n-gram Language Model Report

Ishan Adhaulia

IMS, University of Stuttgart

ishan.adhaulia@ims.uni-stuttgart.de, Matriculation 3319219

Abstract

This report is structured in five sections. The first and second section discusses the design philosophies and implementation techniques. The third section covers the documentation regarding running the project with details. The section fourth and fifth covers the extensions of perplexity and training/development respectively.

1. Design

List of all files/modules in the project are as follows

- corpus.py
  In this module, tokenize() and detokenize() functions are defined for creating text to list and list to text respectively.
  The function normalize() returns normalized version of the probability distribution while the function sample() returns the random selection of the key.
- lm.py
  This module contains LanguageModel class. The __init__() function constructs the object with n-gramsize. The list of important instance variables of class LanguageModel are as follows
  - vocabulary(**set** of tokens in the language. The PAD token used is **None** of type Nonetype)
  - ngrams(**list** of ngrams sequences)
  - counts(**dict of dict** containing word counts i.e [ngram_sequence: [word:word_count]] except for unigram model, then it is dict i.e [word:word_counts])
  - pdf (**dict of dict** containing probability distribution except for unigram, then it is dict of word and their probability. Similar structure to counts)

  The load() function loads corpus from a file and trains the language model for the particular file.
  The train() function trains language model with the particular list of tokens and learns the n-gram statistics.
  The p_next() function returns the estimated probability distribution for the next word that occurs after the given token sequence
  The generate() function generates a random token sequence according to the underlying probability distribution
  The perplexity() function calculates the perplexity of the given Language Model.
- main.py
  This module contains the runner script for the project. While loop is initialized at the start of the script and displays options for running the project.

2. Implementation

The decisions on the implementation of the coding style.

- corpus.py
  In this module, tokenize() function uses **string.punctuation regex** and import string module for using this regex.
  The detokenize() function creates an empty string and adding each word to the string.
  The function sample() uses **random.choice()** and we have to import the random module.
- lm.py
  This module contains LanguageModel class.
  The load() function opens the file, pass the list of sentences for training the model.
  The train() function uses slicing, counting and normalize() from corpus.py to construct the language model parameters like ngrams, counts or pdf instance variables of class LanguageModel.
  The generate() function generates a random token sequence based on generating the first token sequence using random module and then generating the rest of the tokens till EOS is encountered by using p_next().
  The perplexity() function calculates the sum of the negative logarithms of the probability distribution of the language model and taking the nthroot of the sum.
- all_unittests.py
  This module contains two test classes for corpus.py utilities's functions and lm.py's LanguageModel class.
  Integration test for the LanguageModel class for basic functionalities of the n-gram language model and nthroot function.

3. Documentation

Run this script for running the project.

```
python3 main.py
```

The script asks the user for integer input for selecting choice for the appropriate action. The list of actions with respect to choice is as follows

- Press 1

  The program asks for integer input for the n-gramsize of the language model. After entering the integer input, the program creates a new language model with a user-specified n-gram language model size.

- Press 2

  The program asks for string input for the name of the file for loading the corpus for the model. After entering the file name as string input, the program opens the file and create a list of sentences. Then this list of sentences is passed to an internal function train_file() for training the language model and computes the n-gram statistics.

- Press 3

  The program generates a random string of text based on lm.generate() and prints to the console.

- Press 4

  The program asks for integer input for generating the number of the texts to be saved in the file. After the integer input, the program asks for string input for the name of the file to be saved. The program asks for string input for the name of the file for loading the corpus for model. The program generates a random string of text based on lm.generate() and prints to the console. Generate a user-specified number of texts from the language model, and write them to a file

- Press 5

  The program asks for string input for text and which the user want to see for the next predicted word's probability. The program coverts the string in lowercase and then tokenize using the function defined in the corpus.py file. Then, the program prints the resultant dict to the screen.

- Press 6

  The program prints the perplexity of the language model

- Press 7

  The program exits successfully after printing the message "Exiting the main program".

4. Perplexity : Extension

4.1. Definition

Perplexity is a measurement of how well a probability distribution or probability model predicts a sample. It is used to compare probability models. A low perplexity indicates the probability distribution is good at predicting the sample.

4.2. Mathematics

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{p(w_i|w_{i-1})}} \tag{1}$$

To avoid underflow for floating point, take negative log_2 of ngram probability distribution.

4.3. Experiment

The program ran on the file "dev_shakespeare.txt". The perplexity of different language models are defined as follows

- Perplexity of the uni-gram language model is 123532
- Perplexity of the bi-gram language model is 526
- Perplexity of the tri-gram language model is 47
- Perplexity of the four-gram language model is 11

4.3.1. Conclusion

The higher ngram is, the better it is in predicting the sample because it gets a more and more richer context.

4.3.2. Reason for implementing Perplexity

I have implemented this feature because it helps us to understand the quality of the n-gram language model

5. Training/Development : Extension

5.1. Experiment

The program ran on the file ”train_shakespeare.txt” and ”dev_shakespeare.txt”. Multiple sub-experiments were run

- Train ”train_shakespeare.txt” and generate 2 sentences using uni-/bi-/tri-gram language model and also their perplexity
- Train ”dev_shakespeare.txt” and generate 2 sentences using uni-/bi-/tri-gram language model and also their perplexity
- Increasing size of the training corpus ie train on both sets(”dev_shakespeare.txt” and ”train_shakespeare.txt”) and report the generated text and their perplexity.

5.2. Results

5.2.1. Experiment 1(on train_shakespeare.txt)

Generating the random text and perplexity

A) Unigram

s1 : trunk seemeth gamut matter:–wear beatrice encounter hospital nursh-a veni petition’d accidental chafe painfully yours?–and restraining prowess wondering

s2 : porpentine giber fulness truckle-bed heinous wafts b all-hailed braver shielded spakest dotards anything

B) Bigram

s1 : resolutes for men unbind my boxes green goose thou winter-cricket thou on vacancy and text yet shall appear with him.–young prince as yours or thinking that shall speed is hanging presently holla within another’s heels marcus tenders for accusing the veriest shrew than but beshrew my anger him

s2 : devourers to abjure all feeling heaven by daylight

C) Trigram

s1 : thread charm ache with air and when it’s writ for my escape have put me from quarrelling

s2 : roman streets as stars with trains of fire from brutus

5.2.2. Experiment 2(on dev_shakespeare.txt)

Generating the random texts and perplexity

A) Unigram

s1 : greece scratch thick-ribbed unthought antiquary inventoried myrmidons tempest whorish blame vienna gagged dozen chopped chased takest twixt proserpine’s attend general fell shares clearly wrest wit clamours judges o’erlook’d right exaction setting wooers serviteur apparent discharging sphered recordation testril observer

s2 : usest calamity honest pointing house’s drum me:–what

B) Bigram

s1 : hug it be solemnized but offend’st thy senses affections had crotchets in mocking me father father

s2 : grace’s request i challenge read the almighty sun rose he told somewhat darker than event farewell the petty debt for neptune even she were best have courage execute

C) Trigram

s1 : was broke off partly for that

s2 : fever on goodness that is couch’d in seeming gladness is like that lead contains her twere damnation to think upon his brow

5.2.3. Experiment 3(on train_shakespeare.txt and dev_shakespeare.txt)

Generating the random texts and perplexity

A) Unigram

s1 : decrepit doublet earthy propertied gently manager mud wherever

s2 : tossing but pageants thought edifice traffickers love part know sail why when curtsy

B) Bigram

s1 : flea’s death dearth is or ne’er it call and bills here affright an auger-hole may again forsworn

s2 : mad-brain rudesby full appeach’d then if over-boldly we obey necessity am had begun tranio

C) Trigram

s1 : no music in three our kingdom if on earth but you see canary put me into mine ears present me as fearful as a dish fit for thee

s2 : concluded by priam and the detested wife will this fadge my master to some forlorn and all foul ways was ever respected with her being ever from their sport to have well-armed friends

5.2.4. Perplexity table

As from table 1, it is quite evident that on retrainig the model with more data, then the perplexity of the model increases i.e ability to predict better decreases.

	Unigram	Bigram	Trigram
Experiment 1	338657	1126	91
Experiment 2	123532	526	47
Experiment 3	376742	1127	98

Table 1: *Perplexity across experiments for unigram, bigram & trigram*

5.3. Conclusion

As the ngram size increase, generated text gets better ie. quality of sentences generated by trigram ¿ bigram ¿unigram language models

Perplexity of the model increases when we train the model more and more. This is evident across unigram/bigram/trigram models across different experiments.

5.4. Reason for implementing Training/Development

I have implemented this feature because it will help to understand the working of the n-gram language model