In this lecture we present issues underlying statistical decisions in NLP, mostly in the context of *parameter estimation* (and not so much hypothesis testing, which also need to be discussed).

1 Road Map

In order to place this step in the context of the future lectures:

We start by discussing *how* to estimate the probability of events. We will later discuss a bit of how to use these in decision making.

We will observe that there are several problems with making decisions based on these counts, even when we can handle the *how* questions.

One aspect of these problems is related to the *what* question: we want to count more important events. Beyond the important issue of deciding *what* to count, we will need to develop ways to abstract away from the observed data. This is where we will look a bit as the notion of *similarity.*

A second aspect of these problems is related to the way we make decisions: counting does not give rise to estimators that are expressive enough, so we will move to more expressive models, and start talking about classifications.

2 Introduction

In the context of NLP, the word statistics sometimes stands more generally for probability theory, information theory, learning theory and statistics in the sense of statistical inference.

Statistics is really about "how to capture regularities in observed data" and also about "what can be said in cases of sparse data".

```
Th_r_    _s   _nly   _n_   w_y   t_   f_ll   _n Th_   v_w_ls   _n
th_s   s_nt_nc_
```

At a basic level most of low level decisions in NLP are of the form

Is A more likely than B?

```
Illinois bored of education                  {bored; board}



...Nissan Car and truck plant is  divide life into plant and
animal kingdom
                                               {industry; life}


 (This  Art) (can N) (will MD) (rust V)               V,N,N



The dog bit the kid. He was taken to a veterinarian
                                         a hospital
                                                     {dog, kid}

Tiger was in Washington for the PGA Tour
                                         {Person, Location, Animal}
```

How do we make these decisions?

In the statistical approach, the notion is that we want to figure out which of A or B occurs more often.

This is really just a shallow description of what is really going on. The real decision is done via Bayesian decision theory:

- We assume a family of models, one of which governs the generation of the data.
- We observe data
- We estimate the probability of the specific model that generates the data. This is done by estimating the *most likely parameters* given the observed data.
- We use the chosen model to estimate the likelihood of the newly observed data. If there is a need to choose between two hypotheses, we choose the once that is more likely given the model we have.

Today we will be concerned with the problem of *parameter estimation.* This is the *counting* problem.

Of course, there is also the issue of **What to count?** which, based on the above description is really the problem of *what is the assumed model*?

The question of **What to count?** has two parts. What pieces of information to count? What structures to count?

By *pieces of information* we mean that you can count in terms of

- words
- pos tags (paper presented today)
- relations in the sentence (to be presented later)

There are also other subtle issues here: counting words as they appear in the text *wordform*? or counting the *lemma*? What about capitalization?

This also has something to do with **generalization**. You want to make sure that you count something that will allow you to make decisions that are relevant for your specific case.

You can count whether *bored* occurs more often than *board.*

You can also count whether

```
Illinois bored of education
```

occurs more often than

```
Illinois board of education
```

And, a few things in between.

Which is better?

In some intuitive way, we feel that the larger the structure you can count, the more informative it is. However, it is harder to can good counts for larger structures. So there is some tradeoff between the *what* and the *how.*

This tradeoff exists not only with respect to the size of the structure but also it's *quality.*

3 Parameter Estimation: Ngram Models

The Ngram models is essentially an attempt to count structures (estimate the probability of a structure), by counting its substructure. N stands for the size of the substructure.

The process involves two stages: (1) a decomposition stage and (2) an estimation stage.

The *decomposition stage* really involves an independence assumption on the underlying probabilistic model, but this understanding came later. The estimation stage involves another assumption and, typically, a hack (smoothing).

We would like to estimate the probability of a complete string

$$w_1^n = (w_1 w_2 .... w_n)$$

We use the chain rule

$$\begin{aligned} P(w_1 w_2 \dots w_n) &= P(w_n | w_1 w_2 \dots w_{n-1}) P(w_1 \dots w_{n-1}) \\ &= P(w_n | w_1 w_2 \dots w_{n-1}) P(w_{n-1} | w_1 \dots w_{n-2}) P(w_1 \dots w_{n-2}) \\ &= P(w_1) P(w_2 | w_1) P(w_3 | w_1 w_2) \dots P(w_{n-1} | h_{n-1}) P(w_n | h_n) \end{aligned}$$

How to compute the probability of a word w given its history? Markov assumptions. Independence given (far enough) past.

- Unigram:
$$P(w_i | history) = p(w_i)$$
- Bigram:
$$P(w_i | history) = p(w_i | w_{i-1})$$
- trigram
$$P(w_i | history) = p(w_i | w_{i-1} w_{i-2})$$
- quadgram ....

What is the model we assumed? (Specify completely).

Consequence: How do we estimate these?

We take a *training corpus.* For a given bigram (say), we count the number of times it occurs, and divide by the number of all bigrams with the same history (same first word).

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{\sum_w C(w_{i-1}w)}.$$

Notice that: $\sum_w C(w_{i-1}w) = C(w_{i-1})$, which gives:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}.$$

And, in the general case of Ngrams:

$$P(w_i|w_{i-1}\dots w_{i-N+1}) = \frac{C(w_{i-N+1}\dots w_{i-1}w_i)}{C(w_{i-N+1}\dots w_{i-1})} = \frac{C(w_{i-N+1}^{i})}{C(w_{i-N+1}^{i-1})}.$$

3.1 Estimation

Parameter estimation is always done as a maximum likelihood estimate of a Bernoulli distribution.

Assume n independent trials, each with success probability p. The maximum likelihood estimate of p is

$$\frac{k}{m}$$

where k is the number of observed successes.

There are two issues:

- The sensitivity of the estimate to the corpus.
  Generalization issue; related to the type of structures that we count.
- Robustness of the estimate

  The estimate above seems unreasonable when n is small. For example if you flip the coin only once and get a tail should you believe that it is impossible to get a head on the next toss?

4 Smoothing

A key issue in parameter estimation is the sparseness of the data; clearly, this become more severe as the structures one wants to count become more *interesting.* Either larger, or just more discriminative.

An extreme case is that of *zero counts.* We have never seen the word *of* after the word *peace.* Does it mean it can never happen?

We have never seen the word *can* as a *noun* (almost true in the Penn treebank). Does it mean it can never happen?

Smoothing techniques are supposed to "correct" observed counts and replace them by better, more representative counts.

4.1 Add-One Smoothing

Example 4.1 *We want to estimate* $p(a|bc)$

Our training data includes

$$...bcX....bcY......bcY....$$

but never

$$bcZ$$

Should we conclude that

$$p(X|bc) = 1/3; p(Y|bc) = 2/3?$$

NO! Absence of cbZ might just be bad luck.

We want to **discount** *the positive counts somewhat and*

reallocate *that probability to the zeroes.*

This is especially true if the denominator is small (small sample) and the numerator is small (not enough observations).

	String	#observed	estimate	+1-numerator	new estimate
	bcA	1	1/3	2	2/29
	bcB	0	0/3	1	1/29
	bcC	0	0/3	1	1/29
	bcD	2	2/3	3	3/29
	bcE	0	0/3	1	1/29
....					
	bcZ	0	0/3	1	1/29
Total	bc	3	3/3	29	29/29

By adding 1 *to the numerator we are forced to add to the denominators* V *- the size of the vocabulary, to maintain that we have a distribution over the character that follows bc.*

Recall that by *type* we mean a *distinct vocabulary item* and by **token**, the *occurrence of that type.* For example, a dictionary is a list of types (once each), and a corpus is a list of tokens (each type has many tokens).

Let V be the size of the **Vocabulary**: the total number of *types* possible (not the number of types in the corpus). The smoothed bigram estimate is now:

$$P^*(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + V}.$$

This is called Laplace Estimate and, in general, V here is going to stand for the number of possible types of *structures* that we care about in the denominator.

Example 4.2 (continue) *Notice that when you see more observations, say,* 300 *instead of* 3*, less smoothing is required, and this method gives it automatically.*

	String	#observed	estimate	+1-numerator	new estimate
	bcA	100	1/300	101	101/326
	bcB	0	0/300	1	1/326
	bcC	0	0/300	1	1/326
	bcD	200	2/300	201	201/326
	bcE	0	0/300	1	1/326
....					
	bcZ	0	0/300	1	1/326
Total	bc	300	300/300	326	326/326

4.1.1 Some problems

- Suppose the vocabulary size is 20000 rather than 300.

  As we get more and more training data, we see more and more words that need probability the probabilities of existing words keep dropping, instead of converging. This cant be right eventually they drop too low.

  Maybe instead of adding 1 to all counts, add $alpha = 0.01$, to give smaller probability to new events. But how to pick best value for α?

- Are all 0s the same?

  Is it possible to distinguish between events that do not occur in a finite sample due to the small sample and those that do not occur due to inherent constraints?

down the road	100/20000
down the harbor	50/20000
down the report	1/20000
down the grapevine	0/20000
....	
down the write	0/20000

4.1.2 Some justification

This seems like a hack; we no longer do a maximum likelihood estimate.

Let us consider that case of a coin flip. You observe n coins tosses and k of them are `head`.

Instead, we can assume that we have a uniform prior distribution on p, the probability that the toss is `head`. That is, the prior distribution on all the possible coin biases is uniform; the density function is:

$$P[p \leq \theta] = \int_0^\theta dp = \theta.$$

The meaning of this prior is that:

$$\int_0^\theta P[k]dp = \int_0^\theta P[k+1]dp \tag{1}$$

where:

$$\int_0^\theta P[k]dp = \int_0^\theta P[k|p]P[p]dp$$

Eq. 1 can be proved by computing the integral. That means, that that posterior probability:

$$\int_0^\theta P[k|p]P[p]dp = \frac{1}{n+1}$$

From this it can be shown that the expected value of p when we observe a sequence of n coin tosses and k heads:

$$E[P|(k,n)] = \frac{k+1}{n+2}.$$

4.2 Good-Turing Smoothing

As we have seen, the goal of smoothing is to hold out some probability mass for novel events. Different methods differ by how do they split the amount of probability mass among novel events.

The key idea behind this method, attributed to Turing, but first described by Good, is to re-estimate the amount of the probability mass to assign to Ngram with zero or low counts, by looking at the number of Ngrams of higher counts.

Specifically, judge rate of 0-count events by rate of 1-count events.

Let $n_r =$ **# of ngram types with r training tokens**

(Called: the frequency of frequency r.)

n_0 - the number of bigrams of count 0;

n_1 - the number of bigrams of count 1, etc.

Therefore, the total number of training tokens is:

$$N = \sum_r r n_r.$$

Naive estimate: if x is an ngram that has r tokens, then $p(x) =$?

$$p(x) = r/N.$$

Total naive probability of all words with r tokens?

$$n_r \frac{r}{N}.$$

Good-Turing estimate of this total probability is defined as:

$$n_{r+1} \frac{r+1}{N},$$

so the proportion of novel words in the test data is estimated by proportion of singletons in training data. Likewise, the proportion in test data of the n_1 singletons is estimated by proportion of the n_2 doubletons in training data.

What is Good-Turing estimate of $p(x)$?

Good Turing estimate: if x is an ngram that has r tokens, then $p(x) = ?$

$$p(x) = \frac{n_{r+1}}{n_r}\frac{r+1}{N}$$

Another way of thinking about it:

Good-Turing states that an n-gram that occurs r times should be treated as if it had occurred r^* times, where

$$r^* = (r+1)\frac{n_{r+1}}{n_r}$$

where n_r is the number of n-grams that occur exactly r times in the training data.

This immediately give the probability estimate given above for $p(x)$.

For example, the revised count of bigrams that never occurred is:

$$n_0^* = n_1/n_0,$$

the ratio between the number of bigrams that occurred once to those that never occurred.

What is n_0?:

Let V be the size of the vocabulary. The total number of bigrams is V^2. Then

$$n_0 = V^2 - \#(\text{Observed bigrams}).$$

In fact, Good-Turing is more general than this procedure. It is suggested to estimate

$$p(x) = \frac{r^*}{N}$$

where r^* is the new estimate for the true number of occurrences of an ngram that actually occurred r times.

Clearly, we need

$$r^* < r, \quad \text{when} \quad r > 0$$

since we must reduce the total probability of the objects which were seen, to free up space for those that were not seen at all.

The general form of a Good-Turing estimate is:

$$r* = (r+1)\frac{E(n_r)}{E(n_{r+1})},$$

where $E(x)$ represents expectation of the random variable x.

In this way, the total probability of the unseen objects is precisely stated as

$$\frac{E(n_1)}{N}.$$

n_1 is typically the largest and best measured of the n_r. Thus it is not unreasonable to substitute n_1 for $E(n_1)$.

When the substitution of n_r for $E(n_r)$ is made, we term the estimator the *Turing estimator*. However, the larger r is, the less reasonable this substitution is.

Good (1953) suggested that replace the observed n_r with *smoothed values*, $S(n_r)$.

When this approach is taken the estimator is called a Good-Turing estimator.

Thus, there can be many Good-Turing estimators depending on what smoothing process is used. Unfortunately, the uncertainties in the n_r vary greatly with r, and smoothing such data is difficult.

Good-Turing can also be justified, under the assumptions that the distribution of each bigram is binomial. It's also possible to study the rate of convergence of the estimate.

There are other methods that use similar ideas to Good-Turing, and other variations of Add-One.

$r(ML)$	n_r	r^* (GT)
0	74,671,100,000	0.000027
1	2,018,046	0.446
2	449,721	1.26
3	188,933	2.24
4	105,668	3.24
5	68,379	4.22
6	48,190	5.19
7	35,709	6.21
8	27,710	7.24
9	22,280	8.25

Table 1: Bigram "Frequencies of frequencies" from 22 million AP bigrams, and Good-Turing re-estimates, after Church and Gale (1991)

4.3 Backoff

Notice that while both methods are better than allowing zero estimates, there is still is a problem.

The methods do not distinguish between non-zero events. E.g, "observe that" and "observe thou" will have the same estimate.

Like previous smoothing methods we looked at, the idea is to hold out same amount of probability mass for novel events. The key difference is that now we will divide up this amount unevenly. It will be done in proportion to backoff prob that we will now discuss.

A hack, that has been used and is a special case of what we will discuss later is to take the unigram estimate into account, and interpolate.

$$P_{interp}(w_i|w_{i-1}) = \lambda P_{ML}(w_i|w_{i-1}) + (1-\lambda)P_{ML}(w_i).$$

How to estimate λ ?

The idea in a Backoff model is to build an Ngram model based on an (N-1)gram model. If we have zero counts for Ngrams, we *Back off* to (N-1)grams. If there is a non-zero count there, we stop and rely on it. You can also imagine models that will keep on going and will interpolate lower orders.

The recursive definition of the backoff model is therefore:

$$P_{bo}(w_i|w_{i-N+1}^{i-1}) = \hat{P}(w_i|w_{i-N+1}^{i-1}) + \theta(P(w_i|w_{i-N+1}^{i-1}) \cdot \alpha \cdot P_{bo}(w_i|w_{i-N+2}^{i-1}),$$

where

$$\theta \;\; = \;\; \begin{cases} 1, & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}.$$

We will have to say something about the α and the $\hat{P}$.

Specifically, for the case of trigrams:

$$P_{bo}(w_i|w_{i-2}w_{i-1}) \;\; = \;\; \begin{cases} \hat{P}(w_i|w_{i-2}w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha_1\hat{P}(w_i|w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) = 0 \text{ and } C(w_{i-1}w_i) > 0 \\ \alpha_2\hat{P}(w_i), & \text{otherwise} \end{cases}.$$

Why do we need the α and $\hat{P}$?

The original Ngram model, computed using the frequencies, is a probability distribution. That is:

$$\sum_{i,j} P(w_n|w_iw_j) = 1$$

when the sum is over all contexts.

Therefore, when we back off to a lower order model when the count is zero we are adding extra probability mass and the total probability of a word will be greater than 1. Back off probabilities thus need to be *discounted.*

$\hat{P}$ - discount the ML estimate to save some probability mass for lower order Ngrams.

αs - used to ensure that the probability mass from all the lower order Ngrams sums up to the amount saved.

There are normalization parameters, and computing them is pretty messy. We will have:

$$\hat{P}(w_i|w_{i-N+1}^{i-1}) == \frac{C^*(w_{i-N+1}^{i})}{C(w_{i-N+1}^{i-1})},$$

where $C^*(w_{i-N+1}^{i}) < C(w_{i-N+1}^{i-1})$.

This will allow us to leave some probability mass for the lower order Ngrams, which will be done using the αs.

The total left over probability mass for a given (N-1)gram context is:

$$\beta(w_{i-N+1}^{i-1}) = 1 - \sum_{w_i:C(w_{i-N+1}^{i})>0} \hat{P}(w_i|w_{i-N+1}^{i-1}).$$

This mass need to be distributed to all the (N-1)gram. Each individual (N-1)gram will get a fraction of this mass.

$$\alpha(w_{i-N+1}^{i-1}) = \frac{\beta(w_{i-N+1}^{i-1})}{\sum_{w_i:c(w_{i-N+1}^{i})>0} \hat{P}(w_i|w_{i-N+2}^{i-1})}$$

This needs to continue, since now we need to estimate again the probabilities $\hat{P}$ with the discount and split the remaining mass to the lower order Ngrams again.

The important thing to notice is that the αs are a function of the preceding string. That is, the amount by which we discount each Ngram and how we split this mass depends on the (N-1)grams that are the substrings of the Ngram.

Notice that the estimate itself, C^*, can be computed in several ways, using any smoothing techniques you want.

4.3.1 Deleted Interpolation

Instead of just *backing off* to the non-zero Ngram, it is possible to take into account all Ngrams. That is, estimate (in the case of a trigram):

$$\hat{P}(w_i|w_{i-2}w_{i-1}) = \lambda_1 P(w_i|w_{i-2}w_{i-1}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i),$$

where

$$\sum_i \lambda_i = 1.$$

In principle, the λ are made context dependent: $\lambda_i = \lambda_i(w_{i-2}^i)$.

Training the λs can be done so as to maximize the likelihood of a *held-out* data, separate from the main training corpus. That is, for each set of λs, one can compute the likelihood of the given data. Find this set of λs that gives the highest value. This is conceptually a search procedure, that can be done more efficiently using a version of the EM algorithm, which we will discuss later in the course.

4.3.2 Application

The key issue to notice when you think in a concrete way about Ngram in an application is that the model is actually quite large. There are *many* bigrams, trigrams, etc. to consider.

Spelling:

Given a sentence

$$W = \{w_1 w_2 \dots w_n\}$$

where the word w_k has, say, two alternatives, $w_{k'}, w_{k''}$, we choose the spelling that maximize $P(W)$, where $P(W)$ is computed via an Ngram model.

What happens if you use Trigrams:

$$\begin{aligned} P(w_1 w_2 \dots w_n) &= P(w_n|w_1 w_2 \dots w_{n-1}) P(w_1 \dots w_{n-1}) \\ &= P(w_n|w_1 w_2 \dots w_{n-1}) P(w_{n-1}|w_1 \dots w_{n-2}) P(w_1 \dots w_{n-2}) \\ &= P(w_1) P(w_2|w_1) P(w_3|w_1 w_2) \dots P(w_{n-1}|h_{n-1}) P(w_n|h_n) \end{aligned}$$

You end up using only *those three trigrams* that contain the word.

$$P(w'|w_{i-2}w_{i-1})P(w_{i+1}|w'w_{i-1})P(w_{i+2}|w_{i+1}w') \text{ vs. } P(w''|w_{i-2}w_{i-1})P(w_{i+1}|w''w_{i-1})P(w_{i+2}|w_{i+1}w'')$$

You can take log: $\tilde{P} \equiv logP$

$$\tilde{P}(w'|w_{i-2}w_{i-1}) + \tilde{P}(w_{i+1}|w'w_{i-1}) + \tilde{P}(w_{i+2}|w_{i+1}w')$$

vs.

$$\tilde{P}(w''|w_{i-2}w_{i-1}) + \tilde{P}(w_{i+1}|w''w_{i-1}) + \tilde{P}(w_{i+2}|w_{i+1}w'')$$

And remember that it is possible that each of these numbers you compute as a function of lower degree events.

Remember both the functional form, and the fact that you need to remember a lot of numbers...

There are two directions to go from here; one is to try to generalize; why use words? This is part of the *What* question.

A second is to try to figure out a better functional form for the discriminating function.