Natural Language Processing

Info 159/259
Lecture 15: Review (March 10, 2020)

David Bamman, UC Berkeley
Big ideas

• Classification
  • Naive Bayes, Logistic regression, feedforward neural networks, CNN.

• Where does NLP data come from?
  • Annotation process
  • Interannotator agreement

• Language modeling
  • Markov assumption, featurized, neural

  • Probability/statistics in NLP
    • Chain rule of probability, independence, Bayes’ rule
Big ideas

• Lexical semantics and word representations
  • Distributional hypothesis
  • Distributed representations
  • Subword embedding models
  • Contextualized word representations (ELMO)
  • BERT
• Evaluation metrics (accuracy, precision, recall, F score, perplexity, parseval)

• Sequence labeling
  • POS, NER
  • Methods: HMM, MEMM, CRF, RNN, BiRNN, BERT

• Trees
  • Phrase-structure parsing, CFG, PCFG
  • CKY for recognition, parsing
Big ideas

• What defines the models we’ve seen so far? What formally distinguishes an HMM from an MEMM? How do we train those models?

• For all of the problems we’ve seen (sentiment analysis, POS tagging, phrase structure parsing), how do we evaluate the performance of different models?

• If faced with a new NLP problem, how would you decide between the alternatives you know about? How would you adapt an MEMM, for example, to a new problem?
Midterm

• Take-home exam — you’ll have 80 continuous minutes to take the midterm anytime within the 24-hour period (3:30pm 3/12—2:59pm 3/13).

• Mix of multiple choice, short answer, long answer

• Covers all material from lectures and readings
Classification

A mapping $h$ from input data $x$ (drawn from instance space $\mathbf{x}$) to a label (or labels) $y$ from some enumerable output space $\mathbf{y}$

$\mathbf{x} = \text{set of all documents}$
$\mathbf{y} = \{\text{english, mandarin, greek, ...}\}$

$x = \text{a single document}$
$y = \text{ancient greek}$
# Text categorization problems

<table>
<thead>
<tr>
<th>task</th>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>language ID</td>
<td>text</td>
<td>{english, mandarin, greek, …}</td>
</tr>
<tr>
<td>spam classification</td>
<td>email</td>
<td>{spam, not spam}</td>
</tr>
<tr>
<td>authorship attribution</td>
<td>text</td>
<td>{jk rowling, james joyce, …}</td>
</tr>
<tr>
<td>genre classification</td>
<td>novel</td>
<td>{detective, romance, gothic, …}</td>
</tr>
<tr>
<td>sentiment analysis</td>
<td>text</td>
<td>{postive, negative, neutral, mixed}</td>
</tr>
</tbody>
</table>
Bayes’ Rule

Prior belief that \( Y = y \)

(before you see any data)

\[
P(Y = y|X = x) = \frac{P(Y = y)P(X = x|Y = y)}{\sum_y P(Y = y)P(X = x|Y = y)}
\]

Likelihood of the data
given that \( Y = y \)

Posterior belief that \( Y = y \) given that \( X = x \)
Bayes’ Rule

\[
P(Y = y | X = x) = \frac{P(Y = y) P(X = x | Y = y)}{\sum_y P(Y = y) P(X = x | Y = y)}
\]

- Prior belief that \(Y = \text{positive}\) (before you see any data)
- Likelihood of “really really the worst movie ever” given that \(Y=\text{positive}\)
- Posterior belief that \(Y=\text{positive}\) given that \(X=\text{“really really the worst movie ever”}\)
- This sum ranges over \(y=\text{positive} + y=\text{negative}\) (so that it sums to 1)
Logistic regression

\[ P(y = 1 \mid x, \beta) = \frac{1}{1 + \exp \left( - \sum_{i=1}^{F} x_i \beta_i \right)} \]

output space \( \mathcal{Y} = \{0, 1\} \)
\[ x = \text{feature vector} \]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>0</td>
</tr>
<tr>
<td>and</td>
<td>0</td>
</tr>
<tr>
<td>bravest</td>
<td>0</td>
</tr>
<tr>
<td>love</td>
<td>0</td>
</tr>
<tr>
<td>loved</td>
<td>0</td>
</tr>
<tr>
<td>genius</td>
<td>0</td>
</tr>
<tr>
<td>not</td>
<td>0</td>
</tr>
<tr>
<td>fruit</td>
<td>1</td>
</tr>
<tr>
<td>BIAS</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \beta = \text{coefficients} \]

<table>
<thead>
<tr>
<th>Feature</th>
<th>( \beta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>0.01</td>
</tr>
<tr>
<td>and</td>
<td>0.03</td>
</tr>
<tr>
<td>bravest</td>
<td>1.4</td>
</tr>
<tr>
<td>love</td>
<td>3.1</td>
</tr>
<tr>
<td>loved</td>
<td>1.2</td>
</tr>
<tr>
<td>genius</td>
<td>0.5</td>
</tr>
<tr>
<td>not</td>
<td>-3.0</td>
</tr>
<tr>
<td>fruit</td>
<td>-0.8</td>
</tr>
<tr>
<td>BIAS</td>
<td>-0.1</td>
</tr>
</tbody>
</table>
As a discriminative classifier, logistic regression doesn’t assume features are independent like Naive Bayes does.

Its power partly comes in the ability to create richly expressive features with out the burden of independence.

We can represent text through features that are not just the identities of individual words, but any feature that is scoped over the entirety of the input.

<table>
<thead>
<tr>
<th>features</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains like</td>
</tr>
<tr>
<td>has word that shows up in positive sentiment dictionary</td>
</tr>
<tr>
<td>review begins with “I like”</td>
</tr>
<tr>
<td>at least 5 mentions of positive affectual verbs (like, love, etc.)</td>
</tr>
</tbody>
</table>
Stochastic g.d.

• Batch gradient descent reasons over every training data point for each update of $\beta$. This can be slow to converge.

• Stochastic gradient descent updates $\beta$ after each data point.

Algorithm 2 Logistic regression stochastic gradient descent

1: Data: training data $x \in \mathbb{R}^F, y \in \{0, 1\}$
2: $\beta = 0^F$
3: while not converged do
4: for $i = 1$ to $N$ do
5: $\beta_{t+1} = \beta_t + \alpha (y_i - \hat{p}(x_i)) x_i$
6: end for
7: end while
L2 regularization

\[ \ell(\beta) = \sum_{i=1}^{N} \log P(y_i \mid x_i, \beta) \quad - \quad \eta \sum_{j=1}^{F} \beta_j^2 \]

- We can do this by changing the function we’re trying to optimize by adding a penalty for having values of \( \beta \) that are high.

- This is equivalent to saying that each \( \beta \) element is drawn from a Normal distribution centered on 0.

- \( \eta \) controls how much of a penalty to pay for coefficients that are far from 0 (optimize on development data).
we can express $y$ as a function only of the input $x$ and the weights $W$ and $V$
Backpropagation: Given training samples of \(<x,y>\) pairs, we can use stochastic gradient descent to find the values of \(W\) and \(V\) that minimize the loss.
Convolutional networks

\[ h_1 = f(l, \text{hated}, \text{it}) \]

\[ h_2 = f(\text{it}, l, \text{really}) \]

\[ h_3 = f(\text{really}, \text{hated}, \text{it}) \]

\[ h_1 = \sigma(x_1 W_1 + x_2 W_2 + x_3 W_3) \]

\[ h_2 = \sigma(x_3 W_1 + x_4 W_2 + x_5 W_3) \]

\[ h_3 = \sigma(x_5 W_1 + x_6 W_2 + x_7 W_3) \]
Language Model

• Language models provide us with a way to quantify the likelihood of sequence — i.e., plausible sentences.
To see great *Pompey* passe the streets of *Rome*:

And when you saw his Chariot but appeare,

Haue you not made an *Univerall* Shout,

That *Tyber* trembled vnderneath her bankes

To heare the replication of your sounds,

Made in her *Concaue* Shores?

- to fee great Pompey paffe the Areets of Rome:
- to see great Pompey passe the streets of Rome:
Information theoretic view

“One morning I shot an elephant in my pajamas”

\[ Y \]

\[ \text{encode}(Y) \quad \text{decode}(\text{encode}(Y)) \]

Shannon 1948
Noisy Channel

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASR</td>
<td>speech signal</td>
<td>transcription</td>
</tr>
<tr>
<td>MT</td>
<td>target text</td>
<td>source text</td>
</tr>
<tr>
<td>OCR</td>
<td>pixel densities</td>
<td>transcription</td>
</tr>
</tbody>
</table>

\[ P(Y \mid X) \propto \underbrace{P(X \mid Y)}_{\text{channel model}} \underbrace{P(Y)}_{\text{source model}} \]
To see great Pompey passe the streets of Rome:
And when you saw his Chariot but appeare,
Have you not made an Universal shout,
That Tyber trembled underneath her bankes
To heare the replication of your sounds,
Made in her Concaue Shores?

\[ P(Y \mid X) \propto P(X \mid Y) \quad P(Y) \]

channel model  source model
The streets of Rome

\[ P(Y \mid X) \propto P(X \mid Y) \{ \text{channel model} \} P(Y) \{ \text{source model} \} \]
Language Model

• Language modeling is the task of estimating $P(w)$

• Why is this hard?

$P(“It was the best of times, it was the worst of times”)}$
Markov assumption

**bigram model** (first-order markov)

\[
\prod_{i=1}^{n} P(w_i \mid w_{i-1}) \times P(\text{STOP} \mid w_n)
\]

**trigram model** (second-order markov)

\[
\prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})
\times P(\text{STOP} \mid w_{n-1}, w_n)
\]
Smoothing LM

- Additive smoothing; Laplace smoothing
- Interpolating LMs of different orders
- Kneser-Ney
- Stupid backoff
Featurized LMs

- We can use multi class logistic regression for language modeling by treating the vocabulary as the output space

\[ Y = \nu \]
Featurized LMs

\[ P(w_i = dog \mid w_{i-2} = and, w_{i-1} = the) \]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_{i-2}=\text{the} \land w_{i-1}=\text{the} )</td>
<td>0</td>
</tr>
<tr>
<td>( w_{i-2}=\text{and} \land w_{i-1}=\text{the} )</td>
<td>1</td>
</tr>
<tr>
<td>( w_{i-2}=\text{bravest} \land w_{i-1}=\text{the} )</td>
<td>0</td>
</tr>
<tr>
<td>( w_{i-2}=\text{love} \land w_{i-1}=\text{the} )</td>
<td>0</td>
</tr>
<tr>
<td>( w_{i-1}=\text{the} )</td>
<td>1</td>
</tr>
<tr>
<td>( w_{i-1}=\text{and} )</td>
<td>0</td>
</tr>
<tr>
<td>( w_{i-1}=\text{bravest} )</td>
<td>0</td>
</tr>
<tr>
<td>( w_{i-1}=\text{love} )</td>
<td>0</td>
</tr>
<tr>
<td>\text{BIAS}</td>
<td>1</td>
</tr>
</tbody>
</table>
Richer representations

• Log-linear models give us the flexibility of encoding richer representations of the context we are conditioning on.

• We can reason about any observations from the entire history and not just the local context.
Recurrent neural network
Recurrent neural network

Each time step has two inputs:

- $x_i$ (the observation at time step $i$); one-hot vector, feature vector or distributed representation.
- $s_{i-1}$ (the output of the previous state); base case: $s_0 = 0$ vector.
Distributed representations

• Low-dimensional, dense word representations are extraordinarily powerful (and are arguably responsible for much of gains that neural network models have in NLP).

• Lets your representation of the input share statistical strength with words that behave similarly in terms of their distributional properties (often synonyms or words that belong to the same class).
Dense vectors from prediction

- Learning low-dimensional representations of words by framing a predicting task: using context to predict words in a surrounding window.

- Transform this into a **supervised** prediction problem; similar to language modeling but we’re ignoring order within the context window.
Using dense vectors

• In neural models (CNNs, RNNs, LM), replace the V-dimensional sparse vector with the much smaller K-dimensional dense one.

• Can also take the derivative of the loss function with respect to those representations to optimize for a particular task.
Subword models

• Rather than learning a single representation for each word type \( w \), learn representations \( z \) for the set of ngrams \( \mathcal{G}_w \) that comprise it [Bojanowski et al. 2017]

• The word itself is included among the ngrams (no matter its length).

• A word representation is the sum of those ngrams

\[
w = \sum_{g \in \mathcal{G}_w} z_g
\]
FastText

\[ e(\text{where}) = \]

\[ e(<\text{wh}) + e(\text{whe}) + e(\text{her}) + e(\text{ere}) + e(\text{re}>) \]

\[ + e(<\text{whe}) + e(\text{wher}) + e(\text{here}) + e(\text{ere}>) \]

\[ + e(<\text{wher}) + e(\text{where}) + e(\text{here}>) \]

\[ + e(<\text{where}) + e(\text{where}>) + e(<\text{where}>) \]

\[ + e(*) = \text{embedding for } * \]

3-grams

4-grams

5-grams

6-grams

word
Subword models need less data to get comparable performance.
ELMo

Stacked BiRNN trained to predict next word in language modeling task

BERT

Transformer-based model to predict masked word using bidirectional context + next sentence prediction.

Peters et al. 2018

Devlin et al. 2019
Attention

• Let’s incorporate structure (and parameters) into a network that captures which elements in the input we should be attending to (and which we can ignore).
\( v \in \mathbb{R}^H \)

Define \( v \) to be a vector to be learned; think of it as an “important word” vector. The dot product here measures how similar each input vector is to that “important word” vector.
I loved the movie!

\[ \mathbf{v} \in \mathcal{R}^H \]

\[ r_1 = \mathbf{v}^\top \mathbf{x}_1 \]
\[ r_2 = \mathbf{v}^\top \mathbf{x}_2 \]
\[ r_3 = \mathbf{v}^\top \mathbf{x}_3 \]
\[ r_4 = \mathbf{v}^\top \mathbf{x}_4 \]
\[ r_5 = \mathbf{v}^\top \mathbf{x}_5 \]
Convert \( r \) into a vector of normalized weights that sum to 1.

\[
a = \text{softmax}(r)
\]

\[
\begin{array}{c|c|c|c|c}
\hline
a & 0 & 0.64 & 0.02 & 0.02 & 0.32 \\
\hline
r & -3.4 & 2.4 & -0.8 & -1.2 & 1.7 \\
\hline
\end{array}
\]

\[
\begin{align*}
  r_1 &= v^\top x_1 \\
  r_2 &= v^\top x_2 \\
  r_3 &= v^\top x_3 \\
  r_4 &= v^\top x_4 \\
  r_5 &= v^\top x_5 \\
\end{align*}
\]
I loved the movie!

\[ y = \sum_{i=1}^{5} x_i a_i \]

\begin{array}{cccccc}
2.7 & 3.1 & -1.4 & -2.3 & 0.7 \\
-0.7 & -0.8 & -1.3 & -0.2 & -0.9 \\
2.3 & 1.5 & 1.1 & 1.4 & 1.3 \\
-0.9 & -1.5 & -0.7 & 0.9 & 0.2 \\
-0.1 & -0.7 & -1.6 & 0.2 & 0.6 \\
\end{array}
Attention

• Lots of variations on attention:
  • Linear transformation of x into before dotting with v
  • Non-linearities after each operation.
  • “Multi-head attention”: multiple v vectors to capture different phenomena that can be attended to in the input.
  • Hierarchical attention (sentence representation with attention over words + document representation with attention over sentences).
Parts of speech

- Parts of speech are categories of words defined distributionally by the morphological and syntactic contexts a word appears in.

<table>
<thead>
<tr>
<th></th>
<th>-s</th>
<th>-ed</th>
<th>-ing</th>
</tr>
</thead>
<tbody>
<tr>
<td>walk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slice</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>believe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>of</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>red</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Kim saw the elephant before we did dog, idea, *of, *goes.
<table>
<thead>
<tr>
<th>Nouns</th>
<th>fax, affluenza, subtweet, bitcoin, cronut, emoji, listicle, mocktail, selfie, skort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verbs</td>
<td>text, chillax, manspreading, photobomb, unfollow, google</td>
</tr>
<tr>
<td>Adjectives</td>
<td>crunk, amazeballs, post-truth, woke</td>
</tr>
<tr>
<td>Adverbs</td>
<td>hella, wicked</td>
</tr>
<tr>
<td>Determiner</td>
<td></td>
</tr>
<tr>
<td>Pronouns</td>
<td></td>
</tr>
<tr>
<td>Prepositions</td>
<td>English has a new preposition, because internet [Garber 2013; Pullum 2014]</td>
</tr>
<tr>
<td>Conjunctions</td>
<td></td>
</tr>
</tbody>
</table>
POS tagging

Labeling the tag that’s correct for the context.

Fruit *flies* like a banana

Time *flies* like an arrow

(Just tags in evidence within the Penn Treebank — more are possible!)
Sequence labeling

For a set of inputs $x$ with $n$ sequential time steps, one corresponding label $y_i$ for each $x_i$

$\mathbf{x} = \{x_1, \ldots, x_n\}$

$\mathbf{y} = \{y_1, \ldots, y_n\}$

- For a set of inputs $x$ with $n$ sequential time steps, one corresponding label $y_i$ for each $x_i$

- Model correlations in the labels $y$. 
HMM

\[ P(x_1, \ldots, x_n, y_1, \ldots, y_n) \approx \prod_{i=1}^{n+1} P(y_i \mid y_{i-1}) \prod_{i=1}^n P(x_i \mid y_i) \]
Hidden Markov Model

Prior probability of label sequence

\[ P(y) = P(y_1, \ldots, y_n) \]

\[ P(y_1, \ldots, y_n) \approx \prod_{i=1}^{n+1} P(y_i | y_{i-1}) \]

- We’ll make a first-order Markov assumption and calculate the joint probability as the product the individual factors conditioned only on the previous tag.
Hidden Markov Model

\[ P(x \mid y) = P(x_1, \ldots, x_n \mid y_1, \ldots, y_n) \]

\[ P(x_1, \ldots, x_n \mid y_1, \ldots, y_n) \approx \prod_{i=1}^{N} P(x_i \mid y_i) \]

• Here again we’ll make a strong assumption: the probability of the word we see at a given time step is only dependent on its label
Parameter estimation

\[ P(y_t \mid y_{t-1}) \]

\[ \frac{c(y_1, y_2)}{c(y_1)} \]

MLE for both is just counting
(as in Naive Bayes)

\[ P(x_t \mid y_t) \]

\[ \frac{c(x, y)}{c(y)} \]
Decoding

• Greedy: proceed left to right, committing to the best tag for each time step (given the sequence seen so far)

Fruit   flies   like   a   banana
NN      VB      IN      DT      NN
function VITERBI(observations of lcn T, state-graph of lcn N) returns best-path

create a path probability matrix viterbi[N+2,T]

for each state s from 1 to N do ; initialization step
    viterbi[s,1] ← a_{0,s} * b_s(o_1)
    backpointer[s,1] ← 0

for each time step t from 2 to T do ; recursion step
    for each state s from 1 to N do
        viterbi[s,t] ← \max_{s'=1}^N viterbi[s',t-1] * a_{s',s} * b_s(o_t)

        backpointer[s,t] ← \arg\max_{s'=1}^N viterbi[s',t-1] * a_{s',s}

viterbi[q_F,T] ← \max_{s=1}^N viterbi[s,T] * a_{s,q_F} ; termination step

backpointer[q_F,T] ← \arg\max_{s=1}^N viterbi[s,T] * a_{s,q_F} ; termination step

return the backtrace path by following backpointers to states back in time from backpointer[q_F,T]

Figure 10.8 Viterbi algorithm for finding optimal sequence of tags. Given an observation sequence and an HMM \( \lambda = (A,B) \), the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence. Note that states 0 and \( q_F \) are non-emitting.
MEMM

General maxent form

Maxent with first-order Markov assumption: Maximum Entropy Markov Model

$$\arg \max_y P(y \mid x, \beta)$$

$$\arg \max_y \prod_{i=1}^{n} P(y_i \mid y_{i-1}, x)$$
Features

\[ f(t_i, t_{i-1}; x_1, \ldots, x_n) \]

Features are scoped over the previous predicted tag and the entire observed input.

<table>
<thead>
<tr>
<th>feature</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_i = \text{man} )</td>
<td>1</td>
</tr>
<tr>
<td>( t_{i-1} = \text{JJ} )</td>
<td>1</td>
</tr>
<tr>
<td>( i=n ) (last word of sentence)</td>
<td>1</td>
</tr>
<tr>
<td>( x_i ) ends in -ly</td>
<td>0</td>
</tr>
</tbody>
</table>
Viterbi decoding

Viterbi for HMM: max joint probability

\[
P(y)P(x \mid y) = P(x, y)
\]

\[
v_t(y) = \max_{u \in \mathcal{Y}} [v_{t-1}(u) \times P(y_t = y \mid y_{t-1} = u)P(x_t \mid y_t = y)]
\]

Viterbi for MEMM: max conditional probability

\[
P(y \mid x)
\]

\[
v_t(y) = \max_{u \in \mathcal{Y}} [v_{t-1}(u) \times P(y_t = y \mid y_{t-1} = u, x, \beta)]
\]
MEMM Training

\[ \prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta) \]

Locally normalized — at each time step, each conditional distribution sums to 1
Label bias

Because of this local normalization, $P(\text{TO} \mid \text{context})$ will always be 1 if $x=\text{“to”}$

\[
\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)
\]
Label bias

That means our prediction for to can’t help us disambiguate will. We lose the information that MB + TO sequences rarely happen.
Conditional random fields

- We can solve this problem using global normalization (over the entire sequences) rather than locally normalized factors.

\[
P(y \mid x, \beta) = \prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)
\]

\[
P(y \mid x, \beta) = \frac{\exp(\Phi(x, y) \top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x, y') \top \beta)}
\]
Recurrent neural network

- For POS tagging, predict the tag from $y$ conditioned on the context

```
The DT R, O s0
  ↓
  dog NN R, O s1
  ↓
  ran VBD R, O s2
  ↓
  into IN R, O s3
  ↓
  town NN R, O s4
  ↓
```

```
s5
```
Bidirectional RNN

• A powerful alternative is make predictions conditioning both on the past and the future.

• Two RNNs
  • One running left-to-right
  • One right-to-left

• Each produces an output vector at each time step, which we concatenate
• BERT can be used not only as a language model to generate contextualized word representations, but also as a predictive model whose parameters are fine-tuned to a task.
The dog bark #ed
# Sequence labeling models

<table>
<thead>
<tr>
<th>model</th>
<th>form</th>
<th>label dependency</th>
<th>rich features?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden Markov Models</td>
<td>$\prod_{i=1}^{N} P(x_i</td>
<td>y_i) \ P(y_i</td>
<td>y_{i-1})$</td>
</tr>
<tr>
<td>MEMM</td>
<td>$\prod_{i=1}^{N} P(y_i</td>
<td>y_{i-1}, x, \beta)$</td>
<td>Markov assumption</td>
</tr>
<tr>
<td>CRF</td>
<td>$P(y</td>
<td>x, \beta)$</td>
<td>pairwise through entire sequence</td>
</tr>
<tr>
<td>RNN</td>
<td>$\prod_{i=1}^{N} P(y_i</td>
<td>x_{1:i}, \beta)$</td>
<td>none</td>
</tr>
</tbody>
</table>
Evaluation

• A critical part of development new algorithms and methods and demonstrating that they work
## Experiment design

<table>
<thead>
<tr>
<th></th>
<th>training</th>
<th>development</th>
<th>testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>80%</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>purpose</td>
<td>training models</td>
<td>model selection</td>
<td>evaluation; never look at it until the very end</td>
</tr>
</tbody>
</table>

- **Training models**: for training the models.
- **Model selection**: for selecting the best model.
- **Evaluation**: for evaluating the final model.
Accuracy

\[
\frac{1}{N} \sum_{i=1}^{N} I[\hat{y}_i = y_i]
\]

\[
I[x] \begin{cases} 
1 & \text{if } x \text{ is true} \\
0 & \text{otherwise}
\end{cases}
\]

<table>
<thead>
<tr>
<th></th>
<th>NN</th>
<th>VBZ</th>
<th>JJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted ((\hat{y}))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td>100</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>VBZ</td>
<td>0</td>
<td>104</td>
<td>30</td>
</tr>
<tr>
<td>JJ</td>
<td>30</td>
<td>40</td>
<td>70</td>
</tr>
</tbody>
</table>
Precision

Precision(\(NN\)) =

\[
\frac{\sum_{i=1}^{N} I(y_i = \hat{y}_i = \text{NN})}{\sum_{i=1}^{N} I(\hat{y}_i = \text{NN})}
\]

**Precison**: proportion of predicted class that are actually that class.

<table>
<thead>
<tr>
<th>True ((y))</th>
<th>Predicted ((\hat{y}))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NN</strong></td>
<td>100 2 15</td>
</tr>
<tr>
<td><strong>VBZ</strong></td>
<td>0 104 30</td>
</tr>
<tr>
<td><strong>JJ</strong></td>
<td>30 40 70</td>
</tr>
</tbody>
</table>
Recall

Recall(NN) = \[
\frac{\sum_{i=1}^{N} I(y_i = \hat{y}_i = \text{NN})}{\sum_{i=1}^{N} I(y_i = \text{NN})}
\]

Recall: proportion of true class that are predicted to be that class.

Predicted (\(\hat{y}\))

<table>
<thead>
<tr>
<th>True ((y))</th>
<th>NN</th>
<th>VBZ</th>
<th>JJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN</td>
<td>100</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>VBZ</td>
<td>0</td>
<td>104</td>
<td>30</td>
</tr>
<tr>
<td>JJ</td>
<td>30</td>
<td>40</td>
<td>70</td>
</tr>
</tbody>
</table>
F score

\[ F = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \]
A CFG gives a formal way to define what meaningful constituents are and exactly how a constituent is formed out of other constituents (or words). It defines valid structure in a language.

\[
NP \rightarrow \text{Det Nominal} \\
\text{Det} \rightarrow \text{the} \\
\text{Nominal} \rightarrow \text{Noun} \\
\text{Noun} \rightarrow \text{flight}
\]

\[
NP \rightarrow \text{Verb Nominal} \\
\text{Verb} \rightarrow \text{runs} \\
\text{Nominal} \rightarrow \text{Noun} \\
\text{Noun} \rightarrow \text{flight}
\]
Constituents

Every internal node is a phrase

- my pajamas
- in my pajamas
- elephant in my pajamas
- an elephant in my pajamas
- shot an elephant in my pajamas
- I shot an elephant in my pajamas

Each phrase could be replaced by another of the same type of constituent
Parseval (1991):
Represent each tree as a collection of tuples:

\(<l_1, i_1, j_1>, \ldots, <l_n, i_n, j_n>\)

- \(l_k\) = label for \(k\)th phrase
- \(i_k\) = index for first word in \(z\)th phrase
- \(j_k\) = index for last word in \(k\)th phrase

Smith 2017
I shot an elephant in my pajamas.

- <S, 1, 7>
- <NP, 1, 1>
- <VP, 2, 7>
- <VP, 2, 4>
- <NP, 3, 4>
- <Nominal, 4, 4>
- <PP, 5, 7>
- <NP, 6, 7>
Evaluation

I_{1} shot_{2} an_{3} elephant_{4} in_{5} my_{6} pajamas_{7}
Evaluation

Calculate precision, recall, F1 from these collections of tuples

• Precision: number of tuples in predicted tree also in gold standard tree, divided by number of tuples in predicted tree

• Recall: number of tuples in predicted tree also in gold standard tree, divided by number of tuples in gold standard tree
Treebanks

• Rather than create the rules by hand, we can annotate sentences with their syntactic structure and then extract the rules from the annotations

• Treebanks: collections of sentences annotated with syntactic structure
Example rules extracted from this single annotation

<table>
<thead>
<tr>
<th>Rule</th>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>NNP NNP</td>
</tr>
<tr>
<td>NP-SBJ</td>
<td>NP , ADJP ,</td>
</tr>
<tr>
<td>S</td>
<td>NP-SBJ VP</td>
</tr>
<tr>
<td>VP</td>
<td>VB NP PP-CLR NP-TMP</td>
</tr>
</tbody>
</table>

Pierre Virkam will join the board as a non-executive director.
PCFG

• Probabilistic context-free grammar: each production is also associated with a probability.

• This lets us calculate the probability of a parse for a given sentence; for a given parse tree $T$ for sentence $S$ comprised of $n$ rules from $R$ (each $A \rightarrow \beta$):

$$P(T, S) = \prod_{i}^{n} P(\beta \mid A)$$
Estimating PCFGs

\[ \sum_{\beta} P(\beta \mid A) = \frac{C(A \to \beta)}{\sum_{\gamma} C(A \to \gamma)} \]

(equivalently)

\[ \sum_{\beta} P(\beta \mid A) = \frac{C(A \to \beta)}{C(A)} \]
Does any rule generate PRP VBD?
I shot an elephant in my pajamas

Does any rule generate VBD DT?
<table>
<thead>
<tr>
<th></th>
<th>shot</th>
<th>an</th>
<th>elephant</th>
<th>in</th>
<th>my</th>
<th>pajamas</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NP, PRP</strong></td>
<td>ø</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>VBD</strong></td>
<td>ø</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NP, NN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PRP$</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NNS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two possible places look for that split k
I shot an elephant in my pajamas

Two possible places look for that split k
I shot an elephant in my pajamas.

Two possible places look for that split k
Does any rule generate DT NN?
I shot an elephant in my pajamas.

Two possible places look for that split k.
I shot an elephant in my pajamas.

Two possible places look for that split k.
I shot an elephant in my pajamas

Three possible places look for that split k
I shot an elephant in my pajamas

Three possible places look for that split k
<table>
<thead>
<tr>
<th></th>
<th>shot</th>
<th>an</th>
<th>elephant</th>
<th>in</th>
<th>my</th>
<th>pajamas</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBD</td>
<td></td>
<td></td>
<td></td>
<td>VP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DT</td>
<td></td>
<td></td>
<td></td>
<td>NP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP, NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td></td>
<td></td>
<td>PRP$</td>
<td></td>
<td>NNS</td>
</tr>
</tbody>
</table>

Three possible places look for that split k
I shot an elephant in my pajamas

Three possible places look for that split k
I shot an elephant in my pajamas.
I shot an elephant in my pajamas.

* I shot an elephant in my
* Shot an elephant in my
* Elephant in my
* An elephant in my

* Elephant in
* An elephant in
* Shot an elephant in
* I shot an elephant in

*In
* In my
* An elephant in my
* Shot an elephant in my
* I shot an elephant in my

PRP $ [5,6]
I shot an elephant in my pajamas.
I shot an elephant in my pajamas.
I shot an elephant in my pajamas
I shot an elephant in my pajamas.
I shot an elephant in my pajamas.
I shot an elephant in my pajamas.
I shot an elephant in my pajamas.
I shot an elephant in my pajamas.
I shot an elephant in my pajamas.
I shot an elephant in my pajamas.
I shot an elephant in my pajamas.
Possibilities:

- $S_1 \rightarrow NP \ VP_1$
- $S_2 \rightarrow NP \ VP_2$
- $\rightarrow S \ PP$
- $\rightarrow PRP \ VP_1$
- $\rightarrow PRP \ VP_2$
Success! We’ve recognized a total of two valid parses

<table>
<thead>
<tr>
<th></th>
<th>shot</th>
<th>an</th>
<th>elephant</th>
<th>in</th>
<th>my</th>
<th>pajamas</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP, PRP [0,1]</td>
<td>Ø</td>
<td>Ø</td>
<td>S [0,4]</td>
<td>Ø</td>
<td>Ø</td>
<td>S₁, S₂ [0,7]</td>
</tr>
<tr>
<td>VBD [1,2]</td>
<td>Ø</td>
<td>VP [1,4]</td>
<td>Ø</td>
<td>Ø</td>
<td>VP₁, VP₂ [1,7]</td>
<td></td>
</tr>
<tr>
<td>NP, NN [3,4]</td>
<td>Ø</td>
<td>Ø</td>
<td>NP [3,7]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN [4,5]</td>
<td>Ø</td>
<td>PP [4,7]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRP$ [5,6]</td>
<td>NP [5,7]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NNS [6,7]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PCFGs

- A PCFG gives us a mechanism for assigning scores (here, probabilities) to different parses for the same sentence.

- But we often care about is finding the single best parse with the highest probability.

- We calculate the max probability parse using CKY by storing the probability of each phrase within each cell as we build it up.
As in Viterbi, backpointers let us keep track on the path through the chart that leads to the best derivation.
Midterm

• Take-home exam — you’ll have 80 continuous minutes to take the midterm anytime within the 24-hour period (3:30pm 3/12—2:59pm 3/13).

• Mix of multiple choice, short answer, long answer

• Covers all material from lectures and readings