Lecture 9

August 18, 2016

Language Modeling and POS Tagging

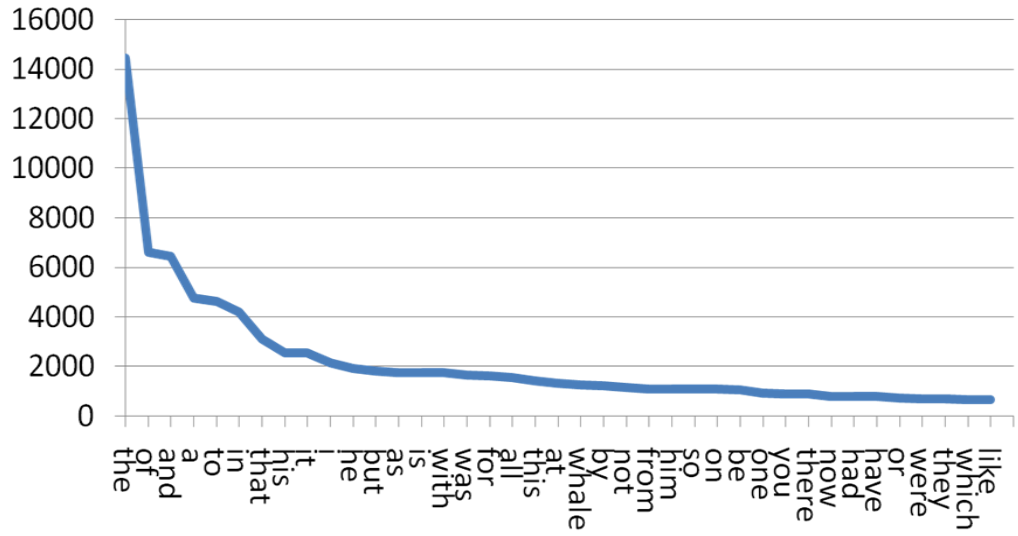
Reminder: start the recording

Announcements

- Assignment 3
  - due now
- Project 3: Thai FST
  - due Tuesday 11:45 pm
- Project 4: due Thursday, 9/1
- Writing Assignment: due Tuesday, 9/6
- Project 5 (final one): now posted, due Thurs. 9/8
  - Bayesian language classifier
  - Theory partially covered today
- Questions?

Project 2: Zipf's Law

- The frequency of a word in a natural language corpus is inversely proportional to its tally rank
- This follows a geometric distribution



Project 2

```
Dictionary<String, int> tallies = new Dictionary<string, int>();

IEnumerable<String> words =
    Directory.GetFiles(args[0])
             .SelectMany(f => new Regex(@"\<.*?\>").Replace(File.ReadAllText(f), " ")
                                .ToLower()
                                .Select(ch => ('a' <= ch && ch <= 'z') || ch == '\'' ? ch : ' ')
                                .NewString()
                                .Split(new Char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)
                           )
              .Select(w => w.Trim('\''));

foreach (String wrd in words)
    if (tallies.ContainsKey(wrd))
        tallies[wrd]++;
    else
        tallies.Add(wrd, 1);

foreach (var tal in tallies.OrderByDescending(t => t.Value))
    Console.WriteLine("{0}\t{1}", tal.Key, tal.Value);
```

Declarative programming with C#/LINQ
Compose elaborate vector manipulations without procedural constructs like loops

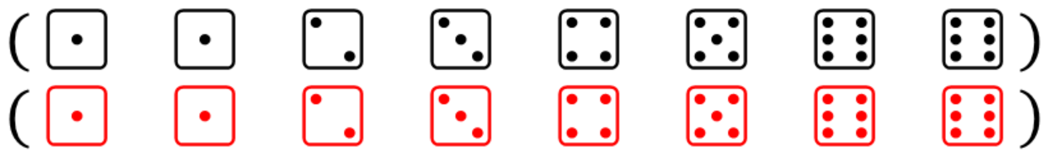
```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text.RegularExpressions;

static class Program
{
    static void Main(string[] args)
    {
        foreach (IGrouping<String, int> grp in Directory.GetFiles(args[0])
                                .SelectMany(f => new Regex(@"\<.*?\>").Replace(File.ReadAllText(f), " ")
                                                    .ToLower()
                                                    .Select(ch => ('a' <= ch && ch <= 'z') || ch == '\'' ? ch : ' ')
                                                    .NewString()
                                                    .Split(new Char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries))
                                .GroupBy(w => w.Trim('\''))
                                .OrderByDescending(g => g.Count()))
            Console.WriteLine("{0}\t{1}", grp.Key, grp.Count());
    }

    static String NewString(this IEnumerable<Char> ie) { return new String(ie.ToArray()); }
}
```

Assignment 3

Consider weighted dice—one white, and one red. For each die,  and  are twice as likely to show as the other four values. What is the probability that the total showing on the two dice will be 7?



The cartesian product has 64 cases.

Ways to get 7:	(1,6)	(2,5)	(3,4)	(4,3)	(5,2)	(6,1)
Number of tuples:	4	1	1	1	1	4

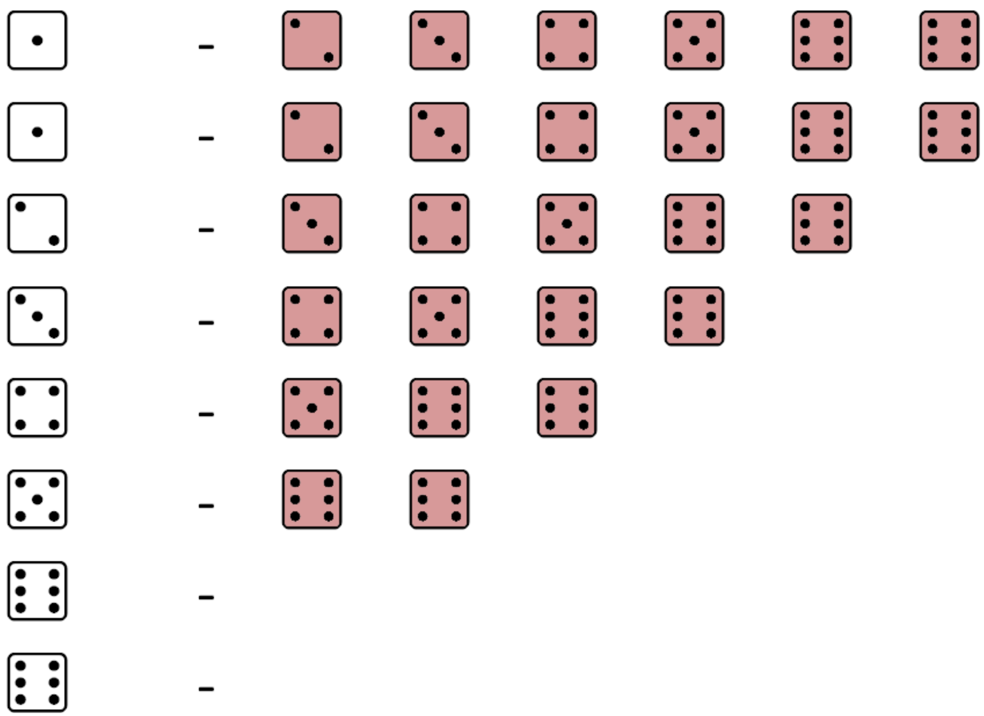
$$\frac{12}{64} = \frac{3}{16} = .1875$$

What is the probability that the total showing on the two dice will be 9 or higher?

(3,6)	(4,5)	(4,6)	(5,4)	(5,5)	(5,6)	(6,3)	(6,4)	(6,5)	(6,6)
2	1	2	1	1	2	2	2	2	4

$$\frac{19}{64} = .296875$$

What is the probability that the red die will show a higher number than the white one?



There are 26 cases

$$\frac{26}{64} = \frac{13}{32} = .40625$$

How many bigrams does the sample contain?

$$158 - 1 = 157$$

$$P(. \mid \text{NN})$$

PRP	VBD	DT	JJ	NN	WP	VBD	RB	IN	DT	NN	IN	DT	NNP	NNP	CC	PRP	VBD	VBN	CD	NNS	RB	IN	VBG	DT	NN	.
he	was	an	old	man	who	fished	alone	in	a	skiff	in	the	gulf	stream	and	he	had	gone	eighty-four	days	now	without	taking	a	fish	.

IN	DT	JJ	CD	NNS	DT	NN	VBD	VBN	IN	PRP	.	CC	IN	CD	NNS	IN	DT	NN	DT	NN	POS	NNS	VBD	VBN	PRP	IN	DT
in	the	first	forty	days	a	boy	had	been	with	him	.	but	after	forty	days	without	a	fish	the	boy	's	parents	had	told	him	that	the

JJ	NN	VBD	RB	RB	CC	RB	VBN	,	WDT	VBZ	DT	JJ	NN	IN	JJ	,	CC	DT	NN	VBD	VBN	IN	PRP$	NNS	IN
old	man	was	now	definitely	and	finally	salao	,	which	is	the	worst	form	of	unluck	,	and	the	boy	had	gone	at	their	orders	in

DT	NN	WDT	VBD	CD	JJ	NN	DT	JJ	NN	.	PRP	VBD	DT	NN	JJ	TO	VB	DT	JJ	NN	VB	IN	DT	NN	IN	PRP$
another	boat	which	caught	three	good	fish	the	first	week	.	it	made	the	boy	sad	to	see	the	old	man	come	in	each	day	with	his

NN	JJ	CC	PRP	RB	VBD	IN	TO	VB	PRP	VB	DT	DT	VBD	NNS	CC	DT	NN	CC	NN	CC	DT	NN	WDT	VBD
skiff	empty	and	he	always	went	down	to	help	him	carry	either	the	coiled	lines	or	the	gaff	and	harpoon	and	the	sail	that	was

VBD	IN	DT	NN	.	DT	NN	VBD	VBN	IN	NN	NNS	CC	,	VBD	,	PRP	VBD	IN	DT	NN	IN	JJ	NN	.
furled	around	the	mast	.	the	sail	was	patched	with	flour	sacks	and	,	furled	,	it	looked	like	the	flag	of	permanent	defeat	.

$$\frac{4}{24} = \frac{1}{6} = .1667$$

$$PR(\text{DT}\ \ \text{JJ})$$

"How common is the bigram DT JJ in the sample?"

PRP	VBD	**DT**	**JJ**	NN	WP	VBD	RB	IN	DT	NN	IN	DT	NNP	NNP	CC	PRP	VBD	VBN	CD	NNS	RB	IN	VBG	DT	NN	.
he	was	an	old	man	who	fished	alone	in	a	skiff	in	the	gulf	stream	and	he	had	gone	eighty-four	days	now	without	taking	a	fish	.

IN	**DT**	**JJ**	CD	NNS	DT	NN	VBD	VBN	IN	PRP	.	CC	IN	CD	NNS	IN	DT	NN	DT	NN	POS	NNS	VBD	VBN	PRP	IN	**DT**
in	the	first	forty	days	a	boy	had	been	with	him	.	but	after	forty	days	without	a	fish	the	boy	's	parents	had	told	him	that	the

JJ	NN	VBD	RB	RB	CC	RB	VBN	,	WDT	VBZ	**DT**	**JJ**	NN	IN	JJ	,	CC	DT	NN	VBD	VBN	IN	PRP$	NNS	IN
old	man	was	now	definitely	and	finally	salao	,	which	is	the	worst	form	of	unluck	,	and	the	boy	had	gone	at	their	orders	in

DT	NN	WDT	VBD	CD	JJ	NN	**DT**	**JJ**	NN	.	PRP	VBD	DT	NN	JJ	TO	VB	**DT**	**JJ**	NN	VB	IN	DT	NN	IN	PRP$
another	boat	which	caught	three	good	fish	the	first	week	.	it	made	the	boy	sad	to	see	the	old	man	come	in	each	day	with	his

NN	JJ	CC	PRP	RB	VBD	IN	TO	VB	PRP	VB	DT	DT	VBD	NNS	CC	DT	NN	CC	NN	CC	DT	NN	WDT	VBD
skiff	empty	and	he	always	went	down	to	help	him	carry	either	the	coiled	lines	or	the	gaff	and	harpoon	and	the	sail	that	was

VBD	IN	DT	NN	.	DT	NN	VBD	VBN	IN	NN	NNS	CC	,	VBD	,	PRP	VBD	IN	DT	NN	IN	JJ	NN	.
furled	around	the	mast	.	the	sail	was	patched	with	flour	sacks	and	,	furled	,	it	looked	like	the	flag	of	permanent	defeat	.

$$\frac{6}{157} = .0382$$

$$P(\mathrm{NN} \mid \mathrm{DT} \;\; \mathrm{JJ})$$

"How often does the unigram NN follow the bigram DT JJ?"

"Out of all the DT JJ bigrams, how many of them are followed by NN?"

PRP	VBD	DT	JJ	NN	WP	VBD	RB	IN	DT	NN	IN	DT	NNP	NNP	CC	PRP	VBD	VBN	CD	NNS	RB	IN	VBG	DT	NN	.
he	was	an	old	man	who	fished	alone	in	a	skiff	in	the	gulf	stream	and	he	had	gone	eighty-four	days	now	without	taking	a	fish	.

IN	DT	JJ	CD	NNS	DT	NN	VBD	VBN	IN	PRP	.	CC	IN	CD	NNS	IN	DT	NN	DT	NN	POS	NNS	VBD	VBN	PRP	IN	DT
in	the	first	forty	days	a	boy	had	been	with	him	.	but	after	forty	days	without	a	fish	the	boy	's	parents	had	told	him	that	the

JJ	NN	VBD	RB	RB	CC	RB	VBN	,	WDT	VBZ	DT	JJ	NN	IN	JJ	,	CC	DT	NN	VBD	VBN	IN	PRP$	NNS	IN
old	man	was	now	definitely	and	finally	salao	,	which	is	the	worst	form	of	unluck	,	and	the	boy	had	gone	at	their	orders	in

DT	NN	WDT	VBD	CD	JJ	NN	DT	JJ	NN	.	PRP	VBD	DT	NN	JJ	TO	VB	DT	JJ	NN	VB	IN	DT	NN	IN	PRP$
another	boat	which	caught	three	good	fish	the	first	week	.	it	made	the	boy	sad	to	see	the	old	man	come	in	each	day	with	his

NN	JJ	CC	PRP	RB	VBD	IN	TO	VB	PRP	VB	DT	DT	VBD	NNS	CC	DT	NN	CC	NN	CC	DT	NN	WDT	VBD
skiff	empty	and	he	always	went	down	to	help	him	carry	either	the	coiled	lines	or	the	gaff	and	harpoon	and	the	sail	that	was

VBD	IN	DT	NN	.	DT	NN	VBD	VBN	IN	NN	NNS	CC	,	VBD	,	PRP	VBD	IN	DT	NN	IN	JJ	NN	.
furled	around	the	mast	.	the	sail	was	patched	with	flour	sacks	and	,	furled	,	it	looked	like	the	flag	of	permanent	defeat	.

$$\frac{5}{6} = .833$$

Estimate $PP(\mathrm{DT\ \ JJ} \mid \mathrm{NN})$

"How often would we expect to see DT JJ following NN in the corpus, based on the prior probabilities of unigram NN and bigram DT JJ, and the measured conditional probability $PP(NNNN|DDDD\ JJJJ)$?"

$$PP(DDDD\ JJJJ|NNNN) = \frac{PP(NNNN|DDDD\ JJJJ)\,PP(DDDD\ JJJJ)}{PP(NNNN)}$$

$$= \frac{\frac{5}{6} \times \frac{6}{157}}{\frac{12}{79}} = \frac{395}{1884} = .20966$$

Note: the observed value in the sample is: $\frac{1}{24} = .042$

A = { *gnat*, *beet* } B = { *loon*, *fee* } C = { *peel*, *pool*, *he*, *sand* }

$$PP(hiiiih|AA) = \frac{1}{2}$$

$$PP(hiiiih|BB) = 1$$

$$PP(hiiiih|CC) = \frac{3}{4}$$

$$PP(hiiiih) = PP(hiiiih|AA)PP(AA) + PP(hiiiih|BB)PP(BB) + PP(hiiiih|CC)PP(CC)$$

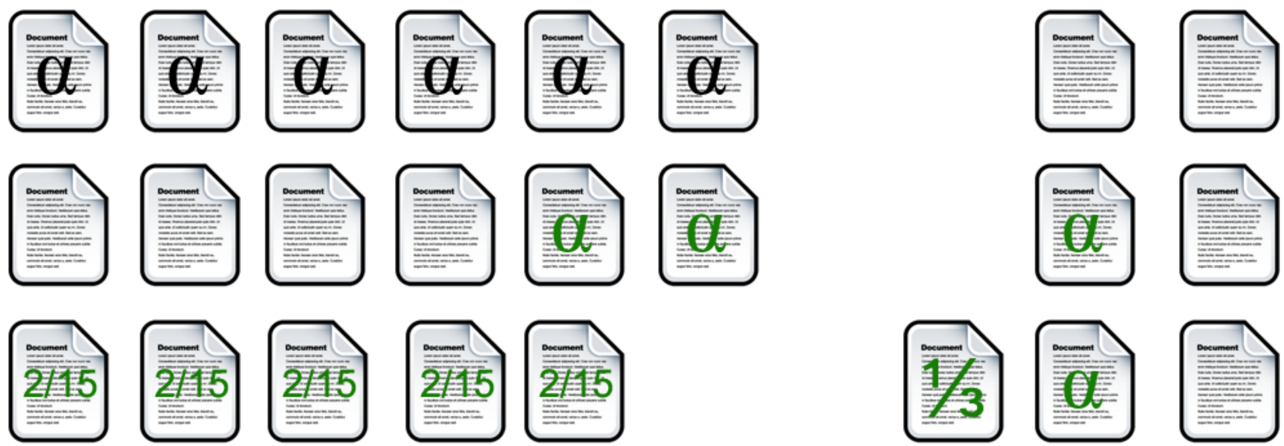
$$PP(hiiiih) = \frac{3}{4}$$

C (yyyyy) $\quad\quad$ $\bar{C}$ (nnnn)

classification result:

gold standard:

after transfer:



D = { the transferred document actually mentions the promoter }
S = { the final selection actually mentions the promoter }

$$P(D) = \frac{1}{3}$$

$$P(S|D) = \frac{2}{3}$$

$$P(\bar{D}) = \frac{2}{3}$$

$$P(S|\bar{D}) = \frac{1}{3}$$

$$P(S) = P(S|D)P(D) + P(S|\bar{D})P(\bar{D})$$

$$= \frac{4}{9}$$

method 1

$$\left(\frac{1}{3} \times \frac{1}{3}\right) + \left(\frac{1}{3} \times 1\right) + \left(\frac{1}{3} \times 0\right)$$

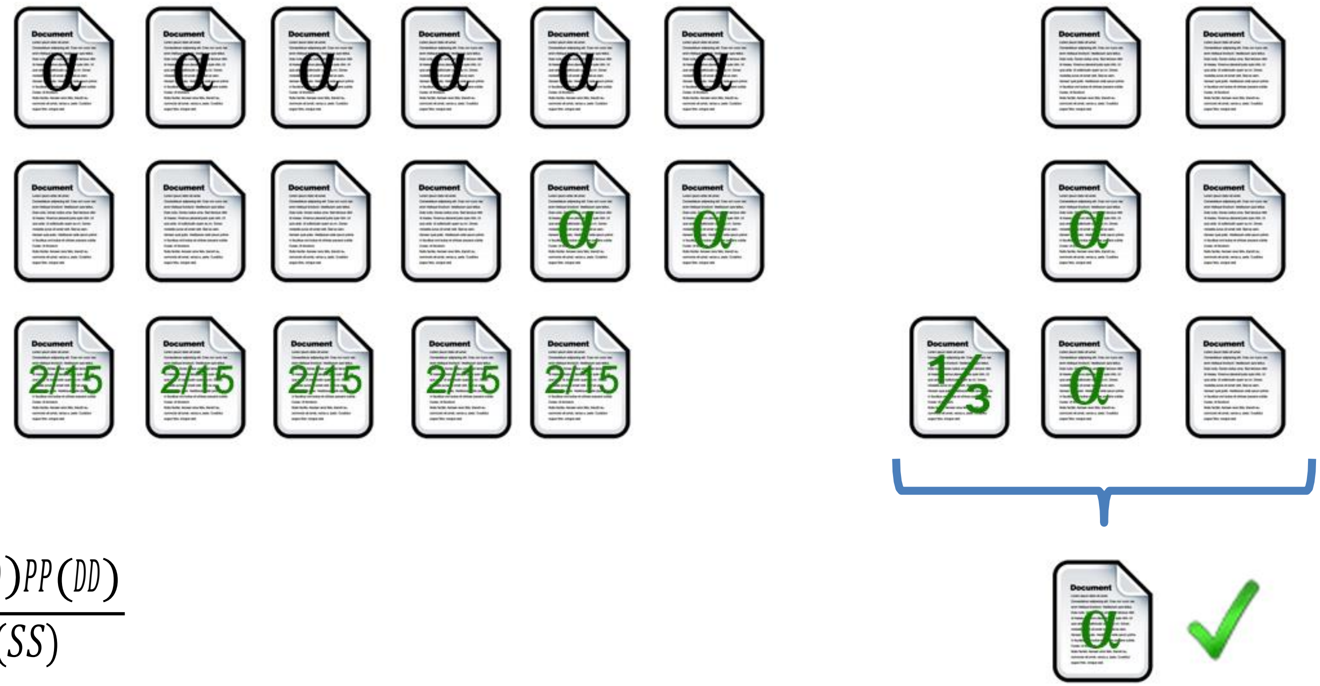
$$= \frac{4}{9}$$

method 2

classification result:

gold standard:

after transfer:



$$P(D|S) = \frac{P(S|D)P(D)}{P(S)}$$

$$P(D|S) = \frac{\frac{2}{3} \times \frac{2}{6}}{\frac{4}{9}}$$

$$P(D|S) = \frac{1}{2}$$

Today's lecture

- Overview of corpus linguistics
- Corpus annotation
- An important tool for automatic annotation: Hidden Markov Model (HMM)
- Case study: HMM Part-of-speech tagger
- Practical issues:
  - Using log-probs to avoid underflow
  - Smoothing unseen values

Corpus linguistics

> The fundamental goal of analysis is to maximize the probability of the observed data.
>
> John Goldsmith, Univ. of Chicago

- Data is important
- It makes (machine) learning possible
- In computational linguistics, our data is organized into corpora. This word is the plural of corpus.

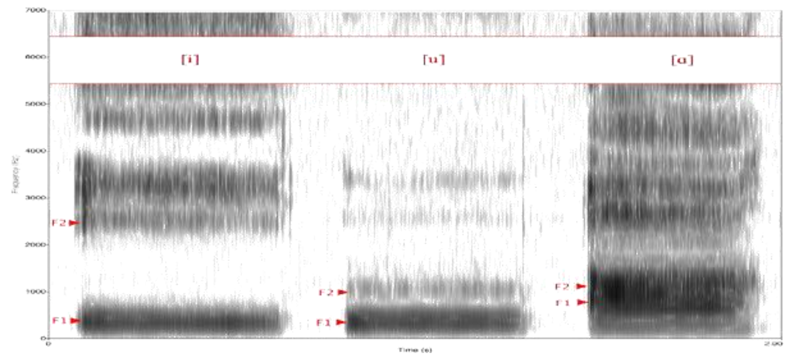
Corpora

- What is a corpus?
  - A collection of text or recorded speech—typically in machine-readable form—compiled to be representative of a particular kind of language.
  - Used as a starting point for quantitative, empirical linguistic research or language description
- Corpus characteristics:
  - Raw
  - Tagged/Annotated (i.e. Penn Treebank)
    - Automatic tagging
    - Human annotation
    - Hybrid approach: automated system refers cases it is unsure of to human annotation

Annotation

- In Project 2 (unigram tallies), we gathered statistics from a raw corpus
- In Project 4 (DNA targets), we are searching a raw corpus, a basic form of Information Extraction (IE)
- In Project 1 (PTB constituents), we gathered statistics from an annotated corpus
- Annotation adds value to a corpus by increasing the number of statistical dimensions we can attempt to correlate. This applies to both
  - automatic methods (machine learning)
  - rule-based (analytical methods)

Annotating audio corpora



- Phonetic transcription
- Phonemic transcription
- Text transcription
- Speaker ascription (discourse/dialogue)
- Formant analysis (vowel resonances)
- Prosody
- Start/stop timings
- FFT analysis (frequencies)
- Gesture correlation

PRAAT is an amazing free tool for phonetic analysis of human speech
http://www.fon.hum.uva.nl/praat/

Annotating text corpora

- Sentence identification (sentence breaking)
- Word identification (tokenization, word-breaking)
- Part-of-speech (POS)
  - http://cst.dk/online/pos_tagger/uk/index.html

- Named entities (NER)
  - http://alias-i.com/lingpipe/web/demo-ne.html
- Anaphora resolution
- Semantic analysis
  - http://redwoods.stanford.edu/

Example: annotating information structure

- Linguistic *information structure* is concerned with the management of elaboration between speaker and hearer in discourse
- This sub-field introduces the notions of:
  - topic (what a proposition is "about")
  - focus (the new information that is being asserted about the topic)

Glenn Slayden. 2010. An Information Structure Annotation of Thai Narrative Fiction. In *University of Washington Working Papers in Linguistics* (UWWPL) (*in press*).

Example: annotating information structure

topic

(I hope Sandy likes the iPod Kim gave her.)
"It's a [BOOK$_F$] that Kim gave Sandy." (not an iPod)
(correctional focus)

(What's in the bag?)
"It's [a book that Kim gave Sandy$_F$]."
(argument focus)

(What did Allie do?)
"She [went to the cricket match$_F$]."
(predicate focus)

Part-of-speech (POS) tagging

- Automatic POS-tagging of a corpora is a fundamental task in computational linguistics
- This task is a prerequisite for building many types of statistical models

Corpus priors **POS n-grams** **lemma n-grams**

DT	NN	VBD	RB	IN	DT	NN	,	CC	DT	VBG	NNS	VBD	DT	NN	VBD
the	cold	passed	reluctantly	from	the	earth	,	and	the	retiring	fogs	revealed	an	army	stretched

IN	IN	DT	NNS	,	VBG	.	IN	DT	NN	VBN	IN	JJ	TO	VB	,	DT	NN	VBN
out	on	the	hills	,	resting	.	as	the	landscape	changed	from	brown	to	green	,	the	army	awakened

,	CC	VBD	TO	VB	IN	NN	IN	DT	NN	IN	NNS	.
,	and	began	to	tremble	with	eagerness	at	the	noise	of	rumors	.

POS tagging

Objective: given sentence

$$S = (ww_0, ww_1, \ldots ww_{nn}),$$

determine tags

$$T = (tt_0, tt_1, \ldots tt_{nn}).$$

DT	NN	VBD	RB	IN	DT	NN	,	CC	DT	VBG	NNS	VBD	DT	NN	VBD
the	cold	passed	reluctantly	from	the	earth	,	and	the	retiring	fogs	revealed	an	army	stretched

IN	IN	DT	NNS	,	VBG	.	IN	DT	NN	VBN	IN	JJ	TO	VB	,	DT	NN	VBN
out	on	the	hills	,	resting	.	as	the	landscape	changed	from	brown	to	green	,	the	army	awakened

,	CC	VBD	TO	VB	IN	NN	IN	DT	NN	IN	NNS	.
,	and	began	to	tremble	with	eagerness	at	the	noise	of	rumors	.

Human annotation

- How to proceed with human tagging is obvious

If cost is no object, is this certainly the “best” thing to do?

- Not necessarily. It is very hard to get consistent results
  - Clear standards and procedures must be defined
  - Empirical quality control sampling is advisable
  - Automatic methods are likely to be more consistent

Automatic tagging does not "pollute" the corpus

POS Tagging

annotation

raw

DT	NN	VBD	RB	IN	DT	NN	,	CC	DT	VBG	NNS	VBD	DT	NN	VBD
the	cold	passed	reluctantly	from	the	earth	,	and	the	retiring	fogs	revealed	an	army	stretched

IN	IN	DT	NNS	,	VBG	.	IN	DT	NN	VBN	IN	JJ	TO	VB	,	DT	NN	VBN
out	on	the	hills	,	resting	.	as	the	landscape	changed	from	brown	to	green	,	the	army	awakened

,	CC	VBD	TO	VB	IN	NN	IN	DT	NN	IN	NNS	.
,	and	began	to	tremble	with	eagerness	at	the	noise	of	rumors	.

Machine Translation

Document Summarization

Information Extraction

Deep (semantic) Processing

Question Answering

Note on notation

"probability that a noun follows a determiner"

- Before, when we looked at n-gram probabilities such as $PP(\mathrm{NN}|\mathrm{DT})$, the conditional "given" symbol '|' meant "reading left-to-right," more precisely:

$$PP(tt_{ii} \mid tt_{ii-1})$$

- We can also refer to the probability of a tag given its word, $PP(tt_{ii} \mid ww_{ii})$ or the reverse, $PP(ww_{ii} \mid tt_{ii})$.

See next 2 slides

- So we need to pay careful attention to the variables and the subscripts

⚠ Deciphering subscript-less notation

$PP(NNNN)$	Probability of a noun (versus all POS unigrams)
$PP(NNNN\|DDDD)$	Probability that a noun follows a determiner
$PP(NNNN\ DDDD)$ sometimes you'll see: $PP(NNNN, DDDD)$	Probability of the POS bigram "NN DT" (versus all POS bigrams)
$PP(\text{the}\|DDDD)$	Probability of a determiner being the word "the"
$PP(DDDD\|\text{the})$	(i.e.) Probability of tagging the word "the" as a determiner
$PP(NNNN\|DDDD\ JJJJ)$	Probability of a noun following the bigram "DT JJ"

$PP(tt_{ii-1}, tt_{ii})$

$PP(tt_{ii}|tt_{ii-2}, tt_{ii-1})$

$PP(tt_{ii}|ww_{ii})$

$PP(ww_{ii}|tt_{ii})$

I don't like the $PP(NNNN, DDDD)$ notation (with a comma), because it implies joint probability, which is normally *commutative*, but we have an ordering constraint such that $PP(NNNN\ DDDD) \neq PP(DDDD\ NNNN)$. This problem is avoided by using subscripts in $PP(tt_{ii-1}, tt_{ii})$, where the comma is ok. In either case, terms should always be written in sentence appearance order.

$$PP(tt|ww)$$

- This type of notation can refer to either:
  - a corpus prior, that is the *observed* (counted) probability of tag tt in the corpus, restricted by word ww.

    i.e. appearing on the **right** side of Bayes' theorem

  - a model term, which is typically used as part of the model's maximized objective function.

    i.e. appearing on the **left** side of Bayes' theorem

  - What's a *maximized objective function*? First, let's define a handy math notation helper, called **argmax**…

$$\mathrm{argmax}_{xx} ff(xx)$$

The result of this expression is:

the value (or values) xx such that $ff(xx)$ is maximized.

argmin works in a similar way

- We don’t care about the actual evaluation result of the function $ff(xx)$. It is discarded.
- You will see this notation often in computational linguistics

ArgMax<TSrc,TArg>

```
public static TSrc ArgMax<TSrc, TArg>(this IEnumerable<TSrc> seq, Converter<TSrc, TArg> objective)
    where TArg : IComparable<TArg>
{
    IEnumerator<TSrc> e = seq.GetEnumerator();
    if (!e.MoveNext())
        throw new InvalidOperationException("Sequence has no elements.");

    TSrc t = e.Current;
    if (e.MoveNext())
    {
        TArg v, max_val = objective(t);
        do
        {
            TSrc t_try = e.Current;
            v = objective(t_try);
            if (v.CompareTo(max_val) > 0)
            {
                t = t_try;
                max_val = v;
            }
        }
        while (e.MoveNext());
    }
    return t;
}
```

example

$$f(x) = (x-3)^2$$

$$\operatorname{argmin}_x f(x) = 3$$

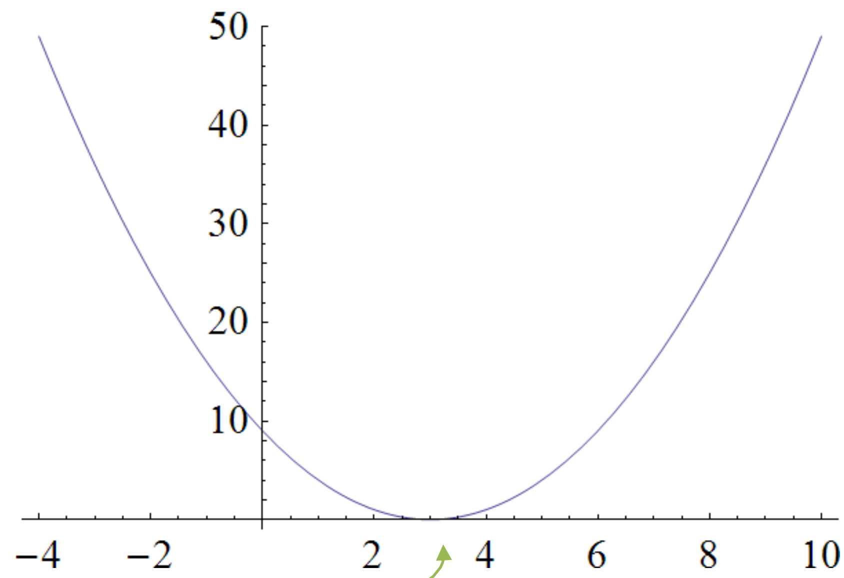
or

$$\operatorname{argmin}_x (x-3)^2 = 3$$

In[3]:=

```
Plot[(x - 3)^2, {x, -4, 10}]
```

Out[3]=



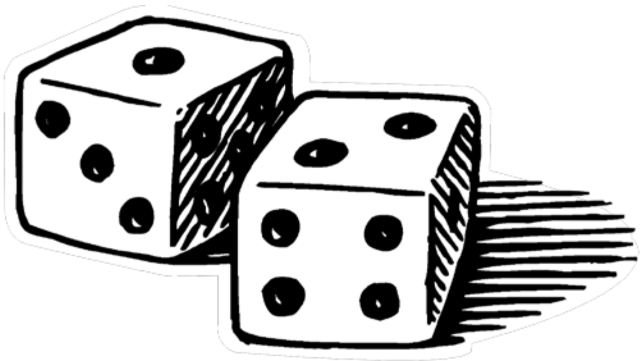
The value of $f(x)$ at 3 is 0, but argmin doesn't care about that, so long as it's the minimum value

argmax example #1

$XX =$ { the total showing on two fair dice }

What is the value of:

$$\text{argmax}_{xx} PP(XX = xx)$$



7

argmax example #2

$$X = \{ \text{ a sample of English language text } \}$$

$$f(x) = |(X|x)| \, , x \in \{ \text{'a'}, \text{'b'}, \text{'c'}, \ldots \text{'z'} \}$$

What is the value of:

$$\text{argmax}_x f(x)$$

'e'

Tagging objective function

Predict a sequence of tags t based on the probability of tags and words $P(t_i | w_i)$. Given sentence

"t-hat"

$$S = (w_0, w_1, \ldots w_n)$$
$$t = \mathbf{argmax}_{t_i} P(t_i \mid w_i).$$

"t is the best sequence of tags that match a tag t_i to its word w_i."

This material is also covered in section 5.5 (p.139) of Jurafsky & Martin, 2nd ed.

Simplistic tagger

$$S = (w_0, w_1, \ldots w_n)$$
$$t = \operatorname{argmax}_{t_i} P(t_i \mid w_i)$$

repeated from last slide

This is surely the function we want to maximize, but it's not clear how to calculate the probabilities $P(t|w)$.

Simplistic tagger: Why don't we use probabilities calculated from a corpus ?

- like you did for Assignment 3

Simplistic tagger

DT	NN	VBD	RB	IN	DT	NN	,	CC	DT	VBG	NNS	VBD	DT	NN	VBD
the	cold	passed	reluctantly	from	the	earth	,	and	the	retiring	fogs	revealed	an	army	stretched

IN	IN	DT	NNS	,	VBG	.	IN	DT	NN	VBN	IN	JJ	TO	VB	,	DT	NN	VBN
out	on	the	hills	,	resting	.	as	the	landscape	changed	from	brown	to	green	,	the	army	awakened

,	CC	VBD	TO	VB	IN	NN	IN	DT	NN	IN	NNS	.
,	and	began	to	tremble	with	eagerness	at	the	noise	of	rumors	.

$$\operatorname{argmax}_{t} P(t|\text{the}) = \texttt{DT}$$

$$\operatorname{argmax}_{t} P(t|\text{cold}) = \texttt{JJ}$$

How well does the simplistic tagger work?

- Such a POS tagger is not really usable

Most probable POS tag: JJ
Correct tag: NN

Most probable POS tag: VBG
Correct tag: JJ

DT		VBD	RB	IN	DT	NN	,	CC	DT		NNS	VBD	DT	NN	VBD
the	cold	passed	reluctantly	from	the	earth	,	and	the	retiring	fogs	revealed	an	army	stretched

IN	IN	DT	NNS	,	VBG	.	IN	DT	NN	VBN	IN	JJ	TO	JJ	,	DT	NN	VBN
out	on	the	hills	,	resting	.	as	the	landscape	changed	from	brown	to	green	,	the	army	awakened

,	CC	VBD	TO	VB	IN	NN	IN	DT	NN	IN	NNS	.
,	and	began	to	tremble	with	eagerness	at	the	noise	of	rumors	.

Use Bayes Theorem

Of course, you have this memorized

$$PP(AA|BB) = \frac{PP(BB|AA)PP(AA)}{PP(BB)}$$

Remember, this was our objective function

$$tt = \mathrm{argmax}_{tt} PP(tt_{ii} | ww_{ii})$$

$$tt = \mathrm{argmax}_{tt} \frac{PP(ww_{ii} | tt_{ii})PP(tt_{ii})}{PP(ww_{ii})}$$

This is one of the most important slides of this entire class

For each evaluated value of ii, $PP(ww_{ii})$ will be the same. We can cancel it.

$$tt = \text{argmax}_{tt} \frac{PP(ww_{ii} | tt_{ii})PP(tt_{ii})}{\cancel{PP(ww_{ii})}}$$

$$tt = \text{argmax}_{tt} PP(ww_{ii} | tt_{ii})PP(tt_{ii})$$

The best sequence of tags is determined by the probability of each word given its tag and also the probability of that tag.

I repeat…

$$t = \mathrm{argmax}_{t} P(w_i \mid t_i) P(t_i)$$

“likelihood”

“prior”

“We compute the most probable tag sequence… by multiplying the **likelihood** and the **prior probability** for each tag sequence and choosing the tag sequence for which this product is greatest.

“Unfortunately, this is still too hard to compute directly…”

Jurafsky & Martin (paraphrase) p.140

We still need to make some assumptions.

we'll come back to this part later

$$t = \text{argmax}_t P(w_i | t_i) P(t_i)$$

Assumption 1: If we want to use corpus probabilities to estimate $P(w_i | t_i)$, we need to formally note that we're assuming

$$P'(w_i | t_i) \approx ?_{i} P(w_i | t_i)$$

"The only POS tag a word depends on is its own."

Any progress?

- So wait: if we're assuming the only *POS tag* a *word* depends on is its own, how is this going to be better than the simplistic tagger from before, which assumed that the only *word* a *POS tag* depends on is its own?
- In other words, Why is $P(w|t)$ going to work better than $P(t|w)$?
- Hint: $|\Omega|$
- Hint: $|T| \ll |W|$

Answer: because there are a lot more distinct words than tags, conditioning on *tags* rather than *words* increases the resolution of the corpus measurements

example

$$P(\text{cold}|\text{NN}) = .00002$$
$$P(\text{cold}|\text{JJ}) = .00040$$

$$P(\text{JJ}|\text{cold}) = .97$$
$$P(\text{NN}|\text{cold}) = .03$$

This value will drown out our calculation and we'd never tag "cold" as a noun!

$$t = \text{argmax}_t P'(w_i | t_i) P(t_i)$$

Assumption 2: The only tags that a tag t_i depends on are the n previous tags, $t_{i-n-1} \ldots t_{i-1}$. For example, in a POS bigram model:

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

This is known as the bigram assumption: "The only POS tag(s) a POS tag depends on are the ones immediately preceding it."

Putting it together

$$tt = \text{argmax}_{tt} PP\,(ww_i | tt_{ii}) PP(tt_{ii})$$

$$PP'(ww_{ii} | tt_{ii}) \approx \text{?} \, PP(\;|\;) \quad ww_{ii}\; tt_{ii} \quad ii$$

$$PP'(tt_{ii}) \approx \text{?} \, PP(\; | tt_{ii}\; tt_{ii-1}) \quad ii$$

$$tt = \text{argmax}_{tt} \text{?}_{ii} \, PP(ww_{ii} | tt_{ii}) \; \text{?}_{ii} \, PP(tt_{ii} | tt_{ii-1})$$

$$tt = \text{argmax}_{tt} \text{?}_{ii} \, PP(ww_{ii} | tt_{ii}) \, PP(tt_{ii} | tt_{ii-1})$$

Reminder: estimating $P(w_i|t_i)$ from a corpus

Definition of conditional probability

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

$$P(A|B) = \frac{\frac{\text{count}(A,B)}{\cancel{|\Omega|}}}{\frac{\text{count}(B)}{\cancel{|\Omega|}}}$$

word likelihood

$$P(w_i|t_i) = \frac{\text{count}(w_i,t_i)}{\text{count}(t_i)}$$

Reminder: estimating $P(t_i \mid t_{i-1})$ from a corpus

Definition of conditional probability

$$P(A \mid B) = \frac{P(A, B)}{P(B)}$$

$$P(A \mid B) = \frac{\dfrac{\text{count}(A, B)}{\cancel{|\Omega|}}}{\dfrac{\text{count}(B)}{\cancel{|\Omega|}}}$$

$$P(t_i \mid t_{i-1}) = \frac{\text{count}(t_{i-1}, t_i)}{\text{count}(t_{i-1})}$$

POS tagging objective function

$$tt = \text{argmax}_{tt} \prod_{ii} \left(\frac{\text{count}(ww_{ii}, tt_{ii})}{\text{count}(tt_{ii})} \times \frac{\text{count}(tt_{ii-1}, tt_{ii})}{\text{count}(tt_{ii-1})} \right)$$

Best POS tag sequence

How often does word ww_{ii} occur with tag tt_{ii} in the corpus?

How often does tt_{ii} follow tt_{ii-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 $PP(ww|tt)$, 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?'"

DT		VBD	RB	IN	DT	NN	,
the	cold	passed	reluctantly	from	the	earth	,

"If we are expecting an **adjective**, how likely is it that the adjective will be 'cold?'" (high) **WEIGHTED BY** our chance of seeing the sequence **DT JJ** (medium)

versus

"If we are expecting a **noun**, how likely is it that the noun will be 'cold?'" (medium) **WEIGHTED BY** our chance of seeing the sequence **DT NN** (very high)

THE WINNER: NN

Multiplying probabilities

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

$$0 \leq pp \leq 1$$

- 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

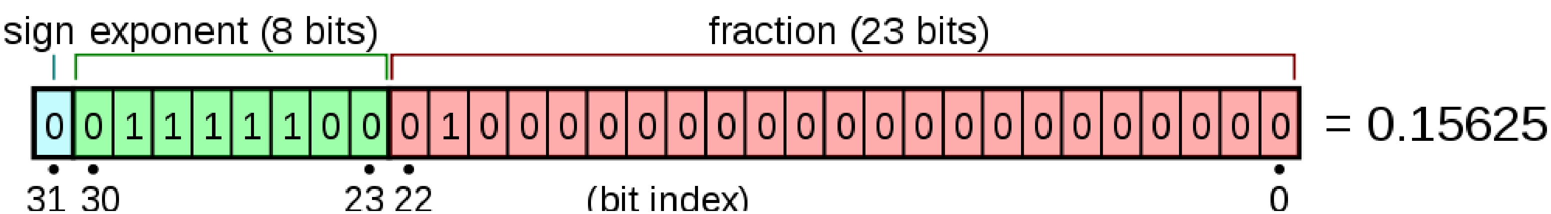
- When multiplying many probability terms together, we need to prevent underflow
  - Due to limitations in the computer's internal representation of floating point numbers, the product quickly becomes zero
- We usually work with the logarithm of the probability values
- This is known as the "log-prob"

$$= \log_{10} pp$$

IEEE 754 floating point

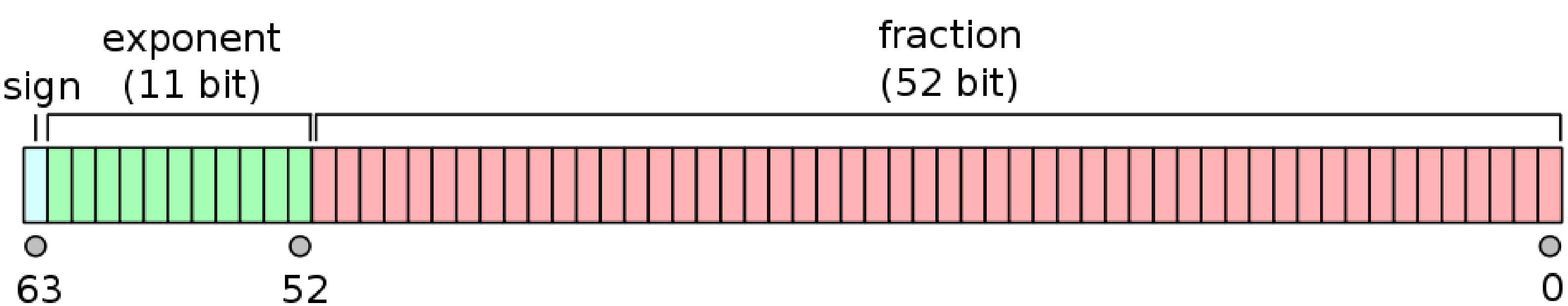
- 32-bit “single” “float”

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



- 64-bit “double”

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



logarithms refresher

definition: $\log_{bb} xx = yy \colon xx = bb^{yy}$

$$bb^{xx} \times bb^{yy} = bb^{xx+yy}$$

$$\log xxyy = \log xx + \log yy$$

$$\log \underset{ii}{?} xx_{ii} = \underset{ii}{?} \log xx_{ii}$$

$$\frac{bb^{xx}}{bb^{yy}} = bb^{xx-yy}$$

$$\log\frac{xx}{yy} = \log xx - \log yy$$

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?

$$tt = \mathrm{argmax}_{tt} \underset{ii}{?} \left(\frac{\mathrm{count}(ww_{ii}, tt_{ii})}{\mathrm{count}(tt_{ii})} \times \frac{\mathrm{count}(tt_{ii-1}, tt_{ii})}{\mathrm{count}(tt_{ii-1})} \right)$$

$$tt = \mathrm{argmax}_{tt} \underset{ii}{?} \left(\log \frac{\mathrm{count}(ww_{ii}, tt_{ii})}{\mathrm{count}(tt_{ii})} + \log \frac{\mathrm{count}(tt_{ii-1}, tt_{ii})}{\mathrm{count}(tt_{ii-1})} \right)$$

Wait, how can you do that, there was no “log” outside of 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

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

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

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

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

Dynamic programming

- The Viterbi algorithm is typically used to decode Hidden Markov Models
  - You might get to implement it in Ling 570
- It is a dynamic programming technique
  - We maintain a trellis of partial computations
- This approach reduces the problem to $OO(|DD|^2 nn)$ 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

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

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
- 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

Next time

- Formal grammars
- Context-free grammars
  - Production rules
  - Lexical rules
- Chomsky normal form
- Parsing

C# Tutorial (continued...)

Interfaces

- `IEnumerable<T>` is one of many system-defined interfaces that a class can elect to implement

An interface is a named set of zero or more function signatures with no implementation(s)

- To implement an interface, a class defines a matching implementation for every function in the interface
- Interfaces are sometimes described as contracts
- You can define and use a reference to an interface just like any other object reference

Contrived Example

```
interface IPropertyGetter
{
    String GetColor();
}

class Strawberry : IPropertyGetter
{
    public String GetColor() { return "red"; }
}

class Ferrari : IPropertyGetter
{
    public String GetColor() { return "yellow"; }
}
```

- This looks like C++ class inheritance
  - yes, but it’s more ad-hoc
  - C# classes can have single inheritance of other classes, and multiple inheritance of interfaces
  - Interfaces can inherit from other interfaces (not shown)

IEnumerable<T>

- This is one of the simplest interfaces defined in the BCL (base class libraries)
- This interface provides just one thing: a way to iterate over elements of type T
- All of the system arrays, collections, dictionaries, hash sets, etc. implement `IEnumerable<T>`
  - Implementing IEnumerable<T> on your own classes can be very useful, but you don't need to worry about that
  - For now, what's important is that you get to use it, because it's available on all of the system collections

IEnumerator<T>

- IEnume**rable**<T> has only one function, which allows a caller or caller(s) to obtain an *enumera*<u>*tor*</u> object which is able to iterate over elements
  - The actual enumerator object is an object that implements a different interface, called IEnumera**tor**<T>
  - This "factory" design allows a caller to initiate and maintain several simultaneous iterations if needed
  - The enumerator object, IEnumerator<T> can only:
    - Get the current element
    - Move to the next element
    - Tell you if you've reached the end
  - Note: There's no count
    - ICollection inherits from IEnumerable to provide this

Interfaces as function arguments

- Using interfaces as function arguments allows you to require the absolute minimum functionality the function actually needs
- In this way, the ad-hoc nature of interfaces allows us to comply with the maxim

```
void ProcessSomeStrings(IEnumerable<String> the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

Now, this function is exposing the weakest (most general) requirement possible for the processing it has to do. This provides more flexibility to callers since they can choose whatever level of specificity is convenient. The function can be used in the widest possible variety of situations.

Example

```
String[] d1 = { "able", "bodied", "cows", "don't", "eat", "fish" };
ProcessSomeStrings(d1);

List<String> d2 = new List<String> { "clifford", "the", "big", "red", "dog" };
ProcessSomeStrings(d2);

HashSet<String> d3 = new HashSet<String> { "these", "must", "be", "distinct" };
ProcessSomeStrings(d3);

Dictionary<String,int> d4 =
        new Dictionary<String, int> { { "the", 334596 }, { "in", 153024 } };
ProcessSomeStrings(d4.Keys);

void ProcessSomeStrings(IEnumerable<String> the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

Python users might not be impressed, but the difference is that this is all 100% strongly typed

Iteration is efficient

- That’s cool, IEnumerable<T> lets a function not care about where a sequence of elements is coming from
  - We don’t copy the elements around
  - Iterators let us access elements right from their source
- All of those examples iterate over elements that already exist somewhere
- Is there a way to iterate over data that’s generated on-the-fly, doesn’t exist yet, or is never persisted at all?
- Yes!

Iterating over on-the-fly data

```
IEnumerable<String> GetNewsStories(int desired_count)
{
    for (int i = 0; i < desired_count; i++)
        yield return RealtimeNewswireSource.GetLatestStory();
}
```

see next slide

```
// ...
IEnumerable<String> d5 = GetNewsStories(7);
ProcessSomeStrings(d5);
// ...
```

This is exactly the same as before, but this time there’s no “collection” of elements sitting anywhere

```
void ProcessSomeStrings(IEnumerable<String> the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

This function doesn’t care. In fact, it can’t even tell.

yield keyword

- The `yield` keyword makes it easy to define your own custom iterator functions
- Any function that contains the `yield` keyword becomes special
  - It must be declared as returning an IEnumerable<T>
  - Deferred execution means that the function's body is not necessarily invoked when you "call" it
  - It must deliver zero or more elements of type T using:

    ```
    yield return t;
    ```
  - Sometime later, control may continue immediately after this statement to allow you to yield additional elements
  - It may signal the end of the sequence by using:

    ```
    yield break;
    ```

Custom iterator function example

```
IEnumerable<String> GetNewsStories(int desired_count)
{
    for (int i = 0; i < desired_count; i++)
        yield return RealtimeNewswireSource.GetLatestStory();
}

// ...
IEnumerable<String> d5 = GetNewsStories(7);
ProcessSomeStrings(d5);
// ...

void ProcessSomeStrings(IEnumerable<String> the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

code from this custom iterator function is *not* executed at this point.

d5 refers to an iterator that "knows how" to get a certain sequence of strings when asked

This finally demands the strings, causing our custom iterator function to execute–interleaved with this loop!