

[1] Aouragh Si Loussain, Gueddah Hicham, Yousfi Abdellah, "Adapting the Levenshtein Distance to Contextual Spelling Correction" (via ResearchGate), March 2015

[2] Thi Thi Soe, Zarmi Sann, "Study on Spell Checking System using Levenshtein Distance Algorithm", International Journal of Recent Development in Engineering and Technology, Vol. 8 (Issue: 9), September 2019

[3] Muhammad Maulana Yulianto, Riza Arifudin, Alamsyah, "Autocomplete and Spell Checking Levenshtein Distance Algorithm to Getting Text Suggest Error Data Searching in Library", Scientific Journal of Informatics, Vol. 5, May 2018

[4] Maake Benard Magara, Sunday O. Ojo, Tranos Zuva, "A Comparative Analysis of Text Similarity Measures and Algorithms in Research Paper Recommender Systems", Conference on Information Communications Technology and Society (ICTAS), 2018

[5] Li Qu, H.X. Lin, "Edit Distance Based Crossover Operator in Gene Expression Programming", 8th International

Conference on Bio-Medical Engineering and Informatics (BMEI), 2015

[6] Yu Zhao, Hui Xing Jiang, XiaoJie Wang, "Minimum Edit Distance-based text matching Algorithm", 6th International Conference on Natural Language Processing and Knowledge Engineering (NLPKE), 2010

[7] Doru Anastasiu Popescu; Ovidiu Domşa; Ion Alexandru Popescu,"Determining the degree of similarity of answers given at a test using edit distance"

[8] Muhammad Ifte Khairul Islam , Rahnuma Islam Meem , Faisal Bin Abul Kasem, Aniruddha Rakshit, Md. Tarek Habib, "Bangla Spell Checking and Correction Using Edit Distance", 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), 2019

[9] Myunggwon Hwang, Do-Heon Jeong, Seungwoo Lee, Hanmin Jung, "Measuring Similarities between Technical Terms based on Wikipedia", International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing, 2011

[10] Dataset for implementation:
https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset