

Learning What's in a Name with Graphical Models

Vu Luong, Justin Selig

A peer-reviewed publication in the [Journal of Visualization and Interaction \(JoVI\)](#).

DOI: [10.54337/jovi.v1i1.10824](https://doi.org/10.54337/jovi.v1i1.10824) (the public peer reviews can also be found at this link)

- Submitted: 2024-03-30
- Accepted: 2025-08-05
- Published: 2025-12-01 (version 1)

License and Copyright

This work is licensed under a [Creative Commons Attribution 4.0 International \(CC-BY 4.0\)](#) license.

All copyrights remain with the authors.

Publisher

Aalborg University Open Publishing
Kroghstræde 1-3
9220 Aalborg Øst
DENMARK
ISSN: 2794-5502

Learning What's in a Name with Graphical Models

Abstract

“The UK” is a country, but “The UK Department of Transport” is an organization within that country. In a named entity recognition (NER) task, where we want to label each word with a name tag (organization/person/location/other/not a name), how can a computer model know one from the other?

In this article, we'll explore three model families that are remarkably successful at NER: Hidden Markov Models (HMMs), Maximum-Entropy Markov Models (MEMMs), and Conditional Random Fields (CRFs). We'll use interactive visualizations to explain the graphical structure of each. Our overarching goal is to demonstrate how visualizations can be effective tools for communicating and clarifying complex, abstract concepts. The visualizations will allow us to compare and contrast between model families, and understand how each builds on and addresses key issues affecting its predecessors.

Tip: refer to the glossary in the top right corner for definitions of key terms like “named entity”.

Which of these words refer to named entity?

The UK Department of Transport



HMM

0

ORG

ORG

0

0

MEMM

0

LOC

LOC

LOC

LOC

CRF

0

ORG

ORG

ORG

ORG

0 = not a name ORG = organization PER = person LOC = location MISC = miscellaneous

I.

Probabilistic Graphical Models

Graphical modeling is a robust framework for representing probabilistic models. Complex multivariate probability distributions can be expressed with compact graphs that are easier to understand and interpret.

Factorizing Joint Distributions

Let's start with a simple example with two random variables, A and B . Assume that B is conditionally dependent on A . Through a canonical application of the chain rule, the joint distribution of A and B is:

$$p(a, b) = p(a) \cdot p(b|a)$$

The shorthand $p(a)$ means $p(A = a)$, that is, the probability of variable A taking value a .

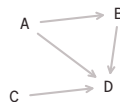
This is a simple example with two factors in the right-hand side. Add more variables, however, and the result can get messy fast. To see this, assume that there are two more variables, C and D , and that D is conditionally dependent on A , B , and C . The factorization becomes:

$$p(a, b, c, d) = p(a) \cdot p(b|a) \cdot p(c) \cdot p(d|a, b, c)$$

The relationship between variables is more opaque, hidden behind second-order dependencies. For example, while it's clear that D is directly dependent on A , we may miss the fact that there is another, second-order dependency between the two (D is dependent on B , which in turn is dependent on A).

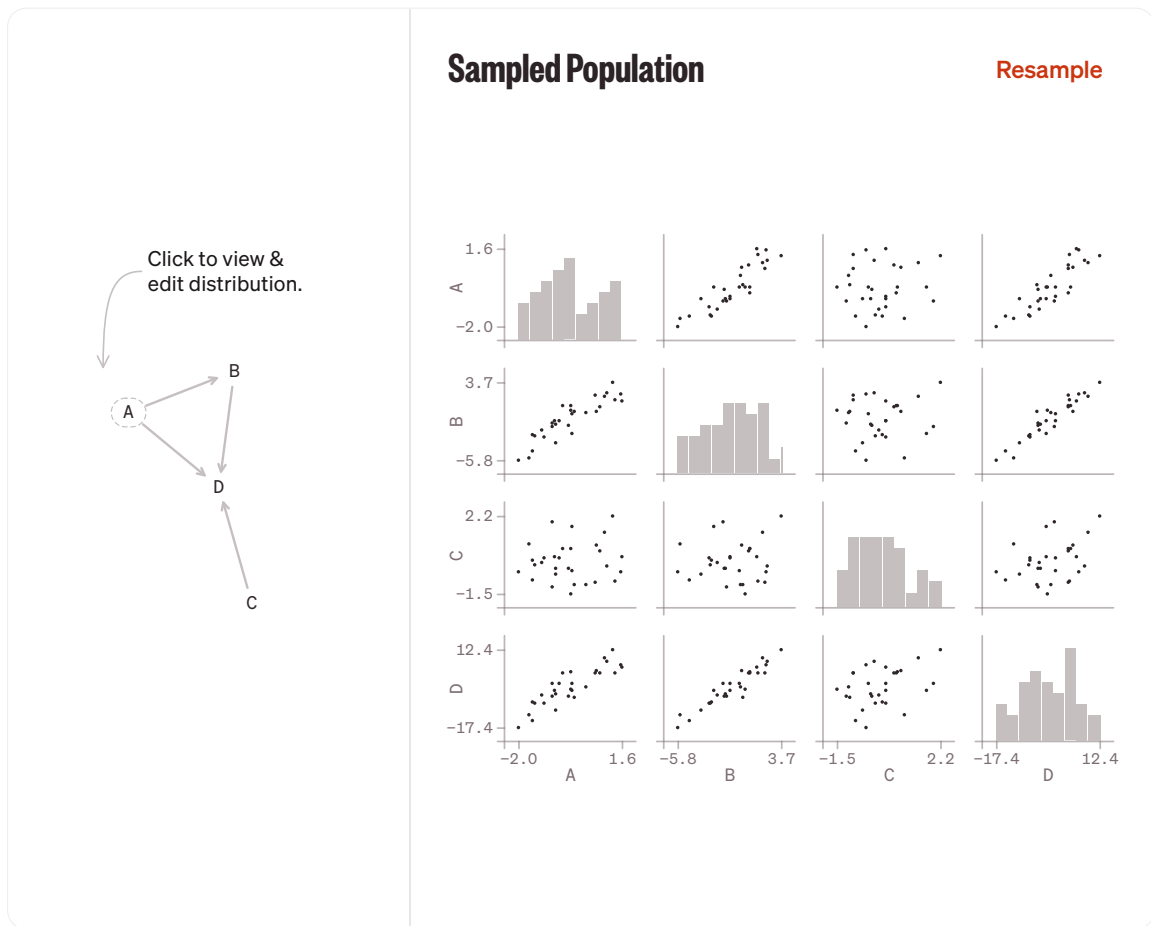
Directed Acyclic Graphs

Directed Acyclic Graphs, or DAGs, offer a natural remedy to this problem. Each factor in the equation can be represented by a node. An arrow indicates conditional dependence. The resulting graph would look like:



With this graph, it's easier to construct a story of how A , B , C and D are sampled. The process proceeds in topological order, for example $A \rightarrow C \rightarrow B \rightarrow D$, to ensure that all dependencies have been resolved by the time each variable is sampled.

Below is what a sampled population of the given distributions would look like. For the sake of demonstration, many distribution parameters are modifiable — in reality these are the quantities that need to be learned from training data.

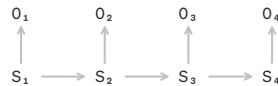


For more detailed accounts of probabilistic graphical models, consider reading the textbooks *Probabilistic Graphical Models: Principles and Techniques* by Daphne Koller and Nir Friedman [1] and *Probabilistic Reasoning in Intelligent Systems* by Judea Pearl [2].

II.

Hidden Markov Models

Hidden Markov Models (HMMs) are an early class of probabilistic graphical models representing partially hidden (unobserved) sequences of events. Structurally, they are built with two main layers, one observed (O_i) and one hidden (S_i):



HMMs have been successfully applied to a wide range of problems, including gene analysis [3], information extraction [4, 5], speech recognition [6], and named entity recognition [7].

The Hidden Layer

The hidden layer is assumed to be a Markov process: a chain of events in which each event's probability depends on the state of only the preceding event. More formally, given a sequence of N random events S_1, S_2, \dots, S_N , the Markov assumption states:

$$p(s_i | s_1, s_2, \dots, s_{i-1}) = p(s_i | s_{i-1})$$

for all $i \in \{2, \dots, N\}$

In a graph, this translates to a linear chain of events where each event has one arrow pointing towards it (except for the first event) and one pointing away from it (except for the last):



A second assumption that HMMs make is time-homogeneity: that the probability of transitioning from one state to the next is independent of time. For example, if the last state was “O”, then the probability of the current state being “B-LOC” is the same value $p(\text{B-LOC}|\text{O})$ regardless of what time step i we are at. In formal terms:

$$p(s_i | s_{i-1}) = p(s_{i+1} | s_i) \\ \text{for all } i \in \{2, \dots, N-1\}$$

$p(s_i | s_{i-1})$ is called the transition probability and is one of the two key parameters to be learned during training.

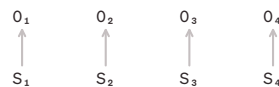
The assumptions about the hidden layer — Markov and time-homogeneity — hold up in various time-based systems where the hidden, unobserved events occur sequentially, one after the other. Together, they meaningfully reduce the computational complexity of both learning and inference.

The Observed Layer

The hidden and observed layers are connected via a one-to-one mapping. We assume the probability of each observation depends only on the state of the hidden event at the same time step. Given a sequence of N observed events O_1, O_2, \dots, O_N and N hidden events S_1, S_2, \dots, S_N we have:

$$p(o_i | s_1, s_2, \dots, s_N) = p(o_i | s_i) \\ \text{for all } i \in \{1, 2, \dots, N\}$$

In a graph, this one-to-one relationship looks like:



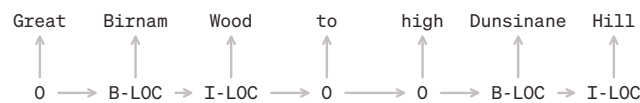
The conditional probability $p(o_i | s_i)$ is called the "emission probability." We also assume this is time-homogenous, further reducing the model's complexity. It is the second key parameter to be learned, alongside the transition probability.

Representing Named Entities

Throughout this article, we'll work with the CoNLL-2003 English dataset [8]. The dataset recognizes 9 possible name tags. Each, apart from the "O" tag, has either a "B-" (for beginning) or "I-" (for internal/inside) prefix, to eliminate confusion about when an entity stops and the next one begins. For example:

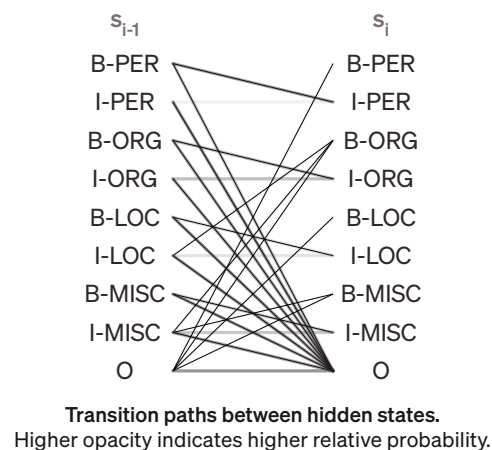


HMMs represent the input word tokens with the observed layer, with the respective name tags constitute the hidden layer:

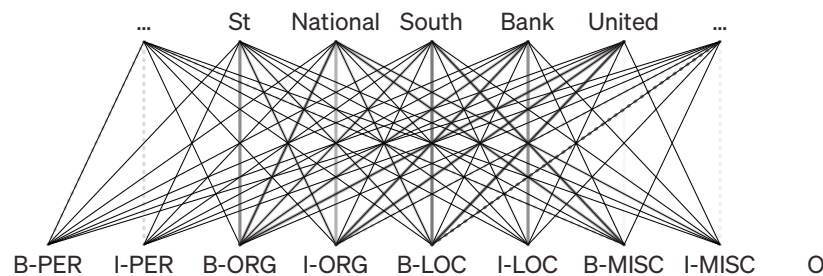


Rather than labeling each node using the name of the variable it represents (X_1, Y_1) as we have until this point, we'll instead display the value of that variable ("O", "Great"). This helps make the graphs easier to read.

Between any two consecutive hidden states, there are $9^2 = 81$ possible transitions. Each transition has its own probability, $p(s_i | s_{i-1})$:



In the observed layer, each node can have any value from the vocabulary, whose size ranges anywhere from the thousands to the hundreds of thousands. The vocabulary created for the HMM in this article contains 23,622 tokens. Let N be the number of tokens in the vocabulary. The number of possible emission probabilities is $9N$ ($n_{states} \cdot n_{tokens}$).



Emission paths from hidden states to observations.
Higher opacity indicates higher relative probability. Dashed lines are used for emission paths to other words in the vocabulary.

Training

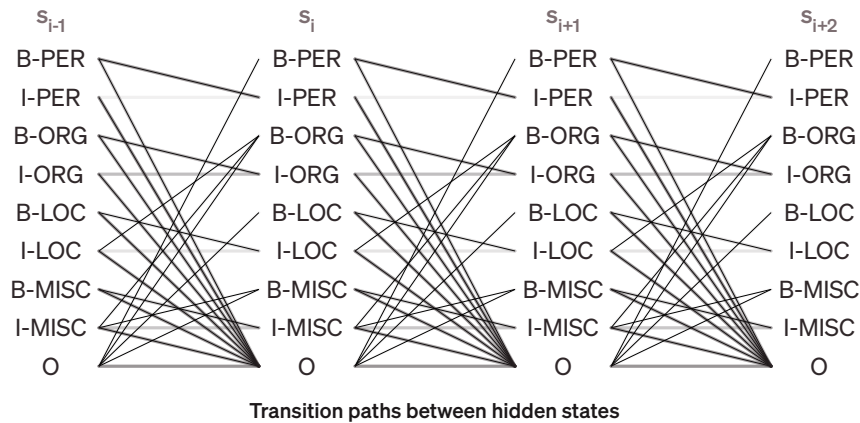
There are three sets of parameters to be learned during training: the transition, emission, and start probabilities. When the hidden states are known in the training set, which is the case with CoNLL-2003, all three can be computed as normalized rates of occurrence from the training data. For example, to get the transition probability from state "O" to state "B-LOC", $p(\text{B-LOC}|\text{O})$, we need two numbers: the number of times state "O" is followed by any other state (that is, it isn't the last state in the sequence), as N_{O} , and the number of times state "O" is followed by state "B-LOC", as $N_{\text{O} \rightarrow \text{B-LOC}}$. The desired transition probability is $\frac{N_{\text{O} \rightarrow \text{B-LOC}}}{N_{\text{O}}}$. The same calculation can be done for each of the remaining probabilities.

If however the hidden states are unknown, then the standard approach is to use the Baum-Welch (forward-backward) algorithm. Baum-Welch is an iterative algorithm, where we start with an initial estimate of the probabilities to be learned and use them to compute progressively better estimates. An excellent explanation of the algorithm can be found in the the book *Speech and Language Processing* by Jurafsky & Martin [9].

Inference

In the context of HMMs, inference involves answering useful questions about hidden states given observed values, or about missing values given a partially-observed sequence. In NER, we are focused on the first type of inference. Specifically, we want to perform maximum a posteriori (MAP) inference to identify the most likely state sequence conditioned on observed values.

There is usually an intractably large number of candidate state sequences. For any two consecutive states, there are $9^2 = 81$ potential transition paths. For three states there are $9^3 = 729$ paths. This number continues to grow exponentially as the number of states increases.



Luckily, there is an efficient dynamic algorithm that returns the most likely path with relatively low computational complexity: the Viterbi algorithm [10]. It moves through the input sequence from left to right, at each step identifying and saving the most likely path in a trellis-shaped memory structure. For more details, refer to the Jurafsky & Martin book [11].

Results

An HMM with the structure outlined above was trained on the CoNLL-2003 dataset. The training set contains 14,987 sentences and a total of 203,621 word tokens. Here's the model in action:

Name tag predictions by HMM:

The London-to-Boston route



| | | | |
 O B-LOC O O O B-LOC O

O = not a name ORG = organization PER = person LOC = location MISC = miscellaneous

Evaluated against a test set, the model achieves satisfactory per-word accuracy:

Accuracy

90.1%

However, precision and recall — calculated per entity [8] — are decidedly low:

Precision	Recall	F ₁ Score
64.2%	55.8%	59.7%

These metrics are lower than per-word accuracy because they are entity-level evaluations that count only exact matches as true positives. Long, multi-word entities are considered incorrect if one or more of their constituent words are misidentified, in effect, ignoring the other correctly identified words in the entity.

A closer look at the results reveals a discrepancy between entities with known words versus those containing at least one out-of-vocabulary (OOV) word. It is common for proper names to contain one or more OOV words, such as the "Pertile" in "Javier Pertile". Both precision and recall are significantly worse in those cases:

Precision			Recall		
ORG	0.8	0.21	ORG	0.64	0.33
PER	0.85	0.62	PER	0.58	0.59
LOC	0.87	0.06	LOC	0.71	0.05
MISC	0.78	0.12	MISC	0.54	0.06
ALL	0.84	0.39	ALL	0.64	0.41
No OOV		1+ OOV	No OOV		1+ OOV

This makes sense: we expect the model to perform worse on tokens that it wasn't trained on. And because the CoNLL-2003 training set has a large number of OOV tokens — 80.6% of all test entities contain at least one OOV token — across-the-board precision and recall are heavily impacted.

Limitations

HMMs are by no means perfect candidates for NER or text labeling problems in general, for at least three reasons.

First is the Markov assumption. We don't compose words and their respective labels ("Noun", "Adjective", "B-ORG", "I-ORG") in a tidy, unidirectional manner. A word may link to another word many positions in the sentence before or after. In "South African Breweries Ltd", both "South African" and "Ltd" provide crucial context to "Breweries", which would otherwise register as not a name. HMMs fail to capture these tangled interdependencies, instead assuming that information passes from left to right in a single, neat chain.

An indication of such limitation lies in the fast deterioration of the precision score as the entity length — counted as the number of tokens inside each entity — increases (the recall scores are much more noisy and thus less conclusive):

		Precision		Recall	
		1	2	3	4
		0.61	0.68	0.3	0.12
ORG					0.28
PER		0.96	0.67	0.7	
LOC		0.88	0.36		
MISC		0.9	0.46	0.24	0.12
ALL		0.77	0.61	0.31	0.12
					0.29
		1	2	3	4
					5
					Entity Length

An empty cell means there were not enough (fewer than five) relevant entities. E.g. in the precision tab, there were not enough entities predicted to be a location name that were five words long.

The second reason why HMMs fail to do well on text labeling problems is the emissions assumption. When composing sentences, we don't go about forming a chain of labels (such as "B-ORG" – "I-ORG" – "I-ORG" – "O"...) before generating words from each label in that chain.

Semantic and grammatical rules may restrict the range of words that can appear in any given observation, but those restrictions are far from the strong emissions assumption made by HMMs. Beyond the current word label, there is a host of other features that, together, help determine which words are chosen. Additionally, while there is a link between part-of-speech tags and the words that are chosen, that link is less clear with name tags.

The third reason: HMMs rely solely on word identity (the exact characters that make up the word) in the observed layer. There are various word features that can provide additional information on the hidden label. Capitalization is a strong signal of named entities. Word shape may help account for common name patterns. HMMs take none of these features into account (in fact, they can't: their generative chain structure requires that all observations be independent of each other).

Since the only information available are word identities, HMMs falter on out-of-vocabulary words. While the models can wring bits of useful information out of nearby predicted name tags — the Viterbi algorithm maximizes likelihood over the whole sequence — the results are nonetheless discouraging:



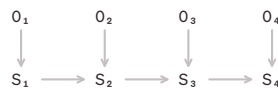
An empty cell means either it is impossible (e.g. a two-word entity can't have an OOV rate of 0.25) or there were not enough (fewer than five) relevant entities to form a reliable score.

The concerns detailed above are a consequence of HMMs' generative graph structure. Below, we'll consider another class of probabilistic graphical models with a different, discriminative structure that will hopefully address some of those concerns and deliver better performance on long, OOV-heavy entities.

III.

Maximum Entropy Markov Models

Maximum Entropy Markov Models (MEMMs) [12] resolve some of the concerns we had with HMMs by way of a simple yet meaningful modification to their graphical structure:



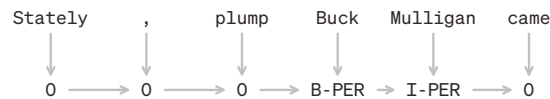
The arrows connecting observations with their respective states have switched directions. We'll discuss the implications of such a change in the sections below.

Similar to HMMs, MEMMs have been successfully applied to a wide range of sequence modeling problems [12, 13, 14].

Discriminative Structure

There are two main approaches to building classification models: generative and discriminative [15]. Suppose there is a system with two variables, X and Y . We want to make predictions about Y based on observations on X . A generative classifier would do that by first learning the prior distribution $p(X, Y)$ and then applying Bayes' rule to find the posterior $p(Y|X)$. This can be thought of as reconstructing the process that generated the observed data. A discriminative classifier, on the other hand, would model the posterior $p(Y|X)$ directly based on training data.

HMMs are generative classifiers, while MEMMs are discriminative. The former are generative because they model the joint distribution over both observations and hidden states (as a product of transition and emission probabilities) before using that joint distribution to find the most likely state sequence given observations (or solve some other inference problem). MEMMs, on the other hand, directly model the conditional probabilities $p(\text{state}|\text{observation}, \text{prevstate})$ without any intermediary.



Word Features

A major advantage of MEMMs' discriminative structure is that it permits overlapping word features as inputs. Two features can overlap when they contain the same or similar pieces of information, such as capitalization and word shape (e.g. "Xxxx", for four-letter words where the first character is capitalized). HMMs don't allow overlapping features. As a result of their generative structure, they require that all events in the observation layer be independent of one another. MEMMs, on the other hand, are discriminative and able to relax the independence requirement, so they can use arbitrary overlapping features [12].

Common practice is to represent features as binary functions, such as:

$$b_{\text{shape=Xxxx}}(o_t) = \begin{cases} 1 & \text{if } o_t \text{ has shape "Xxxx"} \\ 0 & \text{otherwise} \end{cases}$$

Applied to the example above, we have:

o_t	$b_{\text{shape=Xxxx}}(o_t)$
Stately	0
,	0
plump	0
Buck	1
Mulligan	0
came	0

Binary features are paired with the current state s to form feature-state pairs $a = \langle b, s \rangle$, also expressed as binary functions:

$$f_a(o_t, s_t) = f_{\langle b, s \rangle}(o_t, s_t) = \begin{cases} 1 & \text{if } b(o_t) = 1 \text{ and } s_t = s \\ 0 & \text{otherwise} \end{cases}$$

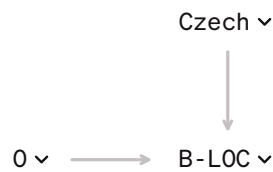
Feature-state pairs help MEMMs learn which features and states go together and which don't. For example, we can expect pairs like "is_capitalized" + "B-ORG" to frequently co-occur, capturing the fact that in English named entities are often capitalized.

State Transitions

MEMMs' state transition distributions have exponential form and contain a weighted sum of all feature-state pairs:

$$p_{s'}(s|o) = \frac{1}{Z(o, s')} \exp \left(\sum_a \lambda_a f_a(o, s) \right)$$

where s' and s are the previous and current state, o is the current observation, $a = \langle b, s \rangle$ is a feature-state pair, λ_a is the learned weight for a , and $Z(o, s')$ is a normalizing term to make the distribution $p_{s'}$ sum to one across all next states s .



Calculating $p_0(\text{B-LOC} \mid \text{"Czech"})$

With weights retrieved from a MEMM trained on CoNLL-2003 data. Numbers are rounded to 3 decimal places for clarity.

Feature-State Pair (a)	λ_a	f_a	$\lambda_a f_a$
lowercase=czech – B-LOC	1.208	1	1.208
word=Czech – B-LOC	1.193	1	1.193
shape=Xxxxx – B-LOC	0.281	1	0.281
is_title_case – B-LOC	0.217	1	0.217
is_not_uppercase – B-LOC	-0.643	1	-0.643
is_not_digit – B-LOC	-0.522	1	-0.522
word=win – 0	0.439	0	0
lowercase=pour – 0	0.424	0	0
.....
		SUM($\lambda_a f_a$)	1.734

$$\begin{aligned}
 p_0(\text{B-LOC} \mid \text{"Czech"}) &= e^{\text{SUM}(\lambda_a f_a)} / Z \\
 &\approx e^{1.734} / 7.276 \\
 &\approx \mathbf{0.778}
 \end{aligned}$$

Those familiar with neural networks will recognize that the function above is a softmax. Its exponential form is a result of the core principle of maximum entropy that underlies MEMMs' statistical structure and gives them their name. Maximum entropy states that the model that best represents our knowledge about a system is one that makes the fewest possible assumptions except for certain constraints derived from prior data from that system [12, 16].

Training & Inference

The training step involves learning the weights λ_a that satisfy MEMMs' maximum entropy constraint [12]. Learning is done through Generalized Iterative Scaling, which iteratively updates the values λ_a in order to nudge the expected value of all features closer to their training set average. Convergence at a global optimum is guaranteed given the exponential form of the transition distribution.

As with HMMs, the Viterbi algorithm makes MAP inference tractable [12, 10]. The variable transition probability $p_s(s|\phi)$ takes the place of HMMs' fixed transition and emission probabilities.

Results

A MEMM was trained on the CoNLL-2003 English dataset [8]. In addition to word identity, features used for training include the word's lowercase version ("Algeria" → "algeria"), shape ("Xxxx"), whether it's in title/upper case, and whether it contains only digits.

A list of the most informative features — those with the largest absolute weights — offers valuable insights into how the model found and remembers linguistic patterns:

Most Informative Features when Previous State is 0 ▾

Current Word Feature	Current State	Weight
word=germany	B-LOC	11.492
word=van	B-PER	8.972
word=wall	B-ORG	8.525
word=della	B-PER	7.86
lowercase=della	B-PER	7.86
is_not_title_case	B-PER	-6.949
word=de	B-PER	6.781
shape=X.X.	O	-6.713
shape=xxxx	B-ORG	-6.642
word=CLINTON	B-ORG	6.456

Many of these features are word identities. This makes intuitive sense: certain words, like "Germany", are almost always used as names irrespective of what comes before or after them.

Other features relate to established linguistic patterns. For example, if the current word has shape "X.X.", such as "U.S." and "U.N.", it's unlikely to have the "O" tag — the feature-state pair's weight is a large negative number. This means the word is likely a named entity, most probably two-letter initialisms.

Here's a live version of the trained model:

Name tag predictions by MEMM:

Chicago Board of Trade



B-ORG I-OR I-OR I-ORG

O = not a name ORG = organization PER = person LOC = location MISC = miscellaneous

The model has better performance than its HMM counterpart. Per-word accuracy is higher than the HMM's 90.1%:

Accuracy

93.1%

Per-entity precision and recall are notably higher, up from the HMM's 64.2% and 55.8%, respectively:

Precision

72.9%

Recall

63.5%

F₁ Score

67.9%

A large part of the performance boost is attributable to higher precision on entities with at least one OOV word:

		MEMM		HMM	
		Precision		Recall	
ORG	0.81	0.36	ORG	0.68	0.12
PER	0.82	0.8	PER	0.72	0.57
LOC	0.82	0.17	LOC	0.89	0.29
MISC	0.74	0.14	MISC	0.78	0.02
ALL	0.8	0.54	ALL	0.79	0.37
	No OOV	1+ OOV		No OOV	1+ OOV

Advantage Over HMMs

The ability to model word features allows MEMMs to fare better with OOV-dense name entities than HMMs. Faced with words that they have never seen before during training, these models can easily stumble. Word identity alone provides no useful information. In those cases, derived features such as word shape and capitalization can function as imperfect yet doubtlessly helpful proxies for word identity, allowing MEMMs to make better guesses at the name tag, resulting higher precision and recall scores:



An empty cell means either it is impossible (e.g. a two-word entity can't have an OOV rate of 0.25) or there were not enough (fewer than five) relevant entities to form a reliable score.

With stronger predictive power on OOV words, we can additionally expect better performance on long, multi-word entities. That's because OOV words are dangerous information gaps inside named entities. They're easy to misclassify, and when they are the entire entity prediction is counted as incorrect. MEMMs are able to fill those gaps to an extent by using word features. As a result, we don't see the drastic performance deterioration for longer entities observed with the HMM:



An empty cell means there were not enough (fewer than five) relevant entities. E.g. in the precision tab, there were not enough entities predicted to be a location name that were five words long.

Label Bias Problem

MEMMs' discriminative structure offers great benefits as demonstrated above, but it comes with a downside: the label bias problem. First reported by Bottou [17], label bias mostly affects discriminative models, causing certain states to effectively ignore their observations, leading to demonstrably higher error rates [18].

Here's how it happens. Recall that MEMMs model state transitions as probability distributions $p_{s'}(s|o)$. For these probabilities to be valid, the distributions must sum up to one:

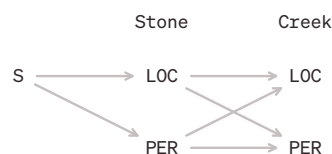
$$\sum_s p_{s'}(s|o) = 1$$

This requirement is satisfied by *local* normalization via the term $Z(o, s')$ baked directly into the definition of $p_{s'}(s|o)$:

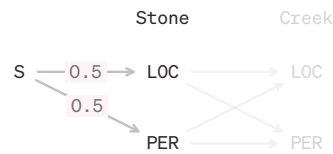
$$p_{s'}(s|o) = \frac{1}{Z(o, s')} \exp \left(\sum_a \lambda_a f_a(o, s) \right)$$

The normalization is “local” since it's applied to individual parts of the graph (in this case, the transition between two states) as opposed to the whole structure. It's here that the problem arises: scores leaving a state are rescaled to become proper probabilities, and in the process information about how likely the transitions are under the given observations is erased. A simple example can help illustrate this effect.

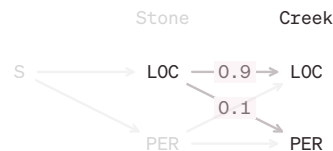
Let's assume that there are only two hidden states in our model: “LOC” if the word is part of a location name, and “PER” if it's part of a person's name. There's a special state “S” for the start of the sentence. We observe the two-word sequence “Stone Creek”. The transition graph is the following:



At inference time we proceed from left to right. The first observation is “Stone”. We need to find $p_S(\text{LOC}|\text{“Stone”})$ and $p_S(\text{PER}|\text{“Stone”})$. Let’s say that they both have the same value, 0.5:



We then go to the second observation, “Creek”. The first pair of transition probabilities that we need to calculate is $p_{\text{LOC}}(\text{LOC}|\text{“Creek”})$ and $p_{\text{LOC}}(\text{PER}|\text{“Creek”})$. Under local normalization, the two must sum up to one. Let’s say that they come out to 0.9 and 0.1 respectively.



So far so good. Now we need to calculate $p_{\text{PER}}(\text{LOC}|\text{“Creek”})$ and $p_{\text{PER}}(\text{PER}|\text{“Creek”})$. Let’s say probabilities are, in turn, 0.05 and 0.95.



Now, intuitively the word “Creek” is much more likely to be part of a location name than a person’s name. But here the first set of transition probabilities, $p_{\text{LOC}} = \{0.9, 0.1\}$, have the same total mass as the second set, $p_{\text{PER}} = \{0.05, 0.95\}$. Both sum up to one because of local normalization. The result is that taken together, the model judges paths that go through the “LOC” state first (i.e. “S” → “LOC” → “LOC” and “S” → “LOC” → “PER”) to be about as likely as those that go through the “PER” state first (“S” → “PER” → “LOC” and “S” → “PER” → “PER”), contradicting our intuition about “Creek” being more consistent with the former than the latter. The model has in effect ignored crucial information contained within the observation “Creek”.

More generally, due to MEMMs' discriminative structure, observations can only affect how the incoming probabilities are distributed across the possible current states, but not how much total probability mass to pass on. That total mass must always equal one, due to local normalization. In the example above, given the previous state "PER", the observation "Creek" may shift the distribution over the current state toward $\{0.05, 0.95\}$ rather than some other split like $\{0.1, 0.9\}$, but it cannot change the total probability mass that the previous state "PER" passes on. We can't have the transition probability pair be, say, $\{0.01, 0.19\}$, which would better reflect our intuition about "PER" being less likely under "Creek".

So that is the crux of the label bias problem. Local probability normalization causes MEMMs to discard useful information about how likely state transitions are under a given observation. The problem extends beyond the simple case above. Hannun [19] offers an excellent, more detailed explanation of the problem and how it generalizes to more scenarios and model structures.

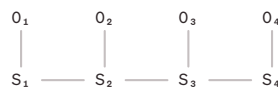
There have been various proposed solutions to the label bias problem. One particularly successful strategy is to eschew local normalization in favor of normalization at the global, graph-wide level. That brings us to Conditional Random Fields, a class of globally normalized, undirected probabilistic models, which we'll cover in the next section.

IV.

Conditional Random Fields

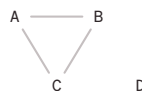
Conditional Random Fields (CRFs) are a class of undirected probabilistic models. They have proved to be powerful models with a wide range of applications, including text processing [18, 20, 21], image recognition [22, 23, 24, 25], and bioinformatics [26, 27].

While CRFs can have any graph structure, in this article we'll focus on the linear-chain version:



Markov Random Fields

CRFs are a type of Markov Random Fields (MRFs) — probability distributions over random variables described by *undirected* graphs [28], for example:



Undirected graphs are appropriate when it's difficult or implausible to establish causal, generative relationships between random variables. Social networks are a good example of undirected relationships. We can think of A , B , C , and D in the graph above as people in a simple network. A and B are friends and tend to share similar beliefs. The same goes for B and C as well as C and A . We might, for example, want to model how each person in the network thinks about a specific topic.

Acyclic Directed Graphs fail to adequately represent the mutual belief propagation that occurs within the group. For example, we might have an edge from A to B but no path from B back to A — there will always be at least one such exclusion in an acyclic directed graph.

Rather than assuming a generative relationship between variables, MRFs model their mutual relationships with non-negative scoring functions ϕ , called *factors*, that assign higher scores if the variables' values are in agreement, for example:

$$\phi(X, Y) = \begin{cases} 3 & \text{if } X = 1 \text{ and } Y = 1 \\ 2 & \text{if } X = 0 \text{ and } Y = 0 \\ 1 & \text{otherwise} \end{cases}$$

Unlike conditional probabilities, there is no assumed directionality in scoring functions. These functions simply return higher scores if the variables “agree” and lower scores if they “disagree” with one another. They model pairwise correlation, not causation.

The joint probability of all variables in the graph is defined as:

$$p(A, B, C, D) = \frac{1}{Z} \phi(A, B) \phi(B, C) \phi(C, A)$$

where Z is a normalization factor

The factors ϕ promote assignments in which their constituent variables (A and B in the case of $\phi(A, B)$) agree with each other. The assignment 1-1-1-0 (for A, B, C , and D respectively) would receive a higher score and thus have higher probability than say 1-1-0-0, since there is more agreement in the former case.

Note that D isn't present in the factorization, which makes intuitive sense. D is separated from A, B , and C — there is no edge connecting it to those other variables. Extending the social network analogy, D isn't friends with the trio of A, B , and C , so it isn't more or less likely to agree with that group (e.g. the assignment 1-1-1-1 is as likely as 1-1-1-0). Given that, it makes sense that the joint probability $p(A, B, C, D)$ does not depend on the value of D .

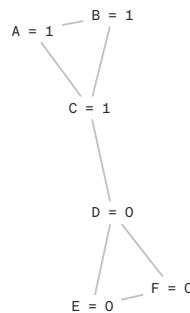
More generally, MRFs are probability distributions p over random variables x_1, x_2, \dots that are defined by an undirected graph G and have the form:

$$p(x_1, x_2, \dots) = \frac{1}{Z} \prod_{c \in C} \phi_c(x_c)$$

where Z is a normalization term
and C is the set of cliques in G

The definition relies on the concept of cliques: groups of nodes in a graph that are fully connected (forming a complete subgraph). A , B , and C in the example above form a clique, so do A and B , B and C , and so on. We don't need to define a factor ϕ_c for every clique c . We may, for example, skip any clique containing only one node.

Here is an example of how the joint probability is calculated:



Calculating $p(A, B, C, D, E, F)$

$$p(1, 1, 1, 0, 0, 0)$$

$$\begin{aligned}
 &= 1/Z \\
 &\times \phi_{ABC}(1, 1, 1) \\
 &\times \phi_{AB}(1, 1) \\
 &\times \phi_{BC}(1, 1) \\
 &\times \phi_{AC}(1, 1) \\
 &\times \phi_{CD}(1, 0) \\
 &\times \phi_{DEF}(0, 0, 0) \\
 &\times \phi_{DE}(0, 0) \\
 &\times \phi_{EF}(0, 0) \\
 &\times \phi_{DF}(0, 0)
 \end{aligned}$$

$$\begin{aligned}
 &= 1 / 28,915 \\
 &\times 3 \\
 &\times 3 \\
 &\times 3 \\
 &\times 3 \\
 &\times 1 \\
 &\times 2 \\
 &\times 2 \\
 &\times 2 \\
 &\times 2
 \end{aligned}$$

$$\approx \mathbf{0.0448}$$

$$\phi_{ABC} = \phi_{AB} = \phi_{\dots}(x) = \begin{cases} 3 & \text{if } x_1 = x_2 = \dots = 1 \\ 2 & \text{if } x_1 = x_2 = \dots = 0 \\ 1 & \text{otherwise} \end{cases}$$

MRFs have a generalized form of which the directed models we've seen so far — HMMs and MEMMs — are special cases. The factors ϕ_c can be defined as conditional probabilities, for example $\phi_c(x_1, x_2) = p(x_2|x_1)$, and act as the transition and emission probabilities that characterize HMMs and MEMMs.

The additional level of generality comes at a cost, however: the normalization term Z is often difficult to compute. They require summing over an exponential number of potential assignments, an infeasible task if the network is large enough. Fortunately, there are configurations that can be solved using efficient decoding algorithms. That includes linear-chain CRFs, which can be decoded with the Viterbi algorithm.

Conditional Form

CRFs are random fields globally conditioned on a set of observations x [18] and have the form:

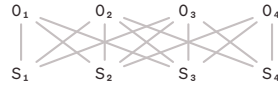
$$p(y|x) = \frac{1}{Z(x)} \prod_{c \in C} \phi_c(y_c, x_c)$$

where Z is a normalization term
and C is the set of cliques in the
graph G representing the labels y

The distribution $p(y|x)$ is parameterized by x . When we replace all the values x_i in the right hand side with real values, what remains has the same form as an MRF. In fact, we get a new MRF for every observation sequence x .

CRFs are globally conditioned on x [18]. They directly model the probability of the entire label sequence y — $p(y|x)$ — rather than local transition/emission probabilities $p(y_i|y_{i-1})$ or $p(y_i|x_i)$.

Global conditioning on x means that the hidden states y_i can depend not only on the current observation but also any other observation in the sequence. Adding more such dependencies to the model does not increase the computational complexity of inference tasks, since we don't have to model the marginal probabilities $p(x_i)$ at train/test time.



Linear-chain CRF where the hidden layer depends on the current, previous, and future observations.

Exponential Factors

While the factors ϕ_c can be any real-valued non-negative function, they usually have exponential form similar to MEMMs' transition functions [29]:

$$\phi_c(y_c, x_c) = \exp \left(\sum_a \lambda_a f_a(y_c, x_c) \right)$$

where f_a is a feature function defined for clique c
and λ_a is the weight parameter for f_a

Like with MEMMs, we can define the feature functions f_a as binary indicators for the current state or observation. One such indicator may be “the previous state is B-LOC and the current state is I-LOC”. Then at train time we can learn the weight parameters λ_a that best fit the given data.

Training & Inference

During training, estimation of the weight vector λ is performed by likelihood maximization. The process has an intuitive and elegant interpretation: by maximizing the training data’s likelihood, we’re also brining the expected value of the feature functions f_a closer to their train-set averages [29]. This is analogous to the maximum-entropy interpretation of parameter estimation in MEMMs [12].

In the case of linear-chain CRFs, inference can be done efficiently using modified versions of the forward, backward, and Viterbi algorithm [29]. Since the distribution $p(y|x)$ is a product of factors ϕ_c , we can progresively move through the chain, going node to node, adding more factors to the cumulative product. The factors ϕ_c play the same role in these algorithms as the transition and emission probabilities of HMMs and MEMMs.

Addressing the Label Bias Problem

Global conditioning on x helps CRFs avoid the label bias problem [12]. As discussed above, the problem mostly affects models with local probability normalization, including MEMMs [19]. CRFs avoid this since they model the probability of the entire state sequence $p(y|x)$, which has a single, global normalization term Z :

$$p(x_1, x_2, \dots) = \frac{1}{Z} \prod_{c \in C} \phi_c(x_c)$$

where Z is a normalization term
and C is the set of cliques in G


CRFs' robustness against label bias, at least compared to MEMMs, has been well demonstrated in experimental conditions [12]. In practical applications, however, it's more difficult to capture the presence and effect of label bias due to a myriad of confounding factors. Still, we can rule out potential confounders and try to isolate label bias in our test systems. We do so in the next section.

Results

A test linear-chain CRF was trained using the same data and features as the MEMM in the previous section. Each hidden state in the CRF depends only on the current observation, forming a unary chain structure similar to the MEMM's. (CRF states can depend on any observation, this test version on uses the current observation for a better comparison with the MEMM.)

Here's a live version of the trained model:

Name tag predictions by unary CRF:



B-MISC **O** **B-PER** **I-PER**

0 = not a name ORG = organization PER = person LOC = location MISC = miscellaneous

The resulting unary CRF closely resembles the baseline MEMM. Both have exponential form with a weighted sum of all word features as the exponent. The only meaningful difference between the two is how normalization is done: globally for the CRF and locally for the MEMM. Since the label bias problem comes from local normalization, we should see more bias, observable as errors, in the MEMM than the CRF.

Overall, the unary CRF strongly outperforms the MEMM. While it has only slightly higher accuracy (compared to the MEMM's 93.1%):

Accuracy

94.8%

...the CRF makes predictions with significantly higher precision and recall:

Precision	Recall	F ₁ Score
77.8%	74.9%	76.3%

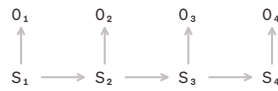
The performance gains are spread out across states and OOV status:

		CRF (Unary)		MEMM	
		Precision		Recall	
ORG	0.87	0.46	ORG	0.79	0.4
PER	0.87	0.72	PER	0.82	0.8
LOC	0.9	0.29	LOC	0.92	0.2
MISC	0.81	0.23	MISC	0.84	0.09
ALL	0.87	0.6	ALL	0.85	0.57
	No OOV	1+ OOV		No OOV	1+ OOV

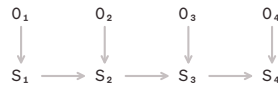
Since both models are trained with the same dataset, features, and unary structure, the increased performance is attributable to global normalization. While these results are not a definitive proof of CRFs' resistance to the label bias problem — global normalization could help improve model performance in other ways, it is nonetheless highly encouraging.

V. Conclusion

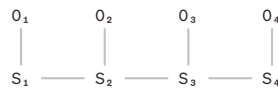
Throughout this article, we've charted the successive development of three families of graphical models, all united for the common task of named entity recognition. We've demonstrated how in various aspects, each was an improvement over the one before. First came HMMs with their simple, intuitive generative structure:



Then MEMMs came along with a discriminative structure, which helps it relax the strict requirement of independence between input variables still held by HMMs:



CRFs, in turn, were developed with an undirected structure and global probability normalization to specifically resolve the label bias problem afflicting MEMMs, thus improving overall performance, especially for long named entities:



Along the way, we see a gradual trend toward fewer assumptions (independence between observations, directionality) and therefore greater expressivity from one model to the next.

Here are all three once again in action:

Which of these words refer to named entity?

Suu Kyi's National League for Democracy
 ↺

	Suu	Kyi's	National	League	for	Democracy	
<u>HMM</u>	PER	PER	0	MISC	MISC	0	0
<u>MEMM</u>	0	0	0	ORG	ORG	0	0
<u>CRF</u>	PER	PER	0	ORG	ORG	ORG	ORG

0 = not a name ORG = organization PER = person LOC = location MISC = miscellaneous

The work continues. CRFs remain a powerful tool for not only language modeling but also various image recognition tasks [29]. In 2016, Zheng et al. [24] reinterpreted CRFs as recurrent neural networks (CRF-RNNs) and demonstrated their effectiveness in semantic image segmentation — an encouraging result and compelling example of the theoretical breadth of CRFs specifically and of graphical models more generally.

We hope that this article has contributed to that understanding. Applied to the right set of problems, probabilistic graphical models are a powerful framework for statistical modeling. Their intuitive structure and high interpretability belie their great representational power.

References

1. **Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning**
Daphne Koller and Nir Friedman. 2009. The MIT Press. ↩
2. **Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference**
Judea Pearl. 1988. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA. <https://doi.org/10.1016/C2009-0-27609-4> ↩
3. **Genes, Themes and Microarrays: Using Information Retrieval for Large-Scale Gene Analysis**
Hagit Shatkay, Stephen Edwards, W John Wilbur, and Mark Boguski. 2000. In Proceedings of the International Conference on Intelligent Systems for Molecular Biology, 317–328. ↩
4. **Information Extraction Using Hidden Markov Models**
Timothy Robert Leek. 1997. Master's Thesis, UC San Diego. ↩
5. **Information Extraction with HMMs and Shrinkage** — [PDF](#)
Dayne Freitag and Andrew McCallum. 1999. In Papers from the AAAI-99 Workshop on Machine Learning for Information Extraction (AAAI Technical Report WS-99-11), 31–36. ↩
6. **A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition**
Lawrence R Rabiner. 1989. Proceedings of the IEEE 77, 2: 257–286. <https://doi.org/10.1109/5.18626> ↩
7. **An Algorithm that Learns What's in a Name** — [PDF](#)
Daniel M. Bikel, Richard Schwartz, and Ralph M. Weischedel. 1999. Machine Learning 34, 1: 211–231. <https://doi.org/10.1023/A:1007558221122> ↩
8. **Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition** — [PDF](#)
Erik F. Tjong Kim Sang and Fien De Meulder. 2003. In Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003, 142–147. <https://doi.org/10.48550/arXiv.cs/0306050> ↩ ↩ ↩
9. **Appendix A.5 — HMM Training: The Forward-Backward Algorithm** — [PDF](#)
Daniel Jurafsky and James H. Martin. 2021. In Speech and Language Processing. 8–10. ↩
10. **Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm**
A. Viterbi. 1967. IEEE Transactions on Information Theory 13, 2: 260–269. <https://doi.org/10.1109/TIT.1967.1054010> ↩ ↩
11. **Appendix A.4 — Decoding: The Viterbi Algorithm** — [PDF](#)
Daniel Jurafsky and James H. Martin. 2021. In Speech and Language Processing. 8–10. ↩
12. **Maximum Entropy Markov Models for Information Extraction and Segmentation** — [PDF](#)
Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. 2000. In Proceedings of the Seventeenth International Conference on Machine Learning (ICML '00), 591–598. ↩ ↩ ↩ ↩ ↩ ↩
13. **Maximum Entropy Models for Antibody Diversity** — [Link](#)
Thierry Mora, Aleksandra M. Walczak, William Bialek, and Curtis G. Callan. 2010. Proceedings of the National Academy of

14. **Human Behavior Modeling with Maximum Entropy Inverse Optimal Control** — [PDF](#)
Brian Ziebart, Andrew Maas, J. Bagnell, and Anind Dey. 2009. In Papers from the 2009 AAAI Spring Symposium, Technical Report SS-09-04, Stanford, California, USA, 92–97.
15. **On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes** — [PDF](#)
Andrew Ng and Michael Jordan. 2001. In Advances in Neural Information Processing Systems.
16. **Inducing Features of Random Fields**
S. Della Pietra, V. Della Pietra, and J. Lafferty. 1997. IEEE Transactions on Pattern Analysis and Machine Intelligence 19, 4: 380–393. <https://doi.org/10.1109/34.588021>
17. **Une Approche théorique de l'Apprentissage Connexionniste: Applications à la Reconnaissance de la Parole**
Léon Bottou. 1991. Université de Paris X.
18. **Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data** — [PDF](#)
John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. In Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01), 282–289.
19. **The Label Bias Problem** — [Link](#)
Awni Hannun. 2019. Awni Hannun — Writing About Machine Learning.
20. **Discriminative Probabilistic Models for Relational Data** — [Link](#)
Ben Taskar, Pieter Abbeel, and Daphne Koller. 2013. <https://doi.org/10.48550/ARXIV.1301.0604>
21. **Accurate Information Extraction from Research Papers using Conditional Random Fields**
Fuchun Peng and Andrew McCallum. 2004. In Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004, 329–336. <https://doi.org/10.1016/j.ipm.2005.09.002>
22. **Discriminative Fields for Modeling Spatial Dependencies in Natural Images** — [PDF](#)
Sanjiv Kumar and Martial Hebert. 2003. In Advances in Neural Information Processing Systems.
23. **Multiscale Conditional Random Fields for Image Labeling** — [Link](#)
Xuming He, R.S. Zemel, and M.A. Carreira-Perpinan. 2004. In Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004, CVPR 2004, II–II. <https://doi.org/10.1109/CVPR.2004.1315232>
24. **Conditional Random Fields as Recurrent Neural Networks** — [Link](#)
Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip H. S. Torr. 2015. In 2015 IEEE International Conference on Computer Vision (ICCV). <https://doi.org/10.1109/iccv.2015.179>
25. **Convolutional CRFs for Semantic Segmentation** — [Link](#)
Marvin T. T. Teichmann and Roberto Cipolla. 2018. <https://doi.org/10.48550/ARXIV.1805.04777>

26. RNA Secondary Structural Alignment with Conditional Random Fields — [Link](#)

Kengo Sato and Yasubumi Sakakibara. 2005. Bioinformatics 21: ii237–ii242. <https://doi.org/10.1093/bioinformatics/bti1139>

27. Protein Fold Recognition Using Segmentation Conditional Random Fields (SCRFS)

Yan Liu, Jaime Carbonell, Peter Weigele, and Vanathi Gopalakrishnan. 2006. J. Comput. Biol. 13, 2: 394–406. <https://doi.org/10.1089/cmb.2006.13.394>

28. Introduction to Markov Random Fields

Andrew Blake and Pushmeet Kohli. 2011. In Markov Random Fields for Vision and Image Processing. The MIT Press.

29. An Introduction to Conditional Random Fields — [Link](#)

Charles Sutton and Andrew McCallum. 2010. arXiv. <https://doi.org/10.48550/ARXIV.1011.4088>

↩ ↪ ↪ ↩

Research Materials Statement

Article: this web article was written with React and built using Gatsby. It's source code is available [on GitHub](#).

Models: Jupyter notebooks for training and validating the three models mentioned in this article are available [on OSF](#).

Data: all three models were trained and evaluated on the [CoNLL-2003 dataset](#), available for download [here](#).

Authorship

Vu Luong (Minerva University): conceptualization, methodology, investigation, software, visualization, writing – original draft.

Justin Selig (Cornell University): conceptualization, writing – review & editing.

License

This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

Conflicts of Interest

The authors declare that there are no competing interests.