CS 458/535 - Natural Language Processing
Text Gerneration using bi-grams model

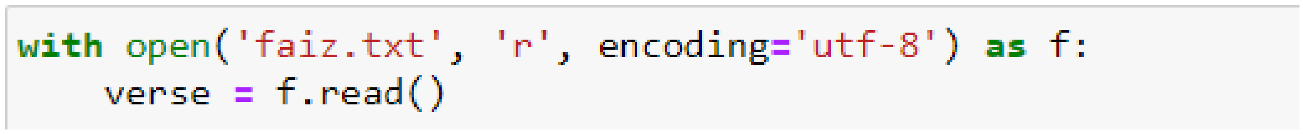
Usman Manzoor
P19-0068

Abstract

Bi-gram based text generation is a natural language processing technique that involves generating new text based on the occurrence of two consecutive words (bi-grams) in a corpus of existing text. The process involves analyzing a large body of text to identify the frequency of each bi-gram, and then using this information to generate new text that follows the same patterns.

1 Explanation

Here i explained code as given below

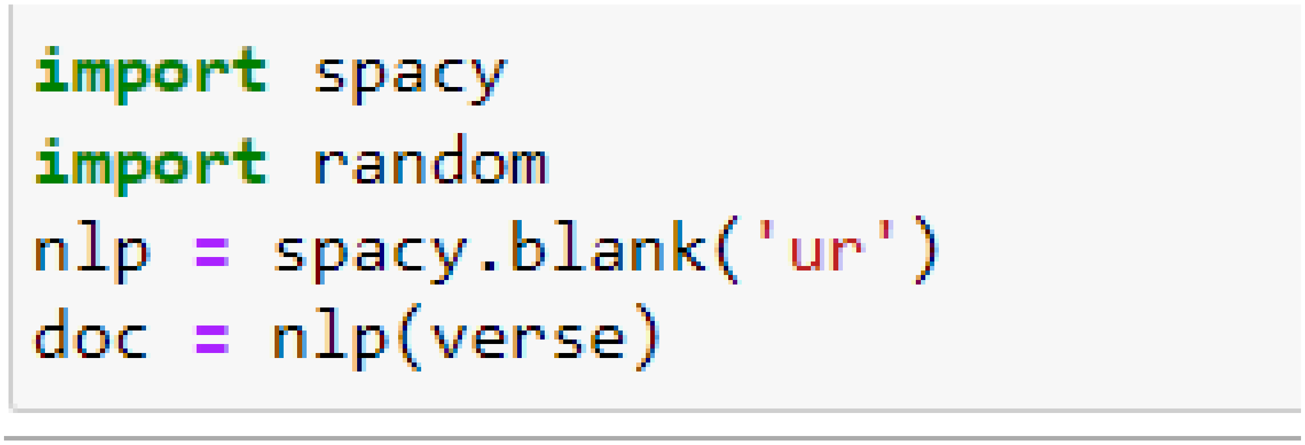


```
with open('faiz.txt', 'r', encoding='utf-8') as f:
    verse = f.read()
```

Figure 1:

This Python code opens a file named "faiz.txt" in read mode with the UTF-8 encoding. The "with" statement is used to ensure that the file is properly closed after it has been read.

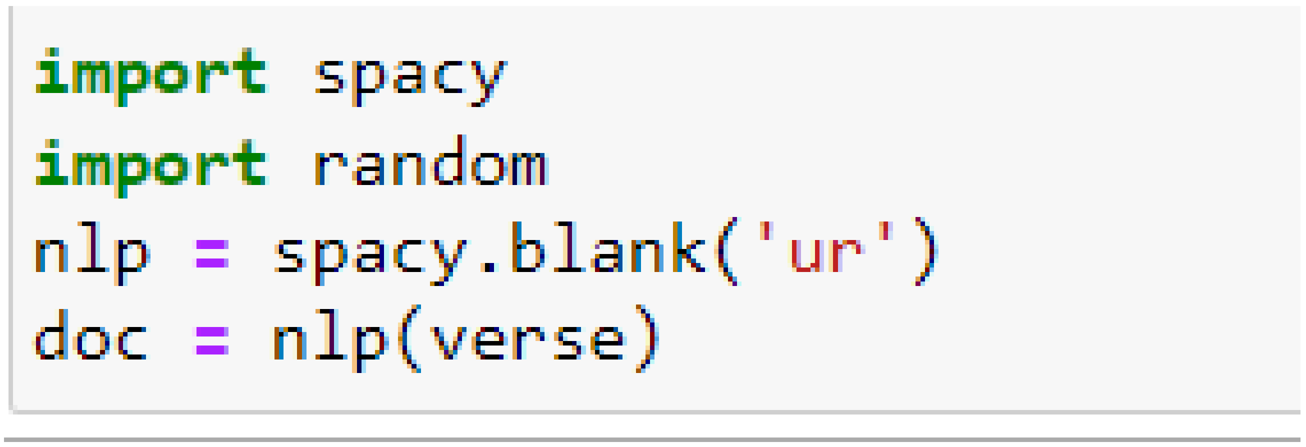
The content of the file is then read using the "read()" method and stored in the variable "verse". This variable will contain the entire text of the file as a single string.



```
import spacy
import random
nlp = spacy.blank('ur')
doc = nlp(verse)
```

Figure 2:

The Spacy library, creates a new blank Urdu language model using the "spacy.blank('ur')" method, and then processes the text contained in the "verse" variable using this model.



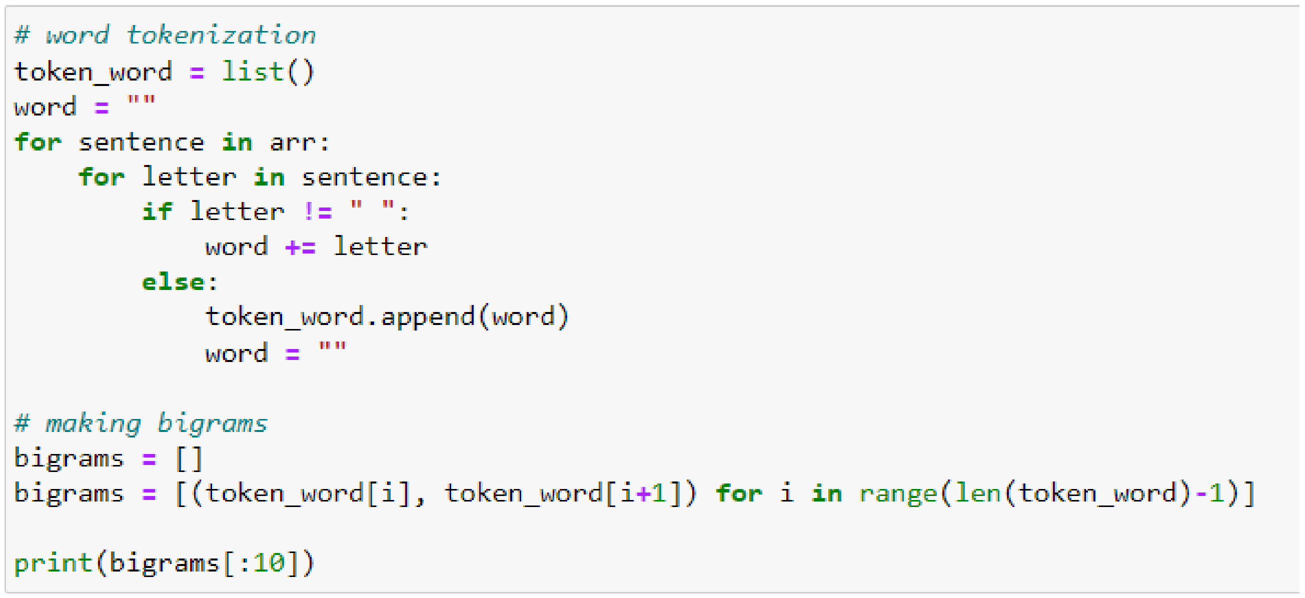
```python
import spacy
import random
nlp = spacy.blank('ur')
doc = nlp(verse)
```

Figure 3:

The "random" library is also imported, which can be used to generate random numbers or random selections from a list.

The Spacy library, creates a new blank Urdu language model using the "spacy.blank('ur')" method, and then processes the text contained in the "verse" variable using this model.

The "random" library is also imported, which can be used to generate random numbers or random selections from a list.



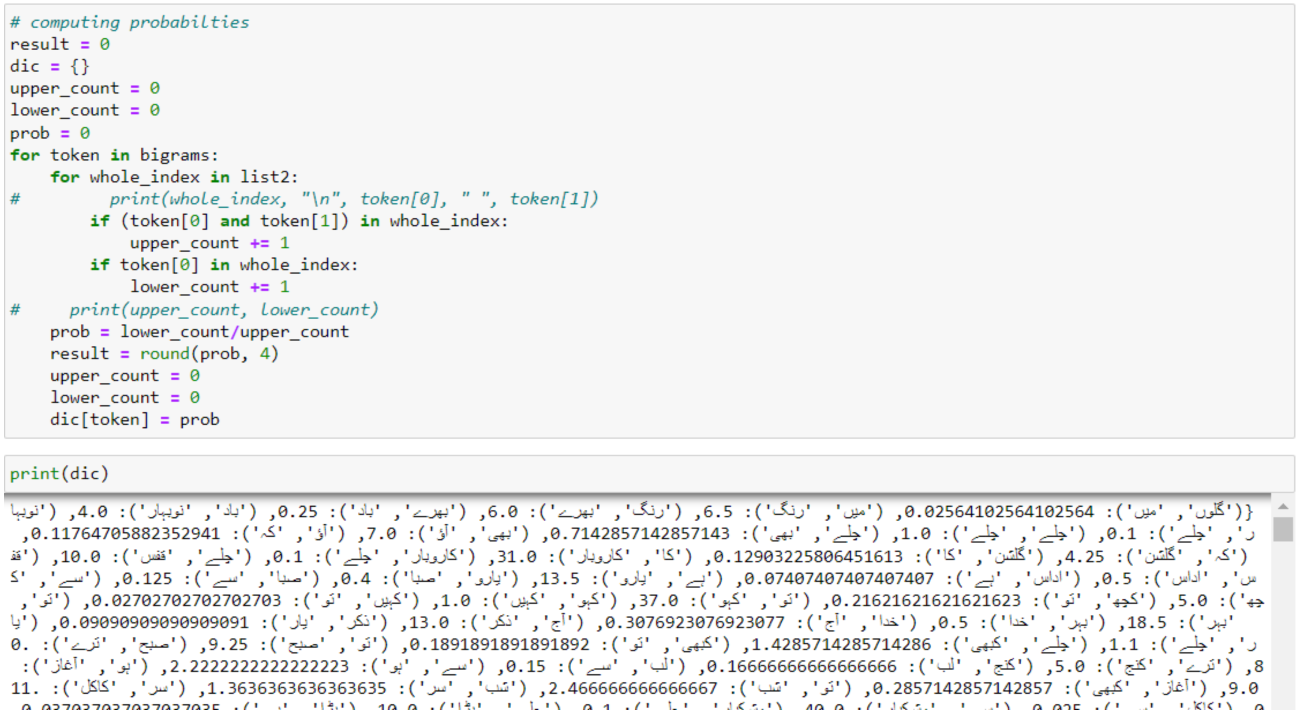
```python
# word tokenization
token_word = list()
word = ""
for sentence in arr:
    for letter in sentence:
        if letter != " ":
            word += letter
        else:
            token_word.append(word)
            word = ""

# making bigrams
bigrams = []
bigrams = [(token_word[i], token_word[i+1]) for i in range(len(token_word)-1)]

print(bigrams[:10])
```

Figure 4:

This Python code performs sentence tokenization on the Spacy document object "doc" created in the previous code snippet.



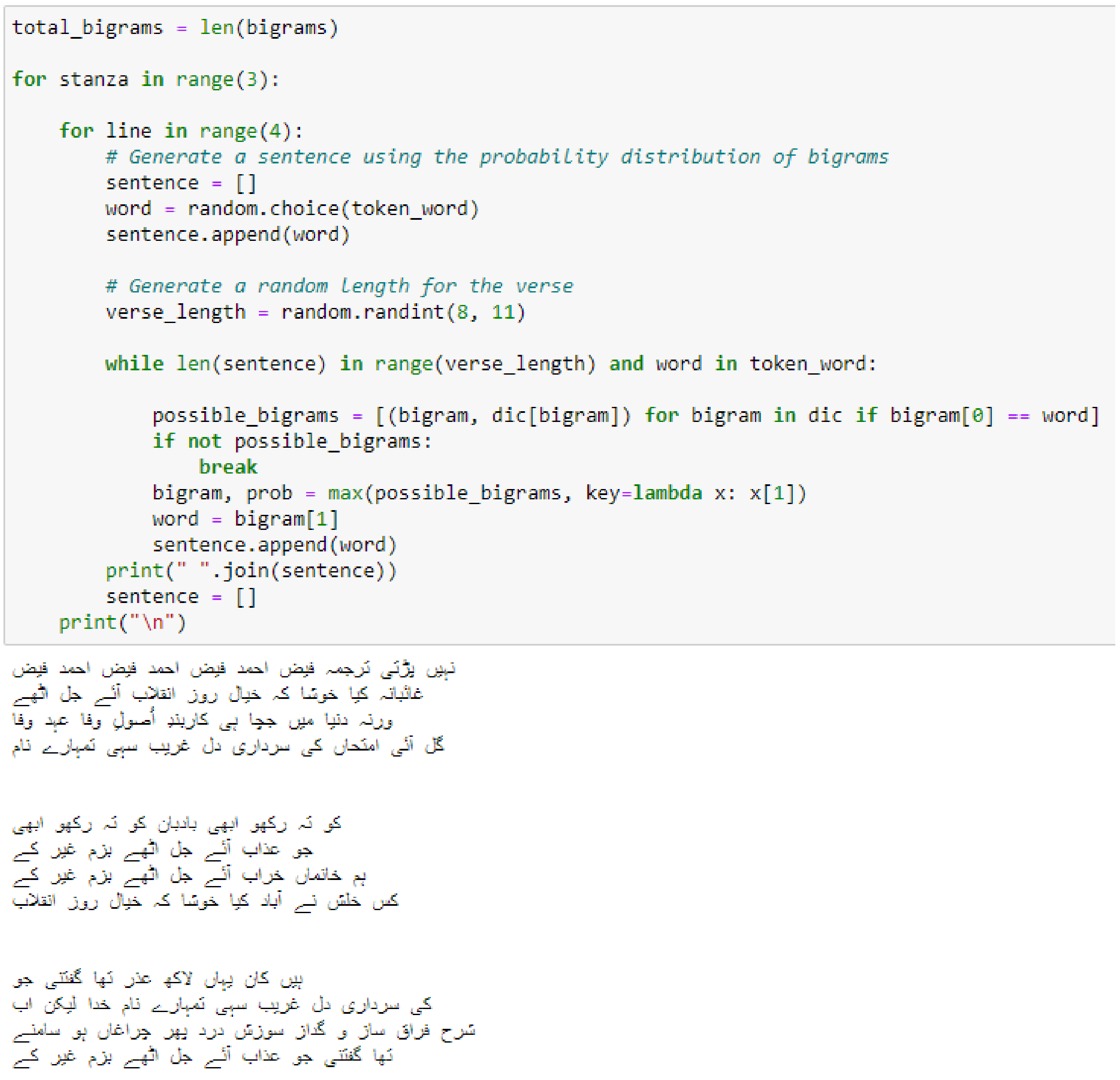
```python
# computing probabilties
result = 0
dic = {}
upper_count = 0
lower_count = 0
prob = 0
for token in bigrams:
    for whole_index in list2:
#         print(whole_index, "\n", token[0], " ", token[1])
        if (token[0] and token[1]) in whole_index:
            upper_count += 1
        if token[0] in whole_index:
            lower_count += 1
#     print(upper_count, lower_count)
    prob = lower_count/upper_count
    result = round(prob, 4)
    upper_count = 0
    lower_count = 0
    dic[token] = prob
```

```python
print(dic)
```

```
{('گلوں', 'میں'): 0.02564102564102564, ('میں', 'رنگ'): 6.5, ('رنگ', 'بھرے'): 6.0, ('بھرے', 'باد'): 0.25, ('باد', 'نوبہار'): 4.0, ('نوبہا
ر', 'چلے'): 0.1, ('چلے', 'چلے'): 1.0, ('چلے', 'بھی'): 0.7142857142857143, ('بھی', 'آؤ'): 7.0, ('آؤ', 'کہ'): 0.11764705882352941,
('کہ', 'گلشن'): 4.25, ('گلشن', 'کا'): 0.12903225806451613, ('کا', 'کاروبار'): 31.0, ('کاروبار', 'چلے'): 0.1, ('چلے', 'قفس'): 10.0, ('قف
س', 'اداس'): 0.5, ('اداس', 'ہے'): 0.07407407407407407, ('ہے', 'یارو'): 13.5, ('یارو', 'صبا'): 0.4, ('صبا', 'سے'): 0.125, ('سے', 'ک
چھ'): 5.0, ('کچھ', 'تو'): 0.21621621621621623, ('تو', 'کہو'): 37.0, ('کہو', 'کہیں'): 1.0, ('کہیں', 'تو'): 0.02702702702702703, ('تو',
'بہر'): 18.5, ('بہر', 'خدا'): 0.5, ('خدا', 'آج'): 0.3076923076923077, ('آج', 'ذکر'): 13.0, ('ذکر', 'یار'): 0.09090909090909091, ('یا
ر', 'چلے'): 1.1, ('چلے', 'کبھی'): 1.4285714285714286, ('کبھی', 'تو'): 0.1891891891891892, ('تو', 'صبح'): 9.25, ('صبح', 'ترے'): 0.
8, ('ترے', 'کنج'): 5.0, ('کنج', 'لب'): 0.16666666666666666, ('لب', 'سے'): 0.15, ('سے', 'ہو'): 2.2222222222222223, ('ہو', 'آغاز'):
9.0, ('آغاز', 'کبھی'): 0.2857142857142857, ('تو', 'شب'): 2.466666666666667, ('شب', 'سر'): 1.3636363636363635, ('سر', 'کاکل'): 11.
```

Figure 5:

This Python code computes the probability of each bigram in the "bigrams" list, based on their occurrence in a reference corpus. This is done using a simple method called maximum likelihood estimation, where the probability of a bigram is estimated as the frequency of that bigram in the reference corpus divided by the frequency of its first word.



```python
total_bigrams = len(bigrams)

for stanza in range(3):

    for line in range(4):
        # Generate a sentence using the probability distribution of bigrams
        sentence = []
        word = random.choice(token_word)
        sentence.append(word)

        # Generate a random length for the verse
        verse_length = random.randint(8, 11)

        while len(sentence) in range(verse_length) and word in token_word:

            possible_bigrams = [(bigram, dic[bigram]) for bigram in dic if bigram[0] == word]
            if not possible_bigrams:
                break
            bigram, prob = max(possible_bigrams, key=lambda x: x[1])
            word = bigram[1]
            sentence.append(word)
        print(" ".join(sentence))
        sentence = []
    print("\n")
```

```
نہیں پڑتی ترجمہ فیض احمد فیض احمد فیض احمد فیض
غالبانہ کیا خوشا کہ خیال روز انقلاب آئے جل اٹھے
ورنہ دنیا میں جچا ہی کاربند اُصولِ وفا عہد وفا
گل آئی امتحاں کی سرداری دل غریب سہی تمہارے نام


کو نہ رکھو ابھی بادبان کو نہ رکھو ابھی
جو عذاب آئے جل اٹھے بزم غیر کے
ہم خانماں خراب آئے جل اٹھے بزم غیر کے
کس خلش نے آباد کیا خوشا کہ خیال روز انقلاب


ہیں کاں یہاں لاکھ عذر تھا گفتنی جو
کی سرداری دل غریب سہی تمہارے نام خدا لیکن اب
شرح فراق ساز و گداز سوزش درد پھر چراغاں ہو سامنے
تھا گفتنی جو عذاب آئے جل اٹھے بزم غیر کے
```

Figure 6:

This Python code generates 3 stanzas of 4 lines each. The generated stanzas follow the same style and pattern as the original text, but the specific words and phrases are chosen randomly based on the probabilities of their occurrence in the reference corpus. The code first initializes a variable "total-bigrams" to store the total number of bigrams in the "bigrams" list. It then loops through each stanza (3 in total), and for each stanza, it loops through each line (4 in total). For each line, the code generates a sentence using the probability distribution of bigrams.

The code starts by initializing an empty list "sentence" to store the words in the generated sentence. It then chooses a random word from the list of all words "token-word" as the starting word for the sentence.

The code then generates a random length for the sentence, between 8 and 11 words. It uses a while loop to generate the sentence, by repeatedly choosing the next word based on the probability of the bigrams that start with the previous word. Specifically, the code first generates a list of all bigrams that start with the current word, and for each such bigram, it looks up its probability in the "dic" dictionary. It then chooses the bigram with the highest probability as the next bigram, and adds its second word to the ”sentence” list. This process continues until the desired length of the sentence is reached, or until there are no bigrams that start with the current word.

Finally, the code prints the generated sentence as a string by joining the words in the ”sentence” list with spaces. After generating all four lines in a stanza, the code prints a newline character to separate the stanzas.

2 For Bonus

for rhyme the last word of sentence i can simply store ”sentence[-1]” index to some variable and interchange it with the next generated sentence last word. this make all last words of sentences rhyming.