iCTSM

Proceedings

of

3rd International Conference on Trends in Sustainable Computing and Machine Intelligence (ICTSM 2025)

19-20, September 2025 | Bangkok, Thailand

Springer

Organized by
Stamford International University
Bangkok, Thailand

Table of Contents

Manuscript Title & Author(s)

N-gram language model for Uzbek texts

*Elov Botir Boltayevich[1],
Tojieva Gulbahor Nomozovna[2],
Tokhtaeva Matluba Xakim qizi[3],
Jurayeva Nargiza Jamolidinovna[4] and Primova Mastura Hakim qizi[5]

[1]Department of computational linguistics and digital technologies,
Tashkent State University of Uzbek Language and Literature named after Alisher Navoi, Tashkent, Uzbekistan
[1*]elov@navoiy-uni.uz,
[2]Uzbek Linguistics department, Qarshi State University, Uzbekistan
[2]tojiyevagulbakhor@gmail.com,
[3]Department of Humanities University of Management and Future Technologies, University of management and future technologies
Tashkent, Uzbekistan
[3]tuxtaeva@umft.uz,
[4]Department of the Psychology of Religion and Pedagogy, International Islamic Academy of Uzbekistan
[4]nargiza20002003@gmail.com,
[5]Department of computational linguistics and digital technologies,
Tashkent State University of Uzbek Language and Literature named after Alisher Navoi, Tashkent, Uzbekistan
[5]primovamastura@navoiy-uni.uz

* primovamastura@navoiy-uni.uz

Abstract. This article presents language models, specifically the statistical method known as the N-gram model. Language models are mathematical or computational models that probabilistically represent sequences of words in natural languages (for example, Uzbek, English). They are used to predict the next word based on a given context, generate text, assess word probabilities or analyze language-specific statistical features. The following smoothing methods used in the N-gram model are discussed: Laplace smoothing, Good-Turing discounting, Katz back-off, Interpolation, and Kneser-Ney smoothing. The article presents the stages of building an n-gram language model for Uzbek texts: corpus preparation, N-gram formation, smoothing, language model format preparation, and prediction mechanisms. The corpus frequencies, i.e., the distribution of N-gram counts, are presented both in logarithmic scale and in linear graph form. As a result, when applying Kneser–Ney smoothing for the Uzbek language, the test perplexity of the 3-

gram model was 121, for the 5-gram model it was 114, for the 7-gram model it was 112, and for the 9-gram model it was 108.

Keywords: NLP, Smoothing, Kneser-Ney, IRSTLM, KenLM.

1 Introduction

The Markov Assumption and the N-gram: The N-gram approach is among the most fundamental statistical techniques utilized in language modeling. When estimating the likelihood of word sequences in a text, an N-gram is a sequence of N consecutive items, usually words. According to the Markov assumption, predicting the next word in a sequence just requires taking into account the final (N−1) words, not the complete context that came before it. A word in a bigram (2-gram) model, for example, is thought to rely solely on the word that comes before it. Generally speaking, an N-gram model is used to determine the likelihood of a sentence made up of N words as follows:

$$P(w_1, w_2 \dots w_N) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2}) \dots P(w_n|w_{1:n-1})$$
$$= \prod_{k=1}^{N} P(w_k|w_{1:k-1})$$

Here, $\boldsymbol{w_i}$ denotes the i-th word in a sentence, and N represents the total number of words in the sequence.

The term $\boldsymbol{P(w_i|w_{i-n+1:i-1})}$refers to the conditional probability of the word $\boldsymbol{w_i}$ occurring after the preceding (N-1) words.

In N-gram language models, such conditional probabilities are estimated based on word frequency counts from a training corpus using the Maximum Likelihood Estimation (MLE) method:

$$P(w_i|w_{i-n+1:i-1}) \approx \frac{C(w_{i-n+1}, \dots, w_{i-1}, w_i)}{C(w_{i-n+1}, \dots, w_{i-1})}$$

Here, the operator $C(\cdot)$ denotes the frequency count of a given sequence of words as it appears in the corpus.

For example, in an Uzbek language corpus, the probability of the bigram “juda ajoyib” is calculated using the following formula:

$$P(\text{“}juda\text{”} \mid \text{“}ajoyib\text{”}) = C(\text{“}juda\ ajoyib\text{”}) \,/\, C(\text{“}juda\text{”})$$

If a sequence such as “juda ajoyib” does not occur even once in the training corpus, then C(“juda ajoyib”) = 0 , and the model assigns a probability of zero to this context.

This leads to the zero-probability problem in language modeling — meaning that any sequence of words not observed in the corpus is treated by the model as impossible (i.e., having zero probability), even though such sequences may naturally occur in real language use.

Conducted Scientific Research

In recent years, there has been extensive global research focused on language corpora and probabilistic language models. The concept of analyzing linguistic phenomena as a probabilistic model was first introduced by Claude Shannon [1] within the framework of information theory, laying the foundation for future studies in the field.
For example, Michel et al. [2] conducted large-scale analysis of textual data from the Google Books N-gram corpus, covering texts from books published between the 1500s and 2000s. This approach demonstrated the potential of using statistical methods to trace linguistic and cultural changes over centuries.

The agglutinative nature of the Uzbek language, characterized by its rich system of affixation, introduces unique challenges when applying standard N-gram modeling techniques. Nevertheless, the development of initial language models for Uzbek has already begun to gain scholarly attention. Notably, in 2024, B. Elov and co-authors published studies focused on building and evaluating N-gram models for Uzbek. Their work involved the construction of unigram, bigram, and trigram models based on corpus data, followed by evaluation using metrics such as average log-probability (perplexity) and other performance indicators.

Specialized Smoothing Techniques in N-gram Models

Smoothing Techniques

To address the issue of zero probabilities and rare events in N-gram models, various specialized smoothing techniques are applied. The primary goal of smoothing is to assign small, non-zero probabilities to unseen word sequences while slightly reducing the probabilities of those that do appear in the corpus - thereby balancing the overall probability distribution.
The following are some commonly used smoothing methods in N-gram language modeling:

- *Laplace (Add-One) Smoothing*: This method increases every observed frequency count by 1, ensuring that no N-gram has a zero probability[3].

- *Good–Turing Discounting*: Adjusts the probabilities of low-frequency events downward in order to reserve probability mass for unseen N-grams[4-5].

- *Katz Back-Off*: Uses lower-order N-gram probabilities when higher-order sequences are not observed in the corpus, effectively backing off to more reliable estimates.

- *Interpolation*: Combines probabilities from N-gram models of different orders using weighted coefficients (λ) for each level of context, rather than backing

off entirely.

▪ *Kneser–Ney Smoothing*: Considered one of the most effective smoothing techniques, it focuses on estimating how likely a word is to appear in a novel context, rather than its absolute frequency alone.

The simplest smoothing technique is **Laplace (add-one) smoothing**, which eliminates zero frequencies by adding 1 to each observed count. However, this method can **artificially inflate probabilities**, especially in corpora with a large vocabulary size. Due to this limitation, more effective smoothing techniques are preferred in practice, such as those listed below [6-7]:

1. **Good–Turing Discounting:** Originally developed in statistics to adjust low-frequency counts, this method redistributes some probability mass from frequently observed N-grams to those that were not seen in the training corpus. The idea is to slightly reduce the estimated probabilities of N-grams that occur 1, 2, 3, etc., times, and reallocate that mass to unseen sequences. The Good–Turing formula estimates the probability of unseen N-grams based on their expected frequency [8]. This approach was later adopted in language modeling by Katz [9].

2. **Katz Back-Off Model:** In this approach, if a higher-order N-gram (e.g., trigram) is not observed in the corpus, the model automatically "backs off" to a lower-order N-gram (e.g., bigram) to estimate the probability for that context.
At the same time, the probability assigned to observed N-grams is discounted, and the remaining probability mass is redistributed to support the lower-order estimates during back-off. Katz smoothing relies on Good–Turing estimates for computing the discounting coefficients[10].

3. **Interpolation:** In this method, for each possible context, the probabilities from higher- and lower-order models are combined using specific weighting factors $\boldsymbol{\lambda}$ values.
For example, in a three-word context:

$\boldsymbol{P_3}$ is the probability from the trigram model,

$\boldsymbol{P_2}$ is the probability from the bigram model,

$\boldsymbol{P_1}$ is the probability from the unigram model, then the final interpolated estimate is computed as:

$$\boldsymbol{\lambda_3 \cdot P_3 \ + \ \lambda_2 \cdot P_2 \ + \ \lambda_1 \cdot P_1}$$

Here, the $\boldsymbol{\lambda_i}$ coefficients must satisfy the following condition:

$$\lambda_3 + \lambda_2 + \lambda_1 = 1$$

Interpolation mitigates the zero-probability problem and combines the strengths of models of different orders.

4. **Kneser–Ney Smoothing:** Kneser–Ney smoothing is considered one of the most effective smoothing techniques to date. In this method, a fixed discount value d (usually selected such that 0<d<1) is subtracted from the count of each observed N-gram, and the leftover probability mass is redistributed to lower-order models. Its key innovation lies in the use of a special continuation probability in the lower-order model, which accounts for the likelihood of a word appearing in novel contexts. This leads to more accurate estimates, particularly for infrequent words. The Kneser–Ney algorithm has been shown to yield superior results in many language modeling tasks, especially in automatic speech recognition (ASR) systems[11].

Although N-gram models rely on relatively simple mechanisms such as word counters and probability formulas, building and applying them effectively requires solid technological foundations. This paper provides a detailed overview of the technical solutions and development stages involved in constructing the N-gram language model.
In particular, it analyzes the modeling process and results using specialized toolkits like IRSTLM and KenLM, as well as custom Python scripts.

Stages of building the n-gram model (irstlm, kenlm)

The development of an N-gram language model is carried out through several stages. As shown in Figure 1, the workflow was performed step by step using IRSTLM, KenLM, and specially developed Python scripts.

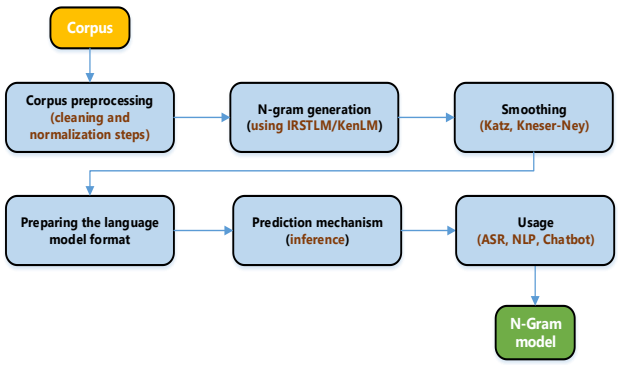

Fig.1. Stages of building an n-gram language model for Uzbek texts

Data Preprocessing. Before training the model, a large-scale text corpus was collected and cleaned. In this stage, unnecessary elements were removed and the text was normalized. For example, HTML tags, improperly encoded characters, extra spaces, and punctuation were cleaned, and all letters were converted to lowercase. To model sentence boundaries, a special <s> token was added at the beginning and a </s> token at the end of each sentence. Additionally, rare or non-standard words (e.g., digit sequences or extremely infrequent names) were replaced with the <unk> (unknown) token [12]. This technique helps reduce the vocabulary size and minimize the negative impact of rare words on the model's performance. As a result, the cleaned and normalized texts were prepared and used for model training.

N-gram Extraction and Vocabulary Construction. After the preprocessing stage, the next key technical step in building the model is extracting N-gram statistics from the cleaned text corpus. In this stage, the frequencies (occurrence counts) of all sequential N-grams-from unigrams (1-gram) up to the desired N level-were computed. Given the large size of the corpus, the number of possible N-grams can also be extremely high. For instance, if the vocabulary size of the corpus is V, then theoretically, for a trigram (N=3) model, up to V^3 combinations could exist-an enormous number. In practice, however, only a fraction of these N-grams actually occurs in the corpus, though the number may still reach into the millions. Therefore, to efficiently perform this computation, specialized and optimized toolkits such as IRSTLM and KenLM were used.

The IRSTLM Toolkit is designed to optimize resource usage when building language models from large corpora. It features a script called build-lm.sh, which enables the corpus to be automatically divided into smaller parts for processing. Specifically, the IRSTLM script can split the text into k chunks (e.g., k = 10), compute N-gram counts for each chunk separately, and then merge the results. While this method may slightly increase processing time, it significantly reduces memory usage. For example, constructing a 3-gram model in IRSTLM can be done using the following command:

build-lm.sh -i "corpus.txt" -n 3 -o model.arpabo -k 10

(Here, -k 10 specifies that the corpus should be divided into 10 parts).

In contrast, the KenLM toolkit is focused on achieving maximum speed and efficiency. In KenLM, the lmplz utility takes the corpus as input and counts N-grams using an internal on-disk sorting algorithm. KenLM also allows users to specify the maximum amount of memory to be used-either as a percentage or in bytes. For example:

lmplz -o 5 -S 80% -T /tmp < corpus.txt > model.arpa

The command extracts a 5-gram model from the corpus, utilizing up to 80% of available RAM, and writes temporary data to the /tmp directory if memory is insufficient. In this way, KenLM is capable of processing extremely large corpora to build comprehensive N-gram models-something that tools like SRILM often struggle with. Unlike IRSTLM, KenLM does not rely on approximate simplifications during model construction, making it well-suited for high-performance language modeling tasks.

At the end of the N-gram counting process, a special text-based ARPA file (ARPA Language Model format) is generated. This file stores the probability-or more commonly, the log-probability-of each N-gram, along with optional back-off weights (explained below). The overall vocabulary of the corpus is also determined at this stage-for instance, in our case, the vocabulary size amounted to {V} words, after merging low-frequency terms into the <unk> (unknown) token. In subsequent stages, all computations were based on this ARPA-formatted language model.

1. Smoothing and Probability Estimation. Direct estimation of the vast number of probabilistic model parameters $P(w_i|w_{i-N+1\ldots i-1})$based on N-grams using corpus frequency counts (maximum likelihood estimation) does not yield satisfactory results. This is because it is natural for many N-grams to be completely absent in the training corpus. For such unseen sequences, a naïve frequency-based calculation assigns a probability of zero, which is problematic from the perspective of language modeling-zero-probability events are unrealistic in natural language, or at least extremely rare.

Therefore, smoothing techniques were employed.
The purpose of smoothing is to assign a non-zero probability mass to unseen N-gram combinations and to slightly lower the estimated probabilities of observed N-grams in order to "smooth" the distribution. The literature describes several well-known smoothing methods, such as Laplace (Add-one), Good-Turing, Kneser-Ney, and Witten-Bell [13]. In our implementation, we selected the **Modified Kneser-Ney smoothing** technique, which is recognized as one of the most effective approaches in modern language modeling.

KenLM software employs this exact method, constructing a combined model that integrates discounting and back-off for probability estimation. In contrast, the IRSTLM toolkit by default applies the lighter **Witten–Bell smoothing** method, which, although efficient for large datasets, may offer slightly lower quality compared to Kneser-Ney. In fact, independent studies have shown that for 5-gram models, **Kneser-Ney smoothing yields better perplexity scores** than Witten–Bell. Therefore, we adopted the Kneser-Ney method; and where supported, we also used IRSTLM's approximate implementation of the *"improved Kneser-Ney"* variant. During the smoothing process, the **back-off scheme** was applied: if a given N-1 context + word combination does not appear in the corpus (i.e., its frequency is

zero), the model “backs off” to a lower-order context-specifically, the (N-1)-gram model uses the N-2 word context.
To ensure the total probability mass remains normalized, a **back-off weight** (λ) is assigned to each lower-order model.
For example, in the case of the bigram model, the Kneser–Ney probability is formulated as follows:

If the word $\boldsymbol{w_i}$has been observed in the context of the preceding word $\boldsymbol{w_i - 1}$ (i.e., if, $\boldsymbol{c(w_{i-1,i}) > 0}$):

$$P(w_i|w_{i-1}) = \frac{c(w_{i-1,i}) - d}{c(w_{i-1})}$$

"Here, *d* is the discount constant."

If the word $\boldsymbol{w_i}$has never occurred in the context of $\boldsymbol{w_i - 1}$ $\boldsymbol{(c(w_{i-1,i}) = 0)}$:

In other words, when the bigram model fails, it backs off to the unigram probability, where λ denotes the back-off weight and $P(w_i)$ represents the unigram probability [14].

The above formulas represent a simplified version of the Kneser–Ney back-off smoothing technique, which can be recursively applied to higher-order N-grams in a similar fashion. As a result of smoothing, the ARPA language model file includes, for each N-gram, either the probability (or its log-base-10 value) along with the back-off weights where necessary.

It is important to note that the developers of IRSTLM implemented a slightly simplified version of the full Kneser–Ney smoothing to optimize memory usage (referred to as the “modified shift-beta” method) [15]. In contrast, KenLM provides a full implementation of the Kneser–Ney algorithm. For this reason, the probabilities computed by KenLM may be considered slightly more reliable when compared to those generated by IRSTLM for the same corpus. Therefore, in constructing the final model, we used the probabilities obtained from KenLM as the primary source. However, we also compared the models built using both toolkits to evaluate their performance (discussed in more detail below).

Language Model Format Preparation. Once the N-grams and their smoothed probabilities are ready, the next step involves preparing the language model for efficient storage and usage. Many language modeling (LM) systems can directly load and operate on text-based ARPA files. However, ARPA files tend to be large in size-especially when the model includes high-order N-grams-and line-by-line searches during runtime may introduce significant latency. Therefore, in

practical applications, it is preferable to compile the language model into a binary format.

IRSTLM provides a dedicated tool called compile-lm for this purpose, which converts an ARPA-formatted model into a compact .bin file. Similarly, KenLM includes a *build_binary* utility that performs the same conversion from ARPA to binary format. A noteworthy feature of KenLM is its support for multiple internal data structures during binary format compilation, allowing the user to choose between trie (tree-based structure), probing (hash table), or sorted (sorted array) implementations depending on the specific performance or memory constraints.

The default format in KenLM is probing, which ensures the fastest query speed but consumes more memory. In contrast, the trie format is slightly slower but significantly more memory-efficient. In our project, we primarily used KenLM's trie-based binary format for integrating the language model. This approach resulted in a noticeably smaller LM file size and eliminated the need to load the entire model into RAM during runtime.

For example, the KenLM trie-format model consumed only 58% of the resident memory required by IRSTLM’s most compact version-almost half the size-while still achieving 81% of the query speed compared to IRSTLM’s fastest configuration. In other words, memory efficiency was improved significantly with only a marginal loss in processing speed.

Similar binary model compaction can also be achieved with IRSTLM through quantization: IRSTLM supports storing probabilities using 1-byte representations instead of the standard 4-byte floating-point format. Using the quantize-lm utility, the ARPA model can be converted into a quantized version, resulting in approximately fourfold reduction in size.

In our experiments, we built the model using KenLM’s trie representation, and also tested its usage with mmap (memory-mapped file access)[8]. In this mode, the model is mapped directly from disk into memory, and only required segments are loaded dynamically during execution. This memory mapping strategy is especially useful for large-scale models, as it allows multiple processes to share a single copy of the model loaded from disk, reducing redundant memory usage.

However, since our model size was relatively modest (around several hundred megabytes), we typically loaded the entire model into memory, which significantly reduced loading time and improved query performance.

Inference Mechanism (Prediction). After constructing the N-gram language model, it was necessary to develop appropriate functionality to evaluate

the model and integrate it into various applications. The core principle of a language model lies in its ability to return a probability distribution over the next possible word given a specific context -that is, the previous N-1 words. In our case, for the KenLM-based model, a custom Python wrapper was developed to provide fast scoring functionality for given text inputs. This function computes the probability (or log-probability) of each word sequence in the sentence using the language model, thereby enabling the calculation of the total log-likelihood of a sentence, as well as its perplexity. Based on the same mechanism, the perplexity values described in the next paragraph were obtained. For using the language model for next word prediction, the following approach was applied: given the sentence-begin token <s>, the word with the highest probability in the model's probability distribution was selected as the first predicted word. This word was then added to the context, another query was made to the model, and the next word was selected. This process continued until the sentence-end token </s> was generated. As a result, a program was developed that allows the model to independently generate sentences.

However, always selecting the word with the highest probability may result in deterministic text (with frequent repetitions and sentences of similar structure). Therefore, in generative mode, a sampling mechanism - such as Boltzmann sampling or top-k random sampling - was also tested to select words from the probability distribution. In general, the prediction mechanism of the N-gram model is a simple tree-based search process, where if a high-order (e.g., 5-gram) context is not found, the model backs off to a lower-level model and ultimately relies on the unigram probability. Thus, using the constructed model, functionalities such as assigning probabilistic scores to various input texts, predicting the next word, and fully generating word sequences were achieved.

Corpus frequency statistics and rare linguistic phenomena

Language models built as N-gram models rely entirely on statistical data. If a word combination appears frequently in a large corpus, the model assigns it a high probability; conversely, rarely occurring or unseen combinations are given very low probabilities. Due to the nature of language, the problem of data sparsity arises even in very large corpora-that is, only a small fraction of all possible word combinations is actually observed in the corpus because of the immense number of potential sequences. For example, Table 1 presents statistics obtained from the Uzbek web corpus (number of sentences: 1,120,458), where the number of unique words (unigrams) is 386,069, the number of bigrams is 4,799,233, and the number of trigrams is 9,549,450. Figure 2 illustrates the distribution of N-grams (in logarithmic scale), Figure 3 shows the linear graph of N-grams, and Figure 4 presents the logarithmic graph of N-grams.

TABLE.1. DISTRIBUTION OF N-GRAM COUNTS (ON A LOGARITHMIC SCALE)

Sentences	**1,120,458**
Tokens	19,984,549
Unigrams (unique)	386,069
Bigrams (unique)	4,799,233
Trigrams (unique)	9,549,450
4-grams (unique)	11,811,234
5-grams (unique)	12,291,580
6-grams (unique)	11,971,849

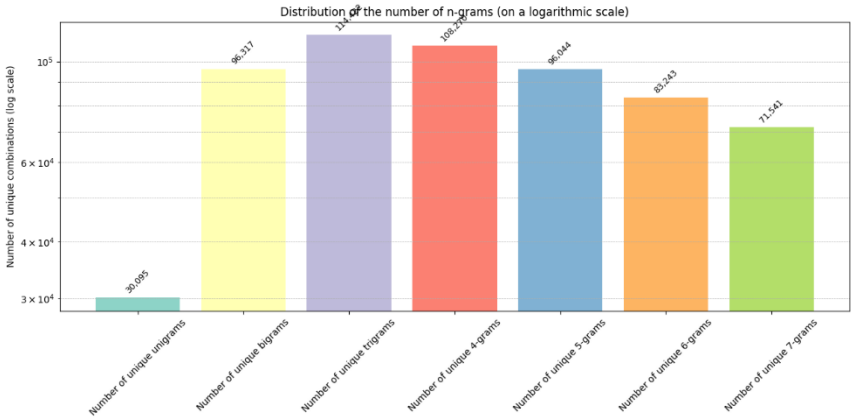

Fig.2. Distribution of N-gram counts (on a Logarithmic Scale)

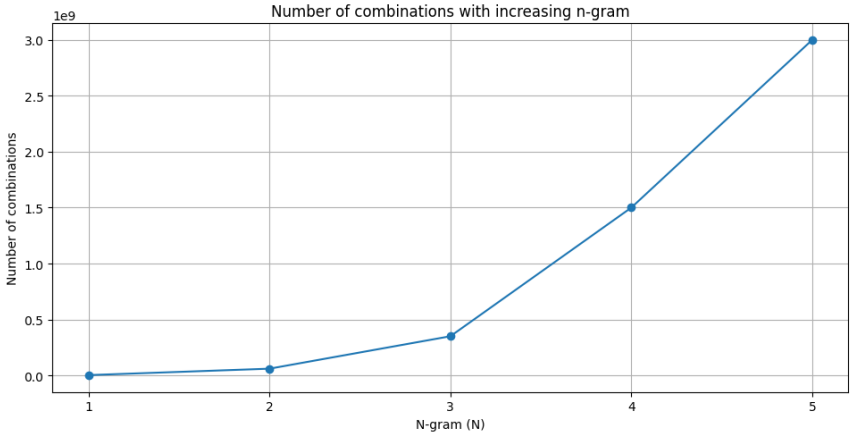

Fig.3. Linear graph of N-gram frequencies

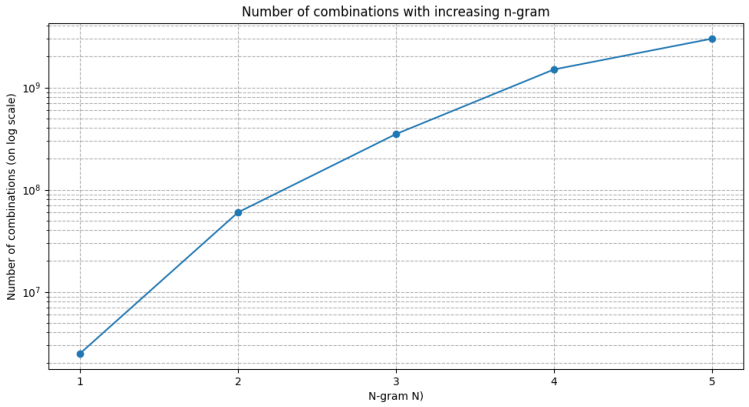

Fig.4. Logarithmic plot of N-gram frequencies

The line chart clearly illustrates how the number of combinations increases with the order of N-grams. Starting from 4-grams, a sharp exponential growth is observed.

Logarifmik grafik — Eksponensial o‘sishni yanada ravshanroq ko‘rsatadi va yuqori tartibli N-gram modellarning resurs talabini vizual baholashga yordam beradi.

The combinatorial explosion of N-grams

As the corpus size and lexical richness increase, the number of higher-order N-grams grows significantly. For instance, in the Uzbek National Corpus, the number of trigrams is several times greater than the number of unigrams, with 9,549,450 unique trigrams observed. Table 1 provides statistics up to 7-grams. This is a language-specific phenomenon for Uzbek: due to its agglutinative structure and relatively free word order, the number of possible word sequence combinations is immense, while only a small portion of them is actually observed in practice. Consequently, the larger the value of N, the more parameters the model requires (i.e., the more N-grams are observed). Therefore, when the corpus size is limited, lower-order models such as trigrams or 4-grams are typically used, as collecting reliable statistics for higher-order N-grams becomes increasingly difficult.

Low-frequency words and contexts pose a particular challenge in N-gram models. For combinations that appear infrequently in the corpus, the maximum likelihood estimate tends to be low-even if such combinations are logically plausible in real language use. Smoothing techniques address this issue by assigning non-zero probabilities to rare or unseen events, thereby improving the generalization capability of the model. As an example, Figure 5 compares the perplexity of 3-, 5-, 7-, and 9-gram models built for the Uzbek language on the test set, along with the evaluation of different smoothing techniques based on their perplexity performance. When Kneser-Ney smoothing is applied, the 3-gram model

yields a test perplexity of 121, the 5-gram model 114, the 7-gram model 112, and the 9-gram model 108. This indicates that higher-order models provide more accurate probabilistic predictions, as reflected by decreasing perplexity. Katz smoothing shows a similar trend, though with slightly higher values (e.g., the 9-gram Katz model has a perplexity of 112), and overall performs less effectively than the Kneser-Ney interpolated approach.

Thus, while increasing N theoretically expands the contextual coverage of statistical models, practical limitations such as smoothing requirements and resource constraints (e.g., memory, corpus size) must be taken into account.

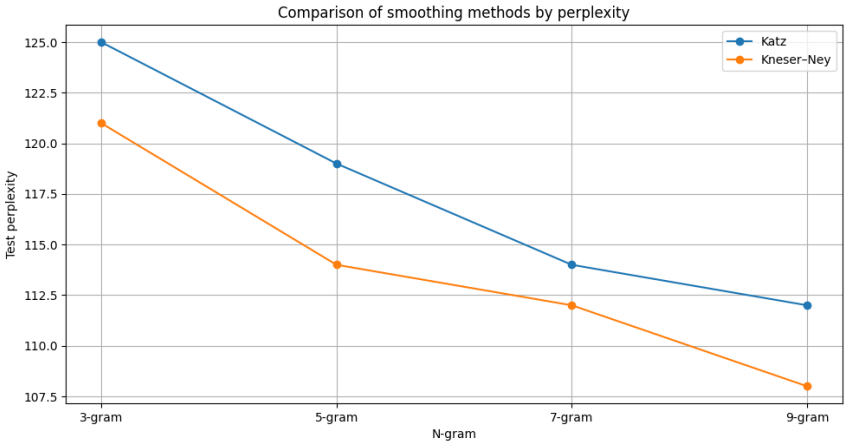

Fig.5. Comparison of smoothing methods based on perplexity

Conclusion

This study presented a detailed analysis of the stages and technical aspects involved in building an n-gram-based language model for the Uzbek language, with a focus on the tools employed. Key stages such as data cleaning, n-gram frequency computation, probability smoothing, and building the model in a resource-efficient manner were carefully addressed. The use of toolkits such as IRSTLM and KenLM was compared, highlighting their respective strengths and weaknesses through practical experiments. Since KenLM provides a full implementation of Kneser-Ney smoothing, the probabilities generated by KenLM on the same corpus may be somewhat more reliable compared to IRSTLM. Therefore, in building the final model, we based our approach on the probabilities obtained from KenLM, while also comparing the models constructed using both toolkits. Specifically, IRSTLM was noted for its memory efficiency, while KenLM demonstrated superior performance and accuracy. Notably, KenLM's implementation of the modified Kneser-Ney smoothing algorithm produced high-quality results and achieved favorable perplexity scores on large corpora. The paper also presents corpus

frequencies and rare occurrences. In particular, the distribution of N-gram counts is provided. The findings confirm that an effective n-gram model can be successfully built for Uzbek textual data. Such a model can serve as a foundation for various natural language processing tasks, including machine translation, text generation, text analysis, and word prediction.

References

1. Shannon, C.E.: A Mathematical Theory of Communication. Bell Syst. Tech. J. 27(3), 379–423 (1948)
2. Davies, M.: Google Books. American English Corpus Project, Brigham Young University (2011). http://googlebooks.byu.edu/
3. Jurafsky, D., & Martin, J. H. (2018). N-gram language models. Speech and language processing, 23.
4. Zhang, W. (2015). Comparing the Effect of Smoothing and N-gram Order: Finding the Best Way to Combine the Smoothing and Order of N-gram.
5. https://www.geeksforgeeks.org/nlp/discounting-techniques-in-language-models/
6. Boltayevich, E.B., Ilxomovna, A.X., Hakim qizi, P.M., Uktambay O‘g‘li, K.N.: Semantic Differentiation of Uzbek Homonyms Using the Lesk Algorithm. In: 2023 8th International Conference on Computer Science and Engineering (UBMK), pp. 137–140. IEEE, Burdur (2023). https://doi.org/10.1109/UBMK59864.2023.10286666
7. Elov, B.B., Primova, M.H.: Development of an Information System for Organizing and Managing the Educational Process Based on Smart Technologies. In: XVI International Scientific and Practical Conference “State and Prospects for the Development of Agribusiness – INTERAGROMASH 2023”. https://doi.org/10.1051/e3sconf/202341303010
8. Nadas, A.: Estimation of Probabilities in the Language Model of the IBM Speech Recognition System. IEEE Trans. Acoust. Speech Signal Process. 32(4), 859–861 (1984)
9. Church, K.W., Gale, W\.A.: A Comparison of the Enhanced Good-Turing and Deleted Estimation Methods for Estimating Probabilities of English Bigrams. Comput. Speech Lang. 5(1), 19–54 (1991)
10. Park, J. H., Song, Y. I., & Rim, H. C. (2007, September). Smoothing algorithm for n-gram model using agglutinative characteristic of Korean. In International Conference on Semantic Computing (ICSC 2007) (pp. 397-404). IEEE.
11. Chen, S. F., & Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. Computer Speech & Language, 13(4), 359-394.
12. Mani, S., Gothe, S.V., Ghosh, S., Mishra, A.K., Kulshreshtha, P., Bhargavi, M., Kumaran, M.: Real-Time Optimized N-Gram for Mobile Devices. In: 2019 IEEE 13th Int. Conf. Semantic Computing (ICSC), pp. 87–92. IEEE, Newport Beach (2019)
13. Ismail, R.: Comparison of Modified Kneser-Ney and Witten-Bell Smoothing Techniques in Statistical Language Model of Bahasa Indonesia. In: 2014 2nd Int. Conf. Information and Communication Technology (ICoICT), pp. 409–412. IEEE, Bandung (2014)
14. Heafield, K., Pouzyrevsky, I., Clark, J.H., Koehn, P.: Scalable Modified Kneser-Ney Language Model Estimation. In: Proc. 51st Annual Meeting of the Association

for Computational Linguistics (Volume 2: Short Papers), pp. 690–696. ACL, Sofia (2013)
15. Heafield, K.: KenLM: Faster and Smaller Language Model Queries. In: Proc. Sixth Workshop on Statistical Machine Translation, pp. 187–197. ACL, Edinburgh (2011). [https://aclanthology.org/W11-21] (https://aclanthology.org/W11-21)

Organized by
Stamford International University
Bangkok, Thailand

http://ictsmet.com/2025