Evaluating Autocomplete Systems: *Performance of Word Embedding vs. N-grams Techniques Using a Wikipedia Dataset*

Shalini Tigga
School of Computer Science
Lovely Professional University
Phagwara, India
shalinitigga26@gmail.com

Aluri Veerabrahmam
School of Computer Science
Lovely Professional University
Phagwara, India
veerabrahmamaluri@gmail.com

Kirti
School of Computer Science
Lovely Professional University
Phagwara, India
kirti.29458@lpu.co.in

***Abstract*—Autocompletion is a part of natural language processing, which gives statistical query completions based on user input and offers suitable suggestions for the completion of a task to make interaction easy and smooth. It saves a user from any typos, reduces keystrokes, and makes interaction better with the system as it tries to predict the next most likely word or phrase that the user may input. This paper presents a comparison of autocomplete systems, focusing on Word Embedding and N-grams model techniques, utilizing a comprehensive Wikipedia dataset for evaluation. Our findings indicate that the N-grams model outperformed other techniques, achieving an impressive accuracy of 92.4%. A detailed quantitative evaluation of the systems is provided, highlighting their performance and effectiveness in practical applications.**

Keywords—Autocompletion, N-grams, Word Embedding

I. INTRODUCTION

Autocomplete systems have become crucial in a wide variety of natural language processing applications, such as general search engines or messaging platforms, word processors and document editors, electronic typewriters, and many other types of interfaces. These systems enhance the effectiveness of user interactions and improve the overall user experience by forecasting probable completions for partially typed queries or phrases. Autocomplete is particularly useful for mobile devices, where input methods are often limited, as it minimizes keystrokes, corrects typographical errors, and provides contextually relevant suggestions.

At the heart of autocomplete functionality are various predictive models that utilize different approaches to language understanding and generation. Simple statistical models, like n-grams, which forecast the subsequent word based on the previous sequence, were the foundation of early autocomplete systems. Although effective to some degree, n-gram models suffer from a lackofcontext awareness, which serious ly poses a problem whenincreasing wordsequences. However , that limitation has led on to even more sophisticated archite cture structures, such as those presented by recurrent neural networks, LSTM networks, and finally the Transformer-based architectures more recently developed with even more extensive contextual understanding.

With these developments in model architectures, the addition of Word2Vec and GloVe word embeddings has made autocomplete considerably more impa ctful, specially at the semantic level of relationship between words. Word embeddings enhance prediction accuracy across various scenarios by allowing algorithms to comprehend not only word frequency but also word meaning and relationships.

This paper provides a comparison between word embedding and n-gram techniques used in autocomplete systems, utilizing a comprehensive Wikipedia dataset for evaluation. Finally, we determine which technique demonstrates superior performance.

II. BACKGROUND

A. Autocomplete Systems

Autocomplete systems is a predictive system that takes partial user input and generates likely completions, enabling faster and more efficient text entry [1]. Autocomplete algorithms reduce the amount of human typing by suggesting complete words, phrases, or even sentences based on the user's beginning characters or words. This feature is particularly valuable in applications requiring frequent text input, such as search engines, messaging platforms, and data entry tools, where it can enhance productivity, reduce errors, and improve user experience.

At its core, autocomplete leverages advanced natural language processing techniques to predict what the user intends to write. Early methods used basic statistical approaches, such as frequency-based n-grams, which relied on historical patterns to suggest likely next words. However, more recent methods use advanced machine learning models, such as word embeddings, which more accurately represent language's syntax, semantics, and context. These developments make autocomplete systems effective tools for helping users across a variety of technologies by enabling them to make contextually appropriate predictions even with little initial input.

B. N-grams Lamguage Models

An N-gram means a sequence of N-words on the modeling of NLP. The N-gram model predicts those words that most likely follow some given sequence consisting of N-1 previous words. The model helps in speech recognition, machine translation, and autocomplete systems, typically generated through copious textual data, thus acquiring knowledge about the probability distributions of the sequences of word involvement. An N-gram language model is essentially about learning probability distributions over sequences of words. [2].

1) Unigram (1-gram): In a unigram model, each word is considered independently, with no dependence on previous words. The probability of a word is based solely on its frequency in the corpus [3].

$$P(w_i) = \frac{count(w_i)}{Total\ number\ of\ words} \quad (1)$$

In (1), w_i represents the current word we are predicting the probability for.

2) Bigram (2-gram): In a bigram model, the probability of a word depends only on the immediately preceding word. This allows the model to capture simple word-to-word dependencies.

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})} \quad (2)$$

In (2), w_{i-1} represents the immediately preceding word (one word before w_i and $P(w_i|w_{i-1})$ indicates that we predict the probability of w_i based on only the previous word w_{i-1}.

3) Trigram (3-gram): In a trigram model, the probability of a word depends on the two preceding words, enabling the model to capture short-range dependencies and context.

$$P(w_i|w_{i-2}, w_{i-1}) = \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2}, w_{i-1})} \quad (3)$$

In (3), w_{i-2} represents the second word before w_i and $P(w_i|w_{i-2},w_{i-1})$ considers the two preceding words w_{i-2} and w_{i-1} when predicting w_i.

Advantages of N-grams language models:

- Helps in simplifying text data for analysis by breaking down complex sentence structures into smaller, more manageable parts.
- Provides context to words by considering the preceding and succeeding words, making language prediction and translation more accurate.
- Increases text analysis pattern recognition, which raises the precision of text-to-speech (TTS), sentiment analysis, and tagging systems.

Disadvantages of N-grams language models:

- *Storage limitations:* As the value of 'n' increases, the number of possible N-grams grows exponentially, which could pose a significant storage challenge.
- *Data sparsity:* It is less likely to detect repeated instances of the same sequence with bigger N-grams, which results in sparse data.
- *Lack of semantic understanding:* While N-grams are good at recognizing patterns, they lack the understanding of context beyond the sequences they were trained on [4].

C. Word Embedding

It is a word embedding technique used for representing the words and documents in a new numeric vector within a much lower-dimensional space. This facilitates words that convey similar meaning to be represented similarly to enable machine learning models that work well with text data by extracting syntactic and semantic features.

1) Word2Vec: Word2Vec, introduced by Tomas Mikolov and others in Google in 2013 [5], is a neural approach for generating word embeddings. It belongs to the family of neural word embedding techniques and represents words as continuous vector spaces. The main idea behind Word2Vec is that words with similar meanings should have similar vector representations. Two primary methods are used to create embeddings in Word2Vec:

a) Continuous Bag-of-Words (CBOW): CBOW predicts the target word given its surrounding context words.

b) Skip-Gram: Skip-Gram is a model that predicts the context words given a target word.

2) GloVe (Global Vectors for Word Representations): GloVe is a word-embedding method that learns from global word co-occurrence statistics and was introduced by Pennington et al. in 2014. It serves as a method to get the 'global context' by examining how often words co-occur within a corpus. GloVe calculates the co-occurrence matrix, where adjacent words get 1, words one word apart will be valued at 1/2, two words apart 1/3, and so on and so forth. It is this method that enables GloVe to encode words based on their general meaning and relationship across the corpus.

Advantages of Word Embeddings:

- It is faster to train than graph-based systems like WordNet.
- Almost all modern NLP applications start with an embedding layer.
- It is available to store an approximation of meaning.

Disadvantages of Word Embeddings:

- High memory utilization.
- Corpus-dependent; in other words, if the data is biased in some way, that would, generally speaking, reflect in your model.
- Cannot recognize homophones. Eg: brake/break, cell/sell, weather/whether etc. [6].

III. Literature Review

Automated word prediction enhances typing for users with disabilities by reducing effort and errors. Using N-gram models, which predict words based on prior sequences, has proven effective across languages. Eunbin Ha et al. [7] and similar studies highlight the model's adaptability, employing algorithms like Viterbi for improved accuracy. This study applies the N-gram approach to Nepali, aiming to enhance prediction precision in Nepali text.

Query auto-completion (QAC) systems traditionally rely on past search logs, ranking suggestions based on popularity, but fail with rare prefixes due to limited historical data. Bhaskar Mitra et al. [8] address this by proposing a QAC design that generates synthetic suggestions using frequent suffixes from historical logs, complemented by a supervised ranking model. Using n-gram statistics and a convolutional latent semantic model (CLSM), this approach integrates both popular and rare queries, enhancing performance significantly over standard popularity-based methods.

Tab-complete features enhance user experience by predicting subsequent words based on initial text, benefiting areas like email composition and report writing. Steffen Bickel et al. [9] evaluate this function by developing a metric and adapting N-gram models to predict words across call-center emails, personal emails, weather reports, and recipes. Empirical results using an instance-based baseline demonstrate the model's effectiveness in improving predictability within these varied applications.

Note-taking during lectures, especially on sensitive topics like Quranic verses, can be challenging due to varied lecture delivery styles and the need for precision. Rian Adam Rajagede et al. [10] address these challenges by developing an autocomplete system for the Indonesian translation of the Quran to assist in accurate note-taking. Leveraging FastText and cosine similarity, the system retrieves semantically similar verses, even handling typographical errors. Evaluation shows promising accuracy, achieving 70.59% for the top 5 and 76.47% for the top 10 retrieved verses, supporting efficient and precise note-taking.

Query auto-completion is essential in search engines, aiding users in query formulation and improving search relevance. Dandagi et al. [11] propose a graph-based machine learning approach, leveraging the Node2vec algorithm—adapted from Word2vec—to represent node features in graphs, enhancing query suggestions. Using a supervised LSTM-based Recurrent Neural Network, the model achieves 89% accuracy, demonstrating significant improvement in query auto-completion performance.

IV. METHODOLOGY

A. Dataset Description

This dataset, which was taken from Wikipedia, was initially in DOCX format and has 101,190 words in total. When converted to a DataFrame, it has 1,225 rows and 1 column, with special characters indicating incomplete or redacted text in each record. The text consists of excerpts related to notable movies, celebrities, and other significant cultural references.

B. Technique Used

1) N-gram Model: We used a trigram model to capture sequences of three words, which helps in understanding contextual relationships between words in a localized manner. This approach enhances the model's understanding of phrases and word patterns.

2) Word-Embedding(Word2Vec): We used Word2Vec to create vector representations of words based on their context. The parameters that were used were:

*a) vector_size:*To Set 250, this means each word of vocabulary is represented by a 250-dimensional vector.

*b) window:*It means in case of 5 words taking words i n five positions to left and right of target word as part of the context.

c) sg: Set to 1, which uses the skip-gram model.

d) epochs: Set to 50, indicating the number of iterations over the dataset, which can help improve model convergence.

e) min_count: Set to 1, ensuring that all words, even those appearing only once, are included in the vocabulary.

f) workers: Set to 4, utilizing four worker threads to speed up training through parallel processing.

C. Data Preprocessing

The dataset was pre-processed in a number of ways to improve its consistency and quality:

1) Removal of Special Characters and Non-English Alphabets: All special characters (e.g., @, <) and non-English alphabets were eliminated to focus solely on relevant textual data.

2) Lowercasing: All text was converted to lowercase to reduce variability and ensure consistency across the dataset.

3) Trimming Whitespace: Leading and trailing whitespace was removed, along with any empty sentences, to streamline the text for analysis.

4) Stop Words Removal: Common stop words (e.g., a, an, in) were removed to focus on the more meaningful words within the text.

5) Lemmatization: This text preprocessing technique was applied to reduce different forms of a word to its base form, or lemma, thereby simplifying the analysis.

D. Train-Test Split

Training set-test split is used as a benchmark for a performance measure. The task is splitting some dataset into both a training and a testing subset of it, considered to be independent but with the current task relating more towards identifying a better model against specified measures on the t esting dataset of your own. Which means: how well does your model perform with new information, which i t's never been presented with.

We split our dataset into the training and test datasets at a 0.20 ratio, where the test dataset constitutes 20% and the training dataset forms 80%. This ratio ensures that an adequate amount of data is provided for training the model, while a sufficient portion is retained for the performance evaluation of the trained model over previously unseen data.

E. Validation and Evaluation

1) Accuracy: This metric defines the ability of the intended word prediction by the model through a test set. The percentage of the right prediction of words over the whole set of words in test data is calculated. High accuracy indicates that the model makes reliable predictions, closely matching the expected outputs.

*2) AvgKeystrokesSaved:*This is actually the word length metric used to measure how wella model is performing with its average word length. Let' s assume we are working with a list of words, or phrases; hence,themodel would eventually reduce keystrokes for part icular words or phrases users would be able to think about, saving a little amount of time and much more energy. A higher valuein keystrokesavings represents the capability of a model in returning useful predictions and hence, does not need too much keystrokes from the side of the user.

3) Avg Completion Time: This measures the average time the model takes to complete its predictions, reflecting its speed and responsiveness. A shorter completion time suggests that the model can generate predictions quickly, enhancing user experience by minimizing waiting periods during interactions.

4) Context-Capture Score: This is a qualitative score ranging from 1 to 3 that indicates the model's ability to understand and retain context. A score of 1 means the model captures context poorly, while a score of 3 suggests it

maintains strong contextual relevance, offering predictions that align closely with the overall meaning or intent of the input.

V. Discussion

TABLE I. Performance Results

Model	Metrics			
	Accuracy	***Avg. Keystrokes used***	***Avg. Completion Time***	***Context-Capture Score***
N-gram	0.924	6.20	0.016	1
Word2Vec	0.096	7.19	2.22	2

Table I compares the N-gram and Word2Vec models across four key performance metrics, with the following results:

1) Accuracy: The N-gram model has an accuracy of 0.9246, indicating it predicts words correctly at 92.46%, which is quite high. The accuracy rates for the Word2Vec models are much lower, at 0.0968 (9.68%), suggesting that it does not do a very good job of matching up words in the test set.

2) Avg Keystrokes Saved: Word2Vec saves slightly more keystrokes on average (7.197343) compared to N-gram (6.201509). This suggests that although Word2Vec is less accurate, when it does predict correctly, its suggestions tend to be longer and potentially more efficient in terms of typing reduction.

3) Avg Completion Time: N-gram is much faster, with an average completion time of 0.016903 seconds, making it highly responsive. Word2Vec, however, has a significantly longer average completion time of 2.220603 seconds, meaning predictions take longer to generate and may feel slow in real-time use.

4) Context-Capture Score: The Context-Capture Score for N-gram is 1, indicating it has minimal ability to understand broader context, leading to simpler, more literal predictions. Word2Vec scores 2, showing it has a better, though still moderate, ability to capture context and provide contextually relevant predictions.

VI. Future Scope

Future improvements and additions to autocomplete systems could focus on several key areas. One significant avenue for research is the integration of advanced neural network architectures, such as Transformers or attention-based models, with existing n-gram and word embedding techniques. This integration could help assess whether combining these approaches enhances prediction accuracy and contextual understanding.

Additionally, expanding the research to include multilingual autocomplete systems could provide valuable insights into how well different models perform across various languages. This could enhance the adaptability and inclusiveness of autocomplete functionalities.

Another important area for exploration is the optimization of real-time performance and response times of autocomplete systems, particularly in mobile applications where resource constraints are a concern. By investigating methods to improve efficiency, developers can ensure smoother user interactions.

Finally, further research should assess the robustness of autocomplete systems in the presence of noisy inputs or errors. This involves examining how different models handle misspellings, abbreviations, and informal language, ultimately leading to more resilient and user-friendly systems. Collectively, these improvements could significantly advance the capabilities and user experience of autocomplete technology in natural language processing.

VII. Conclusion

In this paper, we compared the N-gram model and Word2Vec and found that N-gram outperformed Word2Vec on our dataset. Although Word2Vec achieved slightly better results in average keystrokes saved and context-capture score, N-gram compensated for this with much higher accuracy and significantly faster completion time.

Additionally, the insights gained from our study can serve as a valuable reference for researchers and practitioners in selecting techniques for similar datasets. Our findings provide guidance for choosing autocompletion techniques, especially when working with datasets that share characteristics similar to ours.

In conclusion, autocompletion using N-grams can reduce keystrokes and enhance user experience by accurately predicting the most likely next word or phrase. phrase.

VIII. References

[1] Techopedia, "What is Autocomplete?" [Online]. [Accessed: 30-Oct-2024].

[2] M. Kavlakoglu, "N-gram Language Modeling in Natural Language Processing," *KDnuggets*, Jun. 2022. [Online]. [Accessed: 3q-Oct-2024].

[3] D. Jurafsky and J. H. Martin, *Speech and Language Processing*. Draft, Aug. 20, 2024. Copyright © 2024.

[4] Dremio, "N-grams in NLP," *Dremio Wiki*. [Online]. [Accessed: 30-Oct-2024].

[5] IBM, "Word Embeddings," *IBM*, [Online]. [Accessed: 30-Oct-2024].

[6] GeeksforGeeks, "Word Embeddings in NLP," *GeeksforGeeks*. [Online]. [Accessed: 30-Oct-2024].

[7] Khadka, B.R., "The use of N-gram language model in predicting Nepali words", *Prithvi Academic Journal*. [Online]. [Accessed: 02 November 2024].

[8] B. Mitra and N. Craswell, "Query Auto-Completion for Rare Prefixes," in *Proc. 24th ACM Int. Conf. on Information and Knowledge Management (CIKM '15)*, New York, NY, USA, 2015, pp. 1755–1758.

[9] Bickel, S., Haider, P. & Scheffer, T. (2005). Predicting Sentences using N-Gram language models. Proceeding of Conference on Empirical Methods in Natural Language Processing, HLT'05, 193–200, Stroudsburg, PA, USA. Association for Computational Linguistics.

[10] R. Rajagede, K. Haryono, and R. Qardafil, "Semantic Retrieval for Indonesian Quran Autocompletion," *Jordanian Journal of Computers and Information Technology (JJCIT)*, vol. 9, pp. 94–106, 2023

[11] V. S. Dandagi and N. Sidnal, "Auto-Completion of Queries," in *Lecture Notes on Data Engineering and Communications Technologies*, vol. 57, pp. 435–446. Springer, 2021.