We can now define an SVM objective function to optimize our parameters, characterized by the training pairs but also depends on a set of unobserved, latent variables $z$. Given a training set, we may be interested in predicting which of the unrated songs $\tilde{y}$ they would enjoy the most, which motivates the problem of modeling unseen relationships in the data.

### Latent Structured SVMs

Given a training set $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, standard Structural SVMs try to learn the following prediction function:

$$ h_w(x) = \arg\max_{y \in \mathcal{Y}} w \cdot \phi(x, y) $$

Often the relationship between $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ cannot be completely characterized by the training pairs but also depends on a set of unobserved, latent variables $z \in \mathcal{Z}$.

In this setting, our prediction function becomes:

$$ h_w(x) = \arg\max_{y} h_w(x, y, z) $$

We can now define an SVM objective function to optimize our parameters, $w$, as:

$$ \min_w \frac{1}{2} \|w\|^2 + C \max_{y, z} \left[ \max_{y \neq \hat{y}} f_w(x, \hat{y}, z) + \Delta(y, \hat{y}, z) - C \max_{z} f_w(x, y, z) \right] $$

where $\Delta(y, \hat{y}, z)$ is a loss function to define the error between predicted outputs and the ground truth, and $f$ is an energy function characterized by a structured model.

### Hinge-Loss Markov Random Fields

Hinge-Loss Markov Random Fields (HL-MRFs) specify probability density functions over convex energies of the form:

$$ f_w(x, y) = -\sum_i w_i [\max(\ell_i(x, y), 0)]^{p_i} $$

where $\ell$ is a linear function of $x$ and $y$, and $p \in \{1, 2\}$. The resulting density function can be written as:

$$ P(y|x) \propto \exp[f_w(x, y)] $$

HL-MRFs exhibit efficient inference for the Maximum a’Posteriori (MAP) and Most Probable Explanation (MPE) inference problems. Even more, the Alternating Direction Method of Multipliers (ADMM) can quickly solve inference problems with millions of variables.

### Ranking Loss Functions

In order to fully specify the Latent Structural SVM objective as a preference learning task by introducing HL-MRFs to define our energy functions, we must define the loss function to be specific to rankings/preferences.

The rank of an item $y$ in a preference list $\hat{y}$ can be written as:

$$ \text{rank}(y, \hat{y}) = \sum_{j \in \{1, \ldots, \hat{y}\}} \mathbb{1}[y_j > y] $$

A basic loss function thus takes the form:

$$ \Delta(y, \hat{y}, z) = \sum_{j \in \{1, \ldots, \hat{y}\}} \text{rank}(y, \hat{y}) + \sum_{j \in \{1, \ldots, \hat{y}\}} \cdot \text{rank}(j, \hat{y}) $$

Elements $y_i$ with high value should have low rank (i.e. top of the list), so penalize high scoring elements at the end of the list.

The rank indicator makes computing these losses intractable, but we can approximate with a convex upper bound (e.g. logistic loss):

$$ \mathbb{1}[y_j > y] \leq \log_2 \left( 1 + 2^{-(y_j - y)} \right) $$

With this bound, more complex losses can be developed, including losses based off of Normalized Discounted Cumulative Gain (NDCG):

$$ \Delta(y, \hat{y}, z) = \sum_{j \in \{1, \ldots, \hat{y}\}} \log_2 \left( 1 + \sum_{k=1}^{(\hat{y} - j)} 2^{-(k - (y_j - y))} \right) $$

### Loss-Augmented Inference

Loss-Augmented Inference is efficiently solved via DC programming methods such as DCA (small problems) and stochastic majorization-minimization (large problems).

### Goodreads Dataset Experiments

Goodreads.com is a “social cataloging” website for books, where users interact with each other in a social network and also discuss, rate, and review books. We extracted information including:

- 378K Book Reviews
- 10K Users
- 75K Books
- 401K Social Links

To encode structure in our experiments, we use Probabilistic Soft Logic, a probabilistic programming framework for templating HL-MRFs, to define logical rules as features in our model. Rules to infer user preferences in the model could include:

- "Friends have similar tastes."
- "FavoriteAuthor(Alice, Author) & Wrote(Author, Book) => Likes(Alice, Book)"
- "People enjoy all books from their favorite authors."
- "Friends(Alice, Bob) & Likes(Alice, Book) => Likes(Bob, Book)"