Part Of Speech Tagging

- The goal of sequence labeling is to assign tags to words, or more generally, to assign discrete labels to discrete elements in a sequence.
- There are many applications of sequence labeling in natural language processing, one of the important application is part-of-speech tagging
- In part-of-speech tagging, each word need to be tagged with grammatical category for example, noun, verb, adjective etc.

- Consider a simple input:

  They can fish

- A dictionary of coarse-grained part-of-speech tags might include NOUN as the only valid tag for they
- But both NOUN and VERB as potential tags for can and fish.
- An accurate sequence labeling algorithm should select the verb tag for both can and fish
- But it should select noun for the same two words in the phrase *can of fish*.

Sequence labeling as classification

- One way to solve a tagging problem is to turn it into a classification problem.
- Let f((w, m), y) indicate the feature function for tag y at position m in the sequence $w = (w_1, w_2, \ldots, w_M)$.
- A simple tagging model would have a single base feature, the word itself:

  f((w = they can fish, m = 1), N) =(they, N)

  f((w = they can fish, m = 2), V) =(can, V)

  f((w = they can fish, m = 3), V) =(fish, V)

- Here the feature function takes three arguments as input: the sentence to be tagged (e.g.,they can fish), the proposed tag (e.g., N or V), and the index of the token to which this tag is applied.
- This simple feature function then returns a single feature: a tuple including the word to be tagged and the tag that has been proposed.
- If the vocabulary size is V and the number of tags is K, then there are $V \times K$ features.
- Each of these features must be assigned a weight.
- These weights can be learned from a labeled dataset using a classification algorithm

- However, it is easy to see that this simple classification approach cannot correctly tag both *they can fish* and *can of fish*, because can and fish are grammatically ambiguous.
- To handle both of these cases, the tagger must rely on context, such as the surrounding words.
- We can build context into the feature set by incorporating the surrounding words as additional features:

$$
\begin{aligned}
\boldsymbol{f}((\boldsymbol{w} = \textit{they can fish}, 1), \mathrm{N}) = \{&(w_m = \textit{they}, y_m = \mathrm{N}),\\
&(w_{m-1} = \square, y_m = \mathrm{N}),\\
&(w_{m+1} = \textit{can}, y_m = \mathrm{N})\}\\
\boldsymbol{f}((\boldsymbol{w} = \textit{they can fish}, 2), \mathrm{V}) = \{&(w_m = \textit{can}, y_m = \mathrm{V}),\\
&(w_{m-1} = \textit{they}, y_m = \mathrm{V}),\\
&(w_{m+1} = \textit{fish}, y_m = \mathrm{V})\}\\
\boldsymbol{f}((\boldsymbol{w} = \textit{they can fish}, 3), \mathrm{V}) = \{&(w_m = \textit{fish}, y_m = \mathrm{V}),\\
&(w_{m-1} = \textit{can}, y_m = \mathrm{V}),\\
&(w_{m+1} = \blacksquare, y_m = \mathrm{V})\}.
\end{aligned}
$$

- These features contain enough information that a tagger should be able to choose the right tag for the word fish: words that come after can are likely to be verbs, so the feature (w_{m-1} = can, y_m = V) should have a large positive weight.
- However, even with this enhanced feature set, it may be difficult to tag some sequences correctly.
- One reason is that there are often relationships between the tags themselves.
- For example, in English it is relatively rare for a verb to follow another verb particularly if we differentiate MODAL verbs like can and should from more typical verbs like give, take, cry.
- We would like to incorporate preferences against tag sequences like VERB-VERB, and in favor of tag sequences like NOUN-VERB.

- The need for such preferences is best illustrated by a garden path sentence

  The old man the boat.

- Grammatically, the word *the* is a DETERMINER.
- When you read the sentence, what part of speech did you first assign to *old*? Typically, this word is an ADJECTIVE — abbreviated as J
- Similarly, *man* is usually a noun.
- The resulting sequence of tags is D J N D N.
- It is unlikely that a determiner would directly follow a noun, and it is particularly unlikely that the entire sentence would lack a verb.

- As an alternative, think of the entire sequence of tags as a label itself.
- For a given sequence of words $w = (w_1, w_2, \ldots, w_M)$, there is a set of possible taggings $Y(w) = Y^M$, where Y = {N, V, D, . . .} refers to the set of individual tags, and Y^M refers to the set of tag sequences of length M.
- We can then treat the sequence labeling problem as a classification problem in the label space $Y(w)$

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{w})}{\operatorname{argmax}} \Psi(\boldsymbol{w}, \boldsymbol{y}),$$

- where $y = (y_1, y_2, \ldots, y_M)$ is a sequence of M tags, and Ψ is a scoring function on pairs of sequences, $w^M \times Y^M \rightarrow R$.

- Such a function can include features that capture the relationships between tagging decisions,
  - such as the preference that determiners not follow nouns,
  - all sentences must have verbs.
- However, given that the label space is exponentially large in the length of the sequence M, it is never practical to perform tagging in this way
- In English, it is not unusual to have sentences of length M = 20; part-of-speech tag sets vary in size from 10 to several hundred.
- Taking the low end of this range, we have $\approx 10^{20}$, one hundred billion billion possible tag sequences.
- Enumerating and scoring each of these sequences would require an amount of work that is exponential in the sequence length, so inference is intractable.

- However, the situation changes when we restrict the scoring function.
- Suppose we choose a function that decomposes into a sum of local parts where each ψ(·) scores a local part of the tag sequence.

$$\Psi(\boldsymbol{w}, \boldsymbol{y}) = \sum_{m=1}^{M+1} \psi(\boldsymbol{w}, y_m, y_{m-1}, m),$$

- Note that the sum goes up to M+1, so that we can include a score for a special end-of-sequence tag, $\psi(w_{1:M}, \blacklozenge, y_M, M+1)$.
- We also define a special tag to begin the sequence, $y_0 \triangleq \lozenge$.

- In a linear model, local scoring function can be defined as a dot product of weights and features

$$\psi(\boldsymbol{w}_{1:M}, y_m, y_{m-1}, m) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{w}, y_m, y_{m-1}, m).$$

- The feature vector f can consider the entire input w, and can look at pairs of adjacent tags.
- The weights can assign low scores to infelicitous tag pairs, such as noun-determiner, and high scores for frequent tag pairs, such as determiner-noun and noun-verb.

- In the example *they can fish*, a minimal feature function would include features for word-tag pairs (sometimes called emission features) and tag-tag pairs (sometimes called transition features):

$$
\begin{aligned}
\boldsymbol{f}(\boldsymbol{w} = \textit{they can fish}, \boldsymbol{y} = \mathrm{N\ V\ V}) &= \sum_{m=1}^{M+1} \boldsymbol{f}(\boldsymbol{w}, y_m, y_{m-1}, m) \\
&= \boldsymbol{f}(\boldsymbol{w}, \mathrm{N}, \lozenge, 1) \\
&\quad + \boldsymbol{f}(\boldsymbol{w}, \mathrm{V}, \mathrm{N}, 2) \\
&\quad + \boldsymbol{f}(\boldsymbol{w}, \mathrm{V}, \mathrm{V}, 3) \\
&\quad + \boldsymbol{f}(\boldsymbol{w}, \blacklozenge, \mathrm{V}, 4) \\
&= (w_m = \textit{they}, y_m = \mathrm{N}) + (y_m = \mathrm{N}, y_{m-1} = \lozenge) \\
&\quad + (w_m = \textit{can}, y_m = \mathrm{V}) + (y_m = \mathrm{V}, y_{m-1} = \mathrm{N}) \\
&\quad + (w_m = \textit{fish}, y_m = \mathrm{V}) + (y_m = \mathrm{V}, y_{m-1} = \mathrm{V}) \\
&\quad + (y_m = \blacklozenge, y_{m-1} = \mathrm{V}).
\end{aligned}
$$

- There are seven active features for this example: one for each word-tag pair, and one for each tag-tag pair, including a final tag $y_{M+1} = \blacklozenge$.
- These features capture the two main sources of information for part-of-speech tagging in English: which tags are appropriate for each word, and which tags tend to follow each other in sequence.
- Given appropriate weights for these features, taggers can achieve high accuracy, even for difficult cases like *the old man the boat*.

The Viterbi algorithm

- By decomposing the scoring function into a sum of local parts, it is possible to rewrite the tagging problem as follows:

$$\begin{aligned}\hat{\boldsymbol{y}} &= \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{w})} \Psi(\boldsymbol{w}, \boldsymbol{y}) \\ &= \operatorname*{argmax}_{\boldsymbol{y}_{1:M}} \sum_{m=1}^{M+1} \psi(\boldsymbol{w}, y_m, y_{m-1}, m) \\ &= \operatorname*{argmax}_{\boldsymbol{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}),\end{aligned}$$

where the final line simplifies the notation with the shorthand,

$$s_m(y_m, y_{m-1}) \triangleq \psi(\boldsymbol{w}_{1:M}, y_m, y_{m-1}, m).$$

- This inference problem can be solved efficiently using dynamic programming
- We begin by solving an auxiliary problem: rather than finding the best tag sequence, we compute the score of the best tag sequence

$$\max_{\boldsymbol{y}_{1:M}} \Psi(\boldsymbol{w}, \boldsymbol{y}_{1:M}) = \max_{\boldsymbol{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}).$$

- This score involves a maximization over all tag sequences of length M, written $\max_{y1:M}$. This maximization can be broken into two pieces

$$\max_{\boldsymbol{y}_{1:M}} \Psi(\boldsymbol{w}, \boldsymbol{y}_{1:M}) = \max_{y_M} \max_{\boldsymbol{y}_{1:M-1}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1})$$

Within the sum, only the final term $s_{M+1}(\blacklozenge, y_M)$ depends on y_M, so we can pull this term out of the second maximization,

$$\max_{\boldsymbol{y}_{1:M}} \Psi(\boldsymbol{w}, \boldsymbol{y}_{1:M}) = \left(\max_{y_M} s_{M+1}(\blacklozenge, y_M) \right) + \left(\max_{\boldsymbol{y}_{1:M-1}} \sum_{m=1}^{M} s_m(y_m, y_{m-1}) \right)$$

- The second term in last equation has the same form as our original problem, with M replaced by M−1. This indicates that the problem can be reformulated as a recurrence.

- We do this by defining an auxiliary variable called the Viterbi variable $v_m(k)$, representing the score of the best sequence terminating in the tag k

$$v_m(y_m) \triangleq \max_{\boldsymbol{y}_{1:m-1}} \sum_{n=1}^{m} s_n(y_n, y_{n-1})$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + \max_{\boldsymbol{y}_{1:m-2}} \sum_{n=1}^{m-1} s_n(y_n, y_{n-1})$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + v_{m-1}(y_{m-1}).$$

- Each set of Viterbi variables is computed from the local score $s_m(y_m, y_{m-1})$, and from the previous set of Viterbi variables.
- The initial condition of the recurrence is simply the score for the first tag

$$v_1(y_1) \triangleq s_1(y_1, \lozenge).$$

- The maximum overall score for the sequence is then the final Viterbi variable

$$\max_{\boldsymbol{y}_{1:M}} \Psi(\boldsymbol{w}_{1:M}, \boldsymbol{y}_{1:M}) = v_{M+1}(\blacklozenge)$$

- Thus, the score of the best labeling for the sequence can be computed in a single forward sweep:
  - first compute all variables v1(·)
  - and then compute all variables v2(·) using the recurrence
  - continuing until the final variable $v_{M+1}(\blacklozenge)$

Algorithm 11 The Viterbi algorithm. Each $s_m(k, k')$ is a local score for tag $y_m = k$ and $y_{m-1} = k'$.

---

for $k \in \{0, \dots K\}$ **do**
  $v_1(k) = s_1(k, \Diamond)$
for $m \in \{2, \dots, M\}$ **do**
  for $k \in \{0, \dots, K\}$ **do**
    $v_m(k) = \max_{k'} s_m(k, k') + v_{m-1}(k')$
    $b_m(k) = \text{argmax}_{k'} s_m(k, k') + v_{m-1}(k')$
$y_M = \text{argmax}_k s_{M+1}(\blacklozenge, k) + v_M(k)$
for $m \in \{M-1, \dots 1\}$ **do**
  $y_m = b_m(y_{m+1})$
return $\boldsymbol{y}_{1:M}$

- The Viterbi variables can be arranged in a structure known as a trellis
- Each column indexes a token min the sequence, and each row indexes a tag in Y; every $v_{m-1}(k)$ is connected to every $v_m(k')$, indicating that $v_m(k')$ is computed from $v_{m-1}(k)$.

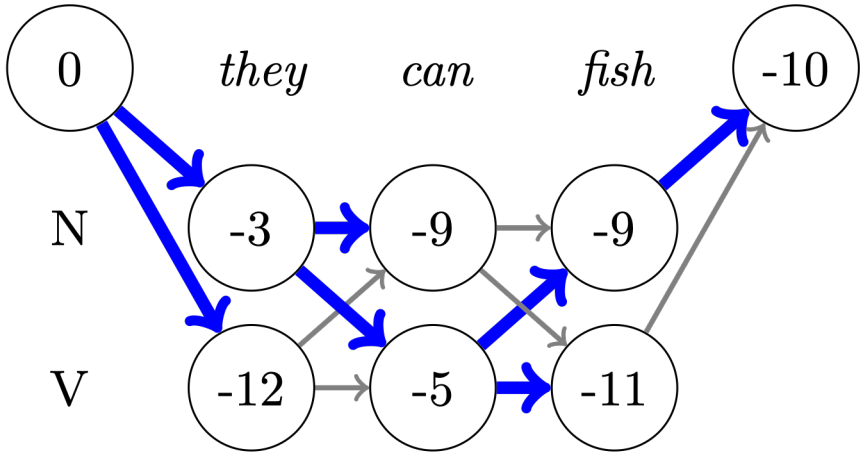
Example

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

(a) Weights for emission features.

	N	V	♦
◊	−1	−2	−∞
N	−3	−1	−1
V	−1	−3	−1

(b) Weights for transition features. The "from" tags are on the columns, and the "to" tags are on the rows.



- Consider the minimal tagset {N,V}, corresponding to nouns and verbs.
- We begin with $v_1(N)$, which has only one possible predecessor, the start tag .
- This score is therefore equal to $v_1(N)=s_1(N,\ )=(w_m = \text{they}, y_m = N) + (y_m = N, y_{m-1} = \ )=-2-1=-3$
- Similarly, $v_1(V)=s_1(V,\ )=(w_m = \text{they}, y_m = V) + (y_m = V, y_{m-1} = \ )=-10-2=-12$

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

(a) Weights for emission features.

	N	V	♦
◊	−1	−2	−∞
N	−3	−1	−1
V	−1	−3	−1

(b) Weights for transition features. The "from" tags are on the columns, and the "to" tags are on the rows.

- $v_2(N) = \max(v_1(N) + s_2(N,N), v_1(V) + s_2(N,V))$

$s_2(N,N)=(w_m = can, y_m = N) + (y_m = N, y_{m-1} = N)=-3-3=-6$

$s_2(N,V)=(w_m = can, y_m = V) + (y_m = V, y_{m-1} = N)=-1-1=-2$

$v_2(N) = \max(-3-6, -12-2)=\mathbf{-9}$

$v_2(V) = \max(v_1(N) + s_2(V,N), v_1(V) + s_2(V,V))$

$s_2(V,N)=(w_m = can, y_m = V) + (y_m = V, y_{m-1} = N)=-1-1=-2$

$s_2(V,V)=(w_m = can, y_m = V) + (y_m = V, y_{m-1} = V)=-1 -3 = -4$

$v_2(V) = \max(-3-2, -12-4)=\mathbf{-5}$

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

(a) Weights for emission features.

	N	V	♦
◊	−1	−2	−∞
N	−3	−1	−1
V	−1	−3	−1

(b) Weights for transition features. The "from" tags are on the columns, and the "to" tags are on the rows.

- $v_3(N) = \max(v_2(N) + s_3(N,N), v_2(V) + s_3(N,V))$
- $s_3(N,N)=(w_m = fish, y_m = N) + (y_m = N, y_{m-1} = N)=-3-3=-6$
- $s_3(N,V)=(w_m = fish, y_m = V) + (y_m = V, y_{m-1} = N)=-3-1=-4$
- $v_3(N) = \max(-9-6,-5-4)=\mathbf{-9}$
- $v_3(V) = \max(v_2(N) + s_3(V,N), v_2(V) + s_3(V,V))$
- $s_3(V,N)=(w_m = fish, y_m = V) + (y_m = V, y_{m-1} = N)=-3-1=-4$
- $s_3(V,V)=(w_m = fish, y_m = V) + (y_m = V, y_{m-1} = V)=-3-3=-6$
- $v_3(V) = \max(-9-4,-5-6)=\mathbf{-11}$

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

(a) Weights for emission features.

	N	V	♦
◊	−1	−2	$-\infty$
N	−3	−1	−1
V	−1	−3	−1

(b) Weights for transition features. The "from" tags are on the columns, and the "to" tags are on the rows.

- $v_4(\quad) = \max(v_3(N) + s_4(\quad, N), v_3(V) + s_4(\quad, V))$
- $s_4(\quad, N) = (y_m = \quad, y_{m-1} = N) = -1$
- $s_4(\quad, V) = (y_m = \quad, y_{m-1} = V) = -1$
- $v_4(\quad) = \max(-9-1, -11-1) = \mathbf{-10}$

- The original goal was to find the best scoring sequence, not simply to compute its score.
- But by solving the auxiliary problem, we are almost there. Recall that each $v_m(k)$ represents the score of the best tag sequence ending in that tag k in position m.
- To compute this, we maximize over possible values of y_{m-1}. By keeping track of the "argmax" tag that maximizes this choice at each step, we can walk backwards from the final tag, and recover the optimal tag sequence.

Complexity of THE VITERBI ALGORITHM

- If there are K tags and M positions in the sequence, then there are M ×K Viterbi variables to compute.
- Computing each variable requires finding a maximum over K possible predecessor tags.
- The total time complexity of populating the trellis is therefore $O(MK^2)$, with an additional factor for the number of active features at each position.
- After completing the trellis, we simply trace the backwards pointers to the beginning of the sequence, which takes O(M) operations.

Hidden Markov Models

- The Viterbi sequence labeling algorithm is built on the scores $s_m(y,y')$.
- These scores can be estimated probabilistically.
- Probabilistic Naıve Bayes classifier selects the label y to maximize $p(y|x) \propto p(y,x)$.
- In probabilistic sequence labeling, our goal is similar: select the tag sequence that maximizes $p(y|w) \propto p(y,w)$.
- Naıve Bayes was introduced as a generative model — a probabilistic story that explains the observed data as well as the hidden label.
- A similar story can be constructed for probabilistic sequence labeling

- First, the tags are drawn from a prior distribution;
- next, the tokens are drawn from a conditional likelihood.
- However, for inference to be tractable, additional independence assumptions are required.
- First, the probability of each token depends only on its tag, and not on any other element in the sequence

$$\mathrm{p}(\boldsymbol{w} \mid \boldsymbol{y}) = \prod_{m=1}^{M} \mathrm{p}(w_m \mid y_m)$$

- Second, each tag y_m depends only on its predecessor

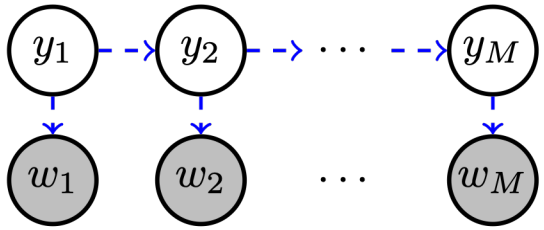
$$\mathrm{p}(\boldsymbol{y}) = \prod_{m=1}^{M} \mathrm{p}(y_m \mid y_{m-1})$$

- where y_0 = in all cases.
- Due to this Markov assumption, probabilistic sequence labelling models are known as hidden Markov models (HMMs).

Algorithm Generative process for the hidden Markov model

$y_0 \leftarrow \lozenge, \quad m \leftarrow 1$
repeat
  $y_m \sim \text{Categorical}(\boldsymbol{\lambda}_{y_{m-1}})$ ▷ sample the current tag
  $w_m \sim \text{Categorical}(\boldsymbol{\phi}_{y_m})$ ▷ sample the current word
until $y_m = \blacklozenge$ ▷ terminate when the stop symbol is generated

- Given the parameters λ and ϕ, we can compute p(w,y) for any token sequence w and tag sequence y.
- The HMM is often represented as a graphical model



- This representation makes the independence assumptions explicit:
  - if a variable v1 is probabilistically conditioned on another variable v2, then there is an arrow $v_2 \rightarrow v_1$ in the diagram.
  - If there are no arrows between v_1 and v_2, they are conditionally independent

Estimation

- Emission probabilities: The probability $p_e(w_m \mid y_m; \phi)$ is the emission probability, since the words are treated as probabilistically "emitted", conditioned on the tags.
- Transition probabilities: The probability $p_t(y_m \mid y_{m-1}; \lambda)$ is the transition probability, since it assigns probability to each possible tag-to-tag transition.
- Both of these groups of parameters are typically computed from smoothed relative frequency estimation on a labeled corpus.
- The un-smoothed probabilities are,

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

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

Discriminative sequence labeling with features

- Today, hidden Markov models are rarely used for supervised sequence labeling.
- This is because HMMs are limited to only two phenomena
  - word-tag compatibility, via the emission probability $p_{W|Y}(w_m \mid y_m)$;
  - local context, via the transition probability $p_Y(y_m \mid y_{m-1})$.
- The Viterbi algorithm permits the inclusion of richer information in the local scoring function $\psi(w_{1:M}, y_m, y_{m-1}, m)$, which can be defined as a weighted sum of arbitrary local features,

$$\psi(w, ym, ym-1, m) = \theta \cdot f(w, y_m, y_{m-1}, m)$$

where f is a locally-defined feature function, and θ is a vector of weights.

- The local decomposition of the scoring function Ψ is reflected in a corresponding decomposition of the feature function

$$\begin{aligned}\Psi(\boldsymbol{w}, \boldsymbol{y}) &= \sum_{m=1}^{M+1} \psi(\boldsymbol{w}, y_m, y_{m-1}, m) \\ &= \sum_{m=1}^{M+1} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{w}, y_m, y_{m-1}, m) \\ &= \boldsymbol{\theta} \cdot \sum_{m=1}^{M+1} \boldsymbol{f}(\boldsymbol{w}, y_m, y_{m-1}, m) \\ &= \boldsymbol{\theta} \cdot \boldsymbol{f}^{(\text{global})}(\boldsymbol{w}, \boldsymbol{y}_{1:M}),\end{aligned}$$

where $\boldsymbol{f}^{(\text{global})}(\boldsymbol{w}, \boldsymbol{y})$ is a global feature vector, which is a sum of local feature vectors

- Let’s now consider what additional information these features might encode
- Word affix features
- Consider the problem of part-of-speech tagging on the first four lines of the 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.

- Many of these words were made up by the author of the poem, so a corpus would offer no information about their probabilities of being associated with any particular part of speech.
- Context helps: for example, the word slithy follows the determiner *the*, so it is probably a noun or adjective.
- The suffix -thy is found in a number of adjectives, like healthy, wealthy, worthy.
- It is also found in a handful of nouns — e.g., apathy, sympathy — but nearly all of these have the longer suffix -pathy, unlike slithy.
- So the suffix gives some evidence that slithy is an adjective, and indeed it is

- **Fine-grained context**
- The hidden Markov model captures contextual information in the form of part-of-speech tag bigrams.
- Consider the noun phrases *this fish* and *these fish*.
- Many part-of-speech tagsets distinguish between singular and plural nouns, but do not distinguish between singular and plural determiners
- A hidden Markov model would be unable to correctly label fish as singular or plural in both of these cases, because it only has access to two features: the preceding tag (determiner in both cases) and the word (fish in both cases).

- Consider the tagging D J N (determiner, adjective, noun) for the sequence *the slithy toves*, so that

  w=the slithy toves

  y=D J N.
- Let's create the feature vector for this example, assuming that we have word-tag features (indicated by W), tag-tag features (indicated by T), and suffix features (indicated by M).
- You can assume that you have access to a method for extracting the suffix -thy from slithy, -es from toves, and Ø from the, indicating that this word has no suffix.

- The resulting feature vector is,

$$
\begin{aligned}
\boldsymbol{f}(\text{the slithy toves}, \text{D J N}) = & \boldsymbol{f}(\text{the slithy toves}, \text{D}, \lozenge, 1) \\
& + \boldsymbol{f}(\text{the slithy toves}, \text{J}, \text{D}, 2) \\
& + \boldsymbol{f}(\text{the slithy toves}, \text{N}, \text{J}, 3) \\
& + \boldsymbol{f}(\text{the slithy toves}, \blacklozenge, \text{N}, 4) \\
= & \{(T : \lozenge, \text{D}), (W : \text{the}, \text{D}), (M : \varnothing, \text{D}), \\
& (T : \text{D}, \text{J}), (W : \text{slithy}, \text{J}), (M : \text{-thy}, \text{J}), \\
& (T : \text{J}, \text{N}), (W : \text{toves}, \text{N}), (M : \text{-es}, \text{N}) \\
& (T : \text{N}, \blacklozenge)\}.
\end{aligned}
$$

- These examples show that local features can incorporate information that lies beyond the scope of a hidden Markov model.
- Because the features are local, it is possible to apply the Viterbi algorithm to identify the optimal sequence of tags.
- The remaining question is how to estimate the weights on these features

- **Structured perceptron**
- The perceptron classifier is trained by increasing the weights for features that are associated with the correct label, and decreasing the weights for features that are associated with incorrectly predicted labels

$$\hat{y} = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y)$$

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \boldsymbol{f}(\boldsymbol{x}, y) - \boldsymbol{f}(\boldsymbol{x}, \hat{y}).$$

- This learning algorithm is called structured perceptron, because it learns to predict the structured output y.

- Example: For the example *they can fish*, suppose that the reference tag sequence is y(i) = N V V, but the tagger incorrectly returns the tag sequence $\hat{y}$= N V N.
- Assuming a model with features for emissions (w_m,y_m) and transitions (y_{m-1},y_m), the corresponding structured perceptron update is

$$\theta_{(\textit{fish},\mathrm{V})} \leftarrow \theta_{(\textit{fish},\mathrm{V})} + 1, \qquad \theta_{(\textit{fish},\mathrm{N})} \leftarrow \theta_{(\textit{fish},\mathrm{N})} - 1$$
$$\theta_{(\mathrm{V},\mathrm{V})} \leftarrow \theta_{(\mathrm{V},\mathrm{V})} + 1, \qquad \theta_{(\mathrm{V},\mathrm{N})} \leftarrow \theta_{(\mathrm{V},\mathrm{N})} - 1$$
$$\theta_{(\mathrm{V},\blacklozenge)} \leftarrow \theta_{(\mathrm{V},\blacklozenge)} + 1, \qquad \theta_{(\mathrm{N},\blacklozenge)} \leftarrow \theta_{(\mathrm{N},\blacklozenge)} - 1.$$