## Sequence Labeling

- Many information extraction tasks can be formulated as *sequence labeling* tasks. Sequence labelers assign a class label to each item in a sequential structure.
- Sequence labeling methods are appropriate for problems where the class of an item depends on other (typically nearby) items in the sequence.
- Examples of sequential labeling tasks: part-of-speech tagging, syntactic chunking, named entity recognition.
- A naive approach would consider all possible label sequences and choose the best one. But that is too expensive, we need more efficient methods.





## Markov Models

- A **Markov Chain** is a finite-state automaton that has a probability associated with each transition (arc), where the input uniquely defines the transitions that can be taken.
- In a **first-order** Markov chain, the probability of a state depends only on the previous state, where $q_i \epsilon Q$ are states:

  Markov Assumption: $P(q_i \mid q_1 ... q_{i-1}) = P(q_i \mid q_{i-1})$

  The probabilities of all of the outgoing arcs of a state must sum to 1.
- The Markov chain can be traversed to compute the probability of a particular sequence of labels.





## Hidden Markov Models

- A **Hidden Markov Model (HMM)** is used to find the best assignment of class labels for a sequence of input tokens. It finds the most likely sequence of labels for the input as a whole.
- The input tokens are the **observed** events.
- The class labels are the **hidden** events, such as part-of-speech tags or Named Entity classes.
- The goal of an HMM is to recover the hidden events from the observed events (i.e., to recover class labels for the input tokens).





## Using Hidden Markov Models

- We typically use first-order HMMs, which is first-order Markov chain which assumes that the probability of an observation depends only on the state that produced it (i.e., it is independent of other states and observations).

  *Observation Independence:* $P(o_i|q_i)$
  where $o_i \epsilon O$ are the observations.
- For information extraction, we typically use HMMs as a **decoder**. Given an HMM and input sequence, we want to discover the label sequence (hidden states) that is most likely.
- Each state typically represents a class label (i.e., the hidden state that will be recovered). Consequently, we need two sets of probabilities: $P(word_i \mid tag_i)$ and $P(tag_i \mid tag_{i-1})$.

## The Viterbi Algorithm

- The Viterbi algorithm is used to compute the most likely label sequence in $O(W * T^2)$ time, where $T$ is the number of possible labels (tags) and $W$ is the number of words in the sentence.
- The algorithm sweeps through all the label possibilities for each word, computing the best sequence leading to each possibility. The key that makes this algorithm efficient is that we only need to know the best sequences leading to the previous word because of the Markov assumption.





## Computing the Probability of a Sentence and Tags

We want to find the sequence of tags that maximizes the formula $P(T_1..T_n \mid w_i..w_n)$, which can be estimated as:

$$\prod_{i=1}^{n} P(T_i \mid T_{i-1}) * P(w_i \mid T_i)$$

$P(T_i \mid T_{i-1})$ is computed by multiplying the arc values in the HMM.

$P(w_i \mid T_i)$ is computed by multiplying the lexical generation probabilities associated with each word.





## The Viterbi Algorithm

Let $T$ = # of tags  $W$ = # of words in the sentence

for t = 1 to $T$ **/* Initialization Step */**
  Score(t, 1) = Pr($Word_1 \mid Tag_t$) * Pr($Tag_t \mid \phi$)
BackPtr(t, 1) = 0;

---

for w = 2 to $W$ **/* Iteration Step */**
  for t = 1 to $T$
    Score(t, w) = Pr($Word_w \mid Tag_t$) *
      $MAX_{j=1,T}$ (Score(j, w-1) * Pr($Tag_t \mid Tag_j$))
    BackPtr(t, w) = index of j that gave the max above

---

Seq($W$) = t that maximizes Score(t,$W$) **/* Sequence Identification */**
for w = $W$-1 to 1
  Seq(w) = BackPtr(Seq(w+1),w+1)





## Assigning Tags Probabilistically

- Instead of identifying only the best tag for each word, another approach is to assign a probability to each tag.
- We could use simple frequency counts to estimate context-independent probabilities.

  P(tag | word) = $\frac{\#times\ word\ occurs\ with\ the\ tag}{\#times\ word\ occurs}$

- But these estimates are unreliable because they do not take context into account.
- A better approach considers how likely a tag is for a word given the specific sentence and words around it!

## An Example

Consider the sentence: *Outside pets are often hit by cars.*

Assume “outside” has 4 possible tags: ADJ, NOUN, PREP, ADVERB.
Assume “pets” has 2 possible tags: VERB, NOUN.

If “outside” is a ADJ or PREP then “pets” has to be a NOUN.
If “outside” is a ADV or NOUN then “pets” may be a NOUN or VERB.

Now we can sum the probabilities of all tag sequences that end with “pets” as a NOUN and sum the probabilities of all tag sequences that end with “pets” as a VERB. For this sentence, the chances that “pets” is a NOUN should be much higher.





## Forward Probability

- The *forward probability* $\alpha_i$(m) is the probability of words $w_1...w_m$ with $w_m$ having tag $T_i$.

$$\alpha_i(m) = P(w_1...w_m \ \& \ w_m/T_i)$$

- The forward probability is computed as the sum of the probabilities computed for all tag sequences ending in tag $T_i$ for word $w_m$.

  Ex: $\alpha_1(2)$ would be the sum of probabilities computed for all tag sequences ending in tag #1 for word #2.

- The lexical tag probability is computed as:

$$P(w_m/T_i \mid w_1...w_m) = \frac{P(w_m/T_i, w_1...w_m)}{P(w_1...w_m)}$$

which we estimate as:

$$P(w_m/T_i \mid w_1...w_m) = \frac{\alpha_i(m)}{\sum_{j=1}^{T} \alpha_j(m)}$$





## The Forward Algorithm

Let $T$ = # of tags  $W$ = # of words in the sentence

for t = 1 to $T$ **/* Initialization Step */**
  $SeqSum(t, 1)$ = Pr($Word_1 \mid Tag_t$) * Pr($Tag_t \mid \phi$)

---

for w = 2 to $W$ **/* Compute Forward Probs */**
  for t = 1 to $T$
    $SeqSum(t, w)$ = Pr($Word_w \mid Tag_t$) *
    $\sum_{j=1,T} (SeqSum(j, w-1)$ * Pr($Tag_t \mid Tag_j$))

---

for w = 1 to $W$ **/* Compute Lexical Probs */**
  for t = 1 to $T$
    Pr($Seq_w$=$Tag_t$) = $\frac{SeqSum(t,w)}{\sum_{j=1,T} SeqSum(j, w)}$





## Backward Probability

- *Backward probability* $\beta_i$(m) is the probability of words $w_m...w_N$ with $w_m$ having tag $T_i$.

$$\beta_i(m) = P(w_m...w_N \ \& \ w_m/T_i)$$

- The backward probability is computed as the sum of the probabilities computed for all tag sequences beginning with tag $T_i$ for word $w_m$.
- The algorithm for computing the backward probability is analogous to the forward probability except that we start at the end of the sentence and sweep backwards.
- The best way to estimate lexical tag probabilities uses both forward and backward probabilities:

$$P(w_m/T_i) = \frac{(\alpha_i(m) * \beta_i(m))}{\sum_{j=1}^{T} (\alpha_j(m) * \beta_j(m))}$$