Entropy Models

Maximum Entropy (MaxEnt) Models in NLP

Complete study notes: motivation, conditional probability modeling, maximum entropy principle, training and comparison with HMMs

Contents

0. Background: entropy in a nutshell
1. Motivation for Maximum Entropy models
2. Conditional probability modeling
3. The Maximum Entropy principle
4. Training MaxEnt models
5. Comparison with HMMs
6. Applications and quick revision

0. Background: Entropy in a Nutshell

Definition

Entropy measures the uncertainty (average surprise) of a probability distribution. For a distribution p over outcomes x:

> **$H(p) = -\Sigma\ p(x) \log_2 p(x)$** (measured in bits; the sum runs over all outcomes x)

Important points

- Entropy is **0** when one outcome is certain, and it is **highest** when all outcomes are equally likely (the uniform distribution).
- For n equally likely outcomes, $H = \log_2 n$ bits. Four equally likely tags give 2 bits.
- High entropy means "I know little"; low entropy means "I am confident". A model with maximum entropy therefore **avoids claiming knowledge it does not have**.

Example: computing entropy

Distribution	Calculation	Entropy
Fair coin (0.5, 0.5)	$-(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 0.5 + 0.5$	**1.000 bit**
Biased coin (0.9, 0.1)	$-(0.9 \times (-0.152) + 0.1 \times (-3.322)) = 0.137 + 0.332$	**0.469 bit**
Certain outcome (1, 0)	$-(1 \times \log_2 1) = 0$	**0 bit**
Four equally likely tags	$\log_2 4$	**2.000 bits**

Application: entropy is used to choose models (this document), to measure how unpredictable a language is, to build decision trees (information gain) and to evaluate language models through perplexity (2^H).

1. Motivation for Maximum Entropy Models

Definition

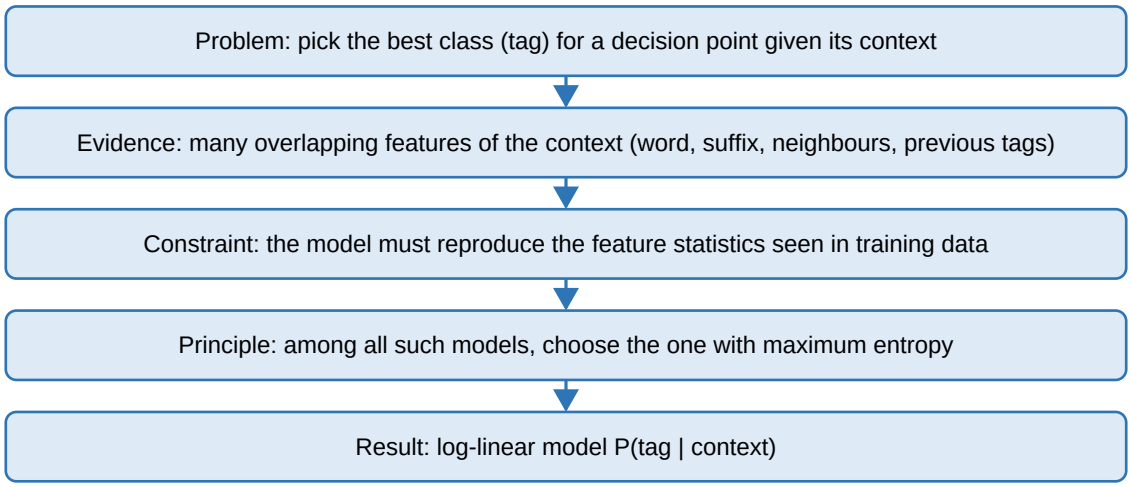
The **motivation** for MaxEnt models is the need for a classifier that can (a) combine many different and overlapping pieces of evidence about a decision, and (b) do so while making **no assumptions beyond the evidence** found in the training data. MaxEnt (Berger, Della Pietra and Della Pietra, 1996; applied to tagging by Ratnaparkhi, 1996) was introduced to fill this gap left by simpler statistical models.

Important points: why earlier models were not enough

- **Narrow evidence in HMMs.** An HMM tagger uses only two facts: the previous tag(s) and the current word identity. Useful clues such as suffix, prefix, capitalization, hyphens, the *next* word or the word two positions back cannot be added easily.
- **Independence assumptions.** Naive Bayes assumes the features are independent. Real features overlap (e.g., "ending in -ing" and "previous tag is a form of be"), so counting them as independent double-counts evidence.

- **Generative burden.** An HMM must model P(word | tag), a huge space, even though we only want the tag. A model that directly estimates P(tag | context) is simpler and focused on the real task.
- **Unknown words.** A word never seen in training gets P(word | tag) = 0 in an HMM and needs special handling. In MaxEnt, suffix and capitalization features give good evidence automatically.
- **Sparse data and zero counts.** Counting full contexts gives many zero counts. Features generalize by sharing weights across many contexts.
- **Principled choice among many possible models.** Many distributions fit the same data. We need a rule for choosing one without bias. MaxEnt supplies it: choose the most uniform distribution that agrees with all the observed facts.

Architecture: where MaxEnt fits



Example: why overlapping features matter

Tag the unknown word **“Vijayawada”** in “He lives in Vijayawada”.

Clue available	HMM tagger	MaxEnt tagger
Word never seen in training	P(word | tag) = 0 for every tag; needs a special unknown-word model	No problem; the word identity is just one feature among many
Starts with a capital letter inside the sentence	Not representable	Feature: capitalized, strong evidence for NNP
Previous tag is IN (preposition)	Used (as a transition)	Feature: previous tag = IN
Next word is a full stop	Not representable	Feature: next token = “.”
All clues together	Cannot combine freely	Weights of all active features are added together

Application: the motivation behind Ratnaparkhi’s MaxEnt tagger (about 96.6% accuracy, with good unknown-word accuracy), the Stanford log-linear tagger, and MaxEnt classifiers for text, named entities and parsing decisions.

2. Conditional Probability Modeling

Definition

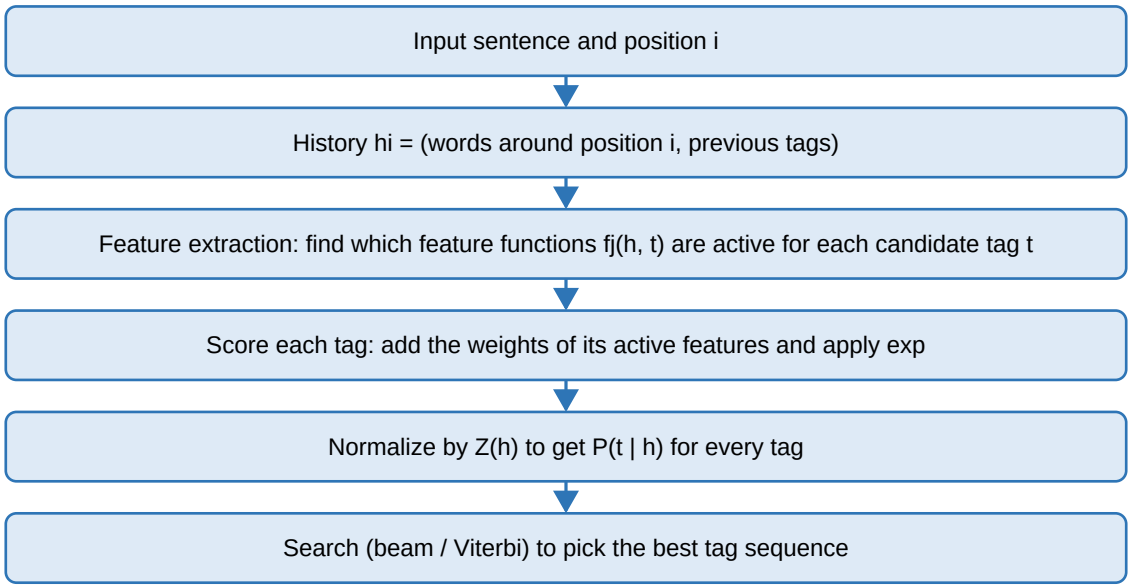
Conditional probability modeling estimates the probability of the output directly given the input, P(y | x), instead of modeling the joint P(x, y). For tagging, the output is the tag t and the input is the history h (the word and its surrounding context). A MaxEnt model represents this as a **log-linear (exponential) model**:

> **$P(t \mid h) = \exp(\sum_j \lambda_j f_j(h, t)) / Z(h)$**
> **$Z(h) = \sum_{t2} \exp(\sum_j \lambda_j f_j(h, t2))$** (normalizer: t2 runs over all candidate tags, so probabilities sum to 1)
> $f_j(h, t)$ = feature function (usually 0 or 1); λ_j = weight of feature j

Important points

- **Discriminative vs generative:** generative models (HMM, naive Bayes) model P(x, y) and use Bayes' rule; discriminative models (MaxEnt, logistic regression) model P(y | x) directly.
- **Feature functions** are binary tests that pair a property of the context with a particular tag, e.g., "previous tag is TO *and* current tag is VB".
- **Weights (λ)** measure how strongly a feature favours its tag. Positive weight pushes the tag up, negative weight pushes it down, zero means no effect.
- **Why exp?** The exponential makes every score positive, and Z(h) normalizes the scores into probabilities. This is the same as the **softmax** function.
- **Sequence tagging:** the probability of a whole tag sequence is the product of the local decisions, $P(t_{1..n} \mid w_{1..n})$ = product of $P(t_i \mid h_i)$, and the best sequence is found with beam search or Viterbi (MEMM).
- **Rich history:** h_i can contain w_i, w_{i-1}, w_{i+1}, w_{i-2}, t_{i-1}, t_{i-2} and character-level properties of w_i.

Architecture



Feature type	Example feature $f_j(h, t) = 1$ when ...
Current word	word = “race” and t = NN
Previous tag	previous tag = TO and t = VB
Two previous tags	tags (t_{i-2}, t_{i-1}) = (DT, JJ) and t = NN
Suffix	word ends with “-ing” and t = VBG
Prefix	word starts with “un-” and t = JJ
Capitalization	word is capitalized (not sentence-initial) and t = NNP
Next / previous word	next word = “the” and t = VB

Example 1: tagging “race” (step by step)

Sentence: “He wants to **race**.” History: previous tag = TO, word = race. Candidate tags: NN and VB. Trained weights:

Feature	Fires for	Weight λ
f_1: previous tag = TO and t = VB	VB	2.0
f_2: previous tag = TO and t = NN	NN	-1.5
f_3: word = race and t = NN	NN	1.2
f_4: word = race and t = VB	VB	0.8

Step 1: Active features: VB gets f_1 and f_4; NN gets f_2 and f_3.

Step 2: Sum the weights: score(VB) = 2.0 + 0.8 = 2.8; score(NN) = -1.5 + 1.2 = -0.3.

Step 3: Exponentiate: exp(2.8) = 16.44; exp(-0.3) = 0.74.

Step 4: Normalize: Z = 16.44 + 0.74 = 17.18.

Step 5: Probabilities: P(VB | h) = 16.44 / 17.18 = **0.957**; P(NN | h) = 0.74 / 17.18 = 0.043.

Step 6: Decision: choose VB, so to/TO race/**VB**.

Example 2: an unknown word with three candidate tags

Sentence: “He lives in **Vijayawada**.” The word is unknown, previous tag = IN, and the word is capitalized.

Tag	Active features and weights	Score	exp(score)	P(t | h)
NNP	capitalized+NNP (2.2), prev IN+NNP (0.6)	2.8	16.44	**0.939**
NN	capitalized+NN (-0.5), prev IN+NN (0.4)	-0.1	0.90	0.052
JJ	capitalized+JJ (-1.0), prev IN+JJ (-0.8)	-1.8	0.17	0.009

Z = 16.44 + 0.90 + 0.17 = 17.51. The model picks **NNP** with 94% confidence, even though it has never seen the word, because the capitalization and context features carry the evidence.

Application: any classification where the output depends on many overlapping clues, e.g., PoS tagging, named entity recognition, sentence-boundary detection, text categorization and sentiment classification.

3. The Maximum Entropy Principle

Definition

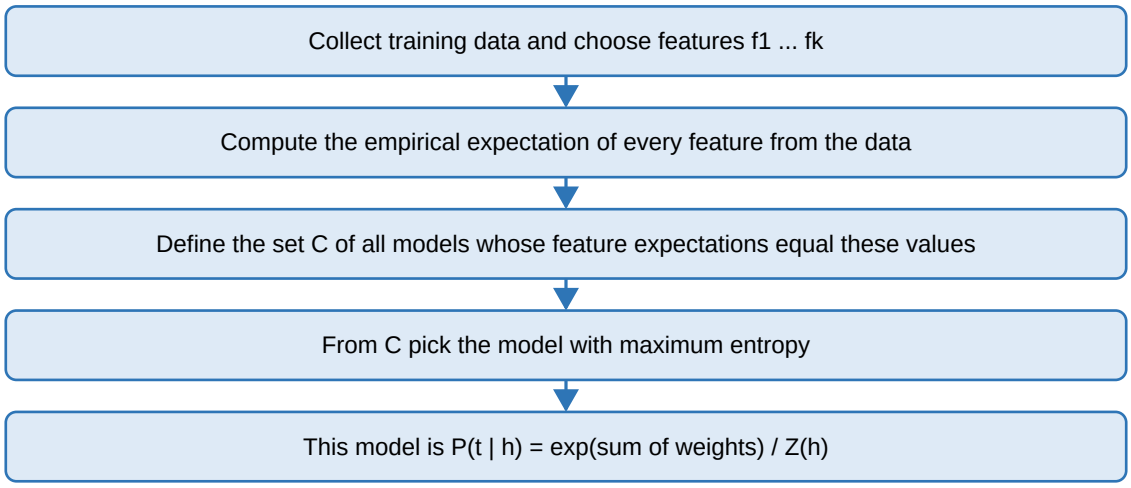
The **Maximum Entropy principle** (E. T. Jaynes, 1957) states: *when estimating a probability distribution, model everything that is known, and assume nothing about what is unknown.* Among all distributions that satisfy the known constraints, choose the one with the **largest entropy**, i.e., the most uniform one.

> **$p^* = \text{argmax } H(p)$ subject to $E_p[f_j] = E_{empirical}[f_j]$ for every feature j**
> where the conditional entropy is $H(p) = - \Sigma\ \tilde{p}(h)\ p(t \mid h) \log p(t \mid h)$
> $E_{empirical}[f_j]$ = average value of feature j in the training data; $E_p[f_j]$ = average value under the model

Important points

- **Constraints come from data.** Each feature gives one constraint: the model must predict the same feature frequency that was observed in training.
- **No constraint means uniform.** If nothing is known, the maximum entropy distribution is uniform: every tag is equally likely.
- **Occam's razor in probability form.** Any non-uniform choice that is not forced by data would add an unjustified assumption.
- **Key theorem.** The distribution that maximizes entropy under linear feature constraints always has the log-linear form of Section 2. It is also the distribution that maximizes the likelihood of the training data within that family (duality). This is why training can use likelihood.
- **The solution is unique** and the optimization problem is convex, so there are no bad local maxima.

Architecture: the principle as a procedure



Example 1: five French translations of "in" (Berger et al.)

A translator chooses among five French words for "in": *dans, en, à, au cours de, pendant*. We know only the following facts.

Step 1: Constraint 1 (always true): p(dans) + p(en) + p(à) + p(au cours de) + p(pendant) = 1.

Step 2: **No other knowledge:** maximum entropy gives the uniform answer, 0.2 for each word (entropy = $\log_2 5 = 2.32$ bits).

Step 3: **New fact from data (Constraint 2):** the translator chose *dans* or *en* in 30% of cases: p(dans) + p(en) = 3/10.

Step 4: **Apply max entropy:** without further information, split the 0.3 equally between the two words and the remaining 0.7 equally between the other three.

Word	dans	en	à	au cours de	pendant
Uniform (Constraint 1 only)	0.200	0.200	0.200	0.200	0.200
Max entropy (Constraints 1 and 2)	0.150	0.150	0.233	0.233	0.233

Step 5: **Check:** 0.15 + 0.15 = 0.30 and 3 x 0.233 = 0.70. The new entropy is 2.29 bits, slightly lower than 2.32 because we added a fact. Adding more constraints (more features) lowers the entropy further but keeps the model honest.

Example 2: tagging the word “can”

Possible tags: MD, NN, VB. The corpus shows that MD occurs 70% of the time.

Step 1: **Constraints:** P(MD) = 0.7 and P(MD) + P(NN) + P(VB) = 1.

Step 2: **Max entropy:** the leftover 0.3 is split equally: P(NN) = P(VB) = 0.15.

Step 3: **Check the entropy:** $H = -(0.7 \log_2 0.7 + 2 \times 0.15 \log_2 0.15) = 0.360 + 0.821 =$ **1.181 bits**.

Step 4: **Compare with a biased alternative** P(NN) = 0.25, P(VB) = 0.05 (also meets the constraint): H = 0.360 + 0.500 + 0.216 = 1.076 bits. It is lower, because it assumes NN is five times as likely as VB without any evidence.

Application: the principle is used to build language models with long-distance triggers, translation models, tagging and parsing models, and in physics and ecology for inference from partial information.

4. Training MaxEnt Models

Definition

Training a MaxEnt model means finding the feature weights $\lambda_1 \dots \lambda_k$ so that the model’s expected feature counts match the empirical counts (equivalently, so that the conditional log-likelihood of the training data is maximal).

$L(\lambda) = \Sigma_i \log P(t_i \mid h_i)$ (conditional log-likelihood of the training set)
Gradient: $dL / d\lambda_j = E_{empirical}[f_j] - E_{model}[f_j]$
At the optimum the gradient is zero, i.e., $E_{model}[f_j] = E_{empirical}[f_j]$ for all j (the maximum-entropy constraints).

Important points

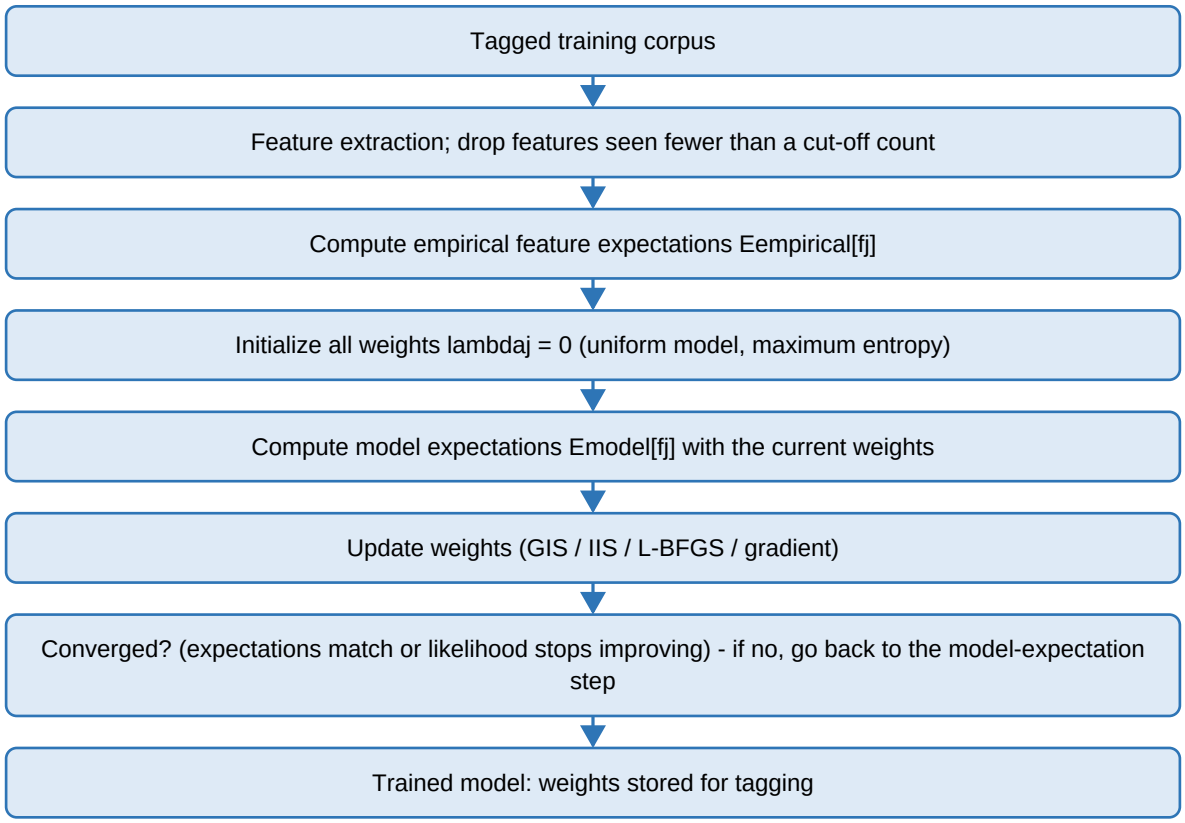
- **Convex objective.** L is concave in λ, so any optimizer reaches the single global optimum.
- **There is no closed-form solution** in general, so weights are found iteratively.

- **The update rule is intuitive:** if the model predicts a feature less often than the data shows, increase its weight; if more often, decrease it.
- **Cost per iteration:** computing E_{model} needs Z(h) for every training event, so the cost grows with data size, the number of tags and the number of features.
- **Regularization** avoids overfitting: add a penalty such as L2 (Gaussian prior), L1 (sparse weights), or drop rare features with a count cut-off.

Training algorithms

Algorithm	Idea	Remarks
GIS (Generalized Iterative Scaling)	Updates λ_j by (1/C) log(E_{emp} / E_{model}), where C is the constant number of active features per event (a correction feature is added to make it constant).	Simple, guaranteed to converge, but slow.
IIS (Improved Iterative Scaling)	Solves a one-dimensional equation for each weight; no constant C needed.	Faster than GIS.
Gradient ascent / conjugate gradient	Moves λ along the gradient E_{emp} - E_{model}.	Much faster than iterative scaling.
L-BFGS (quasi-Newton)	Uses approximate curvature from recent gradients.	Standard choice today for medium-sized problems.
Stochastic gradient descent	Updates the weights from small batches of data.	Used for very large data sets and neural variants.

Architecture: training pipeline



Example: learning one weight by gradient ascent (step by step)

Feature f_1: "previous tag = TO and t = VB". Among 200 training events with previous tag TO, the tag was VB in 166, so the empirical value is $E_{empirical} = 166/200 = 0.83$. The only other tag is NN. Learning rate = 2.

Step 1: Initialize: $\lambda = 0$, so the model is uniform: P(VB) = 0.5 and $E_{model} = 0.5$.

Step 2: Gradient: 0.83 - 0.50 = 0.33 (the model under-predicts VB), so λ increases by 2 x 0.33 = 0.66.

Step 3: Recompute $P(VB) = e^{0.66} / (1 + e^{0.66}) = 0.659$ and repeat.

Iteration	Weight λ	P(VB) (model)	Gradient = 0.83 - P(VB)	New weight
0	0.000	0.500	0.330	0.660
1	0.660	0.659	0.171	1.002
2	1.002	0.731	0.099	1.200
3	1.200	0.769	0.061	1.322
4	1.322	0.790	0.040	1.403

Step 4: Convergence: the gradient shrinks each round and P(VB) approaches 0.83. For this one-feature model the exact answer is $\lambda = \ln(0.83 / 0.17) =$ **1.586**, which gives P(VB) = 0.83. At that point $E_{model} = E_{empirical}$: the constraint is satisfied.

Step 5: With many features the same procedure updates all weights at once; GIS, IIS and L-BFGS only differ in how large and in what direction each step is.

Application: training is run once offline, e.g., with the OpenNLP and Stanford toolkits (MaxEnt tagger, NER) and with logistic regression in scikit-learn; tagging afterwards is fast.

5. Comparison with HMMs

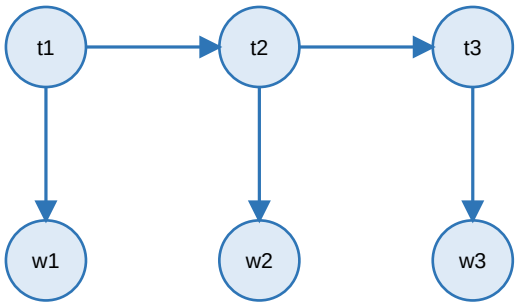
Definition

Both models are statistical sequence taggers, but they model different quantities. An **HMM** is **generative**: it models how tags generate words, P(T) x P(W | T). A **MaxEnt** tagger (used as a MEMM when applied to sequences) is **discriminative**: it models P(T | W) directly.

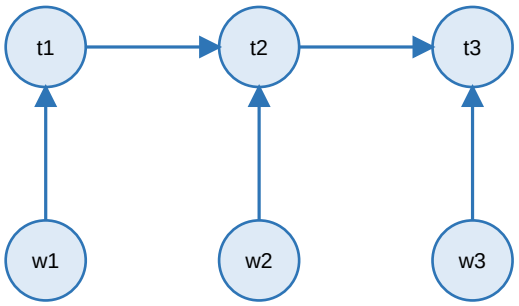
> **HMM:** T* = argmax P(W | T) x P(T) = argmax product of $P(w_i \mid t_i) \times P(t_i \mid t_{i-1})$
> **MaxEnt (MEMM):** T* = argmax product of $P(t_i \mid t_{i-1}, w_{i-1}, w_i, w_{i+1}$, suffix, ...)

Architecture: direction of dependence

HMM (generative): tag -> word



MaxEnt / MEMM (discriminative): context -> tag



Comparison table

Aspect	HMM	Maximum Entropy
Model type	Generative (joint P(W, T))	Discriminative (conditional P(T | context))
What is estimated	Transition P(t | t_{prev}) and emission P(w | t)	Weights of feature functions in P(t | h)
Features	Previous tag(s) and current word only	Any number of overlapping features (suffix, capitalization, next word, ...)
Independence assumptions	Strong: word depends only on its tag; tag only on previous tag	None about features; features may overlap
Unknown words	Needs a special model; emission = 0	Handled naturally by suffix and shape features
Training	Counting (maximum likelihood); fast, closed form	Iterative optimization (GIS, IIS, L-BFGS); slower
Unlabelled data	Can be trained with EM (Baum-Welch)	Needs labelled data
Decoding	Viterbi (exact)	Left-to-right with beam search or Viterbi
Outputs	Joint probability of words and tags	Calibrated P(tag | context)
Known weakness	Cannot use rich features	MEMM label bias: local normalization favours tags with few outgoing options
Typical PoS accuracy	About 95 to 97%	About 96.5 to 97%, better on unknown words

Example: the same word in both models

Tag **“Vijayawada”** in “He lives in Vijayawada”, an unknown word.

Step 1: HMM: the emission P(Vijayawada | t) is 0 for every tag because the word was never seen. Without a special unknown-word model (e.g., suffix-based guessing), the tagger cannot decide.

Step 2: MaxEnt: the active features (capitalized, previous tag IN, next token “.”) vote together, and the model returns NNP with probability 0.94 (Section 2).

Step 3: Conclusion: MaxEnt gains accuracy through additional evidence; the HMM keeps the advantages of simplicity, speed of training and exact decoding.

Related models

- **MEMM (Maximum Entropy Markov Model):** a MaxEnt classifier at every position, linked through the previous tag; allows rich features but has the label-bias problem.
- **CRF (Conditional Random Field):** a globally normalized extension of MaxEnt to sequences; it removes label bias and is one of the best classical sequence taggers.
- **Neural taggers** (BiLSTM, Transformer) replace hand-made features with learned ones, but their final softmax layer is exactly a MaxEnt model.

Application: choose an HMM when training data is small or unlabelled and speed matters; choose MaxEnt, MEMM or CRF when labelled data is available and rich, overlapping features (morphology, context) matter, as in NER and tagging of morphologically rich languages.

6. Applications and Quick Revision

Application	How MaxEnt is used
PoS tagging	Ratnaparkhi tagger, Stanford log-linear tagger; features from word, suffix, prefix and neighbouring tags.
Named entity recognition	Classify each token as person, place, organization or other using capitalization, gazetteers and context.
Chunking and parsing	Choose the next parsing action or chunk label (Ratnaparkhi's MaxEnt parser).
Text classification and sentiment	Multinomial logistic regression over word features.
Machine translation	Choose translations of a word or phrase given its context (Berger et al.'s original application).
Other uses	Sentence-boundary detection and language modelling (combining n-grams and long-distance triggers in one exponential model).

Quick revision points

- **Entropy and motivation:** entropy measures uncertainty and is highest for the uniform distribution; HMMs and naive Bayes cannot combine rich overlapping evidence, MaxEnt can.
- **Conditional model:** P(t | h) = exp(sum of weights of active features) / Z(h); a log-linear (softmax) model.
- **Principle:** match the feature statistics of the data, and otherwise stay as uniform as possible (maximum entropy).
- **Training:** maximize conditional log-likelihood; gradient = empirical expectation minus model expectation; GIS, IIS, L-BFGS; use regularization.
- **Versus HMM:** discriminative vs generative; rich features and better unknown-word handling vs simpler and faster training; MEMM label bias is solved by CRF.

> **Summary:** A Maximum Entropy model is the most unbiased conditional model that agrees with the evidence in the training data. It combines many overlapping features in a log-linear form, is trained by matching expected and observed feature counts, and gives more flexible tagging than an HMM at the price of heavier training.