Language Modeling

1. Def: assign probabilities to sequences of words
2. N-gram Models
   1. Corpus: large and structured set of texts
   2. Compute the probability of a sequence of words
   3. Count (<Specific sentence>) / number of sentences of that length
   4. Chain Rule: Probability of a word sequence
   5. Ex. ‘I see what I eat and I eat what I see’
      1. P(I) = 4 / 12 = .33
      2. P(see | I) = ‘how many see given I’ = #(‘I see’)/ #(I) = 2 / 4 = .50
      3. P(what | I see) = 1 / 2 = .50
      4. By chain rule: P(‘I see what’) = .33 \* .5 \* .5
      5. Problem: the longer the history/ preceeding context, the less likely it will appear in the corpus. The farther we come to the right, the more likely the probability is 0 -> which is a trivial answer. Unable to give prediction for next word.
   6. Solution
      1. Approximate the prob of a word given all the previous words (forget something farther in the past ex. Beginning of the sentence)
      2. Use Markov assumptions: prob will depend limited number of the proceeding words
      3. Trigram model: truncate and look at the two preceding words
      4. Bigram model
         1. Assumes fake beginning token P(w1 | <s>), so that first word can also be a conditional probability

9/1/16

1. Quiz
   1. How many word tapes are in this text: Marseille is a dog. He might do a dog trick next.
      1. 10 (periods count!)
      2. **word types**: number of distinct words in the text
   2. How many word tokens are in this text: Marseille is a dog. He might do a dog trick next.
      1. 13
      2. **word tokens**: number of words
2. Training N-gram models
   1. Numerator:
   2. Denominator: count of preceding words
3. Bigram grammar fragment
   1. Can be read off a data structure that is precompiled
   2. Many sequences do not appear in bigram
   3. Bigram counts to probabilities
      1. Find P(i|i): # of times “i” appears in the corpus / # of time “i” follows “i”
      2. Normalize by unigram counts fro rlaters
4. Language Models
5. Assignment
   1. Add a spuodo
      1. Training the unknown word model
         1. Convert training set to use <unk> for any word
         2. Assume that all words that occur only once in the training set
            1. Ex. I am; I you x -> I am am; I <unk> <unk>
         3. Replace the first occurrence of every word type in the training data by <unk>
            1. Ex. I am am I you . Am I here? -> <unk> <unk> amd I <unk <unk> Am I here
6. Best way to compare models
   1. Intrinsic evaluation
   2. Compute Perplexity
7. Perplexity
   1. For a test set W = w1, …, wn
   2. PP(W) = P(w1…wn) -1/N
   3. Will take the log and add
   4. Must normalize by n
   5. The high the probability, the lower the perplexity; higher probability the model performs better -> lower perplexity = better model
   6. Always has to be done on separate test set
   7. Use bigram model to calculate probability
8. Evaluation using perplexity
   1. Test set: unseen dataset that is dif from training set; completely unused
   2. Evaluation metric: how well our model does on the test set (perplexity)
9. Training on the test set
   1. Perplexity scores only valid on exact same corpus
10. Zero probability bigrams
    1. Cannot compute perplexity (can’t divide by 0)
    2. Must remove zero’s
11. N-gram model “accuracy”
    1. First Assignment
    2. Accuracy increases as N increases
       1. Train multiple N-gram models and then use each to generate random sentences
       2. Example: Corpus: used complete works of Shakespeare (unigram: did worst… trigram: did best -> sequences of three will make sense)
    3. Process
       1. Choose a random bigram (<s>,w) [<s>: start of sentence; tokens in language; w: word] according to its probability (pick bigram pair subject to it’s probability; bigrams with higher probability have higher prob; lower, lower chance)
       2. Now choose next random bigram (w,x) again subject to probabilities
       3. Repeat until </s> (end of sentence marker) (ex. Until period generated = </s> )
       4. Ex. Trigram start = (<s> <s> ?) end = ( ? </s> </s>)
    4. Disadvantages
       1. Accuracy increases as N increases, however, ex. For quadrigrams there is no flexibility. Will just be generating direct sentences from Shakespeare

N-gram models

1. Assignment 1

Running sequence

I saw him

I saw him

I saw her

It them

N = 11

V = 6

Going through corpus, MLE (max likely expect)

Table 1:

I saw -> ~~1~~ 2

Saw him -> ~~1~~ 2

him I -> ~~1~~ 2

saw her -> 1

her it -> 1

it them -> 1

I I -> 0

I him -> 0

… -> 0

PMLE(him | saw) = 2/3 (times you see saw)

Add one smoothing =

Add one to all entries in Table 1:

PADD1(him | saw) = 3/9 (sum of table below)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | him | her | It | I | Then | saw |
| saw | 2 -> 3 | 1->2 | 0->1 | 0->1 | 0->1 | 0->1 |

* 1. Monday: develop sentence output
  2. 1-2 methods for getting rid of 0s

1. Overfitting Problems
   1. MLE: fit a parameter to a distribution
2. Add-one smoothing
   1. Add one to all of the counts before normalizing into probabilities to get rid of 0 counts
3. Adjusted counts c\*
   1. Describe how a smoothing alg affects the numerator by an adjusted count c\*
   2. Bigram Example
   3. D
4. Add-1 smoothing depends on size of vocabulary
   1. Sparse data (lots of 0), large vocabulary – so adding one skews the n-gram probabilities
   2. Too much of the seen bigram mass is being taken to the unseen bigram mass
5. Resolution: k smoothing
   1. Divide training into training and validation set

9/13 Part of Speech Tagging

1. HMM POS Tagger
   1. Given a set a words, find the tag sequence with the highest probability given those words
      1. Apply Bayes’
         1. (probability of tag seq \* prob tag given words) / prob of words
      2. Ignore denominator because it doesn’t depend on the tag sequence (always constant across all tag sequences)

9/29 HMM

1. Window-based Classification
   1. HMM doing optimization across the whole sentence
   2. Trying to optimize the tag sequence, which gives power to the HMM
   3. Classifiers have ability to make use of all features
2. Project 2
   1. Use HMMs or use sequence taggers
3. Chunking/ Partial parsing
   1. Want to identify non-overlapping seq of words, corresponds to spec categories of syntax (Ex. clauses, adv phrases, prep phrases)
   2. Fast and supports lots of NLP tasks
   3. Can use rule-base to classify (use finite-state-tranducer)
   4. Supervised Learning Algorithm
      1. Read off and remove duplicates