Start the recording

- Today
  - Project 2 Solution
  - Text Processing
  - Formal Grammars
  - Project 4

Reminder

- Project 3 due Tuesday 8/21 at 11:45pm
- Assignment 3 due Thursday 8/23 at 4:30pm
- Project 4 due Thursday 8/30 at 11:45pm

Next week

- No office hours next week (sorry- please use the discussion board)
- Guest Lecturers:
  - Sara Ng
  - Suchin Gururangan
- Slides, lectures and solutions will be posted a little late
- I will go over the homework solutions when I return, but the solution will be available on Canvas after the deadline

Project 2

- Word frequencies tend to fall off in accord with “Zipf’s Law”
  - Most common word: 1
  - Second most common word: ½ as common
  - Third most common word: 1/3 as common
  - etc

Project 2

```
import os
import re

tags = re.compile("<.*?>")
invalid_chars = re.compile("[^a-z']")
invalid_apostrophe_start = re.compile(" '+|^'+")
invalid_apostrophe_end = re.compile("'+ |'+$")
dictionary = dict()

for fi in os.listdir('/corpora/LDC/LDC02T31/nyt/2000'):
  f = open('/corpora/LDC/LDC02T31/nyt/2000/' + fi, 'r').read().lower()
  f = tags.sub(' ', f)
  f = invalid_chars.sub(' ', f)
  f = invalid_apostrophe_start.sub(' ', f)
  f = invalid_apostrophe_end.sub(' ', f)
  words = f.split()
  for word in words:
    if word not in dictionary:
      dictionary[word] = 1
    else:
      dictionary[word] += 1

for word in sorted(dictionary.items(), key=lambda word: word[1], reverse=True):
  print (word[0] + "\t" + str(word[1]))
```

POS tagging objective function

$$\hat{t} = \text{argmax}_t \prod_i \left(\frac{\text{count}(w_i, t_i)}{\text{count}(t_i)} \times \frac{\text{count}(t_{i-1}, t_i)}{\text{count}(t_{i-1})} \right)$$

Best POS tag sequence

How often does word w_i occur with tag t_i in the corpus?

How often does t_i follow t_{i-1} in the corpus?

This might seem a little backwards (especially if you aren't familiar with Bayes' theorem). We're trying to find the best *tag sequence*, but we're using $P(w|t)$, which seems to be predicting *words*.

This compares: "If we are expecting an **adjective** (based on the tag sequence), how likely is it that the adjective will be 'cold?'" ***versus*** "If we are expecting a **noun**, how likely is it that the noun will be 'cold?'"

Multiplying probabilities

- We're multiplying a whole lot of probabilities together
- What do we know about probability values?

$$0 \leq p \leq 1$$

- What happens when you multiply a lot of these together?
- This is an important consideration in computational linguistics. We need to worry about underflow.

Underflow

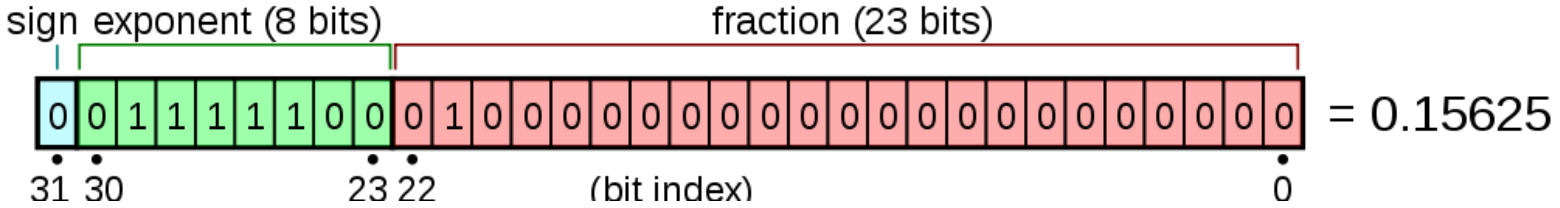
- When multiplying many probability terms together, we need to prevent underflow
  - Due to limitations in the computer's internal representation of floating point numbers, the product quickly becomes zero
- We usually work with the logarithm of the probability values
- This is known as the "log-prob"

$$= \log_{10} p$$

IEEE 754 floating point

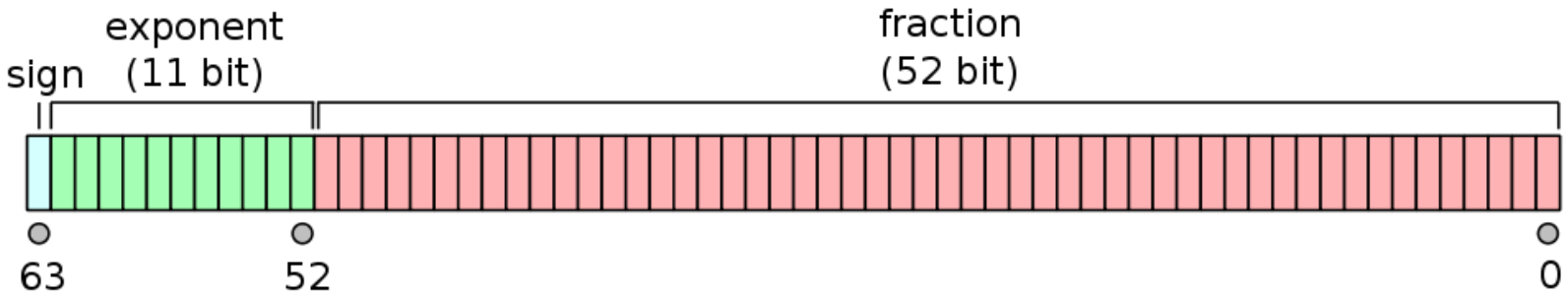
- 32-bit “single” “float”

$$1.2 \times 10^{-38} \; to \; 3.4 \times 10^{38}$$



- 64-bit “double”

$$\approx \pm 1.8 \times 10^{308}$$



logarithms refresher

definition: $\log_b x = y : x = b^y$

$$b^x \times b^y = b^{x+y}$$

$$\log xy = \log x + \log y$$

$$\log \prod_i x_i = \sum_i \log x_i$$

$$\frac{b^x}{b^y} = b^{x-y}$$

$$\log\frac{x}{y} = \log x - \log y$$

Write an expression for Bayes' theorem as log-probabilites

Bayes' theorem as log-prob

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$$\log P(A|B) = \log P(B|A) + \log P(A) - \log P(B)$$

Remember this?

$$\hat{t} = \text{argmax}_t \prod_i \left(\frac{\text{count}(w_i, t_i)}{\text{count}(t_i)} \times \frac{\text{count}(t_{i-1}, t_i)}{\text{count}(t_{i-1})} \right)$$

$$\hat{t} = \text{argmax}_t \sum_i \left(\log \frac{\text{count}(w_i, t_i)}{\text{count}(t_i)} + \log \frac{\text{count}(t_{i-1}, t_i)}{\text{count}(t_{i-1})} \right)$$

Wait, how can we do this, there was no 'log' in the original equation outside the $\prod$!

argmax magic

- Doesn't matter. Since argmax doesn't care about the actual answer, but rather just the *sequence that gives it*, we can drop the overall log
  - this is valid so long as $\log x$ is a monotonically increasing function
- argmax will find the same "best" tag sequence when looking at either probabilities or log-probs because both functions will peak at the same point

$$\hat{t} = \text{argmax}_t \sum_i \log P(w_i|t_i) + \log P(t_i|t_{i-1})$$

Hidden Markov Model

- This is the foundation for the Hidden Markov Model (HMM) for POS tagging
- To proceed further and solve the argmax is still a challenge

$$\hat{t} = \text{argmax}_t \sum_i \log P(w_i|t_i) + \log P(t_i|t_{i-1})$$

- Computing this naïvely is still $O(|T|^n)$

Dynamic programming

- The Viterbi algorithm is typically used to decode Hidden Markov Models
  - You probably will implement it in Ling 570
- It is a dynamic programming technique
  - We maintain a trellis of partial computations
- This approach reduces the problem to $O(|T|^2 n)$ time

POS Trigram model

Recall the bigram assumption:

$$P'(t_i) \approx \prod_i P(t_i|t_{i-1})$$

We can improve the tagging accuracy by extending to a trigram (or larger) model

$$P'(t_i) \approx \prod_i P(t_i|t_{i-2}, t_{i-1})$$

Question: Why not increase 8, 10, 12-grams?

Data sparsity

- However, we might start having a problem if we try to get a value for $P(t_i|t_{i-2}, t_{i-1})$ by counting in the corpus

$$\frac{\text{count}(t_{i-2}, t_{i-1}, t_i)}{\text{count}(t_{i-2}, t_{i-1})}$$

…it was a butterfly in distress that she…

The count of this in our training set is likely to be zero

Unseens

- Our model will predict zero probability for something that we actually encounter
  - This counts as a failure of the model (why?)
- This is a pervasive problem in corpus linguistics
  - At runtime, how do you deal with observations that you never encountered during training (unseen data)?

Smoothing

- We don't want our model to have a discontinuity between something infrequent and something unseen
- Various techniques address this problem:
  - add-one smoothing
  - Good-Turing method
  - Assume unseens have probability of the rarest observation
  - Ideally, smoothing preserves the validity of your probability space

Formal Grammars

- What is a 'grammar'?
- For a *formal grammar* (a mathematical definition):
  - A system that generates and recognizes strings present in some *formal language*
  - A *formal language* is a (possibly infinite) set of strings
- Examples of formal languages:
  - Cricket language: {chirp, chirp-chirp}
  - Sheep language: {ba, baa, baaa, baaaa, …}

Formal Languages

- Some terms:
- Alphabet (Σ)
  - A finite set of symbols
  - "symbol" could be written, auditory, visual, etc.
- String
  - A (possibly empty) ordered sequence of symbols from Σ
- Formal language (L)
  - A set of strings defined over the alphabet Σ.
  - Any language will always be a subset of Σ^*.

$$L \subset \Sigma^*$$

Formal Languages

- One of the simplest languages is Σ^*, all possible strings pulled from the alphabet.
- It's often useful to constrain the language a bit more.
- We can use generative rules to constrain the language.
  - These have the nice effect of generating constituents

Constituency

- Groups of words behave as a single unit or phrase.
- Sentence have parts, which have subparts, etc.
- e.g., Noun Phrase
  - kermit the frog
  - he
  - August 22
  - the failure of the advisory committee to solve the problem

Constituent Phrases

For constituents, we usually name them as phrases based on the word that heads the constituent

the man from Amherst	is a Noun Phrase (NP) because the head man is a noun
extremely clever	is an Adjective Phrase (AP) because the head clever is an adjective
down the river	is a Prepositional Phrase (PP) because the head down is a preposition
killed the rabbit	is a Verb Phrase (VP) because the head killed is a verb

Note that a word is a constituent (a little one). Sometimes words also act as phrases. In:

Joe grew potatoes.
Joe and *potatoes* are both nouns and noun phrases.

Compare with:

The man from Amherst grew beautiful russet potatoes.

We say *Joe* counts as a noun phrase because it appears in a place that a larger noun phrase could have been.

Evidence for constituency

- They appear in similar environments (before a verb)
  - *Kermit the frog comes on stage*
  - *He comes to Massachusetts every summer*
  - *August 22 is when your project 3 is due*
  - *The failure of the advisory committee to solve the problem has resulted in lost revenue.*
- The constituent can be placed in a number of different locations
  - On August 22 I'd like to fly to Seattle.
  - I'd like to fly on August 22 to Seattle.
  - I'd like to fly to Seattle on August 22.
- But not split apart:
  - *On August I'd like to fly 22 to Seattle.
  - *On I'd like to fly August 22 to Seattle.

Substitution test

- Adjective:

  The {sad, intelligent, green, fat, …} one is in the corner.

- Noun:

  The {cat, mouse, dog} ate the bug.

- Verb:

  Kim {loves, eats, makes, buys, moves} potato chips.

Word categories

- Traditional parts of speech

Noun	Names of things	boy, cat, truth
Verb	Action or state	become, hit
Pronoun	Used for noun	I, you, we
Adverb	Modifies V, Adj, Adv	sadly, very
Adjective	Modifies noun	happy, clever
Conjunction	Joins things	and, but, while
Preposition	Relation of N	to, from, into
Interjection	An outcry	ouch, oh, alas, psst

- Most of these form phrases (noun phrase, adverb phrase, etc)

Formal Grammars

- Here's how we'll constrain a language from all possible strings Σ^* : constituents can only be *juxtaposed* as permitted by R
- Each rule is a rule that generates a constituent, including the top constituent of sentence

  Σ – a set of symbols (words, letters, …)
  R – a set of rules
- Our formal grammar is $G = \langle \Sigma, R, \ldots \rangle$

Formal Grammars

- Formal grammars can be viewed in two useful ways.
- Given a configuration of rules in grammar G, find the string(s) S that can be formed.
  - This is called generation.
- Given a string S in a language L, find a rule configuration that generates S.
  - This is called parsing.

Formal Grammars

- A *grammar* is fully defined by the tuple

$$G = \langle V, \Sigma, S, R \rangle$$

  - V is a finite set of *variables* or *preterminals* (our constituents, incl POS)
  - Σ is a finite set of symbols, called *terminals* (words)
  - S is in V and is called the *start symbol*
  - R is a finite set of *productions*, which are *rules* of the form

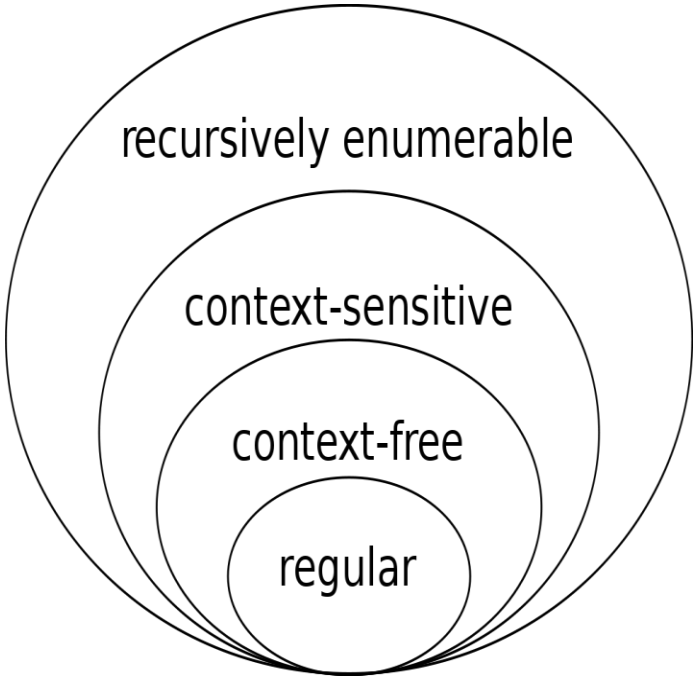
$$\alpha \rightarrow \beta$$

    where α and β are strings consisting of terminals and variables.

Types of formal grammars

- We can rank grammars by expressiveness
- Unrestricted, recursively enumerable (type 0)
- Context-sensitive (type 1)
- Context-free grammars (type 2)
- Regular grammars (type 3)

The Chomsky hierarchy



Types of formal grammars

- Unrestricted, recursively enumerable (type 0)

  $\alpha \rightarrow \beta$

  α and β are any string of terminals and non-terminals
- Context-sensitive (type 1)

  $\alpha X \beta \rightarrow \alpha \gamma \beta$

  X is a nonterminal; α, β, γ are any string of terminals and non-terminals; γ may not be empty.
- Context-free grammars (type 2)

  $X \rightarrow \gamma$

  X is a nonterminal; γ are any string of terminals and non-terminals
- Regular grammars (type 3)

  $X \rightarrow \alpha Y$

  X, Y are nonterminals; α is any string of terminals; Y may be absent.

(i.e. a *right* regular grammar)

Equivalence with automata

- Formal grammar classes are defined according to the type of automaton that can accept the language
- There are various types of automatons, and many correspond to certain types of formal grammars

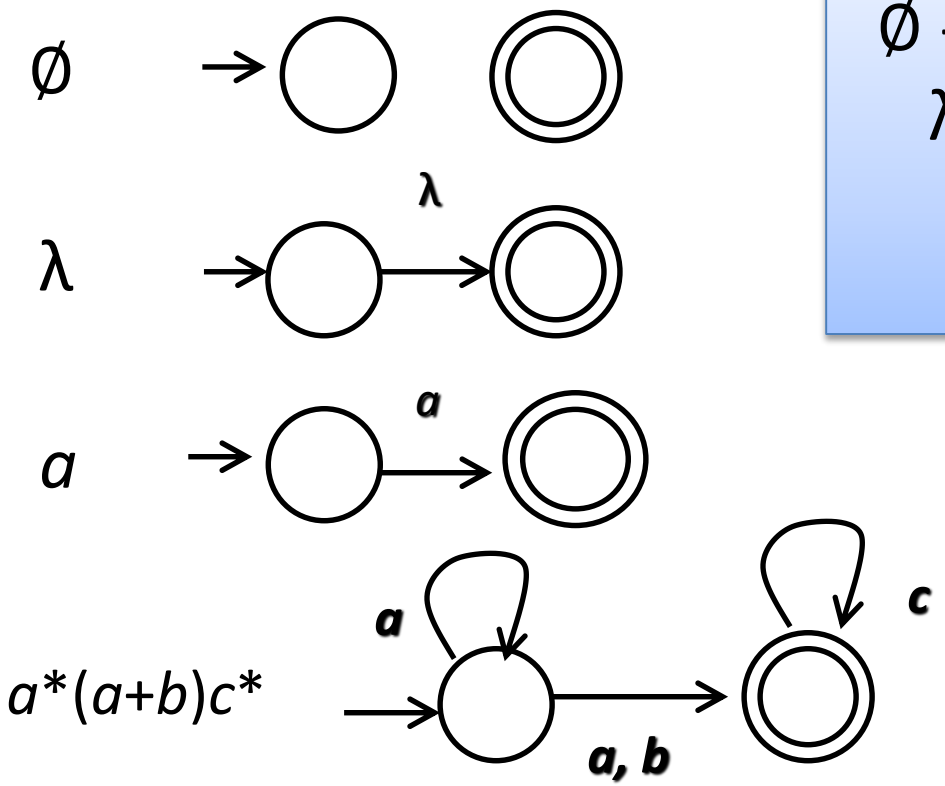
Grammar type	Accepted by
Recursively Enumerable	Turing machine
Context sensitive	non-deterministic linear bounded automaton (LBA)
Context Free	non-deterministic pushdown automaton (NDPA)
Regular	Finite State Automaton

Regular languages

- Let's look at the most restricted case
- A regular language (over an *alphabet* Σ) is any language for which there exists a finite state machine (finite automaton) that recognizes (accepts) it
- This class is equivalent to regular expressions (but no capture groups)

Regular Expressions and Regular Languages

- There are simple finite automata corresponding to the simple regular expressions:



∅ - the empty set
λ – the empty string
$a \in \Sigma$

Each of these has an initial state and one accepting state.

Regular grammars

(type 3)

- Regular grammars can be generated by FSMs
- Equivalence with RegEx
- Definition of a *right-regular grammar:*
  - Every rule in *R* is of the form
    - $A \rightarrow aB$ or
    - $A \rightarrow a$ or
    - $S \rightarrow \lambda$ (to allow λ to be in the language)
    - where *A* and *B* are variables (perhaps the same, but *B* can't be *S*) in *V*
    - and *a* is any terminal symbol

Regular grammars

(type 3)

- Example:

$$V = \{S, A\}$$
$$\Sigma = \{a, b, c\}$$
$$R = \{S \rightarrow aS, S \rightarrow bA, A \rightarrow \lambda, A \rightarrow cA\}$$
$$S = S$$

RegEx: a*bc*

- Cannot express $a^n b^n$

Parsing regular grammars

(type 3)

- Parsing space: $O(1)$
- Parsing time: $O(n)$

Project 4

- Due Thursday August 30 at 11:45pm
- Optimization and data structure selection really matters!
  - Using a regex will take hundreds of hours
- condor.cmd and run.sh required
  - You must use condor to run your jobs!

Project 4

- Write a trie class. It can be specialized for a four-symbol alphabet or more general – it will just run on the input data.
- Initialize the trie with the 4,965 target DNA sequences.
- Passing through the human genome (2.8GB) once, use the trie to locate all occurrences of the target sequences.
- Print out the matches: file name, and offset.
- You must be careful to correctly build the trie and parse with it.

Project 4

- Increasing speed
  - Treat the input as a byte array, not characters or Strings
  - Why? Bytes are ½ the size of unicode code points!
  - Make sure to pass over things once, and don't generate lots of extra strings
- USE CONDOR WHEN RUNNING ON THE CLUSTER
  - Work with a smaller subset of the data until you have your implementation ironed out
  - Run on the full data set when you want to check changes in your optimization

Next Time

- Continue exploring Chomsky Hierarchy and Regular Languages