Natural Language Processing – Assignment I

Reykjavik University – School of Computer Science

Fall 2024

1 Python: Gutenberg corpus (16p)

Write a Python program, *corpusAnalysis.py*, that prints out various statistics/information about a given text file, which is a part of the Gutenberg corpus (accessible using NLTK). The name of the text file is given as a parameter to the program, i.e., you should call this from the command line:

```
python corpusAnalysis.py carroll-alice.txt
```

The program should print out the following information for the given text file:

```
Text:  carroll-alice.txt
Tokens:  34110
Types:  3016
Types excluding stop words:  2872
10 most common tokens:  [(',', 1993), ("'", 1731), ('the', 1527), ('and', 802),
('.', 764), ('to', 725), ('a', 615), ('I', 543), ('it', 527), ('she', 509)]
Long types:  ['affectionately', 'contemptuously', 'disappointment', 'Multiplication']
Nouns ending in 'ation'  ['usurpation', 'station', 'accusation', 'invitation',
'consultation', 'sensation', 'explanation', 'Multiplication', 'conversation', 'Uglification',
'exclamation']
```

Note: Long types are those with more than 13 characters.

Return your program code (.py file) along with the output of your program when running against the file *austen-emma.txt.*

2 NLTK: The Icelandic Gold Standard – 16 pts

In this part, you use Python and NLTK to process a Part-of-Speech tagged Icelandic corpus, the *Icelandic Gold Standard; MIM-GOLD*[1]. A preprocessed version of MIM-GOLD containing one sentence per line (with "/" between tokens and tags) is available as the file *MIM-GOLD.sent* in the zip file linked the assignment description.[2]

This part is divided into the subparts described below, but you should return a single Python program, **mim_gold.py**, which includes all the code needed for carrying out these tasks.

Write code to:

1. Read in MIM-GOLD using the class *TaggedCorpusReader*, display the number of sentences, and display the individual tokens of sentence no. 100.
2. Display the number of tokens and the number of types in MIM-GOLD.
3. Display the 10 most frequent tokens in MIM-GOLD using the class *FreqDist*.
4. Display the 20 most frequent tags in MIM-GOLD using the class *FreqDist*.

[1]See `https://clarin.is/en/resources/gold/`

[2]Please make sure not to distribute the MIM-GOLD corpus.

5. Generate tag bigrams and use the class *ConditionalFreqDist* to print out the 10 most frequent tags that can follow the tag ‘af’ (this tag denotes a preposition).

Example output follows:

```
Number of sentences: 58412
Sentence no. 100:
Púttað á Listatúni í dag , laugardag , kl. 10.30 .

Number of tokens: 1000218
Number of types: 106529

The 10 most frequent tokens
. => 49066
að => 35749
og => 33813
, => 29990
í => 27622
á => 21833
er => 16604
sem => 15199
til => 9888
um => 8799

The 20 most frequent PoS tags:
AF => 109899
AA => 74151
C => 68278
PL => 53212
SFG3EN => 36289
PK => 30647
SNG => 26345
TA => 22992
SFG3EÞ => 19773
CN => 19540
SÞGHEN => 17190
PA => 13043
N----S => 12783
CT => 12664
SFG3FN => 12294
NKEN => 11411
NVEN => 11171
NVEO => 11123
NHEÞ => 10979
NVEÞ => 10281

The 10 most frequent PoS tags following the tag 'af':
NVEÞ => 5714
NHEÞ => 5361
NKEÞ => 4222
CN => 3903
NVEO => 3556
NKEO => 3347
NHEO => 3046
```

```
NHEÞG => 2867
NVEÞG => 2723
FPHEÞ => 2703
```

3 Bias in Word Embeddings – 16 pts

Word embeddings are dense vector representations of words that have successfully improved the state-of-the-art on many NLP tasks. These vectors are produced by training classifiers on text corpora, such as Wikipedia, Twitter, and news sites. This data may contain biases of various kinds, e.g., denigration, stereotyping, recognition, and under-representation. Specifically, word embeddings may contain gender bias [1, 2], sexual orientation bias [3], ethnic bias [4], and ageism 5. These biases may be present in the word embeddings themselves and therefore have an adverse effect on the NLP task they're being used for, e.g., text classification, text summarization, language generation, and machine translation [6].

Investigate whether there are any potential biases to be found in word embeddings created using (1) data from Wikipedia and (2) data from Twitter (e.g. `glove-wiki-gigaword-100` and `glove-twitter-100`). There are several methods you can use to accomplish this task. For example, Lab 3 has methods for finding similarities and relationships between words that you can use and adapt, as you see fit (e.g., *find_word*, *most_similar*, *similarity*, *similar_by_vector*, *doesnt_match* – See more in the Gensim docs, like KeyedVectors). You may for instance explore combining vectors in various ways (e.g., using the NumPy package) or use any other method you discover.

Answer the following questions in essay-form:

1. What kinds of biases are present in either dataset, if any?
2. Are there any differences with respect to bias between the two kinds of data?
3. Can biases between concepts be revealed through visualizations?
4. Can bias in word embeddings be removed? If so, how?

Write about your findings and show example code you used to investigate this, including examples of relevant output. Additionally, there are several web-apps on the internet for visualizing word embeddings (e.g., WebVectors). Use one of these tools is to plot relations between concepts and include in your hand-in.

4 Your Own Personal GPT – 32 points

Objective. Create three small character-level GPT models using Karpathy's nanoGPT repository on a corpus of your choice. Evaluate the quality and creativity of the generated text under different parameter settings.

Background. Large-scale models like GPT-4 and Llama have made headlines; however, understanding the nuances of training and generating from smaller models can provide insights into the workings of generative language models. In this part, you will use nanoGPT to train a model and experiment with its generation capabilities.

Requirements. You must have Python 3 installed. Follow the instructions on `https://www.python.org/` to install the correct version of Python 3 for your machine. The second requirement is having Pytorch installed. Follow the instructions on `https://pytorch.org/get-started/locally/` to select the proper installation command for your system. You don't have to worry about CUDA unless you are going to be using a GPU.

Setup. Download the code for the nanoGPT repository by clicking the 'Code' dropdown on the Github page `https://github.com/karpathy/nanoGPT` and choosing an option that suits your needs. Make sure everything is working normally by following the instructions in the README file starting with the 'quick start' section. Run this command in the terminal:

```
python data/shakespeare_char/prepare.py
```

Address any issues that may arise, like installing missing packages. If you have a GPU, follow the GPU instructions. Otherwise go to the section that starts with 'I only have a macbook (or other cheap computer)'. Run the following command in the terminal:

```
python train.py config/train_shakespeare_char.py -device=cpu -compile=False -eval_iters=20
-log_interval=1 -block_size=64 -batch_size=12 -n_layer=4 -n_head=4 -n_embd=128 -max_iters=2000
-lr_decay_iters=2000 -dropout=0.0
```

Finally, generate some samples to see how the model performs by running:
`python sample.py -out_dir=out-shakespeare-char -device=cpu`.

Read these instructions carefully. With the basic setup step completed, carry out the following tasks:

1. Corpus Selection (8 points)
   - Choose a small corpus for training. This could be a collection of poems, short stories, news articles, or any other text your choice in any language.
   - Justify your choice: Write a brief paragraph explaining why you chose this corpus and what you hope the model might learn from it.
   - Unless you have beefy hardware, keep the length of the corpus under 1 million words (as counted by a standard word count program like wc, or those in Word or Notepad++).
   - Modify the nanoGPT code slightly to have it use your data. We will use the shakespeare_char code for training. Open the `data/shakespeare_char/prepare.py` file. Comment out the following lines:

     ```
     input_file_path = os.path.join(os.path.dirname(__file__), 'input.txt')
     if not os.path.exists(input_file_path):
         data_url = 'https://raw.githubusercontent.com/karpathy/char-rnn/
         master/data/tinyshakespeare/input.txt'
         with open(input_file_path, 'w') as f:
             f.write(requests.get(data_url).text)
     ```

   - Immediately below these lines, add this line with the path to your text file:
     `input_file_path = 'path/to/your/file.txt'`
   - Run the commands in the **Setup** section again. The data files for your corpus should have been created, the model trained on your text and used to generate text based on your data.
   - Note that the model will be called 'shakespeare_char' even though you've trained it on a completely different dataset. This is fine for the purposes of this assignment, it's just a name.

2. Model Training with Hyperparameter Exploration (16 points)
   - Train three different character-level GPT models using the method above, each with a distinct set of hyperparameters (e.g., learning rate, number of layers, number of attention heads, batch size, etc.).
   - For each model:
     - Document the chosen hyperparameters and the rationale behind your choices.
     - Detail the training process, including any challenges faced and how you overcame them.
     - Generate text from the trained model and provide the output.
     - Training a new model will overwrite the previous one (unless you modify the code). Ensure you save the generated outputs and any other necessary details before moving to the next model. To save the model itself, copy the 'out-shakespeare-char/ckpt.pt' file to another folder somewhere.

3. Analysis and Comparison (8 points)

- Reflect on the entire process:
  - What did you learn about the importance of hyperparameters in training GPT models?
  - How might the size or nature of your chosen corpus have influenced the results?
  - Based on your experiments, what further tweaks or tests would you consider if you were to continue this exploration?

5 What to return

A single .zip file containing the following:

1. Two Python programs: **mim_gold.py** and **corpusAnalysis.py** (include a file containing the output of your program when running against the file *austen-emma.txt*). Make sure you use functions (where appropriate) for specific tasks in your Python code, minimizing the duplication of code.

2. A PDF file that contains two parts: (*i*) the essay in Section 3 and (*ii*) responses that address all aspects and questions posed in the task description in Section 4, namely:

   - Your chosen corpus and justification.
   - Documentation of the training process, hyperparameters, and generated outputs for each of the three models.
   - Your analysis and comparison of the three models.
   - Your overall reflection.

As always, please let me know if you run into issues or you need help getting set up. Try to do this as early as possible so we can address any issues in a timely manner.