



David S. Batista (/)

[About \(/about-me/\)](/about-me/)[Blog Posts \(/posts/\)](/posts/)[Datasets \(/nlp_datasets/\)](/nlp_datasets/)[Publications \(/publications/\)](/publications/)[Resources \(/nlp_resources/\)](/nlp_resources/)[CV \(/assets](/assets)[/documents/dsbatista-cv.en.pdf\)](/documents/dsbatista-cv.en.pdf)

Hidden Markov Model and Naive Bayes relationship

📅 2017-11-11 🔖 [hidden-markov-models \(/tag/hidden-markov-models\)](/tag/hidden-markov-models) [naive-bayes \(/tag/naive-bayes\)](/tag/naive-bayes) [sequence-prediction \(/tag/sequence-prediction\)](/tag/sequence-prediction) [viterbi \(/tag/viterbi\)](/tag/viterbi)

This is the first post, of a series of posts, about sequential supervised learning applied to Natural Language Processing. In this first post I will write about the classical algorithm for sequence learning, the Hidden Markov Model (HMM), explain how it's related with the Naive Bayes Model and **it's** limitations.

You can find the second and third posts here:

- [Maximum Entropy Markov Models and Logistic Regression \(/blog/2017/11/12/Maximum_Entropy_Markov_Model/\)](/blog/2017/11/12/Maximum_Entropy_Markov_Model/)
- [Conditional Random Fields for Sequence Prediction \(/blog/2017/11/13/Conditional_Random_Fields/\)](/blog/2017/11/13/Conditional_Random_Fields/)

Introduction

The classical problem in Machine Learning is to learn a classifier that can distinguish between two or more classes, i.e., that can accurately predict a class for a new object given training examples of objects already classified.

NLP typical examples are, for instance: classifying an email as spam or not spam, classifying a movie into genres, classifying a news article into topics, etc., however, there is another type of prediction problems which involve structure.

A classical example in NLP is part-of-speech tagging, in this scenario, each x_i describes a word and each y_i the associated part-of-speech of the word x_i (e.g.: *noun*, *verb*, *adjective*, etc.).

Another example, is named-entity recognition, in which, again, each x_i describes a word and y_i is a semantic label associated to that word (e.g.: *person*, *location*, *organization*, *event*, etc.).

In both examples **the data consist of sequences of (x, y) pairs**, and we want to model our learning problem based on that sequence:



$$p(y_1, y_2, \dots, y_m \mid x_1, x_2, \dots, x_m)$$

in most problems these sequences can have a sequential correlation. That is, nearby x and y values are likely to be related to each other. For instance, in English, it's common after the word *to* to have a word whose part-of-speech tag is a *verb*.

Note that there are other machine learning problems which also involve sequences but are clearly different. For instance, in time-series, there is also a sequence, but we want to predict a value y at point $t + 1$, and

we can use all the previous true observed y to predict. In sequential supervised learning we must predict all y values in the sequence.

The Hidden Markov Model (HMM) was one the first proposed algorithms to **classify sequences**. There are other sequence models, but I will start by explaining the HMM as a sequential extension to the Naive Bayes model.

Naive Bayes classifier

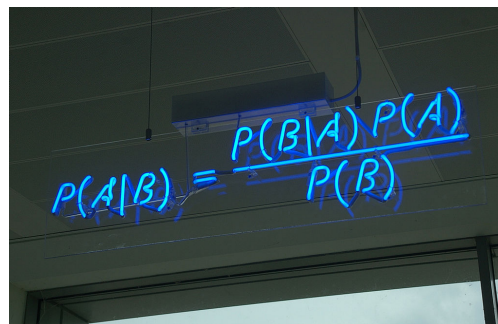
The Naive Bayes (NB) classifier is a **generative model**, which builds a model of each possible class based on the training examples for each class. Then, in prediction, given an observation, it computes the predictions for all classes and returns the class most likely to have generated the observation. That is, it tries to predict which class generated the new observed example.

In contrast **discriminative models**, like logistic regression, tries to learn which features from the training examples are most useful to discriminate between the different possible classes.

The Naive Bayes classifier returns the class that as the maximum posterior probability given the features:

$$\hat{y} = \arg \max_y p(y | \vec{x})$$

where y it's a class and \vec{x} is a feature vector associated to an observation.



Bayes theorem in blue neon.
(taken from Wikipedia)

The NB classifier is based on the Bayes' theorem. Applying the theorem to the equation above, we get:

$$p(y | \vec{x}) = \frac{p(y) \cdot p(\vec{x} | y)}{p(\vec{x})}$$

In training, when iterating over all classes, for a given observation, and calculating the probabilities above, the probability of the observation, i.e., the denominator, is always the same, it has no influence, so we can then simplify the formula:

$$p(y | \vec{x}) = p(y) \cdot p(\vec{x} | y)$$

which, if we decompose the vector of features, is the same as:

$$p(y | \vec{x}) = p(y) \cdot p(x_1, x_2, x_3, \dots, x_1 | y)$$

this is hard to compute, because it involves estimating every possible combination of features. We can relaxed this computation by applying the Naives Bayes assumption, which states that:

"each feature is conditional independent of every other feature, given the class"

formerly, $p(x_i | y, x_j) = p(x_i | y)$ with $i \neq j$. The probabilities $p(x_i | y)$ are independent given the

class y and hence can be 'naively' multiplied:

$$p(x_1, x_2, \dots, x_m \mid y) = p(x_1 \mid y) \cdot p(x_2 \mid y) \cdot \dots \cdot p(x_m \mid y)$$

plugging this into our equation:

$$p(y \mid \vec{x}) = p(y) \prod_{i=1}^m p(x_i \mid y)$$

we get the final Naive Bayes model, which as consequence of the assumption above, **doesn't capture dependencies between each input variables in \vec{x}** .

Training

Training in Naive Bayes is mainly done by **counting features and classes**. Note that the procedure described below needs to be done for every class y_i .

To calculate the prior, we simple count how many samples in the training data fall into each class y_i divided by the total number of samples:

$$p(y_i) = \frac{N_{y_i}}{N}$$

To calculate the likelihood estimate, we count the number of times feature w_i appears among all features in all samples of class y_i :

$$p(x_i \mid y_i) = \frac{\text{count}(x_i, y_i)}{\sum_{x_i \in X} \text{count}(x_i, y_i)}$$

This will result in a big table of occurrences of features for all classes in the training data.

Classification

When given a new sample to classify, and assuming that it contains features x_1, x_3, x_5 , we need to compute, for each class y_i :

$$p(y_i \mid x_1, x_3, x_5)$$

This is decomposed into:

$$p(y_i \mid x_1, x_3, x_5) = p(y_i) \cdot p(y_i \mid x_1) \cdot p(y_i \mid x_3) \cdot p(y_i \mid x_5)$$

Again, this is calculated for each class y_i , and we assign to the new observed sample the class that has the highest score.

From Naive Bayes to Hidden Markov Models

The model presented before predicts a class for a set of features associated to an observation. To predict a class sequence $y = (y_1, \dots, y_n)$ for sequence of observation $x = (x_1, \dots, x_n)$, a simple sequence model can be formulated as a product over single Naïve Bayes models:

$$p(\vec{y} \mid \vec{x}) = \prod_{i=1}^n p(y_i) \cdot p(x_i \mid y_i)$$

Two aspects about this model:

- there is only one feature at each sequence position, namely the identity of the respective observation due the assumption that each feature is generated independently, conditioned on the class y_i .
- it doesn't capture interactions between the observable variables x_i .

It is however reasonable to assume that there are dependencies at consecutive sequence positions y_i , remember the example above about the part-of-speech tags ?

This is where the First-order Hidden Markov Model appears, introducing the **Markov Assumption**:

"the probability of a particular state is dependent only on the previous state"

$$p(\vec{y} | \vec{x}) = \prod_{i=1}^n p(y_i | y_{i-1}) \cdot p(x_i | y_i)$$

which written in it's more general form:



$$p(\vec{x}) = \sum_{y \in Y} \prod_{i=1}^n p(y_i | y_{i-1}) \cdot p(x_i | y_i)$$

where Y represents the set of all possible label sequences \vec{y} .

Hidden Markov Model

A Hidden Markov Model (HMM) is a sequence classifier. As other machine learning algorithms it can be trained, i.e.: given labeled sequences of observations, and then using the learned parameters to assign a sequence of labels given a sequence of observations. Let's define an HMM framework containing the following components:

- states (e.g., labels): $T = t_1, t_2, \dots, t_N$
- observations (e.g., words): $W = w_1, w_2, \dots, w_N$
- two special states: t_{start} and t_{end} which are not associated with the observation

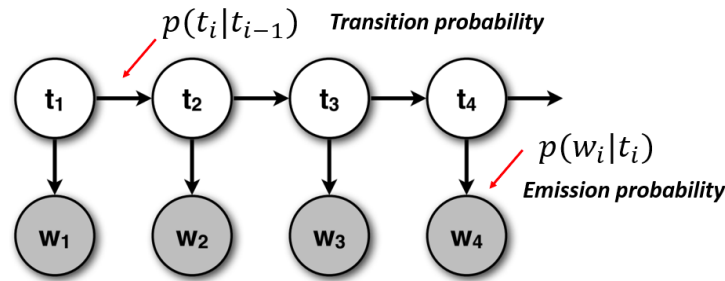
and probabilities relating states and observations:

- **initial probability**: an initial probability distribution over states
- **final probability**: a final probability distribution over states
- **transition probability**: a matrix A with the probabilities from going from one state to another
- **emission probability**: a matrix B with the probabilities of an observation being generated from a state

A First-order Hidden Markov Model has the following assumptions:

- **Markov Assumption**: the probability of a particular state is dependent only on the previous state. Formally: $P(t_i | t_1, \dots, t_{i-1}) = P(t_i | t_{i-1})$
- **Output Independence**: the probability of an output observation w_i depends only on the state that produced the observation t_i and not on any other states or any other observations. Formally: $P(w_i | t_1 \dots q_i, \dots, q_T, o_1, \dots, o_i, \dots, o_T) = P(o_i | q_i)$

Notice how the output assumption is closely related with the Naive Bayes classifier presented before. The figure below makes it easier to understand the dependencies and the relationship with the Naive Bayes classifier:



Transitions and Emissions probabilities in the HMM.
(image adapted from CS6501 of the University of Virginia)

We can now define two problems which can be solved by an HMM, the first is learning the parameters associated to a given observation sequence, that is **training**. For instance given words of a sentence and the associated part-of-speech tags, one can learn the latent structure.

The other one is applying a trained HMM to an observation sequence, for instance, having a sentence, **predicting** each word's part-of-speech tag, using the latent structure from the training data learned by the HMM.

Learning: estimating transition and emission matrices

Given an observation sequence W and the associated states T how can we learn the HMM parameters, that is, the matrices A and B ?

In a HMM supervised scenario this is done by applying the **Maximum Likelihood Estimation** principle, which will compute the matrices.

This is achieved by counting how many times each event occurs in the corpus and normalizing the counts to form proper probability distributions. We need to count 4 quantities which represent the counts of each event in the corpus:

$$\text{Initial counts: } C_{\text{init}}(t_k) = \sum_{m=1}^M 1(t_1^m = t_k)$$

(how often does state t_k is the initial state)

$$\text{Transition counts: } C_{\text{trans}}(t_k, t_l) = \sum_{m=1}^M \sum_{i=2}^N 1(t_i^m = t_k \wedge t_{i-1}^m = t_l)$$

(how often does state t_k transits to another state t_l)

$$\text{Final Counts: } C_{\text{final}}(t_k) = \sum_{m=1}^M 1(t_N^m = t_k)$$

(how often does state t_k is the final state)

$$\text{Emissions counts: } C_{\text{emiss}}(w_j, t_k) = \sum_{m=1}^M \sum_{i=1}^N 1(x_i^m = w_j \wedge t_i^m = t_k)$$

(how often does state t_k is associated with the observation/word w_j)

where, M is the number of training examples and N the length of the sequence, $\mathbf{1}$ is an indicator function that has the value 1 when the particular event happens, and 0 otherwise. The equations scan the training corpus and count how often each event occurs.

All these 4 counts are then normalised in order to have proper probability distributions:

$$P_{\text{init}(c_l|\text{start})} = \frac{C_{\text{init}(t_k)}}{\sum_{l=1}^K C_{\text{init}(t_l)}}$$

$$P_{\text{final}(\text{stop}|c_l)} = \frac{C_{\text{final}(c_l)}}{\sum_{k=1}^K C_{\text{trans}(C_k, C_l)} + C_{\text{final}(C_l)}}$$

$$P_{\text{trans}(c_k|c_l)} = \frac{C_{\text{trans}(c_k, c_l)}}{\sum_{p=1}^K C_{\text{trans}(C_p, C_l)} + C_{\text{final}(C_l)}}$$

$$P_{\text{emiss}(w_j|c_k)} = \frac{C_{\text{emiss}(w_j, c_k)}}{\sum_{q=1}^J C_{\text{emiss}(w_q, c_k)}}$$

These equations will produce the **transition probability** matrix A , with the probabilities from going from one label to another and the **emission probability** matrix B with the probabilities of an observation being generated from a state.

Laplace smoothing

How will the model handle words not seen during training ?

In the presence of an unseen word/observation, $P(W_i | T_i) = 0$ and has a consequence incorrect decisions will be made during the predicting process.

There is a technique to handle this situations called **Laplace smoothing or additive smoothing**. The idea is that every state will always have a small emission probability of producing an unseen word, for instance, denoted by **UNK**. Every time the HMM encounters an unknown word it will use the value $P(\text{UNK} | T_i)$ as the emission probability.

Decoding: finding the hidden state sequence for an observation

Given a trained HMM i.e., the transition matrixes A and B , and a new observation sequence $W = w_1, w_2, \dots, w_N$ we want to find the sequence of states $T = t_1, t_2, \dots, t_N$ that best explains it.

This is can be achieved by using the Viterbi algorithm, that finds the best state assignment to the sequence $T_1 \dots T_N$ as a whole. There is another algorithm, Posterior Decoding which consists in picking the highest state posterior for each position i in the sequence independently.



Viterbi

It's a dynamic programming algorithm for computing:

$$\delta_i(T) = \max_{t_0, \dots, t_{i-1}, t} P(t_0, \dots, t_{i-1}, t, w_1, \dots, w_{i-1})$$

the score of a best path up to position i ending in state t . The Viterbi algorithm tackles the equation above

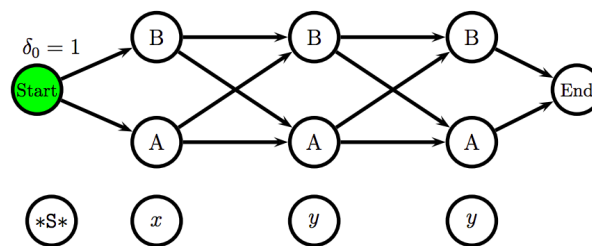
by using the Markov assumption and defining two functions:

$$\delta_i(t) = \max_{t_{i-1}} P(t \mid t_{i-1}) \cdot P(w_{i-1} \mid t_{i-1}) \cdot \delta_i(t_{i-1})$$

the most likely previous state for each state (store a back-trace):

$$\Psi_i(t) = \arg \max_{t_{i-1}} P(t \mid t_{i-1}) \cdot P(w \mid t_{i-1}) \cdot \delta_i(t_{i-1})$$

The Viterbi algorithm uses a representation of the HMM called a **trellis**, which unfolds all possible states for each position and it makes explicit the independence assumption: each position only depends on the previous position.

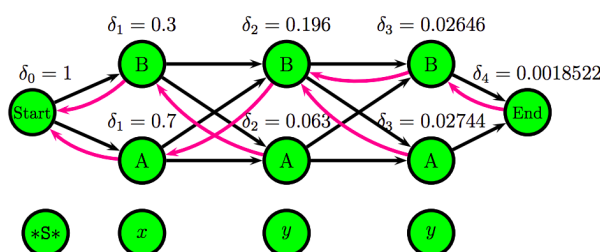


An unfilled trellis representation of an HMM.

State	Word			Current	Next		
	S	x	y		A	B	End
Start	1	0	0	Start	0.7	0.3	0
A	0	0.4	0.6	A	0.2	0.7	0.1
B	0	0.3	0.7	B	0.7	0.2	0.1

Word Emission and State Transitions probabilities matrices.

Using the Viterbi algorithm and the emission and transition probabilities matrices, one can fill in the trellis scores and effectively find the Viterby path.



An filled trellis representation of an HMM.

The figures above were taken from a Viterbi algorithm example by **Roger Levy** (<http://idiom.ucsd.edu/~rlevy/>) for the **Linguistics/CSE 256 class** (<http://idiom.ucsd.edu/~rlevy/teaching/winter2009/ligncse256/>). You can find the full example [here](#) ([/assets/documents/posts/2017-11-11-hmm_viterbi_mini_example.pdf](#)).

HMM Important Observations

- The main idea of this post was to see the connection between the Naive Bayes classifier and the HMM as a sequence classifier
- If we make the hidden state of HMM fixed, we will have a Naive Bayes model.
- There is only one feature at each word/observation in the sequence, namely the identity i.e., the value of the respective observation.
- Each state depends only on its immediate predecessor, that is, each state t_i is independent of all its ancestors t_1, t_2, \dots, t_{i-2} given its previous state t_{i-1} .
- Each observation variable w_i depends only on the current state t_i .

Software Packages

- **seqlearn** (<https://github.com/larsmans/seqlearn>): a sequence classification library for Python which includes an implementation of Hidden Markov Models, it follows the sklearn API.
- **NLTK HMM** (http://www.nltk.org/_modules/nltk/tag/hmm.html): NLTK also contains a module which implements a Hidden Markov Models framework.
- **lxmls-toolkit** (<https://github.com/LxMLS/lxmls-toolkit>): the Natural Language Processing Toolkit used in the Lisbon Machine Learning Summer School also contains an implementation of Hidden Markov Models.

References

- **Machine Learning for Sequential Data: A Review by Thomas G. Dietterich** (<http://web.engr.oregonstate.edu/~tgd/publications/mlsd-ssspr.pdf>)
- **Chapter 6: “Naive Bayes and Sentiment Classification” in Speech and Language Processing.** Daniel Jurafsky & James H. Martin. Draft of August 7, 2017. (<https://web.stanford.edu/~jurafsky/slp3/6.pdf>)
- **A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition** (<http://www.ece.ucsb.edu/Faculty/Rabiner/ece259/Reprints/tutorial%20on%20hmm%20and%20applications.pdf>)
- **LxMLS - Lab Guide July 16, 2017 - Day 2 “Sequence Models”** (<http://lxmls.it.pt/2017/LxMLS2017.pdf>)
- **Chapter 9: “Hidden Markov Models” in Speech and Language Processing.** Daniel Jurafsky & James H. Martin. Draft of August 7, 2017. (<https://web.stanford.edu/~jurafsky/slp3/9.pdf>)
- **Hidden Markov Model inference with the Viterbi algorithm: a mini-example** (/assets/documents/posts/2017-11-11-hmm_viterbi_mini_example.pdf)

Extra

There is also a very good lecture, given by **Noah Smith** (<https://homes.cs.washington.edu/~nasmith/>) at **LxMLS2016** (<http://lxmls.it.pt/2016/>) about Sequence Models, mainly focusing on Hidden Markov Models and it's applications from sequence learning to language modeling.



David S. Batista (/)

[About \(/about-me/\)](/about-me/)[Blog Posts \(/posts/\)](/posts/)[Datasets \(/nlp_datasets/\)](/nlp_datasets/)[Publications \(/publications/\)](/publications/)[Resources \(/nlp_resources/\)](/nlp_resources/)[CV \(/assets](/assets)[/documents/dsbatista-cv.en.pdf\)](/documents/dsbatista-cv.en.pdf)

Maximum Entropy Markov Models and Logistic Regression

2017-11-12

[maximum-entropy-markov-models \(/tag/maximum-entropy-markov-models\)](/tag/maximum-entropy-markov-models)[logistic-regression \(/tag/logistic-regression\)](/tag/logistic-regression)[sequence-prediction \(/tag/sequence-prediction\)](/tag/sequence-prediction)[viterbi \(/tag/viterbi\)](/tag/viterbi)

This is the second part of a series of posts about sequential supervised learning applied to NLP. It can be seen as a follow up on the previous post, where I tried to explain the relationship between HMM and Naive Bayes. In this post I will try to explain how to build a sequence classifier based on a Logistic Regression classifier, i.e., using a discriminative approach.

You can find the first and third posts here:

- [Hidden Markov Model and Naive Bayes relationship \(/blog/2017/11/11/HMM_and_Naive_Bayes/\)](/blog/2017/11/11/HMM_and_Naive_Bayes/)
- [Conditional Random Fields for Sequence Prediction \(/blog/2017/11/13/Conditional_Random_Fields/\)](/blog/2017/11/13/Conditional_Random_Fields/)

Discriminative vs. Generative Models

In a [previous post \(/blog/2017/11/12/HMM_and_Naive_Bayes/\)](/blog/2017/11/12/HMM_and_Naive_Bayes/) I wrote about the **Naive Bayes Model** and how it is connected with the **Hidden Markov Model**. Both are **generative models**, in contrast, **Logistic Regression** is a **discriminative model**, this post will start, by explaining this difference.

In general a machine learning classifier chooses which output label y to assign to an input x , by selecting from all the possible y_i the one that maximizes $P(y | x)$.

The Naive Bayes classifier estimates $p(y | x)$ indirectly, by applying the Bayes's theorem, and then computing the class conditional distribution/likelihood $P(x | y)$ and the prior $P(y)$.

$$\hat{y} = \arg \max_y P(y | x) = \arg \max_y P(x | y) \cdot P(y)$$

This indirection makes Naive Bayes a generative model, a model that is trained to generate the data x from the class y . The likelihood $p(x | y)$, means that we are given a class y and will try to predict which features to see in the input x .



In contrast a discriminative model directly computes $p(y | x)$ by discriminating among the different possible values of the class y instead of computing a likelihood. The Logistic Regression classifier is one of such type of classifiers.

$$\hat{y} = \arg \max_y P(y | x)$$

Logistic Regression

Logistic regression is supervised machine learning algorithm used for classification, which has its roots in linear regression.

When used to solve NLP tasks, it estimates $p(y | x)$ by extracting features from the input text and combining them linearly i.e., multiplying each feature by a weight and then adding them up, and then applying the exponential function to this linear combination:



$$P(y|x) = \frac{1}{Z} \exp \sum_{i=1}^N w_i \cdot f_i$$

where f_i is a feature and w_i the weight associated to the feature. The \exp (i.e., exponential function) surrounding the weight-feature dot product ensures that all values are positive and the denominator Z is needed to force all values into a valid probability where the sum is 1.

The extracted features, are binary-valued features, i.e., only takes the values 0 and 1, and are commonly called indicator functions. Each of these features is calculated by a function that is associated with the input x and the class y . Each indicator function is represented as $f_i(y, x)$, the feature i for class y , given



observation x :

$$P(y|x) = \frac{\exp \left(\sum_{i=1}^N w_i \cdot f_i(x, y) \right)}{\sum_{y' \in Y} \exp \left(\sum_{i=1}^N w_i \cdot f_i(x, y') \right)}$$

Training

By training the logistic regression classifier we want to find the ideal weights for each feature, that is, the weights that will make training examples fit best the classes to which they belong.

Logistic regression is trained with conditional maximum likelihood estimation. This means that we will choose the parameters w that maximize the probability of the y labels in the training data given the observations x :

$$\hat{w} = \arg \max_w \sum_j \log P(y^j | x^j)$$

The objective function to maximize is:

$$L(w) = \sum_j \log P(y^j | x^j)$$

which by replacing with expanded form presented before and by applying the division log rules, takes the following form:




$$L(w) = \sum_j \log \exp \left(\sum_{i=1}^N w_i \cdot f_i(x^j, y^j) \right) - \sum_j \log \sum_{y' \in Y} \exp \left(\sum_{i=1}^N w_i \cdot f_i(x^j, y'^j) \right)$$

Maximize this objective, i.e. finding the optimal weights, is typically solved by methods like stochastic gradient ascent, L-BFGS, or conjugate gradient.

Classification

In classification, logistic regression chooses a class by computing the probability of a given observation belonging to each of all the possible classes, then we can choose the one that yields the maximum probability.

$$\hat{y} = \arg \max_{y \in Y} P(y | x)$$



$$\hat{y} = \arg \max_{y \in Y} \frac{\exp \left(\sum_{i=1}^N w_i \cdot f_i(x, y) \right)}{\sum_{y' \in Y} \exp \left(\sum_{i=1}^N w_i \cdot f_i(x, y') \right)}$$

Maximum Entropy Markov Model

The idea of the Maximum Entropy Markov Model (MEMM) is to make use of both the HMM framework to **predict sequence labels given an observation sequence, but incorporating the multinomial Logistic Regression (aka Maximum Entropy)**, which gives freedom in the type and number of features one can extract from the observation sequence.

The HMM model is based on two probabilities:

- $P(\text{tag} | \text{tag})$ state transition, probability of going from one state to another.
- $P(\text{word} | \text{tag})$ emission probability, probability of a state emitting a word.

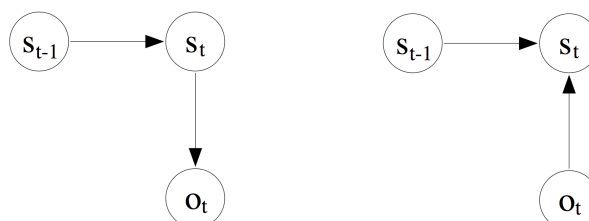
In real world problems we want to predict a tag/state given a word/observation. But, due to the Bayes theorem, that is, a generative approach, this is not possible to encode in the HMM, and the model estimates rather the probability of a state producing a certain word.

The **MEMM was proposed** (<http://www.ai.mit.edu/courses/6.891-nlp/READINGS/maxent.pdf>) as way to have richer set of observation features:

- *“a representation that describes observations in terms of many overlapping features, such as capitalization, word endings, part-of-speech, formatting, position on the page, and node memberships in WordNet, in addition to the traditional word identity.”*

and also to solve the prediction problem with a discriminative approach:

- *“the traditional approach sets the HMM parameters to maximize the likelihood of the observation sequence; however, in most text applications [...] the task is to predict the state sequence given the observation sequence. In other words, the traditional approach inappropriately uses a generative joint model in order to solve a conditional problem in which the observations are given.”*



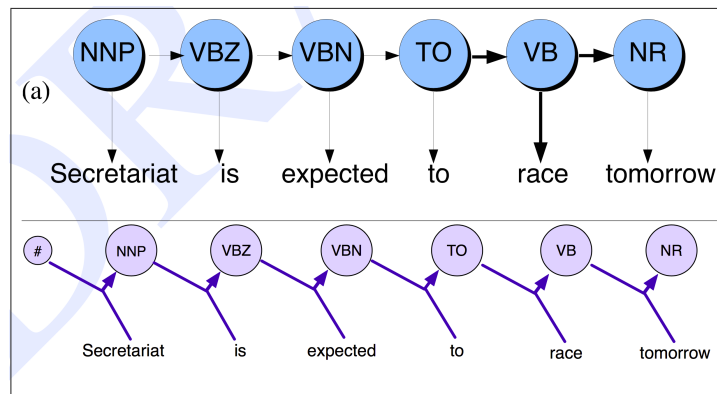
(Left) The dependency graph for a traditional HMM.

(Right) The dependency graph for a Maximum Entropy Markov Model.
(taken from A. McCallum et al. 2000)

In the Maximum Entropy Markov Models the transition and observation functions (i.e., the HMM matrices A and B from the previous post) are replaced by a single function:

$$P(s_t \mid s_{t-1}, o_t)$$

the probability of the current state s_t given the previous state s_{t-1} and the current observation o . The figure below shows this difference in computing the state/label/tag transitions.



Contrast in state transition estimation between an HMM and a MEMM.

(taken from "Speech and Language Processing" Daniel Jurafsky & James H. Martin)

In contrast to HMMs, in which the current observation only depends on the current state, the current observation in an MEMM may also depend on the previous state. The HMM model includes distinct probability estimates for each transition and observation, while the MEMM gives one probability estimate per hidden state, which is the probability of the next tag given the previous tag and the observation.

In the MEMM instead of the transition and observation matrices, there is only one transition probability matrix. This matrix encapsulates all combinations of previous states S_{t-1} and current observation O_t pairs in the training data to the current state S_t .

Let N be the number of unique states and M the number of unique words, the matrix has the shape:

$$(N \cdot M) \cdot N$$

Features Functions

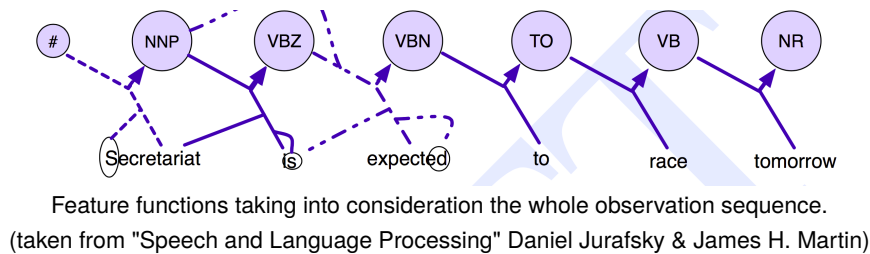
The MEMM can condition on any useful feature of the input observation, in the HMM this wasn't possible because the HMM is likelihood based, and hence we would have needed to compute the likelihood of each feature of the observation.

The use of state-observation transition functions, rather than the separate transition and observation functions as in HMMs, allows us to model transitions in terms of multiple, non-independent features of observations.

This is achieved by a multinomial logistic regression, to estimate the probability of each local tag given the previous tag (i.e., s'), the observed word (i.e. o), and any other features (i.e., $f_i(o, s')$) we want to include:

$$P(s \mid s', o) = \frac{1}{Z(o, s')} \exp \left(\sum_{i=1}^N w_i \cdot f_i(o, s') \right)$$

where, w_i are the weights to be learned, associated to each feature $f_i(o, s')$ and Z is the normalizing factor that makes the matrix sum to 1 across each row.



Training and Decoding

Taken from the original paper: "In what follows, we will split $P(s | s', O)$ into $|S|$ separately trained transition functions $P_{s'}(S | o) = P(s | s', O)$. Each of these functions is given by an exponential model"

THE MEMM trains one logistic regression per state transition, normalised locally. The original MEMM paper, published in 2000, used a generalized iterative scaling (GIS) algorithm to fit the multinomial logistic regression, that is finding the perfect weights according to the training data. That algorithm has been largely surpassed by gradient-based methods such as L-BFGS.

For the decoding, the same algorithm as in the HMM is used, the Viterbi, although just slightly adapted to accommodate the new method of estimating state transitions.

MEMM Important Observations

- The main advantage over the HMM is the use of feature vectors, making the transition probability sensitive to any word in the input sequence.
- There is an exponential model associate to each (state, word) pair to calculate the conditional probability of the next state.
- The exponential model allows the MEMMs to support long-distance interactions over the whole observation sequence together with the previous state, instead of two different probability distributions.
- **MEMM can be also augmented to include features involving additional past states, instead of just the previous one.**
- It also uses the Viterbi algorithm (slightly adapted) to perform decoding.
- **It suffers from the label bias problem, I will detailed in the next post about Conditional Random Fields.**

Software Packages

- <https://github.com/willxie/hmm-vs-memm> (<https://github.com/willxie/hmm-vs-memm>): a project for a class by William Xie which implements and compares HMM vs. MEMM on the task of part-of-speech tagging.
- <https://github.com/yh1008/MEMM> (<https://github.com/yh1008/MEMM>): an implementation by Emily Hua for the task of noun-phrase chunking.
- <https://github.com/recski/HunTag> (<https://github.com/recski/HunTag>): sequential sentence tagging implemented by Gábor Recski and well documented.

References

- Chapter 7: "Logistic Regression" in Speech and Language Processing. Daniel Jurafsky &

David S. Batista (/)

About (/about-me/)

Blog Posts (/posts/)

Datasets (/nlp_datasets/)

Publications (/publications/)

Resources (/nlp_resources/)

CV (/assets

/documents/dsbatista-cv.en.pdf)

Conditional Random Fields for Sequence Prediction

📅 2017-11-13 🏷️ `conditional-random-fields` (/tag/conditional-random-fields)

`sequence-prediction` (/tag/sequence-prediction) `viterbi` (/tag/viterbi)

This is the third and (maybe) the last part of a series of posts about sequential supervised learning applied to NLP. In this post I will talk about Conditional Random Fields (CRF), explain what was the main motivation behind the proposal of this model, and make a final comparison between Hidden Markov Models (HMM), Maximum Entropy Markov Models (MEMM) and CRF for sequence prediction.

You can find the first and second posts here:

- **Hidden Markov Model and Naive Bayes relationship** (.../blog/2017/11/11/HMM_and_Naive_Bayes/)
- **Maximum Entropy Markov Models and Logistic Regression** (.../blog/2017/11/12/Maximum_Entropy_Markov_Model/)

Introduction

CRFs were proposed roughly only year after the Maximum Entropy Markov Models, basically by the same authors. [Reading through the original paper that introduced Conditional Random Fields \(http://repository.upenn.edu/cgi/viewcontent.cgi?article=1162&context=cis_papers\)](http://repository.upenn.edu/cgi/viewcontent.cgi?article=1162&context=cis_papers), one finds at the beginning this sentence:

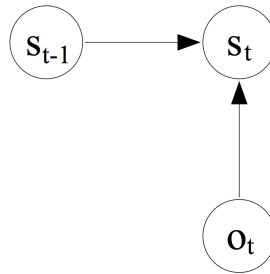
"The critical difference between CRF and MEMM is that the latter uses per-state exponential models for the conditional probabilities of next states given the current state, whereas CRF uses a single exponential model to determine the joint probability of the entire sequence of labels, given the observation sequence. Therefore, in CRF, the weights of different features in different states compete against each other."

This means that in the MEMMs there is a model to compute the probability of the next state, given the current state and the observation. On the other hand **CRF computes all state transitions globally, in a single model.**

The main motivation for this proposal is the so called Label Bias Problem occurring in MEMM, which generates a bias towards states with few successor states.

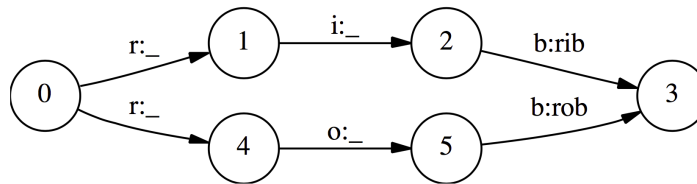
Label Bias Problem in MEMMs

Recalling how the transition probabilities are computed in a MEMM model, from the previous post, we learned that the probability of the next state is only dependent on the observation (i.e., the sequence of words) and the previous state, that is, we have an exponential model for each state to tell us the conditional probability of the next states:



MEMM transition probability computation.
(taken from A. McCallum et al. 2000)

This causes the so called **Label Bias Problem**, and Lafferty et al. 2001 demonstrate this through experiments and report it. I will not demonstrate it, but just give the basic intuition taken also from the paper:



Label Bias Problem.
(taken from Lafferty et al. 2001)

Given the observation sequence: ***ri b***

"In the first time step, r matches both transitions from the start state, so the probability mass gets distributed roughly equally among those two transitions. Next we observe i. Both states 1 and 4 have only one outgoing transition. State 1 has seen this observation often in training, state 4 has almost never seen this observation; but like state 1, state 4 has no choice but to pass all its mass to its single outgoing transition, since it is not generating the observation, only conditioning on it. Thus, states with a single outgoing transition effectively ignore their observations."

[...]

"the top path and the bottom path will be about equally likely, independently of the observation sequence. If one of the two words is slightly more common in the training set, the transitions out of the start state will slightly prefer its corresponding transition, and that word's state sequence will always win."

- Transitions from a given state are competing against each other only.
- Per state normalization, i.e. sum of transition probability for any state has to sum to 1.
- MEMM are normalized locally over each observation where the transitions going out from a state compete only against each other, as opposed to all the other transitions in the model.
- States with a single outgoing transition effectively ignore their observations.
- Causes bias: states with fewer arcs are preferred.



The idea of CRF is to drop this local per state normalization, and replace it by a global per sequence normalisation.

So, how do we formalise this global normalisation? I will try to explain it in the sections that follow.

Undirected Graphical Models

A Conditional Random Field can be seen as an undirected graphical model, or Markov Random Field,

globally conditioned on X , the random variable representing observation sequence.

Lafferty et al. 2001 (http://repository.upenn.edu/cgi/viewcontent.cgi?article=1162&context=cis_papers) define a Conditional Random Field as:

- X is a random variable over data sequences to be labeled, and Y is a random variable over corresponding label sequences.
- The random variables X and Y are jointly distributed, but in a discriminative framework we construct a conditional model $p(Y | X)$ from paired observation and label sequences:

Let $G = (V, E)$ be a graph such that $Y = (Y_v) \ v \in V$, so that Y is indexed by the vertices of G .

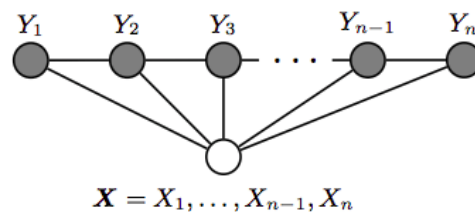
(X, Y) is a conditional random field when each of the random variables Y_v , conditioned on X , obey the Markov property with respect to the graph:

$$P(Y_v | X, Y_w, w \neq v) = P(Y_v | X, Y_w, w \sim v)$$

where $w \sim v$ means that w and v are neighbours in G . Thus, a CRF is a random field globally conditioned on the observation X . This goes already in the direction of what the MEMM doesn't give us, states globally conditioned on the observation.

This graph may have an arbitrary structure as long as it represents the label sequences being modelled, this is also called general Conditional Random Fields.

However the simplest and most common graph structured in NLP, which is the one used to model sequences is the one in which the nodes corresponding to elements of Y form a simple first-order chain, as illustrated in the figure below:



Chain-structured CRFs globally conditioned on X .
(taken from Hanna Wallach 2004)

This is also called linear-chain conditional random fields, which is the type of CRF on which the rest of this post will focus.

Linear-chain CRFs

Let \bar{x} is a sequence of words and \bar{y} a corresponding sequence of n tags:

$$P(\bar{y} | \bar{x}; \bar{w}) = \frac{\exp(\bar{w} \cdot F(\bar{x}, \bar{y}))}{\sum_{\bar{y}' \in Y} \exp(\bar{w} \cdot F(\bar{x}, \bar{y}'))}$$

This can be seen as another log-linear model, but “giant” in the sense that:

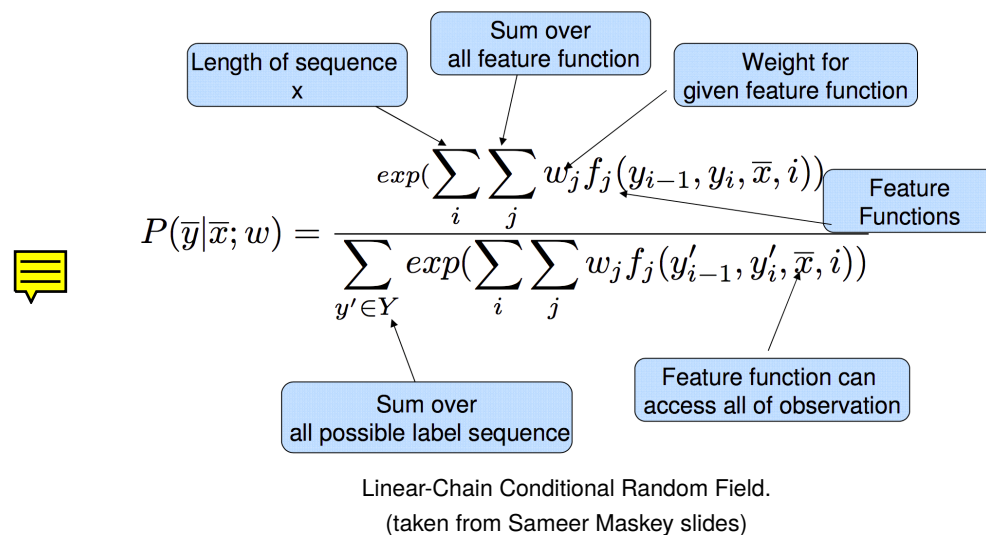
- The space of possible values for \bar{y} , i.e., Y^n , is huge, where n is the size of the sequence.
- The normalisation constant involves a sum over the set Y^n .

F will represent a global feature vector defined by a set of feature functions f_1, \dots, f_d , where each feature function f_j can analyse the whole \bar{x} sequence, the current y_i and previous y_{i-1} positions in the \bar{y} labels sequence, and the current position i in the sentence:

$$F(\bar{x}, \bar{y}) = \sum_i f(y_{i-1}, y_i, \bar{x}, i)$$

we can define an arbitrary number of feature functions. The k 'th global feature is then computed by summing the f_k over all the n different state transitions \bar{y} . In this way we have a “global” feature vector that maps the entire sequence: $F(\bar{x}, \bar{y}) \in \mathbb{R}^d$.

Thus, the full expanded linear-chain CRF equation is:



The diagram shows the equation for the Linear-Chain Conditional Random Field with several annotations in blue boxes:

- Length of sequence x**: points to \bar{x} in the numerator and denominator.
- Sum over all feature function**: points to the inner sum over j in the numerator.
- Weight for given feature function**: points to w_j in the numerator.
- Feature Functions**: points to f_j in the numerator.
- Feature function can access all of observation**: points to \bar{x} in the denominator.
- Sum over all possible label sequence**: points to the sum over $y' \in Y$ in the denominator.

Linear-Chain Conditional Random Field.
(taken from Sameer Maskey slides)

Having the framework defined by the equation above we now analyse how to perform two operations: parameter estimation and sequence prediction.

Inference

Inference with a linear-chain CRF resolves to computing the \bar{y} sequence that maximizes the following equation:

$$\hat{\bar{y}} = \arg \max_{\bar{y}} P(\bar{y} | \bar{x}; \bar{w}) = \frac{\exp(\bar{w} \cdot F(\bar{x}, \bar{y}))}{\sum_{\bar{y}' \in Y} \exp(\bar{w} \cdot F(\bar{x}, \bar{y}'))}$$

We want to try all possible \bar{y} sequences computing for each one the probability of “fitting” the observation \bar{x} with feature weights \bar{w} . If we just want the score for a particular labelling sequence \bar{y} , we can ignore the exponential inside the numerator, and the denominator:

$$\hat{\bar{y}} = \arg \max_{\bar{y}} P(\bar{y} | \bar{x}; w) = \sum_j \bar{w} F(\bar{x}, \bar{y})$$

then, we replace $F(\bar{x}, \bar{y})$ by its definition:

$$\hat{\bar{y}} = \arg \max_{\bar{y}} \sum_i \bar{w} f(y_{i-1}, y_i, \bar{x}, i)$$

Each transition from state y_{i-1} to state y_i has an associated score:

$$\bar{w} f(y_{i-1}, y_i, \bar{x}, i)$$

Since we took the \exp out, this score could be positive or negative, intuitively, this score will be relatively high if the state transition is plausible, relatively low if this transition is implausible.

The decoding problem is then to find an entire sequence of states such that the sum of the transition scores is maximized. We can again solve this problem using a variant of the Viterbi algorithm, in a very similar way

to the decoding algorithm for HMMs or MEMMs.

The denominator, also called the partition function:

$$Z(\bar{x}, w) = \sum_{\bar{y}' \in Y} \exp\left(\sum_j w_j F_j(\bar{x}, \bar{y}')$$

is useful to compute a marginal probability. For example, this is useful for measuring the model's confidence in its predicted labeling over a segment of input. This marginal probability can be computed efficiently using the forward-backward algorithm. See the references section for demonstrations on how this is achieved.

Parameter Estimation

We also need to find the \bar{w} parameters that best fit the training data, a given a set of labelled sentences:

$$\{(\bar{x}_1, \bar{y}_1), \dots, (\bar{x}_m, \bar{y}_m)\}$$

where each pair (\bar{x}_i, \bar{y}_i) is a sentence with the corresponding word labels annotated. To find the \bar{w} parameters that best fit the data we need to maximize the conditional likelihood of the training data:

$$L(\bar{w}) = \sum_{i=1}^m \log p(\bar{x}_i | \bar{y}_i, \bar{w})$$

the parameter estimates are computed as:

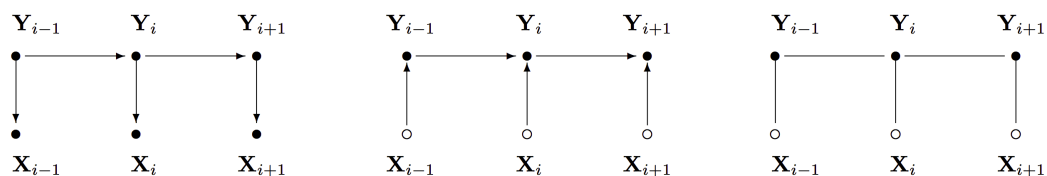
$$\bar{w}^* = \arg \max_{\bar{w} \in \mathbb{R}^d} \sum_{i=1}^m \log p(\bar{x}_i | \bar{y}_i, \bar{w}) - \frac{\lambda}{2} \|\bar{w}\|^2$$

where $\frac{\lambda}{2} \|\bar{w}\|^2$ is an L2 regularization term.

The standard approach to finding \bar{w}^* is to compute the gradient of the objective function, and use the gradient in an optimization algorithm like L-BFGS.

Wrapping up: HMM vs. MEMM vs. CRF

It is now helpful to look at the three sequence prediction models, and compared them. The figure bellow shows the graphical representation for the Hidden Markov Model, the Maximum Entropy Markov Model and the Conditional Random Fields.



Graph representation of HMM, MEMM and CRF.
(taken from Lafferty et al. 2001)

- **Hidden Markov Models:**

$$P(\bar{y}, \bar{x}) = \prod_{i=1}^{|\bar{y}|} P(y_i | y_{i-1}) \cdot P(x_i | y_i)$$

- **Maximum Entropy Markov Models:**

$$P(\bar{y}, \bar{x}) = \prod_{i=1}^{|\bar{y}|} P(y_i | y_{i-1}, x_i) = \prod_{i=1}^{|\bar{y}|} \frac{1}{Z(x, y_{i-1})} \exp \left(\sum_{j=1}^N w_j \cdot f_j(x, y_{i-1}) \right)$$

- **Conditional Random Fields:**

$$P(\bar{y} | \bar{x}, \bar{w}) = \frac{\exp(\bar{w} \cdot F(\bar{x}, \bar{y}))}{\sum_{\bar{y}' \in Y} \exp(\bar{w} \cdot F(\bar{x}, \bar{y}'))}$$

CRF Important Observations

- MEMMs are normalized locally over each observation, and hence suffer from the Label Bias problem, where the transitions going out from a state compete only against each other, as opposed to all the other transitions in the model.
- CRFs avoid the label bias problem a weakness exhibited by Maximum Entropy Markov Models (MEMM). The big difference between MEMM and CRF is that MEMM is locally renormalized and suffers from the label bias problem, while CRFs are globally re-normalized.
- The inference algorithm in CRF is again based on Viterbi algorithm.
- Output transition and observation probabilities are not modelled separately.
- Output transition dependent on the state and the observation as one conditional probability.

Software Packages

- **python-crfsuite** (<https://github.com/scrapinghub/python-crfsuite>): is a python binding for **CRFSuite** (<https://github.com/chokkan/crfsuite>) which is a fast implementation of Conditional Random Fields written in C++.
- **CRF++: Yet Another CRF toolkit** (<https://taku910.github.io/crfpp/>): is a popular implementation in C++ but as far as I know there are no python bindings.
- **MALLET** (<http://mallet.cs.umass.edu/>): includes implementations of widely used sequence algorithms including hidden Markov models (HMMs) and linear chain conditional random fields (CRFs), it's written in Java.
- **FlexCRFs** (<http://flexcrfs.sourceforge.net/>) supports both first-order and second-order Markov CRFs, it's written in C/C++ using STL library.
- **python-wapiti** (<https://github.com/adsva/python-wapiti>) is a python wrapper for **wapiti** (<http://wapiti.limsi.fr>), a sequence labeling tool with support for maxent models, maximum entropy Markov models and linear-chain CRF.

References

- “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data” (http://repository.upenn.edu/cgi/viewcontent.cgi?article=1162&context=cis_papers)
- “Log-linear models and Conditional Random Fields”. Notes for a tutorial at CIKM'08 by Charles Elkan. October 20, 2008” (<https://pdfs.semanticscholar.org/5f5c/171b07540cf739485967cab50fc00dd26ae1.pdf>)
- Video: tutorial at CIKM'08 by Charles Elkan (http://videlectures.net/cikm08_elkan_llmacrf)

1 An Introduction to Conditional Random Fields for Relational Learning



Charles Sutton

Department of Computer Science
University of Massachusetts, USA
casutton@cs.umass.edu
<http://www.cs.umass.edu/~casutton>

Andrew McCallum

Department of Computer Science
University of Massachusetts, USA
mccallum@cs.umass.edu
<http://www.cs.umass.edu/~mccallum>

1.1 Introduction

Relational data has two characteristics: first, statistical dependencies exist between the entities we wish to model, and second, each entity often has a rich set of features that can aid classification. For example, when classifying Web documents, the page's text provides much information about the class label, but hyperlinks define a relationship between pages that can improve classification [Taskar et al., 2002]. Graphical models are a natural formalism for exploiting the dependence structure among entities. Traditionally, graphical models have been used to represent the joint probability distribution $p(\mathbf{y}, \mathbf{x})$, where the variables \mathbf{y} represent the attributes of the entities that we wish to predict, and the input variables \mathbf{x} represent our observed knowledge about the entities. But modeling the joint distribution can lead to difficulties when using the rich local features that can occur in relational data, because it requires modeling the distribution $p(\mathbf{x})$, which can include complex dependencies. Modeling these dependencies among inputs can lead to intractable models, but ignoring them can lead to reduced performance.

A solution to this problem is to directly model the conditional distribution $p(\mathbf{y}|\mathbf{x})$, which is sufficient for classification. This is the approach taken by *conditional random fields* [Lafferty et al., 2001]. A conditional random field is simply a conditional distribution $p(\mathbf{y}|\mathbf{x})$ with an associated graphical structure. Because the model is

conditional, dependencies among the input variables \mathbf{x} do not need to be explicitly represented, affording the use of rich, global features of the input. For example, in natural language tasks, useful features include neighboring words and word bi-grams, prefixes and suffixes, capitalization, membership in domain-specific lexicons, and semantic information from sources such as WordNet. Recently there has been an explosion of interest in CRFs, with successful applications including text processing [Taskar et al., 2002, Peng and McCallum, 2004, Settles, 2005, Sha and Pereira, 2003], bioinformatics [Sato and Sakakibara, 2005, Liu et al., 2005], and computer vision [He et al., 2004, Kumar and Hebert, 2003].

This chapter is divided into two parts. First, we present a tutorial on current training and inference techniques for conditional random fields. We discuss the important special case of linear-chain CRFs, and then we generalize these to arbitrary graphical structures. We include a brief discussion of techniques for practical CRF implementations.

Second, we present an example of applying a general CRF to a practical relational learning problem. In particular, we discuss the problem of *information extraction*, that is, automatically building a relational database from information contained in unstructured text. Unlike linear-chain models, general CRFs can capture long distance dependencies between labels. For example, if the same name is mentioned more than once in a document, all mentions probably have the same label, and it is useful to extract them all, because each mention may contain different complementary information about the underlying entity. To represent these long-distance dependencies, we propose a *skip-chain CRF*, a model that jointly performs segmentation and collective labeling of extracted mentions. On a standard problem of extracting speaker names from seminar announcements, the skip-chain CRF has better performance than a linear-chain CRF.

1.2 Graphical Models

1.2.1 Definitions

We consider probability distributions over sets of random variables $V = X \cup Y$, where X is a set of *input variables* that we assume are observed, and Y is a set of *output variables* that we wish to predict. Every variable $v \in V$ takes outcomes from a set \mathcal{V} , which can be either continuous or discrete, although we discuss only the discrete case in this chapter. We denote an assignment to X by \mathbf{x} , and we denote an assignment to a set $A \subset X$ by \mathbf{x}_A , and similarly for Y . We use the notation $\mathbf{1}_{\{x=x'\}}$ to denote an indicator function of x which takes the value 1 when $x = x'$ and 0 otherwise.

A graphical model is a family of probability distributions that factorize according to an underlying graph. The main idea is to represent a distribution over a large number of random variables by a product of local functions that each depend on only a small number of variables. Given a collection of subsets $A \subset V$, we define

an *undirected graphical model* as the set of all distributions that can be written in the form

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \prod_A \Psi_A(\mathbf{x}_A, \mathbf{y}_A), \quad (1.1)$$

for any choice of *factors* $F = \{\Psi_A\}$, where $\Psi_A : \mathcal{V}^n \rightarrow \mathbb{R}^+$. (These functions are also called *local functions* or *compatibility functions*.) We will occasionally use the term *random field* to refer to a particular distribution among those defined by an undirected model. To reiterate, we will consistently use the term *model* to refer to a family of distributions, and *random field* (or more commonly, distribution) to refer to a single one.

The constant Z is a normalization factor defined as

$$Z = \sum_{\mathbf{x}, \mathbf{y}} \prod_A \Psi_A(\mathbf{x}_A, \mathbf{y}_A), \quad (1.2)$$

which ensures that the distribution sums to 1. The quantity Z , considered as a function of the set F of factors, is called the *partition function* in the statistical physics and graphical models communities. Computing Z is intractable in general, but much work exists on how to approximate it.

Graphically, we represent the factorization (1.1) by a *factor graph* [Kschischang et al., 2001]. A factor graph is a bipartite graph $G = (V, F, E)$ in which a variable node $v_s \in V$ is connected to a factor node $\Psi_A \in F$ if v_s is an argument to Ψ_A . An example of a factor graph is shown graphically in Figure 1.1 (right). In that figure, the circles are variable nodes, and the shaded boxes are factor nodes.

In this chapter, we will assume that each local function has the form

$$\Psi_A(\mathbf{x}_A, \mathbf{y}_A) = \exp \left\{ \sum_k \theta_{Ak} f_{Ak}(\mathbf{x}_A, \mathbf{y}_A) \right\}, \quad (1.3)$$

for some real-valued parameter vector θ_A , and for some set of *feature functions* or *sufficient statistics* $\{f_{Ak}\}$. This form ensures that the family of distributions over V parameterized by θ is an exponential family. Much of the discussion in this chapter actually applies to exponential families in general.

A *directed graphical model*, also known as a Bayesian network, is based on a directed graph $G = (V, E)$. A directed model is a family of distributions that factorize as:

$$p(\mathbf{y}, \mathbf{x}) = \prod_{v \in V} p(v | \pi(v)), \quad (1.4)$$

where $\pi(v)$ are the parents of v in G . An example of a directed model is shown in Figure 1.1 (left).

We use the term *generative model* to refer to a directed graphical model in which the outputs topologically precede the inputs, that is, no $x \in X$ can be a parent of an output $y \in Y$. Essentially, a generative model is one that directly describes how the outputs probabilistically “generate” the inputs.

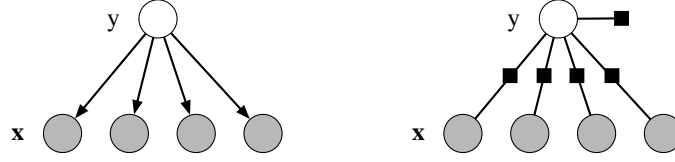


Figure 1.1 The naive Bayes classifier, as a directed model (left), and as a factor graph (right).

1.2.2 Applications of graphical models

In this section we discuss a few applications of graphical models to natural language processing. Although these examples are well-known, they serve both to clarify the definitions in the previous section, and to illustrate some ideas that will arise again in our discussion of conditional random fields. We devote special attention to the hidden Markov model (HMM), because it is closely related to the linear-chain CRF.

1.2.2.1 Classification

First we discuss the problem of *classification*, that is, predicting a single class variable y given a vector of features $\mathbf{x} = (x_1, x_2, \dots, x_K)$. One simple way to accomplish this is to assume that once the class label is known, all the features are independent. The resulting classifier is called the *naive Bayes classifier*. It is based on a joint probability model of the form:

$$p(y, \mathbf{x}) = p(y) \prod_{k=1}^K p(x_k | y). \quad (1.5)$$

This model can be described by the directed model shown in Figure 1.1 (left). We can also write this model as a factor graph, by defining a factor $\Psi(y) = p(y)$, and a factor $\Psi_k(y, x_k) = p(x_k | y)$ for each feature x_k . This factor graph is shown in Figure 1.1 (right).

Another well-known classifier that is naturally represented as a graphical model is logistic regression (sometimes known as the *maximum entropy classifier* in the NLP community). In statistics, this classifier is motivated by the assumption that the log probability, $\log p(y | \mathbf{x})$, of each class is a linear function of \mathbf{x} , plus a normalization constant. This leads to the conditional distribution:

$$p(y | \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \left\{ \lambda_y + \sum_{j=1}^K \lambda_{y,j} x_j \right\}, \quad (1.6)$$

where $Z(\mathbf{x}) = \sum_y \exp \{ \lambda_y + \sum_{j=1}^K \lambda_{y,j} x_j \}$ is a normalizing constant, and λ_y is a bias weight that acts like $\log p(y)$ in naive Bayes. Rather than using one vector per class, as in (1.6), we can use a different notation in which a single set of weights is shared across all the classes. The trick is to define a set of *feature functions* that are

nonzero only for a single class. To do this, the feature functions can be defined as $f_{y',j}(y, \mathbf{x}) = \mathbf{1}_{\{y'=y\}}x_j$ for the feature weights and $f_{y'}(y, \mathbf{x}) = \mathbf{1}_{\{y'=y\}}$ for the bias weights. Now we can use f_k to index each feature function $f_{y',j}$, and λ_k to index its corresponding weight $\lambda_{y',j}$. Using this notational trick, the logistic regression model becomes:

$$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \left\{ \sum_{k=1}^K \lambda_k f_k(y, \mathbf{x}) \right\}. \quad (1.7)$$

We introduce this notation because it mirrors the usual notation for conditional random fields.

1.2.2.2 Sequence Models

Classifiers predict only a single class variable, but the true power of graphical models lies in their ability to model many variables that are interdependent. In this section, we discuss perhaps the simplest form of dependency, in which the output variables are arranged in a sequence. To motivate this kind of model, we discuss an application from natural language processing, the task of *named-entity recognition* (NER). NER is the problem of identifying and classifying proper names in text, including locations, such as *China*; people, such as *George Bush*; and organizations, such as the *United Nations*. The named-entity recognition task is, given a sentence, first to segment which words are part of entities, and then to classify each entity by type (person, organization, location, and so on). The challenge of this problem is that many named entities are too rare to appear even in a large training set, and therefore the system must identify them based only on context.

One approach to NER is to classify each word independently as one of either PERSON, LOCATION, ORGANIZATION, or OTHER (meaning not an entity). The problem with this approach is that it assumes that given the input, all of the named-entity labels are independent. In fact, the named-entity labels of neighboring words are dependent; for example, while *New York* is a location, *New York Times* is an organization.

This independence assumption can be relaxed by arranging the output variables in a linear chain. This is the approach taken by the hidden Markov model (HMM) [Rabiner, 1989]. An HMM models a sequence of observations $X = \{x_t\}_{t=1}^T$ by assuming that there is an underlying sequence of *states* $Y = \{y_t\}_{t=1}^T$ drawn from a finite state set S . In the named-entity example, each observation x_t is the identity of the word at position t , and each state y_t is the named-entity label, that is, one of the entity types PERSON, LOCATION, ORGANIZATION, and OTHER.

To model the joint distribution $p(\mathbf{y}, \mathbf{x})$ tractably, an HMM makes two independence assumptions. First, it assumes that each state depends only on its immediate predecessor, that is, each state y_t is independent of all its ancestors y_1, y_2, \dots, y_{t-2} given its previous state y_{t-1} . Second, an HMM assumes that each observation variable x_t depends only on the current state y_t . With these assumptions, we can

specify an HMM using three probability distributions: first, the distribution $p(y_1)$ over initial states; second, the transition distribution $p(y_t|y_{t-1})$; and finally, the observation distribution $p(x_t|y_t)$. That is, the joint probability of a state sequence \mathbf{y} and an observation sequence \mathbf{x} factorizes as

$$p(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^T p(y_t|y_{t-1})p(x_t|y_t), \quad (1.8)$$

where, to simplify notation, we write the initial state distribution $p(y_1)$ as $p(y_1|y_0)$. In natural language processing, HMMs have been used for sequence labeling tasks such as part-of-speech tagging, named-entity recognition, and information extraction.

1.2.3 Discriminative and Generative Models

An important difference between naive Bayes and logistic regression is that naive Bayes is *generative*, meaning that it is based on a model of the joint distribution $p(y, \mathbf{x})$, while logistic regression is *discriminative*, meaning that it is based on a model of the conditional distribution $p(y|\mathbf{x})$. In this section, we discuss the differences between generative and discriminative modeling, and the advantages of discriminative modeling for many tasks. For concreteness, we focus on the examples of naive Bayes and logistic regression, but the discussion in this section actually applies in general to the differences between generative models and conditional random fields.

The main difference is that a conditional distribution $p(\mathbf{y}|\mathbf{x})$ does not include a model of $p(\mathbf{x})$, which is not needed for classification anyway. The difficulty in modeling $p(\mathbf{x})$ is that it often contains many highly dependent features, which are difficult to model. For example, in named-entity recognition, an HMM relies on only one feature, the word's identity. But many words, especially proper names, will not have occurred in the training set, so the word-identity feature is uninformative. To label unseen words, we would like to exploit other features of a word, such as its capitalization, its neighboring words, its prefixes and suffixes, its membership in predetermined lists of people and locations, and so on.

To include interdependent features in a generative model, we have two choices: enhance the model to represent dependencies among the inputs, or make simplifying independence assumptions, such as the naive Bayes assumption. The first approach, enhancing the model, is often difficult to do while retaining tractability. For example, it is hard to imagine how to model the dependence between the capitalization of a word and its suffixes, nor do we particularly wish to do so, since we always observe the test sentences anyway. The second approach, adding independence assumptions among the inputs, is problematic because it can hurt performance. For example, although the naive Bayes classifier performs surprisingly well in document classification, it performs worse on average across a range of applications than logistic regression [Caruana and Niculescu-Mizil, 2005].

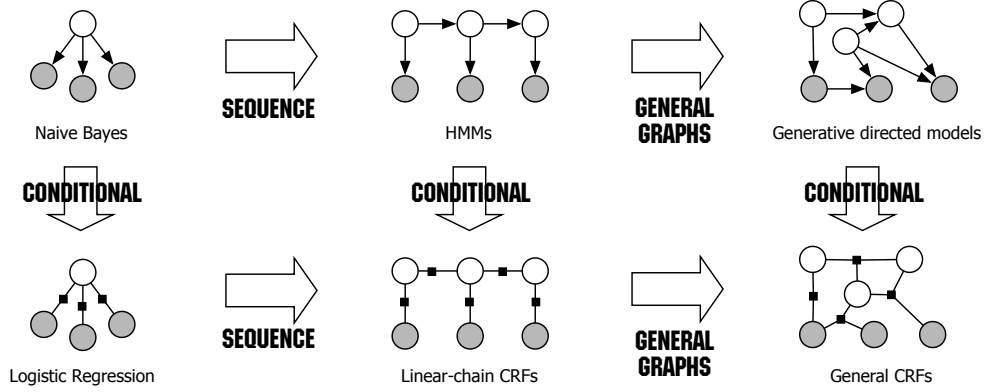


Figure 1.2 Diagram of the relationship between naive Bayes, logistic regression, HMMs, linear-chain CRFs, generative models, and general CRFs.

Furthermore, even when naive Bayes has good classification accuracy, its probability estimates tend to be poor. To understand why, imagine training naive Bayes on a data set in which all the features are repeated, that is, $\mathbf{x} = (x_1, x_1, x_2, x_2, \dots, x_K, x_K)$. This will increase the confidence of the naive Bayes probability estimates, even though no new information has been added to the data. Assumptions like naive Bayes can be especially problematic when we generalize to sequence models, because inference essentially combines evidence from different parts of the model. If probability estimates at a local level are overconfident, it might be difficult to combine them sensibly.

Actually, the difference in performance between naive Bayes and logistic regression is due *only* to the fact that the first is generative and the second discriminative; the two classifiers are, for discrete input, identical in all other respects. Naive Bayes and logistic regression consider the same hypothesis space, in the sense that any logistic regression classifier can be converted into a naive Bayes classifier with the same decision boundary, and vice versa. Another way of saying this is that the naive Bayes model (1.5) defines the same family of distributions as the logistic regression model (1.7), if we interpret it generatively as

$$p(y, \mathbf{x}) = \frac{\exp \{ \sum_k \lambda_k f_k(y, \mathbf{x}) \}}{\sum_{\tilde{y}, \tilde{\mathbf{x}}} \exp \{ \sum_k \lambda_k f_k(\tilde{y}, \tilde{\mathbf{x}}) \}}. \quad (1.9)$$

This means that if the naive Bayes model (1.5) is trained to maximize the conditional likelihood, we recover the same classifier as from logistic regression. Conversely, if the logistic regression model is interpreted generatively, as in (1.9), and is trained to maximize the joint likelihood $p(y, \mathbf{x})$, then we recover the same classifier as from naive Bayes. In the terminology of Ng and Jordan [2002], naive Bayes and logistic regression form a *generative-discriminative pair*.

The principal advantage of discriminative modeling is that it is better suited to

including rich, overlapping features. To understand this, consider the family of naive Bayes distributions (1.5). This is a family of joint distributions whose conditionals all take the “logistic regression form” (1.7). But there are many other joint models, some with complex dependencies among \mathbf{x} , whose conditional distributions also have the form (1.7). By modeling the conditional distribution directly, we can remain agnostic about the form of $p(\mathbf{x})$. This may explain why it has been observed that conditional random fields tend to be more robust than generative models to violations of their independence assumptions [Lafferty et al., 2001]. Simply put, CRFs make independence assumptions among \mathbf{y} , but not among \mathbf{x} .

Another way to make the same point is due to Minka [2005]. Suppose we have a generative model p_g with parameters θ . By definition, this takes the form

$$p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{y}; \theta) p_g(\mathbf{x} | \mathbf{y}; \theta). \quad (1.10)$$

But we could also rewrite p_g using Bayes rule as

$$p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{x}; \theta) p_g(\mathbf{y} | \mathbf{x}; \theta), \quad (1.11)$$

where $p_g(\mathbf{x}; \theta)$ and $p_g(\mathbf{y} | \mathbf{x}; \theta)$ are computed by inference, i.e., $p_g(\mathbf{x}; \theta) = \sum_{\mathbf{y}} p_g(\mathbf{y}, \mathbf{x}; \theta)$ and $p_g(\mathbf{y} | \mathbf{x}; \theta) = p_g(\mathbf{y}, \mathbf{x}; \theta) / p_g(\mathbf{x}; \theta)$.

Now, compare this generative model to a discriminative model over the same family of joint distributions. To do this, we define a prior $p(\mathbf{x})$ over inputs, such that $p(\mathbf{x})$ could have arisen from p_g with some parameter setting. That is, $p(\mathbf{x}) = p_c(\mathbf{x}; \theta') = \sum_{\mathbf{y}} p_g(\mathbf{y}, \mathbf{x}; \theta')$. We combine this with a conditional distribution $p_c(\mathbf{y} | \mathbf{x}; \theta)$ that could also have arisen from p_g , that is, $p_c(\mathbf{y} | \mathbf{x}; \theta) = p_g(\mathbf{y}, \mathbf{x}; \theta) / p_g(\mathbf{x}; \theta)$. Then the resulting distribution is

$$p_c(\mathbf{y}, \mathbf{x}) = p_c(\mathbf{x}; \theta') p_c(\mathbf{y} | \mathbf{x}; \theta). \quad (1.12)$$

By comparing (1.11) with (1.12), it can be seen that the conditional approach has more freedom to fit the data, because it does not require that $\theta = \theta'$. Intuitively, because the parameters θ in (1.11) are used in both the input distribution and the conditional, a good set of parameters must represent both well, potentially at the cost of trading off accuracy on $p(\mathbf{y} | \mathbf{x})$, the distribution we care about, for accuracy on $p(\mathbf{x})$, which we care less about.

In this section, we have discussed the relationship between naive Bayes and logistic regression in detail because it mirrors the relationship between HMMs and linear-chain CRFs. Just as naive Bayes and logistic regression are a generative-discriminative pair, there is a discriminative analog to hidden Markov models, and this analog is a particular type of conditional random field, as we explain next. The analogy between naive Bayes, logistic regression, generative models, and conditional random fields is depicted in Figure 1.2.

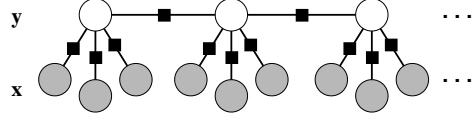


Figure 1.3 Graphical model of an HMM-like linear-chain CRF.

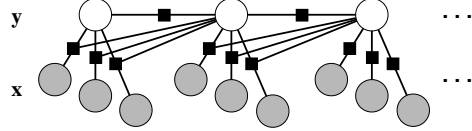


Figure 1.4 Graphical model of a linear-chain CRF in which the transition score depends on the current observation.

1.3 Linear-Chain Conditional Random Fields

In the previous section, we have seen advantages both to discriminative modeling and sequence modeling. So it makes sense to combine the two. This yields a linear-chain CRF, which we describe in this section. First, in Section 1.3.1, we define linear-chain CRFs, motivating them from HMMs. Then, we discuss parameter estimation (Section 1.3.2) and inference (Section 1.3.3) in linear-chain CRFs.

1.3.1 From HMMs to CRFs

To motivate our introduction of linear-chain conditional random fields, we begin by considering the conditional distribution $p(\mathbf{y}|\mathbf{x})$ that follows from the joint distribution $p(\mathbf{y}, \mathbf{x})$ of an HMM. The key point is that this conditional distribution is in fact a conditional random field with a particular choice of feature functions. First, we rewrite the HMM joint (1.8) in a form that is more amenable to generalization. This is

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \exp \left\{ \sum_t \sum_{i,j \in S} \lambda_{ij} \mathbf{1}_{\{y_t=i\}} \mathbf{1}_{\{y_{t-1}=j\}} + \sum_t \sum_{i \in S} \sum_{o \in O} \mu_{oi} \mathbf{1}_{\{y_t=i\}} \mathbf{1}_{\{x_t=o\}} \right\}, \quad (1.13)$$

where $\theta = \{\lambda_{ij}, \mu_{oi}\}$ are the parameters of the distribution, and can be any real numbers. Every HMM can be written in this form, as can be seen simply by setting $\lambda_{ij} = \log p(y' = i | y = j)$ and so on. Because we do not require the parameters to be log probabilities, we are no longer guaranteed that the distribution sums to 1, unless we explicitly enforce this by using a normalization constant Z . Despite this added flexibility, it can be shown that (1.13) describes exactly the class of HMMs in (1.8); we have added flexibility to the parameterization, but we have not added any distributions to the family.