Linear Models for Statistical Natural Language Processing

Jacob Eisenstein

November 10, 2014

Chapter 1

Introduction

This is a collection of notes that I use for teaching Georgia Tech Computer Science 4650 and 7650, "Natural Language." The notes focus on what I view as a core subset of the field of natural language processing, unified by the concept of linear models. This includes approaches to document classification, word sense disambiguation, sequence labeling (part-of-speech tagging and named entity recognition), parsing, coreference resolution, relation extraction, discourse analysis, and, to a limited degree, language modeling and machine translation. The theme was inspired by Fernando Pereira's EMNLP 2008 keynote, "Are linear models right for language."[1] The notes are heavily influenced by several other good resources (e.g., Manning and Schütze, 1999; Jurafsky and Martin, 2009; Figueiredo et al., 2013; Collins, 2013), but for various reasons I wanted to create something of my own.

[1] You can see a version of this talk — not the one I saw — online at `vimeo.com/30676245`

Chapter 2

Notation

w_n	word token at position n
$\boldsymbol{x}_i$	a vector of feature counts for instance i, often word counts
N	number of training instances
V	number of words in vocabulary
$\boldsymbol{\theta}$	a vector of weights
y_i	the label for instance i
$\boldsymbol{y}$	vector of labels across all instances
$\mathcal{Y}$	set of all possible labels
K	number of possible labels $K = \#\lvert\mathcal{Y}\rvert$
$\boldsymbol{f}(\boldsymbol{x}_i, y_i)$	feature vector for instance i with label y_i
$P(A)$	probability function of event A
$\mathrm{p}_B(b)$	the marginal probability of random variable B taking value b
M	length of a sequence (of words or tags)
$\mathcal{T}(\boldsymbol{w})$	the set of possible tag sequences for the word sequence $\boldsymbol{w}$
$\diamond$	the start symbol
$\square$	the stop symbol
λ	the amount of regularization

Chapter 3

Linear classification and features

Suppose you want to build a spam detector. Spam vs. Ham. How would you do it, using only the text in the email?

One solution is to represent document i as a column vector of word counts: $\boldsymbol{x}_i = [0\ 1\ 1\ 0\ 0\ 2\ 0\ 1\ 13\ 0 \ldots]^\mathsf{T}$, where $x_{i,j}$ is the count of word j in document i. Suppose the size of the vocabulary is V, so that the length of $\boldsymbol{x}_i$ is also V.

We've thrown out grammar, sentence boundaries, paragraphs — everything but the words! But this could still work. If you see the word *free*, is it spam or ham? How about *c1al1s*? How about *Bayesian*? One approach would be to define a "spamminess" score for every word in the dictionary, and then just add them up. This is also a commonly-used approach to sentiment analysis, where each word is scored as one of $\{1, 0, -1\}$, with 1 indicating positive sentiment and -1 indicating negative sentiment.

These scores are called **weights**, written $\boldsymbol{\theta}$, and we'll spend a lot of time later talking about where they come from. But for now, let's generalize: suppose we want to build a multi-way classifier to distinguish stories about sports, celebrities, music, and business. Each label is an element y_i in a set of K possible labels $\mathcal{Y}$. Then for any pair $\langle \boldsymbol{x}_i, y_i \rangle$, we can define a *feature vector* $\boldsymbol{f}(\boldsymbol{x}_i, y_i)$, such that:

$$\boldsymbol{f}(\boldsymbol{x}, y = 0) = [\boldsymbol{x}_i^\mathsf{T}\ \mathbf{0}_{V(K-1)}^\mathsf{T}]^\mathsf{T} \tag{3.1}$$

$$\boldsymbol{f}(\boldsymbol{x}, y = 1) = [\mathbf{0}_V^\mathsf{T}\ \boldsymbol{x}_i^\mathsf{T}\ \mathbf{0}_{V(K-2)}^\mathsf{T}]^\mathsf{T} \tag{3.2}$$

$$\boldsymbol{f}(\boldsymbol{x}, y = 2) = [\mathbf{0}_{2V}^\mathsf{T}\ \boldsymbol{x}_i^\mathsf{T}\ \mathbf{0}_{V(K-3)}^\mathsf{T}]^\mathsf{T} \tag{3.3}$$

$$\ldots \tag{3.4}$$

$$\boldsymbol{f}(\boldsymbol{x}, K) = [\mathbf{0}_{V(K-1)}^\mathsf{T}\ \boldsymbol{x}_i^\mathsf{T}]^\mathsf{T}, \tag{3.5}$$

where $\mathbf{0}_{VK}$ is a column vector of VK zeros. Often we'll add an **offset** feature at

the end of $\boldsymbol{x}$, which is always 1; we then have to also add an extra zero to each of the zero vectors. This gives the entire feature vector $\boldsymbol{f}(\boldsymbol{x}, y)$ a length of $(V+1)K$.

Now, given a vector of weights, $\boldsymbol{\theta} \in \mathcal{R}^{(V+1)K}$, we can compute the inner product $\boldsymbol{\theta}^\top \boldsymbol{f}(x, y)$. Then for any document $\boldsymbol{x}_i$, we can predict a label $\hat{y}$ as

$$\hat{y} = \arg\max_y \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y) \tag{3.6}$$

We could just set the weights by hand. If we wanted to distinguish, say, English from Spanish, we could just use English and Spanish dictionaries, and set each weight to 1. For example,

$$\begin{aligned} \theta_{\text{english,bicycle}} &= 1 & \theta_{\text{spanish,bicycle}} &= 0 \\ \theta_{\text{english,bicicleta}} &= 0 & \theta_{\text{spanish,bicicleta}} &= 1 \\ \theta_{\text{english,con}} &= 1 & \theta_{\text{spanish,con}} &= 1 \\ \theta_{\text{english,ordinateur}} &= 0 & \theta_{\text{spanish,ordinateur}} &= 0 \end{aligned}$$

Similarly, if we want to distinguish positive and negative sentiment, we could use positive and negative *sentiment lexicons*, which are defined by expert psychologists (Tausczik and Pennebaker, 2010). You'll try this in Project 1.

But it's usually not easy to set the weights by hand. Instead, we will learn them from data. For example, suppose that an email user has manually labeled thousands of messages as "spam" or "not spam"; or a newspaper may label its own articles as "business" or "fashion." Such **instance labels** are a typical form of labeled data that we will encounter in NLP. In **supervised machine learning**, we use instance labels to automatically set the weights for a classifier. An important tool for this is probability.

3.1 Review of basic probability

This section is inspired/borrowed from Manning and Schütze (1999).

- **Formally**: When we write $P(\dot{})$, this denotes a function $P : \mathcal{F} \rightarrow [0, 1]$ from an **event space** $\mathcal{F}$ to a **probability**. A probability is a real number between zero and one, with zero representing impossibility and one representing certainty.
- The probabilities of disjoint event sets are additive: $A_i \cap A_j = \varnothing \Rightarrow P(A_i \cup A_j) = P(A_i) + P(A_j)$. This is a restatement of the Third Axiom of probability.

- For example, you might ask what is the probability of two heads on three coin flips. There are eight possible series of three flips $HHH, HHT, \ldots$, and each is an equally likely event. Of these events, three meet the criterion, HHT, HTH, THH. So the probability is $\frac{3}{8}$.
- More generally, $P(A_i \cup A_j) = P(A_i) + P(A_j) - P(A_i \cap A_j)$. This can be derived from the third axiom.

$$\begin{aligned} P(A_i \cup A_j) &= P(A_i) + P(A_j - (A_i \cap A_j)) && (3.7)\\ P(A_j) &= P(A_j - (A_i \cap A_j)) + P(A_i \cap A_j) && (3.8)\\ P(A_j - (A_i \cap A_j)) &= P(A_j) - P(A_i \cap A_j) && (3.9)\\ P(A_i \cup A_j) &= P(A_i) + P(A_j) - P(A_i \cap A_j) && (3.10) \end{aligned}$$

- If the probability $P(A \cap B) = P(A)P(B)$, then the events A and B are *independent*, written $A \perp B$.

Conditional probability and Bayes' Rule

A conditional probability is an expression like $P(A \mid B)$, where we are interested in the probability of A conditioned on B happening.

- Conditional probability: $P(A \mid B) = P(A \cap B)/P(B)$
- If $P(A \cap B \mid C) = P(A \mid C)P(B \mid C)$, then the events A and B are **conditionally independent**, written $A \perp B \mid C$.
- Chain rule: $P(A \cap B) = P(A \mid B)P(B)$, which is just a rearrangement of terms.
- We can apply the chain rule multiple times:

$$\begin{aligned} P(A \cap B \cap C) &= P(A \mid B \cap C)P(B \cap C) \\ &= P(A \mid B \cap C)P(B \mid C)P(C) \end{aligned}$$

  We'll do this a lot later in the course.

- Bayes' rule follows from the Chain rule: $P(A \mid B) = P(A \cap B)/P(B) = P(B \mid A)P(A)/P(B)$

Often we want the maximum a posteriori (MAP) estimate

$$\begin{aligned}\hat{B} &= \arg\max_B P(B \mid A) \\ &= \arg\max_B P(A \mid B)P(B)/P(A) \\ &\propto \arg\max_B P(A \mid B)P(B)\end{aligned}$$

- We don't need to normalize the probability because $P(A)$ is the same for all values of B.
- If we do need to compute the conditional $P(A \mid B)$, we can compute $P(A)$ by summing over $P(A \cap B) + P(A \cap \overline{B})$, where $B \cap \overline{B} = \varnothing$ and $B \cup \overline{B} = \Omega$, the entire sample space (such that $P(\Omega) = 1$).
- More generally, if $\bigcup_i B_i = \Omega$ and $\forall_{i,j}, B_i \cap B_j = \varnothing$, then $P(A) = \sum_i P(A \mid B_i)P(B_i)$.

Example Manning and Schütze (1999) have a nice example of Bayes Rule (Bayes Law) in a linguistic setting.

- Suppose one is interested in a rare syntactic construction, perhaps parasitic gaps, which occurs on average once in 100,000 sentences.
  - (An example of a sentence with a parasitic gap is *Which class did you attend __ without registering for __?* -JE)
- Lana Linguist has developed a complicated pattern matcher that attempts to identify sentences with parasitic gaps. Its pretty good, but it's not perfect:
  - If a sentence has a parasitic gap, it will say so with probability 0.95 (this is the **recall** -JE).
  - If it doesn't, it will wrongly say it does with probability 0.005 (this is the **false positive rate**, the additive inverse of **precision** -JE).
- Suppose the test says that a sentence contains a parasitic gap. What is the probability that this is true?
- (This example is usually framed in terms of tests for rare diseases. -JE)

Solution: Let G be the event of a sentence having a parasitic gap, and T be the event of the test being positive.

$$P(G \mid T) = \frac{P(G \mid T)P(T)}{P(G \mid T)P(T) + P(G \mid \overline{T})P(\overline{T})} \tag{3.11}$$

$$= \frac{0.95 \times 0.00001}{0.95 \times 0.00001 + 0.005 \times 0.99999} \approx 0.002 \tag{3.12}$$

Random variables

A random variable takes on a specific value in $\mathbb{R}^n$, typically with $n = 1$, but not always. Discrete random variables can take values only in some countable subset of $\mathbb{R}$.

- Recall the coin flip example. The number of heads, H, can be viewed as a discrete random variable, $H \in 0, 1, 2, 3$.
- The probability mass associated with each number is $\{\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}\}$.
- This set of numbers represents the **probability distribution** over H, written $P(H = h) = \mathrm{p}(h)$.
- To indicate that the RV H is distributed as $\mathrm{p}(h)$, we write $H \sim \mathrm{p}(h)$.
- The function $\mathrm{p}(h)$ is called a probability **mass** function (pmf) if h is discrete, and a probability **density** function (pdf) if h is continuous.
- If we have more than one variable, we can write a joint probability $\mathrm{p}(a, b) = P(A = a, B = b)$.
- We can write a **marginal** probability $\mathrm{p}_A(a) = \sum_b \mathrm{p}(a, b)$.
- Random variables are independent iff $\mathrm{p}_{A,B}(a, b) = \mathrm{p}_A(a)\mathrm{p}_B(b)$.
- We can write a conditional probability as $\mathrm{p}(a \mid b) = \frac{\mathrm{p}(a,b)}{\mathrm{p}_B(b)}$.

Expectations

Sometimes we want the **expectation** of a function, such as $E[g(x)] = \sum_{x \in \mathcal{X}} g(x)\text{p}(x)$.

Expectations are easiest to think about in terms of probability distributions over discrete events:

- If it is sunny, Marcia will eat three ice creams.
- If it is rainy, she will eat only one ice cream.
- There's a 80% chance it will be sunny.
- The expected number of ice creams she will eat is $0.8 \times 3 + 0.2 \times 1 = 2.6$.

If the random variable X is continuous, the sum becomes an integral:

$$E[g(x)] = \int_{\mathcal{X}} g(x)\text{p}(x)dx \tag{3.13}$$

For example, a fast food restaurant in Quebec gives a 1% discount on french fries for every degree below zero. Assuming they used a thermometer with infinite precision, the expected price would be an integral over all possible temperatures.

3.2 Naïve Bayes

Back to classification! A Naïve Bayes classifier chooses the weights $\boldsymbol{\theta}$ to maximize the *joint* probability of a labeled dataset, $\text{p}(\boldsymbol{x}_{1:N}, \boldsymbol{y}_{1:N})$, where $\langle \boldsymbol{x}_i, y_i \rangle$ is a labeled instance.

We first need to define the probability $\text{p}(\boldsymbol{x}, y)$. We'll do that through a "generative model," which describes a hypothesized stochastic process that has generated the observed data.[1]

- For each document i,
  - draw the label $y_i \sim \text{Categorical}(\mu)$
  - draw the vector of counts $\boldsymbol{x}_i \sim \text{Multinomial}(\phi_{y_i})$,

[1] We'll see a lot of different generative models in this course. They are a helpful tool because they clearly and explicitly define the assumptions that underly the form of the probability distribution.

The first thing this generative model tells us is that we can treat each document independently: the probability of the whole dataset is equal to the product of the probabilities of each individual document. The observed word counts and document labels are independent and identically distributed (IID).

$$\mathrm{p}(\boldsymbol{x}, \boldsymbol{y}; \mu, \phi) = \prod_i \mathrm{p}(\boldsymbol{x}_i, y_i; \mu, \phi) \tag{3.14}$$

This means that the words in each document are **conditionally independent** given the parameters μ and ϕ.

When we write $y_i \sim \text{Categorical}(\mu)$, that means y_i is a stochastic draw from a categorical distribution with **parameter** μ. A categorical distribution is just like a weighted die: $\mathrm{p}_{\text{cat}}(y; \mu) = \mu_y$, where μ_y is the probability of the outcome $Y = y$. We require $\sum_y \mu_y = 1$ and $\forall_y, \mu_y \geq 0$.

A multinomial distribution is only slightly more complex:

$$\mathrm{p}_{\text{mult}}(\boldsymbol{x}; \phi) = \frac{\left(\sum_j x_j\right)!}{\prod_j x_j!} \prod_j \phi_j^{x_j} \tag{3.15}$$

We again require that $\sum_j \phi_j = 1$ and $\forall j, \phi_j \geq 0$. The first part of the equation doesn't depend on ϕ, and can usually be ignored. Can you see why we need the first part at all?[2]

We can write $\mathrm{p}(\boldsymbol{x}_i \mid y_i; \phi)$ to indicate the conditional probability of word counts $\boldsymbol{x}_i$ given label y_i, with parameter ϕ, which is equal to $\mathrm{p}_{\text{mult}}(\boldsymbol{x}_i; \phi_{y_i})$.

By specifying the multinomial distribution, we are working with *multinomial naïve Bayes* (MNB). Why "naïve"? Because the multinomial distribution treats each word token independently: the probability mass function factorizes across the counts.[3] We'll see this more clearly later, when we show how MNB is an example of linear classification.

Another version of Naïve Bayes

Consider a slight modification to the generative story of NB:

[2]Technically, a multinomial distribution requires a second parameter, the total number of counts (the number of words in the document). Even more technically, that number should be treated as a random variable, and drawn from some other distribution. But none of that matters for classification.

[3]You can plug in any probability distribution to the generative story and it will still be naïve Bayes, as long as you are making the "naïve" assumption that your features are generated independently.

- For each document i
  - Draw the label $y_i \sim \text{Categorical}(\mu)$
  - For each word $n \leq D_i$
    - Draw the word $w_{i,n} \sim \text{Categorical}(\phi_{y_i})$

This is not quite the same model as multinomial Naive Bayes (MNB): it's a product of categorical distributions over words, instead of a multinomial distribution over word counts. This means we would generate the words in order, like $\text{p}_W(\textit{multinomial})\text{p}_W(\textit{Naive})\text{p}_W(\textit{Bayes})$. Formally, this is a model for the joint probability $\text{p}(\boldsymbol{w}, y)$, not $\text{p}(\boldsymbol{x}, y)$.

However, as a classifier, it is identical to MNB. The final probabilities are reduced by a factor corresponding to the normalization term in the multinomial, $\frac{(\sum_j x_j)!}{\prod_j x_j!}$. This means that the resulting probabilities for a given $\boldsymbol{x}$ are different. However, none of this has anything to do with the label y or the parameters $\boldsymbol{\phi}$. The ratio of probabilities between any two labels y_1 and y_2 will be identical, as will the maximum likelihood estimates for the parameters μ and $\boldsymbol{\phi}$ (defined later).

Prediction

The Naive Bayes prediction rule is to choose the label y which maximizes $\text{p}(\boldsymbol{x}, y; \phi, \mu)$:

$$\begin{aligned}
\hat{y} =& \arg\max_y \text{p}(\boldsymbol{x}, y; \mu, \phi) \\
=& \arg\max_y \text{p}(\boldsymbol{x} \mid y; \phi)\text{p}(y; \mu) \\
=& \arg\max_y \log \text{p}(\boldsymbol{x} \mid y; \phi) + \log \text{p}(y; \mu)
\end{aligned}$$

Converting to logarithms makes the notation easier. It doesn't change the prediction rule because the log function is monotonically increasing.

Now we can plug in the probability distributions from the generative story.

$$\begin{aligned}
\log \mathrm{p}(\boldsymbol{x}, y; \mu, \phi) =& \arg\max_y \log \mathrm{p}(\boldsymbol{x} \mid y; \phi) + \log \mathrm{p}(y; \mu) \\
=& \log \left[\frac{\left(\sum_j x_j\right)!}{\prod_j x_j!} \prod_j \phi_{y,j}^{x_j} \right] + \log \mu_y \\
=& \log \frac{\left(\sum_j x_j\right)!}{\prod_j x_j!} + \sum_j x_j \log \phi_{y,j} + \log \mu_y \\
\propto& \sum_j x_j \log \phi_{y,j} + \log \mu_y \\
=& \boldsymbol{\theta}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}, y),
\end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{\theta} &= [\boldsymbol{\theta}^{(1)\mathsf{T}}, \boldsymbol{\theta}^{(2)\mathsf{T}}, \ldots, \boldsymbol{\theta}^{(K)\mathsf{T}}]^\mathsf{T} \\
\boldsymbol{\theta}^{(y)} &= [\log \phi_{y,1} \;\; \log \phi_{y,2} \;\; \ldots \;\; \log \phi_{y,M} \log \mu_y]^\mathsf{T}
\end{aligned}$$

and $\boldsymbol{f}(\boldsymbol{x}, y)$ is a vector of word counts and an offset, padded by zeros for the labels not equal to y (see equations 3.1-3.5). This ensures that the inner product $\boldsymbol{\theta}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}, y)$ only activates the features in $\boldsymbol{\theta}^{(y)}$, which are what we need to compute the joint log-probability $\log \mathrm{p}(\boldsymbol{x}, y)$ for each y.

Estimation

The parameters of a multinomial distribution have a simple interpretation: they're the expected frequency for each word. Based on this interpretation, it's tempting to set the parameters empirically, as

$$\phi_{y,j} = \frac{\sum_{i:Y_i=y} x_{i,j}}{\sum_{j'} \sum_{i:Y_i=y} x_{i,j'}} = \frac{\text{count}(y, j)}{\sum_{j'} \text{count}(y, j')} \tag{3.16}$$

In NLP this is called a *relative frequency estimator*. It can be justified more rigorously as a *maximum likelihood estimate*.

As in prediction, we want to maximize the joint likelihood of the data,

$$L = \sum_i \log \mathrm{p}_{\text{mult}}(\boldsymbol{x}_i; \boldsymbol{\phi}_{y_i}) + \log \mathrm{p}_{\text{cat}}(y_i; \mu) \tag{3.17}$$

Since $\mathrm{p}(y)$ is unrelated to ϕ, we can forget about it for now. But before we can just optimize L, we have to deal with a constraint:

$$\sum_j \phi_{y,j} = 1 \tag{3.18}$$

We'll do this by adding a Lagrange multiplier. Here's the resulting Lagrangian:

$$\ell[\boldsymbol{\phi}_y] = \sum_{i:Y_i=y} \sum_j x_{ij} \log \phi_{y,j} + \lambda(\sum_j \phi_{y,j} - 1) \tag{3.19}$$

We solve by setting $\frac{\partial \ell}{\boldsymbol{\phi}_j} = 0$.

$$\begin{aligned}
0 &= \sum_{i:Y_i=y} x_{i,j}/\phi_{y,j} - \lambda \\
\lambda\phi_{y,j} &= \sum_{i:Y_i=y} x_{i,j} \\
\phi_{y,j} &\propto \sum_{i:Y_i=y} x_{i,j} = \sum_i \delta(Y_i = y) x_{i,j} \\
&= \frac{\sum_{i:Y_i=y} x_{i,j}}{\sum_{j'} \sum_{i:Y_i=y} x_{i,j'}}
\end{aligned}$$

Similarly, $\mu_y \propto \sum_i \delta(Y_i = y)$, where $\delta(Y_i = y) = 1$ if $Y_i = y$ and 0 otherwise.

Smoothing and MAP estimation

If data is sparse, you can end up with values of $\phi = 0$, allowing a single feature to completely veto a label. This is undesirable, because it imposes high **variance**: depending on what data happens to be in the training set, we could get vastly different classification rules.

One solution is Laplace smoothing: adding "pseudo-counts" of α to each estimate, and then normalize.

$$\phi_{y,j} = \frac{\alpha + \sum_{i:Y_i=y} x_{i,j}}{\sum_{j'} \alpha + \sum_{i:Y_i=y} x_{i,j'}} = \frac{\alpha + \text{count}(i,j)}{V\alpha + \sum_{j'} \text{count}(i,j')} \tag{3.20}$$

Laplace smoothing has a nice Bayesian justification, in which we extend the generative story to include ϕ as a random variable (rather than as a parameter). The resulting estimate is called *maximum a posteriori*, or MAP.

Smoothing reduces **variance**, but it takes us away from the maximum-likelihood estimate: it imposes a **bias** (towards uniform probabilities). Machine learning theory shows that errors on held out data result from the sum of bias and variance. Techniques for reducing variance typically increase the bias, so there is a **bias-variance tradeoff**.

- Unbiased classifiers **overfit** the training data, yielding poor performance on unseen data.
- But if we set a very large smoothing value, we can **underfit** instead. In the limit of $\alpha \rightarrow \infty$, we have zero variance: it is the same classifier no matter what data we see! But the bias of such a classifier will be high.
- Navigating this tradeoff is hard. But in general, as you have more data, variance is less of a problem, so you just go for low bias.

Training, testing, and tuning (development) sets

We'll soon talk about more learning algorithms, but whichever one we apply, we will want to report its accuracy. Really, this is an educated guess about how well the algorithm will do on new data in the future.

To do this, we need to hold out a separate "test set" from the data that we use for estimation (i.e., training, learning). Otherwise, if we measure accuracy on the same data that is used for estimation, we will badly overestimate the accuracy we're likely to get on new data. See `http://xkcd.com/1122/` for a cartoon related to this idea.

Many learning algorithms also have "tuning" parameters:

- the smoothing pseudo-counts α in Naive Bayes
- the regularization λ in logistic regression
- the slack weight C in the support-vector machine

All of these tuning parameters really do the same thing: they navigate the bias-variance tradeoff. Where is the best position on this tradeoff curve? It's hard to tell in advance. Sometimes it is tempting to see which tuning parameter gives the best performance on the test set, and then report that performance. Resist this temptation! It will also lead to overestimating accuracy on truly unseen future

data. For that reason, this is a sure way to get your research paper rejected. Instead, you should split off a piece of your training data, called a "development set" (or "tuning set").

Sometimes, people average across multiple test sets and/or multiple development sets. One way to do this is to divide your data into "folds," and allow each fold to be the development set one time. This is called **K-fold cross-validation**. In the extreme, each fold is a single data point. This is called **leave-one-out**.

The Naivety of Naive Bayes

Naive Bayes is very simple to work with. Estimation and prediction can done in closed form, and the nice probabilistic interpretation makes it relatively easy to extend the model in various ways.

But Naive Bayes makes assumptions which seriously limit its accuracy, especially in NLP.

- The multinomial distribution assumes that each word is generated independently of all the others (conditioned on the parameter ϕ_y). Formally, we assume conditional independence:

$$\text{p}(\textit{naïve}, \textit{Bayes}; \phi) = \text{p}(\textit{naïve}; \phi)\text{p}(\textit{Bayes}; \phi). \tag{3.21}$$

- But this is clearly wrong, because words "travel together." Question for you, is it:

$$\text{p}(\textit{naïve Bayes}) > \text{p}(\textit{naïve})\text{p}(\textit{Bayes}) \tag{3.22}$$

  or...

$$\text{p}(\textit{naïve Bayes}) < \text{p}(\textit{naïve})\text{p}(\textit{Bayes}) \tag{3.23}$$

  Apply the chain rule!

Traffic lights Dan Klein makes this point with an example about traffic lights. In his hometown of Pittsburgh, there is a 1/7 chance that the lights will be broken, and both lights will be red. There is a 3/7 chance that the lights will work, and the north-south lights will be green; there is a 3/7 chance that the lights work and the east-west lights are green.

The *prior* probability that the lights are broken is 1/7. If they are broken, the conditional likelihood of each light being red is 1. The prior for them not being broken is 6/7. If they are not broken, the conditional likelihood of each being light being red is 1/2.

Now, suppose you see that both lights are red. According to Naive Bayes, the probability that the lights are broken is $1/7 \times 1 \times 1 = 1/7 = 4/28$. The probability that the lights are not broken is $6/7 \times 1/2 \times 1/2 = 6/28$. So according to naive Bayes, there is a 60% chance that the lights are not broken!

What went wrong? We have made an independence assumption to factor the probability $P(R, R \mid \text{not-broken}) = P_{\text{north-south}}(R \mid \text{not-broken}) P_{\text{east-west}}(R \mid \text{not-broken})$. But this independence assumption is clearly incorrect, because $P(R, R \mid \text{not-broken}) = 0$.

Less Naive Bayes? Of course we could decide not to make the naive Bayes assumption, and model $P(R, R)$ explicitly. But this idea does not scale when the feature space is large (as it often is in NLP). The number of possible feature configurations grows exponentially, so our ability to estimate accurate parameters will suffer from high variance. With an infinite amount of data, we'd be fine (in theory, maybe not in practice); but we never have that. Naive Bayes accepts some bias (because of the incorrect modeling assumption) in exchange for lower variance.

3.3 Recap

- Bag-of-words representation $\boldsymbol{f}(\boldsymbol{x}, y)$
- Classification as a dot-product $\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y)$
- Naive Bayes
  - Define $\text{p}(\boldsymbol{x}, \boldsymbol{y})$ via a *generative model*
  - Prediction: $\hat{y} = \arg\max_y \text{p}(\boldsymbol{x}_i, y)$
  - Learning:

$$\begin{aligned}
\boldsymbol{\theta} &= \arg\max_{\boldsymbol{\theta}} \text{p}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) \\
\text{p}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) &= \prod_i \text{p}(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) = \prod_i \text{p}(\boldsymbol{x}_i | y_i) \text{p}(y_i) \\
\phi_{y,j} &= \frac{\sum_{i:Y_i=y} x_{ij}}{\sum_{i:Y_i=y} \sum_j x_{ij}} \\
\mu_y &= \frac{\text{count}(Y = y)}{N}
\end{aligned}$$

This gives the maximum-likelihood estimator (MLE; same as relative frequency estimator)

- Bias-variance tradeoff: MLE is high-variance, so add smoothing pseudo counts α. This reduces variance but adds bias.

Chapter 4

Sentiment analysis

Todo: add notes about sentiment analysis here

Chapter 5

Discriminative learning

5.1 Features

Naive Bayes is a simple classifier, where the weights are learned based on the joint probability of labels and words. It includes an independence assumption: all features are mutually independent, conditioned on the label.

- We have defined a **feature function** $\boldsymbol{f}(\boldsymbol{x}, y)$, which corresponds to "bag-of-words" features. While these features do violate the independence assumption, the violation is relatively mild.
- We may be interested in other features, which violate independence more severely. Can you think of any?
  - Prefixes, e.g. *anti-*, *im-*, *un-*
  - Punctuation and capitalization
  - Bigrams, e.g. *not good*, *not bad*, *least terrible*, ...

Rich feature sets generally cannot be combined with Naive Bayes because the distortions resulting from violations of the independence assumption overwhelm the additional power of better features.

$$\text{p}(\textit{not bad food}|y) \approx \text{p}(\textit{not}|y)\text{p}(\textit{bad}|y)\text{p}(\textit{food}|y) \tag{5.1}$$

$$\text{p}(\textit{not bad food}|y) \not\approx \text{p}(\textit{not}|y)\text{p}(\textit{bad}|y)\text{p}(\textit{not bad}|y)\text{p}(\textit{food}|y) \tag{5.2}$$

To use these features, we will need learning algorithms that do not rely on an independence assumption.

5.2 Perceptron

In NB, the weights can be interpreted as parameters of a probabilistic model. But this model requires an independence assumption that usually does not hold, and limits our choice of features.

Why not forget about probability and learn the weights in an error-driven way?

- Until converged, at each iteration t
  - Select an instance i
  - Let $\hat{y} = \arg\max_y \boldsymbol{\theta}_t^\top \boldsymbol{f}(\boldsymbol{x}_i, y)$
  - If $\hat{y} = y_i$, do nothing
  - If $\hat{y} \neq y_i$, set $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \boldsymbol{f}(\boldsymbol{x}_i, \hat{y})$

Basically we are saying: if you make a mistake, increase the weights for features which are active with the correct label y_i, and decrease the weights for features which are active with the guessed label $\hat{y}$.

This seems like a cheap heuristic, right? Will it really work? In fact, there is some nice theory for the perceptron.

- If there is a set of weights that correctly separates your data, then your data is **separable**.
- Formally, your data is (linearly) separable if there exists a set of weights $\boldsymbol{\theta}$ such that

$$\forall \boldsymbol{x}_i, y_i, \ \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y_i) > \max_{y' \neq y_i} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y') \tag{5.3}$$

- If your data is linearly separable, it can be proven that the perceptron algorithm will eventually find a separator.
- What if your data is not separable?
  - the number of errors is bounded...
  - but the algorithm will thrash. That is, the weights will cycle between different values, and will never converge.

The perceptron is an **online** learning algorithm.

- This means that it adjusts the weights after every example.

- This is different from Naïve Bayes, which computes corpus statistics and then sets the weights in a single operation. This is a **batch learning** algorithm.
- Other algorithms are **iterative**, in that they perform multiple updates to the weights, but are also **batch**, in that they have to use all the training data to compute the update. We'll mention two of those algorithms later.

Voted (averaged) perceptron

One solution to thrashing is to average the weights across all iterations:

$$\overline{\boldsymbol{\theta}} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{\theta}_t$$
$$y = \arg\max_y \overline{\boldsymbol{\theta}}^\top \boldsymbol{f}(\boldsymbol{x}, y)$$

There is some analysis showing that voting can improve generalization (Freund and Schapire, 1999; Collins, 2002). However, this rule as described here is not practical. Can you see why not, and how to fix it?

5.3 Loss functions and large-margin classification

Naive Bayes chooses the weights $\boldsymbol{\theta}$ by maximizing the likelihood $\mathrm{p}(\boldsymbol{x}, \boldsymbol{y})$. This can be seen, equivalently, as maximizing the log-likelihood (due to the monotonicity of the log function), and as **minimizing** the negative log-likelihood. This negative log-likelihood can therefore be viewed as a **loss function**, which is minimized:

$$\log \mathrm{p}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \mathrm{p}(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) \tag{5.4}$$
$$\ell_{\mathrm{NB}}(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i) = -\log \mathrm{p}(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) \tag{5.5}$$
$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \ell_{\mathrm{NB}}(\boldsymbol{\theta}, \boldsymbol{x}_i, y_i) \tag{5.6}$$

This may seem confusing and backwards, but loss functions provide a very general framework in which to compare many approaches to machine learning.

For example, even though the perceptron is not a probabilistic model, it is also trying to minimize a **loss function**:

$$\ell_{\text{perceptron}}(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i) = \begin{cases} 0, & y_i = \arg\max_y \boldsymbol{\theta}^\top \boldsymbol{f}(x_i, y) \\ 1, & \text{otherwise} \end{cases} \tag{5.7}$$

This loss function has some pros and cons in comparison with Naive Bayes.

- ℓ_{NB} can suffer **infinite** loss on a single example, which suggests it will overemphasize some examples, and underemphasize others.
- $\ell_{\text{perceptron}}$ treats all errors equally. It only cares if the example is correct, and not about how confident the classifier was. Since we usually evaluate on accuracy, this is a better match.
- $\ell_{\text{perceptron}}$ is non-convex[1] and discontinuous. Finding the global optimum is intractable when the data is not separable.

We can fix this last problem by defining a loss function that behaves more nicely. To do this, let's define the *margin* as

$$\gamma(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i) = \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \max_{y \neq y_i} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y) \tag{5.8}$$

Then we can write a convex and continuous "hinge loss" as

$$\ell_{\text{hinge}}(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i) = \begin{cases} 0, & \gamma(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i) \geq 1, \\ 1 - \gamma(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i), & \text{otherwise} \end{cases} \tag{5.9}$$

Equivalently, we can write $\ell_{\text{hinge}}(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i) = (1 - \gamma(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i))_+$, where $(x)_+$ indicates the positive part of x.

Essentially, we want a *margin* of at least 1 between the score for the true label and the best-scoring alternative, which we have written $\hat{y}$.

The hinge and perceptron loss functions are shown in Figure 5.1.

[1] As a reminder, a function f is convex iff $\alpha f(x_i) + (1-\alpha) f(x_j) \geq f(\alpha x_i + (1-\alpha) x_j)$, for all $\alpha \in [0, 1]$ and for all x_i and x_j on the domain of the function. Convexity implies that any local minimum is also a global minimum, and there are a wide array of techniques for optimizing convex functions (Boyd and Vandenberghe, 2004)

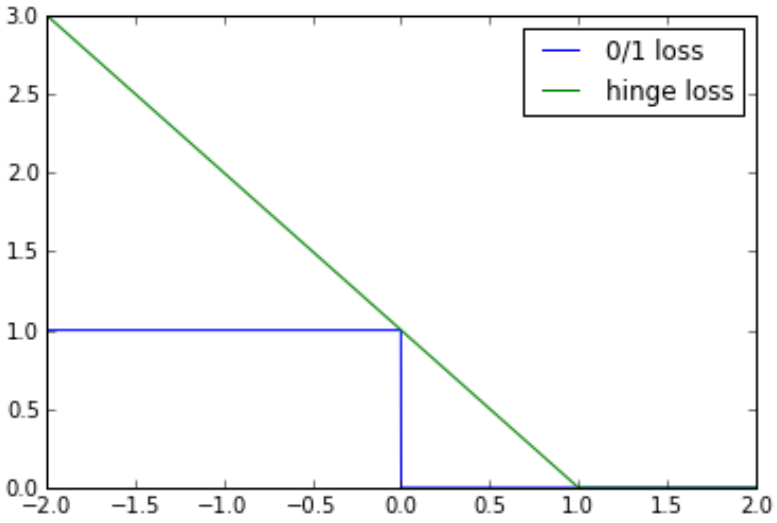


Figure 5.1: Hinge and perceptron loss functions

Large-margin online classification

Note that we can write $\boldsymbol{\theta} = s\boldsymbol{u}$, where $||\boldsymbol{u}||_2 = 1$. Think of s as the magnitude and $\boldsymbol{u}$ as the direction of the vector $\boldsymbol{\theta}$. If the data is separable, there are many values of s which attain zero hinge loss. For generality, we will try to make the smallest magnitude change to $\boldsymbol{\theta}$ possible.[2]

At step t, we optimize:

$$\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}_t||^2 \; s.t. \; \ell_{\text{hinge}}(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i) = 0 \tag{5.10}$$

Assuming that the constraint can be satisfied (i.e., the problem is linearly separabe), the optimal solution is found at,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \tau_t(\boldsymbol{f}(y_i, \boldsymbol{x}_i) - \boldsymbol{f}(\hat{y}, \boldsymbol{x}_i)) \tag{5.11}$$

$$\tau_t = \frac{\ell(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i)}{||\boldsymbol{f}(x_i, y_i) - \boldsymbol{f}(x_i, \hat{y})||^2}, \tag{5.12}$$

[2] In the support vector machine (without slack variables), we choose the smallest magnitude weights that satisfy the constraint of zero hinge loss. Pegasos is an online algorithm for training SVMs (Shwartz et al., 2007); it is similar to Passive-Aggressive.

where again $\hat{y}$ is the best scoring y according to $\boldsymbol{\theta}_t$. This solution can be obtained by introducing τ_t as a Lagrange multiplier for the constraint in (5.10).

If the data is not lienarly separable, there will be instances for which we can't meet this constraint. To deal with this, we introduce a "slack" variable ξ_i. We use the slack variable to trade off between the constraint (having a large margin) and the objective (having a small change in $\boldsymbol{\theta}$). The tradeoff is controlled by a parameter C.

$$\begin{aligned} &\min w\frac{1}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}_t||^2 + C\xi_t \\ &s.t. \ell_{\text{hinge}}(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i) \leq \xi_t, \xi_t \geq 0 \end{aligned} \tag{5.13}$$

The solution to 5.13 is,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \tau_t(\boldsymbol{f}(y_i, \boldsymbol{x}_i) - \boldsymbol{f}(\hat{y}, \boldsymbol{x}_i)) \tag{5.14}$$

$$\tau_t = \min\left(C, \frac{\ell(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i)}{||\boldsymbol{f}(\boldsymbol{x}_i, y_i) - \boldsymbol{f}(\boldsymbol{x}_i, \hat{y})||^2}\right), \tag{5.15}$$

- If C is 0, then infinite slack is permitted, and the weights will never change.
- As $C \rightarrow \infty$, no slack is permitted, and the optimization is identical to equation 5.10 and 5.12.

This algorithm is called "Passive-Aggressive" (PA; Crammer et al., 2006), because it is passive when the margin constraint is satisfied, but it aggressively changes the weights to satisfy the constraints if necessary.[3]

- PA is error-driven like the perceptron, but is more stable to violations of separability, like the averaged perceptron.
- PA allows more explicit control than the Averaged Perceptron, due to the C parameter. When C is small, we make very conservative adjustments to $\boldsymbol{\theta}$ from each instance, because the slack variables aren't very expensive. When C is large, we make large adjustments to avoid using the slack variables.
- You can also apply weight averaging to PA.

[3] A related algorithm without slack variables is called MIRA, for Margin-Infused Relaxed Algorithm (Crammer and Singer, 2003).

- **Support vector machines** (SVMs) are another learning algorithm based on the hinge loss (Burges, 1998), but they try to minimize the norm of the weights, rather than the norm of the change in the weights. They are typically trained in **batch** style, meaning that they have to read all the training instances in to compute each update. However, SVMs can also be trained in an online fashion (Shwartz et al., 2007). The LXMLS lab guide provides a simpler online learning algorithm, based on stochastic subgradient descent (Figueiredo et al., 2013).

Pros and cons of Perceptron and PA

- Perceptron and PA are error-driven, which means they usually do better in practice than naive Bayes.
- They are also online, which means we can learn without having our whole dataset in memory at once. NB can also be estimated online, in the sense that you can stream the data and store the counts.
- The original perceptron doesn't behave well if the data is not separable, and doesn't make it easy to control model complexity.
- All these models lack a probabilistic interpretation. Probabilities are useful because they quantify the classification certainty, allowing us to compute expected utility, and to incorporate the classifier in more complex probabilistic models.

5.4 Logistic regression

Logistic regression is error-driven like the perceptron, but probabilistic like Naive Bayes. This is useful in case we want to quantify the uncertainty about a classification decision.

Recall that NB selects weights to optimize the joint probability $\mathrm{p}(y, \boldsymbol{x})$.

- In NB, we factor this as $\mathrm{p}(y, \boldsymbol{x}) = \mathrm{p}(\boldsymbol{x}|y)\mathrm{p}(y)$.
- But we could equivalently write $\mathrm{p}(y, \boldsymbol{x}) = \mathrm{p}(y|\boldsymbol{x})\mathrm{p}(\boldsymbol{x})$.

Since we always know $\boldsymbol{x}$, we really care only about $\mathrm{p}(y|\boldsymbol{x})$. Logistic regression optimizes this directly. To do this, we have to define the probability function

differently. We define the conditional probability directly, as,

$$\mathrm{p}(y|\boldsymbol{x}) = \frac{\exp\left(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}, y')\right)} \tag{5.16}$$

$$\log \mathrm{p}(y|\boldsymbol{x}) = \sum_{i=1}^{N} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \log \sum_{y' \in \mathcal{Y}} \exp \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y') \tag{5.17}$$

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log \mathrm{p}(y_i|\boldsymbol{x}_i; \boldsymbol{\theta}) \tag{5.18}$$

Inside the sum, we have the (additive inverse of) the **logistic loss**.

- In binary classification, we can write this as

$$\ell_{\text{logistic}}(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i) = -(y_i \boldsymbol{\theta}^\top \boldsymbol{x}_i - \log\left(1 + \exp \boldsymbol{\theta}^\top \boldsymbol{x}_i\right)) \tag{5.19}$$

- In multi-class classification, we have,[4]

$$\ell_{\text{logistic}}(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i) = -(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \log \sum_{y' \in \mathcal{Y}} \exp \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y')) \tag{5.20}$$

The logistic loss is shown in Figure 5.2. Because it is smooth and convex, we can optimize it through gradient steps:

[4]The log-sum-exp term is very common in machine learning. It is numerically instable because you can underflow if the inner product is small, and overflow if the inner product is large. Libraries like `scipy` contain special functions for computing `logsumexp`, but with some thought, you should be able to see how to create an implementation that is numerically stable.

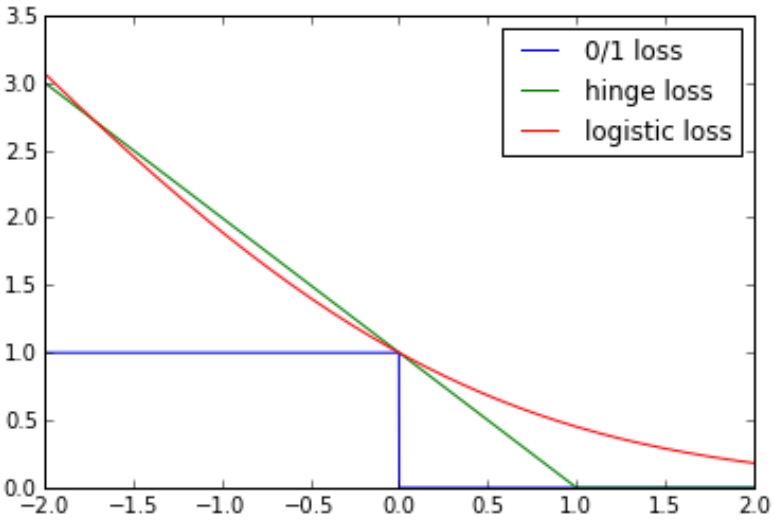


Figure 5.2: Hinge, perceptron, and logistic loss functions

$$\ell = \sum_{i=1}^{N} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \log \sum_{y' \in \mathcal{Y}} \exp \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y') \tag{5.21}$$

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{N} \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \frac{\sum_{y' \in \mathcal{Y}} \exp \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y') \boldsymbol{f}(\boldsymbol{x}_i, y')}{\sum_{y'' \in \mathcal{Y}} \exp \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y'')} \tag{5.22}$$

$$= \sum_{i=1}^{N} \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \sum_{y' \in \mathcal{Y}} \frac{\exp \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y')}{\sum_{y'' \in \mathcal{Y}} \exp \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_i, y'')} \boldsymbol{f}(\boldsymbol{x}_i, y') \tag{5.23}$$

$$= \sum_{i=1}^{N} \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \sum_{y' \in \mathcal{Y}} \mathrm{p}(y' | \boldsymbol{x}_i; \boldsymbol{\theta}) \boldsymbol{f}(\boldsymbol{x}_i, y') \tag{5.24}$$

$$= \sum_{i=1}^{N} \boldsymbol{f}(\boldsymbol{x}_i, y_i) - E[\boldsymbol{f}(\boldsymbol{x}_i, y)] \tag{5.25}$$

This gradient has a pleasing interpretation as the difference between the observed counts and the expected counts.[5] Compare this gradient with the percep-

[5] Recall that the definition of an expected value $E[f(x)] = \sum_x f(x)\mathrm{p}(x)$

tron and PA update rules.

The bias-variance tradeoff is handled by penalizing large $\boldsymbol{\theta}$ in the objective, adding a term of $\frac{\lambda}{2}||\boldsymbol{\theta}||_2^2$. This is called L2 regularization, because of the L2 norm. It can be viewed as placing a 0-mean Gaussian prior on $\boldsymbol{\theta}$.

This penalty contributes a term of $\lambda\boldsymbol{\theta}$ to the gradient, so we have,

$$\ell = \sum_{i=1}^{N} \boldsymbol{\theta}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \log \sum_{y' \in \mathcal{Y}} \exp \boldsymbol{\theta}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y') + \frac{\lambda}{2}||\boldsymbol{\theta}||_2^2 \tag{5.26}$$

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{N} \boldsymbol{f}(\boldsymbol{x}_i, y_i) - E[\boldsymbol{f}(\boldsymbol{x}_i, y)] - \lambda\boldsymbol{\theta}. \tag{5.27}$$

Optimization

Batch optimization In batch optimization, you keep all the data in memory and iterate over it many times.

- The logistic loss is smooth and convex, so we can find the global optimum using gradient descent. But in practice, this can be very slow.
- Second-order (Newton) optimization would incorporate the inverse Hessian. The Hessian is

  $$H_{i,j} = \frac{\partial^2}{\partial w_i \partial w_j} \ell, \tag{5.28}$$

  but this matrix is usually too big to deal with.
- In practice, people usually apply **quasi-Newton optimization**, which approximates the Hessian matrix. The specific method that is particularly popular is L-BFGS[6] NLP people usually treat L-BFGS as a black box; you will typically pass it a pointer to a function that computes the likelihood and gradient. L-BFGS is provided in `scipy.optimize`.

Online optimization In online optimization, you consider one example (or a "mini-batch" of a few examples) at a time. *Stochastic gradient descent* makes a

[6] A friend of mine told me you can remember the order of the letters as "Large Big Friendly Giants." Does this help you?

stochastic online approximation to the overall gradient:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta_t \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^{(t)}, \boldsymbol{x}, \boldsymbol{y}) \tag{5.29}$$

$$= \boldsymbol{\theta}^{(t)} - \eta_t (\lambda \boldsymbol{\theta}^{(t)} - \sum_i^N \boldsymbol{f}(\boldsymbol{x}_i, y_i) - E[\boldsymbol{f}(\boldsymbol{x}_i, y)]) \tag{5.30}$$

$$= (1 - \lambda\eta_t)\boldsymbol{\theta}^{(t)} + \eta_t \sum_i^N \boldsymbol{f}(\boldsymbol{x}_i, y_i) - E[\boldsymbol{f}(\boldsymbol{x}_i, y)] \tag{5.31}$$

$$\approx (1 - \lambda\eta_t)\boldsymbol{\theta}^{(t)} + N\eta_t \left(\boldsymbol{f}(\boldsymbol{x}_{i(t)}, y_{i(t)}) - E[\boldsymbol{f}(\boldsymbol{x}_{i(t)}, y)]\right) \tag{5.32}$$

where η_t is the **stepsize** at iteration t, and $\langle \boldsymbol{x}_{i(t)}, y_{i(t)} \rangle$ is the instance selected at iteration t. (So here we are setting the mini-batch size equal to one.) As always, N is the total number of instances. As above, the expectation is equal to a weighted sum over the labels,

$$E[\boldsymbol{f}(\boldsymbol{x}_{i(t)}, y)] = \sum_{y' \in \mathcal{Y}} \mathrm{p}(y' | \boldsymbol{x}_{i(t)}; \boldsymbol{\theta}) \boldsymbol{f}(\boldsymbol{x}_{i(t)}, y'). \tag{5.33}$$

- Note how similar this update is to the perceptron!
- If we set $\eta_t = \eta_0 t^{-\alpha}$ for $\alpha \in [1, 2]$, we have guaranteed convergence.
- We can also just fix η_t to a small value, like 10^-3. (This is what we will do in the problem set.)
- In either case, we could tune this parameter on a development set. However, it would be acceptable to just find a value that gives a good regularized log-likelihood on the training set, since this parameter relates to the quality of the optimization, and not the generalization capability of the classifier.
- In theory, we select $\langle \boldsymbol{x}_{i(t)}, y_{i(t)} \rangle$ at random, but in practice we usually just iterate through the dataset.
- We can fold N into η and λ, so that $\eta^* = N\eta$ and $\lambda^* = \lambda \frac{\eta^*}{N}$. This gives the more compact form,

$$(1 - \lambda^*\eta_t^*)\boldsymbol{\theta}^{(t)} + \eta_t^* \left(\boldsymbol{f}(\boldsymbol{x}_{i(t)}, y_{i(t)}) - E[\boldsymbol{f}(\boldsymbol{x}_{i(t)}, y)]\right) \tag{5.34}$$

For more on stochastic gradient descent, as applied to a number of different learning algorithms, see (Zhang, 2004) and (Bottou, 1998). Murphy (2012) traces SGD to a 1978 paper by GT's own Arkadi Nemirovski (Nemirovski and Yudin, 1978). You can find several recent chapters about online optimization in the edited volume by Sra et al. (2012).

Adagrad Recent work has shown that you can often learn more quickly by using an **adaptive** step-size, which is different for every feature (Duchi et al., 2011). Specifically, in the **Adagrad** algorithm (adaptive gradient), you keep track of the sum of the squares of the gradients for each feature, and rescale the learning rate by its inverse:

$$\boldsymbol{g}_t = -\boldsymbol{f}(\boldsymbol{x}_i, y_i) + \sum_{y' \in \mathcal{Y}} \text{p}(y' \mid \boldsymbol{x}_i)\boldsymbol{f}(\boldsymbol{x}_i, y_i) + \lambda\boldsymbol{\theta} \tag{5.35}$$

$$\theta_j^{(t+1)} \leftarrow \theta_j^{(t)} - \frac{\eta}{\sqrt{\sum_{t'=1}^{t} g_{t,j}^2}} g_{t,j}, \tag{5.36}$$

where j iterates over features in $\boldsymbol{f}(\boldsymbol{x}, y)$. The effect of this is that features with consistently large gradients are updated more slowly. Another way to view this update is that rare features are taken more seriously, since their sum of squared gradients will be smaller. Adagrad seems to require less careful tuning of η, and Dyer (2014) reports that $\eta = 1$ works for a wide range of problems.

Note that the Adagrad update can apply to any smooth loss function, including the hinge loss defined in Equation 5.9.

Names

Logistic regression is so named because in the binary case where $y \in \{0, 1\}$, we are performing a regression of $\boldsymbol{x}$ against $\boldsymbol{y}$, after passing the inner product $\boldsymbol{\theta}^\text{T}\boldsymbol{x}$ through a logistic transformation. You could always do a linear regression, but this would ignore the fact that the $\boldsymbol{y}$ is limited to a few values.

- Logistic regression is also called **maximum conditional likelihood** (MCL), because it maximizes... the conditional likelihood $\text{p}(y \mid \boldsymbol{x})$.

- Logistic regression can be viewed as part of a larger family, called **generalized linear models**. If you use R, you are probably familiar with `glmnet`.

- Logistic regression is also called **maximum entropy**, especially in the earlier NLP literature (Berger et al., 1996). This is due to an alternative formulation, which tries to find the maximum entropy probability function that satisfies moment-matching constraints.

The moment matching constraints specify that the empirical counts of each label-feature pair should match the expected counts:

$$\forall j, \sum_{i=1}^{N} f_j(\boldsymbol{x}_i, y_i) = \sum_{i=1}^{N} \sum_{y \in \mathcal{Y}} \mathrm{p}(y \mid \boldsymbol{x}_i; \boldsymbol{\theta}) f_j(\boldsymbol{x}_i, y) \tag{5.37}$$

Note that this constraint will be met exactly when the derivative of the likelihood function (equation 5.25) is equal to zero. However, this will be true for many values of $\boldsymbol{\theta}$. Which should we choose?

The entropy of a conditional likelihood function $P(Y|X)$ is

$$H(P) = -\sum_{x \in \mathcal{X}} \tilde{\mathrm{p}}(x) \sum_{y \in \mathcal{Y}} \mathrm{p}(y|x) \log \mathrm{p}(y|x), \tag{5.38}$$

where $\tilde{\mathrm{p}}(x)$ is the *empirical probability* of x. We compute an empirical probability by summing over all the instances in training set.

If the entropy is large, this function is smooth across possible values of y; if it is small, the function is sharp. The entropy is zero if $\mathrm{p}(y|x) = 1$ for some particular $Y = y$ and zero for everything else. By saying we want maximum-entropy classifier, we are saying we want to make the least commitments possible, while satisfying the moment-matching constraints:

$$\max_{\boldsymbol{\theta}} \quad -\sum_{\boldsymbol{x}} \tilde{\mathrm{p}}(\boldsymbol{x}) \sum_{y} \mathrm{p}(y|\boldsymbol{x}; \boldsymbol{\theta}) \log \mathrm{p}(y|\boldsymbol{x}; \boldsymbol{\theta}) \tag{5.39}$$

$$s.t. \quad \forall j, \sum_{i=1}^{N} f_j(\boldsymbol{x}_i, y_i) = \sum_{i=1}^{N} \sum_{y} \mathrm{p}(y|\boldsymbol{x}_i; \boldsymbol{\theta}) f_j(\boldsymbol{x}_i, y) \tag{5.40}$$

Now, the solution to this constrained optimization problem is identical to the maximum conditional likelihood (logistic-loss) formulation we've considered in the previous section.

This view of logistic regression is arguably a little dated, but it's useful to understand what's going on. The information-theoretic concept of entropy will pop up again a few times in the course. For a tutorial on maximum entropy, see `http://www.cs.cmu.edu/afs/cs/user/aberger/www/html/tutorial/tutorial.html`.

5.5 Summary of learning algorithms

- **Naive Bayes**. pros: easy and probabilistic. cons: arguably optimizes wrong objective; usually has poor accuracy, especially with overlapping features.

- **Perceptron and PA**. pros: easy, online, and error-driven. cons: not probabilistic. this can be bad in pipeline architectures, where the output of one system becomes the input for another.

- **Logistic regression**. pros: error-driven and probabilistic. cons: batch learning requires black-box software; hinge loss sometimes yields better accuracy than logistic loss.

What about non-linear classification?

The feature spaces that we consider in NLP are usually huge, so non-linear classification can be quite difficult. When the feature dimension V is larger than the number of instances N — often the case in NLP — you can always learn a linear classifier that will perfectly classify your training instances.[7] This makes selecting an appropriate **non-linear** classifier especially difficult. Nonetheless, there are some approaches to non-linear learning in NLP:

- You can add **features**, such as bigrams, which are non-linear combinations of other features. For example, the base feature $\langle$*coffee house*$\rangle$ will not fire unless both features $\langle$*coffee*$\rangle$ and $\langle$*house*$\rangle$ also fire.

- Another option is to apply non-linear transformations to the feature vector. Recall that the feature function $\boldsymbol{f}(\boldsymbol{x}, y)$ may be composed of a vector of word counts, padded by zeros. We can think of these word counts as basic features, and apply non-linear transformations, such as $\boldsymbol{x} \circ \boldsymbol{x}$ or $|\boldsymbol{x}|$.

- There is some work in NLP on using kernels for strings, bags-of-words, sequences, trees, etc. Kernelized learning algorithms are outside the scope of this class (Collins and Duffy, 2001; Zelenko et al., 2003). Kernel-based learning can be seen as a generalization of algorithms such k-nearest-neighbors, which classifies instances by considering the labels of the k most similar instances in the training set (Hastie et al., 2009).

[7] Assuming your feature matrix is full-rank.

- Boosting (Freund et al., 1999) and decision tree algorithms (Schmid, 1994) sometimes do well on NLP tasks, but they are used less frequently these days, especially as the field increasingly emphasizes big data and simple classifiers.
- More recent work has shown how **deep learning** can perform non-linear classification. One way to use deep learning in NLP is by learning word representations while jointly learning how these representations combine to classify instances (Collobert and Weston, 2008). This approach is very hot at the moment, so I will discuss it towards the end of the semester.

5.6 Summary of classifiers

So now we've talked about four different classifiers. That's it! No more classifiers in this class. Yay? Anyway, let's review.

	Naive Bayes	Logistic Regression	Perceptron	PA
Objective	Joint likelihood	Conditional likelihood	0-1 loss	Hinge loss
	$\max \sum_i \log \mathrm{p}(\boldsymbol{x}_i, y_i)$	$\max \sum_i \log \mathrm{p}(y_i \mid \boldsymbol{x}_i)$	$\min \sum_i \delta(y_i, \hat{y})$	$\sum_i [1 - \gamma(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i)]_+$
estimation	$\theta_{ij} = \frac{c(x_i, y=j)+\alpha}{c(y=j)+V\alpha}$	$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \sum_i \boldsymbol{f}(\boldsymbol{x}_i, y_i) - E[\boldsymbol{f}(\boldsymbol{x}_i, y)]$	$\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} + \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \boldsymbol{f}(\boldsymbol{x}_i, \hat{y})$	$\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} + \tau_t(\boldsymbol{f}(\boldsymbol{x}_i, y_i) - \boldsymbol{f}(\boldsymbol{x}_i, \hat{y}))$
tuning	smoothing α	regularizer $\lambda \|\boldsymbol{\theta}\|_2^2$	weight averaging	slack penalty C
complexity	$\mathcal{O}(NV)$	$\mathcal{O}(NVT)$	$\mathcal{O}(NVT)$	$\mathcal{O}(NVT)$
easy?	very	not really	yes	yes
probabilities?	yes	yes	no	no
features?	no	yes	yes	yes

Table 5.1: Comparison of classifiers. N = number of examples, V = number of features, T = number of instances.

Chapter 6

Word-sense disambiguation

Todo: add notes about WSD here

Chapter 7

Learning without supervision

So far we've assumed the following setup:

- A **training set** where you get observations $\boldsymbol{x}_i$ and labels y_i
- A **test set** where you only get observations $\boldsymbol{x}_i$

What if you never get labels y_i?
For example, you get a bunch of text, and you suspect that there are at least two different meanings for the word *concern*.[1]

The immediate context includes two groups of words:

- services, produces, banking, pharmaceutical, energy, electronics
- about, said, that, over, in, with, had

Suppose we plot each instance of *concern* on a graph

- x-axis is the density of words in group 1
- y-axis is the density of words in group 2

Two blobs might emerge. These blobs would correspond to two different sense of *concern*.

- But in reality, we don't know the word groupings in advance.

[1] example from Pedersen and Bruce (1997)

- We have to try to apply the same idea in a very high dimensional space, where every word gets its own dimension (and most dimensions are irrelevant!)
- Or we have to automatically find a low-dimensional projection. More on that much later in the course.

Here's a related scenario:

- You look at thousands of news articles from today
- Plot them on a graph of *Miley* vs *Syria*
- Three clumps emerge (Miley, Syria, others)
- Those clumps correspond to natural document classes
- Again, in reality this is a hugely high-dimensional graph

So these examples show that we can find structure in data, even without labels.

7.1 K-means clustering

You might know about classic clustering algorithms like K-means. These algorithms are iterative:

1. Guess the location of cluster centers.
2. Assign each point to the nearest center.
3. Re-estimate the centers as the mean of the assigned points.
4. Goto 2.

This is an algorithm for finding coherent "blobs" of documents.
There is a variant called "soft k-means."

- Instead of assigning each point $\boldsymbol{x}_i$ to a specific cluster z_i
- You assign it a distribution over clusters $q_i(z_i)$

We're now going to explore a more principled, statistical version of soft K-means, called EM clustering.

By understanding the statistical principles underlying the algorithm, we can extend it in a number of cool ways.

7.2 The Expectation-Maximization Algorithm

Let's go back to the Naive Bayes model:

$$\log \mathrm{p}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\phi}, \mu) = \sum_i \log \mathrm{p}(\boldsymbol{x}_i | y_i; \boldsymbol{\phi}) P(y_i; \mu)$$

For example, $\boldsymbol{x}$ can describe the documents that we see today, and $\boldsymbol{y}$ can correspond to their labels. But suppose we never observe y_i? Can we still do something?

Since we don't know $\boldsymbol{y}$, let's marginalize it:

$$\log \mathrm{p}(\boldsymbol{x}) = \log \sum_{\boldsymbol{y}} \mathrm{p}(\boldsymbol{x} | \boldsymbol{y}; \boldsymbol{\phi}) \mathrm{p}(\boldsymbol{y}; \mu) \tag{7.1}$$

$$= \log \sum_{\boldsymbol{y}} \prod_i \mathrm{p}(\boldsymbol{x}_i | y_i; \boldsymbol{\phi}) \mathrm{p}(y_i; \mu) \tag{7.2}$$

$$= \sum_i \log \sum_{y_i} \mathrm{p}(\boldsymbol{x}_i | y_i; \boldsymbol{\phi}) \mathrm{p}(y_i; \mu) \tag{7.3}$$

Now we introduce an auxiliary variable $\boldsymbol{q}_i$, for each y_i. We have the usual constraints: $\sum_y q_i(y) = 1$ and $\forall y, q_i(y) \geq 0$. In other words, q_i defines a probability distribution over Y, for each instance i.

Now since $\frac{q_i(y)}{q_i(y)} = 1$,

$$\log \mathrm{p}(\boldsymbol{x}) = \sum_i \log \sum_{y_i} \mathrm{p}(\boldsymbol{x}_i | y_i; \boldsymbol{\phi}) \mathrm{p}(y_i; \mu) \frac{q_i(y)}{q_i(y)}$$

$$= \sum_i \log E_q[\frac{\mathrm{p}(\boldsymbol{x}_i | y; \boldsymbol{\phi}) \mathrm{p}(y; \mu)}{q_i(y)}],$$

by the definition of expectation. (Note that E_q just means the expectation under the distribution q.)

Now we apply *Jensen's inequality*. Jensen's equality says that because $\log$ is concave, we can push it inside the expectation, and obtain a lower bound.

$$\log \mathrm{p}(\boldsymbol{x}) \geq \sum_i E_q[\log \frac{\mathrm{p}(\boldsymbol{x}_i | y; \boldsymbol{\phi}) \mathrm{p}(y_i; \mu)}{q_i(y)}]$$

$$\mathcal{J} = \sum_i E_q[\log \mathrm{p}(\boldsymbol{x}_i | y; \boldsymbol{\phi})] + E_q[\log \mathrm{p}(y; \mu)] - E_q[q_i(y)]$$

By maximizing $\mathcal{J}$, we are maximizing a lower bound on the joint log-likelihood $\log \mathrm{p}(\boldsymbol{x})$.

Now, $\mathcal{J}$ is a function of two arguments:

- the distributions $q_i(\boldsymbol{y})$ for each i
- the parameters μ and $\boldsymbol{\phi}$

We'll optimize with respect to each of these in turn, holding the other one fixed.

The E-step

First, we expand the expectation in the lower bound as:

$$\begin{aligned}\mathcal{J} &= \sum_i E_q[\log \mathrm{p}(\boldsymbol{x}_i|y;\boldsymbol{\phi})] + E_q[\log \mathrm{p}(y;\mu)] - E_q[q_i(y)] \\ &= \sum_i \sum_y q_i(y) \left(\log \mathrm{p}(\boldsymbol{x}_i|Y_i = y;\boldsymbol{\phi}) + \log \mathrm{p}(y;\mu) - \log q_i(y)\right)\end{aligned}$$

As in relative frequency estimation of Naive Bayes, we need to add a Lagrange multiplier to ensure $\sum_y q_i(y) = 1$, so

$$\mathcal{J} = \sum_i \sum_y q_i(y) \left(\log \mathrm{p}(\boldsymbol{x}_i|Y_i = y;\boldsymbol{\phi}) + \log \mathrm{p}(y;\mu) - \log q_i(y)\right) + \lambda_i(1 - \sum_y q_i(y))$$

$$\begin{aligned}\frac{\partial \mathcal{J}}{\partial q_i(y)} &= \log \mathrm{p}(\boldsymbol{x}_i|Y_i = y;\boldsymbol{\phi}) + \log \mathrm{p}(y;\boldsymbol{\theta}) - \log q_i(y) - 1 - \lambda_i \\ \log q_i(y) &= \log \mathrm{p}(\boldsymbol{x}_i|Y_i = y;\boldsymbol{\phi}) + \log \mathrm{p}(y;\mu) - 1 - \lambda_i \\ q_i(y) &\propto \mathrm{p}(\boldsymbol{x}_i|Y_i = y;\boldsymbol{\phi})\mathrm{p}(y;\mu) \\ &\propto \mathrm{p}(\boldsymbol{x}_i, y;\boldsymbol{\phi},\mu) \\ q_i(y) &= \frac{\mathrm{p}(\boldsymbol{x}_i, y;\boldsymbol{\phi},\mu)}{\sum_{y'} \mathrm{p}(\boldsymbol{x}_i, y';\boldsymbol{\phi},\mu)} \\ &= P(Y_i = y|\boldsymbol{x}_i;\boldsymbol{\theta},\phi)\end{aligned}$$

After normalizing, each $q_i(y)$ – which is the soft distribution over clusters for data $\boldsymbol{x}_i$ – is set to the conditional probability $P(y_i|\boldsymbol{x}_i)$ under the current parameters $\mu, \boldsymbol{\phi}$.

This is called the E-step, or "expectation step," because it is derived from updating the expected likelihood under $q(\boldsymbol{y})$.

The M-step

Next, we hold $q(\boldsymbol{y})$ fixed and update the parameters. Let's do $\boldsymbol{\phi}$, which parametrizes $\text{p}(\boldsymbol{x}|y)$. Again, we start by adding Lagrange multipliers to the lower bound,

$$\mathcal{J} = \sum_i \sum_y q_i(y) \left(\log \text{p}(\boldsymbol{x}_i|Y_i = y; \boldsymbol{\phi}) + \log \text{p}(y; \mu) - \log q_i(y)\right) + \sum_y \lambda_y (1 - \sum_j \phi_{y,j})$$

$$\frac{\partial \mathcal{J}}{\partial \phi_{y,j}} = \sum_i q_i(y) \frac{x_{i,j}}{\phi_{y,j}} - \lambda_y$$

$$\lambda_h \phi_{y,j} = \sum_i q_i(y) x_{i,j}$$

$$\phi_{y,j} = \frac{\sum_i q_i(y) x_{i,j}}{\sum_{j'} \sum_i q_i(y) x_{i,j'}} = \frac{E_q[\text{count}(y,j)]}{E_q[\text{count}(y)]}$$

So $\boldsymbol{\phi}_y$ is now equal to the relative frequency estimate of the **expected counts** under the distribution $q(y)$.

- As in supervised Naïve Bayes, we can apply smoothing to add α to all these counts
- The update for μ is identical: $\mu_y \propto \sum_i q_i(y)$, the expected proportion of cluster $Y = y$. If needed, we can add smoothing here too.
- So, everything in the M-step is just like Naive Bayes, except we used expected counts rather than observed counts.

Coordinate ascent

Algorithms that alternate between updating various subsets of the parameters are called "coordinate-ascent" algorithms.

The objective function $\mathcal{J}$ is **biconvex**, meaning that it is separately convex in $q(\boldsymbol{y})$ and $\langle \mu, \boldsymbol{\phi} \rangle$, but it is not jointly convex.

- Each step is guaranteed not to decrease $\mathcal{J}$
- This is called hill-climbing: you never go down.
- Specifically, EM is guaranteed to converge to a **local optima** – a point which is as good or better than any of its immediate neighbors. But there may be many such points.

- But the overall procedure is **not** guaranteed to find a global maximum.
- This means that initialization is important: where you start can determine where you finish.
- This is not true in most of the supervised learning algorithms that we have considered, such as logistic regression; in that case, we are optimizing $\log \mathrm{p}(\boldsymbol{y}|x;\boldsymbol{\theta})$, which is defined so as to be convex with respect to the parameter $\boldsymbol{\theta}$. This means that for logistic regression (and many other supervised learning algorithms), we don't need to worry about initialization, because it won't affect our ultimate solution: we are guaranteed to reach the global minimum.

7.3 Applications of EM

EM is not really an "algorithm" like, say, quicksort. Rather, it's a framework for learning with missing data. The recipe for using EM on a problem of interest to you is something like this:

- Introduce latent variables $\boldsymbol{z}$, such that it's easy to write the probability $P(\mathcal{D}, \boldsymbol{z})$, where $\mathcal{D}$ is your observed data, and easy to estimate the associated parameters.
- Derive the E-step updates for $q(\boldsymbol{z})$, which is typically factored as $q(\boldsymbol{z}) = \prod_i q_{z_i}(z_i)$.

Some applications of this basic setup are presented here.

Word sense clustering

In the "demos" folder, you can find a demonstration of expectation-maximization for word sense clustering. I assume we know that there are two senses, and that the senses can be distinguished by the contextual information in the document. The basic framework is identical to the clustering model of EM as presented above.

Semi-supervised learning

Nigam et al. (2000) offer another application of EM: **semi-supervised learning**. They apply this idea to document classification in the classic "20 Newsgroup" dataset.

- In this setting, we have labels for some of the instances, $\langle \boldsymbol{x}^{(\ell)}, \boldsymbol{y}^{(\ell)} \rangle$, but not for others, $\langle \boldsymbol{x}^{(u)} \rangle$.
- Can unlabeled data improve learning?

We will choose parameters to maximize the joint likelihood,

$$\log \mathrm{p}(\boldsymbol{x}^{(\ell)}, \boldsymbol{x}^{(u)}, \boldsymbol{y}^{(\ell)}) = \log \mathrm{p}(\boldsymbol{x}^{(\ell)}, \boldsymbol{y}^{(\ell)}) + \log \mathrm{p}(\boldsymbol{x}^{(u)}) \tag{7.4}$$

- We treat the labels of the unlabeled documents as missing data. In the E-step we impute $q(y)$ for the unlabeled documents only.
- The M-step computes estimates of μ and $\boldsymbol{\phi}$ from the sum of the observed counts from $\langle \boldsymbol{x}^{(\ell)}, \boldsymbol{y}^{(\ell)} \rangle$ and the expected counts from $\langle \boldsymbol{x}^{(u)} \rangle$ and $q(\boldsymbol{y})$.
- We can further parametrize this approach by weighting the unlabeled documents by a scalar λ, which is a tuning parameter.

Multi-component modeling

- One of the classes in 20 newsgroups is `comp.sys.mac.hardware`.
- Suppose that there are two kinds of posts: reviews of new hardware, and question-answer posts about hardware problems.
- The language in these **components** of the mac.hardware class might have little in common.
- So we might do better if we model these components separately.

We can envision a new generative process here:

- For each document i,
  - draw the label $y_i \sim \text{Categorical}(\theta)$
  - draw the component $z_i | y_i \sim \text{Categorical}(\psi_{y_i})$
  - draw the vector of counts $\boldsymbol{x}_i | z_i \sim \text{Multinomial}(\phi_{z_i})$

Our labeled data includes $\langle \boldsymbol{x}_i, y_i \rangle$, but not z_i, so this is another case of missing data.

$$\begin{aligned}\text{p}(\boldsymbol{x}_i, y_i) =& \sum_z \text{p}(\boldsymbol{x}_i, y_i, z) \\ =& \text{p}(\boldsymbol{x}_i | z; \boldsymbol{\phi}) \text{p}(z | y_i; \psi) \text{p}(y_i; \mu)\end{aligned}$$

Again, we can apply EM

- We need a distribution over the missing data, $q_i(z)$. This is updated during the E-step.
- During the m-step, we compute:

$$\begin{aligned}\psi_{y,z} =& \frac{E_q[\text{count}(y, z)]}{\sum_{z'} E_q[\text{count}(y, z')]} \\ \boldsymbol{\phi}_{j,y,z} =& \frac{E_q[\text{count}(z, j)]}{\sum_{j'} E_q[\text{count}(z, j')]}\end{aligned}$$

- Suppose we assume each class y is associated with K components, $\mathcal{Z}_y$. We can add a constraint to the E-step so that $q_i(z) = 0$ if $z \notin \mathcal{Z}_y \wedge Y_i = y$.

Chapter 8

Language models

A **language model** is used to compute the probability of a sequence of text. Why would we want to do this? Thus far, we have considered problems where text is the **input**, and we want to select an output, such as a document class or a word sense. But in many of the most prominent problems in language technology, text itself is the output:

- machine translation
- speech recognition
- summarization

As we will soon see, we can produce more **fluent** text output by computing the probability of the text.

Specifically, suppose we have a vocabulary of word types

$$\mathcal{V} = \{\textit{aardvark}, \textit{abacus}, \ldots, \textit{zither}\} \tag{8.1}$$

Given a sequence of word tokens $w_1, w_2, \ldots, w_M$, with $w_i \in \mathcal{V}$, we would like to compute the probability $\text{p}(w_1, w_2, \ldots, w_M)$. We will do this in a data-driven way, assuming we have a **corpus** of text.

- For now, we'll assume that the vocabulary $\mathcal{V}$ covers all the word tokens that we will ever see. Of course, we can enforce this by allocating a special token ♠ for unknown words. However, this might not be a great solution, as we will see later.
- Language models typically make an independence assumption across sentences, $\text{p}(\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots) = \prod_j \text{p}(\boldsymbol{s}_j)$, where each sentence $\boldsymbol{s}_j = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{N_j}]$.

So for our purposes, it is sufficient to compute the probability of sentences. The justification for this assumption is that the probability of words that are not in the same sentence don't depend on each other too much. Clearly this isn't true: once I mention *Manuel Noriega* once in a document, I'm far more likely to mention him again (Church, 2000). But the dependencies between words within a sentence are usually even stronger, and are more relevant to the fluency considerations inherent in applications such as translation and speech recognition (which are typically evaluated at the sentence level anyway).

So how can we compute the probability of a sentence? The simplest idea would be to apply a **relative frequency estimator**:

$$\text{p}(\textit{Computers are useless, they can only give you answers}) \tag{8.2}$$

$$= \frac{\text{count}(\textit{Computers are useless, they can only give you answers})}{\text{count}(\text{all sentences ever spoken})} \tag{8.3}$$

It's useful to think about this estimator in terms of bias and variance.

- In the theoretical limit of infinite data, it might work. But in practice, we are asking for accurate counts over an infinite number of events, since sentences can be arbitrarily long.
- Even if we set an aggressive upper bound of, say, $n = 20$, the number of possible sentences is $\#|\mathcal{V}|^{20}$. A small vocabularly for English would have $\#|\mathcal{V}| = 10^4$, so we would have 10^{80} possible sentences.
- Clearly, this estimator is extremely data-hungry. We need to introduce bias to have a chance of making reliable estimates.

Are language models meaningful? What are the probabilities of the following two sentences?

- *Colorless green ideas sleep furiously*
- *Furiously sleep ideas green colorless*

Noam Chomsky used this pair of examples to argue that the probability of a sentence is a meaningless concept:

- Any English speaker can tell that the first sentence is grammatical but the second sentence is not.
- Yet neither sentence, nor their substrings, had ever appeared at the time that Chomsky wrote this article (they have appeared lots since then).
- Thus, he argued, empirical probabilities can't distinguish grammatical from ungrammatical sentences.

Pereira (2000) showed that by identifying *classes* of words (e.g., noun, verb, adjective, adverb — but not necessarily these grammatical categories), it is easy to show that the first sentence is more probable than the second. We will talk about class-based language models later.

Are language models useful? Suppose we want to translate a sentence from Spanish:

- *El cafe negro me gusta mucho*.
- Word-for-word: *The coffee black me pleases much*.
- But a good language model of English will tell us:

$$P(\textit{The coffee black me pleases much}) < P(\textit{I like black coffee a lot}) \tag{8.4}$$

- How can we use this fact?

Warren Weaver on translation as decoding:

> When I look at an article in Russian, I say: 'This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.'

This motivates a generative model (like Naive Bayes!):

- English sentence $\boldsymbol{w}^{(e)}$ generated from language model $\mathrm{p}_e(\boldsymbol{w}^{(e)})$
- Spanish sentence $\boldsymbol{w}^{(s)}$ generated from noisy channel $\mathrm{p}_{s|e}(\boldsymbol{w}^{(s)}|\boldsymbol{w}^{(e)})$

(picture)
Then the **decoding** problem is: $\max_{\boldsymbol{w}^{(e)}} \mathrm{p}(\boldsymbol{w}^{(e)}|\boldsymbol{w}^{(s)}) \propto \mathrm{p}(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(e)}) = \mathrm{p}(\boldsymbol{w}^{(e)})\mathrm{p}(\boldsymbol{w}^{(s)}|\boldsymbol{w}^{(e)})$

- The **translation model** is $\text{p}(\boldsymbol{w}^{(s)}|\boldsymbol{w}^{(e)})$. This ensures the **adequacy** of the translation.
- The **language model** is $\text{p}(\boldsymbol{w}^{(e)})$. This ensures the **fluency** of the translation.

What else can we model with a noisy channel?

- Speech recognition (original = words; encoded = sound)
- Spelling correction (original = well-spelled text; encoded = text with spelling mistakes)
- Part of speech tagging (original = tags; encoded = words)
- Parsing (original = parse tree; encoded = words)
- ...

The noisy channel model allows us to decompose NLP systems into two parts:

- The translation model, which we need labeled data to estimate.
- The language model, which we need only *unlabeled* data to estimate.

Since there is always more unlabeled data, this means we can improve NLP systems just by improving $\text{p}_e(\boldsymbol{w})$.

8.1 N-gram language models

We began with the relative frequency estimator,

$$\text{p}(\textit{Computers are useless, they can only give you answers}) \tag{8.5}$$
$$= \frac{\text{count}(\textit{Computers are useless, they can only give you answers})}{\text{count}(\text{all sentences ever spoken})} \tag{8.6}$$

We'll define the probability of a sentence as the probability of the words (in order): $\text{p}(\boldsymbol{w}) = \text{p}(w_1, w_2, \ldots, w_M)$. We can apply the chain rule:

$$\begin{aligned}\text{p}(\boldsymbol{w}) =& \text{p}(w_1, w_2, \ldots, w_M) \\ =& \text{p}(w_1)\text{p}(w_2 \mid w_1)\text{p}(w_3 \mid w_2, w_1) \ldots \text{p}(w_M \mid w_{M-1}, \ldots, w_1)\end{aligned}$$

Each element in the product is the probability of a word given all its predecessors. We can think of this as a *word prediction* task: *Computers are [BLANK]*. The relative frequency estimate:

$$\text{p}(\textit{useless}|\textit{computers are}) = \frac{\text{count}(\textit{computers are useless})}{\sum_x \text{count}(\textit{computers are } x)} = \frac{\text{count}(\textit{computers are useless})}{\text{count}(\textit{computers are})}$$

Note that we haven't made any approximations yet, and we could have applied the chain rule in reverse order, $\text{p}(\boldsymbol{w}) = \text{p}(w_M)\text{p}(w_{M-1}|w_M)\ldots$, or in any other order. But this means that we also haven't really improved anything either: to compute the conditional probability $P(W_M \mid W_{M-1}, W_{M-2}, \ldots)$, we need to model $\#|\mathcal{V}|^{N-1}$, with $\#|\mathcal{V}|$ events. We can't even **store** this probability distribution, let alone reliably estimate it.

N-gram models

N-gram models make a simple approximation: condition on only the past $n-1$ words.

$$\text{p}(w_m \mid w_{m-1} \ldots w_1) \approx P(w_m \mid w_{m-1}, \ldots, w_{m-n+1})$$

This means that the probability of a sentence $\boldsymbol{w}$ can be computed as

$$\text{p}(w_1, \ldots, w_M) \approx \prod_m \text{p}(w_m \mid w_{m-1}, \ldots, w_{m-n+1})$$

- To compute the probability of a whole sentence, it's convenient to pad the beginning and end with special symbols $\diamond$ and $\square$. Then the bigram ($n = 2$) approximation to the probability of *I like black coffee* is:

$$\text{p}(I \mid \diamond)\text{p}(\textit{like} \mid I)\text{p}(\textit{black} \mid \textit{like})\text{p}(\textit{coffee} \mid \textit{black})\text{p}(\square \mid \textit{coffee}) \tag{8.7}$$

- In this model, we have to estimate and store the probability of only $\#|\mathcal{V}|^n$ events. A very common choice is a trigram model, in which $n = 3$.
- The n-gram probabilities can be determined by relative frequency estimation,

$$\text{p}(w|u, v) = \frac{\text{count}(u, v, w)}{\text{count}(u, v)} = \frac{\text{count}(u, v, w)}{\sum_{w'} \text{count}(u, v, w')} \tag{8.8}$$

There could be too problems with an n-gram language model:

- n **is too small**. In this case, we are missing important linguistic context. Consider the following sentences:
  - Gorillas *always like to groom* ***THEIR*** *friends.*
  - *The* computer *that's on the 3rd floor of our office building* ***CRASHED****.*

  The bolded words depend crucially on their predecessors in italics: *their* depends on knowing that *gorillas* is plural, and *crashed* depends on knowing that the subject is a *computer*. The resulting model would offer probabilities that are too low for these sentences, and too high for sentences that fail basic linguistic tests like number agreement.
- n **is too big**. In this case, we can't make good estimates of the n-gram parameters from our dataset. See the slides for some examples of this.
- These two problems point to another **bias/variance** tradeoff. Can you see how it works?
- In reality, we often have **both** problems! Language is full of long-range dependencies, and datasets are small.

We will seek approaches to keep n large, while still making low-variance estimates of the underlying parameters. To do this, we will introduce a different sort of bias: **smoothing**. But before we talk about that, let's consider how we can evaluate language models.

8.2 Evaluating language models

- Because language models are typically components of larger systems (language modeling is not really an application itself), we would prefer **extrinsic evaluation**: does the LM help the task (translation or whatever). But this is often hard to do, and depends on details of the overall system which may be irrelevant to language modeling.
- **Intrinsic evaluation** is task-neutral. Better performance on intrinsic metrics may be expected to improve extrinsic metrics across a variety of tasks (unless we are over-optimizing the intrinsic metric).

Held-out likelihood

A popular intrinsic metric is the **held-out likelihood**.

- We obtain a test corpus, and compute the (log) probability according to our model. It is crucial that the words in this corpus were not used in estimating the model itself.
- A good model should assign high probability to this held-out data.
- Specifically, we compute

$$\ell(\boldsymbol{w}) = \sum_i \sum_m \log \mathrm{p}(w_m^{(i)} | w_{m-1}^{(i)}, \ldots, w_{m-n+1}^{(i)}), \tag{8.9}$$

  for all sentences $\boldsymbol{w}^{(i)}$ in the held-out corpus.

Perplexity

Perplexity is a transformation of the held-out likelihood, into an information-theoretic quantity. Specifically, we compute

$$PP(\boldsymbol{w}) = 2^{-\frac{\ell(\boldsymbol{w})}{M}}, \tag{8.10}$$

where M is the total number of tokens in the held-out corpus.

- After this transformation, we now prefer lower values. In the limit, we obtain probability 1 for our held-out corpus, with $PP = 2^{-\log 1} = 1$.
- Assume a uniform, unigram model in which $P(s_i) = \frac{1}{V}$ for all V words in the vocabulary. Then,

$$\begin{aligned} PP(\boldsymbol{w}) &= \left[\left(\frac{1}{V} \right)^M \right]^{-\frac{1}{M}} \\ &= \left(\frac{1}{V} \right)^{-1} = V \end{aligned}$$

- We can think of perplexity as the *weighted branching factor* at each word in the sentence.
  - If we have solved the word prediction problem perfectly, $PP(\boldsymbol{w}) = 1$, because there is only one possible choice.

- If we have only a uniform model that assigns equal probability to every word, $PP(\boldsymbol{w}) = V$.
- Most models fall somewhere in between.
- Here's how you remember: lower perplexity is better, because you are less perplexed.

Example On 38M tokens of WSJ, $V \approx 20K$, (Jurafsky and Martin, 2009, page 97) obtain these perplexities on a 1.5M token test set.

- Unigram: 962
- Bigram: 170
- Trigram: 109

Will it keep going down? See slides from (Manning and Schütze, 1999).

Information theory*

Perplexity is very closely related to the concept of entropy, the expected value of the information contained in each word.

$$H(P) = -\sum_{\boldsymbol{w}} \mathrm{p}(\boldsymbol{w}) \log \mathrm{p}(\boldsymbol{w}) \tag{8.11}$$

The true entropy of English (or any real language) is unknown. Claude Shannon, one of the founders of information theory, wanted to compute upper and lower bounds. He would read passages of 15 characters to his wife, and ask her to guess the next character, recording the number of guesses it took for her to get the correct answer. As a fluent speaker of English, his wife could provide a reasonably tight bound on the number of guesses needed per character. **Question: is this an upper bound or a lower bound?**

Cross-entropy is a relationship between two probability distributions, the true one $P(W)$ and an estimate $Q(W)$.

$$\begin{aligned} H(P,Q) =& E_P[\log Q] \\ =& -\sum_{\boldsymbol{w}} \mathrm{p}(\boldsymbol{w}) \log q(\boldsymbol{w}) \\ =& \sum_{\boldsymbol{w}} \mathrm{p}(\boldsymbol{w}) \log \frac{\mathrm{p}(\boldsymbol{w})}{q(\boldsymbol{w})} - \mathrm{p}(\boldsymbol{w}) \log \mathrm{p}(\boldsymbol{w}) \\ =& D_{KL}(P||Q) + H(Q) \end{aligned}$$

So the cross-entropy is the KL-divergence between P and Q – a non-symmetric distance measure between distributions, which we will see again later in the course – plus the entropy of P. Since P is the language itself, we can only control Q, and minimizing the cross-entropy is equivalent to minimizing the KL-divergence.

We do not have access to the true $P(W)$, just a sequence $\boldsymbol{w} = \{w_1, w_2, \ldots, \}$, which is sampled from $P(W)$. In the limit, the length of $\boldsymbol{w}$ is infinite, so we have,

$$\begin{aligned} H(P,Q) =& -\sum_{\boldsymbol{w}} \mathrm{p}(\boldsymbol{w}) \log q(vw) \\ =& -\lim_{M \to \infty} \frac{1}{M} \log q(\boldsymbol{w}) \\ \approx& -\frac{1}{M} \log q(\boldsymbol{w}) \\ PP(S) =& 2^{-\frac{1}{M} \log q(vw)} \end{aligned}$$

A good language model has low cross-entropy with $P(W)$, and thus low perplexity.

Further aside : A related topic in psycholinguistics is the "constant entropy rate hypothesis," also called the "uniform information density hypothesis." The hypothesis is that speakers should prefer linguistic choices that convey a uniform amount of information over time (Jaeger, 2010). Some evidence:

- Speakers shorten predictable words, lengthen unpredictable ones
- High-entropy sentences take longer to read
- Syntactic reductions (e.g., *I'm* versus *I am*) are more likely when the reducible word contains less information.

8.3 Smoothing and discounting

We want to estimate $P(W)$ from sparse statistics, avoiding $\text{p}(\boldsymbol{w}) = 0$.

Laplace/Lidstone smoothing

Simplest idea: just add "pseudo-counts"

$$\text{p}_{\text{Laplace}}(w \mid v) = \frac{\text{count}(v, w) + \alpha}{\sum_{w'} \text{count}(v, w') + V\alpha} \tag{8.12}$$

Anything that we add to the numerator (α) must also appear in the dominator ($V\alpha$). We can capture this with the concept of **effective counts**:

$$c_i^* = (c_i + \alpha)\frac{N}{N + V\alpha}$$

The **discount** for each n-gram is:

$$d_i = \frac{c_i^*}{c_i} = \frac{(c_i + \alpha)}{c_i}\frac{N}{(N + \alpha)}$$

- In general, this is called Lidstone smoothing
- When $\alpha = 1$, we are doing Laplace smoothing
- When $\alpha = 0.5$, we are following Jeffreys-Perks law
- Manning and Schütze (1999) offer more insight on the justifications for Jeffreys-Perks smoothing

Discounting and backoff

Discounting "borrows" probability mass from observed n-grams and redistributes it.

- In Lidstone smoothing, we borrow probability mass by increasing the denominator of the relative frequency estimates, and redistribute it by increasing the numerator for all n-grams.
- Instead, we could borrow the same amount of probability mass from all observed counts, and redistribute it among only the unobserved counts. This is called **absolute discounting.**

- For example, if we set an absolute discount $d = 0.1$ in a trigram model, we get: $\text{p}(w|\textit{denied the}) =$

word	counts c	effective counts c^*	unsmoothed probability	smoothed probability
allegations	3	2.9	0.429	0.414
reports	2	1.9	0.286	0.271
claims	1	0.9	0.143	0.129
request	1	0.9	0.143	0.129
charges	0	0.2	0.000	0.029
benefits	0	0.2	0.000	0.029
...				

- We need not redistribute the probability mass equally. Instead, we can **back-off** to a lower-order language model.
- In other words: if you have trigrams, use trigrams; if you don't have trigrams, use bigrams; if you don't even have bigrams, use unigrams. (And what if you don't even have unigrams?). This is called **Katz backoff**.

$$c^*(u,v) = c(u,v) - d$$

$$\text{p}_{\text{backoff}}(v \mid u) = \begin{cases} \frac{c^*(u,v)}{c(u)} & \text{if } c(u,v) > 0 \\ \alpha(u) \times \frac{\text{p}_{\text{backoff}}(v)}{\sum_{v':c(u,v')=0} \text{p}_{\text{backoff}}(v')} & \text{if } c(u,v) = 0 \end{cases}$$

Typically we can set d to minimize perplexity on a development set.

Interpolation

An alternative to this discounting scheme is to do interpolation: the probability of a word in context is a weighted sum of its probabilities across progressively shorter contexts.

Instead of choosing a single n-gram order, we can take the weighted average:

$$\begin{aligned} \text{p}_{\text{Interpolation}}(w|u,v) = &\lambda_1 \text{p}_1^*(w|u,v) \\ &+ \lambda_2 \text{p}_2^*(w|u) \\ &+ \lambda_3 \text{p}_1^*(w) \end{aligned}$$

- p_k^* is the maximum likelihood estimate (MLE) of a k-gram model
- Constraint: $\sum_z \lambda_z = 1$
- We can tune λ on heldout data...
- Or we can use **expectation maximization**!

EM for interpolation We can add a latent variable z_m, indicating the order of the n-gram that generated word w_m. Generative story:

- For each word m
  - Draw $z_m \sim \text{Categorical}(\lambda(w_m))$
  - Draw $w_m \sim \text{p}^*_{z_m}(w_m | s_{m-1}, \ldots s_{m-z_m})$

As always we have two quantities of interest in our EM application:

- The parameters, λ.
- Our beliefs about the latent variables. Let $q_m(z)$ be our degree of belief that word token w_m was generated from a n-gram of order z.

Having defined these quantities, we can derive EM updates:

- **E-step**: $q_m(z) = \text{p}(z | w_{1:m}) = \frac{\text{p}^*_z(w_m | w_{m-1}, \ldots, w_{m-z+1})}{\sum_{z'} \text{p}^*_{z'}(w_m | w_{m-1}, \ldots w_{m-z'+1})} \text{p}(z' | \lambda(w_m))$
- **M-step**: $\lambda(w)_z = \frac{E_q[\text{count}(W{=}w, Z{=}z)]}{\sum_{z'} E_q[\text{count}(W{=}w, Z{=}z')]}$

By running the EM algorithm, we can obtain a good estimate of λ, which we can then use for unseen data. It should be clear how we can extend this approach to trigrams and beyond; Collins (2013) offers more details.

Kneser-ney smoothing

Kneser-ney smoothing also incorporates discounting, but redistributes the resulting probability mass in a different way. Consider the example:
I recently visited

- *Francisco*?
- *Duluth*?

Key idea: some words are more **versatile** than others.

- Suppose $\text{p}^*(\textit{Francisco}) > \text{p}^*(\textit{Duluth})$, and $c(\textit{visited Francisco}) = c(\textit{visited Duluth}) = 0$.
- We would still guess that $\text{p}(\textit{visited Duluth}) > P(\textit{visited Francisco})$, because *Duluth* is a more versatile word.

We define the Kneser-Ney bigram probability as

$$\text{p}_{KN}(v|u) = \begin{cases} \frac{\text{count}(u,v)-d}{\text{count}(u)}, & \text{count}(u,v) > 0 \\ \alpha(u)\text{p}_{\text{continuation}}(v), & \text{otherwise} \end{cases}$$
$$\text{p}_{\text{continuation}}(v) = \frac{\#|u : \text{count}(u,v) > 0|}{\sum_{v'} \#|u' : \text{count}(u',v') > 0|}$$

- We reserve probability mass using absolute discounting d.
- The *continuation probability* $\text{p}_{\text{continuation}}(u)$ is proportional to the number of observed contexts in which u appears.
- As in Katz backoff, $\alpha(v)$ makes the probabilities sum to 1
- In practice, interpolation works a little better than backoff

$$\text{p}_{KN}(v|u) = \frac{\text{count}(u,v) - d}{\text{count}(u)} + \lambda(u)\text{p}_{\text{continuation}}(v) \tag{8.13}$$

- This idea of counting contexts may seem heuristic, but actually there is a cool justification from Bayesian nonparametrics (Teh, 2006).

8.4 Other types of Language Models

Interpolated Kneser-Ney is pretty close to state-of-the-art. But there are some interesting other types of language models, and they apply ideas that we have already learned.

Mixed-order n-gram models

Saul and Pereira (1997) described a "mixed-order" n-gram model, where you condition on multiple bigram contexts, skipping over intermediate words:

$$\mathrm{p}(w_m|w_{m-1},\ldots,w_{m-n+1}) = \sum_k \lambda_k(w_{m-k})\tilde{\mathrm{p}}(w_m|w_m-k)\prod_j^{k-1}[1-\lambda_j(w_{m-j})] \quad (8.14)$$

- This is an **interpolated** model, because we are taking the weighted average over a bunch of bigram probabilities.
- Note that the interpolation weight depends on the context word, $\lambda_k(w_{m-k})$. This means that some words can prefer certain dependency lengths — for example, adjectives might prefer short dependencies, since they tend to affect adjacent nouns, while verbs might prefer longer dependencies, since they can affect indirect objects that are further away.
- The final product ensures that the weights in any particular context must add up to one: each λ_k is taking a slice of the probability mass that has already been used by the earlier contexts $j < k$.
- The parameters $\lambda_k(w)$ can be estimated by expectation maximization, just like in the interpolated N-gram model above.

Class-based language models

The reason we need smoothing is because the trigram probability model $\mathrm{p}(w|u,v)$ has a huge number of parameters. Let's simplify:

$$\mathrm{p}_{\text{class}}(w|v) = \sum_z P(w|z;\theta)P(z|v;\phi),$$

where $z \in [1,K], K \ll V$.

We get a bigram probability using $2VK$ parameters instead of V^2.

We could use EM to estimate θ and ϕ (Saul and Pereira, 1997).

- The latent variable is the class z, so the e-step updates $q_m(z)$
- The parameters are θ and ϕ, which can be updated in the M-step.

But this is usually too slow, so there are approximate algorithms, like "exchange clustering" (Brown et al 1992), which assigns each word type to a single class.

Discriminative language models

- Or we could just train a model to predict $\text{p}(w_m|w_{m-1}, w_{m-2}, \ldots)$ directly.
- We might be able to use arbitrary features of the history to model long-range dependencies.
- Algorithms such as perceptron and logistic regression have been considered (Rosenfeld, 1996; Roark et al., 2007)
- Currently, "neural probabilistic language models" are attracting a lot of interest. The log-bilinear model (Mnih and Hinton, 2008) looks like this:

$$\text{p}_\theta^h(w) = \frac{\exp(s_\theta(w,h))}{\sum_{w'} \exp(s_\theta(w',h))}$$
$$s_\theta(w,h) = \hat{\boldsymbol{q}}_h^\intercal \boldsymbol{q}_w + b_w,$$

  where h is the history context, $\hat{\boldsymbol{q}}_h$ is a latent description of the history, $\boldsymbol{q}_w$ is a latent description of the word, and b_w is an offset. The history context can be computed from the words themselves, as $\hat{\boldsymbol{q}}_h = \sum_i^{m-1} C_i \boldsymbol{q}_i$, where the matrix C_i is applied to context position i. All parameters can be estimated to directly maximize the probability of a corpus, using gradient ascent.
- Recent work has focused on efficiently training such models, with increasingly convincing results on large training sets (Mikolov et al., 2011).

Chapter 9

Morphology

[1] So far we have been focusing on NLP at the word level. Today we go **inside of words**.

We've already hinted at a morphological problem by introducing the idea of **lemmas**, where *serve/served/serving* all have the lemma *serve*.

From the perspective of document classification, these multiple forms may just seem like an annoyance, which we can get rid of by lemmatization or stemming (more on this later).

But morphology conveys information which might be crucial for some applications:

- Information retrieval
  - With a query like *bagel*, we want to get hits for *bagels*.
  - Same for *corpus/corpora*, *goose/geese*.
  - But we don't always want all the inflected forms. For example, a query for *Apple* may not want hits for *apples*
- Time. Morphology often indicates when events happen. For example, in French:

J'achete un velo	I buy a bicycle (now)
J'acheterai un velo	I will buy a bicycle
J'achetais un velo	I was buying a bicycle
J'ai acheté un velo	I bought a bicycle
J'acheterais un velo	I would buy a bicycle

[1] This chapter is pretty rough; better to see Chapter 2 of (Bender, 2013).

- Causality. Consider the difference between the Spanish examples:

*Si tu vas a GT, tu **seras** rica*	If you go to GT, you will be rich
*Si tu vas a GT, tu **eres** rica.*	If you go to GT, you are rich

- Lexical semantics: suppose *antichrist* is not in your sentiment dictionary. Do you think it is a positive or negative word?

In addition to recognizing morphology, there are applications in which we need to produce it.

- Translation: *you (pl) are smart* → *Ustedes son inteligentes* vs *Tu eres inteligente*
- Text generation: `(has-property you-pl smart)` → *ustedes son inteligentes*

Morphology, Orthography, and Phonology

- **Morphology** describes how meaning is constructed from combining affixes. For example, it is a morphological fact of English that adding the affix *+S* to many nouns creates a plural.

$$\textit{berry} + \textsc{Plural} \rightarrow \textit{berry+s}$$

- **Orthography** specifically relates to writing.
  For example,

$$\textit{berry+s} \rightarrow \textit{berries}$$

  is an orthographic rule. We have lots of these in English, which is one reason English spelling is difficult.

- Morphological rules also include stem changes, such as *goose* + PLURAL → *geese*.

- **Phonology** describes how sounds combine. For example, the different pronunciations of the final *s* in *cats* (s) and *dogs* (z) follow from a phonological rule (example (25) in the Bender text, page 30).

- In English, morphologically distinct words may be pronounced differently even when they are spelled the same, and this can reflect morphological differences. *read*+PRESENT vs. *read*+PAST.

- Conversely, morphological variants may be spelled differently even when they sound the same, like *The Champions' league* vs *The Champion's league* vs *The Champions league*.

Productivity

One idea for dealing with morphology is to build a morphologically aware dictionary:

- Map each **surface form** to its underlying **lemma**
- Include meaning of morphology: tense, number, animacy, posession, etc.
- Then when you encounter a surface form, just look it up.

duck	*duck*/N+SG
ducks	*duck*/N+PL
duck	*duck*/VB+PRESENT
ducks	*duck*/VBZ+PRESENT

Will this work? Besides the problem of ambiguity, still another problem is that morphology is **productive**, meaning that it applies to new words. If you only know the words *Google* or *iPad*, you can immediately understand their inflected forms.

- Have you Googled that yet?
- I have owned three iPads.

Derivational morphology (more on this later) is productive in another way: you can produce new words by applying morphological changes to existing words. hyper+un+desire+able+ity

In some languages, derivational morphology can create extremely complicated words. The J&M textbook has a fun example from Turkish:

In the homework, you'll see examples from Swahili, which also has complex morphology. A dictionary of all possible surface forms in such languages would be gargantuan. So instead of building a static dictionary, we will try to model the underlying morphological and orthographic rules.

9.1 Morphemes

Two broad classes: **stems** and **affixes**.

- Intuitively, stems are the "main" part of meaning, affixes are the modifiers

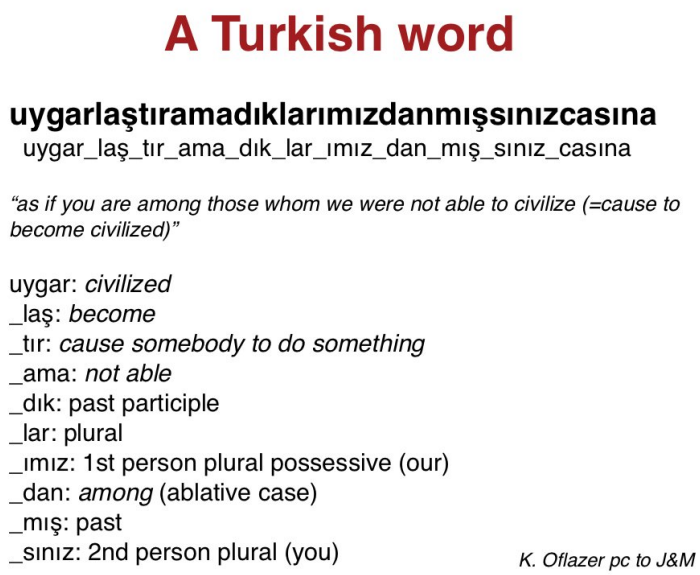


Figure 9.1: From (Jurafsky and Martin, 2009)

- Typically, **stems** can appear on their own (they are **free**) and affixes cannot (they are **bound**).
- Types of affixes:
  - **Prefixes**: *un+learn*, *pre+view*.
    - These examples are derivational. English has few inflectional prefixes, but other languages have many.
    - For example, in Swahili: *u-na-kata* versus *u-me-kata* distinguishes *you are cutting* versus *you have cut*. *na* and *me* are prefixes, *kata* is the root.
  - **Suffixes**: *I learn+ed*, *She learn+s*, *three apple+s*, *four fox+es*.
  - **Circumfixes** go around the stem.
    - None in English.
    - German has a circumfix for the past participle: *sagen* (say) → *ge+sag+t* (said)
    - French negation can be seen as a circumfix: *Je mange+NEG→Je ne mange pas*. (I do not eat).

* More generally, morphemes can be non-continuous, as in the Hebrew example (7) in the Bender reading (page 12).

(7)

Root	Pattern	Part of Speech	Phonological Form	Orthographic Form	Gloss
ktb	CaCaC	(v)	katav	כתב	'wrote'
ktb	hiCCiC	(v)	hixtiv	הכתיב	'dictated'
ktb	miCCaC	(n)	mixtav	מכתב	'a letter'
ktb	CCaC	(n)	ktav	כתב	'writing, alphabet'

[heb]

In this example, the root *ktb* (related to writing) is combined with patterns that indicate where to insert vowels to produce different parts-of-speech and meanings.

– **Infixes** go inside the stem.
  * Tagalog: *hingi+AGENT→h+um+ingi*
  * Lakhota: *m'ani* (he walks), *ma-wá-ni* (I walk). The *wá* marks agreement with a first-person singular subject; it is an infix for this word, although it is a prefix in other words.
  * English: *absolutely+fucking→absofuckinglutely*, but **absfuckingsolutely* arguably doesn't work.

9.2 Types of morphology

- **Inflection** creates different forms of a single word:
  – tense: *to be, being, I am, you are, he is, I was*
  – number: *book, books*
  – case: *he, his, her, they, them, their*
- **Derivation** creates new words:
  grace → disgrace → disgraceful → disgracefully
- **Cliticization** combines *Georgia+'s* into *Georgia's*; the possessive clitic *'s* is syntactically independent but phonologically dependent.

  - Pronouns appear as clitics in French, e.g., *j'accuse* (I accuse), as does negation *Je n'accuse personne* (I don't accuse anyone).
  - Another example is from Hebrew: *l'shana tova* (literally for year good, meaning happy new year); the preposition *for* appears as a clitic.

- **Compounding** combines two words in a new word:
  cream → *ice cream* → *ice cream cone* → *ice cream cone bakery*

- **Portmanteaus** combine words, truncating one or both.
  smoke + *fog* → *smog*
  glass + *asshole* → *glasshole*

- Word formation is *productive*: new words are subject to all of these processes

Inflectional morphology

Inflectional morphology adds information about words. English has a very simple system of inflectional morphology, compared to many languages.

Affix	Syntactic/semantic effect	Examples
-s	NUMBER: plural	*cats*
-'s	possessive	*cat's*
-s	TENSE: present, SUBJ: 3sg	*jumps*
-ed	TENSE: past	*jumped*
-ed/-en	ASPECT: perfective	*eaten*
-ing	ASPECT: progressive	*jumping*
-er	comparative	*smaller*
-est	superlative	*smallest*

Figure 9.2: From (Bender, 2013)

- English nouns are marked for number and possession; many language also mark nouns for **case**, which is the syntactic role that the noun plays in the sentence.

- In English, we do distinguish the case of some pronouns:
  - *He* (NOMINATIVE) *gave her* (OBLIQUE) *his* (GENITIVE) *guitar.*
  - *She gave him her guitar.*
  - *I gave you our guitar.*
  - *You gave me your guitar.*

  Specifically, we distinguish the **nominative** case of personal pronouns (except for 2nd-person), and the **genitive** case; all other uses are the **oblique** case.
- Other languages – such as Latin, Russian, Sanskrit, and Tamil, mark the case of all nouns. These languages have additional cases, such as dative (indirect object), accusative (direct object), and vocative (address).
- In German, noun is not inflected for case, but the articles and adjectives are:
  - *Der alte Mann gab dem kleinen Affen die grosse Banane* (based on example 49 from Bender)
  - The old man (NOM) gave the little monkey (DATIVE) the big banana (ACCUSATIVE)
  - Notice how *der*, *dem*, and *die* all mean the, but carry the case marking.
  - The adjectives are also marked for case.
- Many languages – such as Romance languages – mark the gender and number of nouns by inflecting the article and adjective. e.g., Spanish:
  - *El coche rojo pasó la luz roja*: the red car ran the red light
  - *Los coches rojos pasó las luces rojas*: the red cars ran the red lights
  - Note that the article and adjective must **agree** for the sentence to be grammatical.
  - In English, demonstrative determiners mark number, *this book* vs *these books*. In English, the determiner and noun must agree, e.g. **this books*.
- Gender is not necessarily binary.
  - English pronouns include neuter *it*; German, Sanskrit, and Latin do this for all nouns.
  - Danish and Dutch distinguish **neuter** from **common** gender
  - Other languages distinguish **animate** and **inanimate**

  - Number if not necessarily binary.
    * Many languages, such as Arabic and Sanskrit, include a special **dual** number for two. English has residual traces of the dual number, with *both* vs *all* and *either* vs *any*.
    * Some Austronesian languages have a **trial** number, for groups of three.
    * Some languages, including Arabic, have a **paucal** number, for small groups.

- English verbs are inflected for tense and number distinguishing past (*I ate*), present (*I eat*), and 3rd-person singular (*She eats*). They are also inflected for aspect, distinguishing perfective (*I had eaten*) and progressive (*I am eating*). Note that the perfective and the past tense are identical for regular verbs, e.g. *we had talked*, *we talked*.

  - Many languages (e.g., Chinese and Indonesian), do not mark tense with morphology. Indonesian uses time signals.

Saya makan apel	I eat an apple
Saya sedang makan apel	I am eating an apple
Saya telah makan apel	I already ate an apple
Saya akan makan apel	I will eat an apple

  - Romance languages distinguish many more tenses than English with morphology.
    * Spanish has multiple past tenses: **preterite** and **imperfect**, distinguishing, e.g. *I ate onions yesterday* from *I ate onions every day*. These are distinguished by morphology: *comí cebollas ayer*, *comía cebollas cada día*.
    * Spanish and French have endings for conditional (*comería cebollas*) and future (*comeré cebollas*)
    * All of these are marked with time signals in English; future can also be marked this way in French and Spanish, e.g. *voy a comer cebollas*.
  - Romance also have separate forms for every combination of number and person.
    * *(yo hablo / tu hablas / ella habla / nosotros hablamos / vosotros hableis / ellas hablan)*
    * *(je parle / tu parles / elle parle / nous parlons / vous parlez / ils parlent)*

    - In Spanish, they they eliminate pronouns (pro-drop) in cases where the morphology makes it clear (unless they want to add emphasis). Chinese is also a pro-drop language?
    - This doesn't happen in French, maybe because many different spellings (*parle/parles/parlent*) sound the same.
    - In English, we only distiniguish 3rd-person singular.

- Adjectives in English mark comparative and superlative (*taller*, *tallest*). As we have seen, they can mark gender and number in other languages.

- Other things can be marked with affixes, such as **evidentiality** – how the speaker came to know the information. In Eastern Pomo (a California language), there are verb suffixes for four evidential categories:

-ink'e	nonvisual sensory
-ine	inferential
-le	hearsay
-ya	direct knowledge

  Example (41) from Bender (2013) shows evidentiary marking in Turkish, *Ahmet geldi* (Ahmet came, witnessed by the speaker) vs *Ahmet gelmi s* (not witnessed by the speaker)

The **index of synthesis** measures the ratio of the number of morphemes in a given text to the number of words. Languages with complex morphology are called **synthetic**; languages with simple morphology are called **isolating** or **analytic**. English is relatively, but not extremely, analytic.

An approximation of the index of synthesis is the type-token ratio. Can you see why? If you count the number of unique surface forms in 10K *parallel* sentences from Europarl, you get:

- English: 16k word types
- French: 22k
- German: 32k
- Finnish: 55k

Language	Index of synthesis
Vietnamese	1.06
Yoruba	1.09
English	1.68
Old English	2.12
Swahili	2.55
Turkish	2.86
Russian	3.33
Inuit (Eskimo)	3.72

Figure 9.3: From (Bender, 2013)

Derivational Morphology

Derivational morphology is a way to create new words and change part-of-speech.

- **nominalization**
  - *V + -ation: computerization*
  - *V + -er: walker*
  - *Adj + -ness: fussiness*
  - *Adj + -ity: obesity*
- **negation**: *undo, unseen, misnomer*
- **adjectivization**: *V + -able : doable, thinkable, N + -al : tonal, national, N + -ous: famous, glamorous*
- **abverbization**: *ADJ + -ily: clumsily*
- **lots more**: *rewrite, phallocentrism, ...*

You can create totally new words this way.
word $\rightarrow$ *wordify* $\rightarrow$ *wordification* $\rightarrow$ *wordificationism* $\rightarrow$ *antiwordificiationism* $\rightarrow$ *hyperantiwordificationism*

Irregularities

English morphology contains a lot of irregularities: *know/knew/known*, *foot/feet*, *go/went*.. if you're not a native speaker, learning these was probably a pain in the neck.

- the good news is, there are fewer of these all the time! for example, the past tense of *show* used to be *shew*, like *know/knew* (the past participle is still *shown*).
- the bad news is, the most common words will be the last to change (if ever).

Attaching affixes can cause orthographic and phonological changes:

- walk + ed = walked, but frame+ed = framed, emit+ed = emitted, easy + ier = easier
- this is usually due to phonetic or orthographic *constraints*
- *optimality theory* is an approach to systematizing such interacting constraints. There's a lot of research on finite state models of optimality theory, but you'll have to take a linguistics course for that Karttunen and Beesley (2005).

Chapter 10

Finite-state automata

Finite-state automata are a powerful formalism for representing a subset of formal languages, the **regular** languages. As we will see, this formalism can also be used as a building block for an incredibly wide range of methods for manipulating natural language too (Mohri et al., 2002). This chapter will especially focus on **morphology**, which concerns how words are built out of smaller units. For a good reference on morphology for natural language processing, see (Bender, 2013).

Knight and May (2009) show how finite-state automata can be composed together to create impressive applications. They start with one such application — transliteration — and explain how it works. Here, we'll build the formalism from the ground up, starting with finite-state acceptors, then adding weights, and then adding transduction, finally arriving at the same sorts of applications.

10.1 Automata and languages

Basics of the formalism :

- An alphabet Σ is a set of symbols
- A string ω is a sequence of symbols.
  The empty string ϵ contains zero symbols.
- A language $L \subseteq \Sigma^*$ is a set of strings.

An automaton is an abstract model of a computer which reads an input string, and either accepts or rejects it.

Chomsky Hierarchy Every automaton defines a language. Different automata define different classes of languages. The Chomsky Hierarchy:

- Finite-state automata define **regular** languages
- Pushdown automata define **context-free** languages
- Turing machines define **recursively-enumerable** languages

Finite-state automata A finite-state automaton $M = \langle Q, \Sigma, q_0, F, \delta \rangle$ consists of:

- A finite set of states $Q = \{q_0, q_1, \ldots, q_n\}$
- A finite alphabet Σ of input symbols
- A start state $q_0 \in Q$
- A set of final states $F \subseteq Q$
- A transition function δ

Determinism

- In a deterministic (D)FSA, $\delta : Q \times \Sigma \rightarrow Q$.
- In a nondeterministic (N)FSA, $\delta : Q \times \Sigma \rightarrow 2^Q$
- We can determinize any NFSA using the powerset construction, but the number of states in the resulting DFSA may be 2^n.
- Any **regular expression** can be converted into an NFSA, and thus into a DFSA.

The English Dictionary as an FSA We can build a simple "chain" FSA which accepts any single word. So, we can define the English dictionary with an FSA. However, we can make this FSA much more compact. (see slides)

- Begin by taking the **union** of all of the chain FSAs by defining epsilon transitions (that is, transitions which do not consume an input symbol) from the start state to chain FSAs for each word (5303 states / 5302 arcs using a 850 word dictionary of "basic English")

- Eliminate the epsilon transitions by pushing the first letter to the front (4454 states / 4453 arcs)
- **Determinize** (2609 / 2608)
- **Minimize** (744 / 1535). The cost of minimizing an acyclic FSA is $O(E)$. This data structure is called a trie.

Operations We've now talked about three operations: union, determinization and minimization. Other important operations are:

intersection : only accept strings in both FSAs

negation only accept strings not accepted by FSA M

concatenation . accept strings of the form $s = [s_1 s_2]$, where $s_1 \in M_1$ and $s_2 \in M_2$

FSAs are closed under all these operations, meaning that resulting automaton is still an FSA (and therefore still defines a regular language).

10.2 FSAs for Morphology

Now for some morphology. Suppose that we want to write a program that accepts words that could **possibly** be constructed in accordance with English derivational morphology, but none of the impossible ones:

- *grace, graceful, gracefully*
- *disgrace, disgraceful, disgracefully, ...*
- *Google,Googler,Googleology,...*
- **gracelyful*, **disungracefully*, ...

We could just make a list, and then take the union of the list using ϵ-transitions.

The list would get very long, and it would not account for productivity (our ability to make new words like *antiwordificationist*). So let's try to use finite state machines instead. Our FSA will have to encode rules about morpheme ordering, called *morphotactics*.

Let's start with some examples:

- *grace*: $q_0 \rightarrow_{\text{stem}} q_1$

- *dis-grace*: $q_0 \rightarrow_{\text{prefix}} q_1 \rightarrow_{\text{stem}} q_2$
- *grace-ful*: $q_0 \rightarrow_{\text{stem}} q_1 \rightarrow_{\text{suffix}} q_2$
- *dis-grace-ful*: $q_0 \rightarrow_{\text{prefix}} q_1 \rightarrow_{\text{stem}} q_2 \rightarrow_{\text{suffix}} q_3$

Can we generalize these examples?

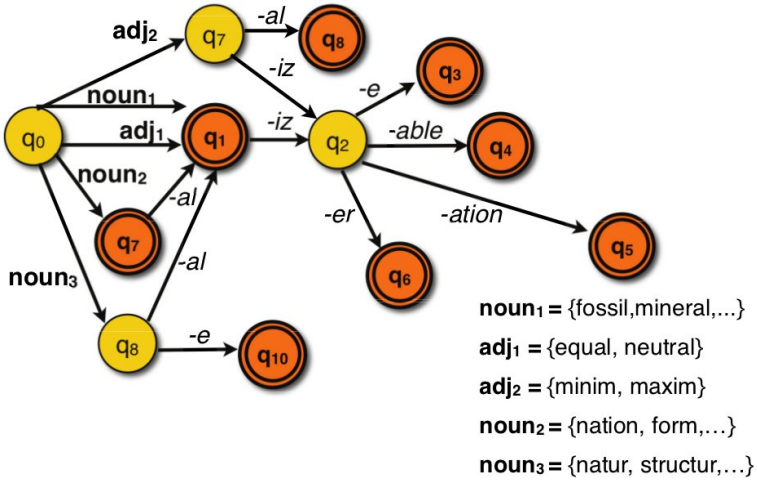


Figure 10.1: I can't find the attribution for this figure right now, sorry! I think it's either from Julia Hockenmaier's slides, or from Jurafsky and Martin (2009).

- This example abstracts away important details, like why *wordificate* is preferred to **wordifycate*. But this rule is part of English **orthography** (spelling), not **morphology**. "Two-level morphology" is an approach to integrating such orthographic transformations in a finite-state framework (Karttunen and Beesley, 2001).
- It also misses a key point: sometimes we have choices, and not all choices are considered to be equally good by fluent speakers.
  - Google counts:
    - *superfast*: 70M; *ultrafast*: 16M; *hyperfast*: 350K; *megafast*: 87K
    - *suckitude*: 426K; *suckiness*: 378K
    - *nonobvious*: 1.1M; *unobvious*: 826K; *disobvious*: 5K

- Rather than asking whether a word is **acceptable**, we might like to ask how acceptable it is.
- But finite state acceptors gives us no way to express *preferences* among technically valid choices.
- We'll need to augment the formalism for this.

10.3 Weighted Finite State Automata

A weighted finite-state automaton $M = \langle Q, \Sigma, \pi, \xi, \delta \rangle$ consists of:

- A finite set of states $Q = \{q_0, q_1, \ldots, q_n\}$
- A finite alphabet Σ of input symbols
- Initial weight function, $\pi : Q \rightarrow \mathbb{R}$
- Final weight function $\xi : Q \rightarrow \mathbb{R}$
- A transition function $\delta : Q \times \Sigma \times Q \rightarrow \mathbb{R}$

We have added a weight function that scores every possible transition.

- We can score any path through the WFSA by the sum of the weights.
- Arcs that we don't draw have infinite cost.
- The shortest-path algorithm can find the minimum-cost path for accepting a given string in $O(V \log V + E)$.

Applications of WFSAs

We can use WFSAs to score derivational morphology as suggested above. But let's start with a simpler example:

Edit distance . We can build an edit distance machine for any word. Here's one way to do this (there are others):

- Charge 0 for "correct" symbols and rightward moves
- Charge 1 for self-transitions (insertions)

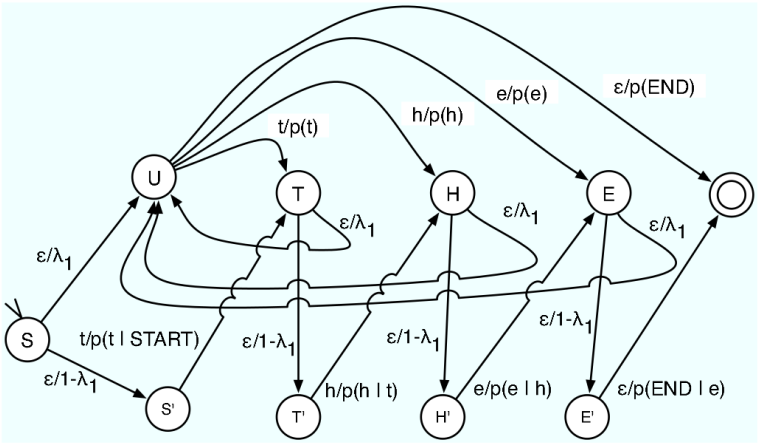


Figure 10.2: From (Knight and May, 2009)

- Charge 1 for rightward epsilon transitions (deletions)
- Charge 2 for "incorrect" symbols and rightward moves (substitutions)
- Charge ∞ for everything else

The total edit distance is the *sum* of costs across the best path through machine.

Probabilistic models For probabilistic models, we make the path costs equal to the likelihood:

$$\delta(q_1, s, q_2) = \text{p}(s, q_2 \mid q_1) \tag{10.1}$$

This enables probabilistic models, such as N-gram language models.

- A unigram language model is just one state, with V edges.
- A bigram language model will have V states, with V^2 edges.

Knight and May (2009) show how to do an interpolated bigram/unigram language model using a WFSA. (Last year I wrote a note that I had found a better way, with only $V + 3$ states rather than $2V + 4$. But now I can't find my solution!)

- Recall that an interpolated bigram language model is

$$\hat{\text{p}}(v|u) = \lambda \text{p}_2(v|u) + (1 - \lambda)\text{p}_1(v), \tag{10.2}$$

with $\hat{\text{p}}$ indicating the interpolated probability, p_2 indicating the bigram probability, and p_1 indicating the unigram probability.

- Unlike the basic n-gram language models, our interpolated model has non-determinism: do we choose the bigram context or the unigram context?
- What should happen to the scores as we encounter a non-deterministic choice?
- For a sequence *a,b,a*, we want the final path score to be

$$\begin{aligned}\psi(a,b,a) =&(\lambda \text{p}_2(a|*) + (1-\lambda)\text{p}_1(a)) \\ &\times (\lambda \text{p}_2(b|a) + (1-\lambda)\text{p}_1(a)) \\ &\times (\lambda \text{p}_2(b|a) + (1-\lambda)\text{p}_1(b))\end{aligned}$$

- So we could multiply along each step, and add probabilities across non-deterministic choices.
- With log-probabilities, we would add along each step, and use the log sum, $\log(e^a + e^b)$, to compute the score for non-deterministic branchings.

10.4 Semirings

We have now seen three examples: an acceptor for derivational morphology, and weighted acceptors for edit distance and language modeling. Several things are different across these examples.

- Scoring
  - In the derivational morphology FSA, we wanted a boolean "score": is the input a valid word or not?
  - In the edit distance WFSA, we wanted a numerical (integer) score, with lower being better.
  - In the interpolated language model, we wanted a numerical (real) score, with higher being better.
- Nondeterminism
  - In the derivational morphology FSA, we accept if there is any path to a terminating state.

  - In the edit distance WFSA, we want the score of the single best path.
  - In the interpolated language model, we want to sum over non-deterministic choices.

- How can we combine all of these possibilities into a single formalism? The answer is semiring notation.

Formal definition

A semiring is a system $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$

- $\mathbb{K}$ is the set of possible values, e.g. $\{\mathbb{R}_+ \cup \infty\}$, the non-negative reals union with infinity
- $\oplus$ is an addition operator
- $\otimes$ is a multiplication operator
- $\overline{0}$ is the additive identity
- $\overline{1}$ is the multiplicative identity

A semiring must meet the following requirements:

- $(a \oplus b) \oplus c = a \oplus (b \oplus c)$, $(\overline{0} \oplus a) = a$, $a \oplus b = b \oplus a$
- $(a \otimes b) \otimes c = a \otimes (b \otimes c)$, $a \otimes \overline{1} = \overline{1} \otimes a = a$
- $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$, $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- $a \otimes \overline{0} = 0 \otimes \overline{a} = \overline{0}$

Semirings of interest :
where $\oplus_{\log}(a, b)$ is defined as $\log(e^a + e^b)$.
Semirings allow us to compute a more general notion of the "shortest path" for a WFSA.

- Our initial score is $\overline{1}$
- When we take a step, we use $\otimes$ to combine the score for the step with the running total.
- When nondeterminism lets us take multiple possible steps, we combine their scores using $\oplus$.

Name	$\mathbb{K}$	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$	Applications
Boolean	$\{0, 1\}$	$\vee$	$\wedge$	0	1	identical to an unweighted FSA
Probability	$\mathbb{R}_+$	+	$\times$	0	1	sum of probabilities of all paths
Log-probability	$\mathbb{R} \cup -\infty \cup \infty$	$\oplus_{\log}$	+	$-\infty$	0	log marginal probability
Tropical	$\mathbb{R} \cup -\infty \cup \infty$	min	+	∞	0	best single path

Example Let's see how this works out for our language model example.

$$\begin{aligned} score(\{a, b, a\}) = \overline{1} \otimes &\big(\lambda \otimes \mathrm{p}_2(a|*) \oplus (1-\lambda) \otimes \mathrm{p}_1(a)\big) \\ &\otimes \big(\lambda \otimes \mathrm{p}_2(b|a) \oplus (1-\lambda) \otimes \mathrm{p}_1(b)\big) \\ &\otimes \big(\lambda \otimes \mathrm{p}_2(a|b) \oplus (1-\lambda) \otimes \mathrm{p}_1(a)\big) \end{aligned}$$

Now if we plug in the **probability semiring**, we get

$$\begin{aligned} score(\{a, b, a\}) = 1 \times &\big(\lambda \mathrm{p}_2(a|*) + (1-\lambda)\mathrm{p}_1(a)\big) \\ &\times \big(\lambda \mathrm{p}_2(b|a) + (1-\lambda)\mathrm{p}_1(b)\big) \\ &\times \big(\lambda \mathrm{p}_2(a|b) + (1-\lambda)\mathrm{p}_1(a)\big) \end{aligned}$$

But if we plug in the **log probability semiring**, we need the edge weights to be equal to $\log \mathrm{p}_1$, $\log \mathrm{p}_2$, $\log \lambda$, and $\log(1-\lambda)$. Then we get:

$$\begin{aligned} score(\{a, b, a\}) &= 0 + \log\big(\exp(\log\lambda + \log \mathrm{p}_2(a|*)) + \exp(\log(1-\lambda) + \log \mathrm{p}_1(a))\big) \\ &\quad + \log\big(\exp(\log\lambda + \log \mathrm{p}_2(b|a)) + \exp(\log(1-\lambda) + \log \mathrm{p}_1(b))\big) \\ &\quad + \log\big(\exp(\log\lambda + \log \mathrm{p}_2(a|b)) + \exp(\log(1-\lambda) + \log \mathrm{p}_1(a))\big) \\ &= 0 + \log(\lambda \mathrm{p}_2(a|*) + (1-\lambda)\mathrm{p}_1(a)) \\ &\quad + \log(\lambda \mathrm{p}_2(b|a) + (1-\lambda)\mathrm{p}_1(b)) \\ &\quad + \log(\lambda \mathrm{p}_2(a|b) + (1-\lambda)\mathrm{p}_1(a)), \end{aligned}$$

which is exactly equal to the log of the score from the probability semiring.

- The score on any specific path will be the semiring **product** of all steps along the path.

- The score of any input will be the semiring **sum** of the scores of all paths that successfully process the input.
- What happens if we use the tropical semiring?

10.5 Finite state transducers

FSAs and WFSAs apply to single strings. FSTs and WFSTs apply to pairs of string.

FSTs define **regular relations** over pairs of strings. We can think of them in a few different ways:

- **Recognizer**: accepts string pairs iff they are in the relation
- **Translator**: reads an input, produces an output

Formally, a finite-state transducer $M = \langle Q, \Sigma, \Delta, q_0, F, \delta, \sigma \rangle$ consists of:

- A finite set of states $Q = \{q_0, q_1, \ldots, q_n\}$
- Finite alphabets Σ for input symbols and Δ for output symbols
- Initial state $q_0 \in Q$ and final states $F \subseteq Q$
- A state transition function $\delta : \langle Q \times \Sigma^* \rangle \rightarrow 2^Q$
- A string transition function $\sigma : \langle Q \times \Sigma^* \rangle \rightarrow 2^{\Delta^*}$

Unlike NFSAs, not all NFSTs can be determinized. However, special subsets of NFSTs called **subsequential** transducers can be determinized efficiently (see 3.4.1 in Jurafsky and Martin (2009)).

We can build some simple NLP systems directly from FSTs.

- A zeroth-order translation system could be made from a single state and a set of self-transitions: $Q_0 \xrightarrow[el]{the} Q_0$, $Q_0 \xrightarrow[los]{the} Q_0$, $Q_0 \xrightarrow[libro]{book} Q_0$, $Q_0 \xrightarrow[libros]{books} Q_0$,...
- First-order translation would require a state per word in the "input" vocabulary: $Q_0 \xrightarrow[\epsilon]{the} Q_{\text{the}} \xrightarrow[los\ libros]{books} Q_0$, $Q_{\text{the}} \xrightarrow[el\ libro]{book} Q_0$.
- Inflectional morphology and orthography:

$$Q_0 \xrightarrow[wit]{wit} Q_{\text{regular}} \xrightarrow[+s]{+\text{PL}} Q_1$$
$$Q_0 \xrightarrow[wish]{wish} Q_{\text{needs-e}} \xrightarrow[+es]{+\text{PL}} Q_1$$

10.6 Weighted FSTs

Weights can be added to FSTs in much the same way as they are added to FSAs.

- For any pair $\langle q \in Q, s \in \Sigma^* \rangle$, we have a set of possible transitions, $\langle q \in Q, t \in \Delta^*, \omega \in \mathbb{K} \rangle$, with a weight ω in the domain defined by the semiring.
- For example, we could augment the translation transducers defined above to allow alternative possible translations for a single word.
- The same semiring operations in WFSAs apply here too.

	acceptor	transducer
unweighted	FSA: $\Sigma^* \rightarrow \{0, 1\}$	WFSA: $\Sigma^* \rightarrow \mathbb{R}$
weighted	FST: $\Sigma^* \rightarrow \Sigma^*$	WFST: $\Sigma^* \rightarrow \langle \Sigma^*, \mathbb{R} \rangle$

Example : General edit distance computer.

- $Q_0 \xrightarrow[a]{a} Q_0 : 0$
- $Q_0 \xrightarrow[\epsilon]{a} Q_0 : 1$
- $Q_0 \xrightarrow[a]{\epsilon} Q_0 : 1$

The shortest path for a pair of strings $\langle \boldsymbol{s}, \boldsymbol{t} \rangle$ in this transducer has a score equal to the minimum edit distance between the strings (in the tropical semiring).

We can think of each path as defining a potential **alignment** between $\boldsymbol{s}$ and $\boldsymbol{t}$.

Operations on FSTs

- Closed under **union**
- Closed under **inversion**, which switches input and output labels.
- Closed under **projection**, because FSAs are a special case of FSTs
- Not closed under **difference**, **complementation**, and **intersection**.

- Closed under **composition**.

FST composition is the basis for implementing the noisy channel model in FSTs, and can be used to support dozens of cool applications.

Finite state composition

Suppose we have a transducer T_1 from language I_1 to O_1, and another transducer T_2 from O_1 to O_2. The composition $T_1 \circ T_2$ is an FST from I_1 to O_2.

- Unweighted definition: iff $\langle x, z \rangle \in T_1$ and $\langle z, y \rangle \in T_2$, then $\langle x, y \rangle \in T_1 \circ T_2$.
- Weighted definition:

$$(T_1 \circ T_2)(x, y) = \bigoplus_{z \in \Sigma^*} T_1(x, z) \otimes T_2(z, y) \tag{10.3}$$

- Designing algorithms for automatic FST composition is relatively straightforward if there are no epsilon transitions; otherwise it's more challenging (Allauzen et al., 2009).

The simplest example

- $T_1 : Q_0 \xrightarrow[a]{x} Q_0, Q_0 \xrightarrow[b]{y} Q_0$
- $T_2 : Q_1 \xrightarrow{a} Q_1, Q_1 \xrightarrow{b} Q_2, Q_2 \xrightarrow{b} Q_2$
- $T_1 \circ T_2 : Q_1 \xrightarrow{x} Q_1, Q_1 \xrightarrow{y} Q_2, Q_2 \xrightarrow{y} Q_2$

For simplicity T_2 is written as a finite-state acceptor, not a transducer. Acceptors are a special case of transducers.

If we had weights, they would be combined through $\otimes$.

10.7 Applications of composition

Edit distance

Consider the general edit distance computer developed in section 10.6. It assigns scores to pairs of strings. If we compose it with an FSA for a given string (e.g., *tech*), we get a WFSA, who assigns score equal to the minimum edit distance from *tech* for the input string.

- Composing an FST with a FSA yields a FSA.
- A very useful design pattern is to build a **decoding** WFSA by composing a general-purpose WFST with an unweighted FSA representing the input.
- The best path through the resulting WFSA will be the minimum cost / maximum likelihood decoding.

Transliteration

English is written in a Roman script, but many languages are not. **Transliteration** is the problem of converting strings between scripts. It is especially important for names, which don't have agreed-upon translations.

A simple transliteration system can be implemented through the noisy-channel model.

- T_1 is an English character model, implemented as a transducer so that strings are scored as $\log \mathrm{p}_r(c_1, c_2, \ldots, c_M)$.
- T_2 is a character-to-character transliteration model. This can be based on explicit rules,[1] or on conditional probabilities $\log \mathrm{p}_t(c^{(f)}|c^{(r)})$.
- T_3 is an acceptor for a given string that is to be transliterated.

The machine $T_1 \circ T_2 \circ T_3$ scores English character strings based on their orthographic fluency (T_1) and adequacy (T_2).

Suppose you were given an Roman-script character model and a set of foreign-script strings, but no equivalent Roman-script strings. How would you use EM to learn a transliteration model?

Knight and May (2009) provide a more complex transliteration model, which transliterates between Roman and Katakana scripts, using a deep cascade that includes models of the underlying phonology. In their model,

Word-based translation

As we have seen, simple models of machine translation can be implemented as finite-state transducers.

- Recall the example: *the books / los libros*. Here we are interested in modeling English-to-Spanish translation, $P(S|E)$.

[1] `http://en.wikipedia.org/wiki/Romanization_of_Russian`

- Earlier we proposed a multi-state translation model to deal with the impact of pluralization on the Spanish article *los*
- If we build our translator by composing a Spanish language model $P(S)$ with an Spanish-to-English transducer $P(E|S)$, this is not necessary. This is an example of the **noisy channel model**.

Here are the specifics:

- $T1$ is a language model, implemented as a transducer, where every path inputs and outputs the same string, with a score equal to $\log \text{p}(w_1, w_2, \ldots, w_M)$. This model's responsibility is to tell us that $\text{p}(\textit{el libros}) \ll \text{p}(\textit{los libros})$.
  - For example, a bigram language model would be implemented with V states, with a score of $\log P_S(W_{n+1} = i|W_n = j)$ for the edge from the state representing word j to the state representing word i.
  - We are free to use higher-order language models. Would we need a trigram language model to correctly identify *los grandes libros* as the right translation of *the great books*?
- $T2$ is a word-to-word translation model.
  - It could be a bilingual dictionary, with edge weights taking the value 0. In this case, the resulting translations will be scored only on boolean acceptability in the translation model, and on fluency according to the language model.
  - It could a probabilistic translation model, with each edge having a score equal to $\log \text{p}(w^{(e)}|w^{(s)})$, the log probability of the English word $w^{(e)}$ given the Spanish word $w^{(s)}$.
- $T3$ is an acceptor for a given foreign string $w^{(f)}$

So the translation model is $T_1 \circ T_2 \circ T_3$, which produces a WFSA, in which path are scored according to the joint log probability $\log \text{p}_{e|s}(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(e)})$. Using semiring notation, here's what happens in our example:

$$\text{p}(\textit{los libros}, \textit{the books}) = \text{p}_S(\textit{los}|\star) \otimes \text{p}_{E|S}(\textit{the}|\textit{los}) \otimes \text{p}_S(\textit{libros}|\textit{los}) \otimes \text{p}_{E|S}(\textit{books}|\textit{libros})$$

The composed FST can thus overcome its simplistic model of translation by having a more intelligent language model. This is useful, because language models can be trained without labeled data, while translation models cannot.

Structure prediction Transliteration and translation are examples of **structure prediction**, which will be a dominant theme in the remainder this course.

- Our goal for English-to-Spanish translation is to predict a sequence $\boldsymbol{w}^{(s)}$, given linguistic input $\boldsymbol{w}^{(e)}$.
- The set of possible sequences is very large — in word-to-word translation, it is V^M. Therefore, we need to score these structures in a decomposable way.
- Finite-state composition give us a way to do that. We are implicitly decomposing the score for the tuple $\langle \boldsymbol{w}^{(e)}, \boldsymbol{w}^{(s)} \rangle$ into scores for adjacent words in $\boldsymbol{w}^{(s)}$ and aligned word pairs.
- This scoring function can be written as $\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(e)})$, where the FST defines a specific, decomposable feature function. More on this later.

Nondeterminism While T_2 might be non-deterministic, there is no non-determinacy about **paths**: a given pair $\langle \boldsymbol{w}^{(e)}, \boldsymbol{w}^{(f)} \rangle$ can only be transduced in one way.

Knight and May (2009) introduce a more complex translation model. They relax the assumption that there is a monotonic word-to-word alignment, allowing reorderings and multiword translations.

Word segmentation

Word segmentation is a challenging problem for speech, and also for languages like Chinese, which don't explicitly segment adjacent words in orthography. We can again use a finite-state approach.

- T_1 is again a language model. We can use a simplistic model representing a dictionary, or we can use a probabilistic model.
- T_2 is a sequence of unsegmented symbols.

Suppose you are given dictionary of permissible words. How would you use EM to learn the language model?

Stemming

As discussed on Tuesday, information retrieval systems that only return exact matches aren't very useful.

- Suppose you query: *is it medically safe to kiss my cat on the lips*
- You want to get a hit even for documents like: *on the medical safety of kissing cats*.
- In morphologically complex languages like Hebrew, these differences could cause early IR systems to miss more than 90% of relevant documents Choueka (1989)!
- Stemming improves the **recall** of information retrieval systems by converting all tokens of *kissing* to *kiss*, etc.

The **Porter Stemmer** is a very popular stemming program. It is written as a set of rules, which are applied in stages, e.g.

- if the word ends in *-ing* and the preceeding part contains a vowel, delete the ending: *hopping* $\rightarrow$ *hopp*
- if the word ends in *-ed*, and the preceeding part contains a vowel, delete the ending: *hopped* $\rightarrow$ *hopp*
- Next, if the remaining part contains double letters (besides ss, ll, or zz), remove one (*hopp* $\rightarrow$ *hop*)

We can think of these rules as a sequence of deterministic finite state transducers:

- We can build a transducer to strip off endings like *-ing*, using two states to indicate whether we have yet seen a vowel
- Next we can build a transducer to strip off double letter endings, with exceptions for s, l, and z.
- We can **compose** these transducers into a single machine.

Morphological analysis

Recall that when we talked about morphology, there were several types of interacting rules:

1. To pluralize regular words, add an *s*; but if the word ends in *sh*, you have to add *es*.

2. To conjugate 3rd-person singular, add an *s*; but if the word ends in *sh*, you have to add *es*.

Pluralization and conjugation are different morphological systems, but once they have decided to append an s, the subsequent orthographical rules are the same.

This suggests an intermediate representation:

- *dish*+PL $\rightarrow$ *dish*$\wedge$*s*# $\rightarrow$ *dishes*
- *fist*+PL $\rightarrow$ *fist*$\wedge$*s*# $\rightarrow$ *fists*
- *fish*+PL $\rightarrow$ *fish*# $\rightarrow$ *fish*
- *fish*+3S $\rightarrow$ *fish*$\wedge$*s*# $\rightarrow$ *fishes*
- Special symbols for morpheme and word boundaries
- The conjugation and pluralization FSTs only need to know what affix to add, and where to add it.
- Then an orthography FST takes over, and figures out how the morphemes should be combined.
- Finite-state composition allows us to automatically build a single machine for conjugation and pluralization, incorporating both the selection of affixes and orthographic constraints.

We have only described how to *generate* the English text. But if we compose this machine with a chain FSA representing an observed string, then we obtain an FSA where the set of accepted strings reveal the acceptable morphological analyses.

- For example, an utterance like *wishes* could have been produced by *wish/N*+3S or by *wish/V*+PL; both will be accepted by the resulting FSA.
- Suppose we want to distinguish the most likely morphological analysis given the context.
  - First, we add a loop back to the beginning in the morphological FST, so that we can accept multiple words.

  - Now we can build something like a language model WFST, but here we need probabilities of morphological analyses rather than just words. We might want to decompose this like

$$P(stem, affixes|context) \approx P(stem|context)P(affix|context) \quad (10.4)$$

  - If we have a morphologically annotated corpus, we can make smoothed relative frequency estimates of these probabilities. If we don't, what do we do?
  - The morphological analyses are missing data, so we might be able to do EM:
    * **E-step**: Decode all observed string sequences in the corpus, given the current probabilities.
    * **M-step**: Treat decoding as ground truth, update probabilities
    * Note: this is "Viterbi" EM, (a.k.a. "Hard" EM), because we are not computing probabilities over all possible decodings. We could do that using something called the expectation semiring Eisner (2002).

Context-sensitive spelling correction

Swype text entry in my old android phone does not consider context, much to my annoyance.

- I mean: *Prepare lecture for my class*
- It says: *Prepare lecture foie my class*

That's not smart. The bigram probabilities $P(\textit{lecture foie})$ and $P(\textit{foie my})$ should be ridiculously low. Once again, we can apply the noisy channel model, starting with T_1 as a language model and T_2 as a spelling model. If T_1 is a bigram language model, then the resulting machine is an FST with one state per word type.

- The cost of the transition from Q_a to Q_b on input s is the semiring product $\otimes$ of two costs:
  - The transition cost from a to b in the language model
  - The "emission" cost from s to a in the edit distance machine. Depending on exactly how we're getting our input, we could memorize these costs (for all pairs of words) and work at the token level, rather than character level. But there's no conceptual difference; alternatively you could think about character-level edit distance FSTs sitting at each state.

- Given an input sequence s, we compose

  select the spelling-corrected string t with the greatest value,

  $$\hat{t} = \max_t \bigoplus_\pi s \rightsquigarrow_\pi t, \tag{10.5}$$

  where π is a path over input s and output t.

  - Why are there multiple paths from s to t? In the edit distance machine, we can always model matching input/output symbols as an insertion and a deletion.
  - Whether we care about these alternative paths depends on the semiring. In the tropical semiring, the solution is simply the minimum-cost path.

Norvig's spelling corrector

We haven't said much about where the weights come from. The bigram language model is probabilistic, and we can compute $\delta(q_s, q_t, t, t)$ as $-\log P(t|s)$. What about the edit distance model?

Before we get into that, here's an alternative approach, from Peter Norvig `http://norvig.com/spell-correct.html` (highly recommended). It may help explain how google is able to quickly and accurately spell-check your queries.

The approach can't easily be framed in exactly terms of the FST/semiring formalism, but it's close:

- Check if the word itself is in the dictionary. If so, return it.
- Else, check if any edit-distance=1 corrections is in the dictionary. If so, return the one with the highest unigram count.
- Else try corrections with edit-distance = 2.

The essay discusses extensions to bigrams a probabilistic model of errors. One way to build such a model is to look at data.

- Norvig suggests the Birbeck spelling error corpus, `http://norvig.com/spell-correct.html`.
  - Such a resource would tell us the likelihood of a word s being mispelled as t.

  - But new mispellings are always possible (even inevitable)

- An alternative would be to parametrize the edit distance model with more specific probabilities: P(insertion), P(deletion), etc.
  - Maximum-likelihood estimation would compute

$$P(\text{insertion}) = \frac{\text{count}(\text{insertion})}{\text{count}(\text{all-characters})} \tag{10.6}$$

  - But we will probably never get a corpus that gives us these counts.
  - Can we bootstrap our way to success? Suppose we could just guess the probabilities.
    * We could find the best path for each example in the training set, and keep track of the counts of insertions and deletions in these paths.

$$\hat{\pi}_{s,t} = \text{arg}\min_{\pi} \text{cost}(s \rightsquigarrow_{\pi} t)$$
$$\text{count}(\text{insertion}) = \sum_{\langle s,t \rangle \in \mathcal{D}} \text{count}(\text{insertion}, \hat{\pi}_{s,t})$$

    * More probabilistically, we could compute the likelihood of each path, and use these likelihood to find the *expected* counts:

$$\text{count}(\text{insertion}) = \sum_{\langle s,t \rangle \in \mathcal{D}} \sum_{\pi} \frac{P(s \rightsquigarrow_{\pi} t)}{\sum_{\pi'} P(s \rightsquigarrow_{\pi'} t)} \text{count}(\text{insertion}, \pi_{s,t})$$

    * Once we have the counts, we can go back and re-estimate the insertion probability (and all the other probabilities).

- This type of bootstrapping is another example of **expectation-maximization**.
  - We introduced EM in the simpler case of clustering.
    * Each document had a distribution over clusters $q(z)$
    * Each cluster had parameters θ.
    * The E-step computed $q(z)$, the M-step optimized θ.
  - Here, the Q distribution is over paths $Q(\pi)$.
  - In the E-step, we can compute $Q(\pi)$ given estimates of the parameters $P(\text{insertion})$.

  - In the M-step, we can update the parameters $P(\text{insertion})$ from expected counts under $Q(\pi)$.

- The number of paths can be very large, often too large to store.
  - In **Viterbi EM**, we just use the best path. This can work very well.
  - Alternatively, we can use the **expectation semiring** so that the "score" of the weighted path sum includes the expected counts Eisner (2001).

    Each edge is a tuple of a probability and a vector of expected counts, $\langle p_i, p_i v_i \rangle$

$$\langle p_i, p_i v_i \rangle \otimes \langle p_j, p_j v_j \rangle = \langle p_i p_j, p_j p_i v_i + p_i p_j v_j \rangle = \langle p_k, p_k v_k \rangle$$
$$\langle p_i, v_i \rangle \oplus \langle p_j, v_j \rangle = \langle p_i + p_j, v_i + v_j \rangle$$

  - If we can compute the semiring shortest path ($\mathcal{O}(n^3)$ at worst), we can compute the expected counts!

Speech Recognition

Speech recognition is yet another application of finite-state methods in NLP. It takes the idea of intermediate representations even further.

We want a mapping from intended words to observed acoustics. This can be seen as a multilevel process:

- G: WFST which generates English sentences: *I went to the bank*
- L: WFST which converts each word to context-indendent phonemes (WFST. Pronunciation dictionaries have this information): *Ay w eh n t t uw ...*
- C: WFST which converts context-independent phonemes to context-dependent phonemes: *Ay w eh n t uw ...*
- A: WFST which converts context-dependent phonemes to acoustic observations

By composing the FST $G \circ L \circ C \circ A$ with an FSA representing the observed acoustics O, we obtain a single WFSA which scores proposed English sentences for the observations O.

Inference The composed FSA would be ridiculously huge. Beam pruning is a technique for pruning away paths which are extremely unlikely, resulting in a faster, smaller FST.

Estimation It's really hard to get labeled data for all of these levels, for example context-dependent phones. We can treat this as a hidden variable and estimate it using EM.

Software There are mature software toolkits for working with finite state machines. OpenFST is a C++ package which I have had some experience with; it's fast and relatively well-documented. XFST and Carmel are other options.

10.8 Recap

We saw how finite-state composition can create powerful NLP systems out of simple, modular components.

- For example, we can compose a translation model (one state!) and a bigram language model (V states) to create a finite-state translation machine.
- If we compose the translation machine with a **chain FSA** representing the "input", we get a WFSA whose shortest path is the best translation of the input.
- We didn't talk about algorithms for composition, but the formal definition is

$$(T_1 \circ T_2)(x, y) = \bigoplus_z T_1(x, z) \otimes T_2(z, y). \tag{10.7}$$

  In other words, to compute the score of $\langle x, y \rangle$ in the composed machine, we take the semiring addition over all strings z, for which we compute the extension of the score of $\langle x, z \rangle$ in T_1 with the score of $\langle z, y \rangle$ in T_2.
- For example, the language model only has edges for $\langle s, s \rangle$, with score $\log P(s_i|s_{i-1})$. The translation model has edges for all $\langle s, t \rangle$, with score $\log P(t|s)$. So the composed machine must have edges for all $\langle s, t \rangle$, with score $\log P(t|s) \otimes \log P(s_i|s_{i-1})$.
- In the chain FSA, each edge takes exactly one string t_i, with score $\bar{1}$. So the composed machine is a WFSA, with edges for each possible s_i, each having score $\log P(t_i|s_i) \otimes \log P(s_i|s_{i-1}) \otimes \bar{1}$. This structure is known as a **trellis**.

Chapter 11

Part-of-speech tagging

Words can be grouped into rough classes based on syntax.

- Why is *colorless green ideas sleep furiously* more acceptable than *ideas colorless furiously green sleep*?
- Why is *teacher strikes idle children* ambiguous?

In both examples, word classes can provide an explanation.

- Word classes have strong ordering constraints:
  - J J N V R is likely
  - N J R J V is unlikely (why?)
- Ambiguity about word class leads to very different interpretations:
  - N N V N
  - N V J N (ouch!)

So clearly we have intuitions about a few parts-of-speech already: noun, verb, adjective, adverb. Jurafsky and Martin (2009) describe these as the four major **open** word classes, although apparently not all languages have all of them.

What other word classes are there?

- The Penn Treebank defined a set of 45 POS tags.[1]

[1] `http://www.comp.leeds.ac.uk/ccalas/tagsets/upenn.html`

- The Brown corpus defined a set of 87 POS tags.[2]
- Petrov et al. (2012) define a "universal" set of 12 tags.

Which is right?

Example Let's look at some data.

> My name is Ozymandias, king of kings:
> Look on my works, ye Mighty, and despair!

The part-of-speech tags for this couplet from Ozymmandias are shown in Table 11.1.

- All tagsets distinguish basic categories like nouns, pronouns, verbs, adjectives, and punctuation.
- The Brown tagset includes a number of fine-grained distinctions:
  - specific tags for the *be*, *do*, and *have* verbs, which the other two tagsets just lump in with other verbs.
  - distinct tags for possessive determiners (*my name*) and possessive pronouns (*mine*)
  - distinct tags for the third-person singular pronouns (e.g., *it, he*) and other pronouns (e.g., *they, we, I*)
- The Universal tagset aggressively groups categories that are distinguished in the other tagsets:
  - all nouns are grouped, ignoring number and the proper/common distinction (see below)
  - all verbs are grouped, ignoring inflection
  - preposition and postpositions are grouped as adpositions
  - all punctuation is grouped
  - coordinating and subordinating conjunctions (e.g. *and* versus *that*) are grouped

[2] `http://www.comp.leeds.ac.uk/ccalas/tagsets/brown.html`

	Brown	PTB	Universal
My	possessive determiner (DD$)	possessive pronoun (PRP$)	pronoun (PRON)
name	noun, singular, common (NN)	NN	NOUN
is	verb "to be" 3rd person, singular (BEZ)	verb 3rd person, singular (VBZ)	VERB
Ozymandias	proper noun, singular (NP)	proper noun, singular (NNP)	NOUN
,	comma (,)	comma (,)	punctuation (.)
king	NN	NN	NOUN
of	preposition (IN)	preposition (IN)	adposition (ADP)
kings	noun, plural, common (NNS)	NNS	NOUN
:	colon (:)	mid-sentence punc (:)	.
Look	verb, base: uninflected present, imperative, or infinite (VB)	VB	VERB
on	IN	IN	ADP
my	DD$	PRP$	PRON
works	NNS	NNS	NOUN
ye	personal pronoun, nominative, non 3S (PPSS)	personal pronoun, nominative (PRP)	PRON
mighty	adjective (JJ)	JJ	adjective (ADJ)
,	comma (,)	comma (,)	punctuation (.)
and	coordinating conjunction (CC)	CC	conjunction (CONJ)
despair	VB	VB	VERB

Table 11.1: Part-of-speech annotations from three tagsets.

So which is "right"? It depends. The Brown tags can be useful for certain applications, and they may have strong tag-to-tag relations that make tagging

easier (see next chapter). But they are more expensive to annotate. The Universal tags are intended to generalize across many types of text, and should be easier to annotate.

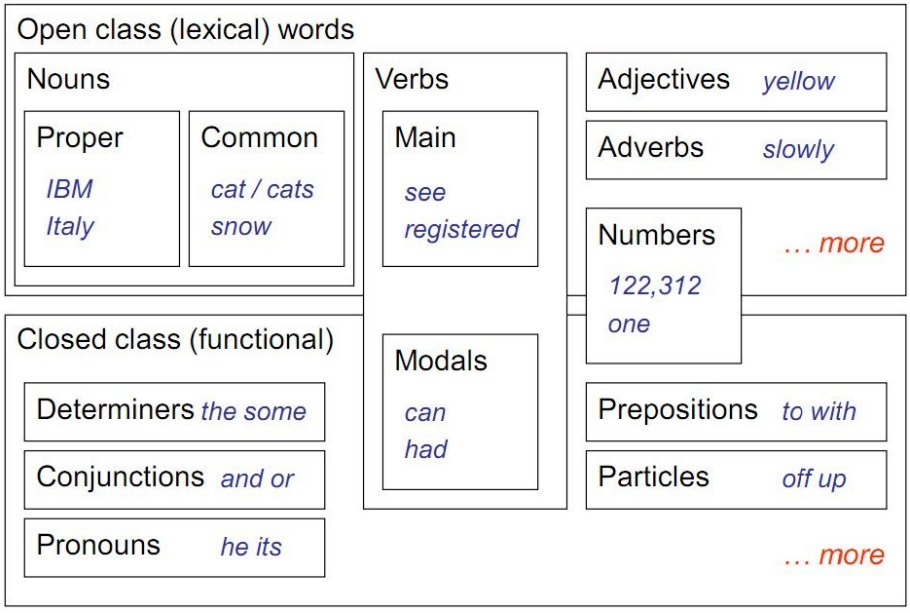


Figure 11.1: needs attribution

11.1 Details about parts-of-speech

As usual, Bender (2013) provides a useful linguistic perspective.

- **Nouns** describe entities and concepts
  - **Proper nouns** name specific people and entities: *Georgia Tech*, *Janet*, *Buddhism*. In English, they're usually capitalized. PTB tags: NNP (singular), NNPS (plural)
  - **Common nouns** cover everything else. In English, they're often preceded by articles, e.g. *the book*, *the university*. Common nouns decompose into
    - **Count nouns** have a plural and need an article in the singular, *dogs*, *the dog*.

    - **Mass nouns** don't have a plural and don't an article in the singular, *snow is cold, gas is expensive*
  - **Pronouns** refer to specific noun phrases or entities or events.
    - **Personal pronouns** refer to people or entities: *you, she, I, it, me*. PTB tag: PRP
    - **Possessive pronouns** are pronouns that indicate possession: *your, her, my, its, one's, our*. PTB tag: PRP$
    - **Wh-pronouns** are used in question forms (*Where are you going?*, WP) and as relative pronouns in forms like (*The girl who played with fire*)

    Unlike other nouns, the set of possible pronouns cannot be expanded! It is a **closed class**.

- **Verbs** describe activities, processes, and events, e.g. *eat, write, sleep*
  - The Penn Treebank differentiates verbs by morphology: VB (infinitive), VBD (past), VBG (present participle), VBN (past participle), VBP (present, non 3rd person-singular), VBZ (present 3rd person singular)
  - **modals** are a closed subclasses of verbs, such as (*should, can, will, must*). They get PTB tag MD
  - **copula** is *be* with a predicate, e.g. *she is hungry*. The Brown Tagset distinguishes copula, but PTB doesn't.
  - **auxiliary** verbs include *be, have, will* to form complex tenses, e.g. *we will have done it twice*.
    - Also includes *do* in questions and negation, e.g. *did you eat yet?*. Apparently this is from Welsh, which was spoken in England before the Anglo-Saxons invaded; *do* doesn't function this way in German.
    - The Brown corpus has special tags for HAVE and DO, but the PTB doesn't.

- **Adjectives** describe properties of entities: *antique, vast, trunkless*
  - **Attributive use**: *an antique land*
  - **Predicative use**: *the land was antique*
  - **Gradable adjectives** (*big*) have **comparative form** (*bigger*) and **superlative form** (*biggest*)

  - Can you think of an adjective that is not gradable?
  - With *big*, we can move to comparative form by adding the suffix *-est*. This is an example of agglutinative morphology. Can you think of an adjective in English where the relationship is not agglutinative? (rather, it's fusional). How about *good*, *better*, *best*?
  - PTB tags: JJ, JJR, JJS

- **Adverbs** describe properties of events.
  - Manner: *slowly, slower, fast, hesitantly*
  - Degree: *extremely, very, highly*
  - Directional and locative: *here, downstairs, near*
  - Temporal: *yesterday, Monday*
  - Besides verbs, adverbs may also modify sentences, adjectives, or other adverbs. Apparently, the very ill man walks extremely slowly
  - Adverbs may also be gradable. Tags: RB, RBR, RBS

- **Prepositions** are a closed class of words that can come before noun phrases, forming a prepositional phrase that relates the noun phrase to something else in the sentence.
  - *I eat sushi WITH soy sauce*
  - *I eat sushi WITH chopsticks*
  - *To* gets its own tag TO, because it forms the infinitive with bare form verbs (VB), e.g. *I want to eat*
  - Everything else is tagged IN in the PTB.

- **Coordinating conjunctions** join two elements,
  - *vast and trunkless legs*
  - *She eats burgers but she drinks soda*
  - PTB tag: CC

- **Subordinating conjunctions** introduce a subordinate clause, e.g. *She thinks THAT Chomsky is wrong*. PTB tag: annotIN

- **Particles** are oddball words that come with verbs and can change their meaning to a new **phrasal verb**, e.g., *come ON*, *he brushed himself OFF*, *let's check OUT that new restaurant*. They are a closed class, PTB tag RP.
- **Determiners** are a closed class of words that precede noun phrases.
  - Articles: *the, an, a*
  - Demonstratives: *this, these, that*
  - Quantifiers: *some, every, few*
  - Wh-determiners, *WHICH burger should I choose?*
  - PTB tag: DT
- **Oddballs**
  - **Existential there**, e.g. *There is no way out of here*, gets its own tag, EX.
  - So does the possessive ending *'s*, POS
  - So do numbers (CD), list items (LS), commas, and other non-alphabetic symbols.

11.2 Part of speech tagging

Part of speech tags relate to a number of other linguistic phenomena:

- Lexical semantics: *can*/V vs *can*/N, *teacher strikes children*, etc
- Pronunciation: *inSULT* vs *INsult*, *conTENT* vs *CONtent*
- Translation: *park*/v $\rightarrow$ *garer*, *park*/N $\rightarrow$ *parque*
- NP chunking: `grep {JJ | NN}* {NN | NNS}`

POS tagging is a useful preprocessing step for downstream applications.
So how can we build an automatic POS tagger?

- Observation 1: it's "easy."
  - 60% of word types have only one possible POS tag (in English).
  - If you choose the majority POS tag for each token, you get 90% right.
- Observation 2: it's not easy: a few words have a lot of possible POS tags

  - *We're taking it **back**/RB*
  - *The shirt off my **back**/NN*
  - *Go **back**/RP where you belong*
  - *If you challenge him, I'll **back**/VBP you*
  - *The **back**/JJ roads are safer*

- Observation 3: 90% is not actually very good. $0.9^{10} \approx .3$, so you will only get 30% of ten-word sentences correct. Sentences have exponentially many possible POS sequences:

VBD		VB			
VBN	VBZ	VBP	VBZ		
NNP	NNS	NN	NNS	CD	NN
fed	*raises*	*interest*	*rates*	*0.5*	*percent*

Anyway, let's look at a tougher poem, Jabberwocky:

'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe:
All mimsy were the borogoves,
And the mome raths outgrabe

Forget *twas*. What about *slithy*? Can you guess the POS? What about *toves*? You don't know these words. What information are you using to guess?

- Word identity: you do know that *and* is CC and *the* is DET
- **Context**
  - JJ NN is likely
  - NN JJ is unlikely
- **Morphology**
  - *-s* $\rightarrow$ noun or verb
  - *-able* $\rightarrow$ adjective (98% of the time!)
  - *-ly* $\rightarrow$ adverb
  - *un-* $\rightarrow$ adjective or verb

  - (not rules, just hints)

Let's put morphology on hold for a minute.

- Suppose we have an annotated corpus, with tagged sentences, $\langle \boldsymbol{w}_{1:N_t}, \boldsymbol{y}_{1:N_t} \rangle_{1:T}$.
- Then we could estimate the likelihood of a word given a tag,

$$P(w|y) = \frac{\text{count}(w, y)}{\text{count}(y)} \tag{11.1}$$

  As always, smoothing is possible...

- Given this same annotated corpus, we could also compute $P(y_n|y_{n-1})$, a sort of language model over tags.

$$P(y_n|y_{n-1}) = \frac{\text{count}(y_{n-1}, y_n)}{\text{count}(y_{n-1})} \tag{11.2}$$

- Let's combine these ideas via a **generative story**
  - For word n, draw tag $y_n \sim \text{Categorical}(\theta_{y_{n-1}})$
  - Then draw word $w_n \sim \text{Categorical}(\phi_{y_n})$

  We've built a generative model that explains our observations $\boldsymbol{w}$ through a bigram generative model over the tags.

- Under this model, we can compute

$$P(\boldsymbol{y}|\boldsymbol{w}) \propto P(\boldsymbol{w}, \boldsymbol{y}) \tag{11.3}$$
$$P(\boldsymbol{w}|\boldsymbol{y})P(\boldsymbol{y}) \tag{11.4}$$
$$\prod_n^N P(w_n|y_n)P(y_n|y_{n-1}) \tag{11.5}$$

- This is a **hidden Markov model**. It's Markov because the probability of y_n depends only on y_{n-1} and not any of the previous history. It's hidden because y_n is unknown when we decode.
- We can treat this as a special case of finite state transduction. Can you see how?

Chapter 12

Sequence Labeling

In sequence labeling, we want to assign tags to words. There are many applications:

- Part-of-speech tagging: *Go*/V *to*/P *Georgia*/N *Tech*/N *next*/J *year*/N ./.
- Named entity recognition: *Go*/O *to*/O *Georgia*/B-ORG *Tech*/I-ORG *next*/B-DATE *year*/I-DATE ./O
- Phrase chunking: *Go*/B-VP *to*/B-PP *Georgia*/B-NP *Tech*/I-NP *next*/B-NP *year*/I-NP ./O

In classification, we would choose each tag independently:

$$\mathrm{p}(y_m|w_m) \perp \mathrm{p}(y_n|w_n), \forall m \neq n \tag{12.1}$$

In sequence labeling, we choose the sequence of tags **jointly**. Probabilistically, we might try to choose $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{T}^M} \mathrm{p}(\boldsymbol{y}|\boldsymbol{w})$. As we will see later, we can also write this in the form of a linear predictor:

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{T}^M} \boldsymbol{\theta}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) \tag{12.2}$$

In either case, we have an immediate problem: finding the best scoring tag sequence in the set $\mathcal{T}^M$. As the notation suggests, the number of possible tag sequences is exponential in the length of the sequence. Consider part-of-speech tagging:

VBD		VB			
VBN	VBZ	VBP	VBZ		
NNP	NNS	NN	NNS	CD	NN
fed	*raises*	*interest*	*rates*	*0.5*	*percent*

Even after using a tag dictionary to restrict the set of possible tags for each word, there are thirty-six possible tag sequences for this six-word sentence. This means we will need clever algorithms to compute $\arg\max_{\boldsymbol{y} \in \mathcal{T}^M}$. We cannot enumerate all possibilities.

12.1 Hidden Markov Models

Let's first think about tagging as a probabilistic model. Specifically, we want to maximize $\text{p}(\boldsymbol{y}|\boldsymbol{w}) \propto \text{p}(\boldsymbol{y}, \boldsymbol{w})$, where $\boldsymbol{w}$ are words and $\boldsymbol{y}$ are tags. This is equivalent to Naive Bayes, but for sequence labeling.

As in Naive Bayes, we define the probability distribution $\text{p}(\boldsymbol{w}, \boldsymbol{y})$ through a *generative story*,

- For word m, draw tag $y_m \sim \text{Categorical}(\lambda_{y_{m-1}})$
- Then draw word $w_m \sim \text{Categorical}(\phi_{y_m})$

Under this model, we can compute

$$\begin{aligned} \text{p}(\boldsymbol{y} \mid \boldsymbol{w}) \propto & \text{p}(\boldsymbol{w}, \boldsymbol{y}) && (12.3) \\ & \text{p}_e(\boldsymbol{w} \mid \boldsymbol{y}; \phi) \text{p}_t(\boldsymbol{y}; \lambda) && (12.4) \\ & \prod_m^M \text{p}_e(w_m \mid y_m; \phi) \text{p}_t(y_m \mid y_{m-1}; \lambda) && (12.5) \end{aligned}$$

- This is a **hidden Markov model**. It's Markov because the probability of y_m depends only on y_{m-1} and not any of the previous history. It's hidden because y_m is unknown when we decode.
  - The probability $\text{p}_e(w_m \mid y_m; \phi)$ is the **emission probability**, since the words are treated as emissions from the tags.
  - The probability $\text{p}_t(y_m \mid y_{m-1}; \lambda)$ is the **transition probability**, since it assigns probability to each possible tag-to-tag transition.

- We can describe this generative story as a graphical model. Note that although graphical models and finite-state models both use circles and arrows, the meaning is completely different.

- Our generative story assumes that the words are conditionally independent, given the tags, so
$$w_n \perp \{\boldsymbol{w}_{m \neq n}\} \mid y_n.$$
Conditional independence is not the same as independence. We do **not** have $\text{p}(w_n, w_m) = \text{p}(w_n)\text{p}(w_m)$ because the tags are related to each other. For example, suppose that (a) nouns always follow determiners, (b) *the* is always a determiner and (c) *bike* is always a noun. Then

$$\begin{aligned}
P(W_m = \textit{the}, W_{m+1} = \textit{bike}) &= \sum_{y_{m+1}, y_m} P(W_m = \textit{the}, W_{m+1} = \textit{bike}, y_{m+1}, y_m) \\
&= \sum_{y_{m+1}, y_m} P(W_{m+1} = \textit{bike} \mid y_{m+1}, y_m, W_m = \textit{the}) \\
&\quad \times P(y_{m+1} \mid y_m, W_m = \textit{the}) P(y_m \mid W_m = \textit{the}) P(W_m = \textit{the}) \\
&= \sum_{y_{m+1}} P(W_{m+1} = \textit{bike} \mid y_{m+1}) \\
&\quad \times \sum_{y_m} P(y_{m+1} \mid y_m) P(y_m \mid W_m = \textit{the}) P(W_m = \textit{the}) \\
&= P(W_{m+1} = \textit{bike} \mid y_{m+1} = \text{NOUN}) \times 1 \times 1 \times P(W_m = \textit{the}) \\
&> P(W_{m+1} = \textit{bike}) P(W_m = \textit{the})
\end{aligned}$$

- Another way to think about independence is that if we are told one tag, it affects all of our other tagging decisions.
  - For example, in the sentence *teacher strikes idle children*, we might choose tag sequence NN VBZ JJ NNS.
  - But if are given $y_3 = \text{VBP}$, then suddenly $y_2 = \text{VBZ}$ looks like a bad choice because $\text{p}_T(\text{VBZ}, \text{VBP})$ is very small.
  - So we might now choose $y_2 = \text{NNS}$.
  - This change might cascade back to y_1, etc (not in this case, but it could happen in theory)

- A classifier-based tagger, which treated the tags as IID, might ignore these dependencies, and produce a tag sequence that contained unlikely transitions like VBZ, VBP.

- A better alternative might be to tag the text from left-to-right; we could then condition on the previous tag, choosing

$$y_m = \arg\max_y \mathrm{p}_e(w_m|y_m)\mathrm{p}_t(y_m|y_{m-1}) \tag{12.6}$$

  But this approach is "greedy," and can mistakenly commit to bad tagging decisions. For example, in *teacher strikes strand children*, we might initially choose $y_2 = \text{VBZ}$, because this is more common than the noun sense of *strikes*. However, we are then stuck, because *strand* has low probability as anything but a verb, yet the verb-verb transition also has low probability. The greedy tagger is unable to recover the globally optimal sequence, NN NNS VBP NNS, without backtracking.

- This is why we need **joint inference** over $\boldsymbol{y}_{1:M}$ to find $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} \mathrm{p}(\boldsymbol{w}, \boldsymbol{y})$. The key challenge is to search over the exponential number of tag sequences efficiently.

12.2 Sequence labeling as finite-state transduction

To see whether efficient joint inference is possible, we first formulate the problem in terms of finite-state transduction.

- Transducer E has one state, and transduces from tags to words, with $\delta^{(e)}_{w,t} = \mathrm{p}_e(w|y)$.

- Transducer T has $\#|\mathcal{T}|$ states (if it's a bigram model), and transduces tags to tags, with $\delta^{(t)}_{y_m,y_{m-1}} = \mathrm{p}_t(y_m|y_{m-1})$.

- Recall the definition of finite state composition,

$$(T \circ E)(y, x) = \bigoplus_z T(y, z) \otimes E(z, x). \tag{12.7}$$

  Since T only accepts identical tag pairs $\langle y, y\rangle$, we can ignore $\bigoplus$; there's only one possible $z = y$. The result of $T \circ E$ transduces tags to words, with edge

weights

$$\begin{aligned}\delta^{(t\circ e)}_{w,y_m,y_{m-1}} =& \delta^{(e)}_{w,y_m} \otimes \delta^{(t)}_{y_{m-1},y_m} \\ =& \mathrm{p}(w \mid y_m) \otimes \mathrm{p}(y_m \mid y_{m-1}) \\ =& \mathrm{p}(w \mid y_m)\mathrm{p}(y_m \mid y_{m-1}) \\ =& \mathrm{p}(w, y_m \mid y_{m-1})\end{aligned}$$

Suppose we wanted to work with log probabilities instead. Then

$$\begin{aligned}\delta_{w,t} =& \log \mathrm{p}(x \mid y) \\ \delta_{y_m,y_{m-1}} =& \log \mathrm{p}(y_m \mid y_{m-1}) \\ a \otimes b :=& a + b \\ \delta_{x,y_m,y_{m-1}} =& \log \mathrm{p}(x \mid y_m) \otimes \log \mathrm{p}(y_m \mid y_{m-1}) \\ =& \log \mathrm{p}(x \mid y_m) + \log \mathrm{p}(y_m \mid y_{m-1}) \\ =& \log \mathrm{p}(x, y_m \mid y_{m-1})\end{aligned}$$

Can you see how many states the resulting FST will have?

- Finally, we compose with an acceptor S, which forms a chain for a sentence $w_1, \ldots w_M$.
- This composition $T \circ E \circ S$ yields a **trellis**-shaped weighted finite state acceptor (WFSA).
  - Number of columns = M, length of input.
  - Number of rows = T, number of tags.
  - Edges from states $\langle m, t_1 \rangle$ to $\langle m+1, t_2 \rangle$ have the score

$$\delta^{(t\circ e)}_{w_{m+1},t_2,t_1} = \delta^{(t)}_{t_2,t_1} \otimes \delta^{(e)}_{t_2,w_{m+1}} = P(Y_{m+1} = t_2 \mid Y_m = t_1) P(W_{m+1} = w_{m+1} \mid Y_{m+1} = t_2) \tag{12.8}$$

  - Each path in the trellis corresponds to a unique sequence of tags, $\boldsymbol{y}_{1:M}$, and every sequence of tags has a unique path. The score of the path is equal to $\mathrm{p}(\boldsymbol{w}_{1:M}, \boldsymbol{y}_{1:M})$ by construction.
  - If we define $\bigoplus = \max$ (as in the tropical semiring), then the score of the semiring shortest path is equal to $\max_{\boldsymbol{y}} \mathrm{p}(\boldsymbol{w}_{1:M}, \boldsymbol{y}_{1:M})$.
- So, can we find this score (and therefore the best path) in polynomial time?

  - How expensive is it to construct the trellis?
    * Generic composition is polynomial but slower than we would like — it depends on the vocabulary size.
    * But since we know what the trellis is supposed to look like, we can just build it directly. This requires constant time per edge.
    * How big is the trellis? $\mathcal{O}(MT)$ states, $\mathcal{O}(MT^2)$ edges.
  - How expensive is it find the shortest path in the trellis?: Generic shorest path has a time cost of $\mathcal{O}(V \log V + E) = \mathcal{O}(MT \log MT + MT^2)$ and a space cost of $\mathcal{O}(V) = \mathcal{O}(MT^2)$.
  - So:
    * Building the trellis is polynomial.
    * Shortest path is polynomial.
    * Therefore, there must be a poly-time algorithm to find the best tag sequence, despite the apparently exponential number of paths.

12.3 The Viterbi algorithm

The Viterbi algorithm is a special-purpose best-path algorithm for FSTs in the shape of a trellis. It has a time cost of $\mathcal{O}(MT^2)$ and a space cost of $\mathcal{O}(MT)$. (This time cost improvement is important, because it is linear in the length of the sequence M, unlike the generic shortest-path algorithm, which is $M \log M$.

Based on the Markov assumption, we can decompose the likelihood recursively.

$$\text{p}(\boldsymbol{w}_{1:M}, \boldsymbol{y}_{1:M}) = \text{p}(w_M \mid y_M)\text{p}(y_M \mid y_{M-1})\text{p}(\boldsymbol{w}_{1:M-1}, \boldsymbol{y}_{1:M-1})$$

- Given y_{m-1}, we can choose y_m without considering any other element of the history.

- Suppose we know the best path to $y_m = k$.
  The best path to $y_{m+1} = k'$ *through* $y_m = k$ must include the best path to $y_m = k$.

- Suppose we know the score (probability) of the best path to each $y_m = k$, which we write $v_m(k) = \max_{y_1 \ldots y_{m-1}} \text{p}(\boldsymbol{w}_{1:m}, \boldsymbol{y}_{1:m-1}, y_m = k)$.

What is the score of the best path to $y_{m+1} = k'$?

$$v_{m+1}(k') = \max_{\boldsymbol{y}_{1:m}} \text{p}(\boldsymbol{w}_{1:m+1}, \boldsymbol{y}_{1:m}, y_{m+1} = k') \tag{12.9}$$
$$= \text{p}_e(w_{m+1} \mid y_{m+1} = k') \max_{\boldsymbol{y}_{1:m}} P_t(Y_{m+1} = k' \mid y_m) \text{p}(\boldsymbol{w}_{1:m}, \boldsymbol{y}_{1:m}) \tag{12.10}$$
$$= \text{p}_e(w_{m+1} \mid y_{m+1} = k') \max_{y_m = k} P_t(Y_{m+1} = k' \mid Y_m = k) \max_{\boldsymbol{y}_{1:m-1}} \text{p}(\boldsymbol{w}_{1:m}, \boldsymbol{y}_{1:m-1}, y_m = k) \tag{12.11}$$
$$= \text{p}_e(w_{m+1} \mid y_{m+1} = k') \max_{y_m = k} P_t(Y_{m+1} = k' \mid Y_m = k) v_m(k) \tag{12.12}$$

The base case is $v_0(\diamond) = 1$, with zero probability for everything else.

- We can generalize this recurrence using semiring notation:

$$v_{m+1}(k') = \delta^{(e)}_{w_{m+1}, y_{m+1} = k'} \otimes \bigoplus_k \delta^{(t)}_{k \rightarrow k'} \otimes v_m(k) \tag{12.13}$$

Then if we want to move to log-probabilities, we have

$$v_{m+1}(k') = \log \text{p}_E(w_{m+1} \mid y_{m+1} = k') \otimes \bigoplus_k \log \text{p}_T(k \rightarrow k') \otimes v_m(k) \tag{12.14}$$
$$= \log \text{p}_E(w_{m+1} \mid y_{m+1} = k') + \max_k \log \text{p}_T(k \rightarrow k') + v_m(k) \tag{12.15}$$

- We will frequently use a semiring in which the edge weights are log probabilities and $\otimes$ is addition. This is partly because addition is notationally clearer than multiplication, and because in practical settings, you will use the log probabilities to avoid underflow.

- Note that we are setting $\oplus = \max$, as in the tropical semiring. This means that the score of the best tag sequence overall is $v_M(\square)$.

- To find the best tag sequence, we just need to keep back-pointers, from $v_m(k)$ to $v_{m-1}(k')$:

$$v_{m+1}(k') = \max_k \log \mathrm{p}_E(w_{m+1} \mid Y_{m+1} = k') + \log P_T(Y_{m+1} = k' \mid Y_m = k) + v_m(k) \tag{12.16}$$

$$= \log \mathrm{p}_E(w_{m+1} \mid y_{m+1} = k') + \left(\max_k \log P_T(Y_{m+1} = k' \mid Y_m = k) + v_m(k) \right) \tag{12.17}$$

$$b_{m+1}(k') = \arg\max_k \log \mathrm{p}_E(w_{m+1} \mid Y_{m+1} = k') + \log P_T(Y_{m+1} = k' \mid Y_m = k) + v_m(k) \tag{12.18}$$

$$= \arg\max_k \log P_T(Y_{m+1} = k' \mid Y_m = k) + v_m(k) \tag{12.19}$$

Note that the computation of the back-pointer doesn't depend on the emission probability $\mathrm{p}_E(w_{m+1} \mid Y_{m+1} = k')$, since Y_m is conditionally independent from w_{m+1} given Y_{m+1}.

- In the probability semiring, we had $\oplus$ as addition; in the log-probability semiring, it was log addition. What happens if we try these addition operators? We'll see in a moment.

Example

Table 12.1: $\log \mathrm{p}_e(w \mid y)$

	they	*can*	*fish*
N	-2	-3	-3
V	-10	-1	-3

Table 12.2: $\log \mathrm{p}_t(y_m \mid y_{m-1})$

	N	V	END
START	-1	-2	$-\infty$
N	-3	-1	-2
V	-1	-3	-2

See the slides for how the Viterbi algorithm works in this example.

12.4 The forward algorithm

In an influential survey on HMMs, Rabiner (1989) defines three problems:

- **Decoding**: find the best tags $\boldsymbol{y}$ for a sequence $\boldsymbol{w}$.
- **Likelihood**: compute the probability $\text{p}(\boldsymbol{w}) = \sum_{\boldsymbol{y}} \text{p}(\boldsymbol{w}, \boldsymbol{y})$
- **Learning**: given only unlabeled data $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_D\}$, estimate the transition and emission distributions.

The Viterbi algorithm solves the decoding problem. We'll talk about the learning problem later. Let's now consider how to compute the likelihood $\text{p}(\boldsymbol{w}) = \sum_{\boldsymbol{y}} \text{p}(\boldsymbol{w}, \boldsymbol{y})$.

- First, we move to a semiring where $a \oplus b = \log(e^a + e^b)$ instead of $\max$. Then

$$\alpha_{m+1}(k') = \bigoplus_k \log \text{p}_E(w_{m+1} \mid Y_{m+1} = k') \otimes \log P_T(Y_{m+1} = k' \mid Y_m = k) \otimes \alpha_m(k) \tag{12.20}$$

$$= \log \text{p}_E(w_{m+1} \mid Y_{m+1} = k') \otimes \bigoplus_k \log P_T(Y_{m+1} = k' \mid Y_m = k) \otimes \alpha_m(k) \tag{12.21}$$

$$= \log \text{p}_E(w_{m+1} \mid Y_{m+1} = k') + \log \sum_k P_T(Y_{m+1} = k' \mid Y_m = k) \times e^{\alpha_m(k)} \tag{12.22}$$

$$= \log \text{p}_E(w_{m+1} \mid Y_{m+1} = k') + \log \sum_k P_T(Y_{m+1} = k' \mid Y_m = k) \text{p}(\boldsymbol{w}_{1:m}, Y_m = k) \tag{12.23}$$

$$= \log \text{p}_E(w_{m+1} \mid Y_{m+1} = k') + \log P_T(Y_{m+1} = k', \boldsymbol{w}_{1:m}) \tag{12.24}$$

$$= \log \text{p}(\boldsymbol{w}_{1:m+1}, Y_{m+1} = k') \tag{12.25}$$

- We used the inductive hypothesis in (12.23), and we used the HMM conditional independence assumptions $W_{m+1} \perp \boldsymbol{W}_{1:m} \mid Y_{m+1}$ and $Y_{m+1} \perp \boldsymbol{W}_{1:m} \mid Y_m$ in the following two steps.
- We can formalize this as an inductive proof by stating the base case,

$$\alpha_1(k) = \log \text{p}_e(w_1 \mid y_1) \otimes \log P_t(Y_1 = k \mid \diamond) = \log \text{p}(w_1, Y_1 = k \mid Y_0 = \diamond). \tag{12.26}$$

This is called the **forward** algorithm. The total probability of a sequence is $\text{p}(\boldsymbol{w}_{1:M}) = \alpha_M(\square)$. For a demo, see the slides.

Why solve the likelihood problem?

Why would we want to compute $\text{p}(\boldsymbol{w}_{1:M})$?

Word class language models

- Remember $\text{p}(\textit{colorless green ideas sleep furiously})$
- We don't care about the specific tags, we just want to know the probability of the utterance, so we can compare it with $\text{p}(\textit{Furiously sleep ideas green colorless})$.

Comparing HMMs

- Suppose we have a few HMMs, each of which could have generated the observations.
- We want to compute the marginal likelihood of the observations given each HMM, regardless of the path.
- This approach is sometimes used in gesture recognition.

Computing marginals : we'll soon be very interested in **marginal** probabilities $\text{p}(y_m \mid \boldsymbol{w}_{1:M})$, for all $m \leq M$. The likelihood $\text{p}(w_{1:M})$ is part of this computation.

12.5 HMM Details

Trigrams

- Can we use trigrams instead of bigrams for an HMM?
- How do we change the trellis? How big is the new trellis?
- Each cell represents a pair of tags, $\langle y_m, y_{m-1} \rangle$.
- The trellis still needs N columns, but now need T^2 rows.
- Each node can only connect to T neighbors in the next column, based on the trigram transition constraint. Number of edges = NT^3.

- We can prune very low probability edges for speed.

Estimation

In principle, we can use relative frequency estimation.

$$\lambda_{k,k'} \triangleq P_T(Y_m = k' \mid Y_{m-1} = k) = \frac{\text{count}(Y_m = k', Y_{m-1} = k)}{\text{count}(Y_{m-1} = k)}$$

$$\phi_{k,i} \triangleq P_E(W_m = i \mid Y_m = k) = \frac{\text{count}(W_m = i, Y_m = k)}{\text{count}(Y_m = k)}$$

In practice, we need smoothing and other tricks.

- The same smoothing ideas from language modeling can be applied.
  - interpolate between trigrams, bigrams, and unigrams

    $P(y_m \mid y_{m-1}, y_{m-2}) = \lambda_2 \hat{P}_2(y_m \mid y_{m-1}, y_{m-2}) + \lambda_1 \hat{P}_1(y_m \mid y_{m-1}) + (1 - \lambda_2 - \lambda_1)\hat{P}_0(y_m)$
  - reserve probability mass for unseen words

12.6 Tagging with features

Let's consider an example:

> 'Twas brillig, and the slithy toves
> Did gyre and gimble in the wabe:
> All mimsy were the borogoves,
> And the mome raths outgrabe

You probably don't know many of these words, yet it's not so hard to see what some of their tags should be. How do you do it?

Recall that the HMM can incorporate two sources of information:

- Word-tag probabilities, via $\text{p}_E(w_m|y_n)$.
- Local context, via $\text{p}_T(y_m|y_{m-1})$.

Local context is helpful, but the word-tag probabilities will be worthless for words like *brillig*, *slithy*, *toves*, *gyre*, etc.

But there are a lot of things that we're missing here. Crucially, unseen words have internal structure which can be exploited:
mimsy, *3.1415*, *Ke$ha*

- **Orthography and morphology**. *Slithy toves* just kind of looks like JJ NNS, because of the apparent suffixes.
  - *-s* $\rightarrow$ NNS, VBZ
  - *-able* $\rightarrow$ JJ
  - *-ly* $\rightarrow$ RB
  - *un-* $\rightarrow$ JJ,RB,V
  - ...

  But morphological features are difficult to incorporate in a generative model, because they break the Naive Bayes assumption:

  $$\text{p}(\textit{mimsy}, \textit{-sy} \mid \text{JJ}) \neq \text{p}(\textit{mimsy} \mid \text{JJ})\text{p}(\textit{-sy} \mid \text{JJ})$$

- **Capitalization**. This is especially relevant for named entity recognition, e.g., *I bought an apple* and *I bought an Apple phone*.

- More advanced HMMs incorporate morphological, orthographic, and typographic features by creating a more complex $\text{p}_E(w \mid y)$ emission probability. For example, the TNT Tagger took this approach, and is one of the best generative taggers (Brants, 2000).

In addition to word-internal features, we might want more fine-grained context. For example, in the PTB, *this* and *these* are both tagged DT. But *this* is likely to be followed by a singular noun NN, and *these* is likely to be followed by a plural noun NNS. So we might like to add word-context features to the probability $\text{p}(y_m \mid y_{m-1}, w_{m-1})$.

How can we incorporate these overlapping features? The solution is to build sequence labeling models based on the perceptron and logistic regression classifiers. The first model is called **structured perceptron**, since the label space consists of structures rather than individual labels (Collins, 2002). The second model is called a **conditional random field (CRF)**, due to its relation to Markov random fields (Lafferty et al., 2001). In this model, we explicitly compute $\text{p}(\boldsymbol{y} \mid \boldsymbol{w})$.

In addition to incorporating overlapping features, these models have another advantage: they are discriminative, directly maximizing the conditional probability $\text{p}(\boldsymbol{y} \mid \boldsymbol{w})$, or minimizing the perceptron loss. As in standard classification, this criterion is more closely connected to the accuracy metrics that we usually care about.

12.7 Structured perceptron

Remember the perceptron update:

$$\hat{y} = \arg\max_y \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}, y) \tag{12.27}$$

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \boldsymbol{f}(\boldsymbol{w}, y^*) - \boldsymbol{f}(\boldsymbol{w}, \hat{y}) \tag{12.28}$$

In sequence labeling, we have a **structured output** $\boldsymbol{y} \in \mathcal{T}(\boldsymbol{w})$. Can we still apply the perceptron rule?

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{T}(\boldsymbol{w})} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{y}) \tag{12.29}$$

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{y}^*) - \boldsymbol{f}(\boldsymbol{w}, \hat{\boldsymbol{y}}) \tag{12.30}$$

This is called structured perceptron, because it learns to predict structured output $\boldsymbol{y}$. The problem is that $\arg\max_y$ must search over the entire set of structures $\mathcal{T}$. The set of permissible outputs depends on the input $\boldsymbol{w}$, and it is very large: $\mathcal{O}(K^M)$. What can we do?

Viterbi inference for structured perceptron

We will place a restriction on the scoring function $\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{y})$, which allows us to apply the Viterbi algorithm to find $\arg\max_{\boldsymbol{y}} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{y})$!

- Specifically, we require $\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{y}) = \sum_m \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}, y_m, y_{m-1}, m)$
- That is, the global score must be a **sum of local scores**.
- The local scores can consider any part of the observation, but must only consider adjacent elements in the label.

To apply Viterbi to structured perceptron, we have

$$\begin{aligned} v_m(k) &= \bigoplus_{k'} \boldsymbol{\theta}^\top \boldsymbol{f}_m(\boldsymbol{w}, k, k', m) \otimes v_{m-1}(k') \\ &= \max_{k'} \boldsymbol{\theta}^\top \boldsymbol{f}_m(\boldsymbol{w}, k, k', m) + v_{m-1}(k') \\ b_m(k) &= \arg\max_{k'} \boldsymbol{\theta}^\top \boldsymbol{f}_m(\boldsymbol{w}, k, k', m) + v_{m-1}(k') \end{aligned}$$

Suppose we want to apply this to POS tagging? What features might we want? Here are some:

- Word-tag features, e.g. $\langle W : \textit{slithy}, \text{JJ} \rangle$
- Adjacent tag-tag features, e.g. $\langle T : \text{JJ}, \text{NNS} \rangle$
- Suffix-tag features, e.g., $\langle M : \textit{-es}, \text{NNS} \rangle$
- Previous-word features, e.g., $\langle P_1 : \textit{the}, \text{JJ} \rangle$
- Next-word features, e.g., $\langle N_1 : \textit{slithy}, \text{DT} \rangle$
- Note that we can consider arbitrarily distant words, e.g. $\langle Y_m, W_{m-15} \rangle$, because this still fits in the constraint, $\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{y}) = \sum_m \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}, y_m, y_{m-1}, m)$.

So to compute $v_m(k)$, we have to iterate over all $y_{m-1} = k'$,

- find the features $\boldsymbol{f}(\boldsymbol{x}_{1:M}, y_m = k, y_{m-1} = k', m)$,
- compute the inner product $\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{x}_{1:M}, y_m = k, y_{m-1} = k', m)$,
- add it to $v_{m-1}(k')$
- take the max over all k'

This only works because of the assumption that the feature function decomposes over local parts of the sequence! If we wanted a feature that considered arbitrary parts of tag sequence, there would be no way to incorporate it into the recurrence relation.

Example

$$\boldsymbol{w} = \ldots \textit{and the slithy toves}$$
$$\boldsymbol{y} = \ldots \text{CC DT JJ NNS}$$

Then we can characterize the tagging DT JJ NNS of the text the slithy toves (from Jabberwocky) in terms of the following features:

$$\begin{aligned}\boldsymbol{f}(\text{the slithy toves}, \text{DT JJ NNS}) = \{&\langle W : \text{the}, \text{DT} \rangle, \langle M : \varnothing, \text{DT} \rangle, \langle T : \diamond, \text{DT} \rangle \\ &\langle W : \text{slithy}, \text{JJ} \rangle, \langle M : \text{-thy}, \text{JJ} \rangle, \langle T : \text{DT}, \text{JJ} \rangle \\ &\langle W : \text{toves}, \text{NNS} \rangle, \langle M : \text{-es}, \text{NNS} \rangle, \langle T : \text{JJ}, \text{NNS} \rangle \\ &\langle T : \text{NNS}, \square \rangle\}\end{aligned}$$

If this is the correct tagging, then we hope to learn a set of weights $\boldsymbol{w}$ such that $\boldsymbol{\theta}^\top \boldsymbol{f}(\text{the slithy toves}, \text{DT JJ NNS})$ is larger than the scores for other tag sequences, such as $\boldsymbol{\theta}^\top \boldsymbol{f}(\text{the slithy toves}, \text{DT NN VBZ})$.

Learning

If we define $\boldsymbol{f}(\boldsymbol{w}_{1:M}, y_m, y_{m-1}, m) = \{\langle W : w_m, y_m\rangle, \langle T : y_{m-1}, y_m\rangle\}$, then our model is identical to the HMM. If we set the weights of these features to the log of their maximum-likelihood estimates,

$$w_{\langle W:w_m,y_m\rangle} = \log \text{count}(w_m, y_m) - \log \text{count}(y_m)$$
$$w_{\langle T:y_{m-1},y_m\rangle} = \log \text{count}(y_{m-1}, y_m) - \log \text{count}(y_{m-1}),$$

then we exactly recover the HMM.

But to use more overlapping features and to get the advantages of error-driven learning, we're going to do perceptron updates. It's exactly the same as the non-structured perceptron:

- $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) - \boldsymbol{f}(\boldsymbol{x}, \hat{\boldsymbol{y}})$ is the standard update, using Viterbi to find $\hat{\boldsymbol{y}}$.
- As before, weight averaging is crucial to get good performance (Collins, 2002).
- As before, we can use Passive-Aggressive to do large-margin training (Crammer et al., 2006), computing the step size by dividing a non-negative **loss** $\ell(\boldsymbol{y}_i, \hat{\boldsymbol{y}})$ by the squared norm of the difference in the feature vectors, $||\boldsymbol{f}(\boldsymbol{y}_i, \boldsymbol{w}_i) - \boldsymbol{f}(\hat{\boldsymbol{y}}, \boldsymbol{w}_i)||^2$. A reasonable choice of loss function is the Hamming loss, which is the number of incorrect tag predictions (Taskar et al., 2003; Tsochantaridis et al., 2004).

12.8 Conditional random fields

Structured perceptron works well in practice, and you will implement in your project 2, where it should work much much better than the Hidden Markov Model.

- But sometimes we need probabilities, and SP doesn't give us that.
- The Conditional Random Field (CRF) is a probabilistic conditional model for sequence labeling.

- Just as structured perceptron is built on the perceptron classifier, conditional random fields are built on the logistic regression classifier.

$$\mathrm{p}(y \mid \boldsymbol{x}) = \frac{e^{\boldsymbol{\theta}^\top \boldsymbol{f}(y, \boldsymbol{x})}}{\sum_{y' \in \mathcal{Y}} e^{\boldsymbol{\theta}^\top \boldsymbol{f}(y', \boldsymbol{x})}} \tag{12.31}$$

We can again move to structured prediction,

$$\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{w}) = \frac{e^{\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{w})}}{\sum_{\boldsymbol{y}' \in \mathcal{T}(\boldsymbol{w})} e^{\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{y}', \boldsymbol{w})}}. \tag{12.32}$$

This is called a Conditional Random Field, because it models the sequence labeling task as a Markov random field, and estimates the probability of a set of variables **conditioned** on the others (as opposed to jointly, which the HMM does). We will need the same restriction on the scoring function as in Structured Perceptron: $\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{y}) = \sum_m \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}, y_m, y_{m-1}, m)$.

An important note about CRFs is that the joint probability $\mathrm{p}(\boldsymbol{w}_{1:M}, \boldsymbol{y}_{1:M})$ is simply the unnormalized conditional probability:

$$\mathrm{p}(\boldsymbol{y}|\boldsymbol{w}) = \frac{\mathrm{p}(\boldsymbol{y}, \boldsymbol{w})}{\sum_{\boldsymbol{y}'} \mathrm{p}(\boldsymbol{y}', \boldsymbol{w})} \tag{12.33}$$

$$\mathrm{p}(\boldsymbol{y}, \boldsymbol{w}) = e^{\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{w})} \tag{12.34}$$

Aside: Maximum Entropy Markov Models (MEMMS)

Suppose we define

$$\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{w}) = \prod_m^M \mathrm{p}(y_m \mid \boldsymbol{w}_{1:M}, y_{1:m-1}) \tag{12.35}$$

$$\approx \prod_m^M \mathrm{p}(y_m|\boldsymbol{w}_{1:M}, y_{m-1}). \tag{12.36}$$

We can then define each local probability $\mathrm{p}(y_m \mid \boldsymbol{w}_{1:M}, y_{m-1})$ as a logistic regression model,

$$\mathrm{p}(y_m \mid \boldsymbol{w}_{1:M}, y_{m-1}) = \frac{\exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}_{1:M}, y_m, y_{m-1}))}{\sum_{y' \in \mathcal{T}} \exp(\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}_{1:m}, y', y_{m-1}))}. \tag{12.37}$$

Recall that logistic regression is sometimes called **maximum entropy**, and observe that we are making a Markov assumption. Thus the name **Maximum Entropy Markov Model**.

Inference in the MEMM can be performed with Viterbi, choosing

$$v_m(k) = \max_{k'} P(Y_m = k \mid Y_{m-1} = k', w_{1:M}) v_{m-1}(k'). \tag{12.38}$$

The local decision model $\text{p}(y_m | \boldsymbol{w}_{1:M}, y_{m-1})$ can be trained as a standard logistic regression classifier. The problem with this model is that learning to optimize individual tagging decisions is not the same as learning to produce optimal tag sequences. The local classifier is trained with the true value of y_{m-1}, not the value likely to be produced by the classifier — so, not necessarily the value that we are most likely to see in a test set tagging situation. This introduces a problem that Lafferty et al. (2001) called **label bias**. Put another way, the MEMM allows structured **prediction**, but it does not perform structured **learning**.

Decoding in CRFs

Back to CRFs! Decoding in the CRF does not depend on the denominator of $\text{p}(\boldsymbol{y} \mid \boldsymbol{w})$

$$\begin{aligned}
\hat{\boldsymbol{y}} =& \arg\max_{\boldsymbol{y}} \text{p}(\boldsymbol{y} \mid \boldsymbol{w}) \\
=& \arg\max_{\boldsymbol{y}} \log \text{p}(\boldsymbol{y} \mid \boldsymbol{w}) \\
=& \arg\max_{\boldsymbol{y}} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{w}) - \log \sum_{\boldsymbol{y}' \in \mathcal{T}(\boldsymbol{w})} e^{\boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{y}', \boldsymbol{w})} \\
=& \arg\max_{\boldsymbol{y}} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{w}) \\
=& \arg\max_{\boldsymbol{y}} \boldsymbol{\theta}^\top \sum_{m=0}^{M} \boldsymbol{f}(\boldsymbol{w}, y_m, y_{m-1}, m) \\
=& \arg\max_{\boldsymbol{y}} \sum_{m=0}^{M} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}, y_m, y_{m-1}, m)
\end{aligned}$$

So we can apply Viterbi in exactly the same way that we did in the Structured Perceptron.

Learning in CRFs

Learning is a little more complicated. As with logistic regression, we need to learn weights to minimize the regularized negative log conditional probability,

$$\begin{aligned}\ell &= \sum_{i=1}^{N} -\log \mathrm{p}(\boldsymbol{y}_i \mid \boldsymbol{w}_i; \boldsymbol{\theta}) + \lambda||\boldsymbol{\theta}||^2, \\ &= -\sum_i \boldsymbol{\theta}^\mathsf{T} \boldsymbol{f}(\boldsymbol{w}_i, \boldsymbol{y}_i) + \log \sum_{\boldsymbol{y}' \in \mathcal{T}(\boldsymbol{w}_i)} \exp\left(\boldsymbol{\theta}^\mathsf{T} \boldsymbol{f}(\boldsymbol{w}_i, \boldsymbol{y}')\right) + \lambda||\boldsymbol{\theta}||^2,\end{aligned}$$

where λ controls the amount of regularization. As in logistic regression, the gradient includes is a difference between observed and expected counts:

$$\begin{aligned}\frac{d\ell}{d\theta_j} &= \sum_i \text{count}(\boldsymbol{w}_i, \boldsymbol{y}_i)_j - E_{\boldsymbol{y}|\boldsymbol{w}_i;\boldsymbol{\theta}}[\text{count}(\boldsymbol{w}_i, \boldsymbol{y})_j] + \lambda\boldsymbol{\theta} \\ \text{count}(\boldsymbol{w}_i, \boldsymbol{y}_i)_j &= \sum_m^M f_j(\boldsymbol{w}_i, y_{i,m}, y_{i,m-1}, m)\end{aligned}$$

For example:

- If feature j is $\langle T : CC, DT \rangle$, then $\text{count}(\boldsymbol{w}_i, \boldsymbol{y}_i)_j$ is the count of times DT follows CC in the sequence $\boldsymbol{y}_i$.
- If feature j is $\langle M : \textit{-thy}, \text{JJ} \rangle$, then $\text{count}(\boldsymbol{w}_i, \boldsymbol{y}_i)_j$ is the count of words ending in *-thy* in $\boldsymbol{w}_i$ that are tagged JJ in $\boldsymbol{y}_i$.

The expected feature counts are more difficult to compute.

$$E_{\boldsymbol{y}|\boldsymbol{w};\boldsymbol{\theta}}[\text{count}(\boldsymbol{w}_i, \boldsymbol{y})_j] = \sum_{\boldsymbol{y} \in \mathcal{T}(\boldsymbol{w}_i)} P(\boldsymbol{y} \mid \boldsymbol{w}_i; \boldsymbol{\theta}) f_j(\boldsymbol{w}_i, \boldsymbol{y}) \tag{12.39}$$

- This looks bad: we have to sum over an exponential number of labelings again.
- But remember that the feature function decomposition implies that

$$f_j(\boldsymbol{w}, \boldsymbol{y}) = \sum_m f_j(\boldsymbol{w}, y_m, y_{m-1}, m) \tag{12.40}$$

This means we can compute the expectation as,

$$E_{\boldsymbol{y}|\boldsymbol{w};\boldsymbol{\theta}}[\text{count}(\boldsymbol{w},\boldsymbol{y})_j] = \sum_{\boldsymbol{y}\in\mathcal{T}(\boldsymbol{w})} \text{p}(\boldsymbol{y} \mid \boldsymbol{w};\boldsymbol{\theta}) f_j(\boldsymbol{w},\boldsymbol{y}) \tag{12.41}$$

$$= \sum_{\boldsymbol{y}\in\mathcal{T}(\boldsymbol{w})} \text{p}(\boldsymbol{y} \mid \boldsymbol{w};\boldsymbol{\theta}) \sum_{m}^{M} f_j(\boldsymbol{w}, y_m, y_{m-1}, m) \tag{12.42}$$

$$= \sum_{m}^{M} \sum_{\boldsymbol{y}\in\mathcal{T}(\boldsymbol{w})} \text{p}(\boldsymbol{y} \mid \boldsymbol{w};\boldsymbol{\theta}) f_j(\boldsymbol{w}, y_m, y_{m-1}, m) \tag{12.43}$$

$$= \sum_{m}^{M} \sum_{k,k'\in\mathcal{T}} \sum_{\boldsymbol{y}\in\mathcal{T}(\boldsymbol{w}):Y_{m-1}=k,Y_m=k'} \text{p}(\boldsymbol{y} \mid \boldsymbol{w};\boldsymbol{\theta}) f_j(\boldsymbol{w}, k', k, m) \tag{12.44}$$

$$= \sum_{m}^{M} \sum_{k,k'\in\mathcal{T}} f_j(\boldsymbol{w}, k', k, m) \sum_{\boldsymbol{y}\in\mathcal{T}(\boldsymbol{w}):y_{m-1}=k,y_m=k'} \text{p}(\boldsymbol{y} \mid \boldsymbol{w};\boldsymbol{\theta}) \tag{12.45}$$

$$= \sum_{m}^{M} \sum_{k,k'\in\mathcal{T}} f_j(\boldsymbol{w}, k, k', m) P(Y_{m-1}=k, Y_m=k' \mid \boldsymbol{w};\boldsymbol{\theta}) \tag{12.46}$$

- The expected feature counts can be computed efficiently if we know the **marginal** probabilities $P(Y_m = k', Y_{m-1} = k \mid \boldsymbol{w};\boldsymbol{\theta})$.
- This is the probability of traversing the trellis edge $\langle m-1, k\rangle \rightarrow \langle m, k'\rangle$, conditioned on the entire observation $\boldsymbol{w}_{1:M}$. [Draw this in trellis]
- This marginal probability can be computed through the combination of two dynamic programming algorithms, the Forward and Backward algorithms.

12.9 The Forward-backward algorithm

The compute the gradient of the CRF objective with respect to the weights $\boldsymbol{\theta}$, we need marginal probabilities,

$$P(Y_m = k', Y_{m-1} = k|\boldsymbol{w}_{1:M}) = \frac{P(Y_m = k', Y_{m-1} = k, \boldsymbol{w}_{1:M})}{\text{p}(\boldsymbol{w}_{1:M})}. \tag{12.47}$$

To compute these, recall that in the CRF,

$$\begin{aligned}
\mathrm{p}(\boldsymbol{y}|\boldsymbol{w}) =& \frac{\exp\left(\boldsymbol{\theta}^\mathsf{T}\boldsymbol{f}(\boldsymbol{w},\boldsymbol{y})\right)}{\sum_{\boldsymbol{y}'}\exp\left(\boldsymbol{\theta}^\mathsf{T}\boldsymbol{f}(\boldsymbol{w},\boldsymbol{y})\right)} && (12.48)\\
\propto& \exp\left(\boldsymbol{\theta}^\mathsf{T}\boldsymbol{f}(\boldsymbol{w},\boldsymbol{y})\right) && (12.49)\\
=& \exp\left(\boldsymbol{\theta}^\mathsf{T}\sum_m \boldsymbol{f}(\boldsymbol{w},y_m,y_{m-1},m)\right) && (12.50)\\
=& \prod_m \exp\left(\boldsymbol{\theta}^\mathsf{T}\boldsymbol{f}(\boldsymbol{w},y_m,y_{m-1},m)\right) && (12.51)\\
=& \prod_m \psi_m(y_m,y_{m-1}), && (12.52)
\end{aligned}$$

where $\psi_m(k,k') \triangleq \exp\left(\boldsymbol{\theta}^\mathsf{T}\boldsymbol{f}(\boldsymbol{w},Y_m=k,Y_{m-1}=k',m)\right)$. This quantity is sometimes called a **potential**. A Hidden Markov Model (HMM) can also be expressed in terms of a product of potential functions, $\psi_m(k,k') = \mathrm{p}_E(w_m \mid Y_m=k)P_T(Y_m=k \mid Y_{m-1}=k')$.

Using this expression for the probability of a label sequence, we will compute the desired marginal probabilities in terms of three quantities:

$$\begin{aligned}
Z \triangleq& \sum_{\boldsymbol{y}'}\prod_m \psi_m(y'_m,y'_{m-1}) && (12.53)\\
\alpha_m(k) \triangleq& \sum_{\boldsymbol{y}'_{1:m-1}} \psi_m(k,y_{m-1})\prod_n^{m-1}\psi_n(y'_n,y'_{n-1}) && (12.54)\\
\beta_m(k) \triangleq& \sum_{\boldsymbol{y}'_{m+1:M}} \psi_{m+1}(y_{m+1},k)\prod_{n=m+2}^{M}\psi_n(y'_n,y'_{n-1}), && (12.55)
\end{aligned}$$

where Z is the **normalization constant** (also called the partition function), α is the set of forward variables (just like in the forward algorithm defined earlier), and β is the set of backward variables.

If we assume that the first tag and last tag have special values, $y_0 = \diamond$ and

$y_M = \square$, then by definition,

$$
\begin{aligned}
Z =& \alpha_M(\square) && (12.56)\\
=& \beta_0(\diamond) && (12.57)\\
=& \sum_{\boldsymbol{y}} \text{p}(\boldsymbol{w}, \boldsymbol{y}) && (12.58)\\
=& \text{p}(\boldsymbol{w}) && (12.59)
\end{aligned}
$$

The forward variables

We define the first sum as $\alpha_m(k)$, called the **forward variable**. This is just like in the forward algorithm defined earlier for Hidden Markov Models; again, we can derive a recursive expression for α as follows:

$$
\begin{aligned}
\alpha_0(\diamond) \triangleq& 1 && (12.60)\\
\alpha_m(k) \triangleq& \sum_{\boldsymbol{y}'_{1:m-1}} \psi_m(k, y'_{m-1}) \prod_{n=1}^{m-1} \psi_n(y'_n, y'_{n-1}) && (12.61)\\
=& \sum_{Y_{m-1}=k'} \psi_m(k, k') \sum_{\boldsymbol{y}_{1:m-2}} \psi_{m-1}(k', y'_{m-2}) \prod_{n=1}^{m-2} \psi_n(y'_n, y'_{n-1}) && (12.62)\\
=& \sum_{Y_{m-1}=k'} \psi_m(k, k') \alpha_{m-1}(k'), && (12.63)
\end{aligned}
$$

where we simply substitute in the recursive expression for $\alpha_{m-1}(k')$. This variable is computed from left to right, with each $\boldsymbol{\alpha}_m$ depending on $\boldsymbol{\alpha}_{m-1}$.

The backward variables

We define the second sum as $\beta_m(k)$, called the **backward variable**. We can derive a recursive expression for β as follows:

$$\beta_M(\square) \triangleq 1 \tag{12.64}$$

$$\beta_m(k) \triangleq \sum_{\boldsymbol{y}'_{m+1:M}} \psi_{m+1}(y_{m+1}, k) \prod_{n=m+2}^{M} \psi_n(y'_n, y'_{n-1}), \tag{12.65}$$

$$= \sum_{Y_{m+1}=k'} \psi_{m+1}(k', k) \sum_{\boldsymbol{y}_{m+2:M}} \psi_{m+2}(y_{m+2}, k') \prod_{n=m+3}^{M} \psi_n(y_n, y_{m-1}) \tag{12.66}$$

$$= \sum_{Y_{m+1}=k'} \psi_{m+1}(k', k)\beta_{m+1}(k') \tag{12.67}$$

This variable is computed from right to left, with each $\boldsymbol{\beta}_m$ depending on $\boldsymbol{\beta}_{m+1}$.

Computing the marginals Now, the product $\alpha_m(k)\beta_m(k)$ is equal to

$$\alpha_m(k)\beta_m(k) = \sum_{\boldsymbol{y}_{1:m-1}} \left(\prod_{n=1}^{m-1} \psi_n(y_n, y_{n-1}) \right) \psi_m(k, y_{m-1}) \tag{12.68}$$

$$\times \sum_{\boldsymbol{y}_{m+1:M}} \psi_{m+1}(y_{m+1}, k) \left(\prod_{n=m+2}^{M} \psi_n(y_n, y_{n-1}) \right) \tag{12.69}$$

$$= \sum_{\boldsymbol{y}_{1:M}: Y_m = k} \prod_{n=1}^{M} \psi_n(y_n, y_{n-1}), \tag{12.70}$$

which is exactly equal to the sum of the unnormalized probabilities of all sequences in which $Y_m = k$. To compute the normalized probability, we simply divide by Z, so that,

$$P(Y_m = k \mid \boldsymbol{w}_{1:M}) = \frac{\alpha_m(k)\beta_m(k)}{Z} \tag{12.71}$$

To obtain the unnormalized probability of a tag-to-tag transition $\langle Y_m = k, Y_{m-1} =$

$k'\rangle$, we compute,

$$\alpha_{m-1}(k')\psi_m(k,k')\beta_m(k) = \sum_{\boldsymbol{y}_{1:m-2}} \left(\prod_{n=1}^{m-2} \psi_n(y_n, y_{n-1}) \right) \psi_{m-1}(k', y_{m-2}) \tag{12.72}$$

$$\times \psi_m(k,k') \sum_{\boldsymbol{y}_{m+1:M}} \psi_{m+1}(y_{m+1}, k) \prod_{n=m+2}^{M} \psi_n(y_n, y_{n-1}) \tag{12.73}$$

$$= \sum_{\boldsymbol{y}_{1:M}: Y_m = k, Y_{m-1} = k'} \prod_{n=1}^{M} \psi_n(y_n, y_{n-1}), \tag{12.74}$$

which is exactly equal to the unnormalized probability of all sequences in which $Y_m = k$ **and** $Y_{m-1} = k'$. To compute the normalized probability, we again divide by Z, so that,

$$P(Y_m = k, Y_{m-1} = k' \mid \boldsymbol{w}_{1:M}) = \frac{\alpha_{m-1}(k')\psi_m(k,k')\beta_m(k)}{Z} \tag{12.75}$$

Learning in CRFs: wrapup

The overall procedure looks like logistic regression:

- Use forward-backward to compute expected feature counts under $P(\boldsymbol{y} \mid \boldsymbol{w}; \boldsymbol{\theta})$:

$$E[f_j(\boldsymbol{y}, \boldsymbol{w})] = \sum_m \sum_{k,k'} P(Y_m = k, Y_{m-1} = k' | \boldsymbol{w}_{1:M}) f_j(\boldsymbol{w}, k, k', m) \tag{12.76}$$

$$= \frac{1}{Z} \sum_m \sum_{k,k'} \alpha_m(k)\psi_m(k,k')\beta_{m+1}(k') f_j(\boldsymbol{w}, k, k', m). \tag{12.77}$$

- Compute gradient as the difference between feature counts and expected counts, $\boldsymbol{f}(\boldsymbol{y}, \boldsymbol{w}) - E[\boldsymbol{f}(\boldsymbol{y}, \boldsymbol{w})]$.
- Update $\boldsymbol{\theta}$, using quasi-newton optimization or stochastic gradient descent. The `CRFsuite` package implements several of these learning algorithms (`http://www.chokkan.org/software/crfsuite/`).
- Iterate, recomputing the expected feature counts.

12.10 Unsupervised sequence labeling

In unsupervised sequence labeling, we want to induce a Hidden Markov Model (HMM) from a corpus of unannotated text $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_N$; this is an example of the general problem of **structure induction**, which is the unsupervised version of **structure prediction**. The tags that result from unsupervised sequence labeling might be useful for some downstream task, or for better understanding the language's inherent structure; or, we might want to do probability density estimation for sequences, as in gesture or activity recognition (Mitra and Acharya, 2007). Another reason would be to do semi-supervised learning, imputing tag sequences for unlabeled data. For part-of-speech tagging, often we use a tag dictionary which lists the allowed tags for each word, simplifying the problem (Christodoulopoulos et al., 2010).

In any case, we can perform unsupervised learning by applying expectation-maximization. In the M-step, we compute the HMM parameters from expected counts:

$$P_E(W = i|Y = k) = \phi_{k,i} = \frac{E[\text{count}(W = i, Y = k)]}{E[\text{count}(Y = k)]}$$
$$P_T(Y_m = k|Y_{m-1} = k') = \lambda_{k',k} = \frac{E[\text{count}(Y_m = k, Y_{m-1} = k')]}{E[\text{count}(Y_{m-1} = k')]}$$

The expected counts are computed in the E-step, using the forward and backward variables.

$$E[\text{count}(W = i, Y = k)] = \sum_m P(Y_m = k|\boldsymbol{w})\delta(W_m = i) \tag{12.78}$$
$$= \sum_m \frac{P(Y_m = k, \boldsymbol{w}_{1:m})\text{p}(\boldsymbol{w}_{m+1:M}|Y_m = k)}{\text{p}(\boldsymbol{w}_{1:M})}\delta(w_m = i) \tag{12.79}$$
$$= \frac{1}{\alpha_M(\square)} \sum_m \alpha_m(k)\beta_m(k)\delta(w_m = i) \tag{12.80}$$

We use the chain rule to separate $\boldsymbol{w}_{1:m}$ and $\boldsymbol{w}_{m+1:M}$, and then use the definitions of the forward and backward variables. In the final step, we normalize by $\text{p}(\boldsymbol{w}_{1:M}) = \alpha_M(\square) = \beta_0(\diamond)$.

$$
\begin{aligned}
E[\text{count}(Y_m = k, Y_{m-1} = k')] &= \sum_m P(Y_m = k, Y_{m-1} = k' | \boldsymbol{w}) && (12.81)\\
&\propto \sum_m P(Y_{m-1} = k', \boldsymbol{w}_{1:m-1}) P(w_{m+1:M} | Y_m = k) \\
&\quad \times P(w_m, Y_m = k | Y_{m-1} = k') && (12.82)\\
&= \sum_m P(Y_{m-1} = k', \boldsymbol{w}_{1:m-1}) P(w_{m+1:M} | Y_m = k) \\
&\quad \times \text{p}(w_m | Y_m = k) P(Y_m = k | Y_{m-1} = k') && (12.83)\\
&= \sum_m \alpha_{m-1}(k') \beta_m(k) \phi_{k,w_m} \lambda_{k' \rightarrow k} && (12.84)
\end{aligned}
$$

Again, we use the chain rule to separate out $\boldsymbol{w}_{1:m-1}$ and $\boldsymbol{w}_{m+1:M}$, and use the definitions of the forward and backward variables. The final computation also includes the parameters ϕ and λ, which govern (respectively) the emission and transition properties between w_m, y_m, and y_{m-1}. Note that the derivation only shows how to compute this to a constant of proportionality; we would divide by $\text{p}(\boldsymbol{w}_{1:M})$ to go from the joint probability $P(Y_{m-1} = k', Y_m = k, \boldsymbol{w}_{1:M})$ to the desired conditional $P(Y_{m-1} = k', Y_m = k \mid \boldsymbol{w}_{1:M})$.

Linear dynamical systems

The forward-backward algorithm can be viewed as Bayesian state estimation in a discrete state space. In a continuous state-space, $y_m \in \mathbb{R}$, the equivalent algorithm is the **Kalman Smoother**. It also computes marginals $\text{p}(y_m \mid \boldsymbol{x}_{1:M})$, using a similar two-step algorithm of forward and backward passes. Instead of computing a table of values at each step ($\alpha_m(k)$ and $\beta_m(k)$), they compute a probability density function, characterized by a mean and covariance around the latent state. Connections between the Kalman Smoother and the forward-backward algorithm are elucidated by Minka (1999).

Alternative unsupervised learning methods

Expectation-maximization is just one of many techniques for structure induction. One alternative is to use a family of randomized algorithms called **Markov Chain Monte Carlo (MCMC)**. In these algorithms, we compute a marginal distribution over the latent variable $\boldsymbol{y}$ **empirically**, by drawing random samples. The randomness explains the "Monte Carlo" part of the name; typically, we employ a Markov

Chain sampling procedure, meaning that each sample is drawn from a distribution that depends only on the previous sample (and not on the entire sampling history). A simple MCMC algorithm is **Gibbs Sampling**, in which we iteratively sample each y_m conditioned on all the others (Finkel et al., 2005):

$$\text{p}(y_m \mid \boldsymbol{y}_{-m}, \boldsymbol{w}_{1:M}) \propto \text{p}(w_m \mid y_m)\text{p}(y_m \mid \boldsymbol{y}_{-m}). \tag{12.85}$$

Gibbs Sampling has been applied to unsupervised part-of-speech tagging by Goldwater and Griffiths (2007). *Beam sampling* is a more complicated sampling algorithm, which randomly draws entire sequences $\boldsymbol{y}_{1:M}$, rather than individual tags y_m; this algorithm was applied to unsupervised part-of-speech tagging by Van Gael et al. (2009).

EM is guaranteed to find only a local optimum; MCMC algorithms will converge to the true posterior distribution $\text{p}(\boldsymbol{y}_{1:M} \mid \boldsymbol{w}_{1:M})$, but this is only guaranteed in the limit of infinite samples. Recent work has explored the use of **spectral learning** for latent variable models, which use matrix and tensor decompositions to provide guaranteed convergence under mild assumptions (Hsu et al., 2012). Georgia Tech faculty Byron Boots and Le Song are among the leaders in this active area of research (Song et al., 2010).

Chapter 13

Context-free grammars

So far we've explored finite-state models, which correspond to regular languages.

- **representations**: (weighted) finite state automata
- **probabilistic models**: HMMs (as a special case), CRFs
- **algorithms**: Viterbi, Forward-Backward, $\mathcal{O}(NK^2)$ time complexity.
- **linguistic phenomena**:
  - morphology
  - language models
  - part-of-speech disambiguation
  - named entity recognition (chunking)

Is the finite state representation enough for natural language?

13.1 Is English a regular language?

Regular languages are closed under intersection:

- $K \cap L$ is the set of strings in both K and L
- $K \cap L$ is regular iff K and L are regular

How to prove English is not regular:

- Let K be the set of grammatical English sentences
- Let L be some regular language
- Show that the intersection is not regular

We're going to prove this using center embedding:

1. *The cat is fat.*
2. *The cat that the dog chased is fat.*
3. **The cat that the dog is fat.*
4. *The cat that the dog that the monkey kissed chased is fat.*
5. **The cat that the dog that the monkey chased is fat.*

Proof sketch:

- K is the set of grammatical english sentences.
  It excludes sentences (3) and (5).
- L is the regular language *the cat* $(\textit{that } N)^+ V_t^+$ *is fat*.
- The language $L \cap K$ is *the cat* $(\textit{that } N)^n V_t^n$ *is fat*.

It is important to understand that the issue here is not just infinite repetition or productivity; FSAs can handle productive phenomena like *the big red smelly plastic figurine*. It is specifically the center-embedding phenomenon, because this leads to the same structure as the classic $a^n b^n$ language. What do you think of this argument?

Is deep center embedding really part of English?

Karlsson (2007) searched for multiple (phrasal) center embeddings in corpora from 7 languages:

- Very few examples of double embedding
- Only 13 examples of triple embedding (none in speech)
- Zero examples of quadruple embeddings

Note that we can build an FSA to accept center-embedding up to any finite depth.

Chomsky and many linguists distinguish between

- **Competence**: the fundamental abilities of the (idealized) human language processing system
- **Performance**: real utterances produced by speakers, subject to non-linguistic factors such as cognitive limitations

Even if English *as performed* is regular, the underlying generative grammar may be context-free... **or beyond**. There is a similar proof that at least some languages are not context-free! I'll post slides with this proof idea.

How much expressiveness do we need?

- Shieber (1985) makes a similar argument, showing that case agreement in Swiss-German cross-serial constructions is homomorphic to a formal language $wa^m b^n x c^m d^n y$, which is weakly non-context free. In response to the objection that all attested constructions are finite, Shieber writes:

  > Down this path lies tyranny. Acceptance of this argument opens the way to proofs of natural languages as regular, nay, **finite**.

- In practice, many real constructions are much simpler to handle in context-free rather than finite-state representations:

  > *The **processor has** 10 million times fewer transistors on it than todays typical microprocessors, **runs** much more slowly, and **operates** at five times the voltage...*

  – The easy way:

  $$\begin{aligned} \text{S} &\rightarrow \text{NN VP} \\ \text{VP} &\rightarrow \text{VP3S} \mid \text{VPN3S} \mid \ldots \\ \text{VP3S} &\rightarrow \text{VP3S, VP3S, } \textit{and} \text{ VP3S} \mid \text{VBZ} \mid \text{VBZ NP} \mid \ldots \end{aligned}$$

  – The hard way: build an FST that basically replicates all of English grammar for VPs with 3S and non-3S subjects.

- Mainstream parsing focuses on CFGs, but there is some work on "mildly" context-sensitive grammars.

13.2 Context-Free Languages

In the Chomsky hierarchy, context-free languages (CFLs) are a strict generalization of regular languages.

regular	**context-free**
regular expressions	context-free grammars (CFGs)
finite-state machines	pushdown automata
paths	derivations

Context-free grammars define CFLs. They are sets of permissible *productions* which allow you to **derive** strings composed of surface symbols.

$$
\begin{aligned}
S &\rightarrow NP\ VP_1 \\
NP &\rightarrow \textit{the}\ N \mid NP\ \textsc{RelClause} \\
\textsc{RelClause} &\rightarrow \textit{that}\ NP\ V_t \\
V_t &\rightarrow \textit{ate} \mid \textit{chased} \mid \textit{befriended} \mid \ldots \\
N &\rightarrow \textit{cat} \mid \textit{dog} \mid \textit{monkey} \mid \ldots \\
VP_1 &\rightarrow \textit{is fat}
\end{aligned}
$$

An important feature of CFGs is *recursion*, in which a nonterminal can be derived from itself.

More formally: a CFG is a tuple $\langle N, \Sigma, R, S \rangle$:

N	a set of non-terminals
Σ	a set of terminals (distinct from N)
R	a set of productions, each of the form $A \rightarrow \beta$, where $A \in N$ and $\beta \in (\Sigma \cup N)^*$
S	a designated start symbol

- Context free grammars provide rules for generating strings.
  - RHS: a non-terminal $\in N$
  - LHS: a sequence of terminals or non-terminals, $\{n, \sigma\}^*, n \in N, \sigma \in \Sigma$.
- A **derivation** t is a sequence of steps from S to a surface string $\boldsymbol{w} \in \Sigma^*$, which is the yield of the derivation.

- If $\exists t : s \in \text{yield}(t)$, then $\boldsymbol{w}$ is grammatical. Equivalently, for grammar G, we say $|\mathcal{T}_G(\boldsymbol{w})| \geq 1$.
- If $\exists t, t' : \boldsymbol{w} \in \text{yield}(t) \wedge s \in \text{yield}(t')$, then $\boldsymbol{w}$ is ambiguous. Equivalently, for grammar G, we say $|\mathcal{T}_G(\boldsymbol{w})| > 1$.
- A derivation can be viewed as trees or as bracketings, as shown in Figure 13.1.

Semantics Ideally, each derivation will have a distinct semantic interpretation, and all possible interpretations will be represented in some derivation.

$$(_{NP}(_{NP}\ \textit{Ban}\ (_{PP}\ \textit{on}\ (_{NP}\ \textit{nude dancing}\ )))\ (_{PP}\ \textit{on}\ (_{NP}\ \textit{Governor's desk}\ )))$$

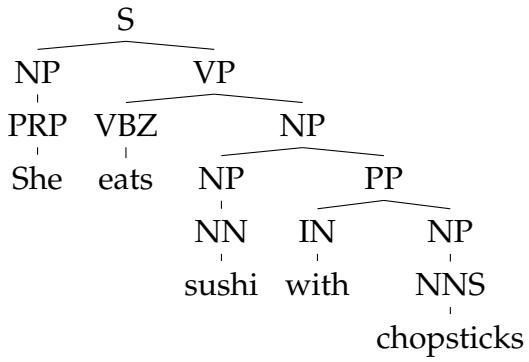
$$(_{NP}\ \textit{Ban}\ (_{PP}\ \textit{on}\ (_{NP}(_{NP}\ \textit{nude dancing}\ )\ (_{PP}\ \textit{on}\ (_{NP}\ \textit{Governor's desk}\ )))))$$

Sadly, this is not always the case.

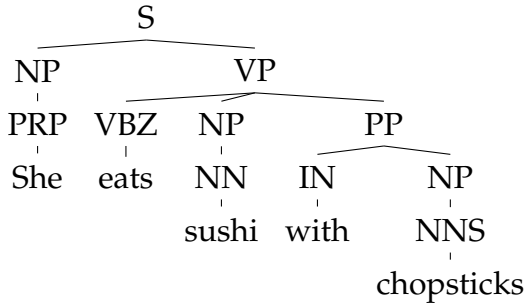
$$(_{NP}(_{JJ}\ \textit{nice})\ (_{JJ}\ \textit{little})\ (_{NN}\ \textit{car}\ ))$$
$$(_{NP}(_{JJ}\ \textit{nice})\ (_{NP}(_{JJ}\ \textit{little})\ (_{NN}\ \textit{car}\ )))$$
$$(_{NP}(_{JJ}\ \textit{nice})\ (_{NP}(_{JJ}\ \textit{little})\ (_{NP}(_{NN}\ \textit{car}\ ))))$$

13.3 Constituents

- In natural language grammars, the non-terminals should reflect syntactic categories.
- Bracketed substrings (e.g., *sushi with chopsticks*) are called **constituents**.
- There are several tests for constituency, including:
  - substitution
  - coordination
  - movement



$(_S(_{NP}(_{PRP}$ *She*$)(_{VP}(_{VBZ}$ *eats*$)$
$(_{NP}(_{NP}(_{NN}$ *sushi*$))(_{PP}$ *with*$(_{NP}(_{NNS}$ *chopsticks*$)))))))$



$(_S(_{NP}(_{PRP}$ *She*$)(_{VP}(_{VBZ}$ *eats*$)$
$(_{NP}(_{NN}$ *sushi*$))$
$(_{PP}$ *with*$(_{NP}(_{NNS}$ *chopsticks*$))))))$

Figure 13.1: Two derivations of the same sentence, shown as both parse trees and bracketings

Substitution Constituents generated by the same non-terminal should be substitutable in many contexts:

- $(_{NP}$ *The ban* $)$ *is on the desk.*
- $(_{NP}$ *The Governor's desk* $)$ *is on the desk.*
- $(_{NP}$ *The ban on dancing on the desk* $)$ *is on the desk.*

- *$(_{PP}$ *On the desk* $)$ *is on the desk.*

A more precise test for whether a set of substrings constitute a single category is whether they can be replaced by the same pronouns.

- $(_{NP}$ *It* $)$ *is on the desk.*

What about verbs?

- *I* $(_{V}$ *gave* $)$ *it to Anne.*
- *I* $(_{V}$ *taught* $)$ *it to Anne.*
- *I* $(_{V}$ *gave* $)$ *Anne a fish*
- **I* $(_{V}$ *taught* $)$ *Anne a fish*

This suggests we need nonterminals which distinguish verbs based on the arguments they can take. The technical name for this is *subcategorization*.

Coordination Constituents generated by the same non-terminal can usually be *coordinated* using words like *and* and *or*:

- *We fought* $(_{PP}$ *on the hills* $)$ *and* $(_{PP}$ *in the hedges* $)$.
- *We fought* $(_{ADVP}$ *as well as we could* $)$.
- **We fought* $(_{ADVP}$ *as well as we could* $)$ *and* $(_{PP}$ *in the hedges* $)$.

This too doesn't always work:

- *She* $(_{VP}$ *went* $)$ $(_{PP}$ *to the store* $)$.
- *She* $(_{VP}$ *came* $)$ $(_{PP}$ *from the store* $)$.
- *She* (*went to*) *and* (*came from*) *the store.*

Movement Valid constituents can be moved as a unit, preserving grammaticality.

- Passivization
  - *(The governor) banned (nude dancing on his desk)*
  - *(Nude dancing on his desk) was banned by (the governor)*
- Wh- movement
  - *(Nude dancing was banned) on (the desk).*
  - *(The desk) is where (nude dancing was banned)*
- Topicalization
  - *(He banned nude dancing) to appeal to conservatives.*
  - *To appeal to conservatives, (he banned nude dancing).*

13.4 A simple grammar of English

Noun phrases

Let's start with noun phrases:

- ***She*** *sleeps* (Pronoun)
- ***Arlo*** *sleeps* (Proper noun)
- ***Fish*** *sleep* (Mass noun)
- ***The fish*** *sleeps* (determiner + noun)
- ***The blue fish*** *sleeps* (DT + JJ + NN)
- ***The girl from Omaha*** *sleeps* (NP + PP)
- ***The student who ate 15 donuts*** *sleeps* (NP + RelClause)

So overall, we can summarize this fragment as

$$\begin{aligned} \text{NP} &\rightarrow \text{PRP} \mid \text{NNP} \mid \text{DT NOM} \\ \text{NOM} &\rightarrow \text{ADJP NOM} \mid \text{NN} \\ \text{NP} &\rightarrow \text{NP PP} \mid \text{NP RELCLAUSE} \end{aligned}$$

We're leaving out some detail, like pluralization and possessives, but you get the idea.

Adjectival and prepositional phrases

- *Very funny*
- *The **large, blue** fish*
- *The man **from la mancha***

$$\begin{aligned} \text{ADJP} &\rightarrow \text{JJ} \mid \text{RB ADJP} \mid \text{JJ ADJP} \\ \text{PP} &\rightarrow \text{IN NP} \mid \text{TO NP} \end{aligned}$$

Verb phrases

- *She **sleeps***
- *She **sleeps restlessly***
- *She **sleeps at home***
- *She **eats sushi***[1]
- *She **gives John sushi***

$$\text{VP} \rightarrow \text{V} \mid \text{VP RB} \mid \text{VP PP} \mid \text{V NP} \mid \text{V NP NP} \mid \text{V NP RB}$$

But what about **She sleeps sushi* or **She speaks John Japanese*?

- Classes of verbs can take different numbers of arguments.

[1] Sushi examples from Julia Hockenmaier

- This is called **subcategorization**

$$\text{VP} \rightarrow \text{V-INTRANS} \mid \text{V-TRANS NP} \mid \text{V-DITRANS NP NP}$$
$$\text{VP} \rightarrow \text{VP RB} \mid \text{VP PP}$$

We would also need to handle modal and auxiliary verbs that allow us to create complex tenses, like *She will have eaten sushi* but not **She will have eats sushi*.

Sentences

- *She eats sushi*

$$\text{S} \rightarrow \text{NP VP}$$

- *Sometimes, she eats sushi*

$$\text{S} \rightarrow \text{ADVP S}$$

- *In Japan, she eats sushi*

$$\text{S} \rightarrow \text{PP S}$$

- What about **I eats sushi*, **She eat sushi*??

$$\text{S} \rightarrow \text{NP.3S VP.3S} \mid \text{NP.N3S VP.N3S}$$

  In general, we need **features** to capture this kind of agreement.

Conjunctions

- *She eats sushi and candy*

$$\text{NP} \rightarrow \text{NP } \textit{and} \text{ NP}$$

- *She eats sushi and drinks soda*

$$\text{VP} \rightarrow \text{VP } \textit{and} \text{ VP}$$

- *She eats sushi and he drinks soda*

$$\text{S} \rightarrow \text{S } \textit{and} \text{ S}$$

- *fresh and tasty sushi*

$$\text{ADJP} \rightarrow \text{JJ } \textit{and} \text{ JJ}$$

We'd need a little more cleverness to properly cover groups larger than two.

Odds and ends

- *I gave sushi to the girl **who eats sushi**.* This is a relative clause,

$$\text{RELCLAUSE} \rightarrow \textit{who}\ \text{VP} \mid \textit{that}\ \text{VP}$$

- *I took sushi from the man **offering sushi**.* This is a gerundive postmodifier.

$$\text{NOM} \rightarrow \text{NOM GERUNDVP}$$
$$\text{GERUNDVP} \rightarrow \text{VBZ} \mid \text{VBZ NP} \mid \text{VBZ PP} \mid \ldots$$

- ***Can** she eat sushi?* (notice it's not *eats*)

$$\text{S} \rightarrow \text{AUX NP VP}$$

- ... and many more

13.5 Grammar design

Our goal is a grammar that avoids

- **Overgeneration**: deriving strings that are not grammatical.
- **Undergeneration**: failing to derive strings that are grammatical.

To avoid undergeneration, we would need thousands of productions.

Typically, grammars are defined in conjunction with large-scale **treebank** annotation projects.

- An annotation guideline specifies the non-terminals and how they go together.
- The annotators then apply these guidelines to data.
- The grammar rules can then be read off the data.

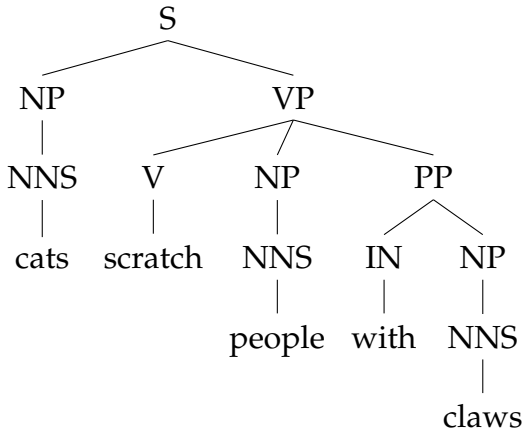
The Penn Treebank (PTB) contains one million parsed words of Wall Street Journal text (Marcus et al., 1993).

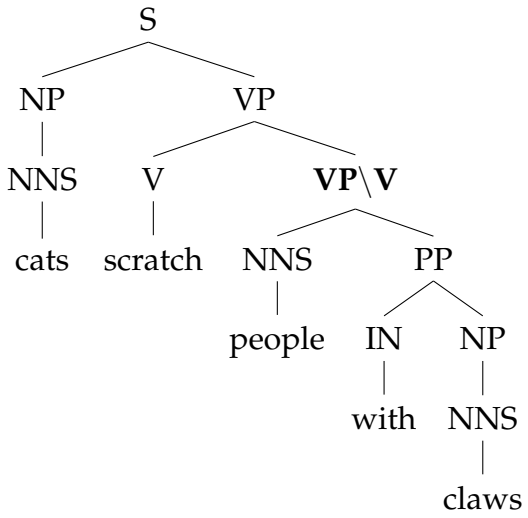
13.6 Grammar equivalence and normal form

- Grammars are **weakly equivalent** if they generate the same strings.
- Grammars are **strongly equivalent** if they generate the same strings **and** assign the same phrase structure to each string.
- In Chomsky Normal Form (CNF), all productions are either:

$$A \rightarrow BC$$
$$A \rightarrow a$$

  - All CFGs can be converted into a weakly equivalent grammar in CNF.
  - This is very handy for parsing algorithms.





- Binarization is easy:
  group right children into new non-terminals.
- Un-binarization is important!
  people with claws is not a constituent in the original parse.
- Unary productions are best handled by modifying the algorithm.

13.7 Parsing

Parsing is the process of determining whether a sentence is in a context-free language, by searching for a legal derivation. Some possibilities:

- **Top-down**: start with the start symbol, and see if we can derive the sentence.
- **Bottom-up**: combine the observed symbols using whatever productions we can, until we reach the start symbol
- **Left-to-right**: move through the input, incrementally building a parse tree

Before we get into these different possibilities, let's see whether exhaustive search is possible. Suppose we only have one non-terminal, X, and it has binary productions

$$\begin{aligned} \text{X} &\rightarrow \text{X X} \\ \text{X} &\rightarrow \textit{the girl} \mid \textit{ate sushi} \mid \ldots \end{aligned}$$

How many different ways could we parse a sentence? This is just equal to the number of binary bracketings of the words in the sentence, which is a Catalan number. Catalan numbers grow **super-exponentially** in the length of the sentence, $C_n = \frac{(2n)!}{(n+1)!n!}$.

13.8 CKY parsing

CKY is a bottom-up parsing allows us to test whether a sentence is in a context-free language, without considering all possible parses. First we form small constituents, then try to merge them into larger constituents.

Let's start with an example grammar:

$$\begin{aligned}
\text{S} &\rightarrow \text{VP NP} \\
\text{NP} &\rightarrow \text{NP PP} \mid \textit{we} \mid \textit{sushi} \mid \textit{chopsticks} \\
\text{PP} &\rightarrow \text{P NP} \\
\text{P} &\rightarrow \textit{with} \\
\text{VP} &\rightarrow \text{VP NP} \mid \text{VP PP} \mid \textit{eat}
\end{aligned}$$

Suppose we encounter the sentence *We eat sushi with chopsticks*.

- The first thing that we notice is that we can apply unary productions to obtain NP VP NP P NP
- Next, we can apply a binary production to merge the first NP VP into an S.
- Or we could merge VP NP into VP
- ... and so on

Let's systematize this. Here is the CKY algorithm:

Algorithm 1 The CKY algorithm for CFG parsing

for j : [1,N] **do**
  $t[j, j-1] \leftarrow \{X \mid X \rightarrow w_j \in R\}$
  for i : [j-2, 0] **do**
    for k : [i+1, j-1] **do**
      $t[i, j] \leftarrow t[i, j] \cup \{X \mid X \rightarrow YZ \in R, Y \in t[i, k], Z \in t[k, j]\}$
    end for
  end for
end for

Note that this algorithm assumes that all productions with non-terminals on the RHS are binary. What about unary productions like S $\rightarrow$ VP $\rightarrow$ V $\rightarrow$ *eat*? To handle this case, we compute the *unary closure* of each non-terminal.

- e.g., if $S \rightarrow VP$, $VP \rightarrow V$, then add $S \rightarrow V$
- At each table entry $t[i, j]$
  - For each non-terminal $A \in t[i, j]$
    - Add all elements of the reflexive unary closure for A

Complexity What is the complexity of CKY?

- Space compexity: $\mathcal{O}(M^2|N|)$
- Time complexity: $\mathcal{O}(M^3|R|)$
- M is length of sentence,
  $|N|$ is the number of non-terminals,
  $|R|$ is the number of production rules
- But in practice... It's worse than worst-case! (Figure 13.2)
- Because longer sentences "unlock" more of the grammar.

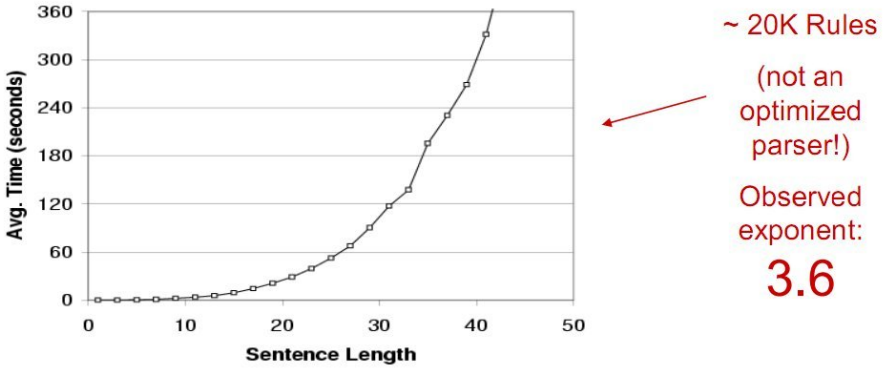


Figure 13.2: Figure from Dan Klein's lecture slides

13.9 Ambiguity in parsing

- Syntactic ambiguity is endemic to natural language:[2]
  - Attachment ambiguity: *we eat sushi with chopsticks*,
    I shot an elephant in my pajamas.
  - Modifier scope: *southern food store*
  - Particle versus preposition: *The puppy tore up the staircase.*

[2]Examples borrowed from Dan Klein

  - Complement structure: *The tourists objected to the guide that they couldn't hear.*
  - Coordination scope: *"I see," said the blind man, as he picked up the hammer and saw.*
  - Multiple gap constructions: *The chicken is ready to eat*

- In morphology, we didn't just want to know which derivational forms are *legal*, we wanted to know which were likely.
- Syntactic parsing is all about choosing among the many, many legal parses for a given sentence.

Here's another example, which we've seen before:

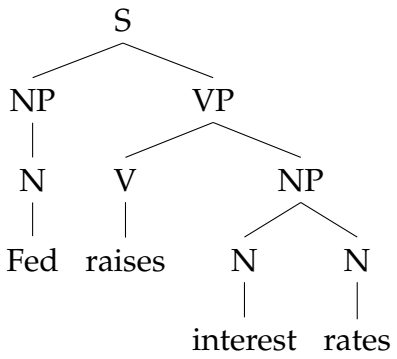


Figure 13.3: An example parse

- A minimal grammar permits 36 parses!
- Real-size broad coverage grammars permit millions of parses of typical sentences.
- Collins (2013) gives another example, *She announced a program to promote safety in trucks.*

Classical parsers faced a tradeoff:

- broad coverage with tons of ambiguity...
- or limited coverage in exchange for constraints on ambiguity

Consequently, deterministic parsers produced no analysis for many sentences.

Local solutions

Some ambiguity can be resolved locally:

- [*imposed* [*a ban* [*on asbestos*]]]
- [*imposed* [*a ban*][*on asbestos*]]
- Hindle and Rooth (1990) proposed a likelihood ratio test:

$$LR(v, n, p) = \frac{\text{p}(p \mid v)}{\text{p}(p \mid n)} = \frac{\text{p}(\textit{on} \mid \textit{imposed})}{\text{p}(\textit{on} \mid \textit{ban})}$$

  where we select VERB attachment if $LR(v, n, p) > 1$.

- But the likelihood-ratio approach ignores important information, like the phrase being attached.
  - ...[*it* [*would end* [*its venture* [*with Maserati*]]]]
  - ...[*it* [*would end* [*its venture*][*with Maserati*]]]
- The likelihood ratio gets this wrong
  - $\text{p}(\textit{with} \mid \textit{end}) = \frac{607}{5156} = 0.118$
  - $\text{p}(\textit{with} \mid \textit{venture}) = \frac{155}{1442} = 0.107$

  Other features (e.g., *Maserati*) argue for noun attachment. How can we add them?

Machine learning solutions Ratnaparkhi et al. (1994) propose a classification-based approach, using logistic regression (maximum-entropy):

$$P(\text{Noun attachment} \mid \textit{would end its venture with Maserati}) = \frac{e^{\boldsymbol{\theta}^\top \boldsymbol{f}(\textit{would end its venture with Maserati})}}{1 + e^{\boldsymbol{\theta}^\top \boldsymbol{f}(\textit{would end its venture with Maserati})}}$$

Features include n-grams and word classes from hierarchical word clustering; accuracy is roughly 80%.

Collins and Brooks (1995) argued that attachment depends on four **heads**:

- the preposition (*with*)

- the VP attachment site (*end*)
- the NP attachment site (*venture*)
- the NP to be attached (*Maserati*)

They propose a backoff-based approach:

- First, look for counts of the tuple $\langle$*with, Maserati, end, venture*$\rangle$
- If none, try $\langle$*with, Maserati, end*$\rangle$ + $\langle$*with,end,venture*$\rangle$ + $\langle$*with,Maserati, venture*$\rangle$
- If none, try $\langle$*with, Maserati*$\rangle$ + $\langle$*with,end*$\rangle$ + $\langle$*with,venture*$\rangle$
- If none, try $\langle$*with*$\rangle$

Accuracy is roughly 84%. This approach of combining relative frequency estimation, smoothing, and backoff was very characteristic of late 1990s statistical NLP.

Beyond local solutions

Framing the problem as attachment ambiguity is limiting:

- assumes the parse is mostly done, leaving just a few attachment ambiguities to solve
- But realistic sentences have more than a few syntactic interpretations.
- Attachment decisions are interdependent:
  - *Cats scratch people with claws with knives.*
  - We may want to attach *with claws* to *scratch*.
  - But then we have nowhere to put *with knives*.

The task of statistical parsing is to produce a single analysis that resolves all syntactic ambiguities.

S	→ NP VP	0.9
S	→ S CC S	0.1
NP	→ N	0.2
NP	→ DT N	0.3
NP	→ N NP	0.2
NP	→ JJ NP	0.2
NP	→ NP PP	0.1
VP	→ V	0.4
VP	→ V NP	0.3
VP	→ V NP NP	0.1
VP	→ VP PP	0.2
PP	→ P NP	1.0

Table 13.1: A fragment of an example probabilistic context-free grammar (PCFG)

13.10 Statistical parsing with PCFGs

We want the parse τ that maximizes $\text{p}(\tau \mid \boldsymbol{w})$.

$$\begin{aligned}
\arg\max_{\tau} \text{p}(\tau \mid \boldsymbol{w}) &= \arg\max_{\tau} \frac{\text{p}(\tau, \boldsymbol{w})}{\text{p}(\boldsymbol{w})} \\
&= \arg\max_{\tau} \text{p}(\tau, \boldsymbol{w}) \\
&= \arg\max_{\tau} \text{p}(\boldsymbol{w} \mid \tau)\text{p}(\tau) \\
&= \arg\max_{\tau : \boldsymbol{w} = \text{yield}(\tau)} \text{p}(\tau)
\end{aligned}$$

- The **yield** of a tree is the string of terminal symbols that can be read off the leaf nodes.
- The set $\{\tau : \boldsymbol{w} = \text{yield}(\tau)\}$ is exactly the set of all derivations of $\boldsymbol{w}$ in a CFG G.

PCFGs extend the CFG by adding probability to each production, as shown in Table 13.1.

The probabilities for all productions involving a single LHS must sum to 1:

$$\sum_{\alpha} P(X \rightarrow \alpha \mid X) = 1$$

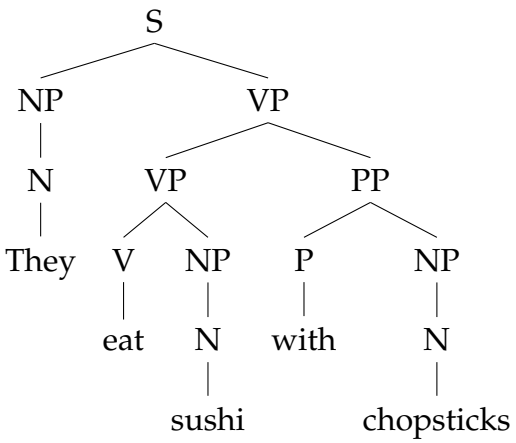


Figure 13.4: An alternative parse of our running example

The probability $\text{p}(\tau)$ is just the product of all the productions. Consider the example in Figure 13.4, which we will score using the probabilities in Table 13.1.

The probability of this parse is:

$$\begin{aligned}
\text{p}(\tau, \boldsymbol{w}) =& P(\text{S} \rightarrow \text{NP VP}) && (13.1)\\
&\times P(\text{NP} \rightarrow \text{N}) \times P(\text{N} \rightarrow \textit{they}) && (13.2)\\
&\times P(\text{VP} \rightarrow \text{VP PP}) && (13.3)\\
&\times P(\text{VP} \rightarrow \text{V NP}) \times P(\text{V} \rightarrow \textit{eat}) && (13.4)\\
&\times P(\text{NP} \rightarrow \text{N}) \times P(\text{N} \rightarrow \textit{sushi}) && (13.5)\\
&\times P(\text{PP} \rightarrow \text{P NP}) \times P(\text{P} \rightarrow \textit{with}) && (13.6)\\
&\times P(\text{NP} \rightarrow \text{N}) \times P(\text{N} \rightarrow \textit{chopsticks}) && (13.7)\\
=& 0.9 \times 0.2 \times 0.2 \times 0.3 \times 0.2 \times 1.0 \times 0.2 && (13.8)\\
&\times \text{probability of terminal productions} && (13.9)
\end{aligned}$$

Estimation

- As in supervised HMMs, estimation is easy (for now!).
- PCFG probabilities can be estimated directly from a treebank:

$$P(\text{VP} \rightarrow \text{VP PP}) = \frac{\text{count}(\text{VP} \rightarrow \text{VP PP})}{\text{count}(\text{VP})}$$

Three basic problems for PCFGs

Let $\tau \in T$ be a derivation, $\boldsymbol{w}$ be a sentence, and λ a PCFG.

- **Decoding**: Find $\hat{\tau} = \arg\max_\tau \text{p}(\tau, \boldsymbol{w}; \lambda)$
- **Likelihood**: Find $\text{p}(\boldsymbol{w}; \lambda) = \sum_\tau \text{p}(\tau, \boldsymbol{w}; \lambda)$
- **(Unsupervised) Estimation**: Find $\arg\max_\lambda \text{p}(\boldsymbol{w}_{1...N} \mid \lambda)$

These three problems are analogous to the problems identified by Rabiner (1989) for Hidden Markov Models.

	Sequences	Trees
model	HMM	PCFG
decoding	Viterbi algorithm	CKY
decoding complexity	$\mathcal{O}(M^2\|K\|)$	$\mathcal{O}(M^3\|R\|)$
likelihood	forward algorithm	inside algorithm
marginals	forward-backward	inside-outside

CKY with probabilities

It is not difficult to extend CKY to include probabilities or other weights. This is shown in Algorithm 2.

Algorithm 2 CKY with weighted productions

for j : [1,N] **do**
  for X : $tags(w_j)$ **do**
    $t[X, j, j-1] \leftarrow P(X, w_j)$
  end for
  for i : [j-2, 0] **do**
    for $(X \rightarrow Y\ Z) \in R$ **do**
      for k : [i+1, j-1] **do**
        $t[X, i, j] \leftarrow t[X, i, j] \bigoplus (P(X \rightarrow Y\ Z) \otimes t[Y, i, k] \otimes t[Z, k+1, j])$
      end for
    end for
  end for
end for

In the boolean semiring, we have $\oplus = \vee$ and $\otimes = \wedge$, so we recover the original CKY algorithm. In the probability semiring, we have $\oplus = \max$ and $\otimes = \times$.

$$t[Y, i, k] = P(Y \rightarrow \boldsymbol{w}_{i:k}) \tag{13.10}$$
$$t[Z, k, j] = P(Z \rightarrow \boldsymbol{w}_{k+1:j}) \tag{13.11}$$
$$t[X, i, j] = \max_{Y,Z,k} P(X \rightarrow Y\ Z) P(Y \rightarrow \boldsymbol{w}_{i:k}) P(Z \rightarrow \boldsymbol{w}_{k+1:j}) \tag{13.12}$$
$$\tag{13.13}$$

The **inside algorithm** computes the probability of producing a span of text $\boldsymbol{w}_{i:j}$ from a non-terminal X. To do this, we move to a semiring where $\oplus = +$,

$$t[X, i, j] = \sum_{Y,Z,k} P(X \rightarrow Y\ Z) P(Y \rightarrow \boldsymbol{w}_{i:k}) P(Z \rightarrow \boldsymbol{w}_{k+1:j}) \tag{13.14}$$
$$= P(X \rightarrow \boldsymbol{w}_{i:j}). \tag{13.15}$$

13.11 Algorithms for PCFG Parsing

PCFGs score the probability of a derivation $P(\tau, \boldsymbol{w})$ as the product of all productions in τ

- CKY is a bottom-up algorithm for finding $\hat{\tau} = \arg\max_\tau \mathrm{p}(\tau, \boldsymbol{w})$.
- The **inside algorithm** finds $\mathrm{p}(\boldsymbol{w}) = \sum_\tau \mathrm{p}(\tau, \boldsymbol{w})$.
- These algorithms are related through semiring notation.
- These algorithms are bottom-up: they parse progressively larger spans until the entire sentence is parsed. This is efficient, but it is pretty implausible as a model of human parsing, since it seems unrelated to the way we hear and read language: left-to-right.

Shift-reduce is a left-to-right parsing algorithm, which you may find more cognitively plausible.

- It is related to the pushdown automata representation of context-free grammars: we move through the sentence while keeping a stack with infinite depth.

- At each step, we have two choices
  - **shift** the next word on to the stack
  - **reduce** the stack by applying some production
- Each reduce move is a production in the derivation.
- If we can clear all the input and end up with just S on the stack, we have parsed the sentence correctly.

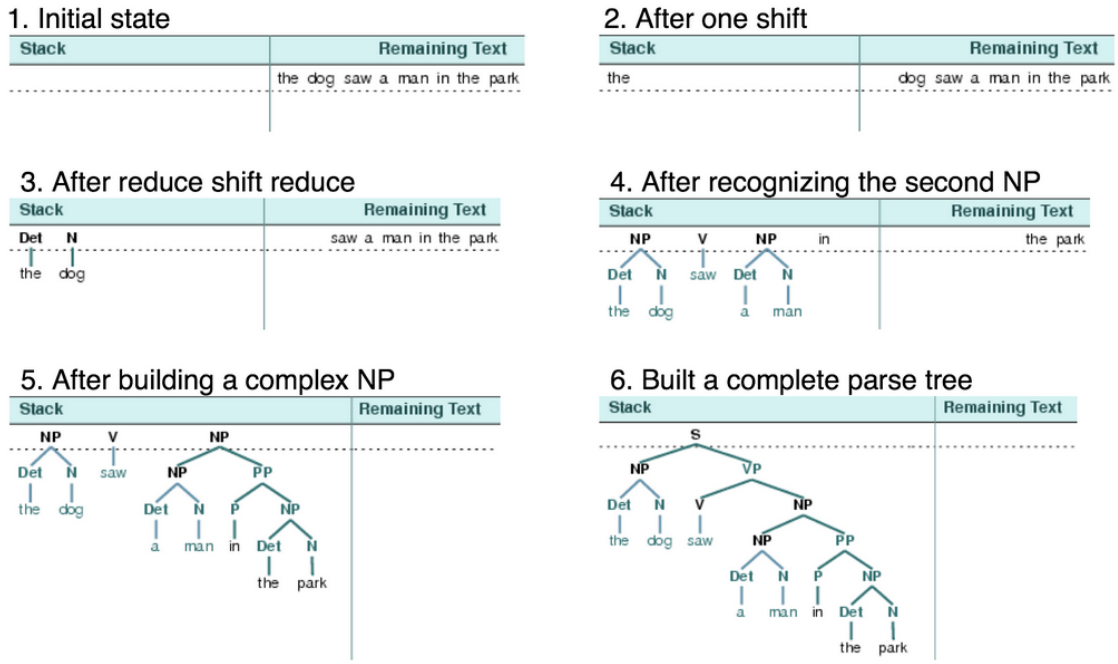


Figure 13.5: Example of shift-reduce CFG parsing, from Bird et al. (2009)

How do we decide whether to shift or reduce?

- We could treat this as a classic search problem, and just backtrack when we get into trouble.
- Or, we could train a classifier to decide between shift and reduce. Note that we have a separate reduce action for each non-terminal in the grammar.

13.12 Parser evaluation

Before continuing to more advanced parsing algorithms, we need to consider how to measure parsing performance. Suppose we have a set of **reference parses** — the ground truth — and a set of **system parses** that we would like to score. A simple solution would be **per-sentence accuracy**: the parser is scored by the proportion of sentences on which the system and reference parses exactly match.[3] But we would like to assign *partial credit* for correctly matching parts of the reference parse. The PARSEval metrics do that, scoring each system parse via:

Precision, the fraction of brackets in the system parse that match a bracket in the reference parse.

Recall, the fraction of brackets in the reference parse that match a bracket in the system parse.

In labeled precision and recall, it is required to also match the non-terminals for each bracket; in unlabeled precision and recall, it is only required to match the bracketing structure. The F-measure is the harmonic mean of precision and recall.

In Figure 13.1, suppose the top tree is the system parse and the bottom tree is the reference parse. We have the following spans:

- S $\rightarrow \boldsymbol{w}_{1:5}$: true positive
- VP $\rightarrow \boldsymbol{w}_{2:5}$: true positive
- NP $\rightarrow \boldsymbol{w}_{3:5}$: false positive
- PP $\rightarrow \boldsymbol{w}_{4:5}$: true positive

So for this parse, we have a (labeled and unlabeled) precision of $\frac{3}{4} = 0.75$, and a recall of $\frac{3}{3} = 1.0$, for an F-measure of 0.86. The best automatic CFG parsers get an F-score of approximately 0.92 on the Penn Treebank (PTB) today (McClosky et al., 2006).

[3] Most parsing papers do not report results on this metric, but Finkel et al. (2008) find that a near-state-of-the-art parser finds the exact correct parse on 35% of sentences of length ≤ 40, and on 62% of parses of length ≤ 15 in the Penn Treebank.

13.13 Improving PCFG parsing

Regardless of the parsing algorithm, pure PCFG parsing on Penn Treebank nonterminals (e.g., NP, VP) doesn't work well: a PCFG build from treebank probabilities scores $F = 0.72$. Why?

Problems with PCFG parsing

Substitutability Recall that substitutability is a criterion for constituency. Are NPs really substitutable? No, because some pronouns cannot be both subjects and objects (Figure 13.6).

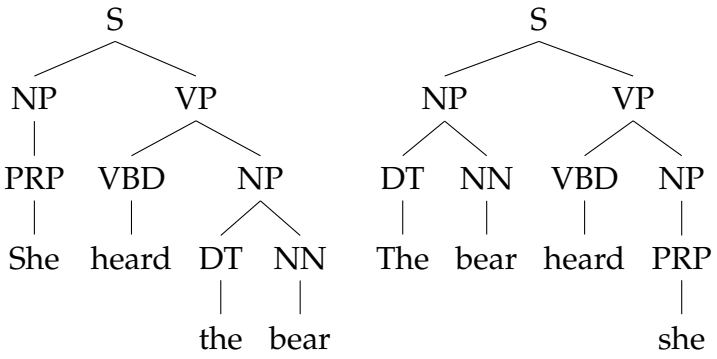


Figure 13.6: A grammar that allows *she* to take the object position is overgenerating.

We might address this problem by **splitting** the NP tag into nominitive (*she*) and oblique (*her*) cases, but this distinction is only relevant for pronouns: other nouns can appear in either position.

A related point is that we have no flexibility on PP attachment. If $P(\text{NP} \rightarrow \text{NP PP}) > P(\text{VP} \rightarrow \text{VP PP})$, we will always prefer NP attachment; if not, we will always prefer VP attachment. More fine-grained NP and VP categories might allow us to make attachment decisions more accurately.

Semantic preferences In addition to grammatical constraints such as case marking, we have semantic preferences: for example, that conjoined entities should be similar (Figure 13.7).

Note that no PCFG can distinguish the parses in Figure 13.7! They contain exactly the same productions.

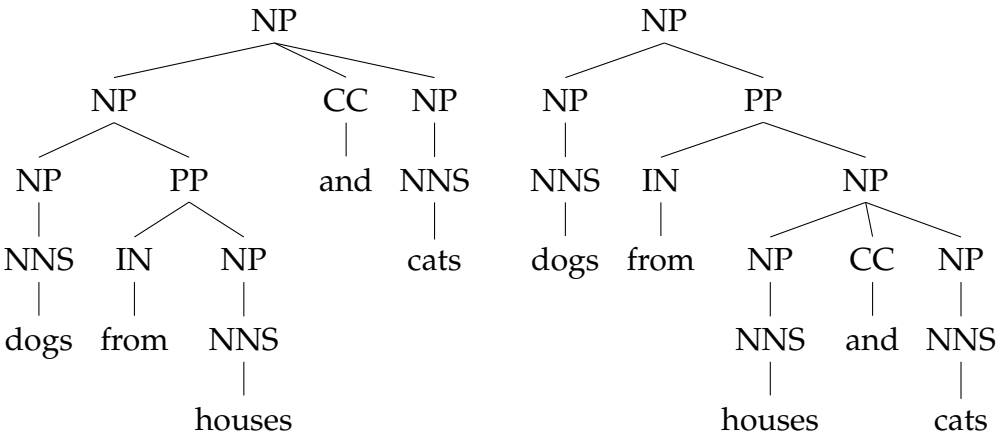
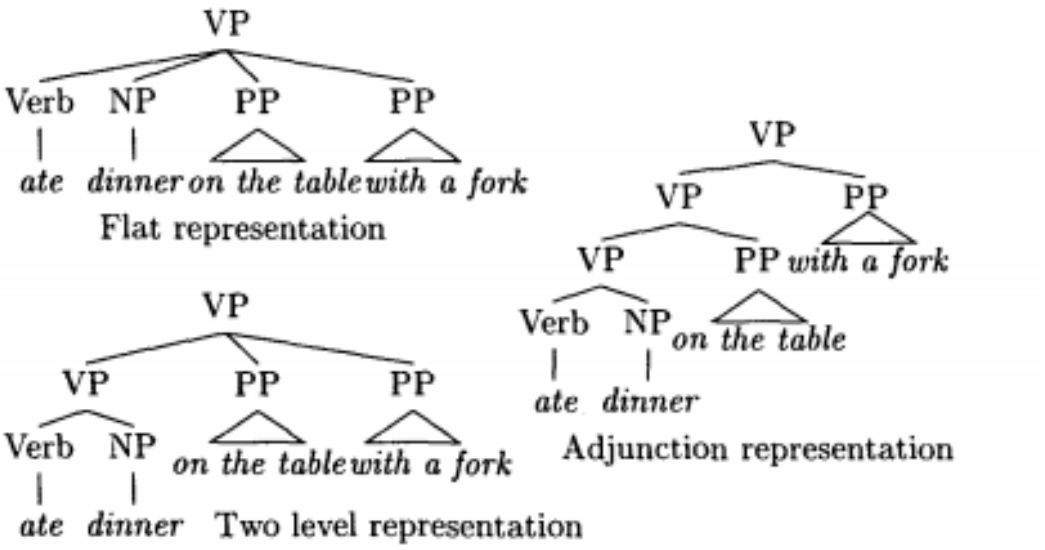


Figure 13.7: The first parse is arguably preferable because of the conjunction of phrases headed by *dogs* and *cats*. Example from Dan Klein's lecture slides.

Subsumption There are several choices for annotating PP attachment



Johnson (1998) shows that even though the two-level representation is chosen in the annotation, it can never be produced by a PCFG because the production is

subsumed.

$$P(\text{NP} \rightarrow \text{NP PP}) = 0.112$$
$$P(\text{NP} \rightarrow \text{NP PP PP}) = 0.006$$
$$P(\text{NP} \rightarrow \text{NP PP})P(\text{NP} \rightarrow \text{NP PP}) = (0.112)^2 = 0.013$$

The probability of applying the NP $\rightarrow$ NP PP production twice is greater than the probability of the two-PP production, so this production will never appear in a PCFG parse. Johnson shows that 9% of all productions are subsumed and can be removed from the grammar!

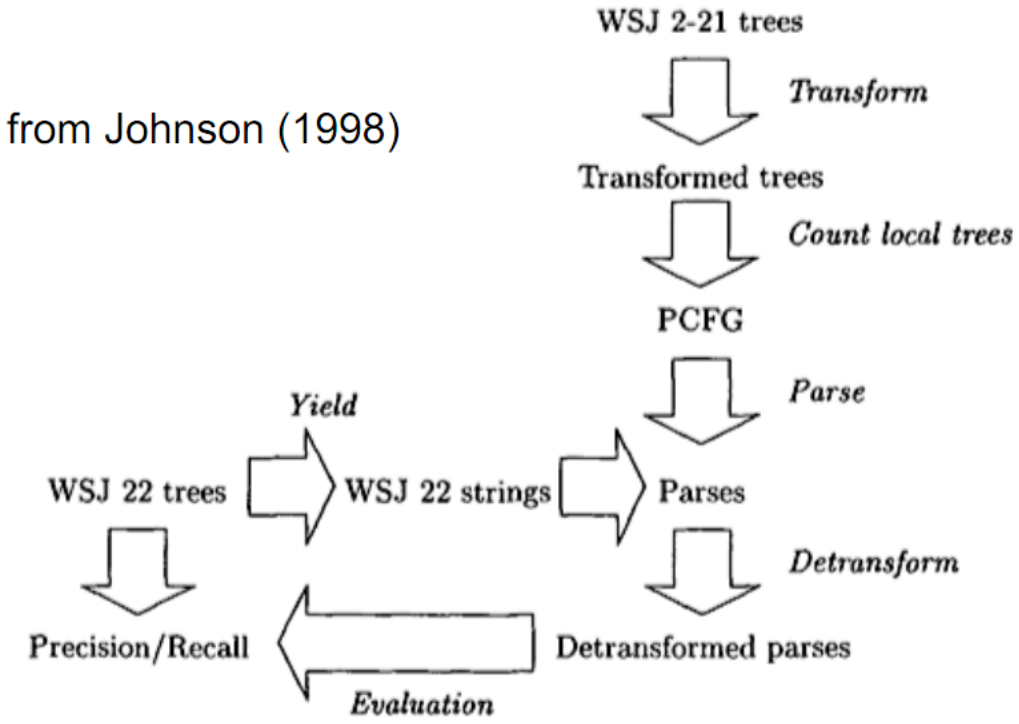
Refining PCFG parsing Modern generative parsing algorithms automatically refine these nonterminals in various ways.

- **Tree transformations**: automatically modify the parse trees — for example, Markovizing by labeling each non-terminal with its **parent**, as in NP-S.

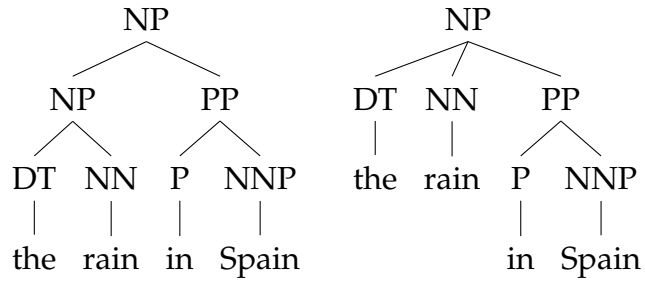
- **Lexicalization**: label each non-terminal with its head **word**

Tree transformations

Johnson proposed a series of transformations to PTB trees that improve parsing accuracy.



Flattening Johnson (1998) proposes "flattening" nested NPs to be more like VP structures.



Flattened rules are of course still context-free, but by reducing recursion, they allow more specific probabilities to be learned. This can eliminate the problems with rule subsumption that we saw earlier.

Parent annotation The expansion of an NP is highly dependent on its parent.

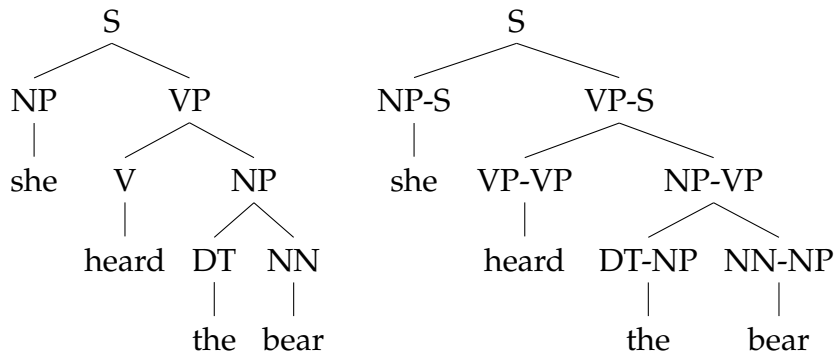
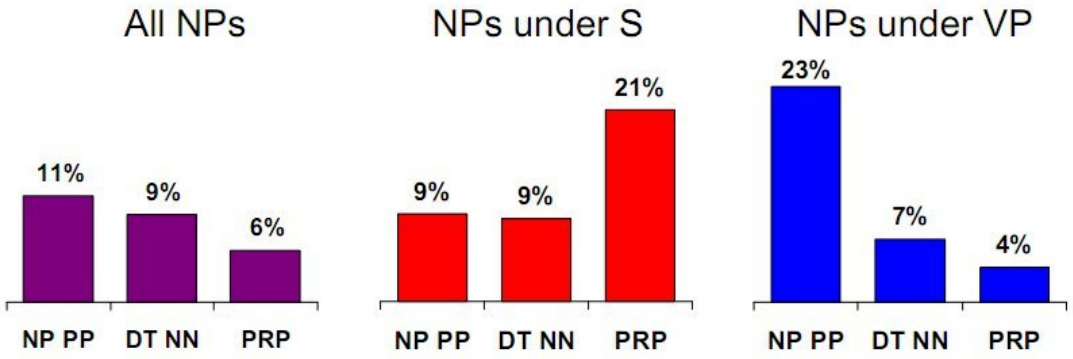


Figure 13.8: Parent annotation, sometimes called vertical Markovization (Klein and Manning, 2003).



$$P(\text{NP} \rightarrow \text{NP PP}) = 11\%$$
$$P(\text{NP-S} \rightarrow \text{NP PP}) = 9\%$$
$$P(\text{NP-VP} \rightarrow \text{NP PP}) = 23\%$$

PP adjunction is more likely for NPs that are under verb phrases (typically in object position) than for NPs that are under sentences (typically in subject position). We can capture this phenomenon via **parent annotation**: augmenting each nonterminal with the identity of its parent (Figure 13.8).

Parent annotation weakens the PCFG independence assumptions

- which could help accuracy by allowing more fine-grained distinctions
- or could hurt accuracy because of data sparseness

Overall, the transformations proposed by Johnson (1998) improve performance on PTB parsing.

- Standard PCFG: 72% F-measure, 14,962 rules
- Parent-annotated PCFG: 80% F-measure, 22,773 rules
- In principle, parent annotation could have increased the grammar size much more dramatically, but many possible productions never occur, or are subsumed.

Lexicalization

A simple way to capture semantics is through the words themselves. We can annotate each non-terminal with **head** word of the phrase.

Head words are deterministically assigned according to a set of rules, sometimes called **head percolation rules**. In many cases, these rules are straightforward: the head of a NP $\rightarrow$ DTN production is the noun, the head of a S $\rightarrow$ NPVP production is the head of the VP, etc. But as always, there are a lot of special cases. Collins (2013) offers the following example for productions whose LHS is NP:

- **If** the RHS contains NN, NNS, or NNP, **then** choose the rightmost NN, NNS, or NNP.
- **Else If** the RHS contains an NP, **then** choose the leftmost NP
- **Else If** the rule contains a JJ, **then** choose the rightmost JJ (e.g., *Sandra is **the best***)
- **Else If** the rule contains a CD (cardinal number), **then** choose the rightmost CD (e.g., *Marco is 27.*)
- **Else** choose the rightmost child.

A fragment of the head percolation rules used in many parsing systems are found in Table 13.2.[4]

The meaning of these rules is that to find the head of an S constituent, we first look for the rightmost VP child; if we don't find one, we look for the rightmost SBAR child, and so on down the list. Verb phrases are headed by left verbs (the

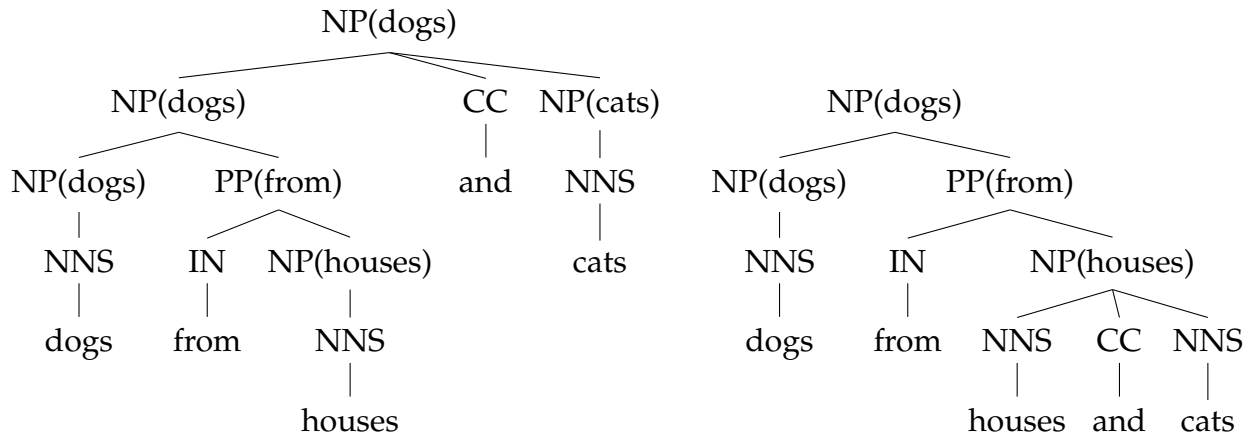
[4] From `http://www.cs.columbia.edu/~mcollins/papers/heads`

Non-terminal	Direction	Priority
S	right	VP SBAR ADJP UCP NP
VP	left	VBD VBN MD VBZ TO VB VP VBG VBP ADJP NP
NP	right	N* EX $ CD QP PRP ...
PP	left	IN TO FW

Table 13.2: A fragment of head percolation rules

head of *can walk home* is *walk*, since *can* is tagged MD), noun phrases are readed by the rightmost noun-like non-terminal (so the head of *the red cat* is *cat*), and prepositional phrases are headed by the preposition (the head of *at Georgia Tech* is *at*). Some of these rules are somewhat arbitrary — there's no particular reason why the head of *cats and dogs* should be *dogs* — but the point here is just to get some lexical information that can support parsing, not to make any deep claims about syntax.

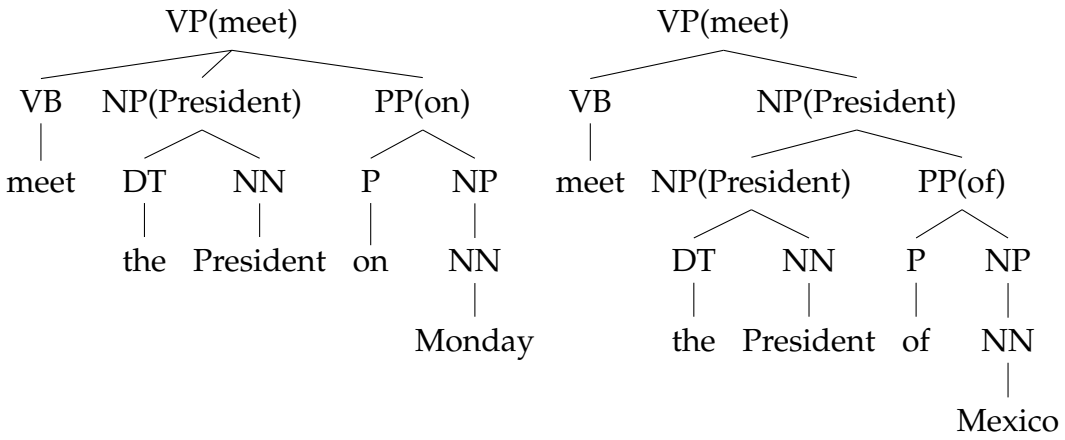
Given these rules, we can lexicalize the parse trees for some of our examples:



Example: coordination scope If $P(\text{NP} \rightarrow \text{NP}(\textit{dogs})\ \text{CC}\ \text{NP}(\textit{cats})) > P(\text{NP} \rightarrow \text{NP}(\textit{houses})\ \text{CC}\ \text{NP}(\textit{cats}))$, we should get the right parse.

Example: PP attachment

- $P(\text{VP}(\textit{meet}) \rightarrow \alpha\ \text{PP}(\textit{on})) \gg P(\text{NP}(\textit{President}) \rightarrow \beta\ \text{PP}(\textit{on}))$



- $P(\text{VP}(\textit{meet}) \rightarrow \alpha\ \text{PP}(\textit{of})) \ll P(\text{NP}(\textit{President}) \rightarrow \beta\ \text{PP}(\textit{of}))$
- In plain English: *Meeting* happens *on* things; *Presidents* are *of* things.

Subcategorization frames

$$\begin{aligned} P(\text{VP} \rightarrow \text{V NP NP}) &= 0.00151 \\ P(\text{VP}(\textit{said}) \rightarrow \text{V}(\textit{said})\ \text{NP NP}) &= 0.00001 \\ P(\text{VP}(\textit{gave}) \rightarrow \text{V}(\textit{gave})\ \text{NP NP}) &= 0.01980 \end{aligned}$$

Lexicalization can capture fine-grained information that the Penn Treebank non-terminals ignore. This had a major impact on parsing accuracy, as shown in Table 13.3.

Vanilla PCFG	72%
Head-annotated PCFG (Johnson, 1998)	80%
Lexicalized PCFG (Collins, 1997, 2003; Charniak, 1997)	87-89%

Table 13.3: Penn Treebank parsing accuracies

Eugene Charniak: "To do better, it is necessary to condition probabilities on the actual words of the sentence. This makes the probabilities much tighter" (at a workshop at Johns Hopkins in 2000).

Algorithms for lexicalized parsing

Naively: we could just augment the non-terminals to include the cross-product of all PTB non-terminals and all words.

The number of possible productions: $\mathcal{O}(N^3V^2)$, where the size of the vocabulary V is $V \approx 10^5$, and N is the number of non-terminals. (The term on the vocabulary size is V^2, rather than V^3, because the head of the entire parent must be identical to the head of one of the children.)

To perform lexicalized parsing, we can work bottom-up by building a table, similar to CKY. However, we need one additional piece of information: the location of the head word of each span. We should therefore store the elements $t[i, j, h, X]$, indicating a span from $\boldsymbol{w}_{i:j}$, headed by w_h ($h \in i \ldots j$), with parent node X.

To recursively construct $t[i, j, h, X]$, we need to consider two possibilities: either h is in the left child, or it is in the right child. If it is in the left child, then we have:

$$t_\ell[i, j, h, X] = \max_{s \geq h} \max_{m > s} \max_{\mathrm{X}(w_h) \rightarrow \mathrm{Y}(w_h)\mathrm{Z}(w_m)} P(\mathrm{X}(w_h) \rightarrow \mathrm{Y}(w_h)\mathrm{Z}(w_m)) \times t[i, s, h, Y] \times t[s, j, m, Z] \tag{13.16}$$

Otherwise, we have

$$t_r[i, j, h, X] = \max_{s < h} \max_{i < m \leq s} \max_{\mathrm{X}(w_h) \rightarrow \mathrm{Y}(w_m)\mathrm{Z}(w_h)} P(\mathrm{X}(w_h) \rightarrow \mathrm{Y}(w_m)\mathrm{Z}(w_h)) \times t[i, s, m, Y] \times t[s, j, h, Z] \tag{13.17}$$

We are building a table of size $\mathcal{O}(M^3N)$. To fill in each cell we perform $\mathcal{O}(M^2G)$ operations, taking maxes over two indices in the sentence, and over all rules. However, Eisner and Satta (1999) show that this time cost can be reduced back to $\mathcal{O}(M^3)$. A more serious problem is **estimation**: we must compute probabilities for $\mathcal{O}(N^3V^2)$ productions. Charniak (1997) and Collins (1997, 2003) offer practical solutions, which decompose the production probabilities using various independence assumptions.

The Charniak Parser

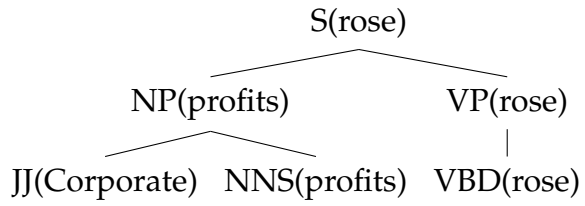
The Charniak (1997) parser gives a relatively straightforward way to lexicalize PCFGs.

Recall that **head probabilities** capture "bilexical" phenomena, like the PP attachment in the example, *President of Mexico*. The idea of the Charniak (1997)

parser is to represent the probability of each constituent by the product of two probabilities:

- The **rule probability**, $P(r \mid w_{\rho m}, t_m, t_{\rho m})$, where r is the rule, m is the index of the head of the left-hand side, t_m is the type of the left-hand side (non-terminal), and $t_{\rho m}$ is the type of the parent of m.
- The **head probability**, $P(w_m \mid w_{\rho m}, t_m, t_{\rho m})$, where w_m is a head word.

Consider this example:



- The rule probability is

$$P(\text{NP} \rightarrow \text{JJ NNS} \mid w_m = \textit{rose}, t_m = \text{NP}, t_{\rho m} = \text{S}) \tag{13.18}$$

- The head probability is

$$\text{p}(\textit{profits} \mid w_{\rho m} = \textit{rose}, t_m = \text{NP}, t_{\rho(m)} = \text{S}) \tag{13.19}$$

We would then multiply these probabilities to fill in the CKY table. This parser therefore combines two ideas that we have seen before:

- Head annotation, since both the rule and head probabilities depend on the parent type t and the grandparent type ℓ.
- Lexicalization, since the rule probability depends on the head word. Such rule probabilities can capture phenomena like verb complement frames:

Local Tree	*come*	*take*	*think*	*want*
VP → V	9.5%	2.6%	4.6%	5.7%
VP → V NP	1.1%	32.1%	0.2%	13.9%
VP → V PP	34.5%	3.1%	7.1%	0.3%
VP → V SBAR	6.6%	0.3%	73.0%	0.2%
VP → V S	2.2%	1.3%	4.8%	70.8%
VP → V NP S	0.1%	5.7%	0.0%	0.3%
VP → V PRT NP	0.3%	5.8%	0.0%	0.0%
VP → V PRT PP	6.1%	1.5%	0.2%	0.0%

- Bilexical probabilities, since the head probability depends on the head words of both the parent and child.

Estimating the Charniak parser The Charniak parser involves fewer parameters than a naive lexicalized PCFG. To estimate the relevant parameters in our example, we have

$$\begin{aligned}
&\mathrm{p}_{\text{head}}(\textit{profits} \mid t_m = \text{NP}, t_{\rho(m)} = \text{S}, w_{\rho(m)} = \textit{rose}) \\
&\quad = \frac{\text{count}(w_m = \textit{profits}, t_m = \text{NP}, t_{\rho(m)} = \text{S}, w_{\rho(m)} = \textit{rose})}{\text{count}(t_m = \text{NP}, t_{\rho(m)} = \text{S}, w_{\rho(m)} = \textit{rose})} \\
&P_{\text{rule}}(\text{NP} \rightarrow \text{JJ NNS} \mid w_{\rho(m)} = \textit{rose}, t_m = \text{NP}, t_{\rho(m)} = \text{S}) \\
&\quad = \frac{\text{count}(\text{NP} \rightarrow \text{JJ NNS}, t_m = \text{NP}, t_{\rho(m)} = \text{S}, w_{\rho(m)} = \textit{rose})}{\text{count}(t_m = \text{NP}, t_{\rho(m)} = \text{S}, w_{\rho(m)} = \textit{rose})}
\end{aligned}$$

The Penn Treebank provides is still the main dataset for syntactic analysis of English. Yet its 1M words is not nearly enough data to accurately estimate lexicalized models such as the Charniak parser, without smoothing.

For example, in 965K annotated constituent spans, there are

- 66 examples of WHADJP
- only 6 of these aren't *how much* or *how many*

In the example above (*corporate profits rose*), the unsmoothed head probability is zero, as estimated from the PTB: there are zero counts of *profits* headed by *rose* in the treebank (hard to believe, but that's what Charniak says).

In general, bilexical counts are going to be very sparse. But the "backed-off" probabilities give a reasonable approximation. We will interpolate between them.

Smoothing the Charniak Parser Head probability:

$$\begin{aligned}
\hat{\mathrm{p}}(w_m \mid t_m, w_{\rho(m)}, t_{\rho(m)}) = &\lambda_1 \mathrm{p}_{mle}(w_m \mid t_m, w_{\rho(m)}, t_{\rho(m)}) \\
&+ \lambda_2 \mathrm{p}_{mle}(w_m \mid t_m, \text{cluster}(w_{\rho(m)}), t_{\rho(m)}) \\
&+ \lambda_3 \mathrm{p}_{mle}(w_m \mid t_m, t_{\rho(m)}) \\
&+ \lambda_4 \mathrm{p}_{mle}(w_m \mid t_m)
\end{aligned}$$

, where cluster($w_{rho(m)}$) is the cluster of word $w_{\rho(m)}$, obtained by applying an automatic clustering method to **distributional** statistics (Pereira et al., 1993).

		p(*profit* $\mid NP, rose, S$)	P(*corp.* $\mid JJ$, *profit*, NP)
For example:	p($w_m \mid t_m, w_{\rho(m)}, t_{\rho(m)}$)	0	.245
	p($w_m \mid t_m, c(w_{\rho(m)}), t_{\rho(m)}$)	.0035	.015
	p($w_m \mid t_m, t_{\rho(m)}$)	.00063	.0053
	p($w_m \mid t_m$)	.00056	.0042

We have to tune $\lambda_1 \ldots \lambda_4$, and an equivalent set of parameters for the rule probabilities.

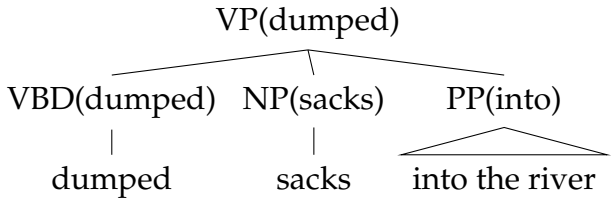
- The Charniak parser suffers from acute sparsity problems because it estimates the probability of entire rules.
- Another extreme would be to generate the children independently from each other.
  e.g., $P(\text{S} \rightarrow \text{NP VP}) \approx P_L(\text{S} \rightarrow \text{NP}) P_R(\text{S} \rightarrow \text{VP})$
- Collins (2003) and Charniak (2000) go for a compromise, conditioning on the parent and the head child.

The Collins Parser

- The Charniak parser focuses on lexical relationships between children and parents.
- The Collins (2003) parser focuses on relationships between adjacent children of the same parent. It decomposes each rule as,

$$X \rightarrow L_m L_{m-1} \ldots L_1 H R_1 \ldots R_{n-1} R_n$$

- Each L and R is a child constituent of X, and they are generated from the head H outwards.
- The outermost elements of L and R are special $\square$ symbols.



To model this rule, we would compute:

$$\text{p}(\text{VP}(\textit{dumped}, \text{VBD}) \rightarrow [\square, \text{VBD}(\textit{dumped}, \text{VBD}), \text{NP}(\textit{sacks}, \text{NNS}), \text{PP}(\textit{into}, \text{P}), \square])$$

- Here's the generative process:
  - Generate the head: $P(H \mid LHS) = P(\text{VBD}(\textit{dumped}, \text{VBD}) \mid \text{VP}(\textit{dumped}, \text{VBD}))$
  - Generate the left dependent: $P_L(\square \mid \text{VP}(\textit{dumped}, \text{VBD}), \text{VBD}(\textit{dumped}, \text{VBD}))$
  - Generate the right dependent: $P_R(\text{NP}(\textit{sacks}, \text{NNS}) \mid \text{VP}(\textit{dumped}, \text{VBD}), \text{VBD}(\textit{dumped}, \text{VBD}))$
  - Generate the right dependent: $P_R(\text{NP}(\textit{into}, PP) \mid \text{VP}(\textit{dumped}, \text{VBD}), \text{VBD}(\textit{dumped}, \text{VBD}))$
  - Generate the right dependent: $P_R(\square \mid \text{VP}(\textit{dumped}, \text{VBD}), \text{VBD}(\textit{dumped}, \text{VBD}))$
- The rule probability is the product of these generative probabilities.
- Collins parser also conditions on a "distance" of each constituent from the head.

Smoothing the Collins Parser Estimation is eased by factoring the rule probabilities, but smoothing is still needed.

$$\begin{aligned}\hat{P}(R_m(rw_m, rt_m) \mid \rho(m), hw, ht) =& \lambda_1 P_{mle}(R_m(rw_m, rt_m) \mid \rho(m), hw, ht) \\ &+ \lambda_2 P_{mle}(R_m(rw_m, rt_m) \mid \rho(m), ht) \\ &+ \lambda_3 P_{mle}(R_m(rw_m, rt_m) \mid \rho(m))\end{aligned}$$

For example,

$$\begin{aligned}&\hat{P}_R(\text{NP}(\textit{sacks}, \text{NNS}) \mid \text{VP}(\textit{dumped}, \text{VBD}), \textit{dumped}, \text{VBD}) \\ =&\lambda_1 \hat{P}(\text{NP}(\textit{sacks}, \text{NNS}) \mid \text{VP}, \textit{dumped}, \text{VBD}) \\ &+ \lambda_2 \hat{P}(\text{NP}(\textit{sacks}, \text{NNS}) \mid \text{VP}, \text{VBD}) \\ &+ \lambda_3 \hat{P}(\text{NP}(\textit{sacks}, \text{NNS}) \mid \text{VP})\end{aligned}$$

We set λ using Witten-Bell smoothing.

Bilexical dependencies The collins parser models bilexical dependencies in $P_{mle}(R_m(rw_m, rt_m) \mid \rho(m), hw, ht)$. Is it worth it?

Back-off level	Number of accesses	Percentage
0	3,257,309	1.49
1	24,294,084	11.0
2	191,527,387	87.4
Total	219,078,780	100.0

- In general, bilexical probabilites are rarely available...
- ...but they are active in 29% of the rules in **top-scoring** parses.
- Still, they don't seem to play a big role in accuracy (Bikel, 2004)

Summary of lexicalized parsing

Lexicalized parsing results in substantial accuracy gains:

Vanilla PCFG	72%
Parent-annotations (Johnson, 1998)	80%
(Charniak, 1997)	86%
(Collins, 2003)	87%

Table 13.4: Accuracies for lexicalized parsers

But the explosion in the size of the grammar required elaborate smoothing techniques and made parsing slow.

- Treebank syntactic categories are too coarse, but lexicalized categories may be too fine; modern approaches have sought middle ground.
- At the same time, natural language processing moved from generative models to more advanced machine learning techniques in the late 1990s and

early 2000s, and researchers have worked to incorporate these techniques into parsing.

13.14 Modern constituent parsing

Reranking

Charniak and Johnson (2005) combine generative and discriminative models for parsing, using the idea of **reranking**. First, a generative model is used to identify its K-best parses. Then a discriminative ranker is trained to select the best of these parses. The discriminative model doesn't need to search over all parses — just the best K identified by the generative model. This means that it can use arbitrary features — such as structural features that capture parallelism and right-branching, which could not be easily incorporated into a bottom-up parsing model. This approach yields substantial improvements in accuracy on the Penn Treebank.

Refinement grammars

Klein and Manning (2003) revisit unlexicalized parsing, expanding on the ideas in (Johnson, 1998).

They apply two types of **Markovization**:

- Vertical Markovization, making the probability of each parsing rule depend not only on the type of the parent symbol, but also on its parent type. This is identical to the parent annotation proposed by Johnson (1998). The amount of vertical Markovization can be written v, with $v = 1$ indicating a standard PCFG.

- Horizontal Markovization, where the probability of each child depends on only some of its siblings. In a standard PCFG $h = \infty$, since there is no decomposition on the right-hand side of the rule. In the Collins parser, different settings of h were explored, with $h = 1$ indicating dependence only on the head, and $h = 2$ indicating dependence on the nearest sibling as well as the head.

A comparison of various Markovization parameters is shown in Figure 13.9:

Second, Klein and Manning note that the right level of linguistic detail is somewhere between treebank categories and individual words.

Vertical Order		Horizontal Markov Order				
		$h = 0$	$h = 1$	$h \leq 2$	$h = 2$	$h = \infty$
$v = 1$	No annotation	71.27	72.5	73.46	72.96	72.62
		(854)	(3119)	(3863)	(6207)	(9657)
$v \leq 2$	Sel. Parents	74.75	77.42	77.77	77.50	76.91
		(2285)	(6564)	(7619)	(11398)	(14247)
$v = 2$	All Parents	74.68	77.42	77.81	77.50	76.81
		(2984)	(7312)	(8367)	(12132)	(14666)
$v \leq 3$	Sel. GParents	76.50	78.59	79.07	78.97	78.54
		(4943)	(12374)	(13627)	(19545)	(20123)
$v = 3$	All GParents	76.74	79.18	79.74	79.07	78.72
		(7797)	(15740)	(16994)	(22886)	(22002)

Figure 13.9: Performance for various Markovization levels (Klein and Manning, 2003).

- Some parts-of-speech and non-terminals are truly substitutable: for example, *cat*/N and *dog*/N.
- But others are not: for example, *on*/PP behaves differently from *of*/PP. This is an example of **subcategorization**.
- Similarly, Klein and Manning distinguish *and* and *but* from other coordinating conjunctions.
- They distinguish recursive and non-recursive noun phrases.

Figure 13.10 shows an example of an error that is corrected through the introduction of a new NP-TMP subcategory.

Klein and Manning (2003) analyze the improvements offered by each additional tag split, shown in Figure 13.11.

Automated state-splitting Later work from Petrov et al. (2006) automated the state-splitting process, using expectation maximization.

- Assign a random subcategory to each node.
- Learn a PCFG.
- Apply the PCFG to relabel the nodes

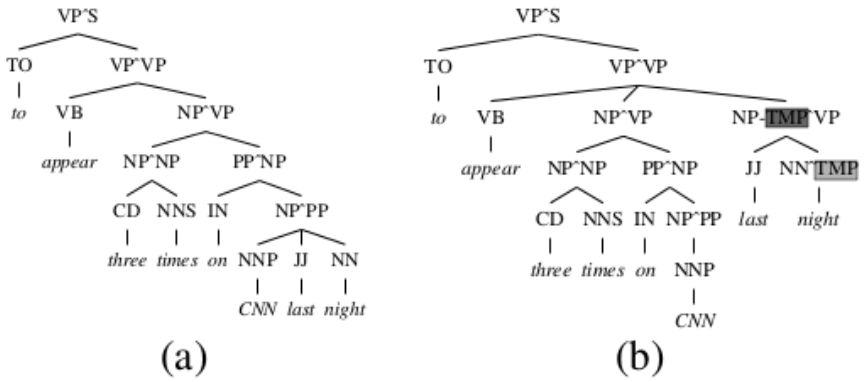


Figure 13.10: State-splitting creates a new non-terminal called NP-TMP, for temporal noun phrases. This corrects the PCFG parsing error in (a), resulting in the correct parse in (b).

	Cumulative			Indiv.
Annotation	Size	F_1	ΔF_1	ΔF_1
Baseline ($v \leq 2, h \leq 2$)	7619	77.77	–	–
UNARY-INTERNAL	8065	78.32	0.55	0.55
UNARY-DT	8066	78.48	0.71	0.17
UNARY-RB	8069	78.86	1.09	0.43
TAG-PA	8520	80.62	2.85	2.52
SPLIT-IN	8541	81.19	3.42	2.12
SPLIT-AUX	9034	81.66	3.89	0.57
SPLIT-CC	9190	81.69	3.92	0.12
SPLIT-%	9255	81.81	4.04	0.15
TMP-NP	9594	82.25	4.48	1.07
GAPPED-S	9741	82.28	4.51	0.17
POSS-NP	9820	83.06	5.29	0.28
SPLIT-VP	10499	85.72	7.95	1.36
BASE-NP	11660	86.04	8.27	0.73
DOMINATES-V	14097	86.91	9.14	1.42
RIGHT-REC-NP	15276	87.04	9.27	1.94

Figure 13.11: Parsing accuracies as new non-terminals are introduced

  - subject to constraints of original annotations:
    VP3 can be relabeled as VP7, but not as an NP

- Repeat

They applied a split-merge heuristic to determine the number of subcategories for each phrase type. See slides for more details.

Conditional Random Field parsing

We can think of a PCFG parser in our usual framework of structured prediction:

$$\hat{\tau} = \arg\max_{\tau} \boldsymbol{\theta}^\mathsf{T} \boldsymbol{f}(\tau, \boldsymbol{w}). \tag{13.20}$$

In this case, the features $\boldsymbol{f}(\tau, \boldsymbol{w})$ count all the CFG productions in τ and the terminal productions to $\boldsymbol{w}$, and the weights $\boldsymbol{\theta}$ count the log-probabilities of those productions.

This suggests that we could try to learn the weights $\boldsymbol{\theta}$ discriminatively. But if we are willing to learn the weights discriminatively, we can also add additional features; we only require a feature decomposition so that $\boldsymbol{f}(\tau, \boldsymbol{w})$ decomposes across the productions in τ, so that we can still perform CKY parsing to find the best-scoring parse. For example, under such a decomposition, we could incorporate lexical features, so that we learn weights for the non-terminal production as well as for lexicalized forms,

$f1$ $\text{NP}(*) \rightarrow \text{NP(*)}\ \text{PP(*)}$

$f2$ $\text{NP}(\textit{cats}) \rightarrow \text{NP}(\textit{cats})\ \text{PP}(*)$

$f3$ $\text{NP}(*) \rightarrow \text{NP}(*)\ \text{PP}(\textit{claws})$

$f4$ $\text{NP}(\textit{cats}) \rightarrow \text{NP}(\textit{cats})\ \text{PP}(\textit{claws})$

Through regularization, we can find weights that strike a good balance between frequently-observed features ($f1$) and more discriminative features ($f4$).

This approach was implemented by Finkel et al. (2008) in the context of PCFG parsing with Conditional Random Fields. They used stochastic gradient descent for training, with the inside-outside algorithm (analogous to forward-backward, but for trees) to compute expected feature counts. However, the time complexity of $\mathcal{O}(M^3)$ posed serious challenges — recall that CRF sequence labeling can be trained in linear time. Finkel et al. (2008) address these issues by "prefiltering"

the CKY parsing chart, identifying the productions which cannot be part of any complete parse.

Carreras et al. (2008) use the averaged perceptron to perform conditional parsing, employing an alternative feature decomposition based on tree-adjoining grammar (TAG). This yields substantially better results ($F = 90.5$), but is still less accurate than the reranked generative parser of McClosky et al. (2006).

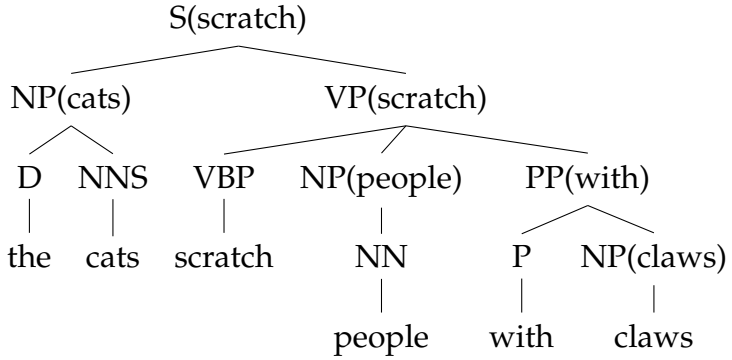
Vanilla PCFG	72%
Parent-annotations (Johnson, 1998)	80%
Lexicalized (Charniak, 1997)	86%
Lexicalized (Collins, 2003)	87%
Lexicalized, reranking, self-training (McClosky et al., 2006)	92.1%
State splitting (Petrov and Klein, 2007)	90.1%
CRF Parsing (Finkel et al., 2008)	89%
TAG Perceptron Parsing (Carreras et al., 2008)	90.5%
Compositional Vector Grammars (Socher et al., 2013)	90.4%

Table 13.5: Parsing scoreboard, circa 2013(Socher et al., 2013)

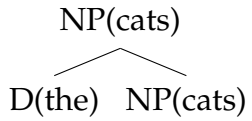
Chapter 14

Dependency Parsing

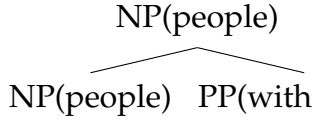
Lexicalized parsing augments the non-terminals with **head words**.



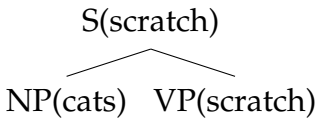
- A set of deterministic **head percolation** rules determine how heads move up the tree through each production.
- Some rules are pretty obvious:
  - the head of a determiner-noun constituent is the noun:

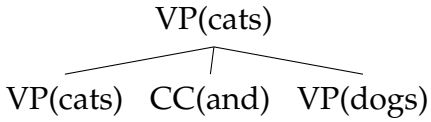
  - prepositional adjuncts are not heads:

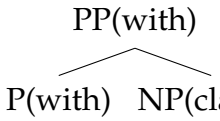
  - verbal predicates are the heads of sentences:



- Others are less clear-cut:
  - the head of a conjunction is the left element:



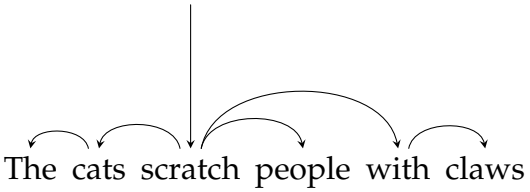
  - the head of a prepositional phrase is the preposition:



    (Alternatively, we can "collapse" out the preposition itself. This is the approach taken by the Stanford dependency parser.)

- We could argue about this stuff, but once we agree on a standard it can be applied deterministically to any parse tree.

A head-annotated parse tree defines a graph over the words in the sentence:

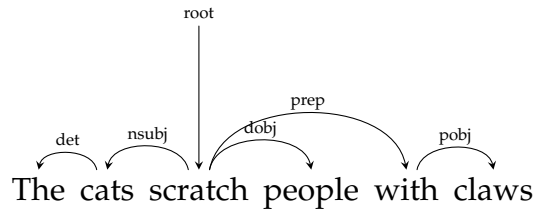


What are the properties of this graph?

- directed
- weakly connected
- every node has one incoming edge
- (therefore) no cycles
- (therefore) a tree

What is the meaning of the edges?

- A dependency edge means there is an asymmetric syntactic relationship between the head and the modifier.
- (sometimes called nucleus/satellite, or governor/dependent, or parent/child)
- Criteria for figuring out who is the head:
  - The modifier may be optional while the head is mandatory: (*red HAT*, *EAT quickly*)
  - The head sets the syntactic category of the construction: (prepositions are the heads of prepositional phrases)
  - The head determines the morphological form of the modifier: (*los LIBROS*, *una CASA*, *these HOUSES*)
- As always, these guidelines sometimes conflict.
- Edges may be **labeled** to indicate their function:

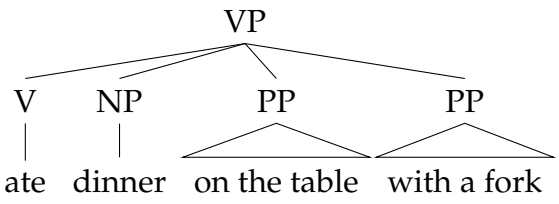


Dependency trees tell us who did what to whom.

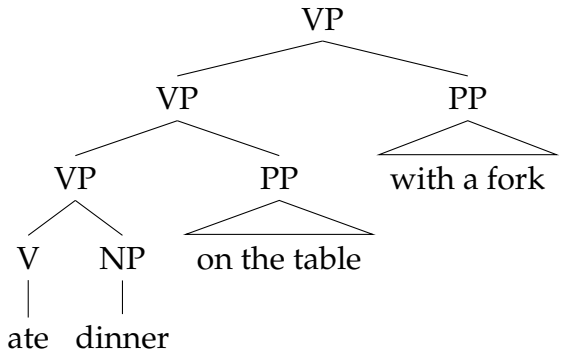
Expressiveness

(Unlabeled) dependency trees are less expressive than CFG derivations. That means they hide some of the ambiguity in CFGs that we may not care about. Remember the different representations for PP modification?

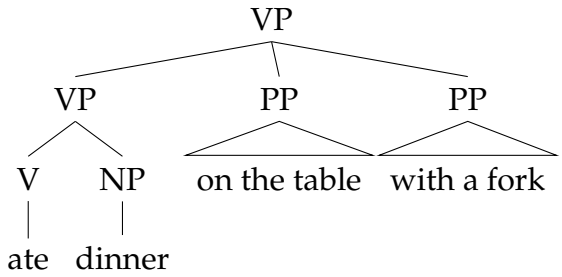
- Flat



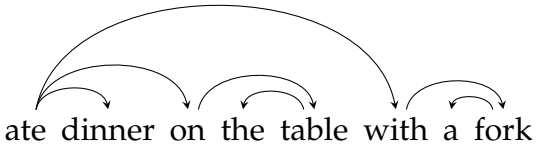
- (Chomsky) adjunction



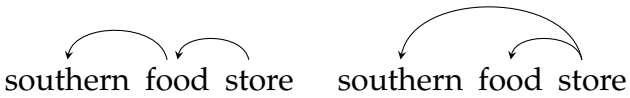
- Two-level (PTB)



These all look the same in a dependency parse:



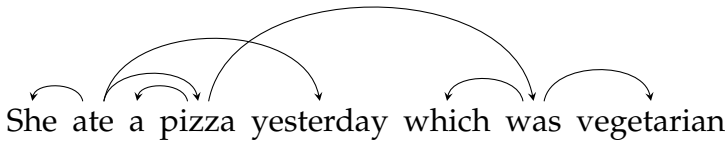
So if you didn't think there was any meaningful difference between these representations, you should be happy. But many kinds of CFG ambiguity remain in the dependency parse:



Projectivity

In projective dependency parsing, there can be no crossing edges in the dependency graph.

- More formally, for every word w_h, there must be a span $i : j$ such a word w_m is a descendant of w_h if and only if it lies within the span $w_i, w_{i+1}, \ldots, w_j$.
- Crossing edges are rare in English:



  However, they are more common in other languages, like Czech.
- We can build projective dependency banks from constituent treebanks, like PTB, by using a set of head-finding rules.
  - In this case, crossing edges are prohibited by construction.
  - What is the CFG analysis for the pizza example?
- In languages where non-projectivity is common, we must annotate dependency trees directly. An example is the Prague dependency Treebank, which contains 1.5M words of Czech.

14.1 Algorithms

Let's assume that the score for a dependency parse can be **factored** across the edges:

$$\Psi(G) = \bigotimes_i \psi(g(i) \rightarrow i, r), \tag{14.1}$$

where $g(i)$ is the parent of node i and r is the relation type. If we can assume this factorization, then we have efficient dynamic programs for dependency parsing. The remainder of this section will concern itself with unlabeled dependency parsing, dropping the label r.

Maximum spanning tree

Non-projective dependency parsing reduces to the maximum spanning tree problem (in directed graphs).

- We can build a connected graph, with edge weights equal to $\psi(i \rightarrow j)$ for all words i, j in the sentence. In the case of labeled parsing, we can build multiple edges between each pair of words, one for each relation r, with weight $\psi(i \rightarrow j, r)$.
- To get a dependency parse, we need a tree that touches all the nodes.
- To get the **best** dependency parse, we need a tree that achieves the maximum score $\bigotimes_i \psi(g(i) \rightarrow i)$, where $g(i)$ is the head of node i.
- The Chu-Liu-Edmonds algorithm computes this in $\mathcal{O}(N^3)$. [See slides.]
- The Tarjan algorithm is $\mathcal{O}(N^2)$.

Dynamic programs

Projective dependency parsing can be performed with dynamic programming. Here is a naive recursive algorithm, for unlabeled dependency parsing.

- $c[i, j, h]$ is the score of the best tree spanning $i \rightarrow j$ with head h

$$c[i, j, h] = \left(\bigoplus_{k, h'} c[i, k, h] \otimes c[k, j, h'] \otimes \psi(h \rightarrow h') \right) \oplus \left(\bigoplus_{k, h'} c[i, k, h'] \otimes c[k, j, h] \otimes \psi(h \rightarrow h') \right)$$

- The first line represents left-branching trees; the second line represents right-branching trees. Projectivity guarantees that we can find a span $[i, j]$ such that all the words $\boldsymbol{w}_{i:j}$ are children of h.
- What is the complexity? The size of the table is $\mathcal{O}(N^3)$. To fill in each node, we must consider $\mathcal{O}(N)$ possible split points and $\mathcal{O}(N)$ possible heads h' for the satellite subtree. So the time complexity is $\mathcal{O}(N^5)$.
- The Eisner algorithm reduces this to $\mathcal{O}(N^3)$, by adapting the CKY algorithm. We keep four tables instead of 1!
  - scores of **incomplete** subtrees from i to j, headed to the left
  - scores of **incomplete** subtrees from i to j, headed to the right

  - scores of **complete** subtrees from i to j, headed to the left
  - scores of **complete** subtrees from i to j, headed to the right

- Why?
  - Incomplete subtrees can subsume complete subtrees heading in the same direction, resulting in a complete subtree.
  - Complete subtrees can combine if they are heading in opposite directions, resulting in an incomplete subtree.
  - Our goal is to produce a complete tree from 0 to M.

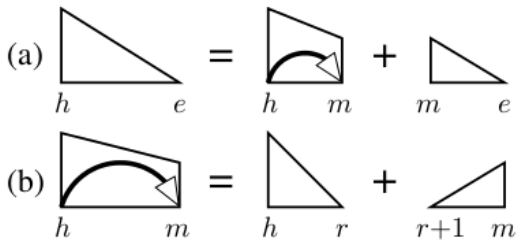


Figure 14.1: Diagram of Eisner algorithm for first-order dependency parsing (Koo and Collins, 2010)

.

The Eisner algorithm in action

Let's consider the unlabeled projective dependency parse for the phrase "plastic cup holders."

Assume we have the following log potentials $\psi_{i,j} = \boldsymbol{\theta}^\mathsf{T} \boldsymbol{f}(i, j)$.

		ROOT	plastic	cup	holders
	ROOT		1	1	1
Edge weights:	plastic	$-\infty$		-1	-1
	cup	$-\infty$	2		-1
	holders	$-\infty$	0	4	

Let's assume we have a 4-dimensional table C, such that $C[i, j, d, c]$ is the score of:

- the best subtree from i to j, with $i, j \in [1, M]$
- in direction $d \in \{\leftarrow, \rightarrow\}$
- with completeness $c \in \{0, 1\}$

If $d = \leftarrow$ then the head of the tree is j, the right-most element; otherwise the head is the left-most element, i. If completeness $c = 1$, then the subtree is not taking any more dependents; otherwise it needs to be completed.

The dynamic program works as follows

- Build an incomplete subtree by merging two adjacent subtrees a and b, where a is right-facing and b is left-facing. The new subtree is left-facing if we add a dependency from the right edge of b to the left edge of a; otherwise it's right-facing.
- Build a complete left-facing subtree by merging a complete left-facing subtree a with an adjacent incomplete left-facing subtree b.
- Build a complete right-facing subtree by merging an incomplete right-facing subtree a with a complete left-facing subtree b.

At each merge, we search for the best split point, which is scored by adding the scores of the subtrees. Specifically:

$$
\begin{aligned}
C[s, t, \leftarrow, 0] &= \psi_{t,s} + \max_{s \leq r < t} (c[s, r, \rightarrow, 1] + c[r + 1, t, \leftarrow, 1]) \\
C[s, t, \rightarrow, 0] &= \psi_{s,t} + \max_{s \leq r < t} (c[s, r, \rightarrow, 1] + c[r + 1, t, \leftarrow, 1]) \\
C[s, t, \leftarrow, 1] &= \max_{s \leq r < t} (c[s, r, \leftarrow, 1] + c[r, t, \leftarrow, 0]) \\
C[s, t, \rightarrow, 1] &= \max_{s \leq r < t} (c[s, r, \rightarrow, 0] + c[r, t, \rightarrow, 1])
\end{aligned}
$$

We also need to keep back pointers. The score of the final parse is $C[1, n, \rightarrow, 1]$.

$c[1, 2, \leftarrow, 0] = c[1, 1, \rightarrow, 1] + c[2, 2, \leftarrow, 1] + \psi(2, 1) = \psi(2, 1) = -\infty$
$c[1, 2, \rightarrow, 0] = c[1, 1, \rightarrow, 1] + c[2, 2, \leftarrow, 1] + \psi(1, 2) = \psi(1, 2) = 1$
$c[1, 2, \leftarrow, 1] = c[1, 1, \leftarrow, 1] + c[1, 2, \leftarrow, 0] = c[1, 2, \leftarrow, 0] = \psi(2, 1) = -\infty$
$c[1, 2, \rightarrow, 1] = c[1, 2, \rightarrow, 0] + c[2, 2, \rightarrow, 1] = c[1, 2, \rightarrow, 0] = \psi(1, 2) = 1$

$c[2, 3, \leftarrow, 0] = \psi(3, 2) = 2$
$c[2, 3, \rightarrow, 0] = \psi(2, 3) = -1$
$c[2, 3, \leftarrow, 1] = c[2, 3, \leftarrow, 0] = \psi(3, 2) = 2$
$c[2, 3, \rightarrow, 1] = \psi(2, 3) = -1$

$c[3, 4, \leftarrow, 0] = \psi(4, 3) = 4$
$c[3, 4, \rightarrow, 0] = \psi(3, 4) = -1$
$c[3, 4, \leftarrow, 1] = c[3, 4, \leftarrow, 0] = \psi(4, 3) = 4$
$c[3, 4, \rightarrow, 1] = \psi(3, 4) = -1$

$c[1, 3, \leftarrow, 0] = \psi(3, 1) + \max(c[1, 1, \rightarrow, 1] + c[2, 3, \leftarrow, 1], c[1, 2, \rightarrow, 1] + c[3, 3, \leftarrow, 1])$
$= \psi(3, 1) + \max(\psi(3, 2), \psi(1, 2)) = \psi(3, 1) + \psi(3, 2) = -\infty$
$c[1, 3, \rightarrow, 0] = \psi(1, 3) + \max(c[2, 3, \leftarrow, 1], c[1, 2, \rightarrow, 1]) = \psi(1, 3) + \psi(3, 2) = 3$
$c[1, 3, \leftarrow, 1] = \max(c[1, 1, \leftarrow, 1] + c[1, 3, \leftarrow, 0], c[1, 2, \leftarrow, 1] + c[2, 3, \leftarrow, 0]) = \max(0 - \infty, -\infty + 2) = -\infty$
$c[1, 3, \rightarrow, 1] = \max(c[1, 2, \rightarrow, 0] + c[2, 3, \rightarrow, 1], c[1, 3, \rightarrow, 0] + c[3, 3, \rightarrow, 1])$
$= \max(\psi(1, 2) + \psi(2, 3), \boldsymbol{\psi(1, 3) + \psi(3, 2)}) = \max(0, 3) = 3$

$c[2, 4, \leftarrow, 0] = \psi(4, 2) + \max(c[3, 4, \leftarrow, 1], c[2, 3, \rightarrow, 1]) = \psi(4, 2) + \max(\boldsymbol{\psi(4, 3)}, \psi(2, 3)) = 4$
$c[2, 4, \rightarrow, 0] = \psi(2, 4) + \max(\boldsymbol{\psi(4, 3)}, \psi(2, 3)) = 3$
$c[2, 4, \leftarrow, 1] = \max(c[2, 4, \leftarrow, 0], c[2, 3, \leftarrow, 1] + c[3, 4, \leftarrow, 0])$
$= \max(\psi(4, 2) + \psi(4, 3), \boldsymbol{\psi(3, 2) + \psi(4, 3)}) = 6$
$c[2, 4, \rightarrow, 1] = \max(c[2, 3, \rightarrow, 0] + c[3, 4, \rightarrow, 1], c[2, 4, \rightarrow, 0])$
$= \max(\psi(2, 3) + \psi(3, 4), \boldsymbol{\psi(2, 4) + \psi(4, 3)}) = 3$

$c[1, 4, \leftarrow, 0] = \psi(4, 1) + \ldots = -\infty$
$c[1, 4, \rightarrow, 0] = \psi(1, 4) + \max(c[2, 4, \leftarrow, 1], c[1, 2, \rightarrow, 1] + c[3, 4, \leftarrow, 1], c[1, 3, \rightarrow, 1])$
$= \psi(1, 4) + \max(\boldsymbol{\psi(3, 2) + \psi(4, 3)}, \psi(1, 2) + \psi(4, 3), \psi(1, 3) + \psi(3, 2)) = 1 + 6 = 7$
$c[1, 4, \leftarrow, 1] = \max(0 + -\infty, -\infty + \ldots, -\infty + \ldots) = -infty$
$c[1, 4, \rightarrow, 1] = \max(c[1, 2, \rightarrow, 0] + c[2, 4, \rightarrow, 1], c[1, 3, \rightarrow, 0] + c[3, 4, \rightarrow, 1], c[1, 4, \rightarrow, 0])$
$= \max(\psi(1, 2) + \psi(2, 4) + \psi(4, 3), \psi(1, 3) + \psi(3, 2) + \psi(3, 4), \boldsymbol{\psi(1, 4) + \psi(4, 3) + \psi(3, 2)})$
$= \max(1 + 3, 3 - 1, 7) = 7$

This corresponds to the dependency parse $\textit{plastic} < (\textit{cup} < \textit{holders})$

Transition-based parsing

An alternative to exact global inference is transition-based parsing: making a series of local decisions. We can apply a shift-reduce algorithm, just as we considered for CFG parsing. The reduce actions are different: rather than combining elements into non-terminals, they create arcs between words, leaving the head of edge.

- Read the sentences left-to-right,
  - **shift**: push a word onto the stack
  - **right-reduce**: make a right-facing edge between the top two elements on the stack
  - **left-reduce**: make a left-facing edge between the top two elements on the stack
  - Alternatively, "arc-eager" dependency parsing distinguishes **reduce** from **arc-right** and **arc-left**, which create arcs between the top of the stack and the first element in the queue. Arc-eager parsing is arguably more cognitively plausible, because it constructs larger connected components incrementally, rather than having a deep stack with lots of disconnected elements (Abney and Johnson, 1991; Nivre, 2004).
- **Beam search** is an improvement on shift-reduce.
  - We keep a "beam" of possible hypotheses.
  - At each stage, let the k best unique hypotheses stay on the beam.
  - In this way, we are robust to greedy bad decisions.
- **Learning**
  - Identify the series of decisions required to produce the correct dependency parse.
  - Each decision in the derivation of the correct parse is a positive instance. Each other possible decision is a negative instance.

– Huang et al. (2012) offer alternative perceptron learning rules that yield improvements when learning in the beam search setting.

A key advantage of transition-based parsing is that there is no restriction to arc-factored features; we can include any feature of the current partial parse, history of decisions, etc. It is also fast: linear time in the length of the sentence.

14.2 Higher-order dependency parsing

Arc-factored dependency parsers can only score dependency graphs as a product across their edges. Higher-order parsers (Koo and Collins, 2010) are able to consider pairs or triples of edges (Figure 14.2)

- Second-order features consider **siblings** and **grandparents**.
- Third-order features consider **grand-siblings** (siblings and grandparents together) and **tri-siblings**.

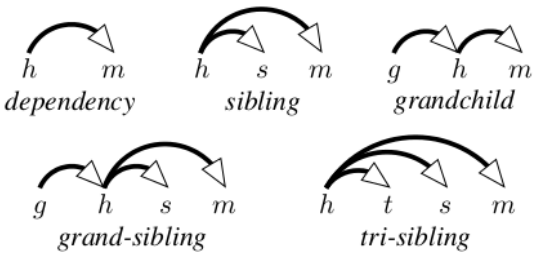


Figure 14.2: Feature templates for higher-order dependency parsing (Koo and Collins, 2010)

Why might we need higher-order dependency features? Again consider the example *cats scratch people with claws*, where the preposition *with* could attach to either *scratch* or *people*. In a lexicalized first-order arc-factored dependency parser, we would have the following feature sets for the two possible parses:

- $\langle ROOT \rightarrow scratch \rangle, \langle scratch \rightarrow cats \rangle, \langle scratch \rightarrow people \rangle, \langle scratch \rightarrow with \rangle, \langle with \rightarrow claws \rangle$
- $\langle ROOT \rightarrow scratch \rangle, \langle scratch \rightarrow cats \rangle, \langle scratch \rightarrow people \rangle, \langle people \rightarrow with \rangle, \langle with \rightarrow claws \rangle$

The only difference between the feature vectors are the features $\langle \textit{scratch} \rightarrow \textit{with} \rangle$ and $\langle \textit{people} \rightarrow \textit{with} \rangle$, but both are reasonable features, both syntactically and semantically. A first-order arc-factored dependency parsing model would therefore struggle to find the right solution to this sentence. However, if we add grandparent features, then our feature sets include:

- $\langle \textit{scratch} \rightarrow \textit{with} \rightarrow \textit{claws} \rangle$
- $\langle \textit{people} \rightarrow \textit{with} \rightarrow \textit{claws} \rangle$,

The first feature is preferable, so a second-order dependency parser would have a better chance of correctly parsing this sentence.

Projective second-order parsing can still be performed in $\mathcal{O}(M^3)$ time (and $\mathcal{O}(M^2)$ space), using a modified version of the Eisner algorithm that includes "sibling spans." Projective third-order parsing can be performed in $\mathcal{O}(M^4)$ time and $\mathcal{O}(M^3)$ space. Non-projective second-order dependency parsing is NP-Hard (Neuhaus and Bröker, 1997), by reduction from the vertex cover problem. One approach is to do projective parsing first, and then post-process the projective dependency parse to add non-projective edges (Nivre and Nilsson, 2005).

14.3 Learning dependency parsers

Returning to arc-factored dependency parsing, we can easily design both generative and discriminative models:

- Generative: $\psi(m \rightarrow n) = \log P(h = m|n)$, estimated from a treebank.
- Log-linear: $\psi(m \rightarrow n) = \boldsymbol{\theta}^\mathsf{T} \boldsymbol{f}(w_m, w_n)$. Features:
  - POS tag of w_m and w_n
  - Word identity of w_m and w_n
  - Word shape (e.g., suffix, prefix)
  - Distance $(m - n)$
  - ...

Structured perceptron

In the log-linear model above, it is easy to learn the weights using structured perceptron

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}' \in \mathcal{T}(\boldsymbol{w})} \boldsymbol{\theta}^\intercal \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{y}')$$
$$\boldsymbol{\theta}^\intercal = \boldsymbol{\theta}^\intercal + \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{y}) - \boldsymbol{f}(\boldsymbol{w}, \hat{\boldsymbol{y}})$$

This is just like sequence labeling, but now $\arg\max_{\boldsymbol{y}' \in \mathcal{T}(\boldsymbol{x})}$ searches over dependency trees, using either MST or the Eisner algorithm.

Conditional Random Fields (CRFs)

CRFs are just globally-normalized conditional models, and they can also applied to any graphical model in which we can efficiently compute marginals. The prediction step is identical to the structured perceptron (using MST or the Eisner algorithm); for learning, we have a gradient that is analogous to the case in sequence labeling:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \sum_m \boldsymbol{f}(x_{h(m)} \rightarrow w_m) - \sum_n P(n \rightarrow m | \boldsymbol{w}) \boldsymbol{f}(w_n \rightarrow w_m) \tag{14.2}$$

We require **marginal** probabilities $P(n \rightarrow m | \boldsymbol{w})$. These can be obtained efficiently using a variant of inside-outside (for projective trees), and the matrix-tree theorem for non-projective trees (Koo et al., 2007).

14.4 Applications

Dependency parsing is used in many real-world applications: any time you want to know about pairs of words which might not be adjacent, you can use dependency links instead of typical regular expression search patterns. For example, we may want to match strings like *delicious pastries*, *delicious French pastries*, and *the pastries are delicious*[1]

- It is now possible to search Google n-grams by dependency edges.

[1] Note that the copula *is* is collapsed in many dependency parsing systems, such as the Stanford dependency parser De Marneffe and Manning (2008).

- Muralidharan and Hearst (2013) show how dependency parsing can be used in humanities research.
- Cui et al. (2005) show how dependency parsing can improve question answering. For example:
  - Question: *What % of the nation's cheese does Wisconsin produce?*
  - Now suppose your corpus contains this sentence: *In Wisconsin, where farmers produce 28% of the nation's cheese, ...*
  - The location of *Wisconsin* in the surface form of this string might make it a poor match for the query. However, in the dependency graph, there is an edge from *produce* to *Wisconsin* in both the question and the potential answer, raising the likelihood that this span of text is relevant to the question.
- In sentiment analysis, the polarity of a sentence can be reversed by negation, e.g. *There is no reason at all to believe the polluters will suddenly become reasonable.* By tracking the sentiment polarity through the dependency parse, we can better identify the overall polarity of the sentence (Wilson et al., 2005; Nakagawa et al., 2010).

Chapter 15

Distributional semantics

A recurring theme in this course is that the mapping from words to meaning is complex.

- **Word sense disambiguation**: multiple meanings for the same form (e.g., *bank*)
- **Morphological analysis**: shared semantic basis among multiple forms (e.g., *speak*, *spoke*, *speaking*)
- **Synonymy**: in English we have lots of synonyms and near neighbors, as English combines influence from lots of other languages (French, Latin, German, etc)
- Both **compositional** and **frame** semantics assume hand-crafted resources that map from words to predicates.

How do we do semantic analysis of words that we've never seen before?

15.1 The distributional hypothesis

Here's a word you may not know: *tezgüino*. If we encounter this word, what can we do? It seems like a big problem for any NLP system, from POS tagging to semantic analysis.

Suppose we see that *tezgüino*is used in the following contexts:

1. A bottle of _____ is on the table.

2. Everybody likes _____.
3. Don't have _____ before you drive.
4. We make _____ out of corn.

What other words fit into these contexts? How about: *loud*, *motor oil*, *tortillas*, *choices*, *wine*?

We can create a vector for each word, based on whether it can be used in each context.

	C1	C2	C3	C4	...
tezgüino	1	1	1	1	
loud	0	0	0	0	
motor oil	1	0	0	1	
tortillas	0	1	0	1	
choices	0	1	0	0	
wine	1	1	1	1	

- Based on these vectors, we see:
  - *wine* is very similar to *tezgüino*
  - *motor oil* and *tortillas* are fairly similar to *tezgüino*
  - *loud* is quite different.
- The vectors describe the **distributional** properties of each word.
- Does vector similarity imply semantic similarity? This is the **distributional hypothesis.** "You shall know a word by the company it keeps." (Firth 1957)
- It is also known as a **vector-space model**, since each word's meaning is captured by a vector.

Vector-space models and distributional semantics are relevant to a wide range of NLP applications.

- **Query expansion**: search for *bike*, match *bicycle*
- **Semi-supervised learning**: use large unlabeled datasets to acquire features which are useful in supervised learning

- **Lexicon and thesaurus induction**: automatically expand hand-crafted lexical resources, or induce them from raw text

Here are some of the practical questions that we encounter when working with vector space representations of distributional semantics:

- What kinds of context should we consider? (see slides)
- How do measure similarity?
- How do we properly weigh frequent versus infrequent events?

15.2 Local context

The Brown et al (1992) clustering algorithm is over 20 years old and is still widely used in NLP!

- Context is just the immediately adjacent words.
- A generative probability model:
  - Assume each word w_i has a class c_i
  - Assume a generative model $\log \mathrm{p}(w) = \sum_i \log \mathrm{p}(w_i|c_i) + \log \mathrm{p}(c_i|c_{i-1})$ (What does this remind you of?)
- Hierarchical clustering algorithm:
  - Start with every word in its own cluster
  - Until tired,
    - Choose two clusters c_i and c_j such that merging them will give the maximum improvement in $\log \mathrm{p}(w)$
    - Equivalently, merge the clusters with the greatest mutual information.
  - The merge path of a word describes its semantics.

Model specifics

- $\mathcal{V}$ is the set of all words
- N number of observed word tokens
- $n(w)$ is the number of times we see word $w \in \mathcal{V}$
- $n(w, v)$ is the number of times w precedes v
- Let $C \rightarrow \{1, 2, \ldots, k\}$ define a partition of words into k classes

$$\mathrm{p}(w_1, w_2, \ldots, w_T; C) = \prod_i \mathrm{p}(w_i|C(w_i))\mathrm{p}(C(w_i)|C(w_{i-1}))$$
$$\log \mathrm{p}(w_1, w_2, \ldots, w_T; C) = \sum_i \log \mathrm{p}(w_i|C(w_i))\mathrm{p}(C(w_i)|C(w_{i-1}))$$

This is kind of like an HMM, but each word can only be produced by a single cluster.

Let's define the "quality" of a clustering as the average log-likelihood:

$$\begin{aligned}
J(C) =& \frac{1}{N} \sum_i^N \log \left(\mathrm{p}(w_i|C(w_i))\mathrm{p}(C(w_i)|C(w_{i-1}))\right) \\
=& \sum_{w,w'} \frac{n(w,w')}{N} \log \left(\mathrm{p}(w'|C(w'))\mathrm{p}(C(w')|C(w'))\right) && \text{sum over word types instead} \\
=& \sum_{w,w'} \frac{n(w,w')}{N} \log \left(\frac{n(w')}{n(C(w'))} \frac{n(C(w), C(w'))}{n(C(w))}\right) && \text{definition of probabilities} \\
=& \sum_{w,w'} \frac{n(w,w')}{N} \log \left(\frac{n(w')}{1} \frac{n(C(w), C(w'))}{n(C(w))n(C(w'))} \frac{N}{N}\right) && \text{re-arrange, multiply by one} \\
=& \sum_{w,w'} \frac{n(w,w')}{N} \log \left(\frac{n(w')}{N} \times \frac{n(C(w), C(w')) \times N}{n(C(w))n(C(w'))}\right) && \text{re-arrange terms} \\
=& \sum_{w,w'} \frac{n(w,w')}{N} \log \frac{n(w')}{N} + \frac{n(w,w')}{N} \log \left(\frac{n(C(w), C(w')) \times N}{n(C(w))n(C(w'))}\right) && \text{distributive law} \\
=& \sum_{w'} \frac{n(w')}{N} \log \frac{n(w')}{N} + \sum_{c,c'} \frac{n(c,c')}{N} \log \left(\frac{n(c,c') \times N}{n(c)n(c')}\right) && \text{sum across classes} \\
=& \sum_{w'} \mathrm{p}(w') \log \mathrm{p}(w') + \sum_{c,c'} \mathrm{p}(c,c') \log \frac{\mathrm{p}(c,c')}{\mathrm{p}(c)\mathrm{p}(c')} && \text{multiply by } \frac{N^{-2}}{N^{-2}} \text{ inside log} \\
=& -H(W) + I(C)
\end{aligned}$$

In the last step, we use the following definitions from information theory:

Entropy The entropy of a discrete random variable is the expected negative log-likelihood,

$$H(X) = -E[\log P(X)] = -\sum_x P(X = x) \log P(X = x). \tag{15.1}$$

For example, for a fair coin we have $H(X) = \frac{1}{2}\log\frac{1}{2} + \frac{1}{2}\log\frac{1}{2} = -\log 2$; for a (virtually) certain outcome, we have $H(x) = 1 \times \log 1 + 0 \times \log 0 = 0$. We have already seen entropy in a few other contexts.

Mutual information The information shared by two random variables is the mutual information,

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} \mathrm{p}_{X,Y}(x,y) \log \left(\frac{\mathrm{p}_{X,Y}(x,y)}{\mathrm{p}_X(x)\mathrm{p}_Y(y)} \right). \tag{15.2}$$

For example, if X and Y are independent, then $\mathrm{p}_{X,Y}(x,y) = \mathrm{p}_X(x)\mathrm{p}_Y(y)$, so the mutual information is $\log 1 = 0$. In

By $I(C)$, we are using a shorthand for the mutual information of random variables C_i and C_{i-1} — the cluster memberships of word i and $i-1$, respectively. So we have

$$I(C) = \sum_{C_i = c, C_{i-1} = c'} \frac{P(C_i = c, C_{i-1} = c')}{P(C_i = c)P(C_{i-1} = c')} \tag{15.3}$$

The entropy $H(W)$ does not depend on the clustering, so this term is constant; choosing a clustering with maximum mutual information $I(C)$ is equivalent to maximizing the log-likelihood. Now let's see how to do that efficiently.

$V \log V$ approximate algorithm

- Take m most frequent words, put each in its own cluster $c_1, c_2, \ldots, c_m$.
- For $i = (m+1) : |\mathcal{V}|$
  - Create a new cluster for the c_{m+1} for word i (ordered by frequency).
  - Choose two clusters c and c' to merge, minimizing the decrease in $I(C)$. This requires $\mathcal{O}(m^2)$ operations.

- Carry out $(m-1)$ final merges, to build full hierarchy

Cost: $\mathcal{O}(|\mathcal{V}|m^2 + n)$, plus time to sort words, $\mathcal{O}(|\mathcal{V}| \log |\mathcal{V}|)$.

15.3 Syntactic context

Local context is contingent on syntactic decisions that may have little to do with semantics:

- I gave Tim the ball.
- I gave the ball to Tim.

Using the syntactic structure of the sentence might give us a more meaningful context, yielding better clusters.

- Pereira et al (1993) cluster nouns based on the verbs for which they are the direct object.
  - The context vector for each noun is **the count of occurences as a direct object of each verb**.
  - As with Brown clustering, a class-based probability model:

$$\begin{aligned}\hat{\text{p}}(n, v) &= \sum_{c \in \mathcal{C}} \text{p}(c, n)\text{p}(v \mid c) \\ &= \sum_{c \in \mathcal{C}} \text{p}(c)\text{p}(n \mid c)\text{p}(v|c)\end{aligned}$$

    where n is the noun, v is the verb, and c is the class
  - Objective: find the maximum likelihood cluster centroids.
- Dekang Lin (1997) extends this to all words, using incoming dependency edges (see slide)
  - For any pair of words i and j and relation r, we can compute:

$$\text{p}(i, j \mid r) = \frac{c(i, j, r)}{\sum_{i', j'} c(i', j', r)}, \qquad \text{p}(i \mid r) = \sum_j \text{p}(i, j \mid r)$$

  - Let $T(i)$ be the set of pairs $\langle j, r \rangle$ such that $\text{p}(i, j \mid r) > \text{p}(i \mid r)\text{p}(j \mid r)$

  - $*$ $T(i)$ contains words j that are especially likely to be joined with word i in relation r.
  - $*$ Note the connection to pointwise mutual information.
- – Similarity between u and v is defined through $T(u)$ and $T(v)$.
  - $*$ Lin considers several similarity measures for $T(u)$ and $T(v)$.
  - $*$ Many of these are used widely, and are worth knowing:
    - · Cosine similarity: $\frac{|T(u) \cap T(v)|}{\sqrt{|T(u)||T(v)|}}$
    - · Dice similarity: $\frac{2 \times |T(u) \cap T(v)|}{|T(u)| + |T(v)|}$
    - · Jaccard similarity: $\frac{|T(u) \cap T(v)|}{|T(u)| + |T(v)| - |T(u) \cap T(v)|}$
  - $*$ Lin's metric is more complex:

$$\frac{\sum_{\langle r,w \rangle \in T(u) \cup T(v)} I(u,r,w) + I(v,r,w)}{\sum_{\langle r,w \rangle \in T(u)} I(u,r,w) + \sum_{\langle r,w \rangle \in T(v)} I(v,r,w)}$$

    where $I(u,r,w)$ is the mutual information between u and w, conditioned on r.
- – See slides for results.

15.4 Latent semantic analysis

(See slides)

Thus far, we have considered context vectors that are large and sparse. We can arrange these vectors into a matrix $\mathbf{X} \in \mathbb{R}^{V \times N}$, where rows correspond to words and columns correspond to contexts. However, for rare words i and j, we might have $\boldsymbol{x}_i^\intercal \boldsymbol{x}_j = 0$. So we'd like to have a more robust representation.

We can obtain this by factoring $\mathbf{X} \approx \mathbf{U}_K \mathbf{S}_K \mathbf{V}_K^\intercal$, where

$$\mathbf{U}_K \in \mathbb{R}^{V \times K}, \qquad \mathbf{U}_K \mathbf{U}_K^\intercal = \mathbb{I} \tag{15.4}$$

$$\mathbf{S}_K \in \mathbb{R}^{K \times K}, \qquad \mathbf{S}_K \text{ is diagonal} \tag{15.5}$$

$$\mathbf{V}_K \in \mathbb{R}^{D \times K}, \qquad \mathbf{V}_K \mathbf{V}_K^\intercal = \mathbb{I} \tag{15.6}$$

Here K is a parameter that determines the fidelity of the factorization; if $K = \min(V, N)$, then $\mathbf{X} = \mathbf{U}_K \mathbf{S}_K \mathbf{V}_K^\intercal$. Otherwise, we have

$$\mathbf{U}_K, \mathbf{S}_K, \mathbf{V}_K = \arg\min_{U,S,V} ||\mathbf{X} - \mathbf{U}_K \mathbf{S}_K \mathbf{V}_K^\intercal||_F, \tag{15.7}$$

meaning that $\mathbf{U}_K, \mathbf{S}_K, \mathbf{V}_K$ give the rank-K matrix $\tilde{\mathbf{X}}$ that minimizes the Frobenius norm, $\sqrt{\sum_{i,j}(x_{i,j} - \tilde{x}_{i,j})^2}$.

This factorization is called Singular Value Decomposition, and is closely related to eigenvalue decomposition of the matrices $\mathbf{X}\mathbf{X}^\intercal$ and $\mathbf{X}^\intercal\mathbf{X}$. In general, the complexity of SVD is $\min(\mathcal{O}(D^2V), \mathcal{O}(V^2N))$. The standard library LAPACK (Linear Algebra PACKage) includes an iterative optimization solution for SVD, and (I think) this what is called by Matlab and Numpy.

However, for large sparse matrices it is often more efficient to take a stochastic gradient approach. Each word-context observation $\langle w, c\rangle$ gives a gradient on $\boldsymbol{u}_w$, $\boldsymbol{v}_c$, and $\boldsymbol{S}$, so we can take a gradient step. This is part of the algorithm that was used to win the Netflix challenge for predicting movie recommendation — in that case, the matrix includes raters and movies (Koren et al., 2009).

Return to NLP applications, the slides provide a nice example from (Deerwester et al., 1990), from titles of computer science research papers. In the example, the context-vector representations of the terms *user* and *human* have negative correlations, yet their distributional representations have high correlation, which is appropriate since these terms have roughly the same meaning in this dataset.

15.5 Neural word embeddings

Discriminatively-trained word embeddings very hot area in NLP. The idea is to replace factorization approaches with discriminative training, where the task may be to predict the word given the context, or the context given the word.

Suppose we have the word w and the context c, and we define

$$u_\theta(w, c) = \exp\left(\boldsymbol{a}_w^\intercal \boldsymbol{b}_c\right) \tag{15.8}$$
$$\tag{15.9}$$

with $\boldsymbol{a}_w \in \mathbb{R}^K$ and $\boldsymbol{b}_c \in \mathbb{R}^K$. The vector $\boldsymbol{a}_w$ is then an **embedding** of the word w, representing its properties. We are usually less interested in the context vector $\boldsymbol{b}$; the context can include surrounding words, and the vector $\boldsymbol{b}_c$ is often formed as a sum of context embeddings for each word in a window around the current word. Mikolov et al. (2013a) draw the size of this context as a random number r.

The popular `word2vec` software[1] uses these ideas in two different types of models:

[1] https://code.google.com/p/word2vec/

Skipgram model In the skip-gram model (Mikolov et al., 2013a), we try to maximize the log-probability of the context,

$$J = \frac{1}{T} \sum_t \sum_{-c \leq j \leq c, j \neq 0} \log \mathrm{p}(w_{t+j} \mid w_j) \tag{15.10}$$

$$\mathrm{p}(w_{t+j} \mid w_j) = \frac{u_\theta(w_{t+j}, w_j)}{\sum_{w'} u_\theta(w', w_j)} \tag{15.11}$$

$$= \frac{u_\theta(w_{t+j}, w_j)}{Z(w_j)} \tag{15.12}$$

This model is considered to be slower to train, but better for rare words.

CBOW The continuous bag-of-words (CBOW) (Mikolov et al., 2013b,c) is more like a language model, since we predict the probability of words given context.

$$J = \frac{1}{T} \sum_t \log \mathrm{p}(w_t \mid c) \tag{15.13}$$

$$= \frac{1}{T} \sum_t \log u_\theta(w_t, c) - \log Z(c) \tag{15.14}$$

$$u_\theta(w_t, c) = \exp \left(\sum_{-c \leq j \leq c, j \neq 0} \boldsymbol{a}_{w_t}^\intercal \boldsymbol{b}_{w_{t+j}} \right) \tag{15.15}$$

The CBOW model is faster to train (Mikolov et al., 2013a). One efficiency improvement is build a Huffman tree over the vocabulary, so that we can compute a hierarchical version of the softmax function with time complexity $\mathcal{O}(\log V)$ rather than $\mathcal{O}(V)$. Mikolov et al. (2013a) report two-fold speedups with this approach.

These models are simplified versions of previous work on recurrent neural network language models (RNNLMs) and the log-bilinear language model (Mnih and Hinton, 2008).

Estimating word embeddings

Training these models can be challenging, because they both probabilities that need to be normalized over the entire vocabulary. This implies a training time complexity of $\mathcal{O}(VK)$ for each instance. Since these models are often trained on hundreds of billions of words, with $V \sim 10^6$ and $K \sim 10^3$, this cost is too high. Estimation techniques eliminate the factor V by making approximations.

One such approximation is negative sampling, which is a heuristic variant of noise-contrastive estimation (Gutmann and Hyvärinen, 2012).

We introduce an auxiliary variable D, where

$$D = \begin{cases} 1, & w \text{ is drawn from the empirical distribution } \hat{\mathrm{p}}(w \mid c) \\ 0, & w \text{ is drawn from the noise distribution } q(w) \end{cases} \tag{15.16}$$

Now we will optimize the objective

$$\sum_{(w,c) \in \mathcal{D}} \log P(D = 1 \mid c, w) + \sum_{i=1, w' \sim q}^{k} \log P(D = 0, \mid c, w'), \tag{15.17}$$

setting

$$P(D = 1 \mid c, w) = \frac{u_\theta(w, c)}{u_\theta(w, c) + k \times q(w)} \tag{15.18}$$

$$P(D = 0 \mid c, w) = 1 - P(D = 1 \mid c, w) \tag{15.19}$$

$$= \frac{k \times q(w)}{u_\theta(w, c) + k \times q(w)}, \tag{15.20}$$

where k is the number of noise samples. Note that we have dropped the normalization $\sum_{w'} u_\theta(w', c)$. This approximation is based on the idea of noise-contrastive estimation, where the normalization term is set to be a parameter z_c. Here we approximate further, assuming $z_c = 1$. This would be trouble if we were trying to directly maximize $\log \mathrm{p}(w \mid c)$, but this is where the auxiliary variable formulation helps us out: if we set θ such that $\sum_{w'} u_\theta(w' \mid c) \gg 1$, we will get a very low probability for $P(D = 0)$.

We can further simplify by setting $k = 1$ and $q(w)$ to a uniform distribution, arriving at

$$P(D = 1 \mid c, w) = \frac{u_\theta(w, c)}{u_\theta(w, c) + 1} \tag{15.21}$$

$$P(D = 0 \mid c, w) = \frac{1}{u_\theta(w, c) + 1} \tag{15.22}$$

The derivative with respect to a is obtained from the objective

$$L = \sum_t \log \mathrm{p}(D = 1 \mid c_t, w_t) + \log \mathrm{p}(D = 0 \mid c, w') \tag{15.23}$$

$$= \sum_t \log u_\theta(w_t, c_t) - \log(1 + u_\theta(w_t, c_t)) - \log(1 + u_\theta(w', c_t)) \tag{15.24}$$

$$\frac{\partial L}{\partial \boldsymbol{a}_i} = \sum_{t:w_t=i} \boldsymbol{b}_{c_t} - \frac{1}{1 + u_\theta(w_t, c_t)} \frac{\partial u_\theta(i, c_t)}{\partial \boldsymbol{a}_i} + \sum_t \frac{q(i)}{1 + u_\theta(i, c_t)} \frac{\partial u_\theta(i, c_t)}{\partial \boldsymbol{a}_i} \tag{15.25}$$

$$= \sum_{t:w_t=i} \boldsymbol{b}_{c_t} - P(D = 1 \mid w_t = i, c_t)\boldsymbol{b}_{c_t} - \sum_t q(i)P(D = 0 \mid i, c_t)\boldsymbol{b}_{c_t} \tag{15.26}$$

$$= \sum_t (\delta(w_t = i) - q(i))P(D = 0 \mid w_t = i, c_t)\boldsymbol{b}_{c_t}. \tag{15.27}$$

The gradient with respect to $\boldsymbol{b}$ is similar. In practice, we simply sample w' at each instance and compute the update with respect to $\boldsymbol{a}_{w_t}$ and $\boldsymbol{a}_{w'}$. In practice, AdaGrad performs well for this optimization.

Connections to matrix factorization

Recent work has drawn connections between this procedure for training the skip-gram model and weighted matrix factorization approaches (Pennington et al., 2014; Levy and Goldberg, 2014). For example, Levy and Goldberg (2014) show that skip-gram with negative sampling is equivalent to factorizing a matrix M, where

$$M_{i,j} = PMI(w_i, c_j) - \log k \tag{15.28}$$

$$PMI(w_i, c_j) = \log \left(\frac{n(w = i, c = j)}{n(w = i)} \frac{|D|}{n(c = j)} \right), \tag{15.29}$$

where k is a constant offset and PMI is the well known pointwise mutual information statistic, usually written as

$$PMI(x, y) = \frac{\mathrm{p}(x, y)}{\mathrm{p}(x)\mathrm{p}(y)}. \tag{15.30}$$

To see the connection, divide both the numerator and denominator by $|D|^2$.

Matrix factorization approaches are computationally advantageous because they can be applied to matrices of word co-occurrence statistics, rather than requiring streaming through an entire dataset.

Chapter 16

Anaphora and Coreference Resolution

Pronouns are one of the most noticeable forms of linguistic ambiguity. A Google search for "ambiguous pronoun" reveals dozens of pages warning you to avoid ambiguity. But as we have seen, people resolve all but the most egregious linguistic ambiguities intuitively, below the level of conscious thought.

Moreover, reference ambiguities need not apply only to pronouns. Consider the following text:

> Apple Inc Chief Executive Tim Cook has jetted into China for talks with government officials as **he**$_1$ seeks to clear up a pile of problems in **[[the firm's]**$_2$ **biggest growth market]**$_3$.

- Who is referred to by *he*$_1$?
- What entity is referred to by *the firm*$_2$?
- What is Apple's biggest growth market?

You probably answered these questions by making some commonsense assumptions. Tim Cook is the only individual mentioned, so the personal pronoun *he* probably refers to him; Apple is the only firm mentioned, so *the firm* probably refers to it; a CEO wouldn't fly to China in order to resolve problems in some other growth market, so *the firm's biggest growth market* probably refers to China.[1][this is

[1]These judgments are formalized in Grice's Maxim of Quantity: make your contribution as informative as required, but not more so.

not a great example; try to find one with ambiguity that requires more than Grice to resolve.]

We can use this example to introduce some terminology:

Referring expressions include *he*, *Tim Cook*, *the firm*, *the firm's biggest growth market*. These are surface strings in the text.

Referents include TIM-COOK, APPLE, CHINA; in formal semantics, these may be viewed as objects in a model, such as a database of entities. But referents need not always be entities, as we will see.

Coreference is a property of pairs of referring expressions, which holds when they refer to the same underlying entity.

Anaphora are referring expressions whose meaning depends on another expression in context, which occurs earlier in the document or talk. **Cataphora** refer to expressions that occur later in the document, like *After she won the lottery, Susan quit her job.* **Exophora** refer to entities not defined in the linguistic context.

16.1 Forms of referring expressions

There are many possibilities for describing a referent.

- Indefinite NPs: *a visit*, *two stores*
- Definite NPs: *the capital*, *his first trip*
- Pronouns: *he, it*
- Demonstratives: *this chainsaw*, *that abandoned mall*
- Names: *Tim Cook*, *China*

How do you know which type of referring expression to use?

- **Language generation** requires getting this right.
  You can't say: *Rob Ford apologized for "a lot of stupid things" but Rob Ford only acknowledged a video showing Rob Ford smoking what appears to be crack cocaine to demand police release it.*

- The **specific** referring expression within a type is determined by syntax and semantic constraints.
- The **type** of referring expression (pronoun, name, etc) is largely determined by the discourse.

One theory about the relationship between discourse structure and forms of referring expressions is the Givenness Hierarchy (Gundel et al., 1993). This theory is based on the **status** of the referent with respect to both the discourse and the hearer.

Type identifiable (you know what dogs are): indefinite
I couldn't sleep, ***a dog*** *kept me awake.*

Referential (some particular dog): indefinite *this*
I couldn't sleep, ***this dog*** *kept me awake.*

Uniquely identifiable definite
I couldn't sleep, ***the neighbor's dog*** *kept me awake.*

Familiar distal demonstrative
That dog *next door kept me awake all night.*

Activated demonstrative
My neighbor bought a new dog, and ***that dog*** *kept me awake last night.*

In focus pronoun
Her dog barks constantly. ***It*** *kept me awake all night.*

The location of an entity in the givenness hierarchy depends (in part) on the discourse:

- *You look tired, did a dog keep you awake?*
- *We bought a dog. It keeps me up all night.*
- Referents which were recently accessed acquire *salience*, and are more likely to be near the top of the givenness hierarchy.

However, background knowledge also plays an important role.

- If a pair of speakers lives with a (single) dog, it is always at least uniquely identifiable.

- Entities may be **inferrable** from the discourse:

  She just bought a new bike.
  The wheels are made of bamboo fiber.

Centering theory (Grosz et al., 1995) formalizes the notion of salience, by incorporating the syntactic role of each referring expression.

At each utterance U_n, we have:

- A backward-looking center $C_b(U_n)$:
  the entity currently **in focus** after U_n.
- A forward-looking center $C_f(U_n)$:
  an ordered list of candidates for $C_b(U_{n+1})$.
- The top choice in $C_f(U_n)$ is $C_p(U_{n+1})$

How do we order the candidates from $C_b(U_{n+1})$ to the forward-looking center? By syntax:

1. Subject
   Abigail *saw an elephant.*
2. Existential predicate nominal
   There is ***an elephant*** *in the room.*
3. Direct object
   Abigail gave ***a snack*** *to the elephant.*
4. Indirect object or oblique
   Abigail gave a snack to ***the elephant****.*
5. demarcated adverbial prepositional phrase
   Inside ***the zoo****, the elephant is king.*

Rule: If any element of $C_f(U_n)$ is realized by a pronoun in U_{n+1}, then $C_b(U_{n+1})$ must also be realized as a pronoun.

- Generate possible C_b and C_f for each set of reference assignments
- Filter by constraints: syntax, semantics, and centering rules

- Rank by transition orderings: continue, retain, smooth-shift, rough-shift

	$C_b(U_{n+1}) = C_b(U_n)$ or $C_b(U_n) = \varnothing$	$C_b(U_{n+1}) \neq C_b(U_n)$
$C_b(U_{n+1}) = C_p(U_{n+1})$	Continue	Smooth-shift
$C_b(U_{n+1}) \neq C_p(U_{n+1})$	Retain	Rough-shift

In a coherent discourse, we select transitions according to the following preferences: continue, retain, smooth-shift, rough-shift

Here's an example of how to use centering to resolve pronouns.

U_n	$C_f(U_n)$	$C_p(U_n)$	$C_b(U_n)$	transition
John saw a beautiful Masi at the bike shop	John, Ford, bike shop	John	$\varnothing$	
He showed it to Bob	John, Masi, Bob	John	John	Continue
He showed it to Bob	John, bike shop, Bob	John	John	Continue
He bought it	John, Masi or bike shop	John	John	Continue
He bought it	Bob, Masi or bike shop	Bob	Bob	Smooth-shift

- Centering theory tells us that we prefer *John* over *Bob* as the referent for *he* in U_3, because this would be a continue transition rather than a smooth-shift.
- Centering doesn't really give us a rule for choosing *Masi* over *bike shop* in U_2, because neither is $C_b(U_2)$. We might apply the grammatical role hierarchy since there is no other basis for this decision.

16.2 Pronouns and reference

Are all referents entities? Nope.

- *They told me that I was too ugly, but I didn't believe **it**.*
- *Alice saw Bob get angry, and I saw **it** too.*
- *They told me that I was too ugly, but **that** was a lie.*
- *Jess said she worked in security.*
  *I suppose **that**'s one way to put it.*

Are all pronouns referential? Also no.

Cataphora are references to entities which are evoked *after* the reference.

When she learned what had happened, Alice took the first bus out of town.

Some pronouns have **generic** referents:

- *A good father takes care of **his** kids.*
- *I want to buy a Porsche, **they** are so fast.*
- *On the moon, **you** have to carry **your** own oxygen.*

Some pronouns don't refer to anything at all:

- Pleonastic: ***It**'s raining. **It**'s crazy out there.*
- Cleft: ***It**'s money that she's really after.*
- Extraposition: ***It** sucks that we have to work so hard.*
- Other languages:
  - *S'**il** vous plaît* (literally: *if it pleases you*)
  - *Wie geht es Ihnen*

How to distinguish these from referential pronouns? Bergsma et al. (2008) propose a substitutability text.

- *You can make it in advance → You can make **them** in advance*
- *You can make it in Hollywood → You can make **them** in Hollywood*

Specifically, consider 5-gram context patterns.
*... said here Thursday that **it** is unnecessary to continue*

said	here	Thursday	that	*				
	here	Thursday	that	*	is			
		Thursday	that	*	is	unnecessary		
			that	*	is	unnecessary	to	
				*	is	unnecessary	to	continue

For each pattern, compute the corpus counts of five **pattern fillers**:

1. *it/its*
2. *they/them/their*
3. other pronouns *she/her/...*

4. rare words (almost always nouns)
5. all other tokens (usually nouns)

These 25 counts are converted into a feature vector, and you can train a supervised classifier.

16.3 Resolving ambiguous references

Anaphora resolution is primarily concerned with pronouns like *it, this, her*

Coreference resolution adds two additional phenomena

- **Names**: *Barack Obama, Obama, President Obama, Barry O, Nobama*
- **Nominals**: *the 44th president, the former senator from Illinois, our first African-American president*

Let's go back to our example:
Apple Inc Chief Executive Tim Cook has jetted into China for talks with government officials as **he** seeks to clear up a pile of problems in the firm's biggest growth market, from **its** contested iPad trademark to treatment of local labor. Cook is on **his** first trip to the country...

- *he* $\stackrel{?}{=}$ *Apple Inc, Tim Cook, China, talks, government officials, government, ...*
- *its* $\stackrel{?}{=}$ *the firm's biggest growth market, the firm, problems, a pile of problems, ...*
- *his* $\stackrel{?}{=}$ *Cook, local labor, its contested iPad trademark, iPad, ...*

How can we resolve these pronouns?

Semantic constraints

- **Number**
  - *Tim Cook has jetted in for talks with officials as **he** seeks to clear up a pile of problems...*
    - Number(*he*) = singular

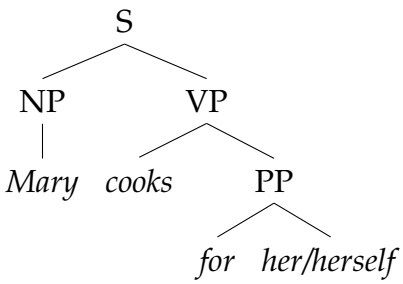
    - Number(*officials*) = plural
    - Number(*Tim Cook*) = singular
  - Mass noun are tricky: *New York has won the superbowl. They are the world champions.*

- **Person**: *We$_1$ *told* them$_1$ *not to go*.
- **Gender and animacy**
  - *Sally met my brother. He charmed her.*
  - *Sally met my brother. She charmed him.*
  - *Putin brought a bottle of vodka. It was from Russia.*

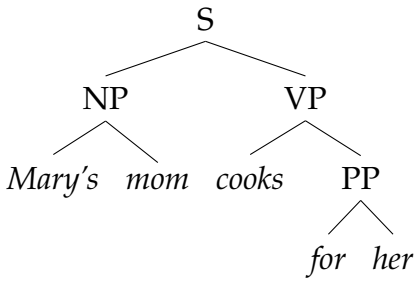
Syntactic constraints

There are general constraints on reference within sentences, which seem to generalize well across languages.

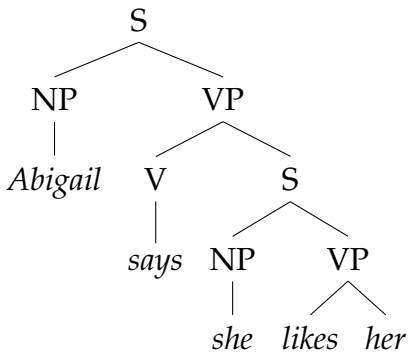
- x **c-commands** y iff the first branching node above x also dominates y.
- x **binds** y iff x and y are co-indexed and x c-commands y
- if y is not bound, it is **free**



- *Mary* c-commands *her/herself*.
- *her/herself* does not c-command *Mary*.
- *her* **cannot** refer to *Mary*, because pronouns cannot refer to antecedents that c-command them.
- *herself* **must** refer to *Mary*.



- *Mary* does **not** c-commands *her*
- *Mary's mom* c-commands *her*
- *her* **can** refer to *Mary* (and we cannot use reflexive *herself* in this context, unless we are talking about Mary's mom)
- But it doesn't have to, because pronouns can be free.



Constraints have a limited domain.

- *she* can refer to *Abigail*
- *her* can also refer to *Abigail*
- But *she* and *her* cannot be coreferent.

Besides these rules, syntax also exercises preferences. See slides.

Combining the evidence

Three **types** of evidence:

- Semantic constraints

- Syntactic constraints
- Discourse/salience preferences

How do we combine them?

- **Hobbs**: Tree search + constraints

  Walk back through the tree in a deterministic order, select the first referent that satisfies the constraints.

- **Centering**: ordered preferences + constraints

  Apply centering theory to recover the references that give the most preferred transition sequence, subject to semantic constraints.

- **Lappin and Lease**: numerical preferences + constraints

  Basically a hand-tuned linear classifier.

  - -100 for each intervening sentence
  - +80 for subject position
  - +70 for existential emphasis, e.g. *there was a woman who...*
  - +50 for accusative emphasis
  - ...

- Ge, Hale, and Charniak (1999): statistical combination of four probabilities

  - probability of the "Hobbs distance" between pronoun and antecedent
  - probability of the pronoun given the antecedent
    (this considers gender and animacy)
  - how well the proposed antecedent fills the pronoun's slot in the sentence
  - frequency of the proposed referent

- Raghunathan et al. (2010) describe a "multipass sieve" for coreference resolution, which applies a series of progressively relaxed matching rules.

16.4 Coreference resolution

This is a generalization of the anaphora resolution task to cover proper nouns and nominals.

- See the slides for an example.
- The coreference task comes from the information extraction community.
- Candidate spans of text for coreference are called **markables**
- In the harder versions of the coreference task, you have to identify the markables as well as their reference chains.

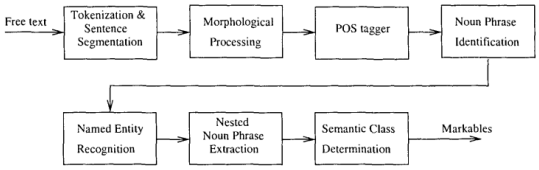
Coreference combines many phenomena: all the ones in anaphora resolution, plus string similarity and knowledge to get nominals.

- *unencrypted Wi-Fi networks* and *networks* have the same head word
- *Dr. King* and *Martin Luther King* can all co-refer
- *Martin Luther King* and *Coretta Scott King* cannot
- **World knowledge**: e.g., *Google* is a *company*, companies possess *cars* but *Tuesday* doesn't.

The mention-pair model

One of the earliest end-to-end machine learning systems for coreference is from Soon et al. (2001).

- Identify markables and their features with an NLP pipeline.



- Train a classifier to predict which pairs of markables corefer. This is the **mention-pair** model.

  - For each markable, go backwards until the classifier selects an antecedent or you reach the beginning of the document.
  - No structured prediction here; each classification decision is made independently.

Learning is performed on mention pairs.

- Given the labeled chain A1-A2-A3-A4, the adjacent pairs A1-A2, A2-A3, A3-A4 are treated as positive examples.
- Negative examples are generated from NPs that occur between the adjacent pairs.
  - Suppose markables A,B,B1 appear between A1 and A2.
  - Then the negative examples are: A-A2, B-A2, B1-A2.

There are fundamental problems with mention-pair approaches.

- They fail to aggregate information across the chain.
- Must reason about transitivity to avoid incoherent chains.
- *Michelle Obama* ← *Obama* ← *Mr. Obama*

Entity-based coreference

Alternatively, we can try to learn at the entity level, using features of the entities themselves

- Number of entities detected so far
- Mention to entity ratio
- Entity to word ratio
- Number of intervening mentions between mention and linked entity
- ...

Can incorporate these by scoring entire clusterings, $\boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$.
But how to train such a model?

One approach is an incremental perceptron. This is like a structured perceptron, but you incrementally build the structure, and you update as soon as you make a mistake.

Bell Tree, Beam Search, and Max-link Coreference The Bell Tree can represent the coreference structure. See slides.

Markov Random Field with Transitive Closure see slides

Summing over antecedent structures Durrett and Klein (2013) propose summing over reference assignments within a clustering. Let the gold standard clustering be written C^*, with C_k^* representing the cluster for document k, and $\mathcal{A}(C_k^*)$ representing the set of possible antecedents structures. Then we treat the specific antecedent structure as a latent variable, and sum over it, obtaining the regularized objective,

$$\ell(\boldsymbol{\theta}) = \sum_{k}^{N} \log \left(\sum_{a \in \mathcal{A}(C_k^*)} \mathrm{p}(a \mid x_k) \right) + \lambda ||\boldsymbol{\theta}|| \tag{16.1}$$

$$\mathrm{p}(a \mid x_k) \propto \exp \left(\sum_i \boldsymbol{\theta}^\mathsf{T} \boldsymbol{f}(i, a_i, \boldsymbol{x}) \right). \tag{16.2}$$

Durrett and Klein (2013) augment this basic model by defining a real-valued loss function, and incorporating it into the objective. They then show that this basic framework supports a number of expressive features, which give good performance compared to prior work.

16.5 Coreference evaluation

16.6 Multidocument coreference resolution

Chapter 17

Semi-supervised learning and domain adaptation

So far we have focused on learning a classifier — typically represented by a set of weights $\boldsymbol{\theta}$ — from a set of labeled examples $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{\ell}$. As we've seen, it's possible to formulate structured prediction tasks such as parsing in this same framework. But what if you don't have those labeled examples for the domain or task that you want to solve?

- You can use some other labeled data and hope it works.
  This rarely works well.
- You can label data yourself.
  This is a lot of work.

This kind of thing happens all the time (especially in class projects). And labeled data is really expensive:

- **The Switchboard corpus** contains phoneme annotations of telephone conversations, e.g.

  $$\textit{film} \rightarrow \text{F IH\_N UH\_GL\_N M}$$
  $$\textit{be all} \rightarrow \text{BCL B IY IY\_TR AO\_TR AO L\_DL}$$

  This took 400 hours of annotation time per hour of speech.
- **The Penn Chinese Treebank** is a set of CFG annotations for Chinese. It took 2 years to get 4000 sentences annotated.

17.1 Learning with less annotation effort

We can think of our annotated data as a *sample* from some underlying distribution.

$$\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{\ell} \sim \mathcal{D} \tag{17.1}$$

This allows us to formulate various learning scenarios:

Semisupervised learning

- $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{\ell} \sim \mathcal{D}$: labeled examples
- $\{(\boldsymbol{x}_i)\}_{i=\ell+1}^{\ell+u}$: unlabeled examples
- often $u \gg \ell$

We've already seen an example of semi-supervised learning in document classification, when we applied expectation maximization to impute labels of unlabeled documents. Today we will see some approaches that tend to work better than EM.

Active learning is closely related to semi-supervised learning, but you can now query the labels for a few examples while learning. Your goal is to select these examples carefully, so that you get the best accuracy from a fixed number of examples, or so that you get to a certain level of accuracy with as few examples as possible.

Domain adaptation

- $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{\ell_S} \sim \mathcal{D}_S$: labeled examples in *source* domain
- Some *target domain* $\mathcal{D}_T$. You may have a small amount of labeled data in the target domain ("supervised domain adaptation") or not ("unsupervised domain adaptation").

  The prototypical example for domain adaptation is product reviews: you have annotated reviews of coffee machines, but you want to train a classifier for reviews of bicycles. Another example is that you have a POS tagger for news genre text, but you want one for social media.

- Suppose that $\mathrm{p}_T(X) = \mathrm{p}_S(X)$, i.e. the distribution over observed data is the same in both domains, but $\mathrm{p}_T(Y|X) \neq \mathrm{p}_S(Y|X)$, i.e. the conditional

distribution over labels is different. This setting is sometimes called **transfer learning** or **multitask learning**.

For example, you have labeled data for part-of-speech tagging, but you want to train a system for named entity recognition. Or, you have a classifier for detecting mentions of people and you want one for detecting mentions of places.

17.2 Why would unlabeled data help?

Suppose you want to do sentiment analysis in French. I give you two labeled examples:

- ☺ émouvant avec grâce et **style**
- ☹ fastidieusement inauthentique et **banale**

You have a bunch of unlabeled examples too:

1. pleine de **style** et d'**intrigue**
2. la **banalité** n'est dépassée que par sa **prétention**
3. **prétentieux**, de la première minute au rideau final
4. imprégné d'un air d'**intrigue**

What can we do? If we just learn from the labeled data, we might decide that *style* is positive and that **banale** is negative. This isn't much. However, we can propagate this information to the unlabeled data, and potentially learn more.

- If we are confident about *style* being positive, then we can guess that example 1 is also positive.
- That suggests that *intrigue* is also positive.
- We can then propagate this information to example 4, and learn more.
- Similarly, we can propagate from the labeled data to example 2, which we guess to be negative. This suggests that *pretention* is also negative, which we propagate to example 3.

What happened here?

- Instances 1 and 2 were "similar" to our labeled examples for positivity and negativity, respectively. We used them to expand those concepts, which allowed us to correctly label instances 3 and 4, which didn't share any important features with our original labeled data.
- We made a key assumption: that similar instances will have similar labels. Is this a strong assumption? Keep this question in mind.
- In this case, we defined similarity in terms of sharing some key words (non-stopwords).
- To see how this can help conceptually, think about similarity just in terms of 1D space. If you have only the two labeled instances, your decision boundary should be right in between. (Do you remember what criterion justifies this choice?) But if you have a bunch of unlabeled instances, you might want to draw this boundary in a different place.
- Let's see how we can operationalize this idea in an algorithm.

Semi-supervised learning with EM

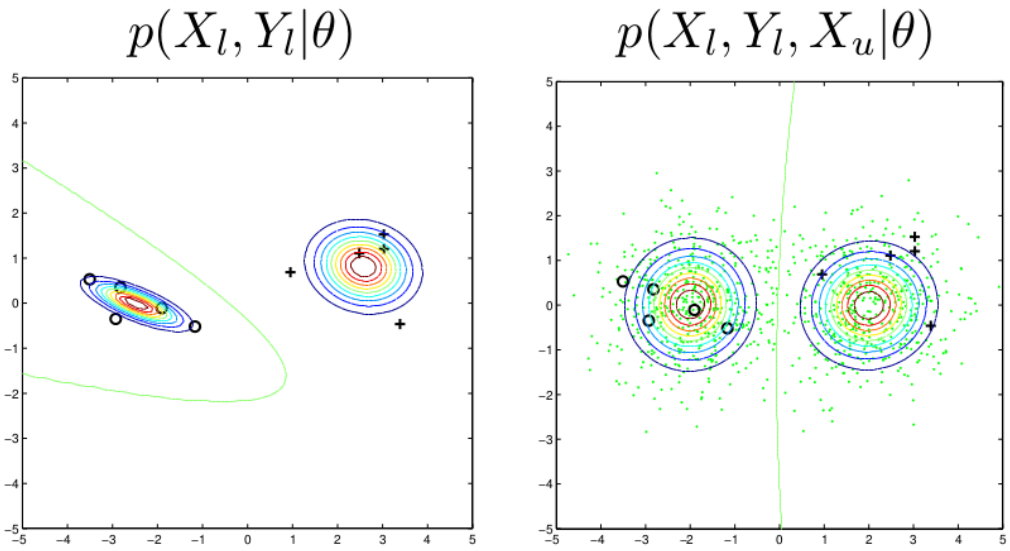
We've already seen one way to do this: use expectation-maximization to marginalize over the labels of the unseen data. So we are maximizing

$$\text{p}(X^\ell, Y^\ell, X^U) = \text{p}(X^\ell, Y^\ell) \sum_{Y^U} \text{p}(X^U, Y^U) \tag{17.2}$$

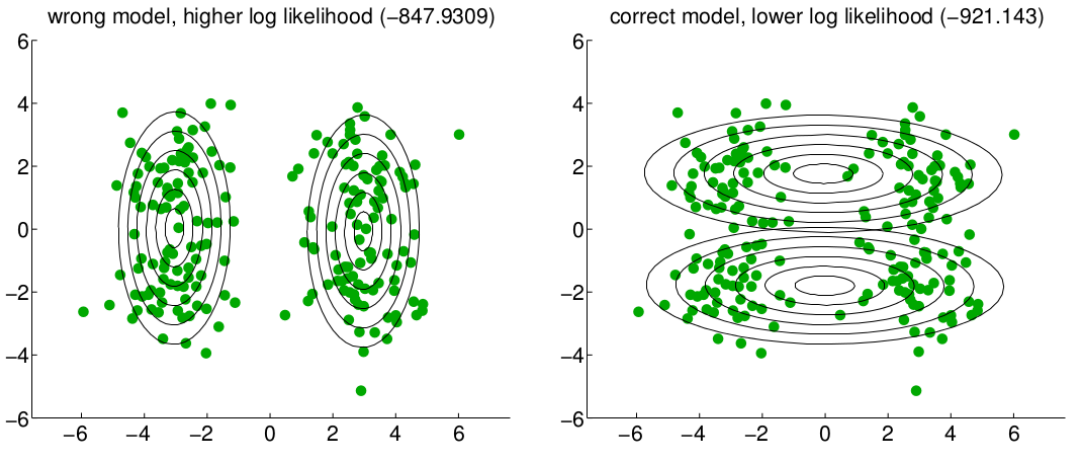
We did this by

- **E**. fitting a distribution $Q(y_i)$ for all unlabeled i,
- **M**. maximizing the expected likelihood under this distribution.

You can see why this can work in an example:



We get a much more reasonable decision boundary. However, things can also go wrong:



The correct model has a lower log-likelihood than the incorrect model. The basic problem is that our model is wrong. The label is related to the observations, but not in the simplistic, Gaussian way that we had assumed. We discussed a heuristic to try to deal with this problem: downweighting the contribution of the unseen data to the likelihood function.

Bootstrapping

EM is sort of like self-training or **bootstrapping**: we use our current model to estimate $Q(y_i)$, and then update the model as if $Q(y_i)$ is correct.

- The probabilistic nature of this is nice, but it limits us to relatively weak classifiers.
- If we are willing to give up on probability, we can bootstrap **any** classifier.

We can try this first using one nearest-neighbor (see slides). Like EM, it can work, but it can also fail. (The failure modes are different though; can you characterize the difference?)

Co-training

"Folk wisdom" about when bootstrapping works:

- Better for generative models (e.g., Naive Bayes) than for discriminative models (e.g., perceptron)
- Better when the Naive Bayes assumption is stronger.
  - Suppose we want to classify NEs as PERSON or LOCATION
  - Features: string and context
    - *located on Peachtree Street*
    - *Dr. Walker said ...*

$$\begin{aligned} &P(X_1 = \textit{street}, X_2 = \textit{on} \mid Y_1 = \text{LOC}) \\ &\quad \approx P(X_1 = \textit{street} \mid Y_1 = \text{LOC}) P(X_2 = \textit{on} \mid Y_1 = \text{LOC}) \end{aligned}$$

Cotraining makes the bootstrapping assumptions explicit.

- Assume two, **conditionally independent**, views of a problem.
- Assume each view is sufficient to do good classification.

Sketch of learning algorithm:

- On labeled data, minimize error.
- On unlabeled data, **constrain** the models from different views to agree with each other.

Co-training example See the slides for an animated version of this. Assume we want to do named entity classification: determine whether an NE is a Location or Person. We have two views: the name itself, and its context.

	$\boldsymbol{x}^{(1)}$	$\boldsymbol{x}^{(2)}$	y
1.	Peachtree Street	located on	LOC
2.	Dr. Walker	said	PER
3.	Zanzibar	located in	? $\rightarrow$ LOC
4.	Zanzibar	flew to	? $\rightarrow$ LOC
5.	Dr. Robert	recommended	? $\rightarrow$ PER
6.	Oprah	recommended	? $\rightarrow$ PER

Algorithm

- Use classifier 1 to label example 5.
- Use classifier 2 to label example 3.
- Retrain both classifiers, using newly labeled data.
- Use classifier 1 to label example 4.
- Use classifier 2 to label example 6.

Multiview Learning is another approach in this style.

- Co-training treats the output of each view's classifier as a labeled instance for the other view.
- In multiview learning, we add a **co-regularizer** that penalizes disagreement between the views on the unlabeled instances.
- This allows us to define a single objective function. In the case of two-view linear regression, the function is

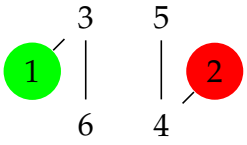
$$\min_{w,v} \sum_{i}^{L} (y_i - \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i^{(1)})^2 + (y_i - \boldsymbol{v}^\mathsf{T}\boldsymbol{x}_i^{(2)})^2 + \lambda_1||\boldsymbol{w}||^2 + \lambda_1||\boldsymbol{v}||^2 + \lambda_2 \sum_{i=L+1}^{L+U} (\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i^{(1)} - \boldsymbol{v}^\mathsf{T}\boldsymbol{x}_i^{(2)})^2$$

The only difference from standard regression is the co-regularizer, which penalizes disagreement on the unlabeled data.

- An early version of this idea is **co-boosting** (Collins and Singer, 1999), where each view is a boosting classifier, and features are added incrementally to each view.

Graph-based approaches

Let's go back to sentiment analysis in French.



We can view this data as a **graph**, with edges between similar instances. Unlabeled instances propagate information through the graph.

Where does the graph come from?

- Sometimes there is a natural similarity metric (time, position in the document).
- Otherwise, we can compute similarity from features. If the features are Gaussian, we could say:

$$\text{sim}(i,j) = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}\right)$$

  If the features are discrete, we might use KL-divergence.

- Then we add an edge between i and j when $\text{sim}(i,j) > \tau$

Given a graph with edge weights s_{ij}, we can formulate semi-supervised learning as an optimization problem:

$$\begin{aligned} y_i &\in \{0,1\} \\ \text{Fix } Y_l &= \{y_1, y_2, \ldots y_\ell\} \\ \text{Solve for } Y_u &= \{y_{\ell+1}, \ldots, y_{\ell+m}\} \\ \min_{Y_u} &\sum_{i,j} s_{ij}(y_i - y_j)^2 \end{aligned}$$

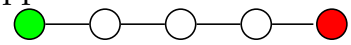
- This looks like a combinatorial problem. Specifically, it looks like (binary) integer linear programming, which is NP-complete.
- But assuming $s_{ij} \geq 0$, this specific problem can be reformulated as maximum-flow, with polynomial time solutions.

Rao and Ravichandran (2009) apply this idea to expanding polarity lexicons.

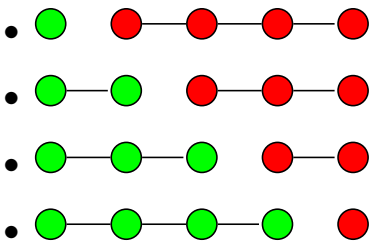
- Nodes are words
- Edges are wordnet relations
- They label a few nodes for sentiment polarity, and propagate those labels to other parts of the graph.
- However, they use a slightly modified version of the mincut idea: randomized min-cut (Blum et al., 2004).

Randomized min-cut

Suppose we have this initial graph:



What is the solution? Actually, the following solutions are all equivalent:



Another problem with mincuts is that it doesn't distinguish high-confidence and low-confidence predictions. Both of these problems can be dealt with by randomization:

- Add random noise to adjacency matrix.
- Rerun mincuts multiple times.
- Deduce the final classification by voting.

Label propagation

A related approach is **label propagation** (Zhu and Ghahramani, 2002), which Rao and Ravichandran also consider. The basic idea is that we relax y_i from an integer $\{0, 1\}$ to a real number $\mathbb{R}$. Then we solve the optimization problem,

$$\min_Y \sum_{i,j} s_{ij}(y_i - y_j)^2$$
$$s.t. Y_L \text{ is clamped to initial values}$$

The advantages are:

- a unique global optimum
- a natural notion of confidence: distance of y_i from 0.5

Let's look at the objective:

$$\begin{aligned}
J =& \frac{1}{2} \sum_{i,j} s_{ij}(y_i - y_j)^2 \\
=& \frac{1}{2} \sum_{i,j} s_{ij}(y_i^2 + y_j^2 - 2y_i y_j) \\
=& \sum_i y_i^2 \sum_j s_{i,j} - \sum_{i,j} s_{ij} y_i y_j \\
=& \boldsymbol{y}^\intercal \mathbf{D} \boldsymbol{y} - \boldsymbol{y}^\intercal \mathbf{S} \boldsymbol{y} \\
=& \boldsymbol{y}^\intercal \mathbf{L} \boldsymbol{y}
\end{aligned}$$

We have introduced three matrices

- Let $\mathbf{S}$ be the $n \times n$ similarity matrix.
- Let $\mathbf{D}$ be the **degree matrix**, $d_{ii} = \sum_j s_{ij}$. $\mathbf{D}$ is diagonal.
- Let $\mathbf{L}$ be the unnormalized **graph Laplacian** $\mathbf{L} = \mathbf{D} - \mathbf{S}$
- So we want to minimize $\boldsymbol{y}^\intercal \mathbf{L} \boldsymbol{y}$ with respect to $\boldsymbol{y}_u$, the labels of the unannotated instances.

In principle, this is easily solveable:

- Partition the Laplacian $\mathbf{L} = \begin{bmatrix} \mathbf{L}_{\ell\ell} & \mathbf{L}_{\ell u} \\ \mathbf{L}_{u\ell} & \mathbf{L}_{uu} \end{bmatrix}$
- Then the closed form solution is $\boldsymbol{y}_u = -\mathbf{L}_{uu}^{-1}\mathbf{L}_{u\ell}\boldsymbol{y}_\ell$
- This is great ... if we can invert $\mathbf{L}_{uu}$.

In practice, $\mathbf{L}_{u,u}$ is huge, so we can't invert it unless it has special structure. Zhu and Ghahramani (2002) propose an iterative solution called **label propagation**.

- Let $\mathbf{T}_{ij} = \frac{s_{ij}}{\sum_k s_{kj}}$, row-normalizing $\mathbf{S}$.
- Let $\mathbf{Y}$ be an $n \times C$ matrix of labels, where C is the number of classes.
- Until tired,
  - Set $\mathbf{Y} = \mathbf{TY}$
  - Row-normalize $\mathbf{Y}$
  - Clamp the seed examples in $\mathbf{Y}$ to their original values
- There's a flavor of EM here, with $\mathbf{Y}$ representing our belief $q_i(y_i)$. But there's no M-step in which we update model parameters. That's because we're in a graph-based framework, and we're assuming the graph is correct.

Both mincut and label propagation are **transductive** learning algorithms: they learn jointly over the training and test data. This is fine in some settings, but not if you want to train a system and then apply it to new test data later — you'd have to retrain it all over again.

Manifold regularization (Belkin et al., 2006) addresses this issue, by learning functions that are smooth on the "graph manifold." In practice, this means that they give similar labels to nearby datapoints in the unlabeled data. Suppose we are interested in learning a classification function f. Then we can optimize:

$$\arg\min_f \frac{1}{\ell}\sum_i \ell(f(\boldsymbol{x}_i), y_i) + \lambda_1||f||^2 + \lambda_2 \sum_{i,j} s_{ij}(f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j))^2$$

- The first term corresponds to the loss on the labeled training data; we can use any convex loss functions, such as logistic or hinge loss.
- The second term corresponds to the smoothness, akin to regularizing the weights in a linear classifier.

- The third term penalizes making different predictions for similar instances in the unlabeled data

The representer theorem guarantees that we can solve for f as long as ℓ is convex. We can then apply f to any new unlabeled test data.

17.3 Domain adaptation

In domain adaptation, we have a lot of labeled data, but it's in the wrong domain. Some features will be shared across domains. For example, if we are classifying movies or toasters, *good* is a good word, and *sucks* is a bad word. But as we've seen, real review text is usually more subtle. What about a word like *unpredictable*? This is a good word for a movie, but not such a good word for a kitchen appliance.

Supervised domain adaptation

In supervised domain adaptation (transfer learning), we have:

- Lots of labeled data in a "source" domain, $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{\ell_S} \sim \mathcal{D}_S$ (e.g., reviews of restaurants)
- A little labeled data in a "target" domain, $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{\ell_T} \sim \mathcal{D}_T$ (e.g., reviews of chess stores)

Here are some (surprisingly-competitive) baselines (see slides)

- Source-only: train on the source data, apply it to the target data.
- Target-only: forget the source data, just train on the limited target data.
- Big blob: merge the source and target data into a single training set. Optionally downweight the source data.
- Prediction: train a classifier on the source data, use its prediction as a feature in the target data.
- Interpolation: train two classifiers, combine their outputs

Here are two less-obvious approaches:

Priors :

Train a (logistic-regression) classifier on the source data. Treat its weights as the priors on the target data, and regularize towards these weights rather than towards zero (Chelba and Acero 2004).

Feature augmentation Create **copies** of each feature, for each domain and for the cross-domain setting.

- The copies fire in the appropriate domains, and the learning algorithm decides whether a feature is general or domain-specific.
- For example, suppose we have domains for Appliances and Movies, and features *outstanding* and *sturdy*. We replicate the features, obtaining

$$\langle \textit{outstanding}, \text{APP.} \rangle, \langle \textit{outstanding}, \text{MOV.} \rangle, \langle \textit{outstanding}, \text{ALL} \rangle$$
$$\langle \textit{sturdy}, \text{APP.} \rangle, \langle \textit{sturdy}, \text{MOV.} \rangle, \langle \textit{sturdy}, \text{ALL} \rangle$$

- Ideally, we will learn a positive weight for $\langle \textit{outstanding}, \text{ALL} \rangle$, because the feature works in both domains, and a small weight for the domain-specific copies of the *outstanding* feature.
- We will also learn a positive weight for $\langle \textit{sturdy}, \text{APP} \rangle$, because the feature works only in the Appliance domain.

See slides for a diagram of how this works.

Unsupervised domain adaptation

Without labeled data in the target domain, can we learn anything? If the source and target domain are somewhat related, then we can. A very popular approach is structural correspondence learning (SCL) (Blitzer et al., 2007).

- Suppose there are a few words that are good predictors in both domains; we'll call these **pivot features**
- Pivot features can be selected by finding words that are
  - Popular in both domains
  - High mutual-information with the label in the source domain

- The label is unknown in the target domain, so we can't learn to predict it. Instead we'll predict the pivots. We train a linear classifier for each pivot, obtaining weights $\boldsymbol{\theta}_n$ for pivot n.
- For example, we can learn that the domain-specific feature *fast-multicore* is a good predictor of the pivot *excellent*.
- We can horizontally concatenate the pivot predictor weights, forming

$$\Theta = [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_N] \tag{17.3}$$

- The matrix Θ is large, and contains redundant information (since many pivots are closely related to each other). We factor $\Theta \approx USV^T$ using singular value decomposition (SVD).
- We use U to **project** features from both domains into a shared space, $U^\intercal \boldsymbol{x}$.
- We then learn to predict the label in the source domain, using the augmented instance $\langle \boldsymbol{x}, U^\intercal \boldsymbol{x} \rangle$. In U contains meaningful correspondences between the domains, then the weights learned on these features will work for the target domain instances too.
- This idea yields substantial improvements in adapting sentiment classifiers across product domains, e.g., books, movies, and appliances (Blitzer et al., 2007).

See the slides for a graphical explanation of these ideas, with slightly different notation.

17.4 Other learning settings

There are many other settings in which we learn from something other than in-domain labeled data:

- **Active learning**. The model can query the annotator for labels (see above)
- **Feature labeling**. Annotators label *features* rather than instances. For example, you provide a list of five prototype words for each POS tag (Haghighi and Klein, 2006).

- **Feature expectations**. Learn from *constraints* on feature-label relationships; for example, the word "the" is a determiner at least 90% of the time. In EMNLP 2013, this idea was applied to multilingual learning (which I'll discuss in the final lecture). The basic idea of this paper is to align words between sentences and insist that aligned words have the same tag most of the time.

- **Multi-instance learning**. The learner gets a "bag" of instances, and a label. If the label is positive, then at least one instance in the bag is positive, but you don't know which one.

  This idea is often related to **distant supervision**. The learner gets a label indicating that there is a relationship, such as BORN-IN(OBAMA, HAWAII), and a set of instances containing sentences that mention the two arguments, *Obama* and *Hawaii*. Many of these sentences do not actually instantiate the desired relation (e.g., *Obama visited Hawaii in 2008...*), but we assume that at least one does, and we must learn from this.

Bibliography

Abney, S. P. and Johnson, M. (1991). Memory requirements and local ambiguities of parsing strategies. *Journal of Psycholinguistic Research*, 20(3):233–250.

Allauzen, C., Riley, M., and Schalkwyk, J. (2009). A generalized composition algorithm for weighted finite-state transducers. In *Proceedings of Interspeech*.

Belkin, M., Niyogi, P., and Sindhwani, V. (2006). Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *The Journal of Machine Learning Research*, 7:2399–2434.

Bender, E. M. (2013). *Linguistic Fundamentals for Natural Language Processing: 100 Essentials from Morphology and Syntax*, volume 6 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool Publishers.

Berger, A. L., Pietra, V. J. D., and Pietra, S. A. D. (1996). A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71.

Bergsma, S., Lin, D., and Goebel, R. (2008). Distributional identification of non-referential pronouns. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 10–18, Columbus, OH.

Bikel, D. M. (2004). Intricacies of collins' parsing model. *Computational Linguistics*, 30(4):479–511.

Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python*. O'Reilly Media.

Blitzer, J., Dredze, M., and Pereira, F. (2007). Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 440–447, Prague.

Blum, A., Lafferty, J., Rwebangira, M. R., and Reddy, R. (2004). Semi-supervised learning using randomized mincuts. In *Proceedings of the International Conference on Machine Learning (ICML)*, page 13.

Bottou, L. (1998). Online learning and stochastic approximations. *On-line learning in neural networks*, 17:9.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, New York, NY, USA.

Brants, T. (2000). Tnt: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, pages 224–231. Association for Computational Linguistics.

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167.

Carreras, X., Collins, M., and Koo, T. (2008). Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 9–16. Association for Computational Linguistics.

Charniak, E. (1997). Statistical techniques for natural language parsing. *AI magazine*, 18(4):33.

Charniak, E. (2000). A maximum-entropy-inspired parser. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 132–139.

Charniak, E. and Johnson, M. (2005). Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 173–180.

Choueka, Y. (1989). Responsa: A full-text retrieval system with linguistic processing for a 65-million word corpus of jewish heritage in hebrew. *Data Engineering*, 12(4):22–31.

Christodoulopoulos, C., Goldwater, S., and Steedman, M. (2010). Two decades of unsupervised pos induction: How far have we come? In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 575–584. Association for Computational Linguistics.

Church, K. W. (2000). Empirical estimates of adaptation: the chance of two Noriegas is closer to $p/2$ than p^2. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 180–186.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 16–23.

Collins, M. (2002). Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pages 1–8.

Collins, M. (2003). Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637.

Collins, M. (2013). Notes on natural language processing. `http://www.cs.columbia.edu/~mcollins/notes-spring2013.html`.

Collins, M. and Brooks, J. (1995). Prepositional phrase attachment through a backed-off model. In *Workshop on Very Large Corpora*.

Collins, M. and Duffy, N. (2001). Convolution kernels for natural language. In *Advances in neural information processing systems*, pages 625–632.

Collins, M. and Singer, Y. (1999). Unsupervised models for named entity classification. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pages 189–196.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585.

Crammer, K. and Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991.

Cui, H., Sun, R., Li, K., Kan, M.-Y., and Chua, T.-S. (2005). Question answering passage retrieval using dependency relations. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 400–407. ACM.

De Marneffe, M.-C. and Manning, C. D. (2008). The stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8. Association for Computational Linguistics.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. (1990). Indexing by latent semantic analysis. *JASIS*, 41(6):391–407.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159.

Durrett, G. and Klein, D. (2013). Easy victories and uphill battles in coreference resolution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Dyer, C. (2014). Notes on adagrad. `www.ark.cs.cmu.edu/cdyer/adagrad.pdf`.

Eisner, J. (2001). Expectation semirings: Flexible em for learning finite-state transducers. In *Proceedings of the ESSLLI workshop on finite-state methods in NLP*.

Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 1–8.

Eisner, J. and Satta, G. (1999). Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 457–464. Association for Computational Linguistics.

Figueiredo, M., Graça, J., Martins, A., Almeida, M., and Coelho, L. P. (2013). LXMLS lab guide. `http://lxmls.it.pt/2013/guide.pdf`.

Finkel, J. R., Grenager, T., and Manning, C. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 363–370.

Finkel, J. R., Kleeman, A., and Manning, C. D. (2008). Efficient, feature-based, conditional random field parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 959–967, Columbus, OH.

Freund, Y., Schapire, R., and Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612.

Freund, Y. and Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296.

Goldwater, S. and Griffiths, T. (2007). A fully bayesian approach to unsupervised part-of-speech tagging. In *Annual meeting-association for computational linguistics*, volume 45.

Grosz, B. J., Weinstein, S., and Joshi, A. K. (1995). Centering: A framework for modeling the local coherence of discourse. *Computational linguistics*, 21(2):203–225.

Gundel, J. K., Hedberg, N., and Zacharski, R. (1993). Cognitive status and the form of referring expressions in discourse. *Language*, pages 274–307.

Gutmann, M. U. and Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, 13(1):307–361.

Haghighi, A. and Klein, D. (2006). Prototype-driven learning for sequence models. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 320–327, New York, NY.

Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Friedman, J., and Tibshirani, R. (2009). *The elements of statistical learning*, volume 2. Springer.

Hindle, D. and Rooth, M. (1990). Structural ambiguity and lexical relations. In *Proceedings of the Workshop on Speech and Natural Language*.

Hsu, D., Kakade, S. M., and Zhang, T. (2012). A spectral algorithm for learning hidden markov models. *Journal of Computer and System Sciences*, 78(5):1460–1480.

Huang, L., Fayong, S., and Guo, Y. (2012). Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151, Montréal, Canada. Association for Computational Linguistics.

Jaeger, T. F. (2010). Redundancy and reduction: Speakers manage syntactic information density. *Cognitive psychology*, 61(1):23–62.

Johnson, M. (1998). Pcfg models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing (2nd Edition) (Prentice Hall Series in Artificial Intelligence)*. Prentice Hall, 2 edition.

Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of Linguistics*, 43(02):365–392.

Karttunen, L. and Beesley, K. R. (2001). A short history of two-level morphology. *ESSLLI-2001 Special Event titled Twenty Years of Finite-State Morphology*.

Karttunen, L. and Beesley, K. R. (2005). Twenty-five years of finite-state morphology. *Inquiries Into Words, a Festschrift for Kimmo Koskenniemi on his 60th Birthday*, pages 71–83.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 423–430.

Knight, K. and May, J. (2009). Applications of weighted automata in natural language processing. In *Handbook of Weighted Automata*, pages 571–596. Springer.

Koo, T. and Collins, M. (2010). Efficient third-order dependency parsers. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 1–11, Uppsala, Sweden.

Koo, T., Globerson, A., Carreras, X., and Collins, M. (2007). Structured prediction models via the matrix-tree theorem. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pages 141–150.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Levy, O. and Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. In *NIPS*.

Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330.

McClosky, D., Charniak, E., and Johnson, M. (2006). Effective self-training for parsing. In *Proceedings of the main conference on human language technology conference of the North American Chapter of the Association of Computational Linguistics*, pages 152–159. Association for Computational Linguistics.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In *Proceedings of International Conference on Learning Representations*.

Mikolov, T., Deoras, A., Povey, D., Burget, L., and Cernocky, J. (2011). Strategies for training large scale neural network language models. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 196–201. IEEE.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013c). Linguistic regularities in continuous space word representations. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 746–751, Atlanta, GA.

Minka, T. P. (1999). From hidden markov models to linear dynamical systems. In *Tech. Rep. 531, Vision and Modeling Group of Media Lab, MIT*.

Mitra, S. and Acharya, T. (2007). Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(3):311–324.

Mnih, A. and Hinton, G. E. (2008). A scalable hierarchical distributed language model. In *Neural Information Processing Systems (NIPS)*, pages 1081–1088.

Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88.

Muralidharan, A. and Hearst, M. A. (2013). Supporting exploratory text analysis in literature study. *Literary and linguistic computing*, 28(2):283–295.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

Nakagawa, T., Inui, K., and Kurohashi, S. (2010). Dependency tree-based sentiment classification using crfs with hidden variables. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 786–794, Los Angeles, CA.

Nemirovski, A. and Yudin, D. (1978). On Cezari's convergence of the steepest descent method for approximating saddle points of convex-concave functions. *Soviet Math. Dokl.*, 19.

Neuhaus, P. and Bröker, N. (1997). The complexity of recognition of linguistically adequate dependency grammars. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 337–343.

Nigam, K., McCallum, A. K., Thrun, S., and Mitchell, T. (2000). Text classification from labeled and unlabeled documents using em. *Machine learning*, 39(2-3):103–134.

Nivre, J. (2004). Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics.

Nivre, J. and Nilsson, J. (2005). Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 99–106. Association for Computational Linguistics.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pages 1532–1543.

Pereira, F. (2000). Formal grammar and information theory: together again? *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 358(1769):1239–1253.

Pereira, F., Tishby, N., and Lee, L. (1993). Distributional clustering of english words. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 183–190. Association for Computational Linguistics.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the Association for Computational Linguistics (ACL)*.

Petrov, S., Das, D., and McDonald, R. (2012). A universal part-of-speech tagset. In *Proceedings of LREC*.

Petrov, S. and Klein, D. (2007). Improved inference for unlexicalized parsing. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 404–411.

Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Raghunathan, K., Lee, H., Rangarajan, S., Chambers, N., Surdeanu, M., Jurafsky, D., and Manning, C. (2010). A multi-pass sieve for coreference resolution. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pages 492–501, Cambridge, MA.

Rao, D. and Ravichandran, D. (2009). Semi-supervised polarity lexicon induction. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 675–682.

Ratnaparkhi, A., Reynar, J., and Roukos, S. (1994). A maximum entropy model for prepositional phrase attachment. In *Proceedings of the workshop on Human Language Technology*, pages 250–255. Association for Computational Linguistics.

Roark, B., Saraclar, M., and Collins, M. (2007). Discriminative¡ i¿ n¡/i¿-gram language modeling. *Computer Speech & Language*, 21(2):373–392.

Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language modelling. *Computer Speech & Language*, 10(3):187–228.

Saul, L. and Pereira, F. (1997). Aggregate and mixed-order markov models for statistical language processing. In *emnlp*.

Schmid, H. (1994). Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the international conference on new methods in language processing*, volume 12, pages 44–49. Manchester, UK.

Shieber, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–343.

Shwartz, S. S., Singer, Y., and Srebro, N. (2007). Pegasos: Primal estimated sub-GrAdient SOlver for SVM. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 807–814.

Socher, R., Bauer, J., Manning, C. D., and Ng, A. Y. (2013). Parsing With Compositional Vector Grammars. In *Proceedings of the Association for Computational Linguistics (ACL)*, Sophia, Bulgaria.

Song, L., Boots, B., Siddiqi, S. M., Gordon, G. J., and Smola, A. J. (2010). Hilbert space embeddings of hidden markov models. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 991–998.

Soon, W. M., Ng, H. T., and Lim, D. C. Y. (2001). A machine learning approach to coreference resolution of noun phrases. *Computational linguistics*, 27(4):521–544.

Sra, S., Nowozin, S., and Wright, S. J. (2012). *Optimization for machine learning*. MIT Press.

Taskar, B., Guestrin, C., and Koller, D. (2003). Max-margin markov networks. In *NIPS*.

Tausczik, Y. R. and Pennebaker, J. W. (2010). The psychological meaning of words: Liwc and computerized text analysis methods. *Journal of Language and Social Psychology*, 29(1):24–54.

Teh, Y. W. (2006). A hierarchical bayesian language model based on pitman-yor processes. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 985–992.

Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104. ACM.

Van Gael, J., Vlachos, A., and Ghahramani, Z. (2009). The infinite hmm for unsupervised pos tagging. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pages 678–687, Singapore.

Wilson, T., Wiebe, J., and Hoffmann, P. (2005). Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pages 347–354.

Zelenko, D., Aone, C., and Richardella, A. (2003). Kernel methods for relation extraction. *The Journal of Machine Learning Research*, 3:1083–1106.

Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM.

Zhu, X. and Ghahramani, Z. (2002). Learning from labeled and unlabeled data with label propagation. Technical report, Technical Report CMU-CALD-02-107, Carnegie Mellon University.