Handout 7: Language Models

1. Simple generative model of text

a. Choose a random token to start. (Why choose from tokens rather than types?)

```
>>> from nltk.corpus import gutenberg
>>> emma = gutenberg.words('austen-emma.txt')
>>> import random
>>> w = random.choice(emma)
>>> w
u'mind'
```

b. Choose a random bigram with that word on the left

```
>>> w = random.choice([r for (l,r) in bigrams(emma) if l == w])
>>> w
u'like'
```

c. Repeat

```
>>> w = random.choice([r for (l,r) in bigrams(emma) if l == w])
>>> w
u'to'
```

d. Write a function called `brute_generate` to generate a random string of length n. It should return a generator.

e. `brute_generate` is very slow! Can we do better?

2. Sampling from a frequency distribution

a. Consider a simple case:

```
>>> tokens = ['a', 'a', 'a', 'a', 'b', 'b', 'c']
>>> ud = FreqDist(tokens)
>>> ud.N()
7
```

b. Choose a target in `range(len(tokens))`.

```
>>> tgt = random.randrange(ud.N())
>>> tgt
4
>>> tokens[tgt]
'b'
```

c. Compute that from `ud`. Is `tgt` in the "a" range?

```
>>> runningtotal = ud['a']
>>> runningtotal
4
>>> tgt < runningtotal
```

```
False
>>> runningtotal += ud['b']
>>> runningtotal
6
>>> tgt < runningtotal
True
```

d. So:

```
def fdchoose (fd):
    tgt = random.randrange(fd.N())
    rtot = 0
    for key in fd:
        rtot += fd[key]
        if tgt < rtot:
            return key
```

3. A **language model**

a. Example:

```
>>> lm = (FreqDist(emma), ConditionalFreqDist(bigrams(emma)))
```

b. Define `generate` to use the language model. After the first word, we want to sample from the distribution of words following w. How do we do that?

c. Usage:

```
>>> ' '.join(generate(model, 10))
u'for beauty happened which was pained by a regard to her'
```

4. A language model defines a probability distribution over texts

a. Suppose we train (estimate) a language model from our running example text `abbdabdbbd`. Model M_1:

	a	b	d
ud	0.2	0.5	0.3
a	0	1	0
b	0	0.4	0.6
d	0.667	0.333	0

b. What is the probability that we choose “b” as the first word?

c. What is the probability that we choose “b” as the next word?

d. So what is the probability of starting “b b”?

e. What is the probability of generating the text “b b d a”?

5. Comparing language models

a. An alternative language model, M_2:

	a	b	d
ud	0.2	0.5	0.3
a	0.2	0.5	0.3
b	0.2	0.5	0.3
d	0.2	0.5	0.3

b. What is the probability of M_2 generating "b b d a"?

c. Which model is more **likely**, given the observation "b b d a"?

6. Bayes' Rule

a. Def. of cond. prob.: $p(M|D) = p(M,D)/p(D)$

b. Chain rule: $p(M,D) = p(M)\, p(D|M)$

c. Hence:

$$\overbrace{p(M|D)}^{\text{posterior}} = \frac{\overbrace{p(M)}^{\text{prior}} \cdot \overbrace{p(D|M)}^{\text{likelihood}}}{p(D)}$$

d. Likelihood is probability of data given model.

e. If models have equal prior probability, highest-likelihood model is the same as highest-posterior model

7. A technical issue

a. Computing language-model likelihoods involves multiplying small numbers together. We may run into problems of **underflow**.

```
>>> p = 1
>>> for n in [.1] * 1000: p *= n
>>> p
0.0
>>> q = 1
>>> for n in [.2] * 1000: q *= n
>>> q
0.0
>>> q > p
False
```

b. The fix: use logarithms

$$p = 10^{\log p}$$

$$pq = 10^{\log p}\, 10^{\log q} = 10^{\log p + \log q}$$

$$\log pq = \log p + \log q$$

$$\log p_1 q_1 > \log p_2 q_2 \quad \text{if and only if} \quad p_1 q_1 > p_2 q_2$$

c. Example

```
>>> import math
>>> math.log10(.1 * .1 * .1)
-3.0
>>> math.log10(.1) + math.log10(.1) + math.log10(.1)
-3.0
>>> math.log10(.1) * 1000
-1000.0
>>> math.log10(.2) * 1000
-698.9700043360187
```

d. Log probability of "b b d a" according to M_1 and M_2

```
>>> lp1 = math.log10(.5) + math.log10(.4) + math.log10(.6) + math.log10(2/3)
>>> lp1
-1.0969100130080565
>>> lp2 = math.log10(.5) + math.log10(.5) + math.log10(.3) + math.log10(.2)
>>> lp2
-1.8239087409443187
>>> lp1 > lp2
True
```

8. Log probability as **cost**

a. We can think of $(-\log p(y|x))$ as the cost that the model pays if the bigram (x, y) occurs in testing.

Bet:	p	.1	.2	.3	.4	.5	.6	.7	.8	.9
Pay:	$-\log p$	1	.70	.52	.40	.30	.22	.15	.10	.05

b. For test data "b b d a":

	b	b	d	a	Total
M_1 pays:	.30	.40	.22	.18	1.10
M_2 pays:	.30	.30	.52	.70	1.82

c. The total cost is called the **cross entropy** of the model with the testing data

9. Overfitting

a. Model M_1 **fits** the training data better: it extracts more information, assigns higher probabilities to the training examples

b. It never saw `ac` in training, so it assumes that `ac` is <u>impossible.</u> What if that assumption is wrong?

c. What is the cost for a bet $p(x) = 0$, if x actually occurs in testing?

d. What are the model likelihoods and cross entropies if the **test data** is "b b a d"?

		b	b	a	d	
M_1	p	.5	.4	0	0	0
	$-\log p$	.30	.40	∞	∞	∞
M_2	p	.5	.5	.2	.3	.015
	$-\log p$	.30	.30	.70	.52	1.82

10. Fit and simplicity

a. Model M_1 **overfits** the training data: it bets too heavily that the future will look exactly like the past.

b. Model M_2 **underfits** the training data: it throws away too much information.

c. There is a trade-off between **fit** and **simplicity**. Recording less information makes for a smaller model, but it also fits the training data less well.

11. **Smoothing**: finding a better trade-off

a. We could beat M_2 on both test sets if we model bigram probabilities (like M_1), but hedge our bets

b. Pretend every bigram occurs ϵ more often than it actually occurs. Smoothed counts for training `abbdabdbbd`:

	a	b	d	Sum
a	ϵ	$2+\epsilon$	ϵ	$2+3\epsilon$
b	ϵ	$2+\epsilon$	$3+\epsilon$	$5+3\epsilon$
d	$2+\epsilon$	$1+\epsilon$	ϵ	$3+3\epsilon$

c. General formula. Let n be the number of different types.

$$\tilde{p}(y|x) = \frac{c(x,y)+\epsilon}{c(x)+n\epsilon} \qquad \tilde{p}(x) = \frac{c(x)+n\epsilon}{N+n^2\epsilon}$$

12. For our example, with $\epsilon = .5$:

a. Model M_3, probabilities and costs:

	a	b	d
ud	3.5/14.5	6.5/14.5	4.5/14.5
a	.5/3.5	2.5/3.5	.5/3.5
b	.5/6.5	2.5/6.5	3.5/6.5
d	2.5/4.5	1.5/4.5	.5/4.5

	a	b	d
ud	.62	.35	.51
a	.85	.15	.85
b	1.11	.41	.27
d	.26	.48	.95

b. Costs on test sets:

		M_3	M_1	M_2
$bbda$	$.35+.41+.27+.26=$	1.29	1.10	1.82
$bbad$	$.35+.41+1.11+.85=$	2.72	∞	1.82

Classes

13. Defining `LangModel` as a class

```
class LangModel (object):

    def __init__ (self, text):
        self.unigram = FreqDist(text)
        self.bigram = ConditionalFreqDist(bigrams(text))
```

14. Creating an **instance**:

```
>>> lm = LangModel('abbdabdbbd')
>>> lm
<LangModel instance>
>>> lm.unigram
<FreqDist>
>>> lm.unigram.freq('b')
0.5
>>> lm.bigram['b'].freq('d')
0.6
```

15. Defining a method:

```
class LangModel (object):
    ...

    def cost (self, test):
        c = math.log10(self.unigram.freq(test[0]))
        for i in range(1, len(test)):
            c += math.log10(self.bigram[test[i-1]].freq(test[i]))
        return c
```

16. Calling it:

```
>>> lm = LangModel('abbdabdbbd')
>>> lm.cost('bbda')
-1.0969100130080565
```