CS 458/535 - Natural Language Processing

Poetry Generation in Urdu

1 Introduction

In this assignment, you will use n-gram language modeling to generate some poetry using the **spaCy** library for text processing. For the purpose of this assignment a poem will consist of three stanzas each containing four verses where each verse consists of 7—10 words. For example, following is a manually generated stanza.

دل سے نکال یاس کہ زندہ ہوں میں ابھی،
ہوتا ہے کیوں اداس کہ زندہ ہوں میں ابھی،
مایوسیوں کی قید سے خود کو نکال کر،
آ جاؤ میرے پاس کہ زندہ ہوں میں ابھی،

آ کر کبھی تو دید سے سیراب کر مجھے،
مرتی نہیں ہے پیاس کہ زندہ ہوں میں ابھی،
مہر و وفا خلوص و محبت گداز دل،
سب کچھ ہے میرے پاس کہ زندہ ہوں میں ابھی،

لوٹیں گے تیرے آتے ہی پھر دن بہار کے،
رہتی ہے دل میں آس کہ زندہ ہوں میں،
نایابؔ شاخ چشم میں کھلتے ہیں اب بھی خواب، س ہ ے ترا
قیاس کہ زندہ ہوں میں ابھی

The task is to print three such stanzas with an empty line in between. The generational model can be trained on the provided Poetry Corpus containing poems from Faiz, Ghalib and Iqbal. You will train unigram and bigram models using this corpus. These models will be used to generate poetry. Several online solutions are available for English that use the NLTK library. However, we will be using the spaCy library to accomplish this task!

2 Assignment Task

The task is to generate a poem using different models. We will generate a poem verse by verse until all stanzas have been generated. The poetry generation problem can be solved using the following algorithm:

1. Load the Poetry Corpus
2. Tokenize the corpus in order to split it into a list of words
3. Generate n-gram models
4. For each of the stanzas
   - For each verse
     * Generate a random number in the range [7...10]
     * Select first word
     * Select subsequent words until end of verse
     * **[bonus]** If not the first verse, try to rhyme the last word with the last word of the previous verse
     * Print verse
   - Print empty line after stanza

2.1 Implementation Challenges

Among the challenges of solving this assignment will be the selection of subsequent words once we have chosen the first word of the verse. But, to predict the next word, what we want to compute is, what is the most probably next word out of all of the possible next words? In other words, find the set of words that occur most frequently after the already selected word and pick the next word from that set. We can use a Conditional Frequency Distribution (CFD) to figure that out! A CFD tells us: given a condition, what is likelihood of each possible outcome. [bonus] Rhyming the generated verses is also a challenge. You can build your dictionary for rhyming. The Urdu sentence is written from **right to left**, so makes your n-gram models according to this style.

2.2 Standard n-gram Models

We can develop our model using the Conditional Frequency Distribution method. First develop a unigram model (Unigram Model) and then the bigram model (Bigram Model). Select the first word of each line randomly from starting words in the vocabulary and then use the bigram model to generate the next word until the verse is complete. Generate the next three lines similarly. Follow the same steps for the trigram model and compare the results of the two n-gram models. **[bonus]** Can we make the sonnet rhyme? (Hint: Build a pronunciation dictionary)

2.3 Backward Bigram Model

Standard n-gram language models model the generation of text from left to right. However, in some cases, words might be better predicted from their right context rather than their left context. The next task is to produce a backward bigram model that models the generation of a sentence from right to left. Think of a very simple way of very quickly using Bigram Model to produce a BackwardBigramModel that is identical except for the modeling direction. Compare the results of the backward bigram model with previous implementations.

2.4 Bidirectional Bigram Model

Next, build a BidirectionalBigramModel that combines the forward and backward model. Both the **Backward- Bigram Model** and **BidirectionalBigramModel** should take the same input and produce the same style of output as Bigram Model. Compare the output with the previous models.