Structured Prediction
w/ Large Margin Methods

Thomas Hofmann
thomas.hofmann@ethz.ch
thofmann@google.com

Data Analytics Laboratory, ETH Zürich
& Google Engineering Center, Zürich

Machine Learning Summer School
Tübingen, September 3-4, 2013
Section 1

Motivation & Overview
Structured Prediction

Generalize supervised machine learning methods to deal with structured outputs and/or with multiple, interdependent outputs.

Structured objects such as sequences, strings, trees, labeled graphs, lattices, etc.

Multiple response variables that are interdependent = collective classification
Jiggsaw Metaphor

Holistic prediction $\neq$ independent prediction of pieces

It is not just about solving one instance of a puzzle, but learning how to solve a whole class of puzzles.

inspired by Ben Taskar’s tutorial
Natural Language Processing

- PoS tagging, named entity detection, language modeling
- Syntactic sentence parsing, dependency parsing
- Semantic parsing

- B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning, Max-Margin Parsing, EMNLP, 2004
- R. McDonald, K. Crammer, F. Pereira, Online large-margin training of dependency parsers. ACL 2005
- R. McDonald, K. Hannan, Kerry, T. Neylon, M. Wells, J. Reynar, Structured models for fine-to-coarse sentiment analysis, ACL 2007
- Y. Zhang, S. Clark, Syntactic processing using the generalized perceptron and beam search, Computational Linguistics 2011.
- C. Cherry, G. Foster, Batch tuning strategies for statistical machine translation, NAACL 2012.
Information Retrieval

- Learning to rank, e.g. search engines
- Multidocument summarization
- Whole page clickthrough prediction
- Entity linking and reference resolution

- L. Li et al: Enhancing diversity, coverage and balance for summarization through structure learning, WWW 2009.
- R. Sipos, P. Shivaswamy, T. Joachims: Large-margin learning of submodular summarization models, ACL 2012
Computer Vision

- Image segmentation
- Scene understanding
- Object localization & recognition

- T. Caetano & R. Hartley, ICCV 2009 Tutorial on Structured Prediction in Computer Vision
- A. Patron-Perez, M. Marszalek, I. Reid, A. Zisserman: Structured learning of human interactions in TV shows, PAMI 2012
Computational Biology

- Protein structure & function prediction
- Gene finding, structure prediction (splicing)

- Y. Liu, E. P. Xing, and J. Carbonell, Predicting protein folds with structural repeats using a chain graph model, ICML 2005
- G. Schweikert et al, mGene: Accurate SVM-based gene finding with an application to nematode genomes, Genome Res. 2009 19: 2133-2143
Overview

1. ⇒ Overview
2. Model
   ▶ Structured prediction SVM
   ▶ Margins & loss functions for structured prediction
3. Oracle-based Algorithms
   ▶ Cutting plane methods
   ▶ Subgradient-based approaches
   ▶ Frank-Wolfe algorithm
   ▶ Dual extragradient method
4. Decomposition-based Algorithms
   ▶ Representer theorem and dual decomposition
   ▶ Conditional random fields
   ▶ Exponentiated gradient
5. Conclusion & Discussion
Section 2

Model
Structured Prediction

- Input space $\mathcal{X}$, output space $\mathcal{Y}$
- $|\mathcal{Y}| = m$ can be large due to combinatorics
  - e.g. label combinations, recursive structures
- Given training data $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}, \ i = 1, \ldots, n$
  - drawn i.i.d. from unknown distribution $\mathcal{D}$
- Goal: find a mapping $F$

$$F : \mathcal{X} \rightarrow \mathcal{Y}$$

- with a small prediction error

$$\text{err}(F) = E_D [\triangle(Y, F(X))]$$

- relative to some loss function $\triangle : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_0^+$, with $\triangle(y, y) = 0$ and $\triangle(y, y') > 0$ for $y \neq y'$. 
Examples: Loss Functions

- **Multilabel prediction**
  - $\mathcal{Y} = \{-1, 1\}^k$
  - $\triangle(y, y') = \frac{1}{2}(k - \langle y, y' \rangle)$ (Hamming loss)

- **Taxonomy classification**
  - $\mathcal{Y} = \{1, \ldots, k\}$, $k$ classes arranged in a taxonomy
  - $\triangle(y, y') =$ tree distance between $y$ and $y'$
  - cf. [CH04, BMK12]

- **Syntactic parsing**
  - $\mathcal{Y} =$ labeled parse trees
  - $\triangle(y, y') =$ # labeled spans on which $y$ and $y'$ do not agree
  - cf. [TKC+04]

- **Learning to rank**
  - $\mathcal{Y} =$ permutations of set of items
  - $\triangle(y, y') =$ mean average precision of ranking $y'$ vs. optimal $y$
  - cf. [YFRJ07]
Multiclass Prediction

- Apply standard multiclass approach to $\mathcal{Y}$ with $|\mathcal{Y}| = m$.
- Define $\mathcal{Y}$-family of discriminant functions $f_y : \mathcal{X} \to \mathbb{R}$, $y \in \mathcal{Y}$
- Prediction based on winner-takes-all rule

$$F(x) = \arg \max_{y \in \mathcal{Y}} f_y(x)$$

- Typical: linear discriminants with weight vector $w_y \in \mathbb{R}^d$ and

$$f_y(x) := \langle \phi(x), w_y \rangle$$

- shared input representation via feature map $\phi : \mathcal{X} \to \mathbb{R}^d$
- Trained via one-vs-all or as a single 'machine'
- References: [RK04, WW99, CS02, LLW04]
Multiclass Prediction ⇐ Structured Prediction

- What happens as $m > n$?
  - Not enough training data to even have a single example for every output.

- Taking outputs as atomic entities without any internal structure does not enable generalization across outputs
  - There is no learning, only memorization of outputs.

- Need to go beyond the standard multiclass setting and enable learning across $\mathcal{X} \times \mathcal{Y}$. Two lines of thought:
  - **Feature-based Prediction**: extract features from inputs & outputs, define discriminant functions with those features
  - **Factor-based Prediction**: decompose output space into variables and identify factors [coming back to this later]
Feature-based Prediction

- Joint feature maps

\[
\psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d,
\quad k_\psi((x, y), (x', y')) := \langle \psi(x, y), \psi(x', y') \rangle
\]

to extract features from input-output pairs.

- Canonical construction by crossing features extracted separately from inputs and outputs

\[
\psi = \phi_X \times \phi_Y : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^{d_X \cdot d_Y},
\quad \psi(x, y) := \phi_X(x) \times \phi_Y(y).
\]

  - Can be more selective about features crossed (subsets).
  - Other constructions (beyond crossing) are possible.

- When using inner products one gets the compelling factorization

\[
k_\psi((x, y), (x', y')) = k_{\phi_X}(x, x') \cdot k_{\phi_Y}(y, y') .
\]
Example: Label Sequence (HMM) [ATH+03]

- Hidden Markov Models: \( \mathcal{X} = (\mathbb{R}^d)^l, \mathcal{Y} = \{1, \ldots, k\}^l \), where
  - \( l \): length of sequence
  - \( k \): cardinality of hidden variable
  - \( d \): dimensionality of observations

- First feature template: local observations

\[
\psi_1^c(x, y) = \sum_{t=1}^{l} \mathbf{1}[y_t = c] \cdot \phi(x^t)
\]

  - adding up all observations that are assigned to same class \( c \in \{1, \ldots k\} \)

- Second feature template: pairwise nearest neighbor interactions

\[
\psi_2^{c, \bar{c}}(x, y) = \sum_{t=1}^{l-1} \mathbf{1}[y_t = c] \cdot \mathbf{1}[y_{t+1} = \bar{c}]
\]

  - counting number of times labels \((c, \bar{c})\) are neighbors
Example: Optimizing Ranking [YFRJ07]

- Kandall’s tau:

\[
\tau = \frac{\text{# concord. pairs} - \text{# discord. pairs}}{\text{# all pairs}} = 1 - \frac{2 \cdot \text{# discordant pairs}}{\text{# all pairs}}
\]

- Output ranking encoded via pairwise ordering

\[Y = \{-1, 1\}^{k \times k}, \quad y_{ij} = \begin{cases} 1 & \text{if } i < j \\ -1 & \text{otherwise} \end{cases}\]

- Combined feature function

\[\psi(x, y) = \sum_{i<j} y_{ij} [\phi(x_i) - \phi(x_j)]\]

Bipartite case \(C^+(x) \cup C^-(x) = \{1, \ldots, k\}\), relev./non-relev. items

\[\psi(x, y) = \sum_{i \in C^+(x)} \sum_{j \in C^-(x)} y_{ij} [\phi(x_i) - \phi(x_j)]\]
Example: Learning Alignments [JHYY09]

- **Input:** two annotated (protein) sequences \( x = (s_a, s_b) \).
- **Output:** alignment \( y \) between two sequences \( x \)
- **Joint features:**

<table>
<thead>
<tr>
<th>AAB24882</th>
<th>TYHMCQFHCRYVNNHSGEKLYECNERSKAFSCPSHL0CHKRRIQIGKTHEHNQCGKAFT</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAB24881</td>
<td>---------------------------------------------------------------</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td><strong><strong>: .</strong></strong>: * <em>:</em>:* :<strong><em><em>:</em>:</em>:</strong>******:</td>
<td></td>
</tr>
<tr>
<td>AAB24882</td>
<td>PSKLQYHERHTHTGEKFYECQCGQAFKCSLLQRKHTHTGEKPYE-CNQCGKAFQ-</td>
<td>116</td>
</tr>
<tr>
<td>AAB24881</td>
<td>HSHLQCHKRTHTGEKFYECNQCGKAFSQMGLQQHRKHTHTGEKPYMNVINMVKPLHNS</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td><strong><strong>: :</strong><strong><em><strong><strong>:</strong></strong>:</em>:*:</strong>:</strong>*****<em>:</em>:.<em>:</em>:</td>
<td></td>
</tr>
</tbody>
</table>


- **Types of features:** combinations of amino acid, secondary structure, solvent accessibility; sliding window; PSI-BLAST profile scores;
Multiclass + Output Features = Structured Prediction

- Generalize multiclass prediction and define linear discriminants
  \[
  \text{multiclass} \rightarrow f_y(x; w) := f(x, y; w) = \langle \psi(x, y), w \rangle \leftrightarrow \text{structured}
  \]

- Parameter sharing across outputs (with same features)

- Recover feature-less multiclass by defining (1 out of \(m\) encoding)
  \[
  \langle \phi_Y(y), \phi_Y(y') \rangle = \delta_{yy'}
  \]
  i.e. feature vectors involving different classes \(y, y'\) are orthogonal.

- Allows to incorporate prior knowledge into multiclass problems
  - Hierarchical classification - encode class taxonomy []
  - Entity reference resolution - encode prior entity names and types

- Requires single 'machine' formulation as weight vectors are not separated \(\rightarrow\) How can we generalize SVMs?
Binary Support Vector Machine

Convex Quadratic Program (primal)

\[
(w^*, \xi^*) = \arg \min_{w,\xi \geq 0} \mathcal{H}(w, \xi) := \frac{\lambda}{2} \langle w, w \rangle + \frac{1}{n} \|\xi\|_1
\]

subject to \( y_i \langle w, \phi(x_i) \rangle \geq 1 - \xi_i \) \( (\forall i) \)

- Examples \((x_i, y_i) \in \mathcal{X} \times \{-1, 1\}, \ i = 1, \ldots, n\)
- Feature map \( \phi : \mathcal{X} \rightarrow \mathbb{R}^d \)
- Weight vector \( w \in \mathbb{R}^d \)
- Slack variables \( \xi_i \geq 0 \)
- Regularization parameter \( \lambda \in \mathbb{R}^+ \)
Margin-rescaled Constraints

For each instance \((x_i, y_i)\) define \(m := |\mathcal{Y}|\) constraints via

\[
f(x_i, y_i; w) - f(x_i, y; w) \geq \triangle(y_i, y) - \xi_i \quad (\forall y \in \mathcal{Y})
\]

- Require correct output \(y_i\) to be scored higher than all incorrect outputs \(y \neq y_i\) by a margin
- Adjust target margin for incorrect outputs to be \(\triangle(y_i, y)\)
- Provides an upper bound on the empirical loss via

\[
\xi_i^* = \max_y \{\triangle(y_i, y) - [f(x_i, y_i; w) - f(x_i, y; w)]
\]

\[
\geq \triangle(y_i, \hat{y}) - [f(x_i, y_i; w) - f(x_i, \hat{y}; w)] \geq \triangle(y_i, \hat{y})
\]

\(\leq 0\) for \(\hat{y}\)

where \(\hat{y}_i := \arg\max_y f(x_i, y; w)\) is the predicted output
Slack-rescaled Constraints

For each instance \((x_i, y_i)\) define \(m := |\mathcal{Y}| - 1\) constraints via

\[
f(x_i, y_i; w) - f(x_i, y; w) \geq 1 - \frac{\xi_i}{\triangle(y_i, y)} \quad (\forall y \in \mathcal{Y} - \{y_i\})
\]

- Require correct output \(y_i\) to be scored higher than all incorrect outputs \(y \neq y_i\) by a margin
- Penalize margin violations proportional to \(\triangle(y_i, y)\)
- Provides an upper bound on the empirical loss via

\[
\xi^*_i = \max_y \{\triangle(y_i, y) - \triangle(y_i, y)[f(x_i, y_i; w) - f(x_i, y; w)]\} \\
\geq \triangle(y_i, \hat{y}) - \triangle(y_i, \hat{y})[f(x_i, y_i; w) - f(x_i, \hat{y}; w)] \geq \triangle(y_i, \hat{y})
\]

where \(\hat{y}_i := \arg \max_y f(x_i, y; w)\) is the predicted output
Softmargin, Illustration

Geometric sketch
Structured Prediction SVM

Convex Quadratic Program (primal)

\[
(w^*, \xi^*) = \arg \min_{w, \xi \geq 0} \mathcal{H}(w, \xi) := \frac{\lambda}{2} \langle w, w \rangle + \frac{1}{n} \|\xi\|_1
\]

subject to

\[
\langle w, \delta\psi_i(y) \rangle \geq \Delta(y_i, y) - \xi_i \quad (\forall i, \forall y \in \mathcal{Y} - \{y_i\})
\]

binary:

\[
\langle w, y_i \phi(x_i) \rangle \geq 1 - \xi_i \quad (\forall i)
\]

where \(\delta\psi_i(y) := \psi(x_i, y_i) - \psi(x_i, y)\).

- Examples \((x_i, y_i) \in \mathcal{X} \times \mathcal{Y}, \ i = 1, \ldots, n\)
- Feature map \(\psi : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d\)
- Weight vector \(w \in \mathbb{R}^d\)
- Slack variables \(\xi_i \geq 0\), regularization parameter \(\lambda \in \mathbb{R}^+\)
- Generalizes multiclass SVM [CS02]
Representer Theorem

- Denote by $\mathcal{H}$ and RKHS on $\mathcal{X} \times \mathcal{Y}$ with kernel $k$. A sample set $S = \{(x_i, y_i) : i = 1, \ldots, n\}$ is given. Furthermore let $C(f; S')$ be a functional that depends on $f$ only through its values on the augmented sample $S' := \{(x_i, y) : (x_i, y_i) \in S\}$. Let $\Lambda$ be a strictly monotonically increasing function. Then the solution of the optimization problem $\hat{f}(S) := \arg \min_{f \in \mathcal{H}} C(f, S) + \Lambda(\|f\|_{\mathcal{H}})$ can be written as

$$\hat{f}(\cdot) = \sum_{i, y} \beta_{iy} k(\cdot, (x_i, y))$$

- Linear case

$$\hat{w} = \sum_{i, y} \beta_{iy} \psi(x_i, y)$$

Deriving the Wolfe Dual (1)

Lagrangian
\[ \mathcal{L}(\ldots) = \frac{\lambda}{2} \| w \|^2 + \frac{1}{n} \| \xi \|_1 - \sum_{i, y \neq y_i} \alpha_{iy} \left[ \langle \delta \psi_i(y), w \rangle - \Delta(y_i, y) + \xi_i \right] - \langle \xi, \hat{\xi} \rangle \]

Gradient components
\[ \nabla_{\xi} \mathcal{L} = \frac{1}{n} - \hat{\xi} - \sum_y \alpha_{\cdot y} = 0 \implies 0 \leq \sum_y \alpha_{iy} \leq \frac{1}{n} \quad (\forall i) \]
\[ \nabla_w \mathcal{L} = \lambda w - \sum_{i, y \neq y_i} \alpha_{iy} v_{iy} \delta \psi_i(y) = 0 \implies w^*(\alpha) = \frac{1}{\lambda} \sum_{i, y \neq y_i} \alpha_{iy} \delta \psi_i(y) \]

Re-writing in matrix notation as \( w(\alpha)^* = Q \alpha \) with
\[ Q := (Q_{r, iy}) \in \mathbb{R}^{d \times n(m-1)}, \quad \text{with} \quad Q_{\cdot, iy} := \frac{1}{\lambda} \delta \psi_i(y) \]
Deriving the Wolfe Dual (2)

Plugging-in solution and exploiting known inequalities

\[
\min_{\alpha \geq 0} h(\alpha) := \frac{1}{2} \|Q\alpha\|^2 - \langle \alpha, \triangle \rangle \quad \text{s.t.} \quad n \sum_y \alpha_{iy} \leq 1 \quad (\forall i)
\]

binary: \[
\frac{1}{2} \|\tilde{Q}\alpha\|^2 - \langle \alpha, 1 \rangle \quad \text{s.t.} \quad n\alpha_i \leq 1 \quad (\forall i)
\]

- Quantity: \( n \cdot m \) dual variables instead of \( n \)
- Quality: structure of dual is very similar
- Constraints only couple variables in blocks \( \{\alpha_{iy}: y \in \mathcal{Y} - \{y_i\}\} \)
- Natural factorization of \( \alpha \in \mathbb{R}^{n(m-1)} = \mathbb{R}^{(m-1)} \times \ldots \times \mathbb{R}^{(m-1)} \)
  \( n \) times
- \( \alpha/n \) is a probability mass function \( \alpha_{iy} := 1 - n \sum_{y \neq y_i} \alpha_{iy} \)
- What is a support vector? pair \( (i, y) \) with active constraint
Looking at the solution $w^*$ we see that

$$w^* = \sum_i \sum_{y \neq y_i} \alpha_{iy} \delta \psi_i(y) = \sum_i \sum_{y \neq y_i} \alpha_{iy} [\psi(x_i, y_i) - \psi(x_i, y)]$$

$$= \sum_i \left( \sum_{y \neq y_i} \alpha_{iy} \right) \psi(x_i, y_i) + \sum_i \sum_{y \neq y_i} (-\alpha_{iy}) \psi(x_i, y)$$

$$:= \beta_{iy}$$

$$= \sum_{i, y} \beta_{iy} \psi(x_i, y)$$

as it should be according to the representer theorem.
Section 3

Oracle-Based Algorithms
The Challenge

SVMstruct QP

$$(w^*, \xi^*) = \arg \min_{w, \xi \geq 0} \mathcal{H}(w, \xi) := \frac{\lambda}{2} \langle w, w \rangle + \frac{1}{n} \|\xi\|_1$$

with $$(w, \xi) \in \bigcap_{i=1}^{n} \Omega_{iy}, \quad y \in \mathcal{Y} - \{y_i\}$$

where \( \Omega_{iy} := \left\{ (w, \xi) : \langle w, \delta\psi_i(y) \rangle \geq \triangle(y_i, y) - \xi_i \right\} $$

- Structure of QP is not changed, but number of constraints can be vastly increased relative to binary classification
  - e.g. if \( \mathcal{Y} \) is vector of binary labels so that \( \mathcal{Y} = \{-1, 1\}^l \) and \( m = 2^l \)
- Scalable algorithms for this challenge? 10 years of research!
Structured Prediction Perceptron

- **Michael Collins 2002**, Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms [Col02]

- Perceptron learning avoids the challenge by only focusing on the worst output at a time
  - Instead of enforcing constraints over all possible incorrect outputs

- Standard perceptron algorithm with the following modifications
  - Compute prediction
    \[ \hat{y}_i := F(x_i) = \arg \max_y \langle w, \psi(x_i, y) \rangle \]
  - Perform update according to
    \[ w \leftarrow \begin{cases} 
    w + \psi(x_i, y_i) - \psi(x_i, \hat{y}) = w + \delta \psi_i(\hat{y}) & \text{if } \hat{y} \neq y_i \\
    w & \text{otherwise}
    \end{cases} \]
  - Novikoff's theorem and mistake bound can be generalized
Separation Oracles

- One idea of the perceptron algorithm turns out to be key: identify the output with the most violating margin constraint

- We call such a sub-routine a separation oracle

\[
\hat{y}_i \in \arg \max_y f(x_i, y; w)
\]

- Margin re-scaling \[
\hat{y}_i \in \arg \max_y \{\triangle(y_i, y) - f(x_i, y_i; w) + f(x_i, y; w)\}
\]

- Slack re-scaling \[
\hat{y}_i \in \arg \max_y \{\triangle(y_i, y)[1 - f(x_i, y_i; w) + f(x_i, y; w)]\}
\]

- Dependent on the method, the separation oracle is used to identify
  - violated constraints (successive strengthening)
  - update directions for the primal (subgradient)
  - variables in the dual (SMO)
  - update directions for the dual (Frank-Wolfe)
Successive QP Strengthening

- Create sequence of QPs that are relaxations of SVMstruct.
- Feasible domain $\Omega = \bigcap_{i y} \Omega_{i y} \cap (\mathbb{R}^d \times \mathbb{R}^n_{\geq 0})$
- Relaxed QP: same objective, yet $\hat{\Omega} \supset \Omega$
  - optimal solution $(\hat{w}, \hat{\xi})$ for relaxed QP will have $\mathcal{H}(\hat{w}, \hat{\xi}) \leq \mathcal{H}(w^*, \xi^*)$, but possibly $(\hat{w}, \hat{\xi}) \in \hat{\Omega} - \Omega$.
  - goal: fulfill remaining constraints with tolerance $\epsilon$, $(\hat{w}, \hat{\xi} + \epsilon) \in \Omega$
  - why? this would give $\mathcal{H}(\hat{w}, \hat{\xi} + \epsilon) = \mathcal{H}(\hat{w}, \hat{\xi}) + \epsilon \geq \mathcal{H}(w^*, \xi^*)$.
- Construct strict sequence of increasingly stronger relaxations via

$$\Omega(0) = \mathbb{R}^d \times \mathbb{R}^n_{\geq 0}, \quad \Omega(t + 1) := \Omega(t) \cap \Omega_{i \hat{y}}$$

where $(i, \hat{y})$ is a constraint selected at step $t$ fulfilling

$$\arg\min_{(w, \xi) \in \Omega(t)} \mathcal{H}(w, \xi) \notin \Omega_i^\epsilon, \quad \Omega_i^\epsilon := \{(w, \xi) : (w, \xi + \epsilon) \in \Omega_{i y}\}$$
Strengthening via Separation Oracle
Strengthening via Separation Oracle

- Loop through all training examples (in fixed order)
- Call separation oracle for \((x_i, y_i)\)
- Concretely for margin re-scaling

\[
\hat{y}_i \in \arg\max_y \left\{ \Delta(y_i, y) - f(x_i, y_i; w) + f(x_i, y; w) \right\}
\]

will identify (one of) the most violating constraint(s) for given \(i\), provided there are such constraints

- We can easily check, whether violation is \(> \epsilon\).
- Termination at step \(T\), if no such constraints exist for \(i = 1, \ldots, n\).

- Significance: can ensure \(T \leq O(n/\epsilon^2)\) or (with mild conditions) even \(T \leq O(n/\epsilon)\). No dependency on \(|\mathcal{Y}|\)!
Strengthening via Separation Oracle; Example \((n = 1)\)

- **Step 0**: \( \hat{(w, \xi)} = \arg \min_{\Omega(0)} \mathcal{H}(w, \xi) = (0, 0) \)
Strengthening via Separation Oracle; Example \((n = 1)\)

- Step 0: \((\hat{w}, \hat{\xi}) = \arg \min_{\Omega(0)} \mathcal{H}(w, \xi) = (0, 0)\)
- Step 1: \((\hat{w}, \hat{\xi}) = \arg \min_{\Omega(1)} \mathcal{H}(w, \xi), \text{ where}\)
  \[
  \hat{y} \in \arg \max_y \triangle(y, y),
  \]
  \[
  \Omega(1) = \Omega(0) \cap \Omega_{1\hat{y}}
  \]
Strengthening via Separation Oracle; Example \((n = 1)\)

- **Step 0:** \(\hat{(w, \xi)} = \text{arg min}_{\Omega(0)} \mathcal{H}(w, \xi) = (0, 0)\)

- **Step 1:** \(\hat{(w, \xi)} = \text{arg min}_{\Omega(1)} \mathcal{H}(w, \xi), \text{ where}\)

\[
\hat{y} \in \arg \max_y \Delta(y_1, y),
\]

\[
\Omega(1) = \Omega(0) \cap \Omega_{1\hat{y}}
\]

- **Step 2:** \(\hat{(w, \xi)} = \text{arg min}_{\Omega(2)} \mathcal{H}(w, \xi), \text{ where}\)

\[
\hat{y} \in \arg \max_y \Delta(y_1, y) - \langle \delta \psi_1(y), \hat{w} \rangle
\]

\[
\Omega(2) = \Omega(1) \cap \Omega_{1\hat{y}}
\]

provided that \(\Omega_{i\hat{y}} \cap \Omega(1) \neq \emptyset\).
Strengthening via Separation Oracle; Example \((n = 1)\)

- **Step 0:** \((\hat{w}, \hat{\xi}) = \arg\min_{\Omega(0)} H(w, \xi) = (0, 0)\)
- **Step 1:** \((\hat{w}, \hat{\xi}) = \arg\min_{\Omega(1)} H(w, \xi)\), where
  \[ \hat{y} \in \arg\max_y \Delta(y_1, y), \]
  \[ \Omega(1) = \Omega(0) \cap \Omega_{1\hat{y}} \]
- **Step 2:** \((\hat{w}, \hat{\xi}) = \arg\min_{\Omega(2)} H(w, \xi)\), where
  \[ \hat{y} \in \arg\max_y \Delta(y_1, y) - \langle \delta\psi_1(y), \hat{w} \rangle \]
  \[ \Omega(2) = \Omega(1) \cap \Omega_{1\hat{y}} \]
  provided that \(\Omega_{1\hat{y}} \cap \Omega(1) \neq \emptyset\).
- **Step 3:** \((\hat{w}, \hat{\xi}) = \arg\min_{\Omega(3)} H(w, \xi)\), where...
Improved Cutting Planes: Motivation

- Successive strengthening (as above) is expensive
  - only one constraint (for one example) gets added in each step
  - requires re-optimization (= solving a QP) after each such step
  - can warm-start, but still...

- How about, we compute all oracles in parallel
  \[
  \hat{y} = (\hat{y}_1, \ldots, \hat{y}_n) \in \mathcal{Y}^n
  \]

- Derive a strengthening from that \( \Omega(t + 1) = \Omega(t) \cap \Omega_{\hat{y}} \)

- Naively could set \( \Omega_{\hat{y}} := \bigcap_i \Omega_{i\hat{y}_i} \)
  - ... but how would that give us improved termination guarantees?
  - ... how can we avoid blow-up in number of constraints?

- Instead summarize into a single linear constraint with a single shared slack variable \( \zeta \geq 0 \). Fulfill margin on average

\[
\sum_{i=1}^{n} \langle \psi_i(\hat{y}_i), w \rangle \geq \sum_{i=1}^{n} \triangle(y_i, \hat{y}_i) - \zeta
\]
Improved Cutting Planes: Algorithm

- [JFY09] show that the QP containing all such average constraints for all combinations $\mathbf{y} \in \mathcal{Y}^n$ is solution equivalent to SVMstruct, if one identifies $\zeta = \|\xi_i\|_1$.

\[
\min_{\mathbf{w}, \xi} \frac{\lambda}{2} \langle \mathbf{w}, \mathbf{w} \rangle + \frac{1}{n} \|\xi\|_1 \quad \text{s.t.} \quad \langle \delta \psi_i(y), \mathbf{w} \rangle \geq \Delta(y_i, y) - \xi_i \\
(\forall i, y \in \mathcal{Y}) \sim n \cdot m
\]

\[
\min_{\mathbf{w}, \xi} \frac{\lambda}{2} \langle \mathbf{w}, \mathbf{w} \rangle + \zeta \quad \text{s.t.} \quad \sum_i \langle \delta \psi_i(y), \mathbf{w} \rangle \geq \sum_i \Delta(y_i, y) - \zeta \\
(\forall y \in \mathcal{Y}^n) \sim m^n
\]

- [JFY09] also provide $O(1/\epsilon)$-bounds on the number of epochs
  - overall runtime $O(n/\epsilon)$ (in the linear case), not counting oracle

- Dual QP optimization
  - one variable for each selected (average constraint), highly sparse
  - complexity of $O(n^2)$; with reduced rank approx. $O(nr + r^3)$
Improved Cutting Planes: Experiments

Experiments from [JFY09]

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>N</th>
<th>CPU-Time 1-slack</th>
<th>CPU-Time n-slack</th>
<th># Sep. Oracle 1-slack</th>
<th># Sep. Oracle n-slack</th>
<th># Support Vec. 1-slack</th>
<th># Support Vec. n-slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiC</td>
<td>522,911</td>
<td>378</td>
<td>1.05</td>
<td>1180.56</td>
<td>4,183,288</td>
<td>10,981,131</td>
<td>98</td>
<td>334,524</td>
</tr>
<tr>
<td>HMM</td>
<td>35,531</td>
<td>18,573,781</td>
<td>0.90</td>
<td>177.00</td>
<td>1,314,647</td>
<td>4,476,906</td>
<td>139</td>
<td>83,126</td>
</tr>
<tr>
<td>CFG</td>
<td>9,780</td>
<td>154,655</td>
<td>2.90</td>
<td>8.52</td>
<td>224,940</td>
<td>479,220</td>
<td>70</td>
<td>12,890</td>
</tr>
</tbody>
</table>

- # calls to separation oracle 2-3x reduced
- CPU time, 5x-1000x dependent on time spent on QP vs. oracle
  ⇒ much more efficient usage of optimization time
- 1000-10000x fewer support vectors, but not when multiplied by $n$
- Approximation result $O(1/\epsilon)$
- Book-keeping overhead for storing $\#$SVs $\cdot n$ descriptors of size $O(\log m)$
Subgradient Method for SVMstruct

- Can we avoid solving many relaxed QPs? How about a gradient descent flavor method?
- We can avoid linearizing (i.e. rolling out) the constraints. Work directly with (unconstrained) piecewise linear objective

\[
w^* = \arg \min_w \frac{\lambda}{2} \langle w, w \rangle + \frac{1}{n} \sum_{i=1}^{n} \max_y \{ \Delta(y_i, y) - \langle \delta \psi_i(y), w \rangle \}
\]

oracle \( \hat{y}_i := \arg \max() \)

- Compute subgradient, e.g. via

\[
g = \lambda w + \frac{1}{n} \sum_{i=1}^{n} \delta \psi_i(\hat{y}_i)
\]

- Perform batch or stochastic updates on \( w \) (learning rate?)
- Proposed by [RBZ07]; see also PEGASOS [SSSSC11]
Let $f: \mathbb{R}^D \rightarrow \mathbb{R}$ be a convex, not necessarily differentiable function. A vector $v \in \mathbb{R}^D$ is called a subgradient of $f$ at $x_0$, if

$$f(x) \geq f(x_0) + \langle v, x - x_0 \rangle$$

for all $x$.

Differentiable point $x_0$: unique subgradient = gradient $\nabla f(x_0)$. 
Frank-Wolfe Algorithm

- Frank & Wolfe, 1956: *An algorithm for quadratic programming*
- Minimize linearization at current iterate over corners of domain

\[
\text{'new iterate'} := (1 - \eta) \cdot \text{'old iterate'} + \eta \cdot \text{'optimal corner'}
\]

- Features
  - **linearity**: linear, not quadratic function minimization in every step
  - **sparseness**: convex combination of selected corners
  - **projection-free**: iterates stay in convex domain
  - **learning rate**: $O(1/t)$ schedule or via line search
  - **duality gap**: implicitly computes duality gap

- Applied to SVMstruct by [LJJSP13]
Frank-Wolfe Algorithm: Quadratic vs. Linearized

- Quadratic objective (contour line plot)
- Linearized objective
Frank-Wolfe Algorithm: Schematic 3D View

[taken from Lacoste-Julien et al., 2013]
Frank-Wolfe Algorithm: Dual SVM-struct Objective

- **Dual objective**

\[ h(\alpha) = \frac{1}{2} \| Q\alpha \|^2 - \langle \triangle, \alpha \rangle \]

- **Gradient**

\[ \nabla_{\alpha} h(\alpha^*) = (Q'Q)\alpha^* - \triangle \]

- **Linearization**

\[ \tilde{h}(\alpha; \alpha^*) = h(\alpha^*) + \langle \nabla_{\alpha} h(\alpha^*), \alpha - \alpha^* \rangle \leq h(\alpha) \]

- **Minimization problem**

\[ e^* := \arg \min_{\{e_r: r=1,\ldots,m\}} \tilde{h}(e_r; \alpha^*), \quad \text{with } e_r: r-th \text{ unit vector} \]
What does the minimization problem over corners mean?

\[
\bar{h}(e_y'; \alpha^*) + \text{const.} = \langle \nabla_\alpha h(\alpha^*), e_y' \rangle \\
= (Q'Q)\alpha^* - \triangle \\
= \langle Qe_y', Q\alpha^* \rangle + \triangle(y, y') \\
= \langle \sum_i \delta\psi_i(y'_i), w \rangle + \sum_i \triangle(y_i, y'_i)
\]

so that

\[
\hat{y}_i = \arg \max_{y'} \{ \langle \delta\psi_i(y'), w \rangle + \triangle(y_i, y') \}
\]

which is just the separation oracle!
Algorithms: Frank-Wolfe, Subgradient, Cutting Plane

- How does Frank-Wolfe relate to the other methods?
  - FW ↔ Subgradients:
    - Same update direction of primal solution \( w \)
    - But: Smarter step-size policy derived from dual (see below)
    - But: Duality gap for meaningful termination condition (see below)
  - FW ↔ improved cutting planes:
    - Selected dual variables correspond to added constraints
    - But: incremental update step vs. optimization of relaxed QP
    - But: \#SV can be larger due to incremental method, no need to re-formulate SVM struct
- Further advantages
  - Simple and clean analysis
  - Per-instance updates (block-coordinate optimization)
Apply Frank-Wolfe to dual QP, but translate into primal updates

Compute primal update direction (subgradient)

\[ \bar{w} := \frac{1}{\lambda} \sum_{i=1}^{n} \delta \psi_i(\hat{y}_i), \quad \bar{\Delta} := \frac{1}{n} \sum_{i=1}^{n} \Delta(y_i, \hat{y}_i) \]

Perform convex combination update

\[ w^{t+1} = (1 - \gamma^t)w^t + \gamma^t \bar{w}, \quad \Delta^{t+1} = (1 - \gamma^t)\Delta^t + \gamma \bar{\Delta} \]

here the optimal \( \gamma^t \) can be computed analytically (closed-form line search) from \( w^t, \bar{\Delta} \) and \( \bar{w} \)

Convergence rate: \( \epsilon \)-approximation is found in at most \( O\left(\frac{R^2}{\lambda \epsilon}\right) \) steps
Block-Coordinate Frank-Wolfe

- Domain of the dual QP factorizes $\alpha \in S^{n}_{m-1}$ (product of simplicies)

\[
\alpha = (\alpha_i)_{i=1}^{n}, \text{ s.t. } \alpha_i \geq 0 \text{ and } \langle \alpha_i, 1 \rangle = 1
\]

- Perform Frank-Wolfe update over each block (randomly selected).
  - single-instance mode: alternates single oracle call and update step
  - back to successive strengthening, but replace: re-optimization with fast updates
  - convergence rate analysis; duality gap as stopping criterion
  - excellent scalability
Frank-Wolfe Methods: Scalability [LJJSP13]

- Frank-Wolfe very similar to improved cutting plane method
- Block-coordinate version much faster, better than stochastic subgradient descent
- Main caveat: primal-dual version needs to store one weight vector per training instance!!
Implicit Oracle as LP Relaxation

- Sometimes, oracle can be integrated into the QP

\[
\max_{y \in Y} \{ \langle \delta \psi_i(y), w \rangle + \Delta(y_i, y) \}
\]

\[
= \max_{z_i \in Z} \langle z_i, c_i + F_i w \rangle + d_i
\]

- Examples: binary MRFs with sub modular potentials, matchings, tree-structured MRFs

- Saddle point formulation:

\[
\min_w \max_z \left\{ \frac{\lambda}{2} \|w\|^2 + \sum_{i=1}^n \langle z_i, c_i + F_i w \rangle - \langle \psi(x_i, y_i), w \rangle \right\}
\]

- Make use of extragradient method [TLJJ06] - gradients & projections
Bi-partite Matching

- Graph $\mathcal{G}(V, E)$ with $V = V^s \cup V^t$, $E = V^s \times V^t$
- Matching scores $s_{jk} \in \mathbb{R}$ for each edge $(j, k) \in E$.
- Alignment variables $y_{jk} \in \{0, 1\}$ and their relaxation $z_{jk} \in [0; 1]$
- LP relaxation of integer program

\[
\max_{0 \leq z \leq 1} \sum_{(j,k) \in E} s_{jk}z_{jk}, \quad \text{s.t.} \quad \sum_{j} z_{jk} \leq 1 \quad (\forall k) \quad \text{and} \quad \sum_{k} z_{jk} \leq 1 \quad (\forall j)
\]

- LP is guaranteed to have integral solutions
- Integrating into SVM struct QP

\[
\max_{\{0 \leq z_i \leq 1\}} \sum_{e \in E} z_{ie} \langle \psi(x_i, y_e), w \rangle + (1 - 2y_{ie})^{\text{Hamming loss}}
\]
Section 4

Decomposition-Based Algorithms
Factor Graphs

- In many cases of practical interest, the compatibility function naturally allows for an additive decomposition over factors or parts

\[ f(x, y) = \sum_{c \in C} f_c(x_c, y_c) \]

which can formally be described as a factor graph.

- In the linear case, this can be induced via a feature decomposition

\[ \psi(x, y) = \sum_{c \in C} \psi_c(x_c, y_c), \text{ such that} \]

\[ f(x, y; w) = \langle w, \psi(x, y) \rangle = \sum_c \langle w, \psi_c(x_c, y_c) \rangle =: f_c(x_c, y_c) \]

- We typically require that the loss decomposes in a compatible manner

\[ \Delta(y, y'; x) = \sum_{c \in C} \Delta_c(y_c, y'_c; x_c) \]
Representer Theorem for the Factorized Case

- Conditions as before but factor structure assumed. Denote configurations for factor \( c \) as \( z \in \mathcal{Z}(c) \).

- Representation

\[
 f(x, y) = \sum_{i=1}^{n} \sum_{c \in C} \sum_{z \in \mathcal{Z}(c)} \mu_{ic} \langle \psi_c(x_{ic}, z), \psi_c(x_{ic}, y_c) \rangle \\
\sum_c |\mathcal{Z}(c)| \ll |\mathcal{Y}| \\
\]

Note that this offers the possibility to

1. define kernels on a \textit{per factor} level
2. use a low-dimensional parametrization that does not need to rely on sparseness
Decomposing the Dual QP

- Dual has the following structure (rescaling by $n$ as appropriate to make $\alpha$ probability mass function)

$$
\min_{\alpha \geq 0} h(\alpha) := \frac{1}{2} \|Q\alpha\|^2 - \langle \alpha, \triangle \rangle \quad \text{s.t.} \quad \sum_y \alpha_{iy} = 1 \quad (\forall i)
$$

- Introduce marginal probabilities

$$
\mu_{icz} := \sum_{i, y} 1[y_c = z] \alpha_{iy}, \quad \sum_z \mu_{icz} = \sum_{i, y} \alpha_{iy} = 1
$$

- Decompose loss (similar for $Q^t Q$)

$$
\sum_y \alpha_{iy} \triangle(y_i, y) = \sum_y \alpha_{iy} \sum_c \triangle_c(y_{ic}, y_c) = \sum_{c, z} \left( \sum_y 1[y_c = z] \alpha_{iy} \right) \triangle_c(y_{ic}, z) = \mu_{icz}
$$
Decomposing the Dual QP (continued)

- Define with multi-index \((icz)\):
  \[
  Q_{\cdot, icz} := \psi_c(x_{ic}, y_{ic}) - \psi_c(x_{ic}, z), \quad \mu_C := (\mu_{icz}), \quad \Delta C := (\Delta_{icz})
  \]

- Factorized QP
  \[
  \mu_C^* = \arg \min_{\mu_C \geq 0} \left\{ \frac{1}{2} \| Q \mu_C \|^2 - \langle \mu_C, \Delta C \rangle \right\}
  \]
  s.t. \(\mu_C\) is on the marginal polytope

  - \(\mu_C\) needs to be normalized and locally consistent (non-trivial).
  - Objective broken up into parts - global view enforced via constraints!

- Example: **Singly connected factor graph**. Local consistency:
  \[
  \sum_{r: (r, s) \in C} \mu_{irs} = \mu_{is} \quad (\forall i, s)
  \]

- For general factor graphs: only enforce local consistency = **relaxation** in the spirit of approximate belief propagation [TGK03]
Conditional Exponential Family Models

- Structured prediction from a statistical modeling angle
- $f$ from some RHKS with kernel $k$ over $\mathcal{X} \times \mathcal{Y}$
- **Conditional exponential families**
  
  \[
p(y|x; f) = \exp[f(x, y) - g(x, f)], \quad \text{where}
  \]
  
  \[
g(x, f) := \int_{\mathcal{Y}} \exp[f(x, y)] \, d\nu(y)
  \]

- Univariate case ($y \in \mathbb{R}$), generalized linear models

  \[
p(y|x; w) = \exp[y\langle w, \phi(x) \rangle - g(x, w)]
  \]

- Non-parameteric models, e.g. ANOVA kernels

  \[
k(((x, y), (x', y')) = yy'k(x, x')
  \]
Conditional Random Fields

- Conditional log-likelihood criterion [LMP01, LZL04]

\[
    f^* := \arg \min_{f \in \mathcal{H}} \frac{\lambda}{2} \| f \|^2_{\mathcal{H}} \text{ stabilizer} - \frac{1}{n} \sum_{i=1}^{n} \log p(y_i|x_i; f)
\]

- Optimization methods:
  - improved iterative scaling [LMP01]
  - pre-conditioned conjugate gradient descent, limited memory quasi-Newton [SP03]
  - finite dimensional case: requires computing expectations of sufficient statistics \( \mathbb{E}[\psi(Y, x)] \) for \( x = x_i, i = 1, \ldots, n \).

\[
\nabla_w[...] \overset{!}{=} 0 \iff \lambda w^* = \frac{1}{n} \sum_{i=1}^{n} \psi(x_i, y_i) - \frac{1}{n} \sum_{i=1}^{n} \sum_{y \in \mathcal{Y}} \psi(x_i, y) p(y|x_i; w^*)
\]

\[
\text{sample statistics} \quad \text{expected statistics}
\]
Dual CRF

- Representer theorems apply to log-loss. Log-linear dual:

\[
\alpha^* = \arg \min_{\alpha \geq 0} h(\alpha) := \frac{1}{2} \| Q \alpha \|^2 + \sum_{i=1}^{n} \sum_{y \in \mathcal{Y}} \alpha_{iy} \log \alpha_{iy}
\]

\[\text{s.t. } \sum_{y \in \mathcal{Y}} \alpha_{iy} = 1 \quad (\forall i)\]

- Compare with SVM struct

\[
\frac{1}{2} \| Q \alpha \|^2 + \sum_{i=1}^{n} \sum_{y \in \mathcal{Y}} \alpha_{iy} \log \alpha_{iy}
\]

\[
- \sum_{i=1}^{n} \sum_{y \in \mathcal{Y}} \alpha_{iy} \Delta(y_i, y)
\]

- same data matrix \(Q\) constructed from \(\delta \psi_i(y)\) (or \(Q^t Q\) via kernels)
- same \(n\)-factor simplex constraints on \(\alpha\)
- entropy maximization, instead of linear penalty (based on loss)
Exponentiated Gradient Descent

- Exponentiated gradient descent [CGK+08] can be applied to solve both duals (hinge loss and logarithmic loss)

- General update equation

\[ \alpha_{iy}^{(t+1)} \propto \alpha_{iy}^{(t)} \cdot \exp [\nabla h(\alpha)] \]

\[ = \alpha_{iy}^{(t)} \cdot \exp [\lambda \langle w^*, \delta\psi_i(y) \rangle - \triangle(y_i, y)] \]

- Can be motivated by performing gradient descent on the canonical/natural parameters (and re-formulating in mean-value parameterization)

\[ \theta^{(t+1)} = \theta^{(t)} + \delta\theta^{(t)} \Rightarrow \alpha^{(t+1)} = \exp[\langle \psi, \theta^{(t+1)} \rangle] = \exp[\langle \psi, \delta\theta^{(t)} \rangle] \alpha^{(t)} \]

- on-line version: generalizes SMO for solving dual problem (when no closed form solution exists)
Factorized Exponentiated Gradient Descent

- Work with factorized dual QP: e.g. [TGK03], SMO over marginal variables $\mu_C$.
- Better: adopt exponentiated gradient descent [CM05, CGK+08]
- Derivation: summing on both sides of the update equation...

$$
\mu_{icz}^{(t+1)} = \sum_{y} 1[y_c = z] \alpha_{iy}^{(t)} \exp \left[ \lambda \langle w^*, \delta \psi_i(y) \rangle - \triangle(y_i, y) \right] \\
\propto \sum_{y} 1[y_c = z] \alpha_{iy}^{(t)} \exp \left[ \lambda \langle w^*, \psi_c(x_i, y_{iz}) - \psi_c(x_i, z) \rangle - \triangle(y_{iz}, z) \right] \\
= \mu_{icz}^{(t)} \cdot \exp \left[ \lambda \langle w^*, \psi_c(x_i, y_{iz}) - \psi_c(x_i, z) \rangle - \triangle(y_{iz}, z) \right]
$$

- $w^*$ can (representer theorem) computed from $\mu$ and $\psi_c$ (or via $k_c$), $\triangle_c$ terms.
- Similar for log-loss, faster convergence rates $O(\log 1/\epsilon)$. 
Section 5

Conclusion & Discussion
Structured Prediction

- Support Vector Machines: can be generalized to structured prediction in a scalable manner
- Oracle-based architecture: decouples general learning method from domain-specific aspects
- Features & loss function: can be incorporated in a flexible manner
- Kernels: efficient dual methods exist that can rely on kernels (crossed feature maps, factor-level kernels)
- Algorithms: rich set of scalable methods; cutting planes, subgradients, Frank-Wolfe, exponentiated gradient
- Decomposition-based methods: can exploit insights and algorithms from approximate probabilistic inference
- Conditional random fields: close relation (decomposition, dual, sparseness?)
- Applications: ever increasing number of applications and use cases
Yasemin Altun, Ioannis Tsochantaridis, Thomas Hofmann, et al.
Hidden Markov Support Vector Machines.

Alexander Binder, Klaus-Robert Müller, and Motoaki Kawanabe.
On taxonomies for multi-class image categorization.

Tighter bounds for structured estimation.

Exponentiated gradient algorithms for conditional random fields and max-margin markov networks.

Lijuan Cai and Thomas Hofmann.
Hierarchical document categorization with support vector machines.
Peter L Bartlett Michael Collins and Ben Taskar David McAllester.

Exponentiated gradient algorithms for large-margin structured classification.


Michael Collins.

Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms.


Koby Crammer and Yoram Singer.

On the algorithmic implementation of multiclass kernel-based vector machines.


Yoonkyung Lee, Yi Lin, and Grace Wahba.
Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. 


John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira.
Conditional random fields: Probabilistic models for segmenting and labeling sequence data.


John Lafferty, Xiaojin Zhu, and Yan Liu.


Nathan D Ratliff, J Andrew Bagnell, and Martin Zinkevich.
(approximate) subgradient methods for structured prediction.

Ryan Rifkin and Aldebaro Klautau.
In defense of one-vs-all classification.

Fei Sha and Fernando Pereira.
Shallow parsing with conditional random fields.

Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter.
Pegasos: Primal estimated sub-gradient solver for SVM.

Ben Taskar, Carlos Guestrin, and Daphne Koller.
Max-margin markov networks.
Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher Manning.
Max-margin parsing.

Ben Taskar, Simon Lacoste-Julien, and Michael I Jordan.
Structured prediction, dual extragradient and Bregman projections.

Jason Weston and Chris Watkins.
Support vector machines for multi-class pattern recognition.

Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims.
A support vector method for optimizing average precision.

Alan L Yuille and Anand Rangarajan.
The concave-convex procedure.