National University of Computer and Emerging Sciences, Islamabad

Fall-2023

Natural Language Processing

Assignment 2

Name:	**Syeda Areeba Nadeem**
Roll No:	21I-0307
Section:	BS(AI) - K

1 Web Scrapping using Scrapy

A web scraping script was developed using the Scrapy framework in Python to extract Urdu poems from the Rekhta website (`www.rekhta.org`). The script is designed to scrape poems written by various poets listed on the website.

1.1 Scrapy Spider

Scrapy spiders are classes that define how a certain site (or a group of sites) will be scraped for data. Scrapy Spider named `I210307urdupoemsspider` was created to crawl the Rekhta website. The spider starts by visiting the URLs of poets' pages, where each page contains a list of poems written by a specific poet.

1.1.1 Name and Allowed Domains

The spider's name was defined as `I210307urdupoemsspider` and the allowed domains to be scraped as `www.rekhta.org` using the `allowed_domains` attribute were specified.

1.1.2 Start URLs

The `start_urls` attribute contains a list of URLs from which the spider will start crawling. Each URL corresponds to a poet's page containing a list of poems.

1.2 Parsing Poem Links

The spider extracts URLs of all poems written by each poet and follows each poem link to extract its data.

1.3 Parsing Poem Data

The **parse()** method is called to handle the response downloaded for each request made. Within this method, the URLs of all poems written by the poet are extracted from the response using CSS selectors. For each poem, the spider extracts the following in Urdu:

- Name of the Poet
- Title of the Poem
- Lines of the Poem

1.3.1 Extracting Poet and Poem Name

The poet's name is extracted using `response.css()` with the CSS selector **div.authorAddFavorite h2 a.ghazalAuthor::text()**, while the poem's name is extracted using **response.xpath()** with the XPath selector **//h1/text()**.

1.3.2 Extracting Poem Lines

All `<p>` elements containing the poem text were selected using the CSS selector `div.c > p`. By iterating over each paragraph, text of each `<span>` element is extracted, joined, and appended to the **urdu_text_lines** list.

1.3.3 JSON Output

The scraped data is stored in a JSON file, containing the poet's name, poem title, and poem lines. This structured format enables easy access to the scraped data for analysis and processing. Later on, it is converted into CSV format using pandas' DataFrame. The data is saved in JSON format by running following command on terminal:

```
scrapy crawl I210307urdupoemsspider -o poems_data.json
```

2 Corpus

The final corpus, converted into CSV from JSON, looked like the figure 1.

Statistic	Value
Number of Poems	484
Number of Poets	25
Total number of starting words	1815
Total number of ending words	1136
Total number of unique words (vocab size)	6116

Table 1: Corpus Statistics

	Poet name	Poem name	Poem
0	فیض احمد فیض	ہم پر تمہاری چاہ کا الزام ہی تو ہے	...دشنام تو ن\nہم پر تمہاری چاہ کا الزام ہی تو ہے
1	فیض احمد فیض	دونوں جہان تیری محبت میں ہار کے	... وہ جا رہا ہے\nدونوں جہان تیری محبت میں ہار کے
2	کیف احمد صدیقی	ہو گئے ناکام تو پچھتائیں کیا	... دوستوں کے سامنے\nہو گئے ناکام تو پچھتائیں کیا
3	آنس معین	ہو جائے گی جب تم سے شناسائی ذرا اور	... بڑھ جائے\nہو جائے گی جب تم سے شناسائی ذرا اور
4	خورشید اکبر	گھر بار کہاں کوچہ و بازار مری جاں	...درویش کو دن\nگھر بار کہاں کوچہ و بازار مری جاں
...	...	...	...
479	ایم کوثرپیاوی راہی	رسم کسی وحشی کا ابھی دیکھ رہا ہے	...یہ پیڑ جو ا\nرسم کسی وحشی کا ابھی دیکھ رہا ہے
480	ایم کوثرپیاوی راہی	اک انجان راہ پر دونوں	...\nمل گئے آج بے خطر دونوں\nاک انجان راہ پر دونوں
481	ایم کوثرپیاوی راہی	جلے گا چاند ستارے دھواں اڑائیں گے	... ہمارے خواب\nجلے گا چاند ستارے دھواں اڑائیں گے
482	ایم کوثرپیاوی راہی	گھوموں نہیں تو کیا میں کہیں جا کے پڑ رہوں	...ہیم\nگھوموں نہیں تو کیا میں کہیں جا کے پڑ رہوں
483	ایم کوثرپیاوی راہی	ہجر کا چاند درد کی ندی	... یہی صورت ہے اپنی دنیا\nہجر کا چاند درد کی ندی

484 rows × 3 columns

Figure 1: Original Corpus after conversion from JSON to CSV

3 Poetry Generation using N-grams

N-grams are contiguous sequences of n items from a given sample of text or speech. In the context of poetry generation, we use N-grams to model the structure and language of existing poems in order to generate new ones.

3.1 Forward Bigrams

Forward bigrams are pairs of consecutive words in a poem, where the second word follows the first. Forward bigrams are used to predict the next word based on the preceding word. This approach allows the generation of poetry with a little consistent and sound flow of words.

3.2 Backward Bigrams

Backward bigrams are pairs of consecutive words in a poem, where the first word follows the second. Unlike forward bigrams, backward bigrams are used to predict the previous word based on the succeeding word. This technique can also contribute to generating poetry with a coherent structure.

3.3 Trigrams

Trigrams are sequences of three consecutive words in a poem. Trigrams are used to capture more context and dependencies between words compared to bigrams. The goal is to make poetry that sounds better and makes more sense by looking at groups of three words together.

3.4 Bidirectional Bigrams

Bidirectional bigrams combine both forward and backward bigrams to generate poetry. This approach considers both preceding and succeeding words to predict the next word, enhancing the model's understanding of contextual dependencies. Bidirectional bigrams can result in poetry with improved flow and consistency.

4 Rhyming Words

The `get_rhyming_words` function takes a single word as input and finds rhyming words for that word. It iterates over each element in the list of unigrams, checking if the last character of the unigram matches the last character of the input word. If there are matching last characters, it creates a list containing the rhyming words and returns it.

5 Perplexity Calculation

Perplexity is a measure of how well a probability model predicts a sample or a test sentence. In the current context of poetry generation using N-grams, perplexity helps evaluate the effectiveness of different N-gram models in predicting the next word in a given poem.

The perplexity of a language model is calculated using the formula:

$$\text{Perplexity} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i|w_{i-1}, w_{i-2}, \ldots, w_{i-n+1})\right)$$

Where:

- N is the total number of words in the poem.
- $P(w_i|w_{i-1}, w_{i-2}, \ldots, w_{i-n+1})$ is the probability of word w_i given its preceding $n-1$ words.

The perplexity for each N-gram model (forward bigrams, backward bigrams, trigrams, and bidirectional bigrams) based on the generated poetry is calculated.

5.1 Results

The results of perplexity calculation are summarized in the table 2. Lower perplexity values indicate better predictive performance of the language model.

Table 2: Perplexity values of different n-gram models across different random examples

Example No.	Forward Bigrams	Backward Bigrams	Bidirectional Bigrams	Trigrams
50	39.786	1.684	40.191	1.500
140	39.861	2.559	47.753	1.712
463	26.994	2.263	30.384	2.011
221	26.028	1.752	27.208	1.618
331	41.655	2.605	49.978	2.270
128	45.664	4.509	49.338	2.579
248	42.864	4.716	55.829	2.124
69	31.914	2.817	45.519	2.026
336	33.627	3.403	42.821	1.929
80	38.836	4.261	45.049	1.812

5.2 Observations

Based on the provided perplexity values for different models (Forward Bigrams, Backward Bigrams, Bidirectional Bigrams, and Trigrams) across multiple examples, several observations can be made:

1. **Performance Variation**: The perplexity values vary significantly across different examples and models. For instance, the perplexity for Forward Bigrams ranges from approximately 26 to 45, showcasing a notable disparity in performance depending on the text sample.

2. **Bidirectional Bigrams Performance**: Bidirectional Bigrams tend to exhibit higher perplexity values compared to Forward and Backward Bigrams in most cases. This suggests that while incorporating both forward and backward contexts can enhance language modeling, it may not consistently lead to better predictive performance.

3. **Trigrams Performance**: Trigrams generally demonstrate the lowest perplexity values among all models across various examples. This indicates that considering the context of two preceding words (Trigrams) often provides better predictive power compared to models relying on only one preceding word (Bigrams).

4. **Impact of Context Size**: The results highlight the importance of context size in language modeling. Models that consider a larger context, such as Trigrams, tend to perform better in terms of perplexity, suggesting that incorporating more context can lead to more accurate predictions.

5. **Trade-off between Complexity and Performance**: While Trigrams generally perform well, they also come with increased computational complexity due to the larger context they consider. This trade-off between model complexity and performance needs to be considered when selecting the appropriate model for a given task.

Based on the observed perplexity values and considering both performance and computational considerations, the Trigrams model emerges as the best-performing option in current case, followed by backward bigram model.

6 Challenges Faced

1. **Making Stanza Rhyme**
   Since there is no library that finds rhyming words in Urdu language, ensuring rhyme scheme consistency, and maintaining the structure of stanzas was a big challenge. Ensuring that rhyming words stay the same throughout various verses and following set stanza patterns was the goal to achieve.

2. **Poetry Generation**
   Selecting appropriate starting and ending words and ensuring consistency in the rhyme scheme proved to be a challenging task. Additionally, choosing the next word based on its probability posed another challenge. Since selecting the word with the highest probability every time was not feasible, alternative strategies had to be devised.

3. **Evaluation Metric**
   A challenge in choosing a suitable metric for evaluating language models lies in selecting one that effectively captures the model's performance. Additionally, determining the appropriate test data, particularly in the context of poetry generation, posed another challenge.

7 Challenges Solved with Improvements

1. **Making Stanza Rhyme**

   - The function `get_rhyming_words(word)` was implemented to find rhyming words by checking if the last character of a word matches the last character of any other word in the unigrams.
   - Rhyming words were selected from this list along with some other conditions to maintain the logical structure of stanza.

2. **Poetry Generation**

   - Starting and ending words were selected randomly from the list of starting and ending words obtained using the `get_start_end_words()` function.
   - Rhyme scheme consistency was ensured by selecting rhyming words based on the last word of the previous verse.
   - Next words were chosen based on their probability, with a scheduler parameter to control randomness. The function `calculate_next_word_prob()` handled this probabilistic selection.

3. **Evaluation Metric**

   - The code calculated perplexity for different n-gram models to evaluate their performance.
   - Perplexity was calculated using the `calculate_perplexity()` function, which considered the probabilities of n-grams based on their counts in the corpus.
   - The poems from original corpus were used as test data.

8 Conclusion

In conclusion, the process of scraping Urdu poems from Rekhta using Scrapy, creating a corpus, and generating poetry with N-grams was a complex task. Challenges like rhyme scheme consistency, word selection, and model evaluation were tackled and resolved. Further refinement could enhance poetry generation.