Statistical Language Models Using N-Grams for Syllables and Characters

Tugay Talha Icen

October, 2024

Contents

1 Introduction

N-grams are sequences of n consecutive items (words, letters, or symbols) taken from a larger sequence. For example, in the phrase "I love cats": "I" is a unigram, "I love" is a bigram, and "I love cats" is a trigram. N-gram models predict what comes next in a sequence by analyzing how frequently these patterns appear in training data. This simple but powerful concept is widely used in language processing, biology, and other fields where pattern prediction is valuable. This project develops two such models for Turkish: 3 based on syllables and the other 3 on characters, comparing their performance through perplexity and generated sentence quality. Syllable-based models capture phonetic structures, while character-based models offer finer granularity but may sacrifice some semantic understanding. This report details the implementation, results, and analysis of both models.

2 Design and Implementation

The project is implemented in Python, utilizing several key libraries:

`turkishnlp`: Employed specifically for its syllabification capabilities, essential for the syllable-based model.

`scipy.linalg.lstsq`: Used for performing least squares regression in the Simple Good-Turing smoothing process.

`collections`: The `defaultdict` class from this module is used for efficient storage and retrieval of n-gram counts.

`math`: Provides mathematical functions such as `log` and `exp`, crucial for perplexity and probability calculations.

`random`: Used for random sampling, particularly in sentence generation.

`numpy`: Used for numerical operations in the smoothing process.

The code is modularized into the following components:

2.1 Data Preparation (clean_data.py, divide_syllables_letters.py, train_test_split.py)

1. **Cleaning (`clean_data.py`):**

   - **Input:** Path to the raw Turkish Wikipedia dump file (`input_path`), path to the output cleaned file (`output_path`).
   - **Output:** A cleaned text file stored at `output_path`.
   - **Functionality:** Removes XML tags and empty lines from the raw Wikipedia data. Converts Turkish characters to their English equivalents (`ç -> c, ğ -> g, etc.`) using `str.maketrans`. Tracks and

displays progress percentage during cleaning. The `turkish_to_english` function handles the character mapping.

2. **Syllabification/Normalization (`divide_syllables_letters.py`):**

   - **Input:** Path to the cleaned Wikipedia file (`input_path`), paths to the syllable output file (`syllable_output_path`) and character output file (`char_output_path`).
   - **Output:** Two files: one with syllabified text and another with normalized character-based text.
   - **Functionality:** Creates syllable-based and character-based datasets from the cleaned data. The `syllabicate_word` function, using the `turkishnlp` library, splits words into syllables. The `normalize_text` function converts text to lowercase for the character-based model. Tracks and displays progress percentage.

3. **Splitting:** `train_test_split.py`

   - **Input:** Path to the input file (`input_path`), paths to the training output file (`train_output_path`) and test output file (`test_output_path`), split ratio (`split_ratio`, default 0.95).
   - **Output:** Two files: one for training data and one for test data.
   - **Functionality:** Splits both syllable-based and character-based datasets into training and testing sets according to the specified split ratio(e.g. if split ratio 0.95 then first %95 of dataset will be train and last %5 are train part). Displays progress percentage.

2.2 N-Gram Calculation (n_grams.py, smoothing.py)

1. **N-gram Creation:** `n_grams.py`

   - **Input:** Path to the input file (`input_path`), path to the output file (`output_path`), n-gram size (`n`), a boolean indicating syllable or character model (`syllable_or_letter`).
   - **Output:** A file containing n-gram counts.
   - **Functionality:** Creates n-grams of the specified size from the preprocessed data (syllable-based or character-based). The `syllable_line` function splits each line into units (syllables or characters). N-gram counts are stored in a `defaultdict`. The `n_gram_to_string` function converts the n-gram and its count into a string format suitable for file storage using `<ayrim>` as a separator. Tracks and displays progress percentage. The `create_all_n_grams` function orchestrates the creation of 1-gram, 2-gram, and 3-gram files for both models.

2. **Smoothing:** `smoothing.py`

   - **Input:** Path to the n-gram file (`n_gram_path`), optional path to the output file (`output_path`), confidence level for Good-Turing smoothing (`confidence_level`).
   - **Output:** A file containing smoothed n-gram probabilities.
   - **Functionality:** Implements Simple Good-Turing smoothing. The `count_of_counts_table` function calculates the frequency of frequencies. `log_linear_regression` performs a least-squares regression to estimate smoothed counts for higher frequencies (those above the first unseen frequency). `simple_good_turing_smoothing` applies the smoothing algorithm and handles unseen n-grams using the `"<UNK>"` token. Probabilities are normalized, and their sum is verified to be close to 1. Smoothed probabilities are written to a file. The `smooth_all_n_grams` function manages smoothing for all six n-gram files (three for each model type).

2.3 Perplexity Calculation (calculate_perplexity.py)

- **Input:** N-gram dictionary (`n_gram_dict`), smoothed n-gram dictionary (`smoothed_n_grams`), path to the test file (`test_file_path`), n-gram size (`n`), a boolean for syllable or character model (`syllable_or_letter`).
- **Output:** Perplexity score (a float).
- **Functionality:** Calculates the perplexity of the language model on the test set using the smoothed n-gram probabilities. Iterates through the test set, retrieving probabilities for each n-gram (or the `"<UNK>"` probability for unseen n-grams). Calculates the log-probability of each n-gram and accumulates the sum. Finally, calculates perplexity using the formula: $\text{Perplexity} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i|w_1, w_2, \ldots, w_{i-1})\right)$

2.4 Sentence Generation (sentence_generation.py)

- **Input:** N-gram dictionary (`n_gram_dict`), n-gram size (`n`), a boolean indicating syllable or character model (`syllable_or_letter`).
- **Output:** A randomly generated sentence (a string).
- **Functionality:** Generates random sentences using the n-gram model. The `generate_sentence` function selects a random starting n-gram from among the most frequent n-grams in the training data. At each step, it randomly picks one of the top 5 most probable next words (or syllables) following the current n-1 word (or syllable) sequence, based on the n-gram probabilities. The process continues until a period ('.') is encountered or a maximum sentence length is reached. The function returns the generated

sentence as a string. The `string_to_n_gram` function parses n-grams for sentence generation.

3 Results and Tables

3.1 Perplexity Results

Table 1: Perplexity Results

Model	1-gram	2-gram	3-gram
Syllable-based	**537.86**	**3106.56**	**10028.26**
Character-based	**71.50**	**418.83**	**2395.15**

3.2 Generated Sentences

Table 2: Generated Sentences

Model	1-gram
Syllable-based	**sisidadalasiridasisidalaririlerilalasirida ledalalasidadadarilaridadasisisiladadasida**
Character-based	**nerienaererrneranraie nirreinririaeearrrran**
	2-gram
Syllable-based	**na ve atinda ve ve ve acak i atin le o acak oman ale ipa o**
Character-based	**. ngiz rcengiz iz ngiz ngiz ngiz rce**
	3-gram
Syllable-based	**ilkrelase de 1206 yi ya da do ta as aritokra hi i kip ko be boz a dela**
Character-based	**is k b bratos k b b b engiz h bdurucingiz h**

4 Analysis and Conclusion

4.1 Perplexity Analysis

The perplexity results reveal several interesting patterns in the performance of syllable-based and character-based models:

1. Character-based models consistently achieved lower perplexity scores across all N-gram sizes:
   - 1-gram: 71.50 vs 537.86
   - 2-gram: 418.83 vs 3106.56
   - 3-gram: 2395.15 vs 10028.26
2. Both models show increasing perplexity as N increases, indicating that higher-order models are less certain about their predictions. This is expected because:
   - Longer sequences are less likely to appear in the training data.
   - The sparsity problem becomes more severe with larger N.
   - The model has to distribute probability mass across more possible combinations.
3. The rate of perplexity increase is steeper for the syllable-based model, suggesting it suffers more from data sparsity issues at higher orders.

4.2 Generated Text Analysis

The quality of generated text varies significantly across models and N-gram sizes:

4.2.1 Character-based Model:

- 1-gram output appears as random character sequences with no clear structure.
- 2-gram begins to show some Turkish-like patterns but lacks coherence.
- 3-gram produces more recognizable fragments (e.g., "engiz" appearing multiple times, likely from names like "Cengiz").

4.2.2 Syllable-based Model:

- 1-gram generates longer sequences but with invalid syllable combinations.
- 2-gram shows improvement with some valid Turkish word fragments ("acak", "ve").
- 3-gram produces the most natural-looking output, including recognizable words and numbers ("1206", "ilkrelase").

4.3 Trade-offs Analysis

4.3.1 Syllable-based Model Advantages:

- Better captures Turkish phonological rules.
- Produces more linguistically plausible sequences at higher N-gram levels.
- More efficient representation of longer sequences.

4.3.2 Syllable-based Model Disadvantages:

- Higher perplexity scores indicate less predictive power.
- Requires more complex preprocessing (syllabification).
- Suffers more from data sparsity.

4.3.3 Character-based Model Advantages:

- Lower perplexity scores across all N values.
- Simpler implementation without need for syllabification.
- More robust to spelling variations and rare words.

4.3.4 Character-based Model Disadvantages:

- Generated text often lacks phonological validity.
- Requires longer sequences to capture meaningful patterns.
- May waste capacity learning invalid character combinations.

4.4 Conclusion

Based on the quantitative and qualitative analysis, we can conclude:

1. For pure predictive power (as measured by perplexity), the character-based model is superior across all N-gram sizes.
2. For generating human-readable Turkish text, the 3-gram syllable-based model produces the most natural results, despite its higher perplexity.
3. The optimal choice depends on the specific application:
    - For tasks requiring accurate prediction (e.g., text completion), the character-based model would be more suitable.
    - For tasks involving text generation, the syllable-based 3-gram model would be more appropriate.
4. Future improvements could focus on:

- Implementing more sophisticated smoothing techniques.
- Combining both approaches in a hybrid model.
- Incorporating morphological analysis to better handle Turkish's agglutinative nature.

In conclusion, while the character-based model shows better statistical performance, the syllable-based model appears more suited for modeling Turkish's linguistic properties. This suggests that raw perplexity scores alone may not be the best metric for evaluating language models for Turkish, and that consideration of the language's unique phonological and morphological characteristics is crucial.

5 Table on LLM Usage

Table 3: LLM Usage

Task	**LLM Usage**
Report Writing	Structuring, phrasing, and explaining concepts. Generating the report template.
Code Implementation	LLM used minimally for code complementation(copilot for complete comment lines lines) and clarifying syntax.