Uncertainty Quantification in Conditional Random Fields for Optical Character Recognition

University of Massachusetts Amherst

Daniel Huang

September 21, 2021

Abstract

This thesis seeks to apply uncertainty quantification (UQ) techniques to conditional random field (CRFs) models specifically purposed for optical character recognition (OCR). CRFs are a particular class of probabilistic graphical models on undirected graphs that are widely used in many scientific areas e.g. image segmentation, image denoising, OCR and natural language processing. Here, a linear chain structured OCR CRF model is trained in Python. The resulting probabilistic graphical model is considered as a baseline model. Learning processes on parameters commonly used in building a CRF such as maximum likelihood estimation and/or the use of variational inference for approximating the "true" model involve uncertainties in the probability distribution that may propagate in predictions of certain quantities of interest (QoIs). We quantify these uncertainties by exploiting the Donsker-Varadhan Representation of the Kullback-Leibler (KL) Divergence and we derive tight UQ bounds for a particular class of QoIs expressed through indicator functions over parametric ambiguity sets. We finally focus on models that satisfy the Markov property and by using the developed approach, we can rank characters based on the impact their individual perturbations have on the overall prediction with respect to our quantities of interest.

Dedication

I dedicate this work to my family and friends. It was a long journey, but your constant support allowed me to push forwards. A special thanks to my parents Mei-Huei Jang and Kuan-Chun Huang for always being there for me and offering invaluable advice, my sister Emily for checking in on me, and my dog Sundae for always cheering me up after a long day.

I would also like to thank my fellow thesis-writers Zubin Baliga and Genglin Liu, who on top of the best friends someone could ask for, spent the year working on their theses alongside me. Their insight and camaraderie will not be forgotten.

Lastly, I would like to dedicate this thesis to Jennifer Lo. Thank you for always being there for me, taking care of me, and loving me. This thesis would not have been possible without your support.

Acknowledgements

I would like to express my deepest gratitude to my committee chair, Professor Panagiota Birmpa. Her dedication to teaching and research galvanized me, and without her guidance this thesis would not have been possible. She has been an amazing mentor and inspiration.

I would like to thank my other committee member Professor Markos Katsoulakis, whose work inspired the direction of my thesis and whose kind words encouraged me to continue onwards.

Contents

Chapter 1

Introduction

1.1 Optical Character Recognition

Optical Character Recognition (OCR) is a type of computer vision [9] where a machine, given a handwritten document or piece of text, returns the correct words in digital form. In the modern digital age, this technology has already become indispensable. Being able to convert a handwritten note into a digital form is present in our everyday lives, from depositing checks electronically to using Google Translate to read a sign in a foreign country. For this problem, we find that Conditional Random Fields (CRFs) are well-suited to developing a robust model as described in [14]. A brief description is given next, while its mathematical formulation is discussed in Chapter 3.

1.2 Conditional Random Fields

Conditional Random Fields (CRFs) are a special case of Markov Random Fields (MRFs) discussed in Chapter 2 when applied to model a conditional distribution. These distributions are used when we are given an input sequence of observed variables and we want to find the hidden (or unknown) state variable that needs to be inferred given the observations. These distributions are widely used in supervised learning settings also known as structured prediction. The two main types of models are generative and discriminative models. Generative models such as Hidden Markov Models (HMMs) model explicitly the distribution of individual classes, while their discriminative counterparts such as CRFs focus on learning the differences between individual classes instead [14]. Given the potential complexity of the given input data, we focus on CRFs in this thesis. This type of model is especially suited to OCR problems, as one is given a word from a sequence of black-and-white character images (input), and the predictor is a sequence of alphabet letters (output). Towards modeling such a problem, we need to segment the handwritten word in the image into pieces that each one includes a handwritten letter and assign a letter from the alphabet. This segmentation actually involves multiple steps, such as segmenting the image into lines of text (line segmentation), segmenting those lines into individual words (word segmentation), and only now segmenting individual words into characters (character segmentation). For the purposes of this thesis, we will assume those segmentation processes have already been completed, and

we are only working with the post-processed character images. A sample process for image processing and segmentation can be found at [1]. Additionally as the sequence of the pieces of the image forms a word, the predictions on different letters should inform each other. Equivalently, the conditional distribution that we model should be:

$$\text{P(correct word | black-and-white image).}$$

As we will see later in the corresponding section for CRFs, their conditional joint distribution is a type of a *Gibbs distribution* (see Chapter 2), i.e. given an input the conditional probability of a state is proportional to the exponential of some energy function. The energy function in CRFs should account for the dependency between a letter in the alphabet and its context in the word sequence and the dependency between neighboring letters. The parameters of the model are included in the energy functions. One can learn parameters of such a model by using maximum likelihood estimation given some data (see Chapter 4) and a prior knowledge set, such as a English corpus, in order to know which letter sequences are more common, and therefore more likely to occur. By including this extra information, our model is able to recognize letters and even words that could be ambiguous otherwise.

1.3 Uncertainty Quantification - The Results

Currently, many different OCR models are already in public use by companies such as Google and Bank of America. Many simple tests are used to evaluate these models by checking performance on a certain data set such as cross-validation [9]. These tests are highly dependent on the testing data, as if they are significantly different from the training data, it is unlikely the model will perform well. In this thesis, we want to quantify uncertainties involved in such models that are built from some approximation procedure (e.g. variational inference) and learning process on parameters from available data. These uncertainties are in the class of *epistemic or model form uncertainties* [4] i.e. those which stem from our lack of knowledge about the model coming from modeling and/or incomplete data and may be inherited in QoIs used in inference and in answering queries. Contrary to that, *aleatoric uncertainties* [4] are statistical and arise from the randomness in repeating the same experiment multiple times e.g. by using standard Monte Carlo schemes.

In this thesis, we accomplish the quantification of model form uncertainties arising from learning processes on parameters and approximation procedures on linear chain structured CRFs by exploiting the Donsker-Varadhan representation [5] as well as recent results in [4, 5, 7, 8] to derive robust UQ bounds for a class of QoIs which are expressed as indicator functions of events of interest explained in Chapter 6. The Donsker-Varadhan representation involves two main quantities: the moment generating function (MGF) and the KL Divergence discussed in Chapter 6 which is pseudometric between two probability distributions. More specifically and in the context of CRFs, we formulate all the above as follows:

Baseline Model. Let p be an approximate or baseline model for OCR CRF. In this thesis, we consider two types of CRFs as a baseline model. To be precise, given an input x with length $\ell(x)$ i.e. a sequence of black-and-white character images we consider a baseline linear

chain structured CRF such that its distribution defined on the universe of all words with length $\ell(x)$ and

C1. It is written as a product of some functions called *clique potentials* that depend on the current character, the preceding one and the given input, divided by a normalizing constant which is usually called *partition function.* Namely,

$$p(y|x) = \frac{1}{Z(x)} e^{\sum_{t=2}^{\ell(x)} \psi_t(y_{t-1}, y_t, x)}$$

for some functions $\psi_t(y_{t-1}, y_t, x)$ and $Z(x)$ is the partition function. This is the form of the baseline model coming from the Python code discussed in Chapter 8.

C2. It satisfies the Markov property [11] and hence its joint probability distribution can be factorized into conditional probabilities on each label of the character given the preceding character label and the input, i.e.

$$p(y|x) = p(y_1|x) \prod_{t=2}^{\ell(x)} p(y_t|y_{t-1}, x)$$

We focus on this model in Chapter 7, specifically where certain conditional distributions $p(y_t|y_{t-1}, x)$ are considered.

Ambiguity Sets. For both baseline models in classes C1 and C2, we consider (non-parametric) families $\mathcal{Q}^\eta$ of alternative models q that are η-close to a baseline model p with respect to the KL Divergence, also including the true model, i.e.

$$\mathcal{Q}^\eta = \left\{\text{all CRFs } q : D_{\text{KL}}(q\|p) \leq \eta\right\} \tag{1.1}$$

We call these families *ambiguity sets* and can be thought of as stress-tests of the QoI under perturbations of the baseline model. However the Markov Property (MP) satisfied by the class C2 allows us to consider other ambiguity sets that focus on one character at a time, i.e. for every $t \in \{1, \cdots, \ell(x)\}$

$$\begin{aligned} \mathcal{Q}^{\eta_t} \;=\; \Big\{&\text{all CRFs } q \text{ with MP} : D_{\text{KL}}(q(y_t|y_{t-1}, x)\|p(y_t|y_{t-1}, x)) \leq \eta_t \text{ for all } y_{t-1}, \\ &q(y_j|y_{j-1}, x) \equiv p(y_j|y_{j-1}, x) \text{ for all } j \neq t\Big\} \end{aligned} \tag{1.2}$$

Quantities of Interest. We define QoI's as the quantities that we seek to evaluate the model with respect to. The most intuitive QoI is either a word or a set of words. Let us introduce a motivating example. Assume we know our OCR CRF model will be used to process medical data. There are certain words such as diseases and other important medical terms that we want to ensure our model evaluates properly. These word(s) will be our QoI. However, as we will see in Chapter 6, our model parameters heavily depend on individual characters or pair of characters and positions as well. Thus, it makes sense to also define QoI's on sequences of characters, or individual characters. We define these QoI's as indicator functions of the event of interest.

Predictive Uncertainty. Then for a QoI g by using the Donsker-Varadhan representation, we get the following inequalities:

$$\pm \left(E_q[g] - E_p[g]\right) \leq \inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) + \log(E_p[e^{\pm c(g - E_p[g])}]) \right\} \tag{1.3}$$

For the particular choice of QoIs, we examine here, the above representation gives bounds for $q(A) - p(A)$ for some event of interest A. Moreover, the MGF is easily computable as we show in Chapter 6. In [4, 5, 7, 8], the authors have proved that the predictive uncertainty for g defined as $\sup_{q \in \mathcal{Q}^\eta} / \inf_{q \in \mathcal{Q}^\eta} \{E_q[g] - E_p[g]\}$ can be practically computed as:

$$\sup_{q \in \mathcal{Q}^\eta} / \inf_{q \in \mathcal{Q}^\eta} \{E_q[g] - E_p[g]\} = \inf_{c>0} \frac{1}{c} \left\{ \log(E_p[e^{\pm c(g - E_p[g])}]) + \eta \right\}. \tag{1.4}$$

In general, the KL divergence and the MGF are high-dimensional quantities that are expensive to compute. However, due to our choice of QoI and form of alternate models q, we find that this formula reduces to a one dimensional problem that is (numerically) computable once the MGF has been computed (or bounded) since one does not deal anymore with the KL divergence as is replaced by the parameter η. Finally in [7], tightness of the bounds have been studied, that is when the sup and inf are attained by appropriate distributions $p^\pm$ given by

$$p^{c_+} = \frac{e^{c+g}}{E_p[e^{c+g}]} p, \qquad \text{and} \qquad p^{c_-} = \frac{e^{-c-g}}{E_p[e^{-c-g}]} p. \tag{1.5}$$

where $c^\pm$ are the unique solutions of the equation $D_{\mathrm{KL}}(p^{c_\pm} \| p) = \eta$. $p^{c_\pm}$ stand for the *worst case scenarios*, the distributions for which we achieve the bounds in (1.3).

We implement all the above results to CRFs for OCR applications and we derive explicit formulas for the predictive uncertainty as well as we prove existence of the worst case scenarios and explicitly compute them. In particular, for a baseline model in the class C2 since the ambiguity set $\mathcal{Q}^{\eta_t}$ allows to examine the predictive uncertainty of each character one at time, we are able to rank the predictive uncertainty of each one from the least to most "dangerous". The strategy to accomplish this relies on a work in progress [2] as well as [6].

Chapter 2

Markov Random Fields

2.1 Introduction to MRFs

Before introducing Markov Random Fields (MRFs), let us briefly introduce Probabilistic Graphical Models (PGMs). A PGM is a probabilistic model with independent and dependent relationships, with the latter being represented by edges in a graph between nodes that represent random variables. For sake of brevity, we will proceed with describing MRFs. The interested reader may find the introduction in [9] useful in gaining an intuition regarding PGMs in general. MRFs are a specific type of PGM used when one does not have any evidence of directionality or simply interactions between features represented by nodes are bidirectional. More specifically, MRFs are undirected graphs satisfying the *Local Markov Properties*, i.e. if the neighbors of a node are observed then the node itself is independent of all the remaining ones.

2.2 The Ising Model

The Ising model is one of the most famous models in Statistical Mechanics, due to its simplicity and richness of its behaviour. In particular, the idea of the Ising model is that a physical system can be interpreted abstractly by lattice arrangements of particles. Each particle has a spin oriented either up or down. We then formally define such a system as follows:

Definition 2.2.1. *A d-dimensional lattice is defined as* $\Lambda := \mathbb{Z}^d$ *or as a finite subset of* $\mathbb{Z}^d$*, i.e.*

$$\Lambda_N = [-N, N]^d := \underbrace{[-N, N] \times \cdots \times [-N, N]}_{d} \subset \mathbb{Z}^d$$

for some positive integer N*. A spin at site* x *in* Λ *or* Λ_N *is expressed through a discrete random variable denoted* $\sigma(x)$ *and takes values* $\{-1, +1\}$*, i.e* $\sigma(x) = -1$ *or* $+1$*. The value* -1 *means that the spin is oriented down and the value* $+1$ *means that the spin is oriented up. A spin configuration is the collection of spins at each site of* Λ *[resp.* Λ_N*]:* $\sigma = \{\sigma(x) : x \in \Lambda\}$ *[resp.* $\sigma = \{\sigma(x) : x \in \Lambda_N\}$*]. The Ising phase space is the space of all possible spin configurations denoted by* $\mathcal{X}_\Lambda$ *[resp.* $\mathcal{X}_{\Lambda_N}$*], i.e* $\mathcal{X}_\Lambda = \{-1, 1\}^\Lambda$ *[resp.* $\mathcal{X}_\Lambda = \{-1, 1\}^{\Lambda_N}$*]*

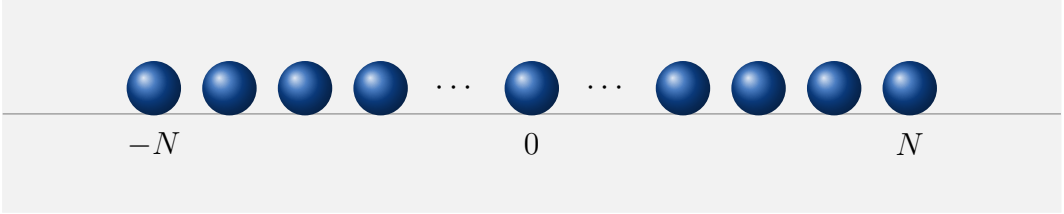


Figure 2.1: One-dimensional Ising spin lattice on Λ (white spins).

In general, the interaction of spins in the lattice as defined above is formulated as a function $J : \Lambda_N \times \Lambda_N \to \mathbb{R}$ such that $J(x, y)$ is the strength of the interaction between the spins located at x and y sites in the lattice. Note that when $J(x, y) = 0$, the spins at x and y do not interact. For the Ising model, we now turn our attention to the special case of nearest-neighbor (n.n) interaction, i.e

$$J_{n.n}(x, y) = \begin{cases} J, & x \neq y \quad \text{and} \quad |x - y| \leq 1 \\ 0, & otherwise \end{cases} \tag{2.1}$$

for some constant J. We also assume a constant external magnetic field, its intensity denoted by h, that is applied on the system. Then the Hamiltonian energy of a spin configuration σ with interaction J and external field h is defined as:

$$H^{J,h}(\sigma) := -J \sum_{x \in \Lambda_N} \sigma(x)\sigma(x + 1) - h \sum_{x \in \Lambda_N} \sigma(x) \tag{2.2}$$

The configuration probability, also known as Gibbs distribution, is given by the Boltzmann distribution with inverse temperature β

$$p_{\beta,J,h}(\sigma) = \frac{1}{Z^{J,h}} e^{\beta H^{J,h}(\sigma)} \tag{2.3}$$

where

$$Z^{\beta,J,h} = \sum_{\sigma \in \mathcal{X}_{\Lambda_N}} e^{\beta H^{J,h}(\sigma)} \tag{2.4}$$

is called partition function. Such a model is one of the first models which is considered as a Markov Random Field introduced in the next subsection.

2.3 Conditional Independence Properties and MRFs

Let us define three conditional independence properties that are necessary for MRFs.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and let $\mathbf{Y} = \{Y_i\}_{i=1}^{|\mathcal{V}|}$ be a set of random variables that each one is attached to a node. $|\mathcal{V}|$ denotes the cardinality of $\mathcal{V}$ and p is the joint probability distribution on $\mathbf{Y}$.

• **Pairwise Markov property (P)**: Any two non adjacent variables are conditionally independent (**CI**) given the rest, i.e. a conditional joint can be written as a product of conditional marginals; CI is denoted by $Y_i \perp Y_j \mid \{Y_k : k \neq i, j\}$,

• **Local Markov property (L)**: Any variable Y_i is conditionally independent of all the others given its neighbors, that is $Y_i \perp \{Y_k : k \notin \mathcal{N}_i\} \mid \{Y_k : k \in \mathcal{N}_i\}$,
• **Global Markov property (G)**: If A, B, C are sets of nodes then any two sets of variables, $\mathbf{Y}_A = \{Y_i : i \in A\}$ and $\mathbf{Y}_B = \{Y_i : i \in B\}$ are conditionally independent given a separating set of variables $\mathbf{Y}_C = \{Y_i : i \in C\}$, that is $\mathbf{Y}_A \perp \mathbf{Y}_B \mid \mathbf{Y}_C$.

We say that $(\mathbf{Y}, p)$ is a **Markov Random Field (MRF)** iff **(G)**, **(L)** and **(P)** are satisfied.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph and let $c \subset \mathcal{V}$.

(*i*) c is called **clique** if any pair of nodes in c is connected by some edge.

(*ii*) c is called **maximal clique** if any superset c' of c (i.e $c' \supset c$) is not a clique any more. The set of all maximal cliques of graph $\mathcal{G}$ is denoted by $\mathcal{C}_\mathcal{G}$.

As MRFs are defined on an undirected graph, it does not allow to use chain rule of conditional probabilities and further describe the probability distribution $p(\mathbf{y})$. However, the Hammersley-Clifford theorem stated below guarantees a factorization of the joint distribution.

Hammersley-Clifford Theorem *A positive distribution $p(\mathbf{y}) > 0$ satisfies one of* **(P), (L)** *and* **(G)** *of an undirected graph G iff p parametrized by some parameters $\mathbf{w} = \{\mathbf{w}_c\}_{c \in \mathcal{C}_\mathcal{G}}$ can be represented as a product of clique potentials, i.e*

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{1}{Z(\mathbf{w})} \prod_{c \in \mathcal{C}_\mathcal{G}} \Psi_c(\mathbf{y}_c \mid \mathbf{w}_c) \tag{2.5}$$

where $\Psi_c(\mathbf{y}_c \mid \mathbf{w}_c)$ is a non-negative function defined on the random variables in clique c and parametrized by some parameters $\mathbf{w}_c$, and is called **clique potential**. *Also $Z(\mathbf{w})$ is the partition function given by*

$$Z(\mathbf{w}) = \sum_{\mathbf{y}} \prod_{c \in \mathcal{C}_\mathcal{G}} \Psi_c(\mathbf{y}_c \mid \mathbf{w}_c) \tag{2.6}$$

Chapter 3

Conditional Random Fields

3.1 Introduction to Conditional Random Fields

Now that we have formally introduced MRFs, let us move on to the specific type of MRF that we will be focusing on - the Conditional Random Field. CRFs are a special case of MRFs, sometimes compared to directed graphs such as Bayesian Networks. We do not explicitly define Bayesian Networks in this thesis, but they can be helpful for gaining an intuition in the CRF model. The interested reader can learn about Bayesian Networks in Chapter 3 of [9]. CRFs are used to encode a conditional distribution, usually in the form $P(Y|X)$, where Y is a set of target variables and X is a set of disjoint observed variables [9]. CRFs are also considered partially directed, meaning that we have the undirected component over the target variables Y with the variables in X as the parents. CRFs excel at allowing us to incorporate a set of observed variables for which we may not know the relationships and dependencies into our domain without having to worry about characterizing the entire joint distribution. For example, in image segmentation, rather than having to define a joint distribution over the pixel values and regions of the image, we can define a conditional distribution over the segments given the pixel values in order to create a more intuitive model [9].

3.2 General CRFs

We define a conditional distribution $p(Y|X)$, where Y is the set of target variables and X is the disjoint set of observed variables. More formally, a CRF is an undirected graph encoded as an MRF over the set of nodes $X \cup Y$. However, instead of modeling the joint distribution $P(Y, X)$, a CRF attempts to model the conditional distribution $P(Y|X)$. The formal definition of a CRF adapted from [9] is given below:

Definition 3.2.1. *A conditional random field is an undirected graph H whose nodes correspond to $\boldsymbol{X} \cup \boldsymbol{Y}$; the network is annotated with a set of factors $\phi_1(\boldsymbol{D_1}), \ldots, \phi_m(\boldsymbol{D_m})$ such that each $\boldsymbol{D_i} \not\subset \boldsymbol{X}$. The network encodes a conditional distribution as follows:*

$$P(Y|X) = \frac{1}{Z(X)}\tilde{P}(Y, X) \tag{3.1}$$

$$\tilde{P}(Y, X) = \prod_{i=1}^{m} \phi_i(D_i) \tag{3.2}$$

$$Z(X) = \sum_{Y} \tilde{P}(Y, X) \tag{3.3}$$

$\tilde{P}(\boldsymbol{Y}, \boldsymbol{X})$ is known as the unnormalized factor product, and $Z(\boldsymbol{X})$ is the partition function. $Z(X)$ is sometimes called the normalizing function, as dividing the unnormalized factor product by it yields a probability distribution (sums to 1). Much of the strength of using a CRF comes from the fact that we are not defining a distribution over $\boldsymbol{X}$. Thus, we are able to include as many observed variables as we want without having to worry about dependencies or relationships between these predictors. Instead, all we care about is their impact on the target variables $\boldsymbol{Y}$.

3.3 Linear Chain CRFs

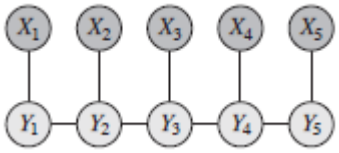


Figure 3.1: A linear chain conditional random field

Let us introduce a specific type of CRF, the linear chain CRF. It is defined very similarly to a general CRF, except the factor functions are more clearly defined:

Definition 3.3.1. *A linear chain conditional random field has* $\boldsymbol{Y} = \{Y_1, \ldots, Y_k\}$ *and* $\boldsymbol{X} = \{X_1, \ldots, X_k\}$. *There is an edge between edges* Y_i *and* Y_{i-1} *for* $i = 2, 3, \ldots, k$ *and edges* X_i *and* Y_i *for* $i = 1, 2, \ldots, k$. *An example is shown in figure 3.1.*

$$P(Y|X) = \frac{1}{Z(X)}\tilde{P}(Y, X) \tag{3.4}$$

$$\tilde{P}(Y, X) = \prod_{i=2}^{k} \phi_i(Y_{i-1}, Y_i) \prod_{i=1}^{k} \phi_i(Y_i, X_i) \tag{3.5}$$

$$Z(X) = \sum_{Y} \tilde{P}(Y, X) \tag{3.6}$$

Linear chain CRFs can be used to model the optical character recognition problem. When we are attempting to recognize the letters of a word, we care about not only what

the original image (X_i) looks like, but also what letter came before (Y_{i-1}). For example, it is much more likely to have the letter combination "qu" rather than the letter combination "qa", even if the "a" and "u" look similar. For purposes of this thesis, we will focus on linear chain CRFs.

Chapter 4

Methods

4.1 Parameter Estimation

When training a model, we need to determine the parameters that define the model. Many methods have been developed in order to determine these parameters from some given data, but we focus here on maximum likelihood estimation.

4.2 Maximum Likelihood Estimation

Maximum likelihood estimation (MLE) is a technique used to determine the parameters of a model for some underlying distribution. To give some intuition behind the idea of MLE, we begin with a hypothetical thought experiment. Let us assume we have some data for which we would like to determine the underlying distribution. MLE finds the parameters that maximize the likelihood of generating these data - thus the name Maximum Likelihood Estimation. To this end, we must define a likelihood function to be maximized using any known methods.

Now we begin by formally defining the learning problem. Assume we have a set of identically distributed and independent observed data $\mathcal{D}$ from some unknown distribution with parameters $\hat{\theta}$. We denote this set $\mathcal{D}$ as the *training set*, and we say that it has m instances of X: $\xi[1], \ldots, \xi[m]$.

Definition 4.2.1. *Given some data set $\mathcal{D}$, we choose parameters $\hat{\theta}$ from the parameter space Θ such that:*

$$L(\hat{\theta} : \mathcal{D}) = \max_{\theta \in \Theta} L(\theta : \mathcal{D}) \tag{4.1}$$

$$L(\theta : \mathcal{D}) = \prod_{m} P(\xi[m] : \theta) \tag{4.2}$$

We can often take advantage of the monotonically increasing nature of the log function in order to find the maximum of the log-likelihood function instead:

$$\ell(\theta : \mathcal{D}) = \sum_{m} \log(P(\xi[m] : \theta) \tag{4.3}$$

Computationally, this calculation is generally easier due to having a sum instead of a product. Note that due to the monotonic behavior of the log function, maximizing the likelihood function and the log-likelihood function are equivalent.

The term $P(\xi[m] : \theta)$ depends on the distribution we are modeling, so let us use an example to illustrate how MLE can be applied. Assume we have n data points coming from a Gaussian Normal distribution with unknown μ and σ. We know the density of a normal distribution to be given by:

$$P_{normal}(x : \mu, \theta) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where $\theta = (\mu, \sigma)$. The parameter space $\Theta_{gaussian} = \mathbb{R} \times \mathbb{R}^+$. We allow all real values for μ and only positive real values for σ. Now we can define the likelihood function to maximize:

$$L(\theta : \mathcal{D}) = \prod_n P(x_n : \mu, \sigma) = \prod_{k=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_k-\mu)^2}{2\sigma^2}}$$

With the respective log-likelihood function being:

$$\ell(\theta : \mathcal{D}) = \sum_{k=1}^{n} \log(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_k-\mu)^2}{2\sigma^2}})$$

To find the maximum likelihood parameters we simply take the partial derivatives with respect to μ and σ and set them equal to 0. Doing the calculations gives:

$$\hat{\mu} = \frac{1}{n} \sum_{k=1}^{n} x_k \tag{4.4}$$

$$\hat{\sigma^2} = \frac{1}{n} \sum_{k=1}^{n} (x_k - \hat{\mu})^2 \tag{4.5}$$

We see that using MLE in fact gives what we know to be the definitions of μ and σ respectively. In general, we can utilize MLE to determine the parameters of our distribution by maximizing the probability that our data is generated by those specific parameters. The parameters of the baseline model p are found by MLE in Python.

Chapter 5

Uncertainty Quantification

5.1 KL Divergence

The Kullback-Leibler Divergence between two probability distributions is a quantity measuring the "distance" between the two distributions coming from information theory. For two probability distributions q and p, the KL divergence is defined as follows:

$$D_{\mathrm{KL}}(q||p) = E_q\left[\log\left(\frac{q}{p}\right)\right] \tag{5.1}$$

$$= \sum q(x)\log\left(\frac{q}{p}\right) \quad \text{for q,p discrete} \tag{5.2}$$

$$= \int \log\left(\frac{q}{p}\right)\, dq \quad \text{for q,p continuous} \tag{5.3}$$

In this paper we focus on discrete models, so we will work with (5.2). Assigning bounds for our summation term we get:

$$D_{\mathrm{KL}}(q||p) = \sum_{i=1}^{N} q(x_i)\log\left(\frac{q(x_i)}{p(x_i)}\right) \tag{5.4}$$

The KL divergence between two distributions can be thought of as the information lost by using one distribution over the other. In our thesis, we will focus on using the model generated by [12] in Python as a baseline model with which we will compare other models.

5.2 Donsker-Varadhan Representation

The Donsker-Varadhan Representation tells us the following:

$$D_{\mathrm{KL}}(q||p) = \sup_{g:\, x\to\mathbb{R}} \{E_q[g(X)] - \log(E_p[e^{g(x)}])\} \tag{5.5}$$

Equivalently, we have:

$$D_{\mathrm{KL}}(q||p) \geq E_q[g(X)] - \log(E_p[e^{g(x)}]) \tag{5.6}$$

Remark 5.2.1. *The term $E_p[e^{g(X)}]$ is known as the moment generating function of g.*

As mentioned earlier, we only focus on the discrete version of the formula. In order to prove the DV formula we know that we are looking for the function g that maximizes the difference $E_q[g(X)] - \log(E_p[e^{g(x)}])$. Thus, we take the derivative of the term with respect to the $g(i)$'s:

$$\left(E_q[g(X)] - \log(E_p[e^{g(X)}])\right)_{g(i)}$$

Now we substitute termwise using the definition of expectation over all states i:

$$\left(\sum_i g(i)q(i) - \log(\sum_i e^{g(i)}p(i))\right)_{g(i)}$$

Now we get for every i the equation:

$$q(i) - \frac{e^{g(i)}p(i)}{\sum_i e^{g(i)}p(i)} = 0$$

Let the term $\sum_i e^{g(i)}p(i)$ be a:

$$q(i) - \frac{e^{g(i)}p(i)}{a} = 0$$

Rearranging:

$$a \times q(i) = e^{g(i)} \times p(i)$$

Now taking log of both sides, using log properties, and rearranging:

$$\log(a) + \log(q(i)) = g(i) + \log(p(i))$$
$$g(i) = \log(a) + \log\left(\frac{q(i)}{p(i)}\right)$$

Noticing that the last term looks remarkably like the KL divergence we saw earlier, we multiply both sides by $q(i)$ and take the sum over all i for both sides:

$$g(i) = \log(a) + \log\left(\frac{q(i)}{p(i)}\right)$$
$$\sum_i g(i)q(i) = \sum_i \log(a)q(i) + \sum_i q(i)\log\left(\frac{q(i)}{p(i)}\right)$$

We see that the last term is indeed the KL divergence, and we can rewrite the other terms with respect to expectation again:

$$E_q(g) = \log(E(e^g)) + D_{\mathrm{KL}}(q||p)$$

Thus, recalling that this value of g is the one that indeed gives us the maximum of the term $E_q[g(X)] - \log(E_p[e^{g(X)}])$, we conclude that:

$$D_{\mathrm{KL}}(q||p) = \sup_{g:\, x\to\mathbb{R}} \{E_q[g(X)] - \log(E_p[e^{g(x)}])\} \quad \square.$$

5.3 DV Representation for Centered QoI's

To illustrate the applications of the DV representation, let us see a relevant example:

$$\text{Let } \bar{g} := g - E_p[g]$$

Then plugging into (5.6) and rearranging, we get:

$$E_q[\bar{g}] \le D_{\mathrm{KL}}(q||p) + \log(E_p[e^{\bar{g}}])$$

plugging in the $g - E_p[g]$ for $\bar{g}$:

$$E_q[g - E_p[g]] \le D_{\mathrm{KL}}(q||p) + \log(E_p[e^{g-E_p[g]}])$$

We take advantage of the linearity of expectation as well as the fact that $E_q[E_p[g]] = E_p[g]$ to get:

$$E_q[g] - E_p[g] \le D_{\mathrm{KL}}(q||p) + \log(E_p[e^{g-E_p[g]}]) \tag{5.7}$$

This formula tells us that we can bound the difference in expectation with respect to two different distributions by their KL divergence and the log of their MGF. To see how this could be useful, consider $q = p + \theta$. Then, the formula above allows us to see how much the distribution q changes with some small or large perturbation θ.

Another example can be considered here is when one plugs in (5.6) the quantity $c(g - E_p[g])$ for some positive parameter c. Then, after taking the inf with respect to c, (5.6) has the form

$$E_q[g] - E_p[g] \le \inf_{c>0} \frac{1}{c}\left\{D_{\mathrm{KL}}(q||p) + \log(E_p[e^{c(g-E_p[g])}])\right\} \tag{5.8}$$

$$E_q[g] - E_p[g] \geq \sup_{c>0} \frac{1}{c} \left\{ -D_{\mathrm{KL}}(q||p) - \log(E_p[e^{-c(g-E_p[g])}]) \right\} \tag{5.9}$$

If we calculate (5.8) further using the properties of expectation:

$$\begin{aligned} E_q[g] - E_p[g] &\leq \inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) + \log(E_p[e^{c(g-E_p[g])}]) \right\} \\ &\leq \inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) + \log(e^{-cE_p[g]} E_p[e^{cg}]) \right\} \\ &\leq \inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) - cE_p[g] + \log(E_p[e^{cg}]) \right\} \\ &\leq -E_p[g] + \inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) + \log(E_p[e^{cg}]) \right\} \end{aligned} \tag{5.10}$$

Finally we can get an optimal upper bound on $E_q(g)$:

$$E_q[g] \leq \inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) + \log(E_p[e^{cg}]) \right\} \tag{5.11}$$

For (5.9) we utilize the property $-\sup\{f(-x)\} = \inf\{f(x)\}$ and convert the lower bound into a very similar bound:

$$\begin{aligned} E_q[g] - E_p[g] &\geq \sup_{c>0} \frac{1}{c} \left\{ -D_{\mathrm{KL}}(q||p) - \log(E_p[e^{-c(g-E_p[g])}]) \right\} \\ &\geq -\inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) + \log(E_p[e^{-c(g-E_p[g])}]) \right\} \\ &\geq -\inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) + \log(e^{cE_p[g]} E_p[e^{-cg}]) \right\} \\ &\geq -\inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) + cE_p[g] + \log(E_p[e^{-cg}]) \right\} \\ &\geq -E_p[g] - \inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) + \log(E_p[e^{-cg}]) \right\} \end{aligned} \tag{5.12}$$

Thus we get the respective optimal lower bound on $E_q(g)$:

$$E_q[g] \geq -\inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) + \log(E_p[e^{-cg}]) \right\} \tag{5.13}$$

5.4 Tightness of UQ bounds (5.8) and Optimal Distributions

The main question of this section is

Do there exist probability distributions that the UQ bounds given by (5.8) *are attained?*

To answer the question, we turn our attention to the mathematical formulation of the model-form uncertainty:

Let p be the probability distribution of a baseline model. We define families of probabilistic models called ambiguity sets as follows:

$$\mathcal{Q}^\eta := \{q : D_{\mathrm{KL}}(q\|p) \leq \eta\} \tag{5.14}$$

for some positive parameter η that stands for the size of the set. Then, we define the *predictive uncertainty* (or bias) for the QoI g when using the baseline model p instead of any alternative model $q \in \mathcal{Q}^\eta$ as the two worst case scenarios:

$$I^\pm(g, p; \mathcal{Q}^\eta) := \sup_{q\in\mathcal{Q}^\eta}/\inf_{q\in\mathcal{Q}^\eta} \{E_q[g] - E_p[g]\} \tag{5.15}$$

where $E_q[g]$ denotes the expected value of the QoI g. In recent work [4, 5, 7, 8], it has been shown that $I^\pm(g, p; \mathcal{D}^\eta)$ (an infinite dimensional optimization problem) can be directly computable by a one dimensional optimization problem:

$$I^\pm(g, p; \mathcal{Q}^\eta) = \pm \inf_{c>0} \left[\frac{1}{c}\log(E_p[e^{\pm c(g-E_p[g])}]) + \frac{\eta}{c}\right] = E_{p^\pm}[g] - E_p[g]. \tag{5.16}$$

Moreover, in [7] the authors have proven the following result:

Theorem 5.4.1. *Let p be a probability distribution and $g(y)$ be a QoI with finite MGF $E_p[e^{cg(y)}]$ in a neighborhood of the origin. Then there exist $0 < \eta_\pm \leq \infty$ such that for any $\eta \leq \eta_\pm$ there exist probability measures $p^\pm = p^\pm(\eta) \in \mathcal{Q}^\eta$ such that (5.8) become an equality. Furthermore, $p^\pm = p^{c_\pm}$ with*

$$p^{c_\pm} = \frac{e^{\pm c_\pm g}}{E_p[e^{\pm c_\pm g}]}p \tag{5.17}$$

and $c_\pm$ being the unique solutions of $D_{KL}(p^{c_\pm}\|p) = \eta$.

Chapter 6

UQ Formulation

6.1 Definition of OCR Linear Chain CRF

In general, the conditional joint distribution of a linear-chain-structured CRF is a type of a *Gibbs distribution* (see Chapter 2), i.e given an input x with length ℓx the conditional probability of a state is proportional to the exponential of some energy function. The energy function in CRFs should account for the dependency between a letter in the alphabet y_t and its context in the word sequence x_t and the dependency between neighboring letters y_{t-1} and y_t. The former is denoted by $A_t(y_t, x)$ while the latter is denoted by $I_t(y_{t-1}, y_t, x)$. The parameters of the model are also in the energy functions and are split into *state* and *transition* lying in A_t and I_t respectively. To be precise, the conditional distribution defined on the universe of all words with length $\ell(x)$ is given by:

$$p(y|x) = \frac{\exp\left(\sum_{t=2}^{\ell(x)} I_t(y_{t-1}, y_t, x) + \sum_{t=1}^{\ell(x)} A_t(y_t, x)\right)}{Z(x)} \tag{6.1}$$

Note that $y = (y_1, \ldots, y_{t-1}, y_t, \ldots, y_{\ell(x)})$ where each y_t is a character/alphabet letter with $t \in \{1, \ldots, \ell(x)\}$. $Z(x)$ is the partition function, i.e.

$$Z(x) = \sum_y \exp\left[\sum_t \left(I_t(y_{t-1}, y_t, x) + A_t(y_t, x)\right)\right] \tag{6.2}$$

6.2 Special classes of CRFs

In this thesis, we focus on two special classes of linear-chain-structured CRFs:

C1. The distributions of CRFs in this class have the form of (6.1) with

$$I_t(y_{t-1}, y_t, x) = \sum_{i,j \in S} \lambda_{ij} \mathbb{1}_{y_{t-1}=i} \mathbb{1}_{y_t=j} \tag{6.3}$$

$$A_t(y_t, x) = \sum_{i \in S} \sum_{o \in O} \mu_{io} \mathbb{1}_{y_t=i} \mathbb{1}_{x_t=o} \tag{6.4}$$

where λ_{ij} and μ_{io} for transition and state parameters respectively, and i, j come from the total alphabet S and o is the binarized handwritten character. For example, let $y =$ tree. Then y will have $t \in \{1, 2, 3, 4\}$ as follows:

$$y = \underset{y_1}{\_\_\_\_} \quad \underset{y_2}{\_\_\_\_} \quad \underset{y_3}{\_\_\_\_} \quad \underset{y_4}{\_\_\_\_}$$

And a corresponding 4 letter word x as follows:

$$x = \underset{x_1}{\_\_\_\_} \quad \underset{x_2}{\_\_\_\_} \quad \underset{x_3}{\_\_\_\_} \quad \underset{x_4}{\_\_\_\_}$$

Thus, if we are interested in calculating $I_3(y_2, y_3, x)$ for the word "tree" we would do:

$$\begin{aligned} I_3(y_2, y_3, x) &= \sum_{i,j \in S} \lambda_{ij} \mathbb{1}_{y_2=i} \mathbb{1}_{y_3=j} \\ &= \sum_{i,j \in S} \lambda_{ij} \mathbb{1}_{r=i} \mathbb{1}_{e=j} \\ &= \lambda_{re} \end{aligned} \tag{6.5}$$

And likewise for $A_3(y_3, x)$:

$$\begin{aligned} A_3(y_3, x) &= \sum_{i \in S} \sum_{o \in O} \mu_{io} \mathbb{1}_{y_t=i} \mathbb{1}_{x_t=o} \\ &= \sum_{i \in S} \sum_{o \in O} \mu_{io} \mathbb{1}_{e=i} \mathbb{1}_{x_t=o} \end{aligned} \tag{6.6}$$

Note that this is as far as we can simplify due to o referring to the $\{0, 1\}$ representation of the character on site x_3.

C2. The distributions of CRFs in this class satisfy the Markov Property defined as

$$p(y|x) = p(y_1|x)p(y_2|y_1, x) \cdots p(y_{\ell(y)}|y_{\ell(y)-1}, x) = \prod_{t=1}^{\ell(y)} p(y_t|y_{t-1}, x) \tag{6.7}$$

with

$$p(y_t|y_{t-1} = j_{t-1}, x) = \begin{cases} \frac{\exp\left(\sum_i \lambda_{ij_{t-1}} \mathbb{1}_{y_t=i} + \sum_i \sum_o \mu_{io} \mathbb{1}_{y_t=i} \mathbb{1}_{x_t=o}\right)}{\sum_{y_t} \exp\left(\sum_i \lambda_{ij_{t-1}} \mathbb{1}_{y_t=i} + \sum_i \sum_o \mu_{io} \mathbb{1}_{y_t=i} \mathbb{1}_{x_t=o}\right)}, & t = 2, 3, 4 \\ \frac{\exp\left(\sum_i \lambda_i \mathbb{1}_{y_t=i} + \sum_i \sum_o \mu_{io} \mathbb{1}_{y_t=i} \mathbb{1}_{x_t=o}\right)}{\sum_{y_1} \exp\left(\sum_i \lambda_i \mathbb{1}_{y_t=i} + \sum_i \sum_o \mu_{io} \mathbb{1}_{y_t=i} \mathbb{1}_{x_t=o}\right)}, & t = 1 \end{cases} \tag{6.8}$$

6.3 Quantities of Interest

The reader may notice that the DV representation leaves the function g undefined. This function g is known as a quantity of interest, some quantity that we would like to examine deeply. However, for our model, we do not have quantitative data from which the model is learned. Instead, we have words and letters that may be difficult to represent as a function in the traditional sense. It is here that we introduce the indicator function.

Definition 6.3.1. *An indicator function of a subset A of a universal set Y is a function:*

$$\mathbb{1}_A : Y \to \{0, 1\} \tag{6.9}$$

where:

$$\mathbb{1}_A(y) = \begin{cases} 1 & \text{if } y \in A \\ 0 & \text{if } y \notin A \end{cases} \tag{6.10}$$

In general, the space Y is the space of all words that we can make using our alphabet. However, once we have our image x_t segmented into characters, we actually know how many letters are in the word we are looking for. Thus, for our purposes let us assume Y to be the space of 4 letter words. Indicator functions have the following property:

Property of Indicator Functions. For an indicator function $g(y) = \mathbb{1}_A(y)$ we have:

$$E_p(g) = p(A) \tag{6.11}$$

where A is the event of interest corresponding to g.

Additionally, we do not necessarily need to define entire words as our quantities of interest. In fact, we can introduce a QoI defined on certain positions in the word:

$$g(y_1, y_2) = \mathbb{1}_{y_1=t}\mathbb{1}_{y_2=r} \tag{6.12}$$

This is the QoI which focuses on the first letter to be "t" and the second letter to be "r". For further generality we can even use a single letter or leave the positions undefined (i.e. words containing "e" or words with "e" as the t^{th} letter). Thus, another class of QoIs is considered here is

$$g(y_t) = \mathbb{1}_{y_t=j} \tag{6.13}$$

where j is a letter of the alphabet. These QoI's are defined on positions within a word, rather than the whole word.

The property given in (6.11) allows us to write the expectation of some QoI as the probability of some event of interest. For example, for the QoI defined in (6.12):

$$E_p[g(y_1, y_2)] = p(B), \tag{6.14}$$

where B is the set of 4 letter words that begin with "tr". Notice that despite having a QoI defined only a few positions, we have ended up at a probability over our entire universe. In this way, we see that we can define any QoI we like (e.g. word, letters, letter) and still be working with probability in the same space.

6.4 Moment Generating Function

Now that we have all the tools necessary, let us finally apply our variational tools to the OCR CRF model. In particular, we will focus on the quantity of interest:

$$\begin{aligned} g(y_1, y_2) &= \mathbb{1}_{tr}(Y) && (6.15) \\ &= \begin{cases} 1 & \text{if } Y \text{ begins with "tr"} \\ 0 & \text{if } Y \text{ begins with "tr"} \end{cases} && (6.16) \end{aligned}$$

Essentially, we want to evaluate our model with respect to how well it models certain words (in our case, words that begin with "tr"). First, let us calculate the moment generating function of our QoI $g(y_1, y_2)$:

$$\begin{aligned} E_p[e^{g(y_1,y_2)}] &= \sum_y e^{g(y_1,y_2)} p(Y = y) \\ &= \sum_{y=\text{"tr..."}} e^{g(y_1,y_2)} p(Y = y) + \sum_{y\neq\text{"tr..."}} e^{g(y_1,y_2)} p(Y = y) \\ &= ep(y = \text{"tr..."}) + \sum_{y\neq\text{"tr..."}} e^0 p(y \neq \text{"tr..."}) \end{aligned} \quad (6.17)$$

Let $p(\text{tr}) = p(Y = \text{"tr..."})$ and notice that $p(y \neq \text{"tr..."}) = 1 - p(\text{tr})$. Then:

$$\begin{aligned} E_p[e^{g(y_1,y_2)}] &= ep(\text{tr}) + 1 - p(\text{tr}) \\ E_p[e^{g(y_1,y_2)}] &= 1 - p(\text{tr})(1 - e) \end{aligned} \quad (6.18)$$

In general, for some subset B of our universe of possible 4 letter words, we have:

$$\begin{aligned} E_p[e^{g(y)}] &= ep(B) + 1 - p(B) \\ E_p[e^{g(y)}] &= 1 - p(B)(1 - e) \end{aligned} \quad (6.19)$$

Now we have a formula for the MGF of a general function $g(y)$. Remember that in order to find the uncertainty bounds using (5.8) and (5.9) we need the MGF of $c(g - E_p[g])$ and $-c(g - E_p[g])$ for $c > 0$. We define B as the specific subset of words corresponding to our QoI - 4 letter words that begin with "tr...". and calculate:

$$\begin{aligned} E_p[e^{c(g-E_p[g])}] &= E_p[e^{cg-cE_p[g]}] \\ &= e^{-cE_p[g]} E_p[e^{cg}] \\ &= e^{-cE_p[g]}(1 - p(B)(1 - e^c)) \end{aligned} \quad (6.20)$$

$$\begin{aligned} E_p[e^{-c(g-E_p[g])}] &= E_p[e^{-cg+cE_p[g]}] \\ &= e^{cE_p[g]}E_p[e^{-cg}] \\ &= e^{cE_p[g]}(1-p(B)(1-e^{-c})) \end{aligned} \tag{6.21}$$

Thus we have calculated the necessary MGF's in order to use the uncertainty bounds from (5.8) and (5.9). Next we need to calculate the KL Divergence between our distributions q and p.

6.5 KL Divergence between two linear-chain CRFs

In order to utilize the Donsker-Varadhan representation for two linear chain CRFs, we must calculate D_{KL} between two such models. In order to do so, let us define the following:

- Let x be a binarized ($\{0,1\}$) representation of a word considered as a image of pixels. Let us assume that after segmentation processes, the length has been identified as 4.
- Let p be defined as a linear chain CRF as in (6.1) with **known** state parameters $\boldsymbol{\mu_{io}}$, known transition parameters $\boldsymbol{\lambda_{ij}}$, and partition function $\boldsymbol{Z(x)}$. The state and transition are parameters are estimated using MLE.
- Let q be defined as a linear chain CRF as in (6.1) with **unknown** state parameters $\boldsymbol{\mu'_{io}}$, **unknown** transition parameters $\boldsymbol{\lambda'_{ij}}$, and partition function $\boldsymbol{Z'(x)}$.

6.5.1 Likelihood ratio

The likelihood ratio $\frac{q}{p}$ is computed as follows:

$$\frac{q}{p} = \frac{Z(x)}{Z'(x)} \exp\left(\sum_t \left[\sum_{i,j}(\lambda'_{ij}-\lambda_{ij})\mathbb{1}_{y_{t-1}=i}\mathbb{1}_{y_t=j} + \sum_i\sum_o(\mu'_{io}-\mu_{io})\mathbb{1}_{y_t=i}\mathbb{1}_{x_t=o}\right]\right) \tag{6.22}$$

Remark 6.5.1. *Note that the transition parameters begin from $t=2$, while the state parameters begin from $t=1$. We write the summation over all values of t for sake of clarity, but please keep this in mind.*

Note that we have grouped the λ_{ij} and λ'_{ij} terms together, as well as the μ_{ij} and μ'_{ij} terms. We have also used the properties of exponents to combine all of them into one term. While technically it is incorrect to sum over all values t for the transition parameters, for the clarity we keep this notation. Now we take the log of the likelihood ratio:

$$\log\left(\frac{q}{p}\right) = \sum_t \left[\sum_{i,j}(\lambda'_{ij}-\lambda_{ij})\mathbb{1}_{y_{t-1}=i}\mathbb{1}_{y_t=j} + \sum_i\sum_o(\mu'_{io}-\mu_{io})\mathbb{1}_{y_t=i}\mathbb{1}_{x_t=o}\right] + \log\left(\frac{Z(x)}{Z'(x)}\right) \tag{6.23}$$

Now, we are interested in comparing models that are only slightly perturbed from each other, meaning their parameters are closely related.

Lemma 6.5.2. *For p and q in class C1 defined in* (6.2) *such that*

$$\lambda_{ij}^{'} = \begin{cases} \lambda_{ij}, & ij \neq an, \\ (1+\gamma)\lambda_{an}, & ij = an \end{cases} \tag{6.24}$$

for some $\gamma \in [-1, 1]$ and $\mu_{io}^{'} = \mu_{io}$, the likelihood ratio is given by

$$\log\left(\frac{q(y|x)}{p(y|x)}\right) = \sum_{t=2}^{\ell(x)} (\gamma\lambda_{an}) \mathbb{1}_{y_{t-1}=a} \mathbb{1}_{y_t=n} + \log\left(\frac{Z(x)}{Z'(x)}\right) \tag{6.25}$$

Proof. Let y be a word, i.e. $y = (y_1, \dots y_4)$. Then the likelihood ratio of q over p where the parameters of q are given by (6.27) is computed:

$$\begin{aligned} \log\left(\frac{q(y|x)}{p(y|x)}\right) &= \sum_t \left[\sum_{i,j} (\lambda_{ij}^{'} - \lambda_{ij}) \mathbb{1}_{y_{t-1}=i} \mathbb{1}_{y_t=j} + \sum_i \sum_o (\mu_{io}^{'} - \mu_{io}) \mathbb{1}_{y_t=i} \mathbb{1}_{x_t=o} \right] \\ &\quad + \log\left(\frac{Z(x)}{Z'(x)}\right) \\ &= \sum_t \Big[\sum_{i,j \neq an} (\lambda_{ij}^{'} - \lambda_{ij}) \mathbb{1}_{y_{t-1}=i} \mathbb{1}_{y_t=j} + (\lambda_{an}^{'} - \lambda_{an}) \mathbb{1}_{y_{t-1}=a} \mathbb{1}_{y_t=n} \\ &\quad + \sum_i \sum_o (\mu_{io}^{'} - \mu_{io}) \mathbb{1}_{y_t=i} \mathbb{1}_{x_t=o} \Big] + \log\left(\frac{Z(x)}{Z'(x)}\right) \\ &= \sum_{t=2}^{\ell(x)} (\gamma\lambda_{an}) \mathbb{1}_{y_t=a} \mathbb{1}_{y_t=n} + \log\left(\frac{Z(x)}{Z'(x)}\right) \end{aligned} \tag{6.26}$$

□

6.5.2 Partition Function

Lemma 6.5.3. *For p and q in class C1 defined in* (6.2) *such that*

$$\lambda_{ij}^{'} = \begin{cases} \lambda_{ij}, & ij \neq an, \\ (1+\gamma)\lambda_{an}, & ij = an \end{cases} \tag{6.27}$$

for some $\gamma \in [-1, 1]$ and $\mu_{io}^{'} = \mu_{io}$, the partition function of q is given by

$$\begin{aligned} Z'(x) &= Z(x) E_p \left[\exp\left(\sum_{t=2}^{\ell(x)} (\lambda_{an}^{'} - \lambda_{an}) \mathbb{1}_{y_t=a} \mathbb{1}_{y_t=n} \right) \right] && (6.28) \\ &= Z(x) \left(\sum_{k=1}^{m} p(A_k) e^{k(\gamma\lambda_{an})} + 1 - p(A) \right) && (6.29) \end{aligned}$$

6.5.3 KL Divergence

In this subsection, based on Lemma 6.5.2, we focus on the computation of KL divergence.

Lemma 6.5.4. *For p and q in class C1 defined in (6.2) such that*

$$\lambda_{ij}^{'} = \begin{cases} \lambda_{ij}, & ij \neq an, \\ (1+\gamma)\lambda_{an}, & ij = an \end{cases} \tag{6.30}$$

for some $\gamma \in [-1, 1]$ and $\mu_{io}^{'} = \mu_{io}$, the KL divergence between q and p is given by

$$\begin{aligned} D_{\mathrm{KL}}(q\|p) &= c(x) \sum_{k=1}^{m} p(A_k) e^{k(\gamma\lambda_{an})} k(\gamma\lambda_{an}) \\ &\quad + \log c(x) \end{aligned} \tag{6.31}$$

where $A_k = \{$all words y that contain k times the pair $an\}$ with $A = \cup_{k=1}^{m} A_k$ for some integer $k > 0$,

$$c(x) = \frac{Z(x)}{Z'(x)} = \left(\sum_{k=1}^{m} p(A_k) e^{k(\gamma\lambda_{an})} + 1 - p(A) \right)^{-1}$$

Proof.

$$\begin{aligned} D_{\mathrm{KL}}(q\|p) &= E_q \left[\log \frac{q}{p} \right] = \sum_y \left(\log \frac{q(y)}{p(y)} \right) q(y) \\ &= \sum_y \left(\log \frac{q(y|x)}{p(y|x)} \right) \frac{q(y|x)}{p(y|x)} p(y|x) = E_p \left[\frac{q}{p} \log \frac{q}{p} \right] \\ &= E_p \Big[\frac{Z(x)}{Z'(x)} \exp \left(\sum_t (\lambda_{an}^{'} - \lambda_{an}) \mathbb{1}_{y_t=a} \mathbb{1}_{y_t=n} \right) \\ &\qquad \times \left(\sum_t (\lambda_{an}^{'} - \lambda_{an}) \mathbb{1}_{y_t=a} \mathbb{1}_{y_t=n} + \log \left(\frac{Z(x)}{Z'(x)} \right) \right) \Big] \\ &= \frac{Z(x)}{Z'(x)} E_p \left[\exp \left(\sum_t (\lambda_{an}^{'} - \lambda_{an}) \mathbb{1}_{y_t=a} \mathbb{1}_{y_t=n} \right) \sum_t (\lambda_{an}^{'} - \lambda_{an}) \mathbb{1}_{y_t=a} \mathbb{1}_{y_t=n} \right] \\ &\quad + \frac{Z(x)}{Z'(x)} \log \left(\frac{Z(x)}{Z'(x)} \right) E_p \left[\exp \left(\sum_t (\lambda_{an}^{'} - \lambda_{an}) \mathbb{1}_{y_t=a} \mathbb{1}_{y_t=n} \right) \right] \\ &= \frac{Z(x)}{Z'(x)} \sum_{k=1}^{m} p(A_k) e^{k(\lambda_{an}^{'} - \lambda_{an})} k(\lambda_{an}^{'} - \lambda_{an}) \\ &\qquad + \frac{Z(x)}{Z'(x)} \log \left(\frac{Z(x)}{Z'(x)} \right) \left(\sum_{k=1}^{m} p(A_k) e^{k(\lambda_{an}^{'} - \lambda_{an})}) + 1 - p(A) \right) \\ &= c(x) \sum_{k=1}^{m} p(A_k) e^{k(\gamma\lambda_{an})} k(\gamma\lambda_{an}) \\ &\qquad + \log c(x) \end{aligned} \tag{6.32}$$

□

Remark 6.5.5. *The KL Divergence between two distributions is an exact measurement of the distance between them. Thus, when we change only the parameter λ_{an}, it makes sense that it is the only parameter to appear in the final expression for $D_{KL}(q||p)$. Additionally, the effect of this pertubation will be multiplied by the number of times it appears in the word, thus giving us the intuition for the $\sum_{k=1}^{m} p(A_k)k$ term.*

Chapter 7

UQ in Linear Chain CRFs

Now we have all the tools necessary to compute the uncertainty bounds for our unknown model q. Using the DV formulas (5.8) and (5.9) we can actually compute the best upper and lower bounds on our distribution q with respect to g. However, in order to do so we must define some values. Ultimately, we would like these bounds to depend solely upon γ, the percentage change of our specific parameter λ_{an}. Thus, we must define a few quantities:

- We let λ_{an} be 3.25 as from the baseline model trained in Python.
- Since we are only focusing on 4 letter words, $m = 3$ and we compute $p(A_k) = \{0.04, 0, 0\}$ for $k = \{1, 2, 3\}$. These values represent our belief that about 4% of 4 letter words have 1 "an" and none of them have 2 or 3.
- We can also compute $p(B) = 0.01$, meaning that about 1% of 4 letter words begin with "tr".

In Figure 7.1 we plot $D_{\mathrm{KL}}(q||p)$ as a function of γ using (6.31). Using this graph with the upper and lower bound MGF calculations (6.20) and (6.21) respectively, we plot the following uncertainty bounds on $q(B)$ as illustrated in Figure 7.2:

$$q(B) \leq \inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) + \log((1 - p(B)(1 - e^{c}))) \right\} \tag{7.1}$$

and

$$q(B) \geq -\inf_{c>0} \frac{1}{c} \left\{ D_{\mathrm{KL}}(q||p) + \log((1 - p(B)(1 - e^{-c}))) \right\} \tag{7.2}$$

Thus, we have determined the optimal upper and lower bounds. We use Python to calculate the values of c to generate the optimal bounds for each value of $\gamma \in [-1, 1]$.

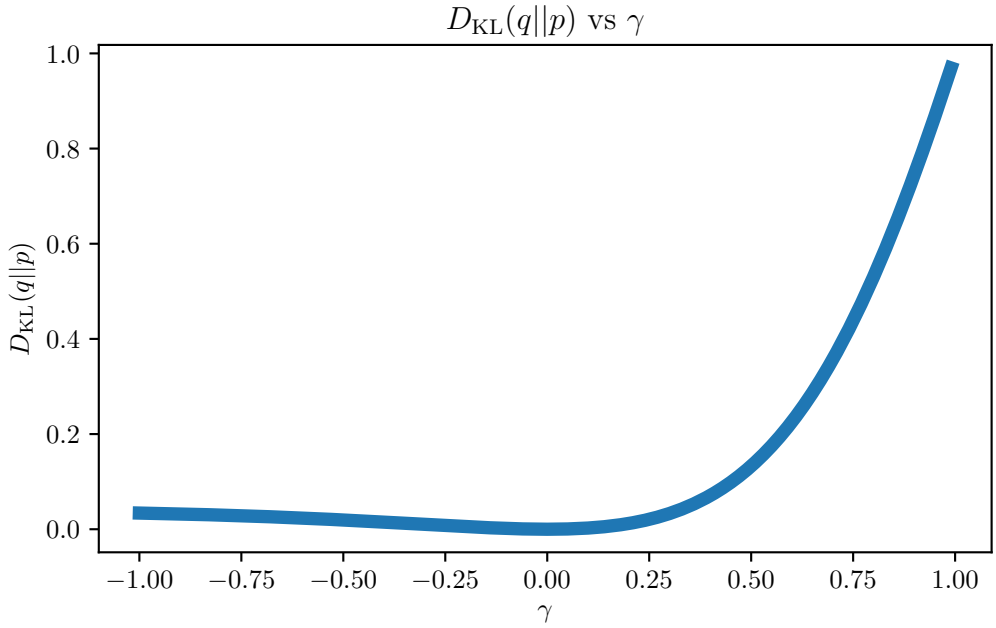


Figure 7.1: Graph of $D_{\mathrm{KL}}(q||p)$ as a function of γ using (6.31). We use the value $\lambda_{an} = 3.25$ as well as $p(A_k) = \{0.04, 0, 0\}$ for $k = \{1, 2, 3\}$.

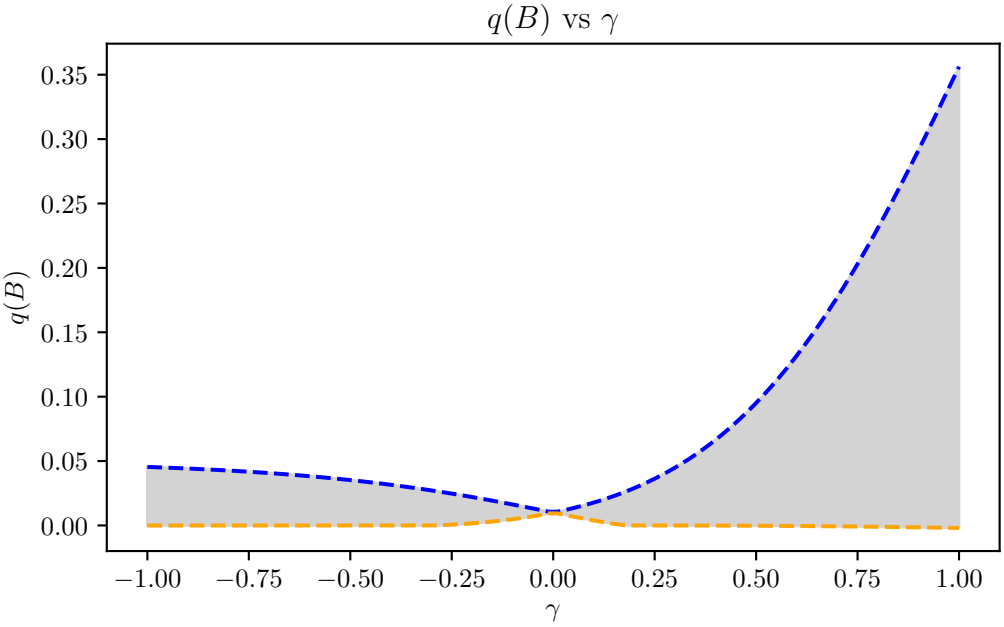


Figure 7.2: Graph of uncertainty bounds on $q(B)$ as a function of γ using equations (7.1) and (7.2). The function g is given in (6.12). We use the value $\lambda_{an} = 3.25$ as well as $p(A_k) = \{0.04, 0, 0\}$ for $k = \{1, 2, 3\}$. For the MGF calculation we use $p(B) = 0.01$

.

7.1 Tightness of UQ bounds (5.8) when the baseline model is in C1

Let us consider a baseline CRF p in C1 and a QoI given by

$$g(y_1, y_2) = \mathbb{1}_{y_1=a}\mathbb{1}_{y_2=n}$$

Then by using (5.16) and Theorem 5.4.1, we get that

$$\sup_{q\in\mathcal{Q}^\eta} / \inf_{q\in\mathcal{Q}^\eta} \{q(B) - p(B)\} = \pm \inf_{c>0} \left\{ \frac{1}{c} \log((1 - p(B)(1 - e^c))) \pm p(B) + \frac{\eta}{c} \right\} \tag{7.3}$$

where $B = \{\text{the words starting with "an"}\}$, and $\mathcal{Q}^\eta$ is the family of distributions satisfying (5.14). Moreover, we have:

$$p^{c_\pm} = \frac{e^{\pm c_\pm g}}{E_p[e^{\pm c_\pm g}]} p = \frac{\exp\left(\sum_{t=2}^4 I_t^\pm(y_{t-1}, y_t, x) + \sum_{t=1}^4 A_t^\pm(y_t, x)\right)}{Z(x)\big(1 - p(B)(1 - e^{\pm c_\pm})\big)} \tag{7.4}$$

and

$$Z^\pm(x) = Z(x)\big(1 - p(B)(1 - e^{\pm c_\pm})\big) \tag{7.5}$$

where $c_\pm$ being the unique solutions of $D_{\mathrm{KL}}(p^{c_\pm} \| p) = \eta$. Note that we now have a new partition function $Z^\pm$ of $p^{c_\pm}$. We also now have:

$$I_t^\pm(y_{t-1}, y_t, x) = \begin{cases} I_2^\pm(y_1, y_2, x) & \text{for } t = 2 \\ I_t(y_{t-1}, y_t, x) & \text{for } t = 3, 4 \end{cases} \tag{7.6}$$

$$A_t^\pm(y_t, x) = A_t(y_t, x) \tag{7.7}$$

with

$$I_2^\pm(y_1, y_2, x) = (\pm c_\pm + \lambda_{an})\mathbb{1}_{y_1=a}\mathbb{1}_{y_2=n} + \sum_{i,j\neq a,n} \lambda_{ij}\mathbb{1}_{y_{t-1}=i}\mathbb{1}_{y_t=j} \tag{7.8}$$

We notice that this model slightly differs from the original baseline model p in C1. In fact, now our parameters are now $\lambda_{i,j,t}$ instead of $\lambda_{i,j}$ - now our parameter value depends on position. This makes sense, as the probability of a certain letter or phrase being at certain positions in a word should be different - the letter "q" is much more likely to be the first letter of a word than the last. Thus, we see that we have in fact defined a new family of distributions within the ambiguity set. However, before exploring this further, we can actually define a QoI that allows us to use our graph from Figure 7.1 to calculate the distributions $p^{c_\pm}$. We define our QoI over all possible positions of the combination "an" as follows:

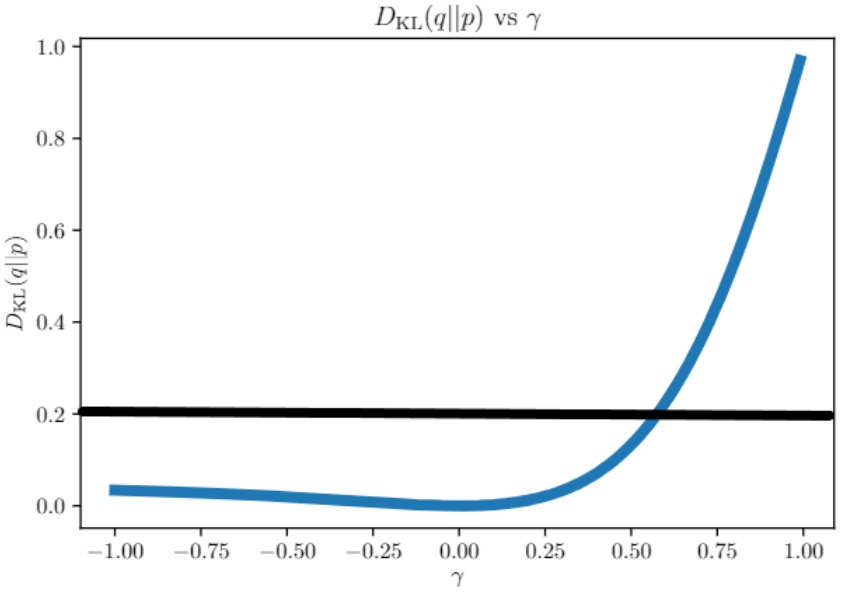
$$\begin{aligned} g(y) &= \sum_{t=2}^4 \mathbb{1}_{y_{t-1}=a}\mathbb{1}_{y_t=n} \\ &= \mathbb{1}_{y_1=a}\mathbb{1}_{y_2=n} + \mathbb{1}_{y_2=a}\mathbb{1}_{y_3=n} + \mathbb{1}_{y_3=a}\mathbb{1}_{y_4=n} \end{aligned} \tag{7.9}$$

This QoI will have the corresponding transition and state functions as follows:

$$I_t^{\pm}(y_{t-1}, y_t, x) = (\pm c_{\pm} + \lambda_{an})\mathbb{1}_{y_{t-1}=a}\mathbb{1}_{y_t=n} + \sum_{i,j\neq a,n} \lambda_{ij}\mathbb{1}_{y_{t-1}=i}\mathbb{1}_{y_t=j} \qquad \text{for } t = 2, 3, 4$$

$$A_t^{\pm}(y_t, x) = A_t(y_t, x)$$

Now we actually have the same parameter λ_{an} for every t, and thus we have reduced back to a model where λ depends only on i, j, not t as well. This allows us to use graph 7.1 of $D_{\mathrm{KL}}(q||p)$ from before. For $\eta = 0.2$:



We see that the intersection corresponds to a γ value of 0.55, so we can further calculate $p^{c_{\pm}}$ using (6.30) and (7.8):

$$\begin{aligned} c_{\pm} + \lambda_{an} &= (1+\gamma)\lambda_{an} \\ \frac{c_{\pm}}{\lambda_{an}} &= \gamma \\ c_+ = c_- &= \lambda_{an}\gamma \\ c_+ = c_- &= (3.25)(0.55) = 1.79 \end{aligned} \tag{7.10}$$

Now that we have gone through a sample calculation, we can return to the interesting fact that our worst case scenario has actually brought us to a different parametric family. As mentioned earlier, we now have transition parameters $\lambda_{i,j,t}$ instead of $\lambda_{i,j}$. However, as we saw in the sample calculation, there are cases where the former reduces to the latter. Specifically, this will happen when $\lambda_{i,j,t}$ is the same for all values of t, thus eliminating the dependence of the parameter on position. These parametric families are represented visually in Figure 7.3.

As we saw earlier, we now are familiar with two parametric families within $\mathcal{Q}^{\eta}$ - first, CRFs with the transition parameters depending only on i, j, and second, CRFs with the

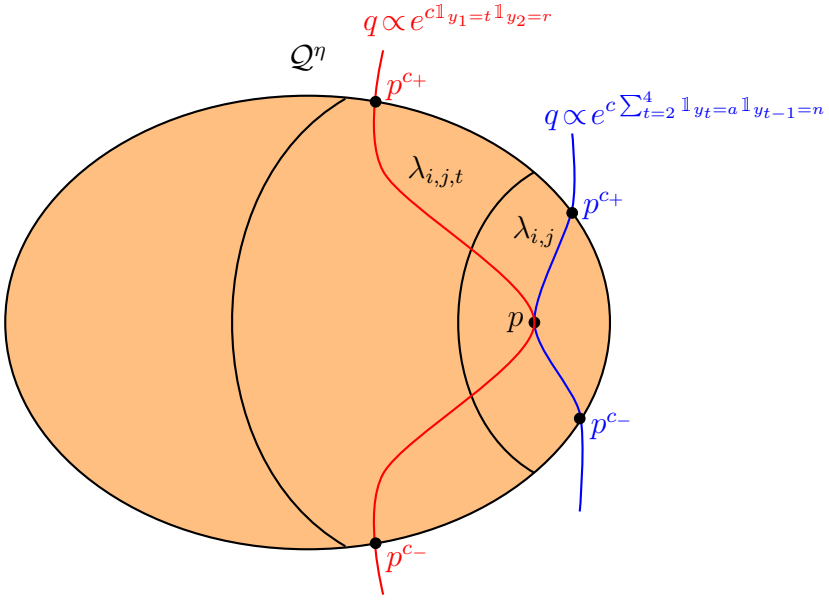


Figure 7.3: Visual representation of the ambiguity set $\mathcal{Q}^\eta$ with respect to the baseline model p and how the QoI affects the predictive uncertainty of worst case scenarios. The blue color represents the parametric family $q \propto e^{cg}$ corresponding to the QoI g given in (7.9) along with its worst case scenarios p^{c+} and p^{c-}. The red color represents the parametric family corresponding to the QoI g given in (6.12) along with its worst case scenarios as well. The choice of QoI determines the parametric family our worst case scenarios $p^{c\pm}$ lie on.

transition parameters depending on i, j, and t. A parametric family is a family of models with the same structure, the same parameters. Of course, these are only some of the families contained within the entire ambiguity set, but what is special about them is that the $\lambda_{i,j}$ family is actually a subset of the $\lambda_{i,j,t}$ family. We already saw this before, when we defined our QoI in such a way to stay within the $\lambda_{i,j}$ family. In fact, it is exactly our decision on QoI that determines where we will be within this entire ambiguity set.

Depending on which QoI we choose, our parametric family will be different. This causes our worst case scenarios $p^{c\pm}$ to be at different points on the boundary. Because of the specific QoI we choose in (7.9), the p^{c+} and p^{c-} in Figure 7.3 are on the border of the $\lambda_{i,j}$ family.

7.2 A baseline model assuming the Markov Property

In this section, we consider a special baseline model p assuming the Markov Property (MP).

Let $\eta_t > 0$ be a misspecification parameter standing for the size of the ambiguity set. For each t, we define the class of CRFs given below:

$$\mathcal{Q}^{\eta_t} = \Big\{\text{all CRFs q with MP} : D_{\mathrm{KL}}(q(y_t|y_{t-1},x)\|p(y_t|y_{t-1},x)) \le \eta_t \text{ for all } y_{t-1},$$
$$q(y_j|y_{j-1},x) \equiv p(y_j|y_{j-1},x) \text{ for all } j \ne t\Big\} \tag{7.11}$$

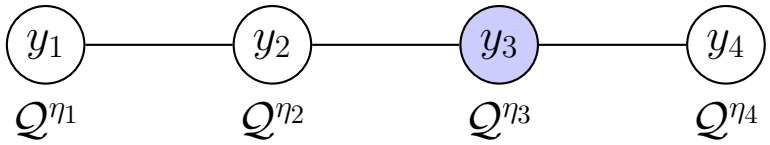


Figure 7.4: Ambiguity sets $\mathcal{Q}^{\eta_t}$ on a 4 letter word with respect to the QoI given in (7.12). As seen in (7.13), once we have determined our QoI, we only worry about the ambiguity sets before and up to it, as the ones after all have 0 predictive uncertainty.

Let

$$g(y_3) = \mathbb{1}_{y_3=r}(y_3) \tag{7.12}$$

be a QoI. Once our QoI has been determined, our word looks like the one in Figure 7.4.

We compute the difference between the expectations with respect to p and an alternative model in (7.11) as follows:

$$\begin{aligned} E_q[g] - E_p[g] &= \sum_y g(y_3) \prod_{t=1}^{4} q(y_t|y_{t-1}, x) - \sum_y g(y_3) \prod_{t=1}^{4} p(y_t|y_{t-1}, x) \\ &= \sum_{y_1,y_2,y_3} g(y_3) \prod_{t=1}^{3} q(y_t|y_{t-1}, x) - \sum_y g(y_3) \prod_{t=1}^{3} p(y_t|y_{t-1}, x) \end{aligned} \tag{7.13}$$

- If $q \in \mathcal{Q}^{\eta_4}$, then $E_q[g] - E_p[g] = 0$.
- If $q \in \mathcal{Q}^{\eta_t}$ with $t \leq 3$, then (7.13) can be further computed. For simplicity we assume $t = 2$, but the following calculation is the same for any $t \leq 3$.

$$\begin{aligned} E_q[g] - E_p[g] &= \sum_{y_1,y_2,y_3} g(y_3) \prod_{t=1}^{3} q(y_t|y_{t-1}, x) - \sum_y g(y_3) \prod_{t=1}^{3} p(y_t|y_{t-1}, x) \\ &= \sum_{y_1,y_2,y_3} g(y_3) p(y_3|y_2, x) q(y_2|y_1, x) p(y_1|x) \\ &\qquad - \sum_{y_1,y_2,y_3} g(y_3) p(y_3|y_2, x) p(y_2|y_1, x) p(y_1|x) \\ &= E_{p_1}[E_{q_{2|1}}[p(r|y_2)]] - E_{p_1}[E_{p_{2|1}}[p(r|y_2)]] \\ &= E_{p_1}\left[E_{q_{2|1}}[p(r|y_2)] - E_{p_{2|1}}[p(r|y_2)]\right] \end{aligned} \tag{7.14}$$

where $p_1 = p(y_1|x)$, $p_{2|1} = p(y_2|y_1, x)$ and $q_{2|1} = q(y_2|y_1, x)$. We apply (5.16) to $E_{q_{2|1}}[p(r|y_2)] - E_{p_{2|1}}[p(r|y_2)]$ over a new ambiguity set that focuses only on alternative conditional probabilities $q_{2|1}$, i.e.

$$\mathcal{S}^{\eta_2} = \left\{\text{all } q(y_2|y_1, x) : D_{\mathrm{KL}}(q(y_2|y_1, x) \| p(y_2|y_1, x)) \leq \eta_2, \text{ for all } y_1\right\} \tag{7.15}$$

we compute

$$\begin{aligned} I^{\pm}(g, p_{2|1}; \mathcal{S}^{\eta_2}) &:= \sup_{q \in \mathcal{S}^{\eta_2}} / \inf \{E_{q_{2|1}}[p(r|y_2)] - E_{p_{2|1}}[p(r|y_2)\} \\ &\leq \pm \inf_{c>0} \Big[\frac{1}{c} \log(E_{p_{2|1}}[e^{g-E_p[g]}]) + \frac{\eta}{c}\Big] \end{aligned} \tag{7.16}$$

We use the above bound for the upper bound $I^+(g, p; \mathcal{S}^{\eta_2})$ and we conclude that

$$I^+(g, p; \mathcal{Q}^{\eta_2}) \leq E_{p_1}\left[\inf_{c>0} \Big[\frac{1}{c} \log(E_{p_{2|1}}[e^{g-E_p[g]}]) + \frac{\eta}{c}\Big]\right]$$

For any given y_1, we use (5.17) and we get the optimal probability distributions

$$p^{c_+}(y_2|y_1, x) = \frac{e^{c_+(y_1)p(r|y_2)}}{E_p[e^{c_+(y_1)p(r|y_2)}]} p(y_2|y_1, x) \tag{7.17}$$

where $c_+(y_1)$ is a function of y_1 determined by $D_{\mathrm{KL}}(q(y_2|y_1, x)\|p(y_2|y_1, x)) = \eta_2$. The conditional densities $p^{c_+}(y_2|y_1, x)$ are in $\mathcal{S}^{\eta_2}$. Therefore

$$p^{c_+}(y|x) = p(y_1|x)p^{c_+}(y_2|y_1, x)p(y_3|y_2)p(y_4|y_3) \tag{7.18}$$

The lower bound of the predictive uncertainty can be calculated similarly. This analysis can be repeated for each character and compute its predictive uncertainty. Then we rank them from the least to most dangerous. This allows us to identify which characters on which our model is the most volatile when predicting. This information can be extremely beneficial when deciding how to improve a model with respect to a certain quantity of interest.

Chapter 8

Python Implementation of CRF

8.1 Overview

In this section we outline an implementation of Optical Character Recognition (OCR) using a Conditional Random Field (CRF) in the Python programming language written by Debora Sujono. A step by step process for training and then testing a model in Python is given below. We will treat this as the baseline model to which models are compared using the variational principles and methods described earlier.

8.2 Image Processing

Before handwritten images are able to be fitted in the model, the images need to be processed in order to separate into the individual letters. In doing so, there are many options available, but we do not focus heavily on them in this paper. The process proceeds generally as follows:

1. Preprocessing the image. This can include:
   (a) Correcting rotated or skewed text
   (b) Binarizing the image into black and white to eliminate noise
2. Line segmentation. We locate where the lines of text are and mark them as areas of interest.
3. Word segmentation. We split each line of text into individual words to be analyzed.
4. Character segmentation. Finally, we go through each word and break it into individual characters to be used by our CRF model.

Some methods for line and word segmentation use other models such as Neural Networks or Gaussian models [1].

8.3 Notation

Let k denote the size of the alphabet, and the function $l(\cdot) = w$ that takes a word and returns its length. In our case, we will be focusing on the 10 most common letters in the alphabet: $\{e, t, a, i, n, o, s, h, r, d\}$. Thus, k will always be 10 for our purposes. We denote X_i as the given character image, and Y_i as the character label given that image. Thus, our CRF will model $P(Y_i|X_i)$, the probability that the given image represents a certain letter Y_i given the original image X_i.

8.4 Data

The data used to train these CRF models is important in understanding how they work and in evaluating their performance. Words are split into individual letters, and each letter is represented by 16×20 binarized pixel array of 0's and 1, $\{0, 1\}^{16\times20}$. We call the i'th original image X_i and the i'th predicted letter Y_i. Both the training and test data are given in the forms of arrays, with the first entry being the original word and the subsequent entries each representing a handwritten letter in the word. This data can be found in the "data" folder of the same repository [12].

8.5 Coding the CRF

Here we explain the implementation of the CRF in Python. Many of the more technical details are omitted, but can be found by visiting the original Github repository linked in Appendix. The parameters included in this specific CRF are state and transition parameters. These represent the weights corresponding to relationships between pixels of the original image and the prediction, X_i and Y_i, and neighboring letters, Y_i and Y_{i-1} respectively. These are the specific parameters that define the model. In other words, once these weights are decided, the model has already been fitted. The parameters are found using MLE, but functionally are found by minimizing the negative log-likelihood function using the Limited-memory BFGS (L-BFGS-B) algorithm.

8.5.1 State Parameters

The state parameters define the relationship between pixels in the original image and pixels in the predicted letter. Since each letter is represented as a 16×20 binary array, for an alphabet of k letters we will have k arrays of 16×20 individual values. For any given letter vector, the parameter array corresponds to the likelihood of there being a 1 at any position in the vector. For example, the letter "T" is more likely to have a 1 near the center rather than on the sides. These parameters are stored in a k length vector of 16×20 arrays.

8.5.2 Transition Parameters

The transition parameters define relationships between neighboring letters. These parameters dictate the likelihood that any given letter is next to another one. For example, after

the letter "q", almost always the letter "u" will follow. This means that the parameter corresponding to this pairing will be high. It is unlikely that another consonant will follow, so the pairing with a letter such as "t" will have a much lower parameter. We require a parameter for every pair of letters in our alphabet, so we will have a k length vector of k length arrays.

8.6 Training

The general algorithm for training a CRF model to perform OCR is actually quite simple. First, one must decide what parameters to include in the model. In the specific Python package included in this paper [12], there are two sets of main parameters: State and transition parameters. These are the weights included in the model used to calculate the marginal probabilities. Once these parameters are decided, the form of the model is determined. The values of the parameters are found by running some minimization algorithm on the negative log-likelihood function given some set of training data. This log-likelihood function is given by the standard Gibbs Measure used in a CRF. As mentioned earlier, these parameters are found by MLE.

8.7 Testing

Once the model has been trained, we can begin evaluating performance. The given code simply calculates a raw percentage by:

$$\frac{\text{Number of correct predictions}}{\text{Number of total predictions}}$$

This metric is straightforward, but can be useful for getting an idea of how good the model is. However, if we are interested in evaluating the model with respect to more specific quantities, we must apply our variational principles in order to get a more in depth analysis.

8.8 Example

A sample model provided with the code (that is, a set of transition and state parameters) allows us to perform analysis immediately. This sample model can be found at the linked Github repository [12] as well.

Bibliography

[1] Manivannan Arivazhagan, Harish Srinivasan, and Sargur Srihari. "A statistical approach to line segmentation in handwritten documents". In: *Document recognition and retrieval XIV*. Vol. 6500. International Society for Optics and Photonics. 2007, 65000T.

[2] Panagiota Birmpa et al.

[3] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[4] Kamaljit Chowdhary and Paul Dupuis. "Distinguishing and integrating aleatoric and epistemic variation in uncertainty quantification". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 47.03 (2013), pp. 635–662.

[5] Paul Dupuis et al. "Path-space information bounds for uncertainty quantification and sensitivity analysis of stochastic dynamics". In: *SIAM/ASA Journal on Uncertainty Quantification* 4.1 (2016), pp. 80–111.

[6] Jinchao Feng et al. "Explainable and trustworthy artificial intelligence for correctable modeling in chemical sciences". In: *Science advances* 6.42 (2020), eabc3204.

[7] Konstantinos Gourgoulias et al. "How biased is your model? Concentration Inequalities, Information and Model Bias". In: *IEEE Transactions on Information Theory (accepted pending revisions)* (2019), arXiv:1706.10260. URL: `http://arxiv.org/abs/1706.10260`.

[8] Markos A Katsoulakis, Luc Rey-Bellet, and Jie Wang. "Scalable information inequalities for uncertainty quantification". In: *Journal of Computational Physics* 336 (2017), pp. 513–545.

[9] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[10] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[11] Shravya Shetty, Harish Srinivasan, and Sargur Srihari. "Handwritten word recognition using conditional random fields". In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 2. IEEE. 2007, pp. 1098–1102.

[12] Debora Sujono. *Python implementation of Conditional Random Fields (CRF) for Optical Character Recognition (OCR)*. `https://github.com/deborausujono/crfocr`. 2015.

[13] Timothy John Sullivan. *Introduction to uncertainty quantification*. Vol. 63. Springer, 2015.

[14] Charles Sutton and Andrew McCallum. “An introduction to conditional random fields for relational learning”. In: *Introduction to statistical relational learning* 2 (2006), pp. 93–128.

Appendix A

Python Code

Here we include a slightly modified version of Debora Sujono's Python implementation of OCR using a CRF. The original code can be found at the following Github repository: The modified code contains 5 primary

A.1 CRF Implementation

```
import numpy as np
import scipy.special

def node_potentials(features, state_params):
    """
    Returns a w x k numpy array of node potentials,
    where w is the word length and k is the size of the alphabet.

    Parameters:
    - features, a w x n numpy array of feature vectors,
      where n is the length of the feature vector; and
    - state_params, a k x n numpy array of state parameters.
    """

    return np.dot(features, np.transpose(state_params))

def clique_potentials(node_factor1, node_factor2, trans_params):
    """
    Computes the clique potentials of a single clique.
    Returns a k x k numpy array, where k is the size of the alphabet.

    Parameters:
    - node_factor1, a k-dimensional numpy array of node potentials;
    - node_factor2, a k-dimensional numpy array of node potentials or a None
    - trans_params, a k x k numpy array of transition parameters.
```

```
    """

    psi = trans_params + node_factor1[:, np.newaxis]
    if node_factor2 is not None:
        psi += node_factor2

    return psi

def clique_tree_potentials(theta, features):
    """
    Computes the clique potentials of the entire chain.
    Returns a (w-1) x k x k numpy array,
    where w is the word length and k is the size of the alphabet.

    Parameters:
    - theta, a [state_params, trans_params] list; and
    - features, a w x n numpy array, where n is the length of the feature vec
    """

    state_params, trans_params = theta
    phi = node_potentials(features, state_params)

    # Include the potentials of the last two nodes in the same clique
    cliques = [(node, None) for node in phi[:-2]] + [(phi[-2], phi[-1])]

    psi = [clique_potentials(n1, n2, trans_params) for n1, n2 in cliques]

    return np.array(psi)

def sum_product_messages(psi):
    """
    Returns the (backward messages, forward messages) tuple.
    Each messages is a (w-2) x k numpy array,
    where w is the word length and k is the size of the alphabet.

    Parameter:
    - psi, a (w-1) x k x k numpy array of clique tree potentials.
    """

    # Backward messages
    bwd = []
    prev_msgs = np.zeros(psi.shape[1])
    for clique in psi[:0:-1]:
        msg = scipy.special.logsumexp(clique + prev_msgs, axis=1)
        bwd.append(msg)
```

```
        prev_msgs += msg

    # Forward messages
    fwd = []
    prev_msgs = np.zeros(psi.shape[1])
    for clique in psi[:-1]:
        msg = scipy.special.logsumexp(clique + prev_msgs[:, np.newaxis], axis
        fwd.append(msg)
        prev_msgs += msg

    return (np.array(bwd), np.array(fwd))

def beliefs(theta, features):
    """
    Returns a numpy array of size (w-1) x k x k,
    where w is the word length and k is the size of the alphabet.

    Parameters:
    - theta, a [state_params, trans_params] list; and
    - features, a w x n numpy array of feature vectors,
      where n is the length of the feature vector.
    """

    psi = clique_tree_potentials(theta, features)
    delta_bwd, delta_fwd = sum_product_messages(psi)

    k = delta_fwd.shape[1]
    delta_fwd = np.concatenate(([np.zeros(k)], delta_fwd))
    delta_bwd = np.concatenate((delta_bwd[::-1], [np.zeros(k)]))
    beta = psi + delta_fwd[:, :, np.newaxis] + delta_bwd[:, np.newaxis]

    return np.array(beta)

def pairwise_prob(beta):
    """
    Computes the pairwise marginal probabilities.
    Returns a numpy array of size (w-1) x k x k,
    where w is the word length and k is the size of the alphabet.

    Parameter:
    - beta, a (w-1) x k x k numpy array of log belief tables.
    """

    return np.exp(beta - scipy.special.logsumexp(beta, axis=(1,2))[:, np.newa
```

```
def single_prob(pairwise_p):
    """
    Computes the singleton marginal probabilities.
    Returns a w x k numpy array, where n is the word length and k is the size

    Parameter:
    - pairwise_p, a numpy array of size (w-1) x k x k of pairwise marginal pr
    """

    p = np.sum(pairwise_p, axis=2)
    q = np.sum(pairwise_p[-1], axis=0) # Last character in the word

    return np.concatenate((p, q[np.newaxis, :]))

def joint_prob(single_p, label, alphabet):
    """
    Computes the joint probability of the label given singleton marginal pro
    Returns a scalar.

    Parameters:
    - single_p, a w x k numpy array of singleton marginal probabilities,
      where n is the word length and k is the size of the alphabet;
    - label, a list of character labels; and
    - alphabet, a list of all possible character labels.
    """

    p = [np.log(marginal[alphabet.index(c)]) for (c, marginal) in zip(label,

    return np.sum(p)

def likelihood(theta, data, alphabet):
    """
    Objective function to minimize.
    Returns the negative average log likelihood of theta given the data.

    Parameters:
    - theta, a [state_params, trans_params] list;
    - data, a list of (word_label, word_features) tuples; and
    - alphabet, a list of all possible character labels.
    """

    # If flattened, reshape theta into a list of state parameter table of siz
    # and transition parameter table of size k x k,
    # where n is the length of the feature vector and k is the size of the al
    if len(theta) != 2:
```

```
        k = len(alphabet)       # number of possible character labels
        n = len(data[0][1][0]) # length of feature vector
        mid = k * n

        state_params = np.reshape(theta[:mid], (k, n))
        trans_params = np.reshape(theta[mid:], (k, k))
        theta = [state_params, trans_params]

    p = []
    for label, features in data:
        beta = beliefs(theta, features)
        pairwise_p = pairwise_prob(beta)
        single_p = single_prob(pairwise_p)
        p.append(joint_prob(single_p, label, alphabet))

    return -np.sum(p)/len(data)

def state_gradient(theta, data, alphabet):
    """
    Returns a flattened k x n numpy array,
    where k is the size of the alphabet and n is the length of the feature ve

    Parameters:
    - theta, a [state_params, trans_params] list;
    - data, a list of (word_label, word_features) tuples; and
    - alphabet, a list of all possible character labels.
    """

    # Initialize a state gradient table of size k x n with zeros
    gradient = np.zeros((len(alphabet), len(data[0][1][0])))

    for label, features in data:
        beta = beliefs(theta, features)
        pairwise_p = pairwise_prob(beta)
        single_p = single_prob(pairwise_p)
        for v, c, p in zip(features, label, single_p):
            for i in range(gradient.shape[0]): # possible labels
                for j in range(gradient.shape[1]): # features
                    indicator = 0
                    if c == alphabet[i]:
                        indicator = 1
                    gradient[i][j] += (indicator - p[i]) * v[j]

    gradient /= len(data)
```

```
    return np.ndarray.flatten(np.negative(gradient))

def transition_gradient(theta, data, alphabet):
    """
    Returns a flattened k x k numpy array, where k is the size of the alphabe

    Parameters:
    - theta, a [state_params, trans_params] list;
    - data, a list of (word_label, word_features) tuples; and
    - alphabet, a list of all possible character labels.
    """

    # Initialize a transition gradient table of size k x k with zeros
    gradient = np.zeros((len(alphabet), len(alphabet)))

    for label, features in data:
        beta = beliefs(theta, features)
        pairwise_p = pairwise_prob(beta)
        label_pairs = zip([None] + label, label + [None])[1:-1]

        for (label1, label2), p in zip(label_pairs, pairwise_p):
            for i in range(gradient.shape[0]):
                for j in range(gradient.shape[1]):
                    indicator = 0
                    if label1 == alphabet[i] and label2 == alphabet[j]:
                        indicator = 1
                    gradient[i][j] += indicator - p[i][j]

    gradient /= len(data)

    return np.ndarray.flatten(np.negative(gradient))

def likelihood_prime(theta, data, alphabet):
    """
    Returns a flattened numpy array of the [feature_gradient, transition_grad

    Parameters:
    - theta, a [state_params, trans_params] list;
    - data, a list of (word_label, word_features) tuples; and
    - alphabet, a list of all possible character labels.
    """

    # Reshape flattened theta into a list of k x n state parameter table and
    # k x k transition parameter table, where k is the size of the alphabet a
    # n is the length of the feature vector. Both parameter tables are numpy
```

```
    k = len(alphabet)       # number of possible character labels
    n = len(data[0][1][0]) # length of feature vector
    mid = k * n

    state_params = np.reshape(theta[:mid], (k, n))
    trans_params = np.reshape(theta[mid:], (k, k))
    theta = [state_params, trans_params]

    return np.concatenate((state_gradient(theta, data, alphabet),
                           transition_gradient(theta, data, alphabet)))

def predict_word(single_p, alphabet):
    """
    Returns a list of predicted characters of a word.

    Parameters:
    - single_p, a w x k numpy array of singleton marginal probabilities,
      where w is the word length and k is the size of the alphabet; and
    - alphabet, a list of all possible character labels.
    """

    indices = np.argmax(single_p, axis=1)

    return [alphabet[i] for i in indices]

def predict(theta, data, alphabet):
    """
    Returns a list of predictions, where each prediction is
    a list of predicted character labels of a word.

    Parameters:
    - theta, a [state_params, trans_params] list;
    - data, a list of (word_label, word_features) tuples; and
    - alphabet, a list of all possible character labels.
    """

    predictions = []
    for _, features in data:
        beta = beliefs(theta, features)
        pairwise_p = pairwise_prob(beta)
        single_p = single_prob(pairwise_p)
        predictions.append(predict_word(single_p, alphabet))

    return predictions
```

A.2 Training the Parameters

```
import argparse
import os
import time

import numpy as np

from crf import likelihood, likelihood_prime
from scipy.optimize import fmin_l_bfgs_b
from util import read_data, print_model

def train(data, alphabet, maxiter, log):
        """
        Returns the learned [state_params, trans_params] list,
        where each parameter table is a numpy array.
        """

        # Initialize state and transition parameter tables with zeros
        state_params = np.ndarray.flatten(np.zeros((len(alphabet), len(data[0
        trans_params = np.ndarray.flatten(np.zeros((len(alphabet), len(alphab
        theta = np.concatenate([state_params, trans_params])

        # Learn by minimizing the negative average log likelihood
        t0 = time.time()
        theta, fmin, _ = fmin_l_bfgs_b(likelihood, theta, fprime=likelihood_pr
                                       args=(data, alphabet), maxiter=max
        t1 = time.time()

        # Write training summary to log
        if log > 0:
                print ("Training data size:", len(data))
                print ("Value of likelihood function at minimum:", np.exp(-fm
                print ("Training time:", t1-t0)

        k = len(alphabet)
        n = len(data[0][1][0])
        mid = k * n
        state_params = np.reshape(theta[:mid], (k, n))
        trans_params = np.reshape(theta[mid:], (k, k))

        return [state_params, trans_params]

def main(train_pattern, model_dir, alphabet, maxiter, log):
```

```
        alphabet = list(alphabet)

        # Read training data
        data = read_data(train_pattern)
        if log > 0:
                print ('Successfully read', len(data), 'data cases')

        # Train the model
        model = train(data, alphabet, maxiter, log)

        # Save the model
        if log > 0:
                print ('Saving model to', model_dir)
        if not os.path.exists(model_dir):
                os.makedirs(model_dir)
        state_file = model_dir + "/state-params.txt"
        trans_file = model_dir + "/transition-params.txt"
        print_model(model, state_file, trans_file)

if __name__ == '__main__':
        parser = argparse.ArgumentParser()
        parser.add_argument('-train', help='Regex pattern of training files',
        parser.add_argument('-model', help='Directory of model files', dest='
        parser.add_argument('-alphabet', help='String of all possible charact
        parser.add_argument('-maxiter', help='Maximum iteration for L-BFGS op
        parser.add_argument('-log', help='Print log to stdout if 1', dest='lo
        args = parser.parse_args()

        main(args.train, args.model, args.alphabet, args.maxiter, args.log)
```

A.3 Testing the Model

```
import argparse

from crf import predict
from util import read_data, read_model, score

def test(theta, data, alphabet, print_tags, print_score):
        predictions = predict(theta, data, alphabet)

        if print_tags:
                for word in predictions:
                        print (''.join(word))

        if print_score:
```

```
                print (score(predictions, data))

def main(test_pattern, model_dir, alphabet, print_tags, print_score):
        alphabet = list(alphabet)

        # Open feature and transition params
        theta = read_model(model_dir)

        # Read test data
        data = read_data(test_pattern)

        test(theta, data, alphabet, print_tags, print_score)

if __name__ == '__main__':
        parser = argparse.ArgumentParser()
        parser.add_argument('-test', help='Regex pattern of test files', dest
        parser.add_argument('-model', help='Directory of model files', dest='
        parser.add_argument('-alphabet', help='String of all possible charact
        parser.add_argument('-tag', help='Print predicted labels to stdout if
        parser.add_argument('-score', help='Calculate and print prediction ac
        args = parser.parse_args()

        main(args.test, args.model, args.alphabet, args.tag, args.score)
```

A.4 Evaluation Method

```
import glob

def read_data(pattern):
    data = []
    files = glob.glob(pattern)
    for name in files:
        f = open(name, 'r')
        label = list(f.readline().strip())
        features = [[int(b) for b in c.split()] for c in f]
        data.append((label, features))

    return data

def print_model(theta, state_file, trans_file):
    files = [state_file, trans_file]
    for (params, name) in zip(theta, files):
        f = open(name, 'w')
        for row in params:
            for cell in row:
```

```
                f.write(str(cell)+" ")
            f.write("\n")
        f.close()

def read_model(model_dir):
    theta = []
    files = ['state-params.txt', 'transition-params.txt']
    for name in files:
        f = open(model_dir + '/' + name, 'r')
        params = [[float(d) for d in line.split()] for line in f]
        theta.append(params)

    return theta

def score(predictions, data):
    true_count = sum([sum([c1 == c2 for c1, c2 in zip(prediction, label)])
                      for prediction, (label, _) in zip(predictions, data)])
    total_count = sum([len(prediction) for prediction in predictions])
    accuracy = 1.0 * true_count / total_count

    return accuracy
```