Natural Language Processing

Lecture 5
Text classification and sequence tagging with classical methods

2021

Text classification

Text classification tasks

A text classification task is to assign the appropriate label from a given $C = \{c_1, \ldots, c_n\}$ set of class/category labels to a d text/document.

Representative examples include

- **Sentiment analysis**: classify according to the sentiment expressed by the document. Label set examples:
  - $\{$positive, negative$\}$,
  - $\{$positive, negative, ambigous$\}$,
  - $\{$admiration, amusement, annoyance, approval, $\ldots$, sadness, surprise$\}$.[1]

[1] The example is based on the 27 emotions of the GoEmotions dataset.

Text classification tasks cont.

- **SPAM detection**: binary classification to decide whether a message is unsolicited.
- **Authorship detection**: who wrote a text from a specified set of authors.
- **Author characteristics detection**: was the author male or female, what was their age etc.
- **Subject/topic detection**: to which subject/topic a document belongs to in a predefined list, e.g., in the Library of Congress Classification system $\{$medicine, agriculture, science, fine arts, $\dots\}$
- **Genre detection**: determine the genre of a text, e.g., assign a label from the set $\{$scifi, adventure, love story, mystery, historical, western$\}$.

Methods

- **Manually designed rule-based systems**: e.g., using carefully designed lists of words positively or negatively correlated with the classes.
  These systems can reach good performance, but require a lot of manual work and are difficult to maintain and adapt.
- **Machine learning methods**: Models learnt on a supervised data set containing labeled documents: $\{\langle d_i, c_i\rangle\}_{i\in\{1,\ldots,N\}}$.
  Methods range from linear machine learning methods such as logistic regression to deep neural networks.

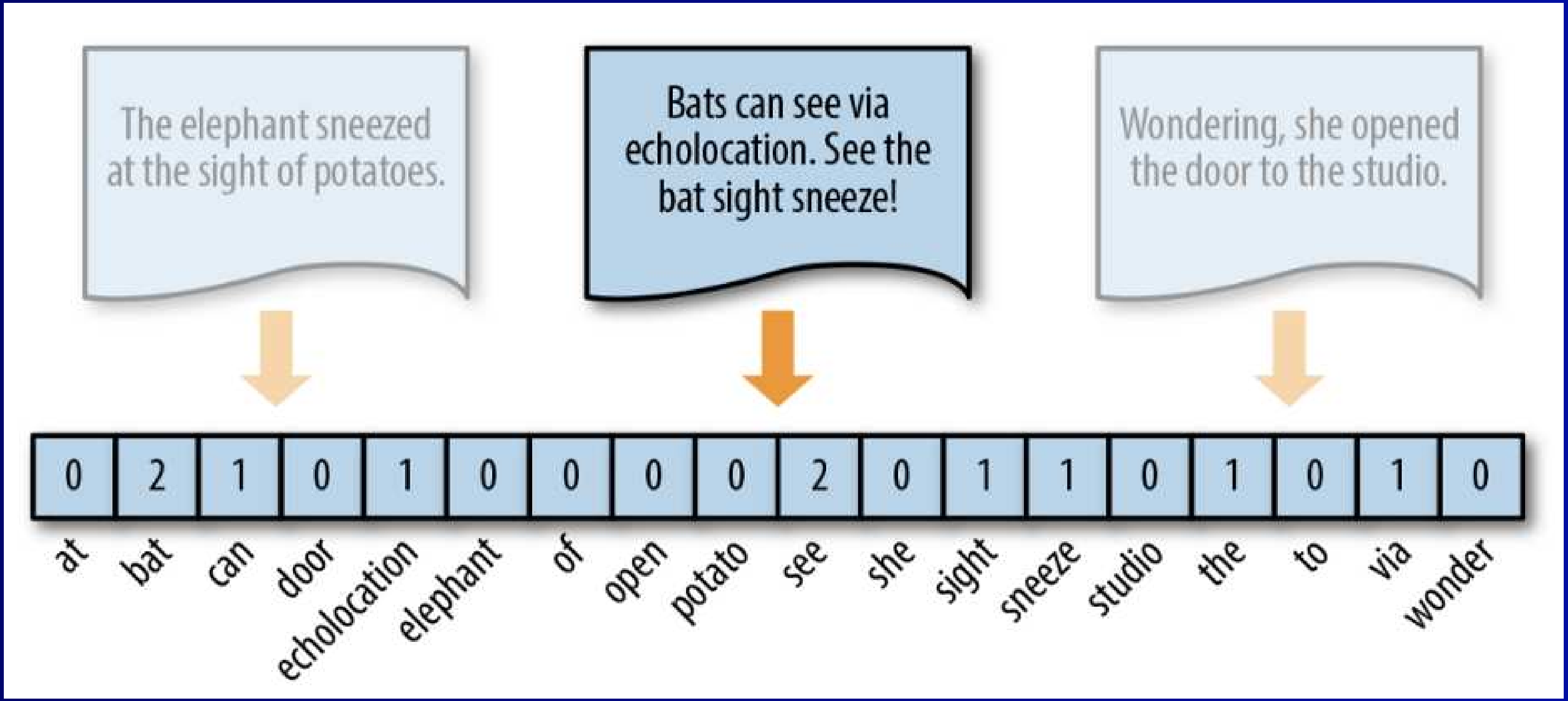
Bag of words (BOW) representation

Many machine learning-based classification methods require their input to be represented as fixed-length numerical vectors. For texts with varying lengths, a common approach is use a bag of words representation:

- Tokenize the input texts using a $V = \{w_1, \ldots, w_N\}$ vocabulary of word types, and
- represent them as $|V| = N$-dimensional vectors of word counts, i.e., for a d document $BOW_V(d) = \langle c_{1,d}, \ldots, c_{N,d} \rangle$, where each $c_{i,d}$ is the count of w_i in d.

Bag of words representation cont.

A simple example:


The elephant sneezed at the sight of potatoes.
Bats can see via echolocation. See the bat sight sneeze!
Wondering, she opened the door to the studio.
0 2 1 0 1 0 0 0 0 2 0 1 1 0 1 0 1 0
at bat can door echolocation elephant of open potato see she sight sneeze studio the to via wonder

(Figure from From Word Embeddings to Pretrained Language Models.)

Bag of words refinements

The basic BOW representation can be refined in several ways, perhaps the three most important are

- omitting *stopword* (non-informative word) counts from the BOW vectors. What counts as stopword is task and domain dependent, but it is common to consider (some) function words, e.g. determiners to be stopwords;
- adding some word sequence counts to the BOW representations, e.g., *bigram or trigram counts*;
- weight words according to their informativity: the most widespread method is weighting according to *term frequency* and *inverse document frequency (TF-IDF)*.

TF-IDF schemes

The basic assumption of TF-IDF weighting schemes is that words that occur in a large fraction of the training documents are not so informative as words occurring only in a few. TF-IDF vectors, accordingly discount word counts (“term frequencies”) by document frequencies. A simple but widespread variant:

$$TF\text{-}IDF(d) = \langle tf_{1,d} \cdot idf_1, \ldots, tf_{N,d} \cdot idf_N \rangle$$

where $tf_{i,d}$ is simply the count of w_i in d, while

$$idf_i = \log \frac{\#\mathrm{of\ \ all\ \ documents}}{\#\mathrm{of\ \ documents\ \ containing}\ \ w_i}$$

Binary bag of words

An interesting simplification of the BOW representation is to indicate only the presence or absence of words:

$$BOW_{bin}(d) = \mathrm{sign}(BOW(d))$$

where the application of the sign function is elementwise, i.e.,

$$BOW_{bin}(d) = \langle \mathrm{sign}(c_{1,d}), \dots, \mathrm{sign}(c_{N,d}) \rangle.$$

It turned out that these simpler and less memory consuming representations can be used instead of normal BOW vectors in many settings without noticeable performance difference.

Naive Bayes classifier with BOW

In its simplest form, the Naive Bayes (NB) classifier is a generative model modeling the joint distribution of $\mathbf{x}$ observation feature vectors and their c class labels as

$$P(\mathbf{x}, c) = P(c) \prod_{i=1}^{D} P(x_i \mid c).$$

The model is "naive", because it is based on the *conditional independence assumption* that given the class label, all observed features are independent of each other.

Naive Bayes classifier with BOW cont.

NB models can be precisely described by specifying

- the class label categorical distribution $P(c)$, and the
- the $P(x_i \mid c_j)$ distributions for each x_i observation feature and c_j label.

$P(c)$ is always a categorical (Bernoulli or “multinoulli”) distribution, while the choice of $P(x_i \mid c_j)$ distributions depends on the type of x_i; for continuous x_i-s it can be any continuous distribution, Gaussian being a common choice.

Naive Bayes classifier with BOW cont.

The NB model can be adapted to text classification by applying the NB assumption to individual tokens: each token is assumed to be chosen independently from others according to a categorical conditional distribution $P(w \mid c)$. If $\mathbf{x}$ is a BOW vector and c is a class label this means

$$P(\mathbf{x}, c) = P(c) \prod_{i=1}^{|V|} P(w_i \mid c)^{x_i}.$$

Taking the logarithm of both sides for numerical stability reasons:

$$\log P(\mathbf{x}, c) = \log P(c) + \sum^{|V|} x_i \log P(w_i \mid c).$$

Naive Bayes classifier with BOW cont.

This means that given an $\mathbf{x}$ BOW vector and a vector

$$\theta_c = \langle \log P(w_1 \mid c), \dots, \log P(w_{|V|} \mid c) \rangle$$

of conditional log probabilities of words for a c class,

$$\log P(\mathbf{x}, c) = \log P(c) + \theta_c \cdot \mathbf{x},$$

i.e., the log probability of $(\mathbf{x}, c)$ is a simple linear function for each c_i. Prediction of the most likely class for a d document is also very simple:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}(\log P(c) + \theta_c \cdot BOW(d))$$

Naive Bayes classifier with BOW cont.

The MLE of the model parameters can be based on simple counts:

$$P(c) \approx \frac{\#\text{of } \ c \ \text{ documents}}{\#\text{of } \ \text{all } \ \text{documents}},$$

$$P(w \mid c) \approx \frac{\#w \ \text{ occurrences } \ \text{in } \ c \ \text{ documents}}{\#of \ \text{ words } \ \text{in } \ c \ \text{ documents}}.$$

As we are basically working with per-class (unigram) language models, data sparsity presents problems again.

Naive Bayes classifier with BOW cont.

Most radically, if a word $w \in V$ does not occur in any c-class documents then the corpus-based MLE for $P(w \mid c) = 0$ and, therefore, for any document with $\mathbf{x}$ BOW vector containing a non-zero count for w

$$P(\mathbf{x}, c) = P(c) \prod_{i=1}^{|V|} P(w_i \mid c)^{x_i} = 0,$$

regardless of any other word they contain.
The solution is, again, using appropriate smoothing methods, e.g., add-one smoothing.

Naive Bayes limitations

Although BOW-based NB models are fairly simple to estimate and use for prediction, and can perform acceptably, there are some negatives:

- The NB conditional independence assumption is rather unrealistic and leads to misleading probability predictions with basic BOW;
- the NB assumption is absurd for and makes it impossible to use refined BOW variants such as N-gram based ones;
- using a full generative model for a discriminative task typically has some performance penalties.

Discriminative linear methods

The most important alternative within the domain of classical learning algorithms is to use one of the well known *discriminative methods* with BOW vectors:

- a perceptron variant,
- logistic regression, or
- SVM.

These models do not assume conditional independence and therefore have no problem with using refined (e.g. N-gram based) BOW representations as input.

Sequence tagging

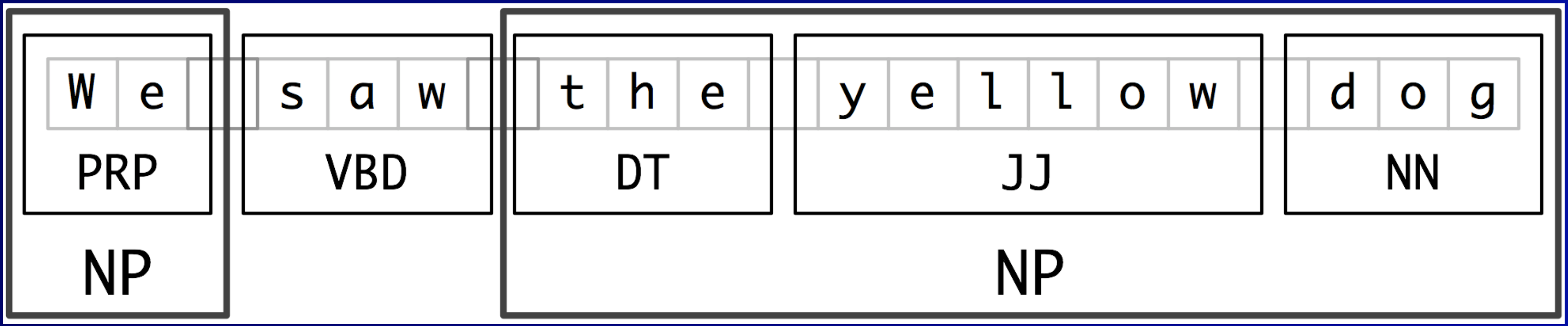
Sequence tagging in NLP

The sequence tagging task in general is to tag each element of a variable length input sequence with one of the labels in a given finite T tag set. In NLP, the input sequence is typically a $\langle w_1, \dots, w_n\rangle$ sequence of *tokens*.

Some tasks in the traditional NLP pipeline are explicitly sequence tagging tasks, e.g. POS-tagging and morphological tagging. Others, like NP-chunking, NER or keyword identification can be transformed into a sequence tagging task with a simple trick.

IOB tagging

These tasks are ostensibly span-finding and span-tagging tasks: the goal is to find token spans belonging the certain categories.
E.g., in the case of (minimal) noun phrase (NP) chunking:



(Figure from the NLTK book.)

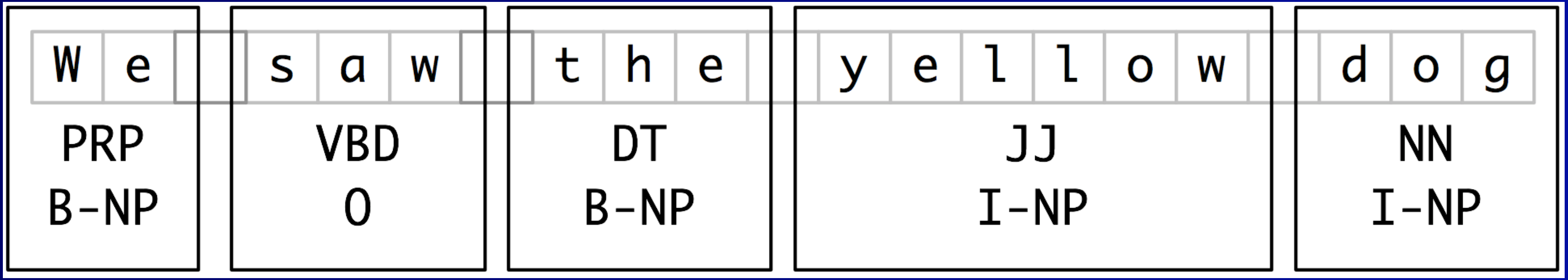
IOB tagging cont.

The IOB trick is to reformulate the span identification/tagging task as a sequence tagging task. If there are $T_1, \ldots, T_N$ span types to be identified, then we introduce three types of token-level tags:

- a B (beginning) tag for all span types: $B - T_1, \ldots, B - T_N$ indicating the first token of a span of a given type;
- an I (inside) tag for all span types $I - T_1, \ldots, I - T_N$; indicating that a token is inside (as second or later element) a span, and, finally,
- a unique O tag for tokens that do not belong to any span type to be found.

IOB tagging cont.

Using these tags the span identification task becomes a sequence tagging task.



(Figure from the NLTK book.)

Although the IOB scheme is the most well known, others are in use as well, e.g., if one introduces $E - T_i$ *end* tags, then a unique I *inside* tag is enough for unambiguous tagging.

Sequence tagging challenges

The main challenge of sequence tagging is the complicated interdependence between the tag of an element and the features of the other elements *including their tags*: in the case of most NLP tagging tasks, the tags are *strongly context dependent*.

Another important problem is feature engineering: which features of the sequence elements are relevant for tagging? If out of vocabulary words are to be handled properly then at least some of the features should probably be based on the word's surface form, e.g. on its capitalization, suffix etc.

Supervised methods for sequence tagging

These methods assume that a

$$D = \{\langle \mathbf{x_1}, \mathbf{y_1} \rangle, \dots, \langle \mathbf{x_N}, \mathbf{y_N} \rangle\}$$

supervised dataset is available, in which each $\langle \mathbf{x}_i, \mathbf{y}_i \rangle$ pair consists of an $\langle x^i_1, \dots, x^i_{n_i} \rangle$ sequence to be tagged and the corresponding $\langle y^i_1, \dots, y^i_{n_i} \rangle$ correct tag sequence.

The methods we will discuss are all *probabilistic*: they model either the $P(\mathbf{X}, \mathbf{Y})$ joint distribution (generative model) or the $P(\mathbf{Y} \mid \mathbf{X})$ conditional distribution (discriminative model).

Hidden Markov models (HMMs)

HMMs are *generative models* of the $P(\mathbf{X}, \mathbf{Y})$ distribution based on the assumption that the elements of the *observable* $\mathbf{x}$ sequences actually depend on the positionally corresponding *hidden* elements of the $\mathbf{y}$ sequences, which, in turn, are distributed according to a Markov model. The conditional independence assumptions collectively follow this graphical model:

$$\begin{tikzcd}
Y_1 \arrow[r] \arrow[d] & Y_2 \arrow[r] \arrow[d] & Y_3 \arrow[r] \arrow[d] & \cdots \arrow[r] & Y_n \arrow[d] \\
X_1 & X_2 & X_3 & \ldots & X_n
\end{tikzcd}$$

Hidden Markov models cont.

Because of the Markov model assumption regarding the Ys, there is an A matrix specifying all tag transition probabilities, so that for any appropriate k, i, j,

$$P(Y_k = y_j \mid Y_{k-1} = y_i) = a_{ij}.$$

HMMs also assume that the $P(X \mid Y)$ “emission” probabilities are position-independent: there is also a B matrix so that for any k, i, j,

$$P(X_k = x_j \mid Y_k = y_i) = b_{ij}.$$

Hidden Markov models cont.

Assuming, finally, an Π vector containing the start probabilities for each possible y_i tag:

$$P(Y_1 = y_i) = \pi_i,$$

the probability of a concrete $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \langle x_{l_1}, \ldots, x_{l_n} \rangle, \langle y_{m_1}, \ldots, y_{m_n} \rangle \rangle$ pair can be calculated as

$$P(\mathbf{x}, \mathbf{y}) = \pi_{m_1} b_{m_1 l_1} \prod_{i=2}^{n} a_{m_{i-1} m_i} b_{m_i l_i}.$$

Hidden Markov models cont.

The MLE of the probabilities in A, B and Π can be calculated simply by counting. If the training dataset contains N sequences then

$$\pi_i = \frac{C(\text{first element is } y_i)}{N}$$

$$a_{ij} = \frac{C(\langle y_i, y_j \rangle)}{\sum_k C(\langle y_i, y_k \rangle)}$$

$$b_{ij} = \frac{C(y_i \text{ emits } x_j)}{\sum_k C(y_i)}$$

Similarly to other counting based MLE methods, smoothing might be necessary in case of sparse data.

The Viterbi algorithm

Given a trained HMM with π, A, B parameters, and an $\mathbf{x}$ input sequence of length n, we want to determine the most probable corresponding $\mathbf{y}$ sequence of tags, i.e. find

$$\operatorname*{argmax}_{\mathbf{y}\in Y^n} P(\mathbf{y} \mid \mathbf{x}, \Pi, A, B),$$

which equals

$$\operatorname*{argmax}_{\mathbf{y}\in Y^n} P(\mathbf{x}, \mathbf{y} \mid \Pi, A, B).$$

An exhaustive search is unfeasible because there are $|Y|^n$ alternative tag sequences.

The Viterbi algorithm cont.

Fortunately, the HMM conditional independence assumptions have the following consequence: If we know, for all $y_i \in Y$, the values

$$\mathbf{y}_i^{n-1} = \operatorname*{argmax}_{\mathbf{y} \in Y^{n-1} \,\wedge\, \mathbf{y}[n-1]=y_i} P(\mathbf{x}[1:n-1], \mathbf{y} \mid \Pi, A, B),$$

then the most probable $\mathbf{y}$ can be computed simply as

$$\operatorname*{argmax}_{\mathbf{y} \in \{\langle \mathbf{y}_i^{n-1}, y \rangle \,\mid\, i \in 1 \ldots |Y| \,\wedge\, y \in Y\}} P(\mathbf{x}, \mathbf{y} \mid \Pi, A, B),$$

i.e., we need to search only among the $|Y|^2$ possible continuations of the $\mathbf{y}_i^{n-1}$ sequences.

The Viterbi algorithm cont.

This suggests the following algorithm (named after Andrew Viterbi, who published a variant in 1967):

```
1: for all $i \in 1 \ldots |Y|$ do
2:     $\mathbf{y}_i^1 \leftarrow \langle y_i \rangle$
3: end for
4: for all $t \in 2 \ldots n-1$ do
5:     for all $i \in 1 \ldots |Y|$ do
6:         $\mathbf{y}_i^t \leftarrow \operatorname{argmax}_{\mathbf{y} \in \{\langle \mathbf{y}_k^{t-1}, y_i \rangle \mid k \in 1 \ldots |Y|\}} P(\mathbf{x}[1:t], \mathbf{y} \mid \Pi, A, B)$
7:     end for
8: end for
9: return $\operatorname{argmax}_{\mathbf{y} \in \{\langle \mathbf{y}_i^{n-1}, y \rangle \mid i \in 1 \ldots |Y| \wedge y \in Y\}} P(\mathbf{x}, \mathbf{y} \mid \Pi, A, B)$
```

The Viterbi algorithm cont.

In stark contrast to an exhaustive search, the standard Viterbi algorithm has a time complexity of $\mathcal{O}(\mathrm{length}(\mathbf{x})|Y|^2)$, and tracking the partial $\mathbf{y}_i^t$ sequence elements and their probabilities can be done using back-pointers within $\mathcal{O}(\mathrm{length}(\mathbf{x})|Y|)$ space.

Calculating the probabilities to be compared directly requires multiplying numbers very close to zero, so it is customary to work with log probabilities (and addition) instead.

Discriminative methods

Similarly to the Naive Bayes sequence classifier, HMMs are generative models, modeling probabilities of the input as well as the labels, which is unnecessary in our setting. We can construct similarly structured but *discriminative* models by “reversing the arrows” between input and labels and conditionalizing on $\mathbf{X}$:

$$\begin{tikzcd}
Y_1 \arrow[r] & Y_2 \arrow[r] & Y_3 \arrow[r] & \cdots \arrow[r] & Y_n \\
 & & \mathbf{X} \arrow[ull] \arrow[ul] \arrow[u] \arrow[ur] \arrow[urr] & &
\end{tikzcd}$$

Maximum entropy Markov models (MEMMs)

According to the previous graphical model's assumptions,

$$P(\mathbf{Y} \mid \mathbf{X}) = P(Y_1 \mid \mathbf{X}) \prod_{m=2}^{n} P(Y_m | Y_{m-1}, \mathbf{X}).$$

MEMMs, in turn, model the individual $P(Y_m|Y_{m-1}, \mathbf{X})$ probabilities analogously to logistic regression as

$$P(Y_m = y_i | Y_{m-1} = y_k, \mathbf{x}) = \frac{\exp(\mathbf{w}_i \cdot \mathbf{f}(y_k, \mathbf{x}, m))}{\sum_{j=1}^{|Y|} \exp(\mathbf{w}_j \cdot \mathbf{f}(y_k, \mathbf{x}, m))},$$

where $\mathbf{f}(\cdot)$ is a feature vector producing function, and each $\mathbf{w}_i$ is a weight vector for the $y_i \in Y$ label.

Feature functions

MEMM feature functions were traditionally designed by field experts. In NLP it is customary to condition only on *local features* of the element to be tagged and consider only a *context window*. E.g., for POS tagging, $\mathbf{f}(y_{m-1}, \mathbf{x}, m)$ would include the features:

- Elements in a context window around x_m, e.g. $\langle x_{m-1}, x_m, x_{m+1}\rangle$,
- suffixes (of a fixed length) of the context window's elements,
- prefixes (of a fixed length) of the context window's elements,
- capitalization information of the context window's elements.

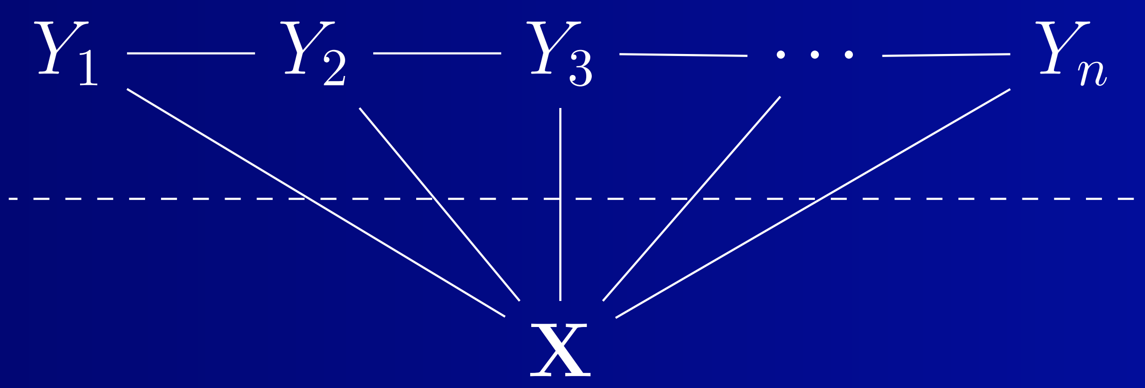
Conditional Random Fields (CRFs)

Although MEMMs are more flexible than HMMs (e.g., tags can depend on other features of the context than the previous tag), they also have important limitations.

Perhaps the most important is that the label probabilities are *locally normalized*: $\sum_{y \in Y} P(y \,|y_{m-1}, \mathbf{x}, m) = 1$ independently of how “familiar” a context is to the model, and, therefore, the model cannot express a general low confidence about the labels in a given context. This leads to the so-called *label bias* problem: the model prefers labels with low entropy next label distributions, even if they have low confidence. Linear-chain CRF models were designed to address these problems.

Conditional Random Fields cont.

Linear chain CRFs are discriminative models assuming the following *undirected* structure:



According to these assumptions,

$$P(\mathbf{Y} \mid \mathbf{X}) = \frac{1}{Z(\mathbf{X})} \prod_{m=1}^{n-1} \phi_m(Y_m, Y_{m+1}, \mathbf{X}).$$

Conditional Random Fields cont.

Somewhat similarly to MEMMs, the $\phi_m(\cdot)$ potential functions are modeled linearly using a feature function and a corresponding weight vector:

$$\phi_m(y_m, y_{m+1}, \mathbf{x}) = \exp(\mathbf{w} \cdot \mathbf{f}(y_m, y_{m+1}, \mathbf{x}, m)).$$

The crucial difference compared to MEMMs is that the normalization is *global*:

$$P(\mathbf{y} \,|\, \mathbf{x}) = \frac{\exp(\sum_{m=1}^{n-1} \mathbf{w} \cdot \mathbf{f}(y_m, y_{m+1}, \mathbf{x}, m))}{\sum_{\mathbf{y}' \in Y^n} \exp(\sum_{m=1}^{n-1} \mathbf{w} \cdot \mathbf{f}(y'_m, y'_{m+1}, \mathbf{x}, m))}.$$

Optimization and inference

Both MEMMs and linear chain CRFs can be optimized using standard convex optimization techniques, e.g., gradient descent, and, having trained a model, the most likely tag sequence for a given input can be efficiently found by using variants of the Viterbi algorithm.