Algorithms for NLP

Classification III

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley
The Perceptron, Again

- Start with zero weights
- Visit training instances one by one
  - Try to classify

\[ \hat{y} = \arg \max_{\mathbf{w} \cdot f_i(y)} \]

\[ y \in \mathcal{Y}(\mathbf{x}) \]

- If correct, no change!
- If wrong: adjust weights

\[
\begin{align*}
\mathbf{w} &\leftarrow \mathbf{w} + f_i(y_i^*) \\
\mathbf{w} &\leftarrow \mathbf{w} - f_i(\hat{y}) \\
\mathbf{w} &\leftarrow \mathbf{w} + (f_i(y_i^*) - f_i(\hat{y})) \\
\mathbf{w} &\leftarrow \mathbf{w} + \Delta_i(\hat{y})
\end{align*}
\]

*mistake vectors*
What is the final value of \( w \)?

- Can it be an arbitrary real vector?
- No! It’s built by adding up feature vectors (mistake vectors).

\[
\begin{align*}
  w & = \Delta_i(y) + \Delta_i'(y') + \cdots \\
  w & = \sum_{i,y} \alpha_i(y) \Delta_i(y) \quad \text{mistake counts}
\end{align*}
\]

Can reconstruct weight vectors (the primal representation) from update counts (the dual representation) for each \( i \)

\[
\alpha_i = \langle \alpha_i(y_1), \alpha_i(y_2), \ldots, \alpha_i(y_n) \rangle
\]
Dual Perceptron

- Track mistake counts rather than weights
- Start with zero counts ($\alpha$)
- For each instance $x$
  - Try to classify
    - If correct, no change!
    - If wrong: raise the mistake count for this example and prediction

\[
\hat{y} = \arg\max_{y \in \mathcal{Y}(x_i)} \sum_{i', y'} \alpha_{i'}(y') \Delta_{i'}(y')^\top f_i(y)
\]

\[
\alpha_i(\hat{y}) \leftarrow \alpha_i(\hat{y}) + 1
\]

\[
\mathbf{w} = \sum_{i, y} \alpha_i(y) \Delta_i(y)
\]

\[
\hat{y} = \arg\max_{y \in \mathcal{Y}(x)} \mathbf{w}^\top f(y)
\]

\[
\mathbf{w} \leftarrow \mathbf{w} + \Delta_i(\hat{y})
\]
Dual / Kernelized Perceptron

- How to classify an example $x$?

$$score(y) = w^\top f_i(y) = \left( \sum_{i', y'} \alpha_{i'}(y') \Delta_{i'}(y') \right)^\top f_i(y)$$

$$= \sum_{i', y'} \alpha_{i'}(y') \left( \Delta_{i'}(y')^\top f_i(y) \right)$$

$$= \sum_{i', y'} \alpha_{i'}(y') \left( f_{i'}(y^*_{i'})^\top f_i(y) - f_{i'}(y')^\top f_i(y) \right)$$

$$= \sum_{i', y'} \alpha_{i'}(y') \left( K(y^*_{i'}, y) - K(y', y) \right)$$

- If someone tells us the value of $K$ for each pair of candidates, never need to build the weight vectors
Issues with Dual Perceptron

- Problem: to score each candidate, we may have to compare to all training candidates

\[
score(y) = \sum_{i', y'} \alpha_{i'}(y') \left( K(y^*_i, y) - K(y', y) \right)
\]

- Very, very slow compared to primal dot product!
- One bright spot: for perceptron, only need to consider candidates we made mistakes on during training
- Slightly better for SVMs where the alphas are (in theory) sparse

- This problem is serious: fully dual methods (including kernel methods) tend to be extraordinarily slow
- Of course, we can (so far) also accumulate our weights as we go...
Kernels: Who Cares?

- So far: a very strange way of doing a very simple calculation

- “Kernel trick”: we can substitute any* similarity function in place of the dot product

- Lets us learn new kinds of hypotheses

* Fine print: if your kernel doesn’t satisfy certain technical requirements, lots of proofs break. E.g. convergence, mistake bounds. In practice, illegal kernels sometimes work (but not always).
Some Kernels

- Kernels **implicitly** map original vectors to higher dimensional spaces, take the dot product there, and hand the result back.

- Linear kernel:
  \[ K(x, x') = x' \cdot x' = \sum_i x_i x'_i \]

- Quadratic kernel:
  \[ K(x, x') = (x \cdot x' + 1)^2 = \sum_{i,j} x_i x_j x'_i x'_j + 2 \sum_i x_i x'_i + 1 \]

- RBF: infinite dimensional representation
  \[ K(x, x') = \exp(-||x - x'||^2) \]

- Discrete kernels: e.g. string kernels, tree kernels
Want to compute number of common subtrees between $T, T'$

Add up counts of all pairs of nodes $n, n'$

- Base: if $n, n'$ have different root productions, or are depth 0:
  $$C(n_1, n_2) = 0$$

- Base: if $n, n'$ are share the same root production:
  $$C(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (1 + C(ch(n_1, j), ch(n_2, j)))$$
Kernelized SVM (trust me)

Primal formulation:

\[
\min_{w, \xi} \quad \frac{1}{2} \|w\|^2 + C \sum_i \xi_i
\]

\[
\forall i, y \quad w^\top f_i(y_i^*) \geq w^\top f_i(y) + \ell_i(y) - \xi_i
\]

\[
w = \sum_{i, y} \alpha_i(y) \left( f_i(y_i^*) - f_i(y) \right)
\]

Dual formulation:

\[
\min_{\alpha \geq 0} \quad \frac{1}{2} \left\| \sum_{i, y} \alpha_i(y) \left( f_i(y_i^*) - f_i(y) \right) \right\|^2 - \sum_{i, y} \alpha_i(y) \ell_i(y)
\]

\[
\forall i \quad \sum_y \alpha_i(y) = C
\]
Dual Formulation for SVMs

- We want to optimize: (separable case for now)

\[
\min_w \frac{1}{2} ||w||^2
\]

\[
\forall i, y \quad w^T f_i(y_i^*) \geq w^T f_i(y) + \ell_i(y)
\]

- This is hard because of the constraints
- Solution: method of Lagrange multipliers
- The *Lagrangian* representation of this problem is:

\[
\min_w \max_{\alpha \geq 0} \Lambda(w, \alpha) = \frac{1}{2} ||w||^2 - \sum_{i, y} \alpha_i(y) \left( w^T f_i(y_i^*) - w^T f_i(y) - \ell_i(y) \right)
\]

- All we’ve done is express the constraints as an adversary which leaves our objective alone if we obey the constraints but ruins our objective if we violate any of them
Lagrange Duality

- We start out with a constrained optimization problem:
  \[ f(w^*) = \min_w f(w) \]
  \[ g(w) \geq 0 \]

- We form the Lagrangian:
  \[ \Lambda(w, \alpha) = f(w) - \alpha g(w) \]

- This is useful because the constrained solution is a saddle point of \( \Lambda \) (this is a general property):
  \[ f(w^*) = \min_w \max_{\alpha \geq 0} \Lambda(w, \alpha) = \max_{\alpha \geq 0} \min_w \Lambda(w, \alpha) \]

Primal problem in \( w \)  
Dual problem in \( \alpha \)
Duality tells us that

\[
\min_w \max_{\alpha \geq 0} \left\{ \frac{1}{2}||w||^2 - \sum_{i,y} \alpha_i(y) \left( w^T f_i(y^*_i) - w^T f_i(y) - \ell_i(y) \right) \right\}
\]

has the same value as

\[ Z(\alpha) \]

\[
\max_{\alpha \geq 0} \min_w \left\{ \frac{1}{2}||w||^2 - \sum_{i,y} \alpha_i(y) \left( w^T f_i(y^*_i) - w^T f_i(y) - \ell_i(y) \right) \right\}
\]

- This is useful because if we think of the \( \alpha \)'s as constants, we have an unconstrained min in \( w \) that we can solve analytically.
- Then we end up with an optimization over \( \alpha \) instead of \( w \) (easier).
Minimize the Lagrangian for fixed $\alpha$’s:

$$\Lambda(w, \alpha) = \frac{1}{2} ||w||^2 - \sum_{i,y} \alpha_i(y) \left( w^T f_i(y_i^*) - w^T f_i(y) - \ell_i(y) \right)$$

$$\frac{\partial \Lambda(w, \alpha)}{\partial w} = w - \sum_{i,y} \alpha_i(y) (f_i(y_i^*) - f_i(y))$$

$$\frac{\partial \Lambda(w, \alpha)}{\partial w} = 0 \quad \Rightarrow \quad w = \sum_{i,y} \alpha_i(y) (f_i(y_i^*) - f_i(y))$$

So we have the Lagrangian as a function of only $\alpha$’s:

$$\min_{\alpha \geq 0} Z(\alpha) = \frac{1}{2} \left\| \sum_{i,y} \alpha_i(y) (f_i(y_i^*) - f_i(y)) \right\|^2 - \sum_{i,y} \alpha_i(y) \ell_i(y)$$
Primal vs Dual SVM

**Primal formulation:**

\[
\min_{\mathbf{w}, \xi} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i \\
\forall i, y \quad \mathbf{w}^\top f_i(y_i^*) \geq \mathbf{w}^\top f_i(y) + \ell_i(y) - \xi_i
\]

\[
\mathbf{w} = \sum_{i, y} \alpha_i(y) \left( f_i(y_i^*) - f_i(y) \right)
\]

**Dual formulation:**

\[
\min_{\alpha \geq 0} \quad \frac{1}{2} \left\| \sum_{i, y} \alpha_i(y) \left( f_i(y_i^*) - f_i(y) \right) \right\|_2^2 - \sum_{i, y} \alpha_i(y) \ell_i(y)
\]

\[
\forall i \quad \sum_y \alpha_i(y) = C
\]
Primal formulation:

\[
\begin{align*}
\min_{w,\xi} & \quad \frac{1}{2}\|w\|^2 + C \sum_{i} \xi_i \\
\forall i, y & \quad w^T f_i(y_i^*) \geq w^T f_i(y) + \ell_i(y) - \xi_i
\end{align*}
\]

\[
\begin{align*}
\min_{w} & \quad \frac{1}{2}\|w\|^2 + C \sum_{i} \left( \max_{y} (w^T f_i(y) + \ell_i(y)) - w^T f_i(y_i^*) \right)
\end{align*}
\]
Learning SVMs (Primal)

Primal formulation:

\[
\min_w \frac{1}{2} \|w\|^2_2 + C \sum_i \left( \max_y (w^T f_i(y) + \ell_i(y)) - w^T f_i(y^*_i) \right)
\]

Loss-augmented decode: \[\bar{y} = \arg\max_y (w^T f_i(y) + \ell_i(y))\]

\[
\min_w \frac{1}{2} \|w\|^2_2 + C \sum_i (w^T f_i(\bar{y}) + \ell_i(\bar{y}) - w^T f_i(y^*_i))
\]

\[\nabla_w = w + C \sum_i (f_i(\bar{y}) - f_i(y^*_i))\]

Use general subgradient descent methods! (Adagrad)
Learning SVMs (Dual)

- We want to find $\alpha$ which minimize

$$\min_{\alpha \geq 0} \frac{1}{2} \left\| \sum_{i,y} \alpha_i(y) (f_i(y^*) - f_i(y)) \right\|_2^2 - \sum_{i,y} \alpha_i(y) \ell_i(y)$$

$$\forall i \sum_y \alpha_i(y) = C$$

- This is a quadratic program:
  - Can be solved with general QP or convex optimizers
  - But they don’t scale well to large problems
  - Cf. maxent models work fine with general optimizers (e.g. CG, L-BFGS)

- How would a special purpose optimizer work?
Coordinate Descent I (Dual)

\[
\min_{\alpha \geq 0} \frac{1}{2} \left\| \sum_{i,y} \alpha_i(y) (f_i(y_i^*) - f_i(y)) \right\|_2^2 - \sum_{i,y} \alpha_i(y) \ell_i(y)
\]

- Despite all the mess, \( Z \) is just a quadratic in each \( \alpha_i(y) \)
- Coordinate descent: optimize one variable at a time

- If the unconstrained \( \text{argmin} \) on a coordinate is negative, just clip to zero...
Ordinarily, treating coordinates independently is a bad idea, but here the update is very fast and simple

\[
\alpha_i(y) \leftarrow \max \left( 0, \alpha_i(y) + \frac{\ell_i(y) - \mathbf{w}^\top (f_i(y^*_i) - f_i(y))}{\| (f_i(y^*_i) - f_i(y)) \|^2} \right)
\]

So we visit each axis many times, but each visit is quick

This approach works fine for the separable case

For the non-separable case, we just gain a simplex constraint and so we need slightly more complex methods (SMO, exponentiated gradient)

\[
\forall i, \quad \sum_y \alpha_i(y) = C
\]
What are the Alphas?

- Each candidate corresponds to a primal constraint

\[
\min_{w, \xi} \frac{1}{2} ||w||^2 + C \sum_i \xi_i \\
\forall i, y \quad w^T f_i(y^*_i) \geq w^T f_i(y) + \ell_i(y) - \xi_i
\]

- In the solution, an \( \alpha_i(y) \) will be:
  - Zero if that constraint is inactive
  - Positive if that constraint is active
  - i.e. positive on the support vectors

- Support vectors contribute to weights:

\[
w = \sum_{i,y} \alpha_i(y) (f_i(y^*_i) - f_i(y))
\]
Structure
Handwriting recognition

Sequential structure

[Slides: Taskar and Klein 05]
The screen was a sea of red

Recursive structure
What is the anticipated cost of collecting fees under the new proposal?

En vertu de nouvelle propositions, quel est le coût prévu de perception de les droits?
Structured Models

\[ \text{prediction}(x, w) = \arg \max_{y \in \mathcal{Y}(x)} \text{score}(y, w) \]

space of feasible outputs

Assumption:

\[ \text{score}(y, w) = w^\top f(y) = \sum_{p} w^\top f(y_p) \]

Score is a sum of local “part” scores

Parts = nodes, edges, productions
What is the anticipated cost of collecting fees under the new proposal?

\[ \sum_{y_{jk} \in y} w^T f(x_{jk}) = w^T f(x, y) \]

- association
- position
- orthography
Efficient Decoding

- Common case: you have a black box which computes

\[
prediction(x) = \arg \max_{y \in \mathcal{Y}(x)} w^T f(y)
\]

at least approximately, and you want to learn w

- Easiest option is the structured perceptron [Collins 01]
  - Structure enters here in that the search for the best y is typically a combinatorial algorithm (dynamic programming, matchings, ILPs, A*...)
  - Prediction is structured, learning update is not
Structured Margin (Primal)

Remember our primal margin objective?

$$\min_w \frac{1}{2} \|w\|^2 + C \sum_i \left( \max_y (w^T f_i(y) + \ell_i(y)) - w^T f_i(y^*_i) \right)$$

Still applies with structured output space!
Structured Margin (Primal)

Just need efficient loss-augmented decode:

$$\bar{y} = \arg\max_y (w^\top f_i(y) + \ell_i(y))$$

$$\min_w \frac{1}{2}\|w\|_2^2 + C \sum_i (w^\top f_i(\bar{y}) + \ell_i(\bar{y}) - w^\top f_i(y^*_i))$$

$$\nabla_w = w + C \sum_i (f_i(\bar{y}) - f_i(y^*_i))$$

Still use general subgradient descent methods! (Adagrad)
Structured Margin (Dual)

- Remember the constrained version of primal:

$$\min_{w, \xi} \frac{1}{2} \|w\|^2 + C \sum_i \xi_i$$

$$\forall i, y \quad w^T f_i(y^*_i) \geq w^T f_i(y) + \ell_i(y) - \xi_i$$

- Dual has a variable for every constraint here
We want:

$$\arg \max_y w^T f(\text{brace}, y) = \text{"brace"}$$

Equivalently:

\[
\begin{align*}
    w^T f(\text{brace}, \text{"brace"}) &> w^T f(\text{brace}, \text{"aaaaa"}) \\
    w^T f(\text{brace}, \text{"brace"}) &> w^T f(\text{brace}, \text{"aaaab"}) \\
    \cdots & \quad \text{a lot!} \\
    w^T f(\text{brace}, \text{"brace"}) &> w^T f(\text{brace}, \text{"zzzzz"})
\end{align*}
\]
We want:

\[ \arg \max_y \mathbf{w}^T f('It was red', y) = \hat{\mathbf{x}}_{ABCD} \]

Equivalently:

\[ \mathbf{w}^T f('It was red', \hat{\mathbf{x}}_{ABCD}) > \mathbf{w}^T f('It was red', \hat{\mathbf{x}}_{ABDF}) \]

\[ \mathbf{w}^T f('It was red', \hat{\mathbf{x}}_{ABCD}) > \mathbf{w}^T f('It was red', \hat{\mathbf{x}}_{ABCD}) \]

\[ ... \]

\[ \mathbf{w}^T f('It was red', \hat{\mathbf{x}}_{ABCD}) > \mathbf{w}^T f('It was red', \hat{\mathbf{x}}_{EFGH}) \]
Alignment example

- We want:

\[
\arg \max_y w^T f(\text{'What is the'}, y) = \begin{bmatrix}
1 \\
2 \\
3 \\
\end{bmatrix}
\]

- Equivalently:

\[
w^T f(\text{'What is the'}, \begin{bmatrix}
1 \\
2 \\
3 \\
\end{bmatrix}) > w^T f(\text{'Quel est le'}, \begin{bmatrix}
1 \\
2 \\
3 \\
\end{bmatrix})
\]

\[
w^T f(\text{'What is the'}, \begin{bmatrix}
1 \\
2 \\
3 \\
\end{bmatrix}) > w^T f(\text{'Quel est le'}, \begin{bmatrix}
1 \\
2 \\
3 \\
\end{bmatrix})
\]

...
A constraint induction method [Joachims et al 09]
- Exploits that the number of constraints you actually need per instance is typically very small
- Requires (loss-augmented) primal-decode only

Repeat:
- Find the most violated constraint for an instance:
\[
\forall y \quad w^T f_i(y^*_i) \geq w^T f_i(y) + l_i(y)
\]
\[
\text{arg max } y w^T f_i(y) + l_i(y)
\]
- Add this constraint and resolve the (non-structured) QP (e.g. with SMO or other QP solver)
Comparison

<table>
<thead>
<tr>
<th>Date</th>
<th>Method</th>
<th>J+M 16, 18, 19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct 20</td>
<td>Structured Classification III</td>
<td>Adagrad, Subgradient SVM</td>
</tr>
<tr>
<td>Oct 25</td>
<td>Structured Classification IV</td>
<td></td>
</tr>
</tbody>
</table>

![Graphs showing comparison between different methods for Constituency Parsing and Constituency Parsing, Neural CRF](image)

**Legend**

- Cutting Plane
- Online Cutting Plane
- Online Primal Subgradient & $L_1$
- Online Primal Subgradient & $L_2$
- Averaged Perceptron
- MIRA
- Averaged MIRA (MST built-in)
- Stochastic Gradient Descent
Option 0: Reranking

Input

x = “The screen was a sea of red.”

N-Best List
(e.g. n=100)

Baseline Parser

Non-Structured Classification

Output

[e.g. Charniak and Johnson 05]
Reranking

- **Advantages:**
  - Directly reduce to non-structured case
  - No locality restriction on features

- **Disadvantages:**
  - Stuck with errors of baseline parser
  - Baseline system must produce n-best lists
  - But, feedback is possible [McCloskey, Charniak, Johnson 2006]