N-Gram Implementation of Character Language Model

This is an implementation of a probabilistic language model at the character level. A probabilistic language model maps sequences of symbols to discrete probabilities based on the context. The vocabulary is the set of all valid UTF-8 encodings of Unicode 10 in the basic multilingual plane, thus V = 65,424. This is an N-gram model where the probability of a character depends on the previous n-1 characters. It is based on Markov chain as seen below. In this case, w refers to a single Unicode character.

$$\mathrm{p}(w_1, \ldots, w_M) \approx \prod_{m}^{M} \mathrm{p}(w_m \mid w_{m-1}, \ldots, w_{m-n+1})$$

PROCESS:

I chose to implement the n-gram model to better understand the basics of implementation of language models. I have used Python 3.6 and basic Python libraries like math, numpy, pickle, sys, gzip. The entire model is coded from scratch without using prepackaged libraries. The main python file is **ngram_language_model.py** which takes as input a random seed and a mode. Random seed is used to sample from the probability distribution of candidate ngrams when generating new characters. The mode can have three values – TRAIN, TEST and RUN. TRAIN mode is used to train the model using files stored in the train_data folder. TEST mode can be used to calculate perplexity score on test files stored in test_data folder. The TRAIN mode trains the model on the training files and generates the absolute frequency and probability distribution of every k-gram encountered. The probability is calculated using below Maximum Liklihood Estimation formula:

$$\Pr(W_m = i \mid W_{m-1} = j, W_{m-2} = k) = \frac{\mathrm{count}(i, j, k)}{\sum_{i'} \mathrm{count}(i', j, k)} = \frac{\mathrm{count}(i, j, k)}{\mathrm{count}(j, k)}$$

T

Two lists of dictionaries are used to store the frequencies and probabilities of the ngrams. Each list has one dictionary for each of the k-grams where k = 1 to n. These dictionaries are serialized, zipped and stored in the bin folder to be used for querying the model. The frequencies are stored to calculate the probabilities of unseen sequences on the fly by referring to the (n-1)th gram which represents the context. The RUN mode is the mode where we can pass input from stdin as a sequence of characters starting with one of the 3 modes – o for observe, q for query and g for generate. The python script is wrapped in a Unix bash script which takes random seed as the only input parameter.

CHALLENGES:

Since the language model needs to be able to generate text and predict probability of characters from any natural language, the training data plays a very important role. Exposing the model to all 65424 Unicode characters which covers hundreds of natural languages is a challenging task. It is very likely that the model has not seen many sequences and will give a probability of zero for valid sequences from languages it has not been exposed to.

SMOOTHING:

To handle overfitting as seen above, I have introduced bias in the model using Laplace smoothing. We assume that each sequence was encountered at least once in the training set. The below formula is used to calculate the adjusted probabilities where alpha is set to 1. This method ensures that perplexity scores do not shoot up for unseen sequences.

$$\mathrm{p}_{\mathrm{smooth}}(w_m \mid w_{m-1}) = \frac{\mathrm{count}(w_{m-1}, w_m) + \alpha}{\sum_{w' \in \mathcal{V}} \mathrm{count}(w_{m-1}, w') + V\alpha}.$$

TRAINING:

The model is trained on a mixture of different languages like English, Portugese, Spanish, Dutch, Finnish, etc. The below are the links to the data sources:

1) 1 Billion Word Language Benchmark - http://www.statmt.org/lm-benchmark/ A few articles from this excellent English News Repository
2) NLTK Corpus - Univ Decl of Human Rights (300+ languages), CoNLL 2007 Dependency Treebanks (Dutch, Spanish), Genesis Corpus (6 languages), Floresta Treebank (Portuguese)

VALIDATION:

A validation set was created and set aside from each training file. It was used to determine the hyperparameter n of the model. Selecting the optimal number of grams is critical. After testing n from 1 to 10, I found that 4 gram gave the best performance in terms of space and time complexity without compromising much on accuracy. 10 grams seemed to perform worse and many longer sequences have very low probability of appearing in the training corpus. For better accuracy and performance, more efficient methods of smoothing like Katz Smoothing or Kneyser Ney smoothing can be used instead of Laplace smoothing.

TESTING:

Accuracy of the model was tested by calculating the perplexity on different test tests. Perplexities ranged from 1.27 for test data which included well-formed words and around ~4.2 for garbled English characters. Unseen sequences get a perplexity score of ~-15.99.

REFERENCES:

Jacob Eisenstein. Natural Language Processing. 2018.
URL https://github.com/jacobeisenstein/gt-nlp-class/blob/master/notes/eisenstein-nlp-notes.pdf.

Noah A. Smith. Probabilistic language models 1.0, 2017.
URL http://homes.cs.washington.edu/~nasmith/papers/plm.17.pdf.

I would like to mention some of my classmates – Hemant, Lohith, Shrinivas, Sayli with whom I had a couple of study and brainstorming sessions to verbally discuss the high level underlying concepts and implementation tradeoffs for this assignment.