

1 Structured Labels

In many applications one is interested in labeling an input with a complex label. For example, in speech recognition one is interested in labeling a sound waveform (coming from a microphone) with a sequence of words. In stereo depth perception one is interested in taking a pair of images and using triangulation to determine the depth of the objects in the picture. In this case one takes as input a pair of images and produces as output a labeling of, say, the left image with a depth at each pixel. In parsing one takes as input a string of words and produces as output a parse tree. In all of these cases the output is a complex or “structured” label. The label itself contains many bits of information.

Let \mathcal{X} be an input space and let \mathcal{Y} be a label space. In “multiclass” labeling we typically have $|\mathcal{Y}|$ is on the order of ten or perhaps one hundred. In structured labeling $|\mathcal{Y}|$ is much larger. In stereo vision, for example, we have that \mathcal{Y} is the set of all labelings of pixels with distances. In a one megapixel image, using 4 bits to represent each distance, it takes four million bits to represent a single label. In that case we have that $|\mathcal{Y}|$ is $2^{4,000,000}$. The equations that we write in these notes apply equally well when $|\mathcal{Y}|$ is 10 as when $|\mathcal{Y}|$ is $2^{4,000,000}$. Of course the structured case where \mathcal{Y} is enormous presents computational challenges. In general we assume that we have training data consisting of labeled inputs.

$$\begin{aligned} S &= \langle x_1, y_1 \rangle, \dots, \langle x_T, y_T \rangle \\ x_t &\in \mathcal{X} \\ y_t &\in \mathcal{Y} \end{aligned}$$

Our objective is to use the training data to construct a predictor $f(x)$ which predicts y from x . We assume a feature map Ψ on $\mathcal{X} \times \mathcal{Y}$ so that for $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ we have $\Psi(x, y) \in R^d$ where d is the number of features. For $1 \leq i \leq d$ we let $\Psi_i(x)$ be the i th coordinate value of $\Psi(x)$. Here we will be interested in predictors of the following form where $\beta \in R^d$ is a parameter vector to be learned from the training data.

$$f_\beta(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \beta \cdot \Psi(x, y) \tag{1}$$

Our problem now is learn β from the training data. Unlike binary classification, in structured labeling there is no natural unique notion “margin” and we will not write any single general learning equation. Instead we will consider various different approaches.

2 Generative Learning

Generative learning can be defined by the following equations where $P_\beta(x, y)$ is defined by a directed model such as a Bayesian network, a Hidden Markov

model (HMM) or a probabilistic context free grammar (PCFG).

$$\begin{aligned} \beta^* &= \operatorname{argmin}_{\beta} \sum_{t=1}^T \ln \frac{1}{P_{\beta}(x_t, y_t)} \\ &= \operatorname{argmax}_{\beta} \prod_{t=1}^T P_{\beta}(x_t, y_t) \end{aligned} \tag{2}$$

When $P_{\beta}(x, y)$ is defined by a directed model, the problem of computing β^* is usually trivial — each component of β can be set to be a ratio of counts of events in the training data.

3 Hidden Markov Models (HMMs)

The Viterbi algorithm for HMMs can be formulated as a particular instance of equation (1). The Viterbi algorithm takes a sequence of observed states and outputs the most likely sequence of hidden states given the observed states. Suppose that there is a finite set of hidden states and a finite set of observations. Each of x_t and y_t are sequences.

$$x_t = x_t^1, \dots, x_t^{N_t}$$

$$y_t = y_t^1, \dots, y_t^{N_t}$$

For a hidden state s let β_s be the natural logarithm of probability of starting in state s . For two hidden states s and w let $\beta_{s,w}$ be the natural logarithm of the probability that the next hidden state is w given that the current hidden state is s . For a hidden state s , and an observation u , let $\beta_{s,u}$ be the natural logarithm of the probability of emitting the observation u from hidden state s . We can now write $P_{\beta}(x, y)$ as follows where N is the length of both sequences x and y .

$$P_{\beta}(x, y) = \exp \left(\beta_{y^1} + \sum_{i=1}^{N-1} \beta_{y^i, y^{i+1}} + \sum_{i=1}^N \beta_{y^i, x^i} \right) \tag{3}$$

We can rewrite (3) as $\beta \cdot \Psi(x, y)$ using an appropriate definition of the feature vector $\Psi(x, y)$. More specifically, we assume a feature corresponding to each of the components of β .

$$\Psi_s(x, y) = \begin{cases} 1 & \text{if } y^1 = s \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$\Psi_{s,w}(x, y) = |\{i : 1 \leq i \leq N - 1, y^i = s, y^{i+1} = w\}| \tag{5}$$

$$\Psi_{s,u}(x, y) = |\{i : 1 \leq i \leq N, y^i = s, x^i = u\}| \tag{6}$$

Note that the features $\Psi_{s,w}(x, y)$ and $\Psi_{s,u}(x, y)$ are counts — the feature value is the number of times a certain event occurs in the pair $\langle x, y \rangle$. For example

$\Psi_{s,w}(x, y)$ is the number of times y contains a transition from state s to state w . Under this definition of $\Psi(x, y)$ we now have the following.

$$P_\beta(x, y) = \exp(\beta \cdot \Psi(x, y)) \quad (7)$$

So we have that (1) computes the most likely y given x which is the sequence computed by the Viterbi algorithm.

If we use generative training we set β^* to be the solution to (2) which gives the following.

$$\beta_s^* = \ln \left(\frac{\sum_{t=1}^T I[y_t^1 = s]}{T} \right) \quad (8)$$

$$\beta_{s,w}^* = \ln \left(\frac{\sum_{t=1}^T \sum_{i=1}^{N_t-1} I[y_t^i = s \wedge y_t^{i+1} = w]}{\sum_{t=1}^T \sum_{i=1}^{N_t-1} I[y_t^i = s]} \right) \quad (9)$$

$$\beta_{s,u}^* = \ln \left(\frac{\sum_{t=1}^T \sum_{i=1}^{N_t} I[y_t^i = s \wedge x_t^{i+1} = u]}{\sum_{t=1}^T \sum_{i=1}^{N_t} I[y_t^i = s]} \right) \quad (10)$$

Each component of β^* is set to the log of a count ratio. Count ratios are also used in generative training of Bayesian networks and probabilistic context free grammars.

4 Log Loss and Conditional Random Fields (CRFs)

Multiclass or structured logistic regression is defined by the following equations.

$$\beta^* = \operatorname{argmin}_\beta \left(\sum_{t=1}^T \ln \frac{1}{P_\beta(y|x)} \right) + \frac{1}{2} \lambda \|\beta\|^2 \quad (11)$$

$$P_\beta(y|x) = \frac{1}{Z_\beta(x)} \exp(\beta \cdot \Psi(x, y)) \quad (12)$$

$$Z_\beta(x) = \sum_{y \in \mathcal{Y}} \exp(\beta \cdot \Psi(x, y)) \quad (13)$$

It is interesting to consider the case of an HMM. We can use the same components of β and the same feature vector $\Psi(x, y)$ as is defined in section 3. However, equation (11) gives a different value in general from (2). In particular, under (11) we do not get equations (8), (9), and (10). The optimization specified by (11) is convex in β (see the homework excercises) and is typically solved by gradient descent. We can compute the gradient of the right hand side of (11) as follows.

$$\begin{aligned}
L &= \left(\sum_{t=1}^T \ln \frac{1}{P_\beta(y_t|x_t)} \right) + \frac{1}{2} \lambda \|\beta\|^2 \\
&= \left(\sum_{t=1}^T \ln Z_\beta(x_t) - \beta \cdot \Psi(x_t, y_t) \right) + \frac{1}{2} \lambda \|\beta\|^2 \\
\frac{\partial L}{\partial \beta_i} &= \left(\sum_{t=1}^T \frac{1}{Z_\beta(x_t)} \frac{\partial Z_\beta(x_t)}{\partial \beta_i} - \Psi_i(x_t, y_t) \right) + \lambda \beta_i \\
&= \sum_{t=1}^T \left(\frac{1}{Z_\beta(x)} \sum_{y \in \mathcal{Y}} \exp(\beta \cdot \Psi(x_t, y)) \Psi_i(x_t, y) - \Psi_i(x_t, y_t) \right) + \lambda \beta_i \\
&= \sum_{t=1}^T \left(\sum_{y \in \mathcal{Y}} \left(\frac{1}{Z_\beta(x)} \exp(\beta \cdot \Psi(x_t, y)) \right) \Psi_i(x_t, y) - \Psi_i(x_t, y_t) \right) + \lambda \beta_i \\
&= \sum_{t=1}^T \left(\sum_{y \in \mathcal{Y}} P_\beta(y|x) \Psi_i(x_t, y) - \Psi_i(x_t, y_t) \right) + \lambda \beta_i \\
&= \sum_{t=1}^T (\mathbb{E}_{y \sim P_\beta(\cdot|x_t)} [\Psi_i(x_t, y)] - \Psi_i(x_t, y_t)) + \lambda \beta_i \tag{14}
\end{aligned}$$

In gradient descent we have a sequence of parameter vectors $\beta^0, \beta^1, \beta^2, \beta^3, \dots$ where β^{j+1} is derived from β^j by moving in the direction of the negative gradient. In (naive) gradient descent we update parameter component β_i as follows where η is a constant called the learning rate.

$$\begin{aligned}
\beta_i^{j+1} &= \beta_i^j + \eta \left(-\frac{\partial L}{\partial \beta_i} \right) \\
&= \beta_i^j + \eta \left(\sum_{t=1}^T \Psi_i(x_t, y_t) - \sum_{t=1}^T \mathbb{E}_{y \sim P_{\beta^j}(\cdot|x_t)} [\Psi_i(x_t, y)] - \lambda \beta_i^j \right) \tag{15}
\end{aligned}$$

In the case where $\lambda = 0$ (no regularization) we have that equation (15) says that β_i should be increased when $\Psi_i(x_t, y_t)$ is larger than the expected value based on the the inputs x_1, \dots, x_t and the probability model $P_\beta(y|x^t)$. At the optimum with $\lambda = 0$ the actual average value and the expected value must agree.

5 HMM CRFs

In an HMM CRF, like in an HMM, we have that x is a sequence of observations and y is a corresponding sequence of hidden states. An HMM CRF uses the same feature map $\Psi(x, y)$ as does an HMM (as described in section 3). Like an HMM, An HMM CRF also uses equation (1) to find the most likely hidden sequence. The only difference between an HMM and an HMM CRF is the training algorithm used. A HMM selects β^* using (2) and (7) while an HMM CRF selects β^* using (11) and (12).

To set β^* using (11) we can use gradient descent using equation (15). To use (15) we need to be able to compute $E_{y \sim P_\beta(\cdot|x_t)}[\Psi_i(x_t, y)]$. For an HMM CRF, like an HMM, the features are defined by (4), (5) and (6). We now define the following quantities where s and w are hidden states and u is an observation.

$$Z_\beta(y^1 = s, x_t) = \sum_{y: y^1=s} \exp(\beta \cdot \Psi(x_t, y)) \quad (16)$$

$$Z_\beta(y^j = s, y^{j+1} = w, x_t) = \sum_{y: y^j=s, y^{j+1}=w} \exp(\beta \cdot \Psi(x_t, y)) \quad (17)$$

$$Z_\beta(y^j = s, x^j = u, x_t) = \sum_{y: y^j=s, x^j=u} \exp(\beta \cdot \Psi(x_t, y)) \quad (18)$$

These quantities can be computed in time linear in the length of the sequence using recursive conditioning or the junction tree algorithm for Markov random fields. Here the Markov random field has a node for each “subvariable” x_t^i or y_t^i with hyperedges of the form $\{y^1\}$, $\{y^j, y^{j+1}\}$, and $\{y^j, x^j\}$. We can then compute expected feature values needed for gradient descent as follows.

$$\begin{aligned} E_{y \sim P_\beta(\cdot|x_t)}[\Psi_s(x_t, y)] &= \frac{Z_\beta(y^1 = s, x_t)}{Z_\beta(x_t)} \\ E_{y \sim P_\beta(\cdot|x_t)}[\Psi_{s,w}(x_t, y)] &= \frac{\sum_{i=1}^{N_t-1} Z_\beta(y^i = s, y^{i+1} = w, x_t)}{Z_\beta(x_t)} \\ E_{y \sim P_\beta(\cdot|x_t)}[\Psi_{s,u}(x_t, y)] &= \frac{\sum_{i=1}^{N_t} Z_\beta(y^i = s, x^{i+1} = u, x_t)}{Z_\beta(x_t)} \end{aligned}$$

6 Minimum Distortion

It is natural to introduce a notion of distortion between structured labels. We let $d(\hat{y}, y_t)$ be the cost of labeling x_t with \hat{y} when the truth is y_t . This has a natural interpretation in terms of lossy compression. We can compress a file y_t into file x_t , and uncompress x_t into \hat{y} using equation (1). In lossy compression \hat{y} , can be

different from the source file y_t . In many AI applications the “compression” is done by nature. For example, a camera constructs an image x_t from a scene (set of 3-D objects) y_t and the decompression tries to reconstruct a scene \hat{y} from the image x_t . We want decompression to try to minimize the distortion between the recovered object \hat{y} and the true source y_t . We let d denote a distortion function where $d(\hat{y}, y_t)$ is the amount by which \hat{y} is distorted relative to y_t . We let D be a probability distribution over pairs $\langle x_t, y_t \rangle$. We can think of D as a “compression algorithm” — it is the relationship between x_t and y_t . We want the decompression algorithm (1) to minimize distortion as follows.

$$\beta^* = \operatorname{argmin}_{\beta} \mathbb{E}_{\langle x, y \rangle \sim D} [d(f_{\beta}(x_t), y_t)] \quad (19)$$

Minimizing (19) is similar to minimizing 0-1 loss in the case of binary classification. As in 0-1 loss, equation (19) cannot be regularized directly. The problem, as in the case of 0-1 loss, is that the distortion on the training data depends only the direction of β and not on its magnitude. For any arbitrarily small $\epsilon > 0$, we have $f_{\epsilon\beta}(x) = f_{\beta}(x)$. Rather than work directly with distortion loss on the training data we have to work with some sigmoidal or hinge loss which is sensitive to the length of β .

7 Generalization Bounds

Insight can be gained into regularization for distortion minimization by bounding the generalization distortion in terms of quantities measurable from the sample. Throughout this section we assume a fixed but arbitrary distribution D on $\mathcal{X} \times \mathcal{Y}$ and a feature map Ψ with $\Psi(x, y) \in \mathbb{R}^d$. Here we will assume that d is finite and a logarithmic dependence on d appears in the generalization bound. This bound can be improved to depend on “local dimension” — the maximum over $x \in \mathcal{X}$ of the dimension of the space spanned by $\{\Psi(x, y) : y \in \mathcal{Y}\}$. There are natural examples, e.g., HMMs with vector-valued observations, where d is infinite but the local dimension is finite. But for simplicity, here we assume that d is finite.

Theorem. For any distribution D and feature map Ψ with $\Psi(x, y) \in \mathbb{R}^d$ there exists a functional $Q(\beta)$ mapping a feature vector β into a distribution on \mathbb{R}^d such that with probability at least $1 - \delta$ over the choice of a sample S of T pairs drawn IID from D we have the

following for all β (simultaneously) in R^d .

$$\begin{aligned} & \mathbb{E}_{\langle x, y \rangle \sim D, \beta' \sim Q(\beta)} [d(y, f_{\beta'}(x))] \\ & \leq \frac{\sum_{t=1}^T L(\beta, x_t, y_t)}{T} + \text{pen}(T, d, \beta, \delta) \end{aligned} \quad (20)$$

$$\text{pen}(T, d, \beta, \delta) = \frac{\|\beta\|^2}{T} + \sqrt{\frac{\|\beta\|^2 \ln\left(\frac{2dT}{\|\beta\|^2}\right) + \ln\left(\frac{T+1}{\delta}\right)}{2T}} \quad (21)$$

$$L(\beta, x_t, y_t) = \max_{\hat{y} \in \mathcal{Y}} d(\hat{y}, y_t) I[m_t(\hat{y}) \leq r_t(\hat{y})] \quad (22)$$

$$m_t(\hat{y}) = \beta \cdot \Psi(x_t, f_\beta(x_t)) - \beta \cdot \Psi(x_t, \hat{y}) \quad (23)$$

$$r_t(\hat{y}) = \|\Psi(x_t, f_\beta(x_t)) - \Psi(x_t, \hat{y})\|_1 \quad (24)$$

$$\|\Psi\|_1 = \sum_i |\Psi_i|$$

Here we have that $m_t(\hat{y})$ is the margin relative to the candidate label \hat{y} and $r_t(\hat{y})$ is the margin requirement. An important property of this bound (or any generalization bound) is that it is consistent. More specifically, we consider the learning algorithm that minimizes the bound as follows.

$$\beta^* = \underset{\beta}{\text{argmin}} \frac{\sum_{t=1}^T L(\beta, x_t, y_t)}{T} + \text{pen}(T, d, \beta, \delta) \quad (25)$$

Training algorithm (25) is consistent — in the limit of infinite training data we have that (25) agrees with (19).

8 Hinge Loss

There are various notions of hinge loss for the structured case all of which are convex in β and reduce to the standard notion of hinge loss in the binary case. The generalization bound (20) loosely motivates the following.

$$\beta^* = \underset{\beta}{\text{argmin}} \left(\sum_{i=1}^T \max_{\hat{y} \in \mathcal{Y}} d(\hat{y}, y_t) (r_t(\hat{y}) - m_t(\hat{y})) \right) + \lambda \|\beta\|^2 \quad (26)$$

$$m_t(\hat{y}) = \beta \cdot \Psi(x_t, y_t) - \beta \cdot \Psi(x_t, \hat{y}) \quad (27)$$

$$r_t(\hat{y}) = \|\Psi(x_t, y_t) - \Psi(x_t, \hat{y})\|_1 \quad (28)$$

Note that (27) and (28) are subtly different from (23) and (24). To make the hinge loss convex in β we must replace $f_\beta(x_t)$ in (23) and (24) by y_t in (27) and (28).

Perhaps the most influential structured hinge loss has been that of Taskar et al. A variant of this hinge loss corresponds to simply dropping the distortion term from (26) as follows where $m_t(\hat{y})$ and $r_t(\hat{y})$ are defined by (27) and (28).

$$\beta^* = \operatorname{argmin}_{\beta} \left(\sum_{i=1}^T \max_{\hat{y} \in \mathcal{Y}} (r_t(\hat{y}) - m_t(\hat{y})) \right) + \lambda \|\beta\|^2 \quad (29)$$