CS 124/LINGUIST 180
From Languages to Information

Language Modeling Bonanza!
Thomas Dimson
PA2 (due Friday)

• Implementation of a noisy channel model spelling corrector

• You implement:
  – Error model, language models, correction code

• Given a one-error sentence, returns the most probable corrected error [is this a good assumption?]
Agenda for Today

• The topic is language models
  – Anything that assigns a probability to sentences (or sequences of words)
• Our focus: n-gram models
  – Chain rule: \( P(<s> I am sam</s>) = P(<s>|sam,am,I,<s>) \times P(sam|am,I,<s>) \times P(am|I,<s>) \times P(I|<s>) \times P(<s>) \)
  – Approximate each term by truncating the chain to length n-1
  – Bigram (2-gram): \( P(I am sam) \approx P(<s>|sam)P(sam|am)P(am|I)P(I|<s>) \)
  – Estimate: \( P(sam|am) = \frac{\text{count}(sam,am)}{\text{count}(am)} \)
• We will explore a few “fun” things to do with language models over a limited corpus. Sit near someone who knows Python.
NLP Tasks in the ‘real’ world

• Given a giant blob of unstructured text, try to make some sense of it
• Lots of assumptions you’ve made about input are no longer valid
  – Data probably isn’t segmented into sentences and words
  – Vocabulary may be dramatically different than what your models are trained on (e.g. scientific domain)
  – Data is certainly not annotated
  – Words aren’t words, sentences aren’t sentences: “heyyyyy! How r u?”
“The artist is the creator of beautiful things. To reveal art and conceal the artist is art's aim. The critic is he who can translate into another manner or a new material his impression of beautiful things. The highest as the lowest form of criticism is a mode of autobiography. Those who find ugly meanings in beautiful things are corrupt without being charming. This is a fault.”

- Let’s try to count n-grams
  - What’s the problem?
- This paragraph isn’t tokenized into sentences! What can we do?
- Write a regular expression!
- Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems
  - Jamie Zawinski
- You might come up with something like “[.?!]”
  - Let’s try it
Perfect! But wait. What if...

“Others filled me with terror. There was an exquisite poison in the air. I had a passion for sensations... Well, one evening about seven o'clock, I determined to go out in search of some adventure.”

Patch it up: ‘[.?!]+’
  -- Let’s try it

Wait, someone decided to use unicode: “...” isn’t “...”

Patch it up: ‘[.?!...]+’
• Perfect! But wait. What if...
  – "You have a wonderfully beautiful face, Mr. Gray. Don't frown. You have. And beauty is a form of genius--is higher, indeed, than genius, as it needs no explanation."

• Can we patch it?
  – Maybe ‘(?<!Mr|Ms|Dr)[.?!...]’

• The point is, even a “simple” task like splitting sentences is tricky

• In real tasks you should utilize tools that others have built: NLTK, Stanford CoreNLP, etc. all have sentence tokenizers
Back to the Task

• We started with wanting a *language model*
  – Something that assigns a probability to a given sentence
  – Bulk of the work is counting n-grams over some corpus

• Given these counts, we can figure out “reasonable looking” text
First Task: Text Generation

• Can we generate text that suits the style of an author?
• Given previous words, choose a likely next word according to your language model
  – Roll a biased $|V|$-sided dice and choose that word as the next one
  – Stop if the word is $</s>$
  – Could also choose the most likely next word (a pseudo auto-complete)
• I’ve pre-computed N-gram counts for a bunch of Public Domain books, let’s see what we can do.
Text Generation

• One click: https://www.stanford.edu/class/cs124/lec/language_bonanza.tar.gz
• Code is available here: /afs/ir/class/cs124/sections/section2
  – Copy this to a local directory
• Data files are available here: /afs/ir/class/cs124/sections/section2/data
  – Data files are serialized “BookNgrams” objects representing counts of ngrams in a particular book.
  – alice_in_wonderland.lm, beatles.lm, edgar_allen_poe.lm, michael_jackson.lm, shakespeare.lm, ulyses.lm, art_of_war.lm, devils_dictionary.lm, king_james_bible.lm, odyssey.lm, tale_of_two_cities.lm
  – If you want another book that is more than 100 years old, ask me and I can prepare it quickly
• Run by “python2.7 bonanza.py generate <provided-data-file.lm> <n-gram-order>”
  – E.g. “python2.7 bonanza.py generate /afs/ir/class/cs124/sections/section2/data/beatles.lm 3”
  – Nothing for you to write, just play around with the code. Take a peek inside if you get bored.
• Some questions to answer:
  – What’s the most humorous / bizarre / interesting sentence you can generate?
  – How does the quality of text change as you vary ‘n’ in your language model (e.g. bigram model, trigram model)?
  – What works best? Poetry, prose or song lyrics?
  – Why is Michael Jackson occasionally so verbose? Conversely, why does he sometimes start with and end with “.”
The Beatles Are Back!

• I'll bet you I'm so tired
  Good night sleep tight
  Dream sweet dreams for me
  Dream sweet dreams for me and my monkey

• Maybe not...

• Try at home: dump your e-mail/text history
  and create an n-gram word model for yourself
Small Notes

• For small corpora, most words get 0 probability, so with high values of ‘n’, there is only one choice for the next word (the one we’ve seen before)

• We could ‘fix’ this by having a small chance to choose some other word
  – Any smoothing method would do this, with varying degrees of “stolen” probability mass
Second Task: Tip of Your Tongue

• Given a sentence with a missing word, fill in the word:
  – “The ____ minister of Canada lives on Sussex Drive”
  – Auto-complete with arbitrary position

• How can you do this using your n-gram models?
  – Try all words, and see what gives you the highest probability for the sentence
Tip of Your Tongue

• This time you have to write code for calculating sentence probabilities
  – Start with unsmoothed, you can add smoothing for the bonus
• Look for “##### PART 2: YOUR CODE HERE ######” in the starter code from before
  – /afs/ir/class/cs124/sec/sections/section2/bonanza.py or on the class site
  – Reminder: data files are available here: /afs/ir/class/cs124/sec/sections/section2/data
• Run by ‘python2.7 bonanza.py tongue <provided-data-file.lm> <n-gram order> <sentence with ____ for missing character>’
  – E.g. “python2.7 bonanza.py tongue /afs/ir/class/cs124/sec/sections/section2/data/beatles.lm 3 ____ to ride”
  – Don’t include the end of sentence punctuation
  – Vary n-gram order for amusing results. [why?]
• Complete the following sentences:
  – “Let my ____ die for me” in ullyses.lm
  – “You’ve been ____ by a _____ criminal” in michael_jackson.lm
  – “Remember how ____ we are in happiness , and how ____ he is in misery” in tale_of_two_cities.lm
  – **Bonus**: Add Laplace Smoothing to your model and complete:
    • “I fired his ____ towards the sky”
Small Notes

• These examples were contrived. When you venture “off script” (n-grams previously unseen) you run into zero probabilities
  – This is why we need smoothing
• Interesting generalization: “The _____ minister of _____ is _____”
  – $|V|$ possibilities for each word => sentence has $|V|$ cubed possibilities. Exhaustive search will kill you
  – Could do a greedy scan. Will this maximize probability?
Third Task: Scramble!

• New noisy channel:
  – Person writes down a sentence, cuts out each word and throws the pieces in the air

• Given the pieces, can you reassemble the original sentence?
  • Error model is a constant probability

• “In world the best Thomas material is teaching”
  • Thomas is teaching the best material in the world
Scramble!

• This time you have to figure out code for choosing the best unscrambling
  – Use the code you had previously written for calculating sentence probabilities
  – itertools.permutations is your friend
• Look for “####### PART 3: YOUR CODE HERE #######” in the starter code
  – Available here: /afs/ir/class/cs124/sections/section2
  – Reminder: data files are available here: /afs/ir/class/cs124/sections/section2/data
• Run by “python2.7 bonanza.py scramble <provided-data-file.lm> <n-gram-order> <my sentence>”
  – E.g. “python2.7 bonanza.py tongue /afs/ir/class/cs124/sections/section2/data/beatles.lm 3 ride to ticket”
• Descramble the following sentences:
  – “the paul walrus was” in beatles.lm
  – “of worst was times the it” in tale_of_two_cities.lm
  – “a crossing away river far after should you get” in art_of_war.lm
    • This may melt your computer [why?]
  – **Bonus:** If you implemented smoothing before, you can see how different authors would rearrange any words of your choice. You should stick to small values of ‘n’ to make this work.
Small Notes

• The algorithm you just wrote is $O(n!)$ in sentence size.
  – There are certain constraints you could impose to prune the search space (adjectives next to nouns, etc.)
  – Also could randomly sample from the search space
  – I’m not actually sure of the best algorithm for this
Questions and Comments

• Any questions about anything?
• PA2 is due Friday. Start early.
  – I’ll be at office hours all week if anyone needs help.
  – Group code jam is tomorrow night.
• Code you wrote today should be pretty helpful