CS4248 Natural Language Processing Assignment 1: Regexes and Language Models

Distributed on 27 Jan 2022
Due in LumiNUS `Files` by 10 Feb 2022 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 LumiNUS. 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.
4. 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 LumiNUS. Remember to rename both the writeup document and the `.zip` file with your matriculation number (*e.g.*, `A000000X.pdf`; `A000000X.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* `LumiNUS` *in the respective Assignment header or by emailing the instructors,* **before** *attempting the action.*

1 Programming

[Questions 1–4] 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.

1. **[10%] - 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 `obj1_tokenizer.py` to accomplish the following tasks. The `init` method of class `Tokenizer` must handle the commands below:

   ```
   OBJ1 path=STRING bpe=YES/NO lowercase=YES/NO

   Example: OBJ1 path=../data/Pride_and_Prejudice.txt bpe=NO lowercase=YES
   ```

   Argument Explanation:

   - `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.

   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 rank of the corpus 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 `1-1-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. Are the results still consistent with Zipf's law? Attach a plot in your writeup with your finding, titling your plot `1-1-B`.

2. **[20%] - Regular Expressions, Edit Distance**
   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 three scenarios of interest. Accomplish the following three tasks in your writeup.

   A. Write a regular expression R_1 to match all the words that begin with the same letter it starts with in the reverse order, *i.e.*, the word starts and ends with the same letter (*e.g.*, `'cbc'`, `'01230'`) and these two letters should be at different positions (*e.g.*, `'a'` should not be matched).

   B. Write a regular expression R_2 to **avoid** matching any word that has the format of `abba`, here 'a' and 'b' can be any characters except for space (*e.g.*, `'otto'`, `'trillion'`, `'xxxx'`, `'-++-'`).

   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 your answers (R_4 and the relations) in your writeup.

   Now define the expressions of our interests as R_3, let us utilize it for text retrieval. One problem in actual practice is that there is noise in the text (*e.g.*, typographical errors) that makes it difficult for us to retrieve them. So instead of exactly matching, we will allow some distance between the text and our target. In class, we have learned the algorithm of minimal edit distance calculating. Based on this, complete the skeleton code `obj2_edit_distance.py` to accomplish the tasks below:

   A. Implement the class `EditDistanceCalculator` to calculate the minimal edit distance (Levenshtein). The method must handle the command as follows:

   ```
   OBJ2_A source=STRING target=STRING
   ```

   Here, we edit based on the source string and aim to get the target string.

   B. The algorithm we have implemented just deals with the distance between text. Now, set our target as the regex R_3 you obtained from the previous Task 2-1-C, and implement a method to retrieve any word that can be matched by R_3 within an edit distance of k. Implement the class `RegexChecker` for this task. You can choose to utilize the class `EditDistanceCalculator` you have just implemented or not. In general, the core method should be able to handle the commands as follows:

   ```
   OBJ2_B word=STRING k=INT

   Example: OBJ2_B word=token k=2
   ```

   Here, specifically we approximate the distance from the word (as source) to the regex R_3 (as target). Briefly justify how your method works in the `writeup.pdf`.

3. **[15%] - Regular Expressions**

   Emoticons are widely used in our online social interactions (*e.g*, Email, Twitter). Unlike emojis, emoticons can be easily displayed in a pure, raw text corpus. This makes it convenient for us to analyze them. Some studies have shown that emoticons can be important in NLP applications such as sentiment analysis. Let's try this with the social medium of tweets.

   Given the file of `data/labeled_tweets.json`, complete the skeleton code file `obj3_sentiment.py` to accomplish the following tasks:

   A. Utilize regular expressions to identify emoticons in the text (in the class `SentimentClassifier`). To simplify this task, we have provided a emoticon list in file `data/emoticons.txt`. Your solution only need to work with the emoticons in this file. You may adapt the functions you've implemented in `Question 1` to do appropriate tokenization first. Plot a figure of relative frequency of each emoticon versus rank (use logarithms). Is the figure (somewhat) consistent with Zipf's law? Attach the plot in the writeup and answer the question in the writeup.

   B. Browse and understand the `classify_sentiment` method in the `SentimentClassifier` class. In a nutshell, you will observe that it determines the polarity of a tweet based on the sentiment label of each non-emoticon word in the text (provided in `data/lexicon_sentiment.txt`). However, we haven't utilized the information from emoticons, so the accuracy of classification (as shown by the function `cal_accu`) is quite low.

   C. Based on the current solution, write regex(es) to categorize emoticons with different polarity tags (*i.e.*, positive, negative, neutral) and utilize it to improve classification performance. Compare the label distribution with the labels we provide in `data/labeled_tweets.json`. Can the emoticons correct the error cases?
   Also tune the value of `threshold` manually by trying different values. How does it affect the predictions, both with and without emoticons, respectively?
   Reflect on these questions in your writeup.

4. **[25%] - Language Modeling, Regular Expressions**
   Given a corpus (such as what you used in Question 1), you'll need to develop a class `NgramLM` in the code file `obj4_ngram_lm.py`, which can be used to train a language model. The class construction must handle the following variables:

   ```
   OBJ4 path=STRING smooth=TYPE n_gram=INT k=FLOAT text=STRING
   ```

   Argument Explanation:

   - `path` gives the filesystem path to the input file that we should construct the language model from.
   - `smooth` (string; default `add-k`) selects what type of smoothing to use. Here by default you need to handle `add-k` smoothing, handling other forms is optional.
   - `n_gram` (integer; choices `[1, 2]`) specifies the number of grams for constructing the language model. *e.g.*, 1 would correspond to a unigram model, 2 to a bigram model.
   - `k` (floating point number; default `1.0`) specifies the amount of smoothing to apply to the model. Take note to add this float to the counts of individual words.
   - `text` (string) specifies the (short) text/prompt on which you run your language model and predict the next word(s).

   Example Setting:

   ```
   $ OBJ4 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 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 `obj4_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 `obj4_ngram_lm.py`.

   B. Utilize the three sentences given at the beginning of `obj4_ngram_lm.py` (in the comments) as input and get the corresponding outputs for calls to the functions `generate_word`, `generate_text`, `perplexity`). Include samples of input–output pairs that use your methods in your `writeup.pdf`.

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 5-A).

5. **[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*