LING 520: Computational Analysis of English

Semester: FALL '16

Instructor: Sowmya Vajjala

Iowa State University, USA

27 September 2016

Class outline

- Review of language modeling from thursday's class
- More on language models

Katrin Erk's code

- How many of you successfully finished working with Katrin Erk's code samples? What did you understand?

Katrin Erk's code

- How many of you successfully finished working with Katrin Erk's code samples? What did you understand?
- How many read Nitin Madnani's article? How many MacOS and Linux users actually tried those terminal commands he showed?

Katrin Erk's code

- How many of you successfully finished working with Katrin Erk's code samples? What did you understand?
- How many read Nitin Madnani's article? How many MacOS and Linux users actually tried those terminal commands he showed?
- How much did you understand from Madnani's article? What did you not understand?

Katrin Erk's code

- How many of you successfully finished working with Katrin Erk's code samples? What did you understand?
- How many read Nitin Madnani's article? How many MacOS and Linux users actually tried those terminal commands he showed?
- How much did you understand from Madnani's article? What did you not understand?
- What seems to be an easier way to build a simple language model in Python?

Katrin Erk's code

- How many of you successfully finished working with Katrin Erk's code samples? What did you understand?
- How many read Nitin Madnani's article? How many MacOS and Linux users actually tried those terminal commands he showed?
- How much did you understand from Madnani's article? What did you not understand?
- What seems to be an easier way to build a simple language model in Python?
- Did you understand what is "smoothing" and what is "perplexity"?

Katrin Erk's code

- How many of you successfully finished working with Katrin Erk's code samples? What did you understand?
- How many read Nitin Madnani's article? How many MacOS and Linux users actually tried those terminal commands he showed?
- How much did you understand from Madnani's article? What did you not understand?
- What seems to be an easier way to build a simple language model in Python?
- Did you understand what is "smoothing" and what is "perplexity"?
- What is "training" and "testing"?

Katrin Erk's code

- How many of you successfully finished working with Katrin Erk's code samples? What did you understand?
- How many read Nitin Madnani's article? How many MacOS and Linux users actually tried those terminal commands he showed?
- How much did you understand from Madnani's article? What did you not understand?
- What seems to be an easier way to build a simple language model in Python?
- Did you understand what is "smoothing" and what is "perplexity"?
- What is "training" and "testing"?
- Finally, did anyone watch Week 6 lectures by Radev?

- If anyone is ready to come and talk about what they learnt from working with Katrin Erk's Python example, they are welcome to do that now.

Quick Recap of Language Modeling basics

Language Model

- goal: predict the next word based on some context. E.g., what is most likely to occur after "Ngram is a sequence of "
- how do we do?: store frequencies of occurrences of ngram patterns, and calculate the conditional probability of a word occurring given some context of previous words.
- chain rule of probability for a sentence:
    - If we assume a sentence S to be a sequence of words (w1, w2, ... wn), then, the probability of this sentence will be:
    - P(S) = p(w1|beginning of sentence)*p(w2|w1)*p(w3|w1,w2)*p(w4|w1,w2,w3)*... ... ... *p(wn|w1,w2,w3...wn-1)

The Markov assumption

- What is it?: It says instead of computing probabilities using entire history, we can use only the last few previous words.
- Let us say there is a sentence fragment : "ngram is a sequence of" and we want to estimate the probability of next word being "whatever".
- P(whatever|ngram is a sequence of) can be approximated as P(whatever|sequence of) if we choose a trigram model (2 words before current word).
- P(whatever) - unigram model. P(whatever|of) -bigram model .. and so on.

Maximum Likelihood Estimation

- This is the formula for getting bigram probabilities for one word: $\mathrm{P}(\mathrm{w}_n|\mathrm{w}_{n-1}) = \mathrm{C}(\mathrm{w}_{n-1},\mathrm{w}_n)/\Sigma_w(\mathrm{w}_{n-1}\mathrm{w})$

Maximum Likelihood Estimation

- This is the formula for getting bigram probabilities for one word: $\mathrm{P}(\mathrm{w}_n|\mathrm{w}_{n-1}) = \mathrm{C}(\mathrm{w}_{n-1},\mathrm{w}_n)/\Sigma_w(\mathrm{w}_{n-1}\mathrm{w})$
- However, $\Sigma_w(\mathrm{w}_{n-1}\mathrm{w})$ just means $\mathrm{C}(\mathrm{w}_{n-1})$.

Bigram Model Example from J&M Textbook

Jurafsky's Language Modeling lecture - slides 16–21.

Major problem with this approach so far

- As we discussed last time, most of the bi-gram probabilities are zero. This causes two issues:
  1. Sparse data representation. Solution: Efficient data structures to store data
  2. One ngram having a zero probability makes the whole sentence get zero probability.
- ...and what will we do when we see new words that did not exist in the corpus used to train the language model?
- How will we assign probabilities to an unknown event?

”Smoothing” in Language Modeling

Smoothing

- Used in language model construction to address issues that arise because of having zero probabilities for certain ngrams.
- intuition: use information about what is known to estimate the unknown.
- simplest way: start with a count of 1 instead of 0 for everything.
- better ways: the idea of "discounting" - use the information about ngrams that occur only once or twice in the corpus to estimate frequency of unseen ngrams.

Laplace Smoothing/Add one Smoothing

- Add 1 to all ngram counts in the model - this ensures we never get a non-zero probability.

MLE estimate: $$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Add-1 estimate: $$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

- We never assign a zero count to any ngram in this. We start with 1.

Handling Unknown, Out of vocabulary words

- What if we actually see a word that was never seen in training in the first place?

Handling Unknown, Out of vocabulary words

- What if we actually see a word that was never seen in training in the first place?
- Typically, such situations are handled in two ways:
  - Split the training set into two parts, and replace all words in part 2 that are not seen in part 1 as $< UNK >$. Use that as a proxy for any new word you see in test data.
  - Replace the first occurrence of any word in training set with $< UNK >$ and develop the model. Replace any new word in test data with $< UNK >$ and test.

Backoff and Interpolation

- Back-off: If a trigram in the test sentence is not seen in training data, back off to the bigram probabilities. If a bigram is also not seen, back off to unigram model.
- When we back-off, we take a discounted version of the lower order ngram probability.
- Interpolation: If let us say I have a trigram model. Then, I use uni, bi, trigrams to estimate probabilities, by assigning weights to these.
- E.g., in interpolated LM, P(test_sentence) = weight1*trigram_prob + weight2*bigram_prob + weight3*unigram_prob

(Note: Discussion about their implementation is beyond the scope of this course. Read Chapter 4 in J&M)

Advanced Smoothing Methods

- Good-Turing discounting
- Witten-Bell discounting
- Kneser-Ney smoothing
- Class based ngrams (based on grouping similar ngrams)

(Note: Discussion about their working is beyond the scope of this course. Read Chapter 4 in J&M)

Evaluating a Language Model

What is evaluation for a language model?

- How good is our model at assigning higher probabilities to well formed sentences?
- Is the model giving low probability to unlikely sentence constructions?
- We "train" the parameters of our language model (probabilities) using a large corpus, called "Training set"
- We test how the model is doing with a new set of sentences called "Test set"
- We use some evaluation metric to judge the model performance, after looking at how the model does with test set.

Training and Test Set

- Training set: the corpus of sentences we use to develop our language model.
- Test set: the corpus of sentences we use to evaluate of the model.
- Sometimes, there is also an initial test set called "development set" which you use to do preliminary evaluations, and finally test on the actual test-set.
- If there is a corpus of 100K sentences, generally, we use 80K for training, 10K for development, 10K for testing.
- Bad science: using the same data for training and testing.

Extrinsic and Intrinsic Evaluation

- Best way to evaluation one language model against another is to put it to use for some task (speech recognition, grammatical error correction etc)
- Get the accuracy for that task for model A and model B and decide which is the better model for that task.
- Drawback: Expensive and time consuming too.
- Solution: Do intrinsic evaluation using small test sets. This is good only for pilot experiments and initial research. Not for deciding about actual usage in a real application.

Intrinsic Evaluation of Language Models

Perplexity and Log Probability

- Perplexity is given by the formula: $P(w_1, w_2 \ldots w_n)^{-1/N}$ where N is the number of tokens in the sentence.
- Madnani's article represents Perplexity interms of log probability. Mathematically both these representations are the same (Derive yourself if interested!).
- Minimizing perplexity $==$ Maximizing the probability of a sentence.

Intrinsic Evaluation of Language Models

Perplexity and Log Probability

- Perplexity is given by the formula: $P(w_1,w_2...w_n)^{-1/N}$ where N is the number of tokens in the sentence.
- Madnani's article represents Perplexity interms of log probability. Mathematically both these representations are the same (Derive yourself if interested!).
- Minimizing perplexity $==$ Maximizing the probability of a sentence.
- Why is taking a logarithm of probability useful instead of normal probability?

Intrinsic Evaluation of Language Models

Perplexity and Log Probability

- Perplexity is given by the formula: $P(w_1, w_2 \ldots w_n)^{-1/N}$ where N is the number of tokens in the sentence.
- Madnani's article represents Perplexity interms of log probability. Mathematically both these representations are the same (Derive yourself if interested!).
- Minimizing perplexity $==$ Maximizing the probability of a sentence.
- Why is taking a logarithm of probability useful instead of normal probability?
- $\log(0.00000000000000001) = -17$ (Such extremely small or extremely large values are easy to interpret and compare on log scale).
- Also, $\log(a*b) = \log(a) + \log(b)$. Adding is faster than multiplying computationally.

Using perplexity in real experiments - 1

- Scenario 1: I trained three language models (1 unigram, 1 bigram and 1 trigram) using all texts written by Shakespeare.
- Now, I have a newly discovered Shakespearean text and I want to find out how close it is to these models.
- The perplexity scores of these 3 language models for this text are (450, 200, 120).
- Which is the better model to represent my new text?

Using perplexity in real experiments - 1

- Scenario 1: I trained three language models (1 unigram, 1 bigram and 1 trigram) using all texts written by Shakespeare.
- Now, I have a newly discovered Shakespearean text and I want to find out how close it is to these models.
- The perplexity scores of these 3 language models for this text are (450, 200, 120).
- Which is the better model to represent my new text?
- The model with lower perplexity is always a better model.
- If we expect our training data and test data to have the same style of word usage and so on, a trigram model is always better than a bigram model, which is always better than a unigram model.

Using perplexity in real experiments - 2

- Let us say I want to understand whether English writing of Chinese learners is closer to Korean learners or to German learners.
- I train one language model for Korean learner corpus, one language model for German learner corpus.
- Now, give some Chinese learner writing to both these models and get a perplexity score.
- If perplexity of Korean model is 100, and perplexity of German model is 200, we can conclude: Chinese learner's English writing is closer to Korean learner's English writing compared to German learners.

Using perplexity in real experiments - 3

Important thing to note: Perplexity score as a standalone number does not mean anything. It can only be used to compare between models.

Training and Storing Language Models

- Katrin Erk's example shows how to build simple and small language models in Python.
- Some common toolkits to train large, and storage efficient language models: SRILM, CMULM, KenLM, Kylm etc.
- They all expect corpus to be in: one line per sentence, tokens seperated by whitespace format.
- Trained model is stored in a format called ARPA, which stores 1–N gram log probabilities, along with Backoff weights and other such necessary stuff.
- The toolkits support various smoothing methods, and other optimization options.
- Madnani's article tells how to use these in Python.
- Other ways of training LM: Use POS tag ngrams, Word-POS tag combos, Phrase ngrams etc.

Some relevant resources

- Google Ngram viewer: https://books.google.com/ngrams (and a criticism: https://goo.gl/LwE7cn)
- Python libraries to use pre-existing language models:
  1. https://pypi.python.org/pypi/arpa/0.1.0b1
  2. https://github.com/awni/py-arpa-lm
- SRILM: http://www.speech.sri.com/projects/srilm/
- CMULM: http://www.speech.cs.cmu.edu/SLM/toolkit.html
- KenLM: https://kheafield.com/code/kenlm/
- KyLM: http://www.phontron.com/kylm/

Practice Exercise

Modify Katrin Erk's code sample to incorporate Laplace smoothing and return Log probabilities instead of normal probability.
Note: This is going to be useful in doing Question 1 of Assignment 3. So, try to do this.

Next Class

- Topics: POS Tagging Introduction, Assignment 2 discussion, Assignment 3 description
- ToDo: Submit Assignment 2