CS4248 Natural Language Processing
Assignment 1:
Regexes and Language Models

Distributed on 02 Feb 2024
Due in Canvas `Files` by 17 Feb 2024 11:59 PM SGT

This assignment contributes 10 marks towards your final mark for the class, and is graded out of a rubric of 100 points.

Instructions

1. Download the according scaffold submission `.zip` file from Canvas. Rename it as per directed (you may need to rename both the folder and individual files; please refer to the `README.md` file.

2. In the `writeup.pdf` file, write your answers for each question according to its requirements (for plots or sample displays, please attach a screenshot). There is a template writeup in the `.zip` folder for your reference. Make sure to include your writeup as a `.pdf` file, as we will not process other word processing or raw text formats.

3. For programming tasks, complete the skeleton code files (which start with `obj` in the `codes` folder) based on the requirements of each question. Please strictly follow the instructions in the questions and the comments in the skeleton codes. We will run your code with several test cases to verify it. We also add some codes for you to run and test your implementations. For the methods which are not included in these codes, you can adjust the input/output formats according to your implementations.

4. Please use **Python ==3.10**. Additionally, please ensure that all codes submitted can be run directly on our machines locally. Examples of good code practices include ensuring that filepaths are relative so that we can run directly without having to edit your code, and no other external functions like `nltk` are being imported and implemented anywhere in your code. If the graders find themselves having to debug/edit a large amount of your code, it can result in a 30% deduction in your score.

5. To submit, remove the `data` folder and only keep the `codes` folder, the `plots` folder, and the `writeup.pdf`. Wrap them up in a `.zip` file and submit it to the "`Assignment #1 Submission`" folder on Canvas. Remember to rename both the writeup document and the `.zip` file with your matriculation number (*e.g.*, `A0000000X.pdf`; `A0000000X.zip`).

Integrity Note. *Since this assignment is similar to other assignments for Natural Language Processing courses at other institutions, there are (undoubtedly) solutions posted somewhere. Under the NUS Code of Conduct, you must follow class policy in working on this individual assignment. When in doubt of whether an action would constitute a violation of policy, please ask us on* `Canvas` *in the respective Assignment header or by emailing the instructors,* **before** *attempting the action.*

1 Programming Questions

[Questions 1–3] Complete the skeleton code provided to accomplish tasks in the questions. Functions implemented in different questions can be utilized or adapted for use in the following exercises. Please also strictly follow the requirements written in the comments of the skeleton code, as we will run your codes to verify the outputs. For example, do not modify the function signature of each of the functions mentioned on the top of the code files. To test your implementations, you can run the codes using the commands we additionally provide at the bottom of each code file.

1. **[15% + 2%] - Regular Expressions**
   Searching for regular expressions is important in many applications, such as text retrieval and text editing. In this task, we utilize regular expressions to retrieve words of interest. To start, we'll need to create suitable regular expressions to capture these scenarios of interest. Accomplish the following tasks and enter the expressions respectively in the provided skeleton code `obj1_regex.py`.

   A. Write a regular expression R_1 to match all the strings that begin and end with the same character except for space (*e.g.* `'cbc'`, `'01230'`), and the strings must be more than 1 in length (*i.e.*, `'a'` should not be matched).

   B. Write a regular expression R_2 to **avoid** matching any string that has a substring (including itself) with the format of `abba`, here 'a' and 'b' can be any characters except for space (*e.g.*, `'otto'`, `'-++-' should not be matched, while 'abcd' should be matched.`). To make things simple, we are now creating a regular expression to avoid a 4 character 'abba' string/substring. Note that 'bb' means repeated characters.

   C. Write a regular expression R_3 to meet both of the previous requirements. Can you find another regex R_4 to construct R_1 together with R_3 as $R_1 = R_3 \cup R_4$? Give the R_4 and describe the relations between R_3, R_4 and R_1 in the finite state automaton.

      Write their relations in the writeup and enter the expressions for R_3 and R_4 into the code file. You do not need to provide a diagram in the writeup.

   D. **[Bonus 2%]** Write a regular expression R_5 to identify emoticons given in the text `../data/emoticons.txt`. To be clear, we only concern ourselves in this problem with the emoticon types listed in the file.

   Note that you may run the provided, non-exhaustive mini-tests in the skeleton to test out your answers, but passing all the tests does not guarantee full marks, and you are encouraged to write your own tests.

2. **[25%] - Tokenization, Zipf's Law**
   Go to Project Gutenberg and download the text of the book *Pride and Prejudice* (in plain text). Then complete the skeleton code `obj2_tokenizer.py` to accomplish the following tasks. The `init` method of class `Tokenizer` must handle following arguments:

   - `bpe` (enumerated value: `YES` or `NO`; default `NO`): if set to `YES`, then you need to do byte pair encoding, otherwise use regular expressions for tokenization.
   - `lowercase` (enumerated value: `YES` or `NO`; default `YES`): if set to `YES`, then you need to convert the entire text to lowercase.
   - `path` (string value): the path of the corpus file to load and get the vocabulary.

   As an illustrative example, your class could be run through a command line structure which would be passed arguments on the command line. Do note that we will be running your class *only* through the class methods as specified in the scaffold file; this is just an illustration.

   ```
   OBJ2 path=STRING bpe=YES/NO lowercase=YES/NO

   Example: OBJ2 path=../data/Pride_and_Prejudice.txt bpe=NO lowercase=YES
   ```

   Your tokenizer should be able to perform the following tasks. Write up any of the requested data or justifications in `writeup.pdf`:

   A. Construct the tokenizer based on the text of the book with the default settings. Plot a figure of relative frequency versus rank (use logs for both). Is the figure (somewhat) consistent with Zipf's law? Recall Zipf's law states that the relative frequency of a word type (f_c^i) is inversely proportional to its rank i: $f_c^i \propto \frac{1}{i}$. Title your plot `2-A`. Attach a plot in the writeup and answer the question in the writeup.

   B. Turn on the `bpe` setting, and repeat the first task (you are suggested to limit the number of `bpe` iterations to let it finish within a reasonable amount of time). Are the results still consistent with Zipf's law? Attach a plot in your writeup with your finding, titling your plot `2-B`.

   Additional Clarifications. Please refer to the comments in `obj2_tokenizer.py:tokenize()` for detailed requirements on tokenization. For any ambiguous and exceptional cases, you may check whether your tokenization works similar to NLTK or the spaCy library's tokenization.

3. **[30%] - Language Modeling, Regular Expressions**
   Given a corpus (such as what you used in the last question), you will implement a class `NgramLM` in the code file `obj3_ngram_lm.py`, which can be used to train a language model. The class takes in the following arguments:

   - `path`: filesystem path to the input file that we should construct the language model from.
   - `smooth` (string; default `'add-k'`): type of smoothing to use. Here by default you need to handle `add-k` smoothing; handling other forms is optional.
   - `n_gram` (integer; choose from `[1, 2]`): number of grams for constructing the language model. *e.g.*, 1 would correspond to a unigram model, 2 to a bigram model.
   - `k` (float; default `1.0`): amount of smoothing to apply to the model. Take note to add this float to the counts of individual words. Make sure you handle corner cases (such as when `k=0`).
   - `text` (string): text/prompt on which you run your language model and predict the next word(s). Try to limit your text/prompt to 1024 tokens so as to prevent exceeding the time limit!

   An illustrative command line invocation could be of the form:

   ```
   OBJ3 path=STRING smooth=TYPE n_gram=INT k=FLOAT text=STRING
   ```

   Example Setting:

   ```
   OBJ3 path=../data/Pride_and_Prejudice.txt smooth=add-k n_gram=2 k=1.0 text="They had now entered a beautiful walk by"
   ```

   After suitable tokenization (you may reuse the tokenizer from last question, or make use of tokenizers from external libraries such as NLTK and spaCy) and normalization, input your processed corpus into your language model method to perform the following tasks. Document any of the requested data or justifications in an appropriate section of your `writeup.pdf`:

   A. Complete the skeleton code in `obj3_ngram_lm.py` which should train a language model on the specified input text. Please strictly follow the instructions (written in the comments) about the output formats to construct the methods of class `NgramLM` in the code file `obj3_ngram_lm.py`. Note that in the scaffold file we have provided a number of suggested methods that you can optionally implement, but only the methods `generate_word`, `generate_text` and `get_perplexity` are necessary to implement as-is (as also marked in the scaffold code).

   B. Utilize the three sentences given at the beginning of `obj3_ngram_lm.py` (in the comments) as input and get the corresponding outputs for calls to the functions `generate_word` and `get_perplexity`. Also get the outputs of the function `generate_text` for different input lengths, *e.g.*, 3-10 (you will need a start token to initialize the generated text). Include samples of input–output pairs that use your methods in your `writeup.pdf`.

   Test your code and make sure the given tests finish within a minute. If we failed to compile or run your code under the time limit, you will receive **zero** marks!

2 Theory Questions

This section gives you the chance to practice and reinforce your understanding of the lecture's theoretical material. Write the answers to each question in your `writeup.pdf`, where each explanation is presented prefixed with a suitable header (e.g., Question 4-A).

4. **[30%] - Language Modeling**.

   A language model consists of a vocabulary V, and a function $p(x_1 \dots x_n)$ such that for all sentences $x_1 \dots x_n \in V^+$, $p(x_1 \dots x_n) > 0$, and in addition $\sum_{x_1 \dots x_n \in V^+} p(x_1 \dots x_n) = 1$. Here V^+ is the set of all sequences $x_1 \dots x_n$ such that $n \geq 1$, $x_i \in V$ for $i = 1 \dots (n-1)$, and x_n = **STOP**.

   We assume that we have a bigram language model, with:

   $$p(x_1, \dots x_n) = \Pi_{i=1}^{n} q(x_i | x_{i-1})$$

   The parameters $q(x_i|x_{i-1})$ are estimated from a training corpus using a discounting method, with discounted counts:

   $$c^*(v, w) = c(v, w) - \beta$$

   where $\beta = 0.5$. From lecture we also know the definition of perplexity $PP(W)$ for a test set $W = w_1 w_2 ... w_n$. $PP(W)$ is defined as:

   $$PP(W) = P(w_1 w_2 ... w_n)^{-\frac{1}{N}}$$

   We assume in this question that all words seen in any test corpus are in the vocabulary V, and each word in any test corpus has been seen at least once during training.
   With these assumptions, please answer the following questions:

   A. True or False?

      For any test corpus, the perplexity score $PP(W)$ under the language model will be less than ∞.

      Please first give a mathematical proof and then give an intuitive interpretation of your observation of natural language.

   B. True or False?

      For any test corpus, the perplexity under the language model will be at most $N + 1$, where N is the number of words in the vocabulary V. Please first give a mathematical proof and then interpret its intuition in language.

   C. Now consider a bigram language model where for every bigram (v, w) where $w \in V$ or $w =$ **STOP**, $q(w|v) = 1/(N+1)$, where N is the number of words in the vocabulary V.

      True or False?

      For any test corpus, the perplexity under the language model will be equal to $N + 1$.

      Please first give a mathematical proof and then interpret its intuition in language.

Students, you must include the text of the two statements below in your submitted work as part of your `writeup.pdf` *and digitally sign your homework using your Student ID number (starting with* `A...`*; N.B., not your NUSNET email identifier). Make sure you have attached this statement to your submission either in written or typed form.*

Delete (and where appropriate, fill in) one of the two forms of Statement 1:

1A. Declaration of Original Work. *By entering my Student ID below, I certify that I completed my assignment independently of all others (except where sanctioned during in-class sessions), obeying the class policy outlined in the introductory lecture. In particular, I am allowed to discuss the problems and solutions in this assignment, but have waited at least 30 minutes by doing other activities unrelated to class before attempting to complete or modify my answers as per the Pokémon Go rule.*

1B. Exception to the Class Policy. *I did not follow the CS4248 Class Policy in doing this assignment. This text explains why and how I believe I should be assessed for this assignment given the circumstances explained.*

Signed, [Enter your A0000000X Student ID here]

2. References *I give credit where credit is due. I acknowledge that I used the following websites or contacts to complete this assignment (but please note that many uses of Web search and detailed discussion are not allowed:*

- *Sample. Website 1, for following mathematical proofs.*
- *Sample. My friend, A0000001Y, whom helped me figure out the course deadlines*