Natural Language Processing

Lecture IV. Part-of-speech (POS) tagging and Named Entity Recognition (NER)

Forrest Sheng Bao, Ph.D.

Dept. of Computer Science
Iowa State University
Ames, IA 50011

September 23, 2021

Outline

What is POS tagging

POS tagging in HMM

CRF for POS tagging and NER

POS

Natural	Language	Processing	is	a	field	of	computer	science.
Adj.	n.	n.	v.	dt.	n.	cj.	n.	n.

Part of speech tagging

- "In traditional grammar, a part of speech (abbreviated form: PoS or POS) is a category of words (or, more generally, of lexical items) which have similar grammatical properties. "
- "In corpus linguistics, part-of-speech tagging (POS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech,
- based on both its definition and its context, i.e., its relationship with adjacent and related words in a phrase, sentence, or paragraph."
- Brill tagger (circa. 1993): the first English POS tagger, rule-based. It assigns initial tags to words first and then use rules to iteratively update tags based on, e.g., context.
- Brill tagger has hundreds of rules.

Part of speech tagging

- “In traditional grammar, a part of speech (abbreviated form: PoS or POS) is a category of words (or, more generally, of lexical items) which have similar grammatical properties. ”
- “In corpus linguistics, part-of-speech tagging (POS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech,
- based on both its definition and its context, i.e., its relationship with adjacent and related words in a phrase, sentence, or paragraph.”
- Brill tagger (circa. 1993): the first English POS tagger, rule-based. It assigns initial tags to words first and then use rules to iteratively update tags based on, e.g., context.
- Brill tagger has hundreds of rules.

Part of speech tagging

- “In traditional grammar, a part of speech (abbreviated form: PoS or POS) is a category of words (or, more generally, of lexical items) which have similar grammatical properties. ”
- “In corpus linguistics, part-of-speech tagging (POS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech,
- based on both its definition and its context, i.e., its relationship with adjacent and related words in a phrase, sentence, or paragraph.”
- Brill tagger (circa. 1993): the first English POS tagger, rule-based. It assigns initial tags to words first and then use rules to iteratively update tags based on, e.g., context.
- Brill tagger has hundreds of rules.

Part of speech tagging

- "In traditional grammar, a part of speech (abbreviated form: PoS or POS) is a category of words (or, more generally, of lexical items) which have similar grammatical properties. "
- "In corpus linguistics, part-of-speech tagging (POS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech,
- based on both its definition and its context, i.e., its relationship with adjacent and related words in a phrase, sentence, or paragraph."
- Brill tagger (circa. 1993): the first English POS tagger, rule-based. It assigns initial tags to words first and then use rules to iteratively update tags based on, e.g., context.
- Brill tagger has hundreds of rules.

Part of speech tagging

- “In traditional grammar, a part of speech (abbreviated form: PoS or POS) is a category of words (or, more generally, of lexical items) which have similar grammatical properties. ”
- “In corpus linguistics, part-of-speech tagging (POS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech,
- based on both its definition and its context, i.e., its relationship with adjacent and related words in a phrase, sentence, or paragraph.”
- Brill tagger (circa. 1993): the first English POS tagger, rule-based. It assigns initial tags to words first and then use rules to iteratively update tags based on, e.g., context.
- Brill tagger has hundreds of rules.

Tags used in Penn Treebank

- Nine common parts of speech in English: noun, verb, article, adjective, preposition, pronoun, adverb, conjunction, and interjection.
- Most NLP researchers use Penn Treebank tags, which are finer than common English POSes mentioned above.
- https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Tags used in Penn Treebank

- Nine common parts of speech in English: noun, verb, article, adjective, preposition, pronoun, adverb, conjunction, and interjection.
- Most NLP researchers use Penn Treebank tags, which are finer than common English POSes mentioned above.
- https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Tags used in Penn Treebank

- Nine common parts of speech in English: noun, verb, article, adjective, preposition, pronoun, adverb, conjunction, and interjection.
- Most NLP researchers use Penn Treebank tags, which are finer than common English POSes mentioned above.
- `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`

Probabilistic Model for Tagging

- Problem of using rule-based system: Very difficult to verify and to scale.
- Probabilistic approach: there are many sequences of tags, but only one yields (i.e., argmax) the highest probability.

	Natural	Language	Processing	is	a	field	of	computer	science.
$\mathbf{T}_1$	aj.	n.	n.	v.	dt.	n.	cj.	n.	n.
$\mathbf{T}_2$	n.	n.	n.	v.	n.	n.	cj.	n.	n.
$\mathbf{T}_3$	v.	n.	av.	v.	dt.	n.	cj.	n.	n.
$\mathbf{T}_4$	dt.	n.	n.	v.	v.	n.	aj.	v.	n.
$\mathbf{T}_5$	cj.	n.	n.	v.	dt.	n.	cj.	n.	n.

$\mathbf{T}_2$ - $\mathbf{T}_5$ apparently make no sense and hence their $P()$'s are very low.

- The goal is to find the most likely sequence of tags ($\mathbf{T}$), given the sequence of words ($\mathbf{W}$), i.e.,

$$\underset{\mathbf{T}}{\operatorname{argmax}}\, P(\mathbf{T}|\mathbf{W})$$

Probabilistic Model for Tagging

- Problem of using rule-based system: Very difficult to verify and to scale.
- Probabilistic approach: there are many sequences of tags, but only one yields (i.e., argmax) the highest probability.

	Natural	Language	Processing	is	a	field	of	computer	science.
$\mathbf{T}_1$	aj.	n.	n.	v.	dt.	n.	cj.	n.	n.
$\mathbf{T}_2$	n.	n.	n.	v.	n.	n.	cj.	n.	n.
$\mathbf{T}_3$	v.	n.	av.	v.	dt.	n.	cj.	n.	n.
$\mathbf{T}_4$	dt.	n.	n.	v.	v.	n.	aj.	v.	n.
$\mathbf{T}_5$	cj.	n.	n.	v.	dt.	n.	cj.	n.	n.

$\mathbf{T}_2$ - $\mathbf{T}_5$ apparently make no sense and hence their $P()$'s are very low.

- The goal is to find the most likely sequence of tags ($\mathbf{T}$), given the sequence of words ($\mathbf{W}$), i.e.,

$$\underset{\mathbf{T}}{\operatorname{argmax}}\, P(\mathbf{T}|\mathbf{W})$$

Probabilistic Model for Tagging

- Problem of using rule-based system: Very difficult to verify and to scale.
- Probabilistic approach: there are many sequences of tags, but only one yields (i.e., argmax) the highest probability.

	Natural	Language	Processing	is	a	field	of	computer	science.
$\mathbf{T}_1$	aj.	n.	n.	v.	dt.	n.	cj.	n.	n.
$\mathbf{T}_2$	n.	n.	n.	v.	n.	n.	cj.	n.	n.
$\mathbf{T}_3$	v.	n.	av.	v.	dt.	n.	cj.	n.	n.
$\mathbf{T}_4$	dt.	n.	n.	v.	v.	n.	aj.	v.	n.
$\mathbf{T}_5$	cj.	n.	n.	v.	dt.	n.	cj.	n.	n.

  $\mathbf{T}_2$ - $\mathbf{T}_5$ apparently make no sense and hence their $P()$'s are very low.
- The goal is to find the most likely sequence of tags ($\mathbf{T}$), given the sequence of words ($\mathbf{W}$), i.e.,

$$\operatorname*{argmax}_{\mathbf{T}} P(\mathbf{T}|\mathbf{W})$$

A generative model for POS tagging

- Make use of Bayes' rule:

$$\operatorname*{argmax}_{\mathbf{T}} P(\mathbf{T}|\mathbf{W}) = \operatorname*{argmax}_{\mathbf{T}} \frac{P(\mathbf{W}|\mathbf{T})P(\mathbf{T})}{P(\mathbf{W})} \tag{1}$$

$$= \operatorname*{argmax}_{\mathbf{T}} P(\mathbf{W}|\mathbf{T})P(\mathbf{T}) \tag{2}$$

Note that we go from Eq. (1) to Eq. (2) because $P(\mathbf{W})$ is not a function of $\mathbf{T}$.

- tag-to-tag *transition* probabilities: $P(\mathbf{T}) = \Pi_{i=1}^{l+1} P_T(t_i|t_{i-1})$: e.g., NN comes after DET
- tag-to-word *emission* probabilities: $P(\mathbf{W}|\mathbf{T}) = \Pi_{i=1}^{l} P_E(w_i|t_i)$: e.g., "natural" is probably a JJ.
- A sequence of words is generated in two phases:
  1. Produce a sequence of tags, e.g., , NN, DET, …, based on probability between each two consecutive tags.
  2. For each tag, produce a word, e.g., NN "language", DT "the".
- Yes, a semantically meaningless sentence can be tagged and further parsed correctly, e.g., "NLP is a automobile of GOP".

A generative model for POS tagging

- Make use of Bayes' rule:

$$\operatorname*{argmax}_{\mathbf{T}} P(\mathbf{T}|\mathbf{W}) = \operatorname*{argmax}_{\mathbf{T}} \frac{P(\mathbf{W}|\mathbf{T})P(\mathbf{T})}{P(\mathbf{W})} \tag{1}$$

$$= \operatorname*{argmax}_{\mathbf{T}} P(\mathbf{W}|\mathbf{T})P(\mathbf{T}) \tag{2}$$

Note that we go from Eq. (1) to Eq. (2) because $P(\mathbf{W})$ is not a function of $\mathbf{T}$.

- tag-to-tag *transition* probabilities: $P(\mathbf{T}) = \Pi_{i=1}^{l+1} P_T(t_i|t_{i-1})$: e.g., NN comes after DET
- tag-to-word *emission* probabilities: $P(\mathbf{W}|\mathbf{T}) = \Pi_{i=1}^{l} P_E(w_i|t_i)$: e.g., "natural" is probably a JJ.
- A sequence of words is generated in two phases:
- Yes, a semantically meaningless sentence can be tagged and further parsed correctly, e.g., "NLP is a automobile of GOP".

A generative model for POS tagging

- Make use of Bayes' rule:

$$\underset{\mathbf{T}}{\operatorname{argmax}} P(\mathbf{T}|\mathbf{W}) = \underset{\mathbf{T}}{\operatorname{argmax}} \frac{P(\mathbf{W}|\mathbf{T})P(\mathbf{T})}{P(\mathbf{W})} \quad (1)$$

$$= \underset{\mathbf{T}}{\operatorname{argmax}} P(\mathbf{W}|\mathbf{T})P(\mathbf{T}) \quad (2)$$

Note that we go from Eq. (1) to Eq. (2) because $P(\mathbf{W})$ is not a function of $\mathbf{T}$.

- tag-to-tag *transition* probabilities: $P(\mathbf{T}) = \Pi_{i=1}^{l+1} P_T(t_i|t_{i-1})$: e.g., NN comes after DET
- tag-to-word *emission* probabilities: $P(\mathbf{W}|\mathbf{T}) = \Pi_{i=1}^{l} P_E(w_i|t_i)$: e.g., "natural" is probably a JJ.
- A sequence of words is generated in two phases:
  1. Produce a sequence of tags, e.g., ▷, NN, DET, ..., based on probability between each two consecutive tags.
  2. For each tag, produce a word, e.g., NN → "language", DT → "the".
- Yes, a semantically meaningless sentence can be tagged and further parsed correctly, e.g., "NLP is a automobile of GOP".

A generative model for POS tagging

- Make use of Bayes' rule:

$$\operatorname*{argmax}_{\mathbf{T}} P(\mathbf{T}|\mathbf{W}) = \operatorname*{argmax}_{\mathbf{T}} \frac{P(\mathbf{W}|\mathbf{T})P(\mathbf{T})}{P(\mathbf{W})} \quad (1)$$

$$= \operatorname*{argmax}_{\mathbf{T}} P(\mathbf{W}|\mathbf{T})P(\mathbf{T}) \quad (2)$$

  Note that we go from Eq. (1) to Eq. (2) because $P(\mathbf{W})$ is not a function of $\mathbf{T}$.
- tag-to-tag *transition* probabilities: $P(\mathbf{T}) = \Pi_{i=1}^{l+1} P_T(t_i|t_{i-1})$: e.g., NN comes after DET
- tag-to-word *emission* probabilities: $P(\mathbf{W}|\mathbf{T}) = \Pi_{i=1}^{l} P_E(w_i|t_i)$: e.g., "natural" is probably a JJ.
- A sequence of words is generated in two phases:
  1. Produce a sequence of tags, e.g., $\triangleright$, NN, DET, ..., based on probability between each two consecutive tags.
  2. For each tag, produce a word, e.g., NN $\rightarrow$ "language", DT $\rightarrow$ "the".
- Yes, a semantically meaningless sentence can be tagged and further parsed correctly, e.g., "NLP is a automobile of GOP".

A generative model for POS tagging

- Make use of Bayes' rule:

$$\operatorname*{argmax}_{\mathbf{T}} P(\mathbf{T}|\mathbf{W}) = \operatorname*{argmax}_{\mathbf{T}} \frac{P(\mathbf{W}|\mathbf{T})P(\mathbf{T})}{P(\mathbf{W})} \quad (1)$$

$$= \operatorname*{argmax}_{\mathbf{T}} P(\mathbf{W}|\mathbf{T})P(\mathbf{T}) \quad (2)$$

Note that we go from Eq. (1) to Eq. (2) because $P(\mathbf{W})$ is not a function of $\mathbf{T}$.

- tag-to-tag *transition* probabilities: $P(\mathbf{T}) = \Pi_{i=1}^{l+1} P_T(t_i|t_{i-1})$: e.g., NN comes after DET
- tag-to-word *emission* probabilities: $P(\mathbf{W}|\mathbf{T}) = \Pi_{i=1}^{l} P_E(w_i|t_i)$: e.g., "natural" is probably a JJ.
- A sequence of words is generated in two phases:
  1. Produce a sequence of tags, e.g., $\triangleright$, NN, DET, ..., based on probability between each two consecutive tags.
  2. For each tag, produce a word, e.g., NN $\rightarrow$ "language", DT $\rightarrow$ "the".
- Yes, a semantically meaningless sentence can be tagged and further parsed correctly, e.g., "NLP is a automobile of GOP".

A generative model for POS tagging

- Make use of Bayes' rule:

$$\operatorname*{argmax}_{\mathbf{T}} P(\mathbf{T}|\mathbf{W}) = \operatorname*{argmax}_{\mathbf{T}} \frac{P(\mathbf{W}|\mathbf{T})P(\mathbf{T})}{P(\mathbf{W})} \quad (1)$$

$$= \operatorname*{argmax}_{\mathbf{T}} P(\mathbf{W}|\mathbf{T})P(\mathbf{T}) \quad (2)$$

  Note that we go from Eq. (1) to Eq. (2) because $P(\mathbf{W})$ is not a function of $\mathbf{T}$.
- tag-to-tag *transition* probabilities: $P(\mathbf{T}) = \Pi_{i=1}^{l+1} P_T(t_i|t_{i-1})$: e.g., NN comes after DET
- tag-to-word *emission* probabilities: $P(\mathbf{W}|\mathbf{T}) = \Pi_{i=1}^{l} P_E(w_i|t_i)$: e.g., "natural" is probably a JJ.
- A sequence of words is generated in two phases:
  1. Produce a sequence of tags, e.g., $\triangleright$, NN, DET, ..., based on probability between each two consecutive tags.
  2. For each tag, produce a word, e.g., NN $\rightarrow$ "language", DT $\rightarrow$ "the".
- Yes, a semantically meaningless sentence can be tagged and further parsed correctly, e.g., "NLP is a automobile of GOP".

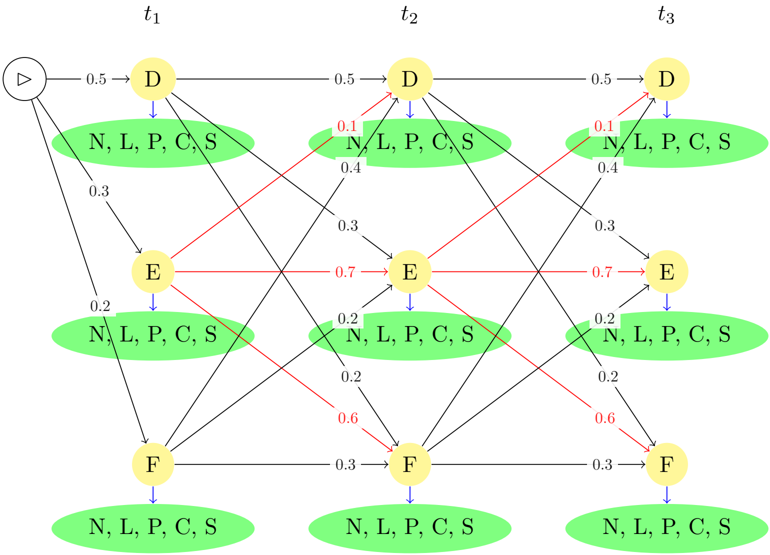
A generative model for POS tagging

- Make use of Bayes' rule:

$$
\begin{aligned}
\operatorname*{argmax}_{\mathbf{T}} P(\mathbf{T}|\mathbf{W}) &= \operatorname*{argmax}_{\mathbf{T}} \frac{P(\mathbf{W}|\mathbf{T})P(\mathbf{T})}{P(\mathbf{W})} && (1) \\
&= \operatorname*{argmax}_{\mathbf{T}} P(\mathbf{W}|\mathbf{T})P(\mathbf{T}) && (2)
\end{aligned}
$$

  Note that we go from Eq. (1) to Eq. (2) because $P(\mathbf{W})$ is not a function of $\mathbf{T}$.

- tag-to-tag *transition* probabilities: $P(\mathbf{T}) = \Pi_{i=1}^{l+1} P_T(t_i|t_{i-1})$: e.g., NN comes after DET
- tag-to-word *emission* probabilities: $P(\mathbf{W}|\mathbf{T}) = \Pi_{i=1}^{l} P_E(w_i|t_i)$: e.g., "natural" is probably a JJ.
- A sequence of words is generated in two phases:
  1. Produce a sequence of tags, e.g., $\triangleright$, NN, DET, ..., based on probability between each two consecutive tags.
  2. For each tag, produce a word, e.g., NN $\rightarrow$ "language", DT $\rightarrow$ "the".
- Yes, a semantically meaningless sentence can be tagged and further parsed correctly, e.g., "NLP is a automobile of GOP".



Three tags/states: D, E, and F. Five words/obserations: N, L, P, C, and S.

A generative model for POS tagging

$\mathbf{W}$	Natural	Language	Processing	is	a	field	of	computer	science.
$\mathbf{T_1}$	JJ	NN	NN	VBZ	DT	NN	IN	NN	NN.

- The generative model we just see is a typical Hidden Markov Model (HMM), where a tag is a *state* and a word is an *observation*.
- In each state/tag, a word/observation emits. After each emission, transit to the next state/tag and emit a word/observation again.
- Why Markovian? The probably of a tag is only conditioned on the previous tag. No further history. First-order Markovian.
- For example, the probability of the first tag sequence: $P(\mathbf{T_1}) = P_T(JJ|\triangleright) \times P_T(NN|JJ) \times P_T(NN|NN) \times \cdots$
- Also, the probability for generating the sentence from the first tag sequence: $P(\mathbf{W}|\mathbf{T_1}) =$ $P_E(\text{“}natural''|JJ) \times P_E(\text{“}language''|NN) \times P_E(\text{“}processing''|NN) \times \cdots$
- The transition and emission probabilities can be obtained by scanning the corpus once.

A generative model for POS tagging

- In principle, we just need to enumerate all possible tag sequences, $\mathbf{T}_1, \mathbf{T}_2, \ldots$ and find the one that yields the largest $P(\mathbf{W}|\mathbf{T})P(\mathbf{T})$.
- But this is costly: If we have N different tags and l words in the sentence, there are N^l possible/hidden tag sequences.
- Smarter way? Viterbi algorithm.

A generative model for POS tagging

- In principle, we just need to enumerate all possible tag sequences, $\mathbf{T}_1, \mathbf{T}_2, \ldots$ and find the one that yields the largest $P(\mathbf{W}|\mathbf{T})P(\mathbf{T})$.
- But this is costly: If we have N different tags and l words in the sentence, there are N^l possible/hidden tag sequences.
- Smarter way? Viterbi algorithm.

A generative model for POS tagging

- In principle, we just need to enumerate all possible tag sequences, $\mathbf{T}_1, \mathbf{T}_2, \ldots$ and find the one that yields the largest $P(\mathbf{W}|\mathbf{T})P(\mathbf{T})$.
- But this is costly: If we have N different tags and l words in the sentence, there are N^l possible/hidden tag sequences.
- Smarter way? Viterbi algorithm.

HMM decoding in Viterbi algorithm

- The problem of estimating the sequence of hidden states given a sequence of observations is known as *decoding* in HMM.
- Basic principle (Lemma 1): $\max(x \cdot y) = \max(x) \cdot y$, if x is a real variable and y is a real constant.
- In each step i (except the start and end), we have N possible states/tags t_i's, each of which can come from N possible t_{i-1}'s.

HMM decoding in Viterbi algorithm

- The problem of estimating the sequence of hidden states given a sequence of observations is known as *decoding* in HMM.
- Basic principle (Lemma 1): $\max(x \cdot y) = \max(x) \cdot y$, if x is a real variable and y is a real constant.
- In each step i (except the start and end), we have N possible states/tags t_i's, each of which can come from N possible t_{i-1}'s.

HMM decoding in Viterbi algorithm

- The problem of estimating the sequence of hidden states given a sequence of observations is known as *decoding* in HMM.
- Basic principle (Lemma 1): $\max(x \cdot y) = \max(x) \cdot y$, if x is a real variable and y is a real constant.
- In each step i (except the start and end), we have N possible states/tags t_i's, each of which can come from N possible t_{i-1}'s.

Viterbi algorithm in math induction

- If the sentence has only one word: $\mathbf{W} = [w_1]$. The best tag t_1 should maximize $P_T(t_1|\triangleright)P_E(w_1|t_1)$ where $\triangleright = t_0$ is the beginning of the sentence.
- If it has two: $\mathbf{W} = [w_1, w_2]$. The best tags t_1 and t_2 shall maximize $\Pi_{i=1}^{2} P_T(t_i|t_{i-1})P_E(w_i|t_i) =$ $P_T(t_1|\triangleright)P_E(w_1|t_1)P_T(t_2|t_1)P_E(w_2|t_2)$. Just search over the N^2 combinations of t_1 and t_2, time complexity $O(N^2)$.
- If it has three: $\mathbf{W} = [w_1, w_2, w_3]$. Given tags t_2 and t_3,

$$\begin{aligned}
&\max P_T(t_1|\triangleright)P_E(w_1|t_1)P_T(t_2|t_1)P_E(w_2|t_2) \quad \overbrace{P_T(t_3|t_2)P_E(w_3|t_3)}^{\text{both constants}} \\
&= [\max P_T(t_1|\triangleright)P_E(w_1|t_1)P_T(t_2|t_1)P_E(w_2|t_2)] \quad P_T(t_3|t_2)P_E(w_3|t_3)
\end{aligned}$$

- No need to check N^3 combinations of tags t_1, t_2 and t_3, many of which will not maximize the final number regardless of the value of $P_T(t_3|t_2)P_E(w_3|t_3)$.

Viterbi algorithm in math induction

- If the sentence has only one word: $\mathbf{W} = [w_1]$. The best tag t_1 should maximize $P_T(t_1|\rhd)P_E(w_1|t_1)$ where $\rhd = t_0$ is the beginning of the sentence.
- If it has two: $\mathbf{W} = [w_1, w_2]$. The best tags t_1 and t_2 shall maximize $\Pi_{i=1}^2 P_T(t_i|t_{i-1})P_E(w_i|t_i)$ = $P_T(t_1|\rhd)P_E(w_1|t_1)P_T(t_2|t_1)P_E(w_2|t_2)$. Just search over the N^2 combinations of t_1 and t_2, time complexity $O(N^2)$.
- If it has three: $\mathbf{W} = [w_1, w_2, w_3]$. Given tags t_2 and t_3,

$$\begin{aligned}
&\max P_T(t_1|\rhd)P_E(w_1|t_1)P_T(t_2|t_1)P_E(w_2|t_2) \quad \overbrace{P_T(t_3|t_2)P_E(w_3|t_3)}^{\text{both constants}} \\
=&[\max P_T(t_1|\rhd)P_E(w_1|t_1)P_T(t_2|t_1)P_E(w_2|t_2)] \quad P_T(t_3|t_2)P_E(w_3|t_3)
\end{aligned}$$

- No need to check N^3 combinations of tags t_1, t_2 and t_3, many of which will not maximize the final number regardless of the value of $P_T(t_3|t_2)P_E(w_3|t_3)$.

Viterbi algorithm in math induction

- If the sentence has only one word: $\mathbf{W} = [w_1]$. The best tag t_1 should maximize $P_T(t_1|\rhd)P_E(w_1|t_1)$ where $\rhd = t_0$ is the beginning of the sentence.
- If it has two: $\mathbf{W} = [w_1, w_2]$. The best tags t_1 and t_2 shall maximize $\Pi_{i=1}^{2} P_T(t_i|t_{i-1})P_E(w_i|t_i)$ = $P_T(t_1|\rhd)P_E(w_1|t_1)P_T(t_2|t_1)P_E(w_2|t_2)$. Just search over the N^2 combinations of t_1 and t_2, time complexity $O(N^2)$.
- If it has three: $\mathbf{W} = [w_1, w_2, w_3]$. <u>Given tags t_2 and t_3</u>,

$$
\begin{aligned}
&\max P_T(t_1|\rhd)P_E(w_1|t_1)P_T(t_2|t_1)P_E(w_2|t_2) \quad \overbrace{P_T(t_3|t_2)P_E(w_3|t_3)}^{\text{both constants}} \\
=&[\max P_T(t_1|\rhd)P_E(w_1|t_1)P_T(t_2|t_1)P_E(w_2|t_2)] \quad P_T(t_3|t_2)P_E(w_3|t_3)
\end{aligned}
$$

- No need to check N^3 combinations of tags t_1, t_2 and t_3, many of which will not maximize the final number regardless of the value of $P_T(t_3|t_2)P_E(w_3|t_3)$.

Viterbi algorithm in math induction

- If the sentence has only one word: $\mathbf{W} = [w_1]$. The best tag t_1 should maximize $P_T(t_1|\rhd)P_E(w_1|t_1)$ where $\rhd = t_0$ is the beginning of the sentence.
- If it has two: $\mathbf{W} = [w_1, w_2]$. The best tags t_1 and t_2 shall maximize $\Pi_{i=1}^{2} P_T(t_i|t_{i-1})P_E(w_i|t_i)$ = $P_T(t_1|\rhd)P_E(w_1|t_1)P_T(t_2|t_1)P_E(w_2|t_2)$. Just search over the N^2 combinations of t_1 and t_2, time complexity $O(N^2)$.
- If it has three: $\mathbf{W} = [w_1, w_2, w_3]$. <u>Given tags t_2 and t_3</u>,

$$\max P_T(t_1|\rhd)P_E(w_1|t_1)P_T(t_2|t_1)P_E(w_2|t_2) \quad \overbrace{P_T(t_3|t_2)P_E(w_3|t_3)}^{\text{both constants}}$$
$$= [\max P_T(t_1|\rhd)P_E(w_1|t_1)P_T(t_2|t_1)P_E(w_2|t_2)] \quad P_T(t_3|t_2)P_E(w_3|t_3)$$

- No need to check N^3 combinations of tags t_1, t_2 and t_3, many of which will not maximize the final number regardless of the value of $P_T(t_3|t_2)P_E(w_3|t_3)$.

HMM decoding in Viterbi algorithm

- Let's generalize:

$$\max \Pi P_T(t_i|t_{i-1})P_E(w_i|t_i) \qquad P_T(t_{i+1}|t_i)P_E(w_{i+1}|t_{i+1})$$
$$= [\max \Pi P_T(t_i|t_{i-1})P_E(w_i|t_i)] \qquad P_T(t_{i+1}|t_i)P_E(w_{i+1}|t_{i+1})$$

- Denote $v(\overbrace{i}^{\text{step or word index}}, \overbrace{j}^{\text{state or tag index}})$ as the probability that the HMM is in state j after seeing the first i observations and passing through **the most probable** preceding sequence of states. We call $v(i,j)$ the *previous Viterbi path probability*.
-

$$v(i,j) = \max_{j=1}^{N} v(i-1,j)P_T(t_i|t_j)P_E(w_i|t_i)$$

- If we repeat this step by step, we can find the maximum $P(\mathbf{T}|\mathbf{W})$.
- Then a traceback allows us to find the tags that maximizes it.

HMM decoding in Viterbi algorithm

- Let's generalize:

$$\begin{aligned} &\max \Pi P_T(t_i|t_{i-1})P_E(w_i|t_i) \qquad P_T(t_{i+1}|t_i)P_E(w_{i+1}|t_{i+1}) \\ =& [\max \Pi P_T(t_i|t_{i-1})P_E(w_i|t_i)] \qquad P_T(t_{i+1}|t_i)P_E(w_{i+1}|t_{i+1}) \end{aligned}$$

- Denote $v(\overbrace{i}^{\text{step or word index}}, \overbrace{j}^{\text{state or tag index}})$ as the probability that the HMM is in state j after seeing the first i observations and passing through **the most probable** preceding sequence of states. We call $v(i,j)$ the *previous Viterbi path probability*.
- $$v(i,j) = \max_{j=1}^{N} v(i-1,j)P_T(t_i|t_j)P_E(w_i|t_i)$$
- If we repeat this step by step, we can find the maximum $P(\mathbf{T}|\mathbf{W})$.
- Then a traceback allows us to find the tags that maximizes it.

HMM decoding in Viterbi algorithm

- Let's generalize:

$$
\begin{aligned}
&\max \Pi P_T(t_i|t_{i-1})P_E(w_i|t_i) \qquad && P_T(t_{i+1}|t_i)P_E(w_{i+1}|t_{i+1}) \\
= &[\max \Pi P_T(t_i|t_{i-1})P_E(w_i|t_i)] \qquad && P_T(t_{i+1}|t_i)P_E(w_{i+1}|t_{i+1})
\end{aligned}
$$

- Denote $v(\overbrace{i}^{\text{step or word index}}, \overbrace{j}^{\text{state or tag index}})$ as the probability that the HMM is in state j after seeing the first i observations and passing through **the most probable** preceding sequence of states. We call $v(i,j)$ the *previous Viterbi path probability*.
- $$v(i,j) = \max_{j=1}^{N} v(i-1,j)P_T(t_i|t_j)P_E(w_i|t_i)$$
- If we repeat this step by step, we can find the maximum $P(\mathbf{T}|\mathbf{W})$.
- Then a traceback allows us to find the tags that maximizes it.

HMM decoding in Viterbi algorithm

- Let's generalize:

$$\begin{aligned} &\max \Pi P_T(t_i|t_{i-1})P_E(w_i|t_i) \qquad P_T(t_{i+1}|t_i)P_E(w_{i+1}|t_{i+1}) \\ = &[\max \Pi P_T(t_i|t_{i-1})P_E(w_i|t_i)] \qquad P_T(t_{i+1}|t_i)P_E(w_{i+1}|t_{i+1}) \end{aligned}$$

- Denote $v(\overbrace{i}^{\text{step or word index}}, \overbrace{j}^{\text{state or tag index}})$ as the probability that the HMM is in state j after seeing the first i observations and passing through **the most probable** preceding sequence of states. We call $v(i,j)$ the *previous Viterbi path probability*.
-
$$v(i,j) = \max_{j=1}^{N} v(i-1,j)P_T(t_i|t_j)P_E(w_i|t_i)$$

- If we repeat this step by step, we can find the maximum $P(\mathbf{T}|\mathbf{W})$.
- Then a traceback allows us to find the tags that maximizes it.

HMM decoding in Viterbi algorithm

- Let's generalize:

$$\max \Pi P_T(t_i|t_{i-1})P_E(w_i|t_i) \qquad P_T(t_{i+1}|t_i)P_E(w_{i+1}|t_{i+1})$$
$$= [\max \Pi P_T(t_i|t_{i-1})P_E(w_i|t_i)] \qquad P_T(t_{i+1}|t_i)P_E(w_{i+1}|t_{i+1})$$

- Denote $v(\overbrace{i}^{\text{step or word index}}, \overbrace{j}^{\text{state or tag index}})$ as the probability that the HMM is in state j after seeing the first i observations and passing through **the most probable** preceding sequence of states. We call $v(i,j)$ the *previous Viterbi path probability*.
- $$v(i,j) = \max_{j=1}^{N} v(i-1,j)P_T(t_i|t_j)P_E(w_i|t_i)$$
- If we repeat this step by step, we can find the maximum $P(\mathbf{T}|\mathbf{W})$.
- Then a traceback allows us to find the tags that maximizes it.

HMM decoding in Viterbi algorithm

- Time complexity: $O(lN^2)$ instead of $O(N^3)$ where N is the number of tags and l is the length of the sentence under POS-tagging.
- For details, read chapter 9.4 of `https://web.stanford.edu/~jurafsky/slp3/9.pdf`.

What do you need?

- A set of N states/tags.
- A set of M obserations/words.
- Transition probabilities, from one state/tag to another, usually as an $N \times N$ matrix
- Emitting probabilities, from one state/tag to an obseration/word, usually as another matrix $N \times M$.
- Initial state/tag probabilities, usually denoted as pi. But this can be easily resolved by introducing a origin state and the transition probabilities from the origin state to all other states.

What do you need?

- A set of N states/tags.
- A set of M observations/words.
- Transition probabilities, from one state/tag to another, usually as an $N \times N$ matrix
- Emitting probabilities, from one state/tag to an observation/word, usually as another matrix $N \times M$.
- Initial state/tag probabilities, usually denoted as pi. But this can be easily resolved by introducing a origin state and the transition probabilities from the origin state to all other states.

What do you need?

- A set of N states/tags.
- A set of M obserations/words.
- Transition probabilities, from one state/tag to another, usually as an $N \times N$ matrix
- Emitting probabilities, from one state/tag to an obseration/word, usually as another matrix $N \times M$.
- Initial state/tag probabilities, usually denoted as pi. But this can be easily resolved by introducing a origin state and the transition probabilities from the origin state to all other states.

What do you need?

- A set of N states/tags.
- A set of M observations/words.
- Transition probabilities, from one state/tag to another, usually as an $N \times N$ matrix
- Emitting probabilities, from one state/tag to an observation/word, usually as another matrix $N \times M$.
- Initial state/tag probabilities, usually denoted as pi. But this can be easily resolved by introducing a origin state and the transition probabilities from the origin state to all other states.

What do you need?

- A set of N states/tags.
- A set of M observations/words.
- Transition probabilities, from one state/tag to another, usually as an $N \times N$ matrix
- Emitting probabilities, from one state/tag to an observation/word, usually as another matrix $N \times M$.
- Initial state/tag probabilities, usually denoted as pi. But this can be easily resolved by introducing a origin state and the transition probabilities from the origin state to all other states.

Computational problems

- What is the problem of multiplying a lenghty list of probabilities?
- Like gradient vanishing, the product becomes very very small.
- Hence, a solution is to logarithmize all probabilities and use summation rather than multiplication.
- See Neubig's slide 8 on HMM. `http://www.phontron.com/slides/nlp-programming-en-04-hmm.pdf`

Computational problems

- What is the problem of multiplying a lenghty list of probabilities?
- Like gradient vanishing, the product becomes very very small.
- Hence, a solution is to logarithmize all probabilities and use summation rather than multiplication.
- See Neubig's slide 8 on HMM. `http://www.phontron.com/slides/nlp-programming-en-04-hmm.pdf`

Computational problems

- What is the problem of multiplying a lenghty list of probabilities?
- Like gradient vanishing, the product becomes very very small.
- Hence, a solution is to logarithmize all probabilities and use summation rather than multiplication.
- See Neubig's slide 8 on HMM. `http://www.phontron.com/slides/nlp-programming-en-04-hmm.pdf`

Computational problems

- What is the problem of multiplying a lenghty list of probabilities?
- Like gradient vanishing, the product becomes very very small.
- Hence, a solution is to logarithmize all probabilities and use summation rather than multiplication.
- See Neubig's slide 8 on HMM. `http://www.phontron.com/slides/nlp-programming-en-04-hmm.pdf`

Conditional Random Fields

- HMM uses Bayes theorem to find the most likely tag sequences $\underset{T}{\operatorname{argmax}} P(\mathbf{T}|\mathbf{W})$.
- CRFs directly estimates it.
- It works by evaluting the chances of each tag sequence over all possible tag sequences in a softmax fashion:

$$P(\mathbf{T}|\mathbf{W}) = \frac{\exp\left(\sum_{k=1}^{K} u_k F_k(\mathbf{W}, \mathbf{T})\right)}{\sum_{\mathcal{T}' \in \mathcal{T}} \exp\left(\sum_{k=1}^{K} u_k F_k(\mathbf{W}, \mathbf{T}')\right)}$$

- Then just need to find $\underset{T \in \mathcal{T}}{\operatorname{argmax}} P(T|W)$.
- u_k is the weight for the k-th feature F_k which is a function of both word sequence and tag sequence.
- Linear-chain CRF: $F_k(\mathcal{W}, \mathcal{T}) = \sum_{i=1}^{l} f_k(w_{i-1}, w_i, \mathcal{T}, i)$ The sum of a function of the word sequence and only the current and previous tags.

Conditional Random Fields

- HMM uses Bayes theorem to find the most likely tag sequences $\operatorname{argmax}_{T} P(\mathbf{T}|\mathbf{W})$.
- CRFs directly estimates it.
- It works by evaluting the chances of each tag sequence over all possible tag sequences in a softmax fashion:

$$P(\mathbf{T}|\mathbf{W}) = \frac{\exp\left(\sum_{k=1}^{K} u_k F_k(\mathbf{W}, \mathbf{T})\right)}{\sum_{\mathcal{T}' \in \mathcal{T}} \exp\left(\sum_{k=1}^{K} u_k F_k(\mathbf{W}, \mathbf{T}')\right)}$$

- Then just need to find $\operatorname{argmax}_{T \in \mathcal{T}} P(T|W)$.
- u_k is the weight for the k-th feature F_k which is a function of both word sequence and tag sequence.
- Linear-chain CRF: $F_k(\mathcal{W}, \mathcal{T}) = \sum_{i=1}^{l} f_k(w_{i-1}, w_i, \mathcal{T}, i)$ The sum of a function of the word sequence and only the current and previous tags.

Conditional Random Fields

- HMM uses Bayes theorem to find the most likely tag sequences $\underset{T}{\operatorname{argmax}} P(\mathbf{T}|\mathbf{W})$.
- CRFs directly estimates it.
- It works by evaluting the chances of each tag sequence over all possible tag sequences in a softmax fashion:

$$P(\mathbf{T}|\mathbf{W}) = \frac{\exp\left(\sum_{k=1}^{K} u_k F_k(\mathbf{W}, \mathbf{T})\right)}{\sum_{\mathcal{T}' \in \mathcal{T}} \exp\left(\sum_{k=1}^{K} u_k F_k(\mathbf{W}, \mathbf{T}')\right)}$$

- Then just need to find $\underset{T \in \mathcal{T}}{\operatorname{argmax}} P(T|W)$.
- u_k is the weight for the k-th feature F_k which is a function of both word sequence and tag sequence.
- Linear-chain CRF: $F_k(\mathcal{W}, \mathcal{T}) = \sum_{i=1}^{l} f_k(w_{i-1}, w_i, \mathcal{T}, i)$ The sum of a function of the word sequence and only the current and previous tags.

Conditional Random Fields

- HMM uses Bayes theorem to find the most likely tag sequences $\underset{T}{\operatorname{argmax}} P(\mathbf{T}|\mathbf{W})$.
- CRFs directly estimates it.
- It works by evaluting the chances of each tag sequence over all possible tag sequences in a softmax fashion:

$$P(\mathbf{T}|\mathbf{W}) = \frac{\exp\left(\sum_{k=1}^{K} u_k F_k(\mathbf{W},\mathbf{T})\right)}{\sum_{\mathcal{T}'\in\mathcal{T}} \exp\left(\sum_{k=1}^{K} u_k F_k(\mathbf{W},\mathbf{T}')\right)}$$

- Then just need to find $\underset{T\in\mathcal{T}}{\operatorname{argmax}} P(T|W)$.
- u_k is the weight for the k-th feature F_k which is a function of both word sequence and tag sequence.
- Linear-chain CRF: $F_k(\mathcal{W},\mathcal{T}) = \sum_{i=1}^{l} f_k(w_{i-1}, w_i, \mathcal{T}, i)$ The sum of a function of the word sequence and only the current and previous tags.

Conditional Random Fields

- HMM uses Bayes theorem to find the most likely tag sequences $\operatorname*{argmax}_T P(\mathbf{T}|\mathbf{W})$.
- CRFs directly estimates it.
- It works by evaluting the chances of each tag sequence over all possible tag sequences in a softmax fashion:

$$P(\mathbf{T}|\mathbf{W}) = \frac{\exp\left(\sum_{k=1}^{K} u_k F_k(\mathbf{W},\mathbf{T})\right)}{\sum_{\mathcal{T}'\in\mathcal{T}} \exp\left(\sum_{k=1}^{K} u_k F_k(\mathbf{W},\mathbf{T}')\right)}$$

- Then just need to find $\operatorname*{argmax}_{T\in\mathcal{T}} P(T|W)$.
- u_k is the weight for the k-th feature F_k which is a function of both word sequence and tag sequence.
- Linear-chain CRF: $F_k(\mathcal{W},\mathcal{T}) = \sum_{i=1}^{l} f_k(w_{i-1}, w_i, \mathcal{T}, i)$ The sum of a function of the word sequence and only the current and previous tags.

Conditional Random Fields

- HMM uses Bayes theorem to find the most likely tag sequences $\underset{T}{\operatorname{argmax}} P(\mathbf{T}|\mathbf{W})$.
- CRFs directly estimates it.
- It works by evaluting the chances of each tag sequence over all possible tag sequences in a softmax fashion:

$$P(\mathbf{T}|\mathbf{W}) = \frac{\exp\left(\sum_{k=1}^{K} u_k F_k(\mathbf{W},\mathbf{T})\right)}{\sum_{\mathcal{T}'\in\mathcal{T}} \exp\left(\sum_{k=1}^{K} u_k F_k(\mathbf{W},\mathbf{T}')\right)}$$

- Then just need to find $\underset{T\in\mathcal{T}}{\operatorname{argmax}} P(T|W)$.
- u_k is the weight for the k-th feature F_k which is a function of both word sequence and tag sequence.
- Linear-chain CRF: $F_k(\mathcal{W},\mathcal{T}) = \sum_{i=1}^{l} f_k(w_{i-1}, w_i, \mathcal{T}, i)$ The sum of a function of the word sequence and only the current and previous tags.

Features of using CRFs in POS tagging

- Manually engineered features
- See §8.5.1 of Jurafsky's book.
- Also see https://towardsdatascience.com/pos-tagging-using-crfs-ea430c5fb78b
- It's now even common to use DL to extract features and then hook up to CRF, e.g., BiLSTM-CRF (NAACL 2016) https://arxiv.org/pdf/1508.01991.pdf
- Sklearn-CRFsuite https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html

Features of using CRFs in POS tagging

- Manually engineered features
- See §8.5.1 of Jurafsky's book.
- Also see https://towardsdatascience.com/pos-tagging-using-crfs-ea430c5fb78b
- It's now even common to use DL to extract features and then hook up to CRF, e.g., BiLSTM-CRF (NAACL 2016) https://arxiv.org/pdf/1508.01991.pdf
- Sklearn-CRFsuite https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html

Features of using CRFs in POS tagging

- Manually engineered features
- See §8.5.1 of Jurafsky's book.
- Also see `https://towardsdatascience.com/pos-tagging-using-crfs-ea430c5fb78b`
- It's now even common to use DL to extract features and then hook up to CRF, e.g., BiLSTM-CRF (NAACL 2016) `https://arxiv.org/pdf/1508.01991.pdf`
- Sklearn-CRFsuite `https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html`

Features of using CRFs in POS tagging

- Manually engineered features
- See §8.5.1 of Jurafsky's book.
- Also see https://towardsdatascience.com/pos-tagging-using-crfs-ea430c5fb78b
- It's now even common to use DL to extract features and then hook up to CRF, e.g., BiLSTM-CRF (NAACL 2016) https://arxiv.org/pdf/1508.01991.pdf
- Sklearn-CRFsuite https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html

Features of using CRFs in POS tagging

- Manually engineered features
- See §8.5.1 of Jurafsky's book.
- Also see https://towardsdatascience.com/pos-tagging-using-crfs-ea430c5fb78b
- It's now even common to use DL to extract features and then hook up to CRF, e.g., BiLSTM-CRF (NAACL 2016) https://arxiv.org/pdf/1508.01991.pdf
- Sklearn-CRFsuite https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html

Named Entity Recognition (NER)

- NEs are proper nouns. It is not rare for them to contain more than one word, e.g., New York City.
- Common categories of NEs: organization, people, location, etc.
- Just like POS tagging, NER can be modeled as a tagging problem.
- Instead of deciding the POS tags, we decide a different kind of tags.
- A common type of tags used in NER is BIO: begin, inside, and outside. See `https://medium.com/analytics-vidhya/bio-tagged-text-to-original-text-99b05da6664`
- See also `https://github.com/scofield7419/sequence-labeling-BiLSTM-CRF`
- It seems that CRF is more widely used in NER than in POS tagging.

Named Entity Recognition (NER)

- NEs are proper nouns. It is not rare for them to contain more than one word, e.g., New York City.
- Common categories of NEs: organization, people, location, etc.
- Just like POS tagging, NER can be modeled as a tagging problem.
- Instead of deciding the POS tags, we decide a different kind of tags.
- A common type of tags used in NER is BIO: begin, inside, and outside. See https://medium.com/analytics-vidhya/bio-tagged-text-to-original-text-99b05da6664
- See also https://github.com/scofield7419/sequence-labeling-BiLSTM-CRF
- It seems that CRF is more widely used in NER than in POS tagging.

Named Entity Recognition (NER)

- NEs are proper nouns. It is not rare for them to contain more than one word, e.g., New York City.
- Common categories of NEs: organization, people, location, etc.
- Just like POS tagging, NER can be modeled as a tagging problem.
- Instead of deciding the POS tags, we decide a different kind of tags.
- A common type of tags used in NER is BIO: begin, inside, and outside. See `https://medium.com/analytics-vidhya/bio-tagged-text-to-original-text-99b05da6664`
- See also `https://github.com/scofield7419/sequence-labeling-BiLSTM-CRF`
- It seems that CRF is more widely used in NER than in POS tagging.

Named Entity Recognition (NER)

- NEs are proper nouns. It is not rare for them to contain more than one word, e.g., New York City.
- Common categories of NEs: organization, people, location, etc.
- Just like POS tagging, NER can be modeled as a tagging problem.
- Instead of deciding the POS tags, we decide a different kind of tags.
- A common type of tags used in NER is BIO: begin, inside, and outside. See `https://medium.com/analytics-vidhya/bio-tagged-text-to-original-text-99b05da6664`
- See also `https://github.com/scofield7419/sequence-labeling-BiLSTM-CRF`
- It seems that CRF is more widely used in NER than in POS tagging.

Named Entity Recognition (NER)

- NEs are proper nouns. It is not rare for them to contain more than one word, e.g., New York City.
- Common categories of NEs: organization, people, location, etc.
- Just like POS tagging, NER can be modeled as a tagging problem.
- Instead of deciding the POS tags, we decide a different kind of tags.
- A common type of tags used in NER is BIO: begin, inside, and outside. See https://medium.com/analytics-vidhya/bio-tagged-text-to-original-text-99b05da6664
- See also https://github.com/scofield7419/sequence-labeling-BiLSTM-CRF
- It seems that CRF is more widely used in NER than in POS tagging.

Named Entity Recognition (NER)

- NEs are proper nouns. It is not rare for them to contain more than one word, e.g., New York City.
- Common categories of NEs: organization, people, location, etc.
- Just like POS tagging, NER can be modeled as a tagging problem.
- Instead of deciding the POS tags, we decide a different kind of tags.
- A common type of tags used in NER is BIO: begin, inside, and outside. See `https://medium.com/analytics-vidhya/bio-tagged-text-to-original-text-99b05da6664`
- See also `https://github.com/scofield7419/sequence-labeling-BiLSTM-CRF`
- It seems that CRF is more widely used in NER than in POS tagging.

Named Entity Recognition (NER)

- NEs are proper nouns. It is not rare for them to contain more than one word, e.g., New York City.
- Common categories of NEs: organization, people, location, etc.
- Just like POS tagging, NER can be modeled as a tagging problem.
- Instead of deciding the POS tags, we decide a different kind of tags.
- A common type of tags used in NER is BIO: begin, inside, and outside. See https://medium.com/analytics-vidhya/bio-tagged-text-to-original-text-99b05da6664
- See also https://github.com/scofield7419/sequence-labeling-BiLSTM-CRF
- It seems that CRF is more widely used in NER than in POS tagging.

Modern ways?

Neural network-based generative models, e.g., seq2seq.