NLP 1a assignment

dinmukhammed lessov

March 2023

1 Plan of Work

1.1 Problem Description:

This project aims to create Python software to estimate a 2-gram language model for a speech recognition application. According to a disgruntled client, the application's existing language model is unsuitable for dictating books. As a result, a brand-new language model must be created from the start. The main procedures for the program's development are described in this paper. Available data: The provided dataset is a book excerpt, which will be used to train the language model. The dataset's word count, sentence structure, vocabulary, and word frequency represent a few of its main characteristics. It is not allowed to use other language models and sources of data cause of copyright laws.

1.2 Methods and Metrics Used in Evaluation:

- Data Preprocessing: The proper data preparation involves cleaning the text data by eliminating special characters, punctuation, and stop words such as "a," "an," "the," etc. because they have no functional significance. Tokenization, or breaking up text into individual words, is also required in order to build a list of tokens for further processing. Using FST to preprocess text data will be looked into.

- Language Model Architecture: Python will be used to create a brand-new application for estimating language models. A deep learning method, such as a recurrent neural network (RNN) or a long short-term memory (LSTM) network, will be applied in order to create the language model.

- Training and Testing the Language Model: The Adam optimizer and cross-entropy loss function will be used to train the language model on the provided dataset. In order to assess its accuracy and efficiency for book transcription, the trained language model will be put to the test on a different test set that was divided at the beginning.

1.3 Requirement Analysis:

Reviewing the client's needs and expectations for the voice recognition system should be the first step. The sort of language model and the necessary data requirements for the application will be determined. We will define the specifications for estimating a 2-gram language model M1 based on this.

1.4 Data structures:

In order to select appropriate data structures, the following should be considered: 1)size of text: because of the huge text corpus size: memory-efficient data structure should be chosen, 2) access time, 3)update time: because the corpus is so massive it can lead to the need for an update the model regularly, therefore the required data structures that can be effectively updated, 4) iteration time: to compute probabilities or produce text, the program needs to iterate through the data structures, therefore it is required to have

data structures that allow for quick iteration. Based on previous considerations, Hash Tables as the main data structure are chosen, because Hash tables are effective at updating and gaining access to 2-gram frequencies. For both accessing and modifying values, they have an O(1) average time complexity, or O(n) as max value, and also will be very useful when a new M2 language model with back off will come.

1.5 Stretch-goals:

1. FST for preprocessing of text data: to increase the program's effectiveness, preprocess text data using a Finite State Transducer (FST).

2. The data structures and code framework should be built in a modular and extendable manner to make sure that the design takes into account the later implementation of a language model M2 with back-off. This will make it simple to include the M2 model with the current M1 model. Therefore, modularity and extensibility should be given top priority in the design to make it easier to implement the M2 model with back-off in the future.

2 Progress Report

2.1 Short description:

In this report, the methods used for preprocessing of train and test datasets are fully explained, unfortunately, in this report, only the PTB dataset was considered, however in the final report both datasets would be included, as well as the creation of a Language Model that includes the training method based on the train dataset, functions for calculating probability among all other words in the sentence[Chi22] and functions for calculating the perfectibility of this model. Importing required libries: We'll use the math.log function to calculate the log-probability of each word in the test set and Regular expression operations [Fri09] In the last part, 2 new variables are created, then they are preprocessed after a new object(LM) is created and trained on preprocessed train dataset and complexity measured

2.2 Preprocess:

This piece of code preprocesses the input file (train or test datasets). First of all, it takes a filename as input, then divides it into separate sentences, in order to count the probability of each individual token inside the sentence. Also, to prepare the text data for the model building we perform text preprocessing [Dee21] that includes: Removing punctuations and lowercase to make it more efficient, changing digits and unknown words with a special tag

2.3 Language Model:

This is the definition of the LanguageModel class. The __init__ method initializes various variables that will be used throughout the class. q_number is the "q of q-gramm" of the model (i.e., the number of preceding words used as context), vocab is a set of all unique words in the training corpus, counts is a dictionary that stores the count of each (q_number-1)-gram/context followed by each word in the vocabulary, and total_count is the total number of words in the training corpus.

2.4 Training:

This is the train method, which trains the language model on a given corpus. The method takes in a list of lists of strings which was splitter in preprocessing part, where each sublist represents a sentence in the corpus. The method iterates over each sentence in the corpus and over each (q_number)-gram/context in the sentence. For every context and word in the (q_number)-gram, the method increments the count of

the word given the context in the dictionary. The method also updates total_count and adds the word to the vocab set, thereby expanding the vocabulary of LM.

2.5 Computing the probability and perplexity:

The get_probability method calculates the probability of a sentence using the n-gram language model. It does this by first breaking the sentence down into n-grams (subsequences of length q_number) and then looking up the probabilities of each n-gram in the model's probability distribution. The probability of the sentence is then computed as the product of these individual probabilities. The get_perplexity method, on the other hand, is used to evaluate the performance of the language model on a test set of sentences. It calculates the average perplexity of the model over all the sentences in the test set. Perplexity is a metric that measures how well a language model predicts a set of test sentences. It is calculated as the inverse of the geometric mean of the probabilities of the test sentences, normalized by the number of words in the sentences. In other words, a lower perplexity score indicates a better-performing language model[Cam20]. Together, the get_probability and get_perplexity methods allow us to use the n-gram language model to calculate the probability of individual sentences and to evaluate the model's overall performance on a test set. By comparing the model's perplexity score to those of other language models, we can assess the relative effectiveness of our model and make improvements as needed.

3 Final Report

3.1 Evaluation and discussion:

The new language model was successfully developed using Python, q-grams model, and hash tables "in theory" as the main data structure, and the text data was preprocessed to replace numbers and unknown words and punctuation marks with tags, however, the use of FST for preprocessing was not implemented. The model was trained and tested on Wikitext2 and PTB datasets, and the performance of the language model was based on the perplexity of the language model on the test dataset. The perplexity value for Wikitext2 was 6.2, and for PTB, it was 28. The lower perplexity value indicates better performance of the language model. The language model developed showed a significant improvement in the perplexity value after long adjustments of the implementation part 'code'. Comparing these results to existing state-of-the-art language models, the developed language model is inferior. However, it should be noted that the developed language model was built from scratch, without using any pre-existing solutions. Additionally, the specific needs of the customer were taken into account, which may not have been the case with existing language models. Therefore, while the performance of the developed language model is not state-of-the-art, it is a promising starting point for further development and customization.

3.2 Potential:

However, there were still some limitations in the model. One potential problem with the developed solution is its use of hash tables as the main data structure, and suggestions for improvement include exploring alternative data structures and incorporating additional features such as FST for preprocessing and an m2 model with backoff. The limitations of the model include the use of q-gram as the language model, which has limitations in handling long-distance dependencies[Wik]. Future work could explore alternative language models, such as neural language models "as it was planned in the plan report but not done due to complexity of model and hardness of implementation on the current level", to overcome this limitation. Furthermore, the M2 model with backoff was not implemented due to time and resource constraints. Additionally, the use of FST for preprocessing and the implementation of the M2 model with backoff could further improve the performance of the language model. Finally, the use of larger datasets for training the model could also improve the model's performance.

References

[Cam20] Chiara Campagnola. Perplexity in language models. *Towards Data Science*, 18.05.2020.

[Chi22] Fabio Chiusano. Two minutes NLP — Perplexity explained with simple probabilities. *NLPanet*, 2022.

[Dee21] Deepanshi. Text Preprocessing in NLP with Python codes. *Data Science Blogathont*, 2021.

[Fri09] Jeffrey Friedl. Mastering regular expressions. 3rd ed. *O'Reilly Media*, 2009.

[Wik] Wikipedia. n-gram, year =.