Assignment 1: Language Modelling and Tokenization

Course: Introduction to Natural Language Processing

Deadline: January 28th, 2024 | 23:59

1 General Instructions

1. The assignment must be implemented in Python.
2. No standard library for calculating n-grams, tokenization or LMs should be used.
3. Submitted assignment must be your original work. Please do not copy any part from any source including your friends, seniors, and/or the internet. If any such attempt is caught, then serious actions including an F grade in the course is possible.
4. A single .zip file needs to be uploaded to the Moodle Course Portal.
5. Your grade will depend on the correctness of answers and output. In addition, due consideration will be given to the clarity and details of your answers and the legibility and structure of your code.
6. Please start early since no extension to the announced deadline would be possible.

2 Tokenization

You have been given two corpuses for cleaning. Your task is to design a tokenizer using regex, which you will later use for smoothing and language modelling as well.

1. Create a Tokenizer to handle following cases:
   (a) Sentence Tokenizer: Divides text into sentences.
   (b) Word Tokenizer: Splits sentences into individual words.
   (c) Numbers: Identifies numerical values.
   (d) Mail IDs: Recognizes email addresses.
   (e) Punctuation: Detects punctuation marks.
   (f) URLs: Identifies website links.
   (g) Hashtags (#omg): Recognizes social media hashtags.
   (h) Mentions (@john): Recognizes social media mentions.
2. For the following cases, replace the tokens with appropriate placeholders:
   (a) URLS: <URL>
   (b) Hashtags: <HASHTAG>
   (c) Mentions: <MENTION>
   (d) Numbers: <NUM>
   (e) Mail IDs: <MAILID>

You are also encouraged to try other tokenization and placeholder substitution schemes based on your observations from the corpora used for the smoothing task to achieve a better language model. You may find percentages, age values, expressions indicating time, time periods occurring in the data.
You're free to explore and add multiple such reasonable tokenization schemes in addition from the ones listed above. Specify any such schemes you use in the final README.

3 N-Grams

Design a function in Python that takes a value of 'N' and the <corpus_path> and generates an N-sized N-gram model from both the given corpus.

4 Smoothing and Interpolation

You have been given two corpora: “Pride and Prejudice” and “Ulysses”. Your task is to design Language Models for both using smoothing techniques. Be sure to use the tokenizer created in the Tokenization task.

1. Create language models with the following parameters:
   (a) On “Pride and Prejudice” corpus:
      i. LM 1: Tokenization + 3-gram LM + Good-Turing Smoothing
      ii. LM 2: Tokenization + 3-gram LM + Linear Interpolation

   (b) On “Ulysses” corpus:
      i. LM 3: Tokenization + 3-gram LM + Good-Turing Smoothing
      ii. LM 4: Tokenization + 3-gram LM + Linear Interpolation
2. For each of these corpora, create a test set by randomly selecting 1000 sentences. This set will not be used for training the LM.
   (a) Calculate perplexity score for each sentence of “Pride and Prejudice” corpus and “Ulysses” corpus for each of the above models and also get average perplexity score on the train corpus.

   (b) Report the perplexity scores for all the sentences in the training set. Report the perplexity score on the test sentences as well, in the same manner above

Note: Good-Turing Smoothing is a technique used to adjust the probability distribution of unseen events in N-gram models, while Linear Interpolation is a method of combining different N-gram models (like unigram, bigram, trigram) to get a better estimate of the probabilities. For more details, refer to the resources section at the end.

5 Generation

You should be able to generate text from a given input using each of the Language Models that you’ve developed. This task is commonly known as Next Word Prediction.

This means, that for a given language model, when you give it an input, it'll use the input context to generate the most likely word that comes next, or the word with the highest probability.

1. Using the generated N-gram models (without the smoothing techniques), try generating sequences. Experiment with different values of N and report which models perform better in terms of fluency.

2. Attempt to generate a sentence using an Out-of-Data (OOD) scenario with your N-gram models. Analyze and discuss the behavior of N-gram models in OOD contexts.

3. Now try to generate text using the models with the smoothing techniques (LM1, LM2, LM3, LM4).

Provide the analysis along with the examples for each of the points in the README.

6 Corpus

The following corpora have been given to you for training:

1. Pride and Prejudice corpus (1,24,970 words)

2. Ulysses Corpus (2,68,117 words)

Please download the corpus files from this link.

7 Submission Format

Zip the following into one file and submit in the Moodle course portal. Filename should be <roll_number>_<assignment1>.zip, eg: 2022xxxxxx_assignment1.zip:

1. Source Code for Tokenization:

```
tokenizer.py
your text: Is that what you mean? I am unsure.
tokenized text: [['Is', 'this', 'what', 'you', 'mean', '?'],
['I', 'am', 'unsure', '.']]
```

On running this file, the expected output is a prompt which asks for an input and outputs the tokenized sentences.

2. Source Code for Language Models:

   ```
   language_model.py <lm_type> <corpus_path>
   ```

   LM type can be g for Good-Turing Smoothing Model and i for Interpolation Model.
   On running the file, the expected output is a prompt, which asks for a sentence and provides the probability of that sentence using the given mechanism. Therefore, an example would be:

   ```
   python3 language\_model.py i ./corpus.txt
   input sentence: I am a woman.
   score: 0.69092021
   ```

3. Source Code for Generation:

   ```
   generator.py <lm_type> <corpus_path> <k>
   ```

   LM type can be g for Good-Turing Smoothing Model and i for Interpolation Model.
   k denotes the number of candidates for the next word to be printed.
   On running the file, the expected output is a prompt, which asks for a sentence and outputs the most probable next word of the sentence along with it's probability score using the given mechanism. Therefore, an example would be:

   ```
   python3 generator.py i ./corpus.txt 3
   input sentence: An apple a day keeps the doctor
   output:
   away 0.4
   happy 0.2
   fresh 0.1
   ```

4. Report containing the perplexity scores of all LMs and your analysis of the results in a PDF:

   (a) For each LM, submit a text file with perplexity scores of each sentence in the following format:

avg_perplexity
sentence_1 <tab> perplexity
sentence_2 <tab> perplexity

(b) Naming must be:

<roll number>_LM1_train-perplexity.txt,
<roll number>_LM1_test-perplexity.txt

`Note:` Don't submit perplexity scores on dev/val splits.

5. README on how to execute the files, how to get perplexity of sentences along with any other information

6. In total, you will have:

   (a) 8 text files - one for each of the Language Model and Corpus Combination
   (b) 3 .py files
   (c) Report
   (d) README

8 Grading

Evaluation will be individual and will be based on your viva, report, submitted code review. In the slot you are expected to walk us through your code, explain your experiments, and report. You will be graded based on correctness of your code, accuracy of your results and quality of the code.

1. Tokenization (10)

2. N-grams (5)

3. Smoothing and Interpolation

   (a) Good Turing (20)
   (b) Interpolation (20)

4. Generation and Analysis (15)

5. Quality and Efficiency (10)

6. Viva during Evaluation (20)

9 Resources

1. An Introduction to N-Gram Language Modelling and Methods
2. Paper to help with lambda calculation for Interpolation
3. Slides for Language Modelling and Good Turing
4. Paper on Good Turing Smoothing
5. Wikipedia Page on Good Turing Smoothing
6. You can also refer to other resources, including lecture slides!