# Graphical Models for Part-of-Speech Tagging

Alexander Simcock (2146154)

## 1 Introduction

Natural language processing is approaching ubiquity in the modern world, encompassing technologies from spam-email filters to large language models and much more. Unordered 'bag-of-words' approaches yield effective results despite their simplicity, but can be easily misled and fail to capture finely-grained detail. By utilising the sequential character of spoken word and text it is possible to incorporate new, advantageous features crucial for both natural language understanding and generation.

Broadly speaking, any approach to natural language processing can be categorised as utilising formalised language theory or probabilistic methods (or both). Models of the former kind parse and generate language according to pre-defined rule-sets. Creating such rule-sets requires extensive domain expertise in linguistics and produces inherently rigid models which may struggle to adapt to the fluid development of natural language. In contrast, probabilistic models learn their own rule-sets by employing extensive reference data and are consequently as flexible as their inputs are broad. The improved accessibility of quality reference corpora[1] in the internet-age has accelerated the development of such models, establishing them as the modern standard for natural language processing tasks.

This work analyses the mathematical foundations of two key probabilistic sequence-labelling models, the **hidden Markov model** and **linear-chain conditional random field**. The similarities, advantages and key characteristics of both approaches will be highlighted by an illustrative case-study from the field of natural language processing: part-of-speech tagging.

### 1.1 Preliminary: Part-of-Speech (POS) Tagging

The term 'part-of-speech' (POS) refers to the syntactic category of a word, for example noun, verb or adjective.[2] The algorithmic categorisation of words is made difficult by the ambiguities of regular language. The challenge of POS tagging is thus:

> Given a sequence of words, tag each word occurrence with the correct syntactic category. Consider the following example (Eisenstein 2019), should the grammatically-ambiguous words 'can' and 'fish' in the sequence 'they can fish' be tagged *noun* or *verb*? What about in the sequence 'can of fish'?[3]

#### 1.1.1 Motivation

POS tagging is a key preliminary step in many natural language processing pipelines and assists word-sense disambiguation and word-sequence parsing algorithms alike (Allen 1995).

[1] The Penn Treebank is a notable industry-standard example of an annotated corpus. Generally speaking, a corpus is simply a very large dataset of text documents.

[2] These tags would be considered 'coarse-grained' in a typical annotated corpus, but befit the examples in this report. The tagset of the BNC (Leech 1995) can be seen for an example of industry-grade POS tags.

[3] The correct tags for the given examples would be (*verb*, *verb*) and (*noun*, *noun*) respectively.

In providing both lexical and syntactic clarification, POS tagging can be of great benefit to many modern applications that are required to derive semantic meaning from natural language (such as search engines, conversational agents and machine translation). For these reasons (in addition to ease of interpretation) POS tagging will serve as the core pretext for sequence labelling within this report.

### 1.1.2 POS Tagging as Classification

In order to compose a rigorous approach to tagging any given sequence of words, we introduce sequential random variables $\mathbf{x}$ and $\mathbf{y}$ which enable a reframing of POS tagging which is fit for mathematical manipulation.

> Let $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_T)$ be random variables such that $\mathbf{x}$ varies over all (word) sequences to be labelled and $\mathbf{y}$ over all label sequences. All elements of $\mathbf{y}$ are selected from a finite alphabet $\mathcal{Y}$ (the set of all POS tags). The POS tagging problem can then be stated: '**given x, predict the correctly matched tag sequence y**'.[4]

Approached in this manner, POS tagging is an example of **statistical classification**, in which we wish to predict the unknown discrete class label $\mathbf{y}$ underlying the observed set of features $(x_1, x_2, \ldots, x_T)$. In this project we will derive tag-sequence predictions (denoted $\hat{\mathbf{y}}$) by applying **maximum likelihood estimation (MLE)**, whereby we assume that the optimal prediction will maximise an appropriate likelihood function. A sensible choice of function to begin with is the posterior probability distribution $p(\mathbf{y} \mid \mathbf{x})$. We are now tasked with devising models that will facilitate the computation of this distribution and hence provide the desired POS classifier.

# 2 The Naïve Bayes Classifier

Any probabilistic classifier must be trained on a dataset in order to achieve reasonable results. Within this project a dataset of pre-labelled examples $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$ will be referred to, where $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_T^{(i)})$ and $\mathbf{y}^{(i)} = (y_1^{(i)}, y_2^{(i)}, \ldots, y_T^{(i)})$. For notational convenience, the sequences in $\mathcal{D}$ are universally presented as length $T$, in practice $T$ will depend on $i$ and the methods put forth in this project can be easily adapted to reflect this.

The classifier we look to derive must be able to handle sequences that do not appear in our reference data, since gathering a dataset that contains all sequences and their potential labels at the correct frequencies is an impossible task (consider that we can construct an infinite number of sequences by joining pre-existing ones). As a result, we should look to construct our likelihood function (and thus $\hat{\mathbf{y}}$) from localised parts of $\mathbf{x}$ which we can safely assume will appear in the dataset. Taking the most granular approach, by first considering the individual elements of $\mathbf{x}$ seems to be a suitable start point.

Considering each value $x_t$ in turn will require that we model the interactions between them, the most straightforward approach will assume there are no such interactions, that is, $x$ and $x'$ are independent given a fixed value of $\mathbf{y}$. We can impose this condition with the following assumptions:

**Assumption 1 (Independence of Observations)** *The probability of each word $x_t$ depends exclusively on the corresponding tag $y_t$ and is independent of all other elements in either sequence: $p(\mathbf{x} \mid \mathbf{y}) = \prod_{t=1}^{T} p(x_t \mid y_t)$.*

[4] The framework presented here, and as such all mathematics in this project, can be applied to any problem that involves the labelling of sequential data by reconsidering the spaces which $\mathbf{x}$, $\mathbf{y}$ and $\mathcal{Y}$ vary over. Possible examples include gene prediction and speech recognition.

**Assumption 2 (Independence of Tags)** *The probability of each tag $y_t$ is independent of all other tags in* $\mathbf{y}$*:* $p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t)$.

By first applying Bayes' theorem and then our assumptions, we are able to quickly derive $\hat{\mathbf{y}}$. We note as we do so that $\frac{1}{p(\mathbf{x})}$ is constant over $\mathbf{y}$ and so has no bearing on the output of the arg max function, allowing us to simplify the final expression.

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \mathbf{y})p(\mathbf{y})}{p(\mathbf{x})} = \frac{1}{p(\mathbf{x})} \prod_{t=1}^{T} p(x_t \mid y_t)p(y_t) = \frac{1}{p(\mathbf{x})} \prod_{t=1}^{T} p(x_t, y_t), \tag{1}$$

$$\therefore \hat{\mathbf{y}} = \arg\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x}) = \arg\max_{\mathbf{y}} \prod_{t=1}^{T} p(x_t, y_t). \tag{2}$$

The classifier we have obtained is known as the **naïve Bayes classifier** (here with a single feature, the word). Naïve Bayes classifiers are based on the joint probability model

$$p(\mathbf{y}, \mathbf{x}) = p(\mathbf{y}) \prod_{t=1}^{T} p(x_t \mid \mathbf{y}), \tag{3}$$

which emerges from our assumptions. The connection between our classifier and the naïve Bayes model becomes more apparent if one notes that we may have equivalently derived eq. (1) by writing it in terms of the joint probability and applying eq. (3) before our assumptions. To complete the classifier, we estimate the necessary probabilities $p(x_t, y_t)$ by taking the relative frequency counts from $\mathcal{D}$ i.e. $p(x_t = k, y_t = i) \widehat{=} [\sum_{i=1}^{N} \sum_{t=1}^{T} \mathbf{1}_{\{x_t^{(i)}=k\}} \mathbf{1}_{\{y_t^{(i)}=i\}}]/TN$ where '$\widehat{=}$' indicates estimation rather than equality and $\mathbf{1}$ denotes an indicator function that returns 1 when its condition is true and 0 otherwise.

On inspection of the classifier it becomes apparent that it merely assigns the tag to each word which is most frequently associated with that word. Such a rudimentary approach is still likely to have a high accuracy rate since many words in the English language have only one appropriate POS tag, but its failure to distinguish between different contexts is a critical flaw and will lead it to fail in many cases of grammatical-ambiguity.

# 3 Hidden Markov Models (HMMs)

As we look to build upon the naïve Bayes approach we might first consider reworking the restrictive transition assumption. Intuitively, an adjective or noun tag should be more likely to succeed the determiner 'the' than a verb. We would expect that as more finely-grained tags are included in the model, many such significant relationships between the tags are likely to emerge. For this reason we substitute the transition assumption for a Markov assumption.

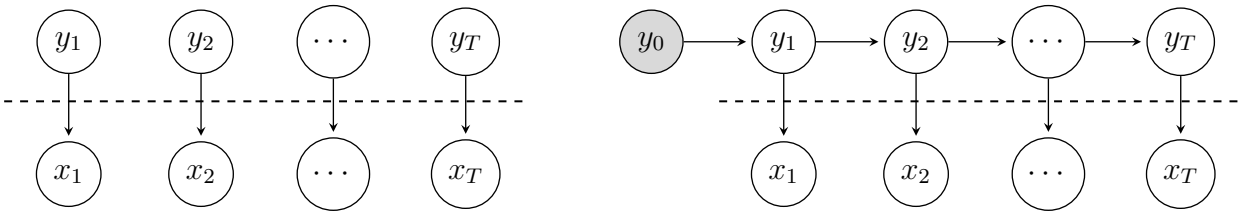
**Assumption 3 (Markov Assumption)** *The probability of each tag $y_t$ is dependent (only) on its predecessor $y_{t-1}$. Thus $p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t \mid y_{t-1})$. Under this assumption* $\mathbf{y}$ *is considered a stochastic process that exhibits the Markov property.*

We also introduce a 'start' node $y_0$, which will allow the calculations defined by the Markov assumption to extend to $y_1$. Adopted alongside the emission assumption, the Markov assumption specifies a **hidden Markov model (HMM)**. The framework of the HMM was first described by Leonard E. Baum (Baum and Petrie 1966) and saw widespread uptake in the field of speech recognition during the 1970's. Since then, HMMs have also seen popular use in fields such as bioinformatics and, indeed, natural language processing.

The HMM can be thought of as a sequential adaptation of the naïve Bayes model where the current tag-state $y_t$ probabilistically determines not only the observation $y_t$ but also the succeeding tag $y_{t+1}$. Much like naïve Bayes, the HMM can also be summarised concisely by the factorisation of the joint probability that arises from its assumptions:

$$p(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^{T} p(y_t \mid y_{t-1}) p(x_t \mid y_t). \tag{4}$$

The connection between the two models is even more apparent when both are viewed graphically, as in fig. 1.




(a) A naïve Bayes model.

(b) A hidden Markov model.

Figure 1: Graphical structures of the naïve Bayes and hidden Markov model.

Both graphs share the node set $\mathcal{Y} \cup \mathcal{X}$, where $\mathcal{X}$ denotes a finite set of all the words that could appear in $\mathbf{x}$. The directed edges of these graphs indicate either transition from one tag to another or the emission of a word from a tag and are weighted according to the probability of the emission/transition they represent. The graphical interpretation of the HMM therefore assumes that $\mathbf{y}$ is generated by a walk on a 'hidden' Markov chain over $\mathcal{Y}$ that starts at $y_0$ and ends after $T$ steps. Likewise, we can understand that the HMM assumes that each word $x_t$ is probabilistically generated by $y_t$, in the same manner as the naïve Bayes.

As the graphical representation elucidates, the HMM can be summarised entirely by its transition and emission probabilities, which we store in the matrices $A$, $B$ and $\pi$ (where the vector $\pi$ is the unique case of transition from $y_0$).

$$\begin{aligned} A &= [a_{i,j}] \quad i, j \in \mathcal{Y}, & a_{i,j} &\coloneqq p(y_t = j \mid y_{t-1} = i), \\ B &= [b_i(k)] \quad i \in \mathcal{Y}, & b_i(k) &\coloneqq p(x_t = k \mid y_t = i), \\ \pi &= [\pi_i] \quad i \in \mathcal{Y}, & \pi_i &\coloneqq p(y_1 = i). \end{aligned}$$

Taken together, these matrices form a complete set of model parameters which we call $\theta = (A, B, \pi)$. Since the probability distributions we derive from our HMM will be dependent on these parameters we also introduce the notation $p_\theta(\mathbf{x}) \coloneqq p(\mathbf{x} \mid \theta)$. By adjusting the components of $\theta$ we are able to tweak the performance of the HMM or even re-purpose it for entirely different tasks.

Having established the HMM, three questions central to enabling its practical application, including for POS tagging, emerge (Rabiner 1989):

1. **(Computing Marginals)** Given $\mathbf{x}$ and a HMM $\theta = (A, B, \pi)$, how do we efficiently compute $p_\theta(\mathbf{x})$?

2. **(Sequence Labelling)** Given $\mathbf{x}$ and a HMM $\theta = (A, B, \pi)$, how do we choose a corresponding state sequence $\mathbf{y}$ that best explains $\mathbf{x}$?

3. **(Parameter Estimation)** How might we adjust the model parameters $\theta = (A, B, \pi)$, in order to maximise $p_\theta(\mathbf{x})$?

These questions respectively motivate the introduction of the forward-backward, Viterbi and Baum-Welch algorithms.

## 3.1 The Forward-Backward Algorithm

The derivation of this algorithm is motivated by the first question, that is, how can we efficiently compute the marginal distribution $p_\theta(\mathbf{x})$? Answering this question will be pivotal in answering the succeeding questions, as we will need to call this calculation as a sub-procedure many times, particularly in parameter estimation. The marginal distribution can also be of interest itself, for example, a comparison between $p_\theta(\mathbf{x})$ and $p_{\theta'}(\mathbf{x})$ (where $\theta$ and $\theta'$ have been trained separately) could be used to identify the most likely author of a passage $\mathbf{x}$.

We first attempt to directly derive $p_\theta(\mathbf{x})$ by noting that it is equivalent to the sum of $p_\theta(\mathbf{x}, \mathbf{y})$ over all possible state-sequences. To do so, we require the following two identities that result from the emission independence and Markov assumptions:

$$p_\theta(\mathbf{x} \mid \mathbf{y}) = \prod_{t=1}^{T} p_\theta(x_t \mid y_t) = \prod_{t=1}^{T} b_{y_t}(x_t),$$
$$p_\theta(\mathbf{y}) = \prod_{t=1}^{T} p_\theta(y_t) = \pi_{y_1} a_{y_1,y_2} a_{y_2,y_3} \cdots a_{y_{T-2},y_{T-1}} a_{y_{T-1},y_T}.$$

Hence, the marginal probability can be expressed

$$p_\theta(\mathbf{x}) = \sum_{\mathbf{y}} p_\theta(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{y}} p_\theta(\mathbf{x} \mid \mathbf{y}) p_\theta(\mathbf{y}) = \sum_{\mathbf{y}} \pi_{y_1} b_{y_1}(x_1) a_{y_1,y_2} b_{y_2}(x_2) \cdots a_{y_{n-1},y_n} b_{y_n}(x_n).$$

Unfortunately, this expression is intractable, requiring calculations of the order $2T|\mathcal{Y}|^T$, owing to the $2T-1$ multiplications required for each of the $|\mathcal{Y}|^T$ possible values of $\mathbf{y}$. Such complexity scales immensely quickly with $T$ and $|\mathcal{Y}|$, making it infeasible for practical-use. For example, a model trained on the base tagset of the BNC ($|\mathcal{Y}| = 61$) (Leech 1995) would require $7.99 \times 10^{19}$ calculations to calculate $p_\theta(\mathbf{x})$ for a sequence of only five words given no prior trimming of the search space of $\mathbf{y}$.

In order to calculate in a more efficient way we will define the forward-function, the first of a family of variables which allow for the marginal distribution to be found inductively.

**Definition 3.1 (Forward-Function)** *Let $\alpha_t(i)$ denote the joint probability of observing the sequence $\mathbf{x}_{1:t} \coloneqq (x_1, x_2, \ldots, x_t)$ $(1 \leq t \leq T)$ and the tag $y_t$ being $i$, given model parameters $\theta$.*

$$\alpha_t(i) \coloneqq p_\theta(\mathbf{x}_{1:t}, y_t = i)$$

Having defined $\alpha_t(i)$, we note that $p_\theta(\mathbf{x}) = \sum_{i \in \mathcal{Y}} \alpha_T(i)$. Hence, if we can efficiently compute $\alpha_T(i)$ we will have solved the problem at hand. Thankfully, the value of any $\alpha$ can be found efficiently by the inductive 'forward-algorithm'. The base case $t = 1$ is trivial, being the product of the probability of starting at $i$ and the probability of emitting the word $x_1$. Next, note that any $\alpha_t(i)$ can be calculated similarly by taking all states at $t-1$ and multiplying by the transition and emission probabilities that would output $y_t = i$ and $x_t = x_t$ from each of these, summing them to find the result.

$$\begin{aligned} &\text{Base Case:} && \alpha_1(i) = \pi_i b_i(x_1) && \\ &\text{Inductive Step:} && \alpha_{t+1}(j) = \left[\sum_{i \in \mathcal{Y}} \alpha_t(i) a_{i,j}\right] b_j(x_t) && \begin{matrix} i \in \mathcal{Y} \\ 1 \leq t \leq T-1 \end{matrix} \end{aligned}$$

At each step of induction we calculate the value $\alpha_t$ for all $|\mathcal{Y}|$ states, where each is derived from $|\mathcal{Y}|$ previous states. Carrying out this calculation for each of the $T-1$ steps required to find $\alpha_T(i)$ results an overall complexity of order $T|\mathcal{Y}|^2$. In real terms, the earlier 5-word example where $|\mathcal{Y}| = 61, n = 5$ reduces from over $7.99 \times 10^{19}$ calculations to 18605.

Although the forward-function is sufficient in computing the marginal, we also introduce a mirrored approach: the backward-function. Whilst trivial at this point, defining both the forward and backward-functions will prove essential in parameter estimation.

**Definition 3.2 (Backward-Function)** *Let $\beta_t(i)$ denote the joint probability of observing the sequence $\mathbf{x}_{t+1:T}$ $(1 \leq t \leq T)$ and the tag $y_t$ being $j$ given model parameters $\theta$.*

$$\beta_t(j) \coloneqq p_\theta(\mathbf{x}_{t:T} \mid y_t = k)$$

In parallel to the forward-function, $p_\theta(\mathbf{x}) = \sum_{j \in \mathcal{Y}} \beta_1(j)$ and the values of $\beta$ can be calculated inductively by the 'backward-algorithm', starting at $t = T$ and terminating at $t = 1$.

$$\begin{aligned} \text{Base Case:} \quad & \beta_T(i) = 1 && i \in \mathcal{Y} \\ \text{Inductive Step:} \quad & \beta_t(i) = \sum_{j \in \mathcal{Y}} a_{i,j} b_j(x_{t+1}) \beta_{t+1}(i) && 1 \leq t \leq T-1 \end{aligned}$$

Unsurprisingly, the backward-algorithm makes the same improvements in computational efficiency as the forward-algorithm when computing the marginal distribution $p_\theta(\mathbf{x}) = \beta_0(y_0)$.

## 3.2 The Viterbi Algorithm

Despite the importance of each of the three key questions for HMMs, the second is perhaps the most clearly pertinent to POS tagging. Indeed, given a HMM pre-trained on words and their parts-of-speech, answering this question will provide us with our desired classifier. In answering, we must first address how we determine the 'best' matched tag-sequence $\mathbf{y}$. We may be tempted, as in the naïve Bayes, to assume that the best sequence can be constructed by selecting the most likely tag index by index: $\hat{\mathbf{y}}_t \coloneqq \arg\max_{i \in \mathcal{Y}} p_\theta(y_t = i \mid \mathbf{x})$. The forward and backward functions we have derived make this possible and efficient since together, they account for the entire tag sequence. The probability we are looking for, that of reaching any one specified tag at index $t$, is equal to the normalised product of the forward and backward functions. It will prove to be a useful value in subsequent calculations and so we denote it as a new function $\gamma_t(i)$.

$$\gamma_t(i) \coloneqq p_\theta(y_t = i \mid \mathbf{x}) = \frac{\alpha_t(i)\beta_t(i)}{p_\theta(\mathbf{x})} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j \in \mathcal{Y}} \alpha_t(j)\beta_t(j)}$$

Ultimately, taking such an approach to prediction is misguided: making choices that maximise only the local probability of specific nodes can result in an overall prediction with highly incoherent tags and potentially impossible transitions. We must instead maximise the holistic probability $p_\theta(\mathbf{y} \mid \mathbf{x})$. As with the marginal distribution, finding this value directly will be far too computationally intensive.

The **Viterbi algorithm** (Viterbi 1967) is perfectly suited to this task. The algorithm enables the efficient computation of a most-likely sequence of hidden states given a prior distribution. In our use case, we wish to compute the most likely 'Viterbi path' $\mathbf{y}$ given our model parameters $\theta$. We first define the Viterbi variable $\delta_t(i)$ which returns the maximum probability of any one path length $t$ through the Markov chain, given that we observe $\mathbf{x}_{1:t}$ and the path terminates at tag $i$.

$$\delta_t(i) \coloneqq \max_{\mathbf{y}_{1:t-1}} p_\theta(\mathbf{y}_{1:t-1}, y_t = k, \mathbf{x}_{1:t})$$

This value can, again, be found by induction. However, unlike the forward-backward algorithm, the Viterbi algorithm requires that we track the most probable sequence at each step i.e. the

largest value of $\delta_t(i)$. Tracking this variable will allow for the formation of the most likely sequence $\hat{\mathbf{y}}$ by backtracking. For this reason we also define the tracking variable $\phi_t(i)$.

$$\begin{aligned}
&\text{Base Case:} && \delta_1(i) = \pi_i b_i(x_1) && \\
& && \phi_1(i) = 0 && i, j \in \mathcal{Y} \\
&\text{Inductive Step:} && \delta_t(j) = \max_i \left[\delta_{t-1}(i) a_{i,j}\right] b_k(x_t) && 2 \leq t \leq T \\
& && \phi_t(j) = \arg\max_i \left[\delta_{t-1}(i) a_{i,j}\right] &&
\end{aligned}$$

The full algorithm, adapted from (Rabiner 1989; Eisenstein 2019), is given in algorithm 1. As

**Algorithm 1** Viterbi Algorithm

**for** $i \in \mathcal{Y}$ **do**
  $\delta_1(i) \leftarrow \pi_i b_i(x_1)$
**end for**
**for** $t \in \{2, \ldots, T\}$ **do**
  **for** $i \in \mathcal{Y}$ **do**
    $\delta_t(i) \leftarrow \max_{j \in \mathcal{Y}} \left[\delta_{t-1}(j) a_{j,i}\right] b_k(x_t)$
    $\phi_t(i) \leftarrow \arg\max_{j \in \mathcal{Y}} \left[\delta_{t-1}(j) a_{j,i}\right]$
  **end for**
**end for**
$\hat{\mathbf{y}}_T \leftarrow \arg\max_{i \in \mathcal{Y}} \delta_T(i)$
**for** $t \in \{T-1, \ldots, 1\}$ **do**
  $\hat{\mathbf{y}}_t \leftarrow \phi_{t+1}(\hat{\mathbf{y}}_{t+1})$
**end for**
**return** $\hat{Y}$

earlier, applying this dynamic programming results in large gains in cost efficiency. There will be $T \times |\mathcal{Y}|$ Viterbi variables to compute, each of which will require finding a maximum over $|\mathcal{Y}|$ possible previous tags. Overall necessitating $T|\mathcal{Y}|^2)$ calculations to derive all necessary variables and $T$ operations to trace the best matched sequence (Eisenstein 2019).

## 3.3 The Baum-Welch Algorithm

Our current answer to the POS tagging problem with HMMs is not quite complete, since we have pre-supposed a fully trained model but have not yet devised methods for estimating the parameters of such a model. A reasonable starting point is to take the relative frequency counts of transitions and emissions from $\mathcal{D}$ (Eisenstein 2019). Since there are so many possible transitions and emissions it is likely that many will appear very few times, resulting in highly varying, or even zero, probabilities. As such, if we are to take relative frequency counts as our estimates, it is sensible to counter the resulting variance by introducing smoothing parameters:

$$\begin{aligned}
a_{i,j} \coloneqq p(y_t = i \mid y_{t-1} = j) &\leftarrow \frac{\lambda_A + p(y_t = j, y_{t-1} = i)}{\lambda_A |\mathcal{Y}| + p(y_{t-1} = k')} \\
b_i(k) \coloneqq p(x_t = k \mid y_t = i) &\leftarrow \frac{\lambda_B + p(x_t = k, y_t = i)}{\lambda_B |\mathcal{Y}| + p(y_t = k)}
\end{aligned} \qquad \lambda_A, \lambda_B \in \mathbb{R}^+ \tag{5}$$

We use separate hyper-parameters for the transition and emission values as we would expect that the emission probability may need more smoothing due to the word-vocabulary $\mathcal{X}$ being much larger than the set of tags $\mathcal{Y}$ and subsequently more predisposed to over-fitting. This technique is known as additive (or Laplace) smoothing.

Although this approach has merit, there is no guarantee that it will select the 'best' parameters, or even locally optimal ones, instead we will describe an algorithm which is guaranteed to converge: the **Baum-Welch algorithm** (Baum 1972). Baum-Welch is an iterative gradient-descent algorithm and has the desirable property of convergence, although it is not necessarily guaranteed to reach a global minimum. The Baum-Welch algorithm builds on the forward-backward algorithm, combining it with expectation maximisation to inform the iterative changes made to $\theta$.

Let us first define a final helper variable $\xi_t(i,j)$ which returns the probability of transitioning from tag $i$ at $y_t$ to tag $j$ at $y_{t+1}$ given $\mathbf{x}$ and $\theta$. This probability can be re-written in terms of the pre-defined variables by considering the event it describes in components: firstly reaching tag-state $i$ at $y_t$ given $\mathbf{x}_{1:t}$, then transitioning to $j$ from $i$, emitting $x_{t+1}$ and finally reaching tag-state $j$ at $t+1$ given $\mathbf{x}_{t+1:T}$. Hence,

$$\begin{aligned}\xi_t(i,j) &\coloneqq p_\theta(y_t = i, y_{t+1} = j \mid \mathbf{x}) \\ &= \frac{\alpha_t(k) a_{i,j} b_j(x_{t+1}) \beta_{t+1}(j)}{p_\theta(\mathbf{x})} \qquad\qquad i,j \in \mathcal{Y} \\ &= \frac{\alpha_t(k) a_{i,j} b_j(x_{t+1}) \beta_{t+1}(j)}{\sum_{i\in\mathcal{Y}} \sum_{j\in\mathcal{Y}} \alpha_t(i) a_{i,j} b_j(x_{t+1}) \beta_{t+1}(j)} \qquad\qquad 1 \le t \le T-1\end{aligned}$$

We now note that holding $i$ constant and summing $\xi$ over all possible values of $j$ is equivalent to the probability of being in tag-state $i$ at position $t$, which is the same condition that $\gamma$ measures: $\gamma_t(i) = \sum_{j\in\mathcal{Y}} \xi_t(i,j)$. With $\alpha, \beta, \gamma, \xi$ all defined we can now make the iterative step of the Baum-Welch algorithm by noting that

- $\sum_{t=1}^{T-1} \gamma_t(i)$ is the expected number of times that tag $i$ will be visited (as a non-final node) in $\mathbf{y}$.
- $\sum_{t=1}^{T-1} \xi_t(i,j)$ is the expected number of times that tag $i$ will transition to tag $j$ in $\mathbf{y}$.

These interpretations give rise to the following iterations:

$$\begin{aligned}\pi'_k &= \text{expected frequency of label } i \text{ at } (t=1) = \gamma_1(i), \\ a'_{i,j} &= \frac{\text{expected frequency of transitions } i \to j}{\text{expected frequency of transitions from } i} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, \\ b_i(k)' &= \frac{\text{expected frequency of observing word } k \text{ at label } i}{\text{expected frequency of being at label } i} = \frac{\sum_{t=1}^{T} \gamma_t(i) \mathbf{1}_{\{x_t = k\}}}{\sum_{t=1}^{T} \gamma_t(i)}.\end{aligned}$$

In order to utilise the entire dataset $\mathcal{D}$ we calculate the iterations for all reference sequences and take the average result as our adjusted parameter set $\theta'$. Since the Baum-Welch algorithm has been proven to be convergent (Baum 1972), we can re-run the same step again and again until the change in values is as small as a user-decided tolerance.

## 3.4 Limitations of HMMs

Although remarkably stronger than the naïve Bayes, HMMs of the kind presented here are limited in the information that they can incorporate into their estimations. This may become an issue as we look to develop our model further by detecting our own useful features, for example suffixes, the word's position in the sequence or even neighbouring words. Such features would allow our model to make reasonable predictions for words entirely unseen in our data, an impossible task for either the naïve Bayes or HMM.

# 4 Factor Graphs

Having defined the HMM and its algorithms, it is beneficial to formalise the class of models which it and the naïve Bayes belong to. Doing so will allow for parallels to be made between the naïve Bayes, HMM and the succeeding approach, particularly, making the transferability of certain algorithms apparent and succinct. The methods we have discussed are both amenable to graphical representations, however it is their approach to modelling, specifically factorising, the joint probability $p(\mathbf{y}, \mathbf{x}) = p(\mathbf{y})p(\mathbf{x} \mid \mathbf{y})$ (eqs. (3) and (4)) that has mathematically defined their structure and usage. As a result, the **factor graph** is an ideal formalism for these approaches.

Factor graphs are bipartite graphs that provide compact representations of the factorisation of a function and are typically applied to probability distributions. By doing so, they enable efficient computing of certain characteristics of interest, such as marginal distributions. Formally, the bipartite graph $G = (V, F, E)$ is a factor graph when the sets of nodes $V$ and $F$ respectively index the random variables ($\mathbf{x}$ and $\mathbf{y}$) and factors (a set of local functions denoted $\{\Psi_a\}_{a=1}^{A}$). Semantically, any edge in $E$ between a variable node $x_t$ or $y_t \in V$ and factor node $\Psi_a \in F$ indicates that that variable is an argument of $\Psi_a$ in the factorisation of the distribution function.

**Definition 4.1** *Distribution $p(\mathbf{y})$ factorises according to factor graph $G$ if there exists a set of local functions $\{\Psi_a\}_{a=1}^{A}$ such that $p(\mathbf{y})$ can be expressed*

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{a \in F} \Psi_a(\mathbf{y}_{N(a)})$$

*where $N(a)$ is the neighbourhood of node $a$ and $Z$ is a normalising constant defined as*

$$Z = \sum_{\mathbf{y}} \prod_{a \in F} \Psi_a(\mathbf{y}_{N(a)}).$$

For a trivial example, consider the naïve Bayes model of the joint probability eq. (3), which can be written as a factor graph by defining local functions $\Psi(\mathbf{y}) = p(\mathbf{y})$ and $\Psi_t(\mathbf{y}, x_t) = p(x_t \mid \mathbf{y})$. Likewise, the model of the joint probability provided by the HMM eq. (4) can be factorised according to a factor graph where $Z = 1$

$$\Psi_t(j, i, x) \coloneqq p(y_t = j \mid y_{t-1} = i)p(x_t = x \mid y_t = j), \tag{6}$$

$$p(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, x_t). \tag{7}$$

These models are wholly defined by their joint probability distributions, and so by factorising these distribution by a factor graph we have proven that both models themselves are factor graphs.

# 5 Generative and Discriminative Models

In modelling the joint probability distribution $p(\mathbf{y}, \mathbf{x})$, both naïve Bayes and HMMs implicitly construct the marginal distribution $p(\mathbf{x})$, requiring the dictation of a process by which $\mathbf{x}$ is probabilistically generated. This characterising feature is the reason these models are considered 'generative'. Generative models may begin to falter as we look to incorporate new, particularly overlapping, features, such as those alluded to in section 3.4, in our prediction. These models require that we either assume complete feature independence or fully specify all inter-feature dependencies. Independence is unrealistic in such a context, consider as an example that the

value of a feature which returned a word's suffix would be entirely dependent on the matching word, which we already incorporate as the feature $x_t$. Assuming independence regardless of this observation, as naïve Bayes would, will result in probability estimates that are overconfident, since the overlapping information will disproportionately influence overall estimates. On the other hand, choosing to model the dependencies between the features will quickly become prohibitively difficult as the number of features scales. Moreover, even when supposing a sensible formation of dependencies is found, we are increasingly likely to encounter intractability when calculating estimates.

Discriminative models bypass making these problematic assumptions of $p(\mathbf{x})$ by modelling the conditional probability $p(\mathbf{y} \mid \mathbf{x})$ in place of $p(\mathbf{y}, \mathbf{x})$. Theoretically, these two approaches are interchangeable as any joint probability can be converted to a conditional probability (and vice versa) by application of Bayes' theorem. In practice, however, our models are only approximations of the 'true' distributions and so even two models that are considered to form a "generative-discriminative pair" (Ng and Jordan 2001) will produce different estimates.

An example of the relationship between discriminative and generative models are the naïve Bayes and **logistic regression**, which form one such generative-discriminative pair. Both logistic regression and naïve Bayes perform discrete classification on a set of unordered variables $\mathbf{x}$. Unlike naïve Bayes, logistic regression does not require the elements of $\mathbf{x}$ to be independent and hence typically outperforms it in many applications (Caruana and Niculescu-Mizil 2006).[5] The differences in these two modelling approaches are exclusively attributed to one being generative and one discriminative. In fact, any naïve Bayes classifier can be converted to a logistic regression classifier with an identical decision boundary by training the model with the goal of maximising the conditional distribution rather than the joint one (the inverse is true for the logistic regression classifier).

The definition of the logistic regression classifier is given below as it will prove to be illustrative when deriving the next model. In order to concisely write the logistic regression classifier, we define feature functions $f_{\mathbf{y}'}(\mathbf{y}, \mathbf{x}) = \mathbf{1}_{\{\mathbf{y}'=\mathbf{y}\}}$ and $f_{\mathbf{y}',i}(\mathbf{y}, \mathbf{x}) = \mathbf{1}_{\{\mathbf{y}'=\mathbf{y}\}} x_i$ which take the same inputs, allowing for reference to a generic feature function $f_k$ which ranges over both $f_{\mathbf{y}'}$ and $f_{\mathbf{y}',i}$. In the same manner, $\theta_k$ will denote a generic parameter by which the trained model will weight $f_k$ in order to maximise the conditional likelihood. The logistic regression classifier is thus:

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(\mathbf{y}, \mathbf{x})\right\} = \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \Psi_k, \tag{8}$$

where $Z(\mathbf{x})$ is a normalising function and $\Psi_k = \exp\{\theta_k f_k(\mathbf{y}, \mathbf{x})\}$. Hence, logistic regression can also be considered a factor graph.

We may now begin to wonder whether a discriminative pair to the HMM exists, with the hope that it will offer similar benefits in improvement as the logistic model does to naïve Bayes.

# 6 Linear-Chain Conditional Random Fields (CRFs)

The conditional random field (CRF) (Lafferty, A. McCallum, and Pereira 2001) is another graphical model that can be understood as a generalisation of the HMM, retaining a similar structure but adopting more flexible assumptions. This project will be concerned with only a subset of the family of CRFs, which form a generative-discriminative pair with the HMM: the linear-chain CRF.

[5] Despite being outperformed in other contexts, naïve Bayes excels at document classification tasks. Naïve Bayes is also easier to implement, less computationally demanding and requires less data than similar models and so may still be the correct choice of model depending on the intended use.

The linear-chain CRF can be understood graphically as an undirected HMM which permits any number of connections between the observed sequence $\mathbf{x}$ and the nodes of $\mathbf{y}$. The edges of a CRF no longer indicate processes of transition or emission, but instead a general dependency between the two nodes. This relaxations permits many new structures, allowing for a much richer context window and more reliable results when estimating $p(\mathbf{y} \mid \mathbf{x})$. Of particular interest is the resulting ability to incorporate new features without giving rise to concerns of inter-dependencies. We could include any number of features to create a CRF that is able to leverage an immense amount of contextual information, such features could include suffixes, capitalisation, domain-specific lexicon membership and more. Most crucially when compared to the HMM, we are able to utilise information that extends outside of the current index, for example by defining a feature which checks whether the word is preceded by 'the'. An example of potential linear-chain CRF structure is given in figure 2.

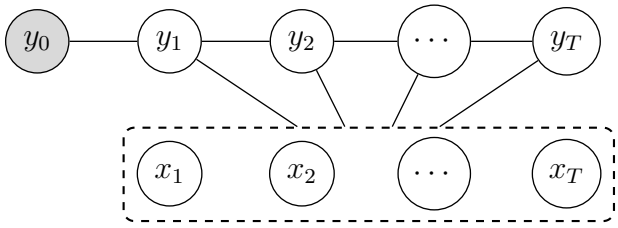



Figure 2: A linear-chain conditional random field applied to the part-of-speech tagging problem.

General CRFs are even less restrictive, allowing for connections to be drawn between non-neighbouring nodes of $\mathbf{y}$. However, this addition gives rise to considerably increased complexity even for singular inferences, requiring completely new graphical inference methods. The general CRF demands an immense amount of data and computing power to implement and so the linear-chain CRF can be seen as a more economical approach.

To define the linear-chain CRF we first note that the linear-chain CRF itself can be thought of as a HMM with particular feature functions. Hence, it is sensible to find a generalised but equivalent formulation of the HMM joint probability eq. (4), which we can then commute to find the conditional probability. This process is parallel to the transformation of the naïve Bayes to logistic regression. An appropriate form for such a generalisation is

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} \exp \left\{ \sum_{i,j \in \mathcal{Y}} \theta_{i,j} \mathbf{1}_{\{y_t = i\}} \mathbf{1}_{\{y_{t-1} = j\}} + \sum_{i \in \mathcal{Y}} \sum_{k \in \mathcal{X}} \mu_{k,i} \mathbf{1}_{\{y_t = i\}} \mathbf{1}_{\{x_t = k\}} \right\}, \tag{9}$$

where $\theta = \{\theta_{i,j}, \mu_{i,k}\}$ are the parameters of the HMM and $Z$ is a normalisation constant such that the distribution sums to one. Thankfully this formulation permits more flexibility without adding any new classes of model (Sutton, Andrew McCallum, et al. 2012). The HMM of the kind we have encountered can be summarised in this format by taking

$$\begin{aligned} \theta_{i,j} &= \log p(y_{t-1} = i \mid y_t = j) = \log a_{i,j}, \\ \mu_{i,k} &= \log p(x_t = k \mid y_t = i) = \log b_i(k), \\ Z &= 1 \end{aligned}$$

and declaring feature functions $f_{i,j}(y, y', x) = \mathbf{1}_{\{y=i\}} \mathbf{1}_{\{y'=j\}}$ and $f_{i,k}(y, y', x) = \mathbf{1}_{\{y=i\}} \mathbf{1}_{\{x=k\}}$. Hence, we can simplify eq. (9) to

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} \exp \left\{ \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t) \right\}.$$

We then apply this formula to derive the distribution of the linear-chain CRF:

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{p(\mathbf{y}, \mathbf{x})}{\sum_{\mathbf{y}'} p(\mathbf{y}', \mathbf{x})} = \frac{\prod_{t=1}^{T} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t)\right\}}{\sum_{\mathbf{y}'} \prod_{t=1}^{T} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y'_t, y'_{t-1}, x_t)\right\}}.$$

Since this model has arisen from a HMM, it is currently only configured to use the same feature set as the HMM. This is easily resolved by simply allowing the feature functions be more general. To do so neatly, we define $\mathbf{x}_t$ which is the vector of all features in $\mathbf{x}$ that the linear-chain CRF accesses across all of its feature functions at any one index $t$.

**Definition 6.1 (Linear Chain CRF)** *Let* $\mathbf{x}$, $\mathbf{y}$ *be random sequences/vectors variables.* $\theta_k$ *be parameter vector and* $\mathcal{F} = \{f_k(y, y', \mathbf{x}_t)\}_{k=1}^{K}$ *be a set of feature functions. The linear-chain conditional random field is then a probability distribution.*

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\}. \tag{10}$$

*Where* $Z(\mathbf{x})$ *is a normalising function over all values of* $\mathbf{y}$*:*

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{t=1}^{T} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\}.$$

Note, once more, that the linear-chain CRF can also be shown to be a factor graph by re-writing eq. (10) with the choice of $\Psi_t$

$$\Psi_t(y_t, y_{t-1}, x_t) = \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\}. \tag{11}$$

One may note the similarities of this distribution to the logistic regression model eq. (8). Indeed, as the HMM was a sequential extension of the naïve Bayes, the linear-chain CRF is a sequential logistic regression model. The interrelatedness of the methods in this project is summarised in fig. 1.

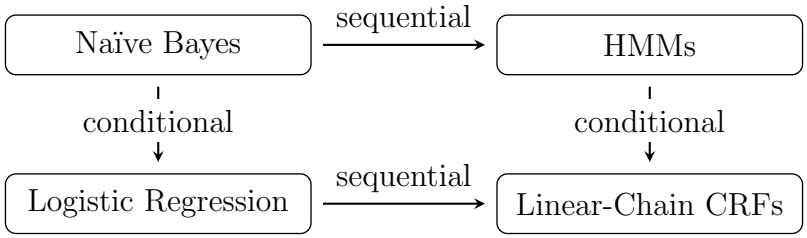



Figure 3: Diagram of the relationship between the naïve Bayes, logistic regression, HMMs and linear-chain CRFs. Adapted from Figure 2.4 of (Sutton, Andrew McCallum, et al. 2012)

## 6.1 Inference

Given the clear similarities between the two models, it should be unsurprising that the suite of iterative algorithms for HMMs derived in sections 3.1 and 3.2 can be easily adapted for use on linear-chain CRFs. Indeed the leaps in computational efficiency these algorithms make are even more crucial given the much larger set of features we would aim to incorporate in a CRF

model. First, we re-define the algorithms for the HMM, but this time using the factor graph notation introduced by eq. (6) that will more easily transfer to linear-chain CRFs.

$$\alpha_t(j) \coloneqq p(\mathbf{x}_{1:t}, Y_t = j) = \sum_{\mathbf{y}_{1:t-1}} \Psi(j, y_{t-1}, x_t) \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}),$$

$$\beta_t(j) \coloneqq p(\mathbf{x}_{t+1:T}, y_t = j) = \sum_{\mathbf{y}_{t+1:T}} \prod_{t'=t+1}^{T} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}),$$

$$\delta_t(j) \coloneqq \max_{\mathbf{y}_{1:t-1}} \Psi_t(j, y_{t-1}, x_t) \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}).$$

Next, the recursive steps are also written in terms of $\Psi_t$,

$$\begin{aligned}
&\text{Base Case:} && \alpha_1(j) = \Psi_1(j, y_0, x_1) && \\
&\text{Inductive Step:} && \alpha_t(j) = \sum_{i \in \mathcal{Y}} \Psi_t(j, i, x_t) \alpha_{t-1}(i) && \\
&\text{Base Case:} && \beta_T(i) = 1 && i, j \in \mathcal{Y} \\
&\text{Inductive Step:} && \beta_{t-1}(i) = \sum_{j \in \mathcal{Y}} \Psi_t(j, i, x_t) \beta_t(j) && 2 \leq t \leq T \\
&\text{Base Case:} && \delta_1(j) = \max_i \Psi_1(j, y_0, x_1) && \\
&\text{Inductive Step:} && \delta t(j) = \max_i \Psi_t(j, i, x_t) \delta_{t-1}(i) &&
\end{aligned}$$

Finally, we reformulate $\xi$ which we will need for parameter estimation:

$$\begin{aligned}
\xi_t(i, j) \coloneqq & p_\theta(y_t = i, y_{t+1} = j \mid \mathbf{x}) \\
= & \frac{\alpha_t(i) \Psi_t(i, j, x_{t+1}) \beta_{t+1}(j)}{p_\theta(\mathbf{x})} \qquad \begin{array}{l} i, j \in \mathcal{Y} \\ 1 \leq t \leq T - 1 \end{array}
\end{aligned}$$

This reformulation has not altered the values of the helper functions in any way and so these new identities can be applied in the forward-backward and Viterbi algorithms in exactly the same manner as before. Crucially, the factor graph representation presented here allows for these two algorithms to be easily extended to linear-chain CRFs by substituting $\Psi_t$ with its linear-chain CRF form (given in eq. (11)). We can therefore perform the forward-backward and Viterbi algorithms on linear-chain CRFs, with the sole side effect that the helper functions are no longer neatly interpretable as probability distributions. Instead, the forward-backward algorithm will now return the normalising function $Z(\mathbf{x})$ rather than $p_\theta(\mathbf{x})$,

$$Z(\mathbf{x}) = \sum_{i \in \mathcal{Y}} \alpha_T(i) = \beta_0(y_0).$$

Thankfully, this does not alter the computation of the marginal distributions:

$$\gamma_t(i) = p_\theta(y_t \mid \mathbf{x}) = \frac{\alpha_t(y_t) \beta_t(y_t)}{Z(\mathbf{x})},$$

$$\xi_t(i, j) = p_\theta(y_t = i, y_{t+1} = j \mid \mathbf{x}) = \frac{\alpha_t(i) \Psi_t(i, j, x_{t+1}) \beta_{t+1}(j)}{Z(\mathbf{x})}.$$

## 6.2 Parameter Estimation

The Baum-Welch algorithm is no longer appropriate when training the linear-chain CRF, indeed we have many more parameters and calculating the expected values for each would be highly

cumbersome, if not completely infeasible. Thankfully, there are several other approaches from the field of nonlinear optimisation which are well-suited to this task. In fact, the exponential form of the linear-chain CRF has highly desirable properties for optimisation. We begin with the typical log-likelihood function $L(\theta)$ which we wish to maximise,

$$L(\theta) = \sum_{n=1}^{N} \log p_\theta(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}).$$

We then substitute this function into the linear-chain CRF model eq. (10):

$$L(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \theta_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}). \tag{12}$$

It is then sensible to introduce a regularisation term $\frac{\theta_k^2}{2\sigma^2}$:

$$L(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \theta_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}) - \sum_{k=1}^{K} \frac{\theta_k^2}{2\sigma^2}.$$

Much like the earlier example of additive smoothing (eq. (5)) the regulariser will reduce excessively high variance, here by penalising weight vectors with norms that are too large. In this expression $\sigma^2$ is a free parameter that can be chosen to determine the level of penalisation. Regularisation is particularly important in the context of CRFs as it is not abnormal to encounter models with sets of parameters in the hundreds of thousands (Sutton, Andrew McCallum, et al. 2012). Both the log-likelihood function eq. (12) and the regularisation term we have chosen to add to it (the euclidean norm) are strictly concave, since both are products of non-negative numbers.[6] Since both our initial likelihood function and regulariser are strictly concave, any local solution we are able to converge to will provide globally optimal parameters. There are several approaches to finding this point of convergence although all will require the derivation of the partial derivative:

$$\frac{\partial L}{\partial \theta_k} = \sum_{i=1}^{N} \sum_{t=1}^{T} f_k(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)}) - \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{y,y'} f_k(y, y', x_t^{(i)}) p(y, y' \mid \mathbf{x}^{(i)}) - \frac{\theta_k}{\sigma^2}.$$

Gradient ascent is the simplest approach to implement, but is typically considered to converge too slowly. Newton methods are quicker but require the inverting incredibly large matrices as a result of our large parameter space, which is too complex. Instead, quasi-Newton approaches such as BFGS are most widely implemented and can favourably be used without need for particular specification.

## 6.3 Limitations of Linear-Chain CRFs

Despite the advantageous characteristics of linear-chain CRFs, it cannot be said that implementing such a model is a catch-all solution. There are several contexts where the generative approach offered by the HMM may indeed be preferable despite its limitations. Generative models are much less computationally demanding to implement and train and also tend to perform better on smaller datasets (Ng and Jordan 2001). Moreover, were the context for sequence labelling simple enough that a 'true' generative model could be built, then its estimates would be exact. The same cannot be said for models trainined discriminatively.

[6] The log-likelihood function is of the form $\log \sum_i \exp x_i$ which is universally convex.

One especially powerful use-case of HMMs applies an adapted Baum-Welch algorithm to train parameters on unsupervised data, that is, data that is not already annotated with POS tags (Kupiec 1992). Analogous methodologies do not exist for discriminative learning, limiting the linear-chain CRFs ability to tag languages with less widely available corpora.

Regardless, when the dataset is large and features are well chosen, CRFs will perform exceedingly well. Models that build upon the general CRF have been shown to achieve state-of-the-art results of up to 97.24% accuracy (tested on the Penn Treebank WSJ corpus, a standard industry benchmark) (Toutanova et al. 2004).

# References


Allen, James (1995). "Ambiguity Resolution: Statistical Methods". In: *Natural language understanding*. 2nd ed. Benjamin/Cummings, pp. 189–222. ISBN: 0805303340.

Baum, Leonard E. (1972). "An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes". In: *Inequalities III: Proceedings of the Third Symposium on Inequalities*. University of California, Los Angeles: Academic Press, pp. 1–8.

Baum, Leonard E. and Ted Petrie (1966). "Statistical Inference for Probabilistic Functions of Finite State Markov Chains". In: *The Annals of Mathematical Statistics* 37.6, pp. 1554–1563.

Caruana, Rich and Alexandru Niculescu-Mizil (June 2006). "An Empirical Comparison of Supervised Learning Algorithms". In: *Proceedings of the 23rd international conference on Machine learning - ICML '06* 2006, pp. 161–168.

Eisenstein, Jacob (2019). "Sequence Labelling". In: *Introduction to natural language processing / Jacob Eisenstein*. Adaptive computation and machine learning, pp. 268–311. ISBN: 9780262042840.

Kupiec, Julian (1992). "Robust part-of-speech tagging using a hidden Markov model". In: *Computer speech & language* 6.3, pp. 225–242.

Lafferty, J., A. McCallum, and F. Pereira (2001). "Conditional random fields: probabilistic models for segmenting and labeling sequence data". In: *Proceedings, Eighteenth International Conference on Machine Learning, ICML*, pp. 282–289.

Leech, Geoffrey (1995). *A Brief Users' Guide to the Grammatical Tagging of the British National Corpus*. http://www.natcorp.ox.ac.uk/docs/gramtag.html. Accessed: 2023-12-07.

Ng, Andrew and Michael Jordan (2001). "On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes". In: *Advances in Neural Information Processing Systems*. Ed. by T. Dietterich, S. Becker, and Z. Ghahramani. Vol. 14. MIT Press.

Rabiner, Lawrence R (1989). "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE*. Vol. 77. 2. Ieee, pp. 257–286.

Sutton, Charles, Andrew McCallum, et al. (2012). "An introduction to conditional random fields". In: *Foundations and Trends® in Machine Learning* 4.4, pp. 267–373.

Toutanova, Kristina et al. (Mar. 2004). "Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network". In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology—NAACL '03* 1.

Viterbi, A. (1967). "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". In: *IEEE transactions on information theory* 13.2, pp. 260–269. ISSN: 0018-9448.