Iman Barati

Iran University of Science and Tech.
www.iust.ac.ir

Assignment 3 Problems

NLP : Fall 1401 : Dr. Minaei
Due Monday, Aban 30, 1401

Contents

Tokenization (30 points)

One of the important parts in natural language processing (NLP) is how Tokenization is performed on the input data. In this section, we want to learn some methods of this work and its basic concepts.

(a) (7 points)

A simple algorithm that is used for Tokenization of Chinese language is Maximum Matching Word Segmentation Algorithm. Explain this greedy algorithm and tell why it works well in Chinese but not in Persian or English. Give an example for the Persian language that does not perform well.

(b) (16 points)

In this section, we want to review two common Tokenization methods and their related details. The following two methods are considered:
1) Algorithm Byte Pair Encoding (BPE)
2) WordPiece Algorithm
Explain each of the two mentioned methods, compare with each other and describe the differences and similarities of each. To learn more about the mentioned methods, you can use the article in link.

(c) (7 points)

Explain Lemmatization and Stemming, then describe their differences with an example in the Persian language.

Language Models(Theory) (35 points)

(a) (7 points)

What is a language model? Name the types of language models and briefly explain each one.

(b) (8 points)

Consider a unigram language model over a vocabulary of size V. Suppose that a word appears m times in a corpus with M tokens in total. With Lidstone smoothing of α, for what values of m is the smoothed probability greater than the unsmoothed probability?

(c) (20 points)

In this part, you have to solve a question in Probabilistic Language Models. Consider the following table. Calculate the probability of the test data, once using unigram model and once using bigram model based on the train data and compare their results. Please write your calculations in detail. You should use Lemmatization and use Laplace Smoothing in your calculations.

Text	Data
<s> ایران سرای من است </s>	Train
<s> ایران سرای امید </s>	Train
<s> بوستان بانوان </s>	Train
<s> آبی آسمان دخترانه است </s>	Train
<s> علی دایی اسطوره ایران است </s>	Train
<s> آسمان آبی است </s>	Test
<s> دختر دایی امید ایرانی است </s>	Test
<s> ایران بانو در سرای بوستان است </s>	Test
<s> سرای من ایران است </s>	Test

Language Models(Code) (45 points)

In this section, you will build a simple language model that can be used to generate random text resembling a source document.

A skeleton file homework3.ipynb containing empty definitions for each question has been provided. Since portions of this assignment will be graded automatically, none of the names or function signatures in this file should be modified. However, you are free to introduce additional variables or functions if needed.

You may import definitions from any standard Python library, and are encouraged to do so in case you find yourself reinventing the wheel.Your use of external code should be limited to built-in Python modules, which excludes, for example, NumPy and NLTK.

You will find that in addition to a problem specification, most programming questions also include a pair of examples. These are meant to illustrate typical use cases, and should not be taken as comprehensive test suites.

You are strongly encouraged to follow the Python style guidelines set forth in PEP 8, which was written in part by the creator of Python. However, your code will not be graded for style.

(a) (2 points)

Write a simple tokenization function **tokenize(text)** which takes as input a string of text and returns a list of tokens derived from that text. Here, we define a token to be a contiguous sequence of non-whitespace characters, with the exception that any punctuation mark should be treated as an individual token. Hint: Use the built-in constant *string.punctuation*, found in the *string* module.

```
[ ]  tokenize("منم کاوه دادخواه یکی بی‌زبان مرد آهنگرم")

     ['منم', 'کاوه', 'دادخواه', 'یکی', 'بی', 'زبان', 'مرد', 'آهنگرم']
```

```
[ ]  tokenize("I'm Kaveh, and a blacksmith, sire")

     ['I', 'm', 'Kaveh', ',', 'and', 'a', 'blacksmith', ',', 'sire']
```

(b) (5 points)

Write a function **ngrams(n, tokens)** that produces a list of all n-grams of the specified size from the input token list. Each n-gram should consist of a 2-element tuple (context, token), where the context is itself an (n - 1)-element tuple comprised of the n - 1 words preceding the current token. The sentence should be padded with n - 1 "$\langle START \rangle$" tokens at the beginning and a single "$\langle END \rangle$" token at the end. If n = 1, all contexts should be empty tuples. You may assume that n $\geq$ 1.

```
[ ]  ngrams(1, ["دادخواه" ,"آهنگر" ,"کاوه"])

     [((), 'کاوه'), ((), 'آهنگر'), ((), 'دادخواه'), ((), '<END>')]
```

```
[ ]  ngrams(2, ["دادخواه" ,"آهنگر" ,"کاوه"])

     [(('<START>',), 'کاوه'),
      (('کاوه',), 'آهنگر'),
      (('آهنگر',), 'دادخواه'),
      (('دادخواه',), '<END>')]
```

```
[ ]  ngrams(3, ["دادخواه" ,"آهنگر" ,"کاوه"])

     [(('<START>', '<START>'), 'کاوه'),
      (('<START>', 'کاوه'), 'آهنگر'),
      (('آهنگر' ,'کاوه'), 'دادخواه'),
      (('دادخواه' ,'آهنگر'), '<END>')]
```

(c) (5 points)

In the **NgramModel** class, write an initialization method **__init__(self, n)** which stores the order n of the model and initializes any necessary internal variables. Then write a method **update(self, sentence)** which computes the n-grams for the input sentence and updates the internal counts. Lastly, write a method **prob(self, context, token)** which accepts an (n - 1)-tuple representing a context and a token, and returns the probability of that token occuring, given the preceding context.

```
[ ] m = NgramModel(1)
    m.update("کاوه آهنگر ضحاک ستمگر")
    m.update("کاوه آهنگر کاوه آهنگر")
    print(m.prob((), "کاوه"))
    print(m.prob((), "ضحاک"))
    print(m.prob((), "<END>"))
```

```
0.3
0.1
0.2
```

```
[ ] m = NgramModel(2)
    m.update("کاوه آهنگر ضحاک ستمگر")
    m.update("کاوه آهنگر کاوه آهنگر")
    print(m.prob(("<START>",), "کاوه"))
    print(m.prob(("آهنگر",), "ضحاک"))
    print(m.prob(("کاوه",), "ماردوش"))
```

```
1.0
0.3333333333333333
0.0
```

(d) (10 points)

In the **NgramModel** class, write a method **random_token(self, context)** which returns a random token according to the probability distribution determined by the given context. Specifically, let T=$\langle t_1, t_2, ..., t_n \rangle$ be the set of tokens which can occur in the given context, sorted according to Python's natural lexicographic ordering, and let $0 \leq r < 1$ be a random number between 0 and 1. Your method should return the token t_i such that

$$\sum_{j=1}^{i-1} P(t_j \mid \text{context}) \leq r < \sum_{j=1}^{i} P(t_j \mid \text{context}).$$

You should use a single call to the **random.random()** function to generate r.

```
[ ] m = NgramModel(1)
    m.update("کاوه آهنگر ضحاک ستمگر")
    m.update("کاوه آهنگر کاوه آهنگر")
    random.seed(1)
    [m.random_token(()) for i in range(8)]
```

```
['<END>', 'کاوه', 'ضحاک', 'آهنگر', 'آهنگر', 'آهنگر', 'کاوه', 'کاوه']
```

```
[ ] m = NgramModel(2)
    m.update("کاوه آهنگر ضحاک ستمگر")
    m.update("کاوه آهنگر کاوه آهنگر")
    random.seed(2)
    print([m.random_token(("<START>",)) for i in range(6)])
    print([m.random_token(("آهنگر",)) for i in range(6)])
```

```
['کاوه', 'کاوه', 'کاوه', 'کاوه', 'کاوه', 'کاوه']
['کاوه', '<END>', 'ضحاک', 'ضحاک', 'ضحاک', '<END>']
```

(e) (10 points)

In the **NgramModel** class, write a method **random_text(self, token_count)** which returns a string of space-separated tokens chosen at random using the **random_token(self, context)** method. Your starting context should always be the (n - 1)-tuple ("⟨START⟩", ..., "⟨START⟩") , and the context should be updated as tokens are generated. If n = 1, your context should always be the empty tuple. Whenever the special token "⟨END⟩" is encountered, you should reset the context to the starting context.

```
[ ] m = NgramModel(1)
    m.update("کاوه آهنگر ضحاک ستمگر")
    m.update("کاوه آهنگر کاوه آهنگر")
    random.seed(1)
    m.random_text(16)
```

```
'آهنگر کاوه <END> کاوه آهنگر کاوه <END> <END> کاوه کاوه آهنگر آهنگر آهنگر ضحاک کاوه <END>'
```

```
[ ] m = NgramModel(2)
    m.update("کاوه آهنگر ضحاک ستمگر")
    m.update("کاوه آهنگر کاوه آهنگر")
    random.seed(2)
    m.random_text(16)
```

```
'کاوه آهنگر کاوه <END> کاوه آهنگر ضحاک ستمگر <END> کاوه آهنگر کاوه آهنگر کاوه <END> کاوه آهنگر'
```

(f) (6 points)

Write a function **create_ngram_model(n, path)** which loads the text at the given path and creates an n-gram model from the resulting data. Each line in the file should be treated as a separate sentence.

```
[ ] # No random seeds, so your results may vary
    m = create_ngram_model(1, "Shahnameh.txt")
    m.random_text(16)
```

```
'had wrote was your Kashmir don Rostam is loaded sun over Shah corpses he , ,'
```

```
[ ] # No random seeds, so your results may vary
    m = create_ngram_model(2, "Shahnameh.txt")
    m.random_text(16)
```

```
'Then Bahram s brother , The camel - echoed to cry out a voice was poisoned'
```

```
[ ] # No random seeds, so your results may vary
    m = create_ngram_model(3, "Shahnameh.txt")
    m.random_text(16)
```

```
'heard the popular diffusion of the splendid and honorable king because you were still feasting Afrasyab'
```

```
[ ] # No random seeds, so your results may vary
    m = create_ngram_model(4, "Shahnameh.txt")
    m.random_text(16)
```

```
'KHOSROW FIGHTS AGAINST BAHRAM CHUBINEH 893 <END> made the bridges and roads as flat as the'
```

(g) (7 points)

Suppose we define the perplexity of a sequence of m tokens $<w_1, w_2, ..., w_m>$ to be

$$\sqrt[m]{\frac{1}{P(w_1, w_2, \cdots, w_m)}}.$$

For example, in the case of a bigram model under the framework used in the rest of the assignment, we would generate the bigrams $<(w_0 = <START>, w_1), (w_1, w_2), , (w_m - 1, w_m), (w_m, w_m + 1 = <END>)>$, and would then compute the perplexity as

$$\sqrt[m+1]{\prod_{i=1}^{m+1} \frac{1}{P(w_i \mid w_{i-1})}}.$$

Intuitively, the lower the perplexity, the better the input sequence is explained by the model. Higher values indicate the input was "perplexing" from the model's point of view, hence the term perplexity.
In the **NgramModel** class, write a method **perplexity(self, sentence)** which computes the n-grams for the input sentence and returns their perplexity under the current model. *Hint: Consider performing an intermediate computation in log-space and re-exponentiating at the end, so as to avoid numerical overflow.*

```
[ ]  m = NgramModel(1)
     m.update("کاوه آهنگر ضحاک ستمگر")
     m.update("کاوه آهنگر کاوه آهنگر")
     m.perplexity("کاوه آهنگر")
```

```
3.815714141844439
```

```
[ ]  m = NgramModel(2)
     m.update("کاوه آهنگر ضحاک ستمگر")
     m.update("کاوه آهنگر کاوه آهنگر")
     m.perplexity("کاوه آهنگر")
```

```
1.4422495703074085
```

Notes

Codes should be implemented in .ipynb format (notebooks)
• All Code cells should be executed before turning in the assignment (Make sure your outputs are there before you submit your assignment)
• Please explain the code and the results in the document or notebook
• If you have any questions, feel free to ask. You can ask your questions in the Telegram group.
• Please upload your assignments as a zipped folder with all necessary components. Upload your file in HW3-NLP-YourStudentID-YourName.zip format.