CS 4650/7650, Lecture 10
Hidden Markov Models and Structured Perceptron

Jacob Eisenstein

September 17, 2013

1 Review of Part of Speech

Bender defines parts-of-speech as capturing "the morphological and syntactic contexts in which words appear."

- There are four major POS classes:
  - **Nouns**: colloquially these refer to entities, although such "metaphysical" definitions have been found to be unhelpful by linguists. Nouns can be defined functionally as words that typically appear as arguments.
  - **Verbs**: colloquially these refer to actions, but they can be defined functionally as words whose primary use is as predicates.
  - **Adjectives** modify nouns
  - **Adverbs** modify verbs and also clauses
- In addition, we have a number of closed-class tags, including **determiners**, **prepositions**, **conjunctions**, **particles**, etc.
- We can also specialize POS classes. For example, the Penn Treebank differentiates
  - Pronouns, possessive pronouns, proper nouns, and plural nouns

  - Modal verbs; past tense verbs, bare verbs, participles, and 3rd-person singular
  - Comparative and superlative adjectives and adverbs
- The Twitter POS set that you will work with in Project 2 is more coarse-grained; the Brown POS set is more fine-grained.

The number of possible POS sequences for a sentence grows exponentially in the length of the sentence.

We have three basic types of features that can help us identify a word's part-of-speech.

- The word itself. 60% of English word types are unambiguous.
- The context. This can mean many things.
  - Local: determiners precede nouns and adjectives; adjectives rarely precede verbs.
  - Global: tense should usually (but not always) agree. Conjunctions tend to have the same number, e.g. *the blue apples and the red oranges*.
- The word morphology. *-ing* is a strong indicator of VBG and against VBD, for example.

2 Hidden Markov Models

HMMs treat part-of-speech tagging probabilistically. Specifically, we want to maximize $P(\boldsymbol{y}|\boldsymbol{x}) \propto P(\boldsymbol{y}, \boldsymbol{x})$, where $\boldsymbol{x}$ are words and $\boldsymbol{y}$ are tags.

We need to define the probability distribution, $P(\boldsymbol{x}, \boldsymbol{y})$, which we do through a *generative story*,

- For word n, draw tag $y_n \sim \text{Categorical}(\theta_{y_{n-1}})$
- Then draw word $x_n \sim \text{Categorical}(\phi_{y_n})$

Under this model, we can compute

$$P(\boldsymbol{y}|\boldsymbol{x}) \propto P(\boldsymbol{x}, \boldsymbol{y}) \tag{1}$$
$$P(\boldsymbol{x}|\boldsymbol{y}; \phi) P(\boldsymbol{x}; \theta) \tag{2}$$
$$\prod_{n}^{N} P(x_n|y_n; \phi) P(y_n|y_{n-1}; \theta) \tag{3}$$

- 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 describe this generative story as a graphical model. Note that although graphical models also use circles and arrows, the meaning is completely different from finite-state models.
- Our generative story assumes that the words are conditionally independent, given the tags, so

$$x_n \perp \{\boldsymbol{x}_{m \neq n}\} \mid y_n.$$

- Note that conditional independence is not the same as independence. We do **not** have $P(x_n, x_m) = P(x_n)P(x_m)$ 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

$$\begin{aligned}
P(x_n = \textit{the}, x_{n+1} = \textit{bike}) &= \sum_{y_{n+1}, y_n} P(x_n = \textit{the}, x_{n+1} = \textit{bike}, y_{n+1}, y_n) \\
&= \sum_{y_{n+1}, y_n} P(x_{n+1} = \textit{bike}|y_{n+1}, y_n, x_n = \textit{the}) \\
&\quad \times P(y_{n+1}|y_n, x_n = \textit{the}) P(y_n|x_n = \textit{the}) P(x_n = \textit{the}) \\
&= \sum_{y_{n+1}} P(x_{n+1} = \textit{bike}|y_{n+1}) \\
&\quad \times \sum_{y_n} P(y_{n+1}|y_n) P(y_n|x_n = \textit{the}) P(x_n = \textit{the}) \\
&= P(x_{n+1} = \textit{bike}|y_{n+1} = \textsc{Noun}) \times 1 \times 1 \times P(x_n = \textit{the}) \\
&> P(x_{n+1} = \textit{bike}) P(x_n = \textit{the})
\end{aligned}$$

- 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 $P_T(\text{VBZ}, \text{VBP})$ 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)
- We need **joint inference** over $\boldsymbol{y}_{1:N}$ to find $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} P(\boldsymbol{x}, \boldsymbol{y})$
- But the number of possible sequences in $\mathcal{T}^N$ is (of course) $\#|\mathcal{T}|^N$. How can we do joint inference efficiently?

3 Part-of-speech tagging as a finite-state transducer

- Transducer E has one state, and transduces from tags to words, with $\delta_{w,t} = \log P(x|y)$.
- Transducer T has $\#|\mathcal{T}|$ states (if it's a bigram model), and transduces tags to tags, with $\delta_{y_i, y_{i-1}} = \log P(t_n|t_{n-1})$.
- Recall the definition of finite state composition,

$$(T \circ E)(y, x) = \bigoplus_z T(y, z) \otimes E(z, x). \tag{4}$$

  Since T only accepts identical tag pairs $\langle y, y \rangle$, we can ignore $\bigoplus$. The result of $T \circ E$ transduces tags to words, with edge weights

$$\begin{aligned} \delta_{x, y_n, y_{n-1}} &= \log P(x|y_n) \otimes \log P(y_n|y_{n-1}) \\ &= \log P(x|y_n) + \log P(y_n|y_{n-1}) \\ &= \log P(x, y_n|y_{n-1}) \end{aligned}$$

  Can you see how many states it will have?

- Finally, we compose with an acceptor which forms a chain for a sentence $x_1, \ldots x_n$.
- This composition $T \circ E \circ S$ yields a **trellis**-shaped weighted finite state accept (WFSA).
  - Number of columns = N, length of input.
  - Number of rows = T, number of tags.
  - Edges from states $\langle n, k_1 \rangle$ to $\langle n{+}1, k_2 \rangle$ with score $\log P_e(x_{n+1}|k_2)P_t(k_2|k_1)$.
  - Best path in the trellis = $\text{argmax}_{\boldsymbol{y}} \log P(\boldsymbol{x}_{1:N}, \boldsymbol{y}_{1:N})$. This means we want to define $\bigoplus = \max$.
- Is efficient inference possible?
  - How expensive is it to construct the trellis?
    - Generic composition is polynomial but slower than we'd like (depends on 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? $\mathcal{O}(NT)$ states, $\mathcal{O}(NT^2)$ edges.
  - How expensive is it find the shortest path in the trellis?: Generic shorest path has a time cost of $\mathcal{O}(V \log V{+}E) = \mathcal{O}(NT \log NT{+}NT^2)$ and a space cost of $\mathcal{O}(V) = \mathcal{O}(NT^2)$.
  - 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.

4 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 $\mathcal{O}(NT^2)$ and a space cost of $\mathcal{O}(NT)$.

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

$$P(\boldsymbol{x}_{1:N}, \boldsymbol{y}_{1:N}) = P(x_N|y_N)P(y_N|y_{N-1})P(\boldsymbol{x}_{1:N-1}, \boldsymbol{y}_{1:N-1})$$

- Given y_{n-1}, we can choose y_n without considering any other element of the history.
- Suppose we know the best path to $y_n = k$.
  The best path to $y_{n+1} = k'$ *through* $y_n = k$ must include the best path to $y_n = k$.
- Suppose we know the score of the best path to each y_n, which we write $v_n(k)$.
  What is the score of the best path to $y_{n+1} = k$?

$$\begin{aligned}
v_{n+1}(k') &= \max_k v_n(k) \otimes \delta_{n,k,k'} \\
&= \max_k v_n(k) \otimes (\log P_E(x_n|y_n = k')) \otimes (\log P_T(y_n = k'|y_{n-1} = k)) \\
&= \log P_E(x_n|y_n) \otimes \left(\max_k v_n(k) \otimes \log P_T(y_n = k'|y_{n-1} = k)\right)
\end{aligned}$$

- The score of the best tag sequence overall is $\max_k v_N(k)$.
- To find the best tag sequence, we just need to keep back-pointers, from $v_n(k)$ to $v_{n-1}(k')$:

$$\begin{aligned}
v_{n+1}(k') &= \log P_E(x_n|y_n = k') \otimes \left(\max_k v_n(k) \otimes \log P_T(y_n = k'|y_{n-1} = k)\right) \\
b_{n+1}(k') &= \arg\max_k v_n(k) \otimes \log P_T(y_n = k'|y_{n-1} = k)
\end{aligned}$$

- We can convert to fully semiring notation by replacing max with $\oplus$,

$$v_{n+1}(k') = \log P_E(x_n|y_n = k') \otimes \left(\bigoplus_k v_n(k) \otimes \log P_T(y_n = k'|y_{n-1} = k)\right)$$

  We'll see why this is useful in a moment, but first let's do an example.

4.1 Example

- $\log P(w|t)$:

	they	*can*	*fish*
N	-1	-2	-2
V	-5	-1	-2

- $\log P(y_n|y_{n-1})$:

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

5 The forward algorithm

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

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

The Viterbi algorithm solves the decoding problem. We'll talk about the learning problem later. Let's now talk about the likelihood problem.

How can we compute the joint likelihood $P(\boldsymbol{x})$?

- Move to a semiring where $\oplus(a, b) = \log(e^a + e^b)$ instead of max.
- Claim: in this semiring, $v_n(k) = \log P(\boldsymbol{x}_{1:n}, y_n = k)$
- Let's work inductively
  - **base case**:

$$v_0(k) = \log P(x_o|y_0) \otimes \log P(y_0 = k|\textsc{Start}) = \log P(x_0, y_0 = k|\textsc{Start}) \tag{5}$$

– **inductive hypothesis**: $v_{n-1}(k) = \log P(\boldsymbol{x}_{1:n-1}, y_{n-1} = k)$.

$$
\begin{aligned}
v_n(k') &= \log \sum_k \exp(v_{n-1}(k)) + \exp\left(\log P_T(y_n|y_{n-1}=k) + \log P_e(x_n|y_n=k')\right) \\
&= \log \sum_k \exp(\log P(x_{1:n-1}, y_{n-1}=k) + \log Pr(y_n=k'|y_{n-1}=k) P(x_n|y_n=k')) \\
&= \log \sum_k P(x_{1:n-1}, y_{n-1}=k) P(x_n, y_n=k'|y_{n-1}=k') \\
&= \log \sum_k P(x_{1:n}, y_n=k', y_{n-1}=k) \\
&= \log P(x_{1:n}, y_n=k')
\end{aligned}
$$

This is called the **forward** algorithm.

5.1 Why solve the likelihood problem?

Word class language models

- Remember $P(\textit{colorless green ideas sleep furiously})$
- We don't care about the specific tags, we just want to know the probability of the utterance.

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 $P(y_n|x_{1:N})$. The likelihood $P(x_{1:N})$ is part of this computation.

5.2 Trigram HMMs

- 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, $\langle y_n, y_{n-1} \rangle$.
- 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.

6 Estimation

In principle, we can use relative frequency estimation.

$$\theta_{k,k'} \triangleq P(y_n = k' | y_{n-1} = k) = \frac{\text{count}(y_n = k', y_{n-1} = k)}{\text{count}(y_{n-1} = k)}$$

$$\phi_{k,i} \triangleq P(x_n = i | y_n = k) = \frac{\text{count}(x_n = i, y_n = k)}{\text{count}(y_n = k)}$$

In practice, we need smoothing and other tricks.

- The same smoothing ideas from language modeling can be applied.
  - backoff from trigrams to bigrams to unigrams

    $$P(t_n | t_{n-1}, t_{n-2}) = \lambda_2 \hat{P}_2(t_n | t_{n-1}, t_{n-2}) + \lambda_1 \hat{P}_1(t_n | t_{n-1}) + (1 - \lambda_2 - \lambda_1) \hat{P}_0(t_n)$$

  - reserve probability mass for unseen words
- Unseen words have internal structure which can be exploited: *mimsy*, *3.1415*, *Ke$ha*
  - More advanced approaches model morphological and orthographic features, creating a more complex $P_E(x|y)$ emission probability.
  - This isn't easy to do in an HMM: such features badly violate the conditional independence assumption.

    $$P(\textit{capable}, \textit{-able} | \text{JJ}) \neq P(\textit{capable} | \text{JJ}) P(\textit{-able} | \text{JJ})$$

- The TNT tagger, published in 2000, manages to accomplish this [Bra00], and is one of the best generative taggers.
- However, a better solution is to give up on the Naive Bayes model that is at the foundation of generative Hidden Markov Models, and move to a feature-based, discriminative approach.
- In 2001, Lafferty, Pereira, and McCallum introduced **conditional random fields**, which modeled $P(\boldsymbol{y}|\boldsymbol{x})$ directly as a log-linear model. We'll talk about that next week. For now, we'll talk about a simpler, but later approach called **structured perceptron**, which was introduced by Michael Collins [Col02] in 2002.

7 Tagging with features

In POS tagging we identified three types of features

- The word itself: (90% accurate). $P_E(x|y)$
- Context, $P_T(y_n|y_{n-1})$
- Orthography and morphology (*the mimsy toves*)
  - *-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:

8 Structured perceptron

Remember the perceptron update:

$$\hat{y} = \arg\max_y \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}, y) \tag{6}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + \boldsymbol{f}(\boldsymbol{x}, y^*) - \boldsymbol{f}(\boldsymbol{x}, \hat{y}) \tag{7}$$

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

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} \boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) \tag{8}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}^*) - \boldsymbol{f}(\boldsymbol{x}, \hat{\boldsymbol{y}}) \tag{9}$$

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 $\mathcal{Y}$. The set of permissible outputs depends on the input $\boldsymbol{x}$, and it is very large: $\mathcal{O}(K^N)$. What can we do?

8.1 Viterbi inference for structured perceptron

- Suppose we restrict the scoring function $\boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$...
- ... then we can apply the Viterbi algorithm to find $\arg\max_{\boldsymbol{y}} \boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$!
- What restriction is necessary?
  - We require $\boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) = \sum_n \boldsymbol{w}^\top \boldsymbol{f}(y_n, y_{n-1}, \boldsymbol{x}, n)$
  - 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.
- In Viterbi, we have

$$\begin{aligned} v_n(k) &= \max_{k'} \boldsymbol{w}^\top \boldsymbol{f}_n(\boldsymbol{x}, k, k') + v_{n-1}(k') \\ &= \bigoplus_{k'} \boldsymbol{w}^\top \boldsymbol{f}_n(\boldsymbol{x}, k, k') \otimes v_{n-1}(k') \end{aligned}$$

- Suppose $\boldsymbol{w}^\top \boldsymbol{f}_n(\boldsymbol{x}, y_n, y_{n-1}) = \log P(x_n|y_n) + \log P(y_n|y_{n-1})$.
  - We could attain this by having two features, $\langle x_n, y_n \rangle$ and $\langle y_n, y_{n-1} \rangle$.
  - We would then have to set their weights, $w_{\langle x_n, y_n \rangle} = \log P(x_n|y_n)$, and $w_{\langle y_n, y_{n-1} \rangle} = \log P(y_n|y_{n-1})$

- Then we exactly recover HMM Viterbi.
- But we could also learn these weights discriminatively, possibly achieving better performance.
- And the real advantage is that we can add more features, for capitalization, morphology, etc.

8.2 Example

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

- Word features: $\{\langle \textit{and}, \text{CC}\rangle = 1, \langle \textit{the}, \text{DT}\rangle = 1, \langle \textit{slithy}, \text{JJ}\rangle = 1, \langle \textit{toves}, \text{NNS}\rangle = 1\}$
  (but we have never seen $\langle \textit{slithy}, \text{X}\rangle$ or $\langle \textit{toves}, \text{NNS}\rangle$ before).
- Orthography features: $\{\langle \textit{-thy}, \text{JJ}\rangle = 1, \langle \textit{-es}, \text{NNS}\rangle = 1\}$
- Sequence features: $\{\langle CC, DT\rangle = 1, \langle DT, JJ\rangle = 1, \langle JJ, NNS\rangle = 1\}$

These features are typical, but others are permitted:

- $\{\langle \textit{and} - \text{PREV}, \text{DT}\rangle, \langle \textit{the} - \text{PREV}, \text{JJ}\rangle ...$
- We can have features that consider arbitrarily distant parts of the observations: $\langle x_{n-10}, y_n\rangle$.
- But we can only consider adjacent y_n.
- Typically, we want to implement a **feature function** of the form $\texttt{feat}(\boldsymbol{x}, y_n, y_{n-1}, n)$, which computes the features for assigning tag $y_n = k$ at position n in the input string $\boldsymbol{x}$.

9 Sequence labeling

Part-of-speech tagging is a specific instance of a general class of problems, called **sequence labeling**.

The output is a sequence of labels, which may depend on each other. We typically apply Viterbi to decode.

9.1 NER

Another classic sequence labeling problem is **named-entity recognition** (NER). We want to recognize strings of text that refer to a named entity, like *Miley Cyrus*, *iPhone*, *The President of the United States of America*, *The Georgia Institute of Technology*, and *e. e. cummings*.

- The situation is similar to POS tagging: we care about the strings themselves, as well as the context:
  - *I'm going to [PLACE] tomorrow*
  - *[PERSON] bought a controlling share in [COMPANY] for [DOLLAR AMOUNT]*
  - *In wake of the [*ORG *Komen] controversy, [*PER *Mayor Bloomberg] says he will personally give $250,000 to [*ORG *Planned Parenthood].*
- We can formulate this as a sequence labeling problem using **BIO** notation:
  - B: begin-entity
  - I: inside-entity
  - O: outside-entity
- This captures three insights
  1. external context is most important to the beginning of the entity
  2. the probability of remaining inside an entity is different than the probability of entering one
  3. the generative probability for some words (e.g., *of*) is really different in all three cases
- We can specialize to B-ORG, B-PERSON, etc.
- We can introduce additional tags: L: last-token-in-entity, and U: unique token in entity. This is **BILOU** notation.

9.2 Phrase chunking

Also known as *shallow parsing*, we want to collect groups of words into phrases.

*[*NP *The morning flight] [*PP *from] [*NP *Denver] [*VP *has arrived].*

We can use BIO tagging here too.

9.3 Dialogue act labeling

Here we want to classify utterances as STATEMENT, QUESTION, BACKCHANNEL, etc. Utterances are entire sentences, so the model must be a little different, but the same basic principles can apply

References

[Bra00] Thorsten Brants. Tnt – a statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, pages 224–231, Seattle, Washington, USA, April 2000. Association for Computational Linguistics.

[Col02] Michael Collins. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.