Probability and Structure in Natural Language Processing

Noah Smith, Carnegie Mellon University

2012 International Summer School in Language and Speech Technologies
Where We Left Off

• Graphical models ... inference ... “max” inference and decoding with linear models (five views).

• Supervised learning:
  – Log loss
  – Error as loss
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Generative (Log Loss)</th>
<th>Error as Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>Convex optimization.</td>
<td>Optimizing a piecewise constant function.</td>
</tr>
<tr>
<td>Error-awareness</td>
<td>None</td>
<td>😊</td>
</tr>
<tr>
<td>Guarantees</td>
<td>Consistency.</td>
<td>None.</td>
</tr>
</tbody>
</table>
Discriminative Learning

• Various loss functions between log loss and error.

• Three commonly used in NLP:
  – Conditional log loss (”max ent,” CRFs)
  – Hinge loss (structural SVMs)
  – Perceptron’s loss

• We’ll discuss each, compare, and unify.
Log Loss (Second Version)

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \text{loss}(x_i, y_i; h_\mathbf{w}) + R(\mathbf{w})
\]

\[
\text{loss}(\mathbf{x}, \mathbf{y}; h_\mathbf{w}) = - \log p_\mathbf{w}(\mathbf{y} | \mathbf{x})
\]

• Can be understood as a generative model over \( \mathbf{Y} \), but does not model \( \mathbf{X} \).
  – “Conditional” model
Log Loss (Second Version)

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \text{loss}(x_i, y_i; h_{\mathbf{w}}) + R(\mathbf{w})
\]

\[
\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{y} | \mathbf{x})
\]

Examples:

• Logistic regression (for classification)
• MEMMMs
• CRFs
Log Loss (Second Version)

\[
\min_{w \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \text{loss}(x_i, y_i; h_w) + R(w)
\]

\[
\text{loss}(x, y; h_w) = - \log p_w(y \mid x)
\]

Computationally ...

• Convex and differentiable.
• Requires posterior inference, which can be expensive depending on the model’s structure.
• Linear decoding (for some parameterizations).
Log Loss (Second Version)

\[
\min_{w \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(x_i, y_i; h_w) + R(w)
\]

\[
\text{loss}(x, y; h_w) = -\log p_w(y \mid x)
\]

Error ...

• No notion of error.

• Learner wins by moving as much probability mass as possible to training examples’ correct outputs.
Log Loss (Second Version)

$$\min_{w \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \text{loss}(x_i, y_i; h_{w}) + R(w)$$

$$\text{loss}(x, y; h_{w}) = - \log p_{w}(y \mid x)$$

Guarantees...

• Consistency: if the true conditional model is in the right family, enough data will lead you to it.
Log Loss (Second Version)

\[
\min_{w \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \text{loss}(x_i, y_i; h_w) + R(w)
\]

\[
\text{loss}(x, y; h_w) = - \log p_w(y | x)
\]

Different parameterizations ... 

• Global log-linear (CRF):
  \[
  -w^\top g(x, y) + \log \sum_{y'} \exp w^\top g(x', y')
  \]

• Locally normalized log-linear (MEMM):
  \[
  - \sum_e \text{freq}(e; x, y) \left( w^\top g(e) - \log \sum_{e' \in C(e)} \exp w^\top g(e') \right)
  \]
## Comparing the Two Log Losses

<table>
<thead>
<tr>
<th></th>
<th>(-\log p_w(x, y))</th>
<th>(-\log p_w(y \mid x))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameterization</strong></td>
<td>Usually multinomials (BN-like).</td>
<td>Almost always log-linear (MN-like).</td>
</tr>
<tr>
<td><strong>Under the usual parameterization ...</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Computation</strong></td>
<td>Count and normalize.</td>
<td>Convex optimization.</td>
</tr>
<tr>
<td><strong>Error-awareness</strong></td>
<td>None.</td>
<td>Aware of the (Y)-prediction task, (approximates zero-one).</td>
</tr>
<tr>
<td><strong>Guarantees</strong></td>
<td>Consistency of joint.</td>
<td>Consistency of cond.</td>
</tr>
</tbody>
</table>
Hinge Loss

$$\min_{w \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} loss(x_i, y_i; h_w) + R(w)$$

$$loss(x, y; h_w) = -w^T g(x, y) + \max_{y'} w^T g(x, y') + error(y', y)$$

• Penalizes the model for letting competitors get close (in terms of score) to the correct answer $y$.
  – Can penalize them in proportion to their error.
Hinge Loss

\[
\min_{w \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} loss(x_i, y_i; h_w) + R(w)
\]

\[
loss(x, y; h_w) = -w^\top g(x, y) + \max_{y'} w^\top g(x, y') + error(y', y)
\]

Examples ...

- Perceptron (including Collins’ structured version)
  - Classic version ignores error term
- SVM and some structured variants:
  - Max-margin Markov networks (Taskar et al.)
  - MIRA (1-best, k-best)
Hinge Loss

\[
\min_{w \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \text{loss}(x_i, y_i; h_w) + R(w)
\]

\[
\text{loss}(x, y; h_w) = -w^\top g(x, y) + \max_{y'} w^\top g(x, y') + \text{error}(y', y)
\]

Computationally ...

• Convex, not everywhere differentiable.
  – Many specialized techniques now available.
• Requires MAP or “cost-augmented” MAP inference.
• Linear decoding.
Hinge Loss

\[
\min_{w \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \text{loss}(x_i, y_i; h_w) + R(w)
\]

\[
\text{loss}(x, y; h_w) = -w^\top g(x, y) + \max_{y'} w^\top g(x, y') + \text{error}(y', y)
\]

Error ...

• Built in.
• Most convenient when error function factors similarly to features \( g \).
Hinge Loss

\[
\min_{w \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \text{loss}(x_i, y_i; h_w) + R(w)
\]

\[
\text{loss}(x, y; h_w) = -w^\top g(x, y) + \max_{y'} w^\top g(x, y') + \text{error}(y', y)
\]

Guarantees ...

• Generalization bounds.
  – Not clear how seriously to take these in NLP; may not be tight enough to be meaningful.

• Often you will find *convergence* guarantees for optimization techniques.
They Are All Related

\[ \frac{1}{\beta} \log \sum_{y'} \exp \left[ \beta \left( w^\top (g(x, y') - g(x, y)) + \gamma \text{error}(y', y) \right) \right] \]

<table>
<thead>
<tr>
<th>Loss Type</th>
<th>( \beta )</th>
<th>( \gamma )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional log loss</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Perceptron’s hinge loss</td>
<td>( \infty )</td>
<td>0</td>
</tr>
<tr>
<td>Structural SVM’s hinge loss</td>
<td>( \infty )</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Softmax-margin (Gimpel and Smith, 2010)</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
CRFs, Max Margin, or Perceptron?

• For supervised problems, we do not expect large differences.

• Perceptron is easiest to implement.
  – With cost-augmented inference, it should get better and begins to approach MIRA and M³Ns.

• CRFs are best for probability fetishists.
  – Probably most appropriate if you are extending with latent variables; the jury is out.

• Not yet “plug and play..”
\( R(w) \)

• Regularization term – avoid overfitting
  – Usually means “avoid large magnitudes in \( w \)”

• (Log) Prior – respect background beliefs about the predictor \( h_w \)
• Usual starting point: squared $L_2$ norm
  – Computationally convenient (it’s strongly convex, it is its own Fenchel conjugate, ...)
  – Probabilistic view: Gaussian prior on weights (Chen and Rosenfeld, 2000)
  – Geometric view: Euclidean distance (original regularization method in SVMs)
  – Only one hyperparameter

$$R(w) = \lambda \| w \|_2^2 = \lambda \sum_{j} w_j^2$$
Another option: $L_1$-norm

- Computationally less convenient (not everywhere differentiable)
- Probabilistic view: Laplacian prior on weights (originally proposed as “lasso” in regression)
- Sparsity inducing (“free” feature selection)

$$R(w) = \lambda \| w \|_1 = \lambda \sum_{j} |w_j|$$
R(w)

• Lots of attention to this in machine learning.
• “Structured sparsity”
  – Want groups of features to go to zero, or group-
    internal sparsity, or ...
• Interpolation between $L_1$ and $L_2$ – “elastic
  net”
  – Sparsity but maybe better behaved
• This is not yet “plug and play.”
  – Optimization algorithm is heavily affected.
MAP Learning is Inference

• Seeking “most probable explanation” of the data, in terms of \( w \).
  – Explain the data: \( p(x, y \mid w) \)
  – Not too surprising: \( p(w) \)

• If we think of “\( W \)” as another random variable, MAP learning is MAP inference.
  – Looks very different from decoding!
  – But at a high level of abstraction, it is the same.
MAP Learning as a Graphical Model

\[
\exp -R(w) = p(w)
\]

\[
p_w(y)
\]

\[
p_w(x | y)
\]

- This is a view of learning a “noisy channel” model.
MAP Learning as a Graphical Model

\[ \exp -R(w) = p(w) \]

\[ p_w(Y | X) \]

- This is a view of learning in a CRF.
MAP Estimation for CRFs

$$\max_w p(w \mid x, y), \text{ which is MAP inference}$$

iterate to obtain gradient:

sufficient statistics from $p(y \mid x, w)$, obtained by posterior inference
How To Think About Optimization

• Depending on your choice of loss and R, different approaches become available.
  – Learning algorithms can interact with inference/decoding algorithms, too.

• In NLP today, it is probably more important to focus on the features, error function, and prior knowledge.
  – Decide what you want, and then use the best available optimization technique.
Key Techniques

• Quasi-Newton – batch method for differentiable loss functions
  – LBFGS, OWLQN when using $L_1$ regularization
• Stochastic subgradient ascent – online
  – Generalizes perceptron, MIRA, stochastic gradient ascent
  – Sometimes sensitive to step size
  – Can often use “mini-batches” to speed up convergence
• For error minimization: randomization
Pitfalls

• Engineering online learning procedures is tempting and *may* help you get better performance.
  – Without at least some analysis in terms of loss, error, and regularization, it’s unlikely to be useful outside your problem/dataset.

• When randomization is involved, look at variance across runs (Clark et al., 2011)

• Always tune hyperparameters (e.g., regularization strength) on *development* data!
Major Topics in Current Work

• Coping with approximate inference
• Exploiting incomplete data
  – Semisupervised learning
  – Creating features from raw text
  – Latent variable models (discussed tomorrow)
• Feature management
  – Structured sparsity (R)