Algorithms for NLP

Classification II

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley
Minimize Training Error?

- A loss function declares how costly each mistake is:

\[ \ell_i(y) = \ell(y, y_i^*) \]

- E.g. 0 loss for correct label, 1 loss for wrong label
- Can weight mistakes differently (e.g. false positives worse than false negatives or Hamming distance over structured labels)

- We could, in principle, minimize training loss:

\[
\min_w \sum_i \ell_i \left( \arg \max_y w^\top f_i(y) \right)
\]

- This is a hard, discontinuous optimization problem
Examples: Perceptron

- Separable Case
Examples: Perceptron

- Non-Separable Case
What do we want from our weights?

- Depends!
- So far: minimize (training) errors:

\[
\sum_i \text{step} \left( w^\top f_i(y_i^*) - \max_{y \neq y_i^*} w^\top f_i(y) \right)
\]

- This is the “zero-one loss”
  - Discontinuous, minimizing is NP-complete
  - Maximum entropy and SVMs have other objectives related to zero-one loss
Linear Models: Maximum Entropy

- Maximum entropy (logistic regression)
  - Use the scores as probabilities:
    \[ P(y|x, w) = \frac{\exp(w^T f(y))}{\sum_{y'} \exp(w^T f(y'))} \]
  - Maximize the (log) conditional likelihood of training data
    \[
    L(w) = \log \prod_i P(y_i^* | x_i, w) = \sum_i \log \left( \frac{\exp(w^T f_i(y_i^*))}{\sum_y \exp(w^T f_i(y))} \right)
    \]
    \[= \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right) \]
Maximum Entropy II

- **Motivation for maximum entropy:**
  - Connection to maximum entropy principle (sort of)
  - Might want to do a good job of being uncertain on noisy cases...
  - ... in practice, though, posteriors are pretty peaked

- **Regularization (smoothing)**

\[
\begin{align*}
\max_w & \quad \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right) - k ||w||^2 \\
\min_w & \quad k ||w||^2 - \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right)
\end{align*}
\]
Log-Loss

- If we view maxent as a minimization problem:

\[
\min_w \ k \|w\|^2 + \sum_i - \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right)
\]

- This minimizes the “log loss” on each example

\[
- \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right) = - \log P(y_i^*|x_i, w)
\]

- One view: log loss is an upper bound on zero-one loss
Maximum Margin

- **Non-separable SVMs**
  - Add slack to the constraints
  - Make objective pay (linearly) for slack:

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

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

- C is called the *capacity* of the SVM – the smoothing knob

- **Learning:**
  - Can still stick this into Matlab if you want
  - Constrained optimization is hard; better methods!
  - We’ll come back to this later

Note: exist other choices of how to penalize slacks!
Remember SVMs...

- We had a constrained minimization

\[
\min_{\mathbf{w}, \xi} \frac{1}{2} \| \mathbf{w} \|^{2} + C \sum_{i} \xi_{i}
\]

\[\forall i, y, \quad \mathbf{w}^{\top} \mathbf{f}_{i}(y_{i}^{\ast}) + \xi_{i} \geq \mathbf{w}^{\top} \mathbf{f}_{i}(y) + \ell_{i}(y)\]

- ...but we can solve for \(\xi_{i}\)

\[\forall i, y, \quad \xi_{i} \geq \mathbf{w}^{\top} \mathbf{f}_{i}(y) + \ell_{i}(y) - \mathbf{w}^{\top} \mathbf{f}_{i}(y_{i}^{\ast})\]

\[\forall i, \quad \xi_{i} = \max_{y} \left( \mathbf{w}^{\top} \mathbf{f}_{i}(y) + \ell_{i}(y) \right) - \mathbf{w}^{\top} \mathbf{f}_{i}(y_{i}^{\ast})\]

- Giving

\[
\min_{\mathbf{w}} \frac{1}{2} \| \mathbf{w} \|^{2} + C \sum_{i} \left( \max_{y} \left( \mathbf{w}^{\top} \mathbf{f}_{i}(y) + \ell_{i}(y) \right) - \mathbf{w}^{\top} \mathbf{f}_{i}(y_{i}^{\ast}) \right)
\]
Hinge Loss

- Consider the per-instance objective:

\[
\min_w \ k ||w||^2 + \sum_i \left( \max_y \left( w^T f_i(y) + \ell_i(y) \right) - w^T f_i(y_i^*) \right)
\]

- This is called the “hinge loss”
  - Unlike maxent / log loss, you stop gaining objective once the true label wins by enough
  - You can start from here and derive the SVM objective
  - Can solve directly with sub-gradient decent (e.g. Pegasos: Shalev-Shwartz et al 07)

Plot really only right in binary case
Max vs “Soft-Max” Margin

- **SVMs:**

\[
\min_w k||w||^2 - \sum_i \left( w^T f_i(y_i^*) - \max_y \left( w^T f_i(y) + \ell_i(y) \right) \right)
\]

You can make this zero

- **Maxent:**

\[
\min_w k||w||^2 - \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp \left( w^T f_i(y) \right) \right)
\]

... but not this one

- Very similar! Both try to make the true score better than a function of the other scores
  - The SVM tries to beat the augmented runner-up
  - The Maxent classifier tries to beat the “soft-max”
Loss Functions: Comparison

- **Zero-One Loss**

\[
\sum_i \text{step} \left( w^T f_i(y_i^*) - \max_{y \neq y_i^*} w^T f_i(y) \right)
\]

- **Hinge**

\[
\sum_i \left( w^T f_i(y_i^*) - \max_y \left( w^T f_i(y) + \ell_i(y) \right) \right)
\]

- **Log**

\[
\sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp \left( w^T f_i(y) \right) \right)
\]
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 nouvelles propositions, quel est le coût prévu de perception de les droits?
Structured Models

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

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
Apple Computer bought Smart Systems Inc. located in Arkansas.
Bilingual word alignment

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

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

En vertu de les nouvelles propositions, quel est le coût prévu de perception des droits?
Efficient Decoding

- Common case: you have a black box which computes

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

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?

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

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

Just need efficient loss-augmented decode:

\[
\bar{y} = \arg\max_y \left( w^\top f_i(y) + \ell_i(y) \right)
\]

\[
\min_w \quad \frac{1}{2} \|w\|_2^2 + C \sum_i \left( w^\top f_i(\bar{y}) + \ell_i(\bar{y}) - w^\top f_i(y_i^*) \right)
\]

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

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

- Remember the constrained version of primal:

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

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

\[ \arg \max_y w^\top f(\text{brace}, y) = \text{“brace”} \]

Equivalently:

\[ w^\top f(\text{brace}, \text{“brace”}) > w^\top f(\text{brace}, \text{“aaaaa”}) \]
\[ w^\top f(\text{brace}, \text{“brace”}) > w^\top f(\text{brace}, \text{“aaaab”}) \]
\[ \ldots \]
\[ w^\top f(\text{brace}, \text{“brace”}) > w^\top f(\text{brace}, \text{“zzzzz”}) \]
We want:

$$\arg \max_y \ w^T f(\text{"It was red"}, y) = \hat{x}_{AB}^{CD}$$

Equivalently:

$$w^T f(\text{"It was red"}, \hat{x}_{AB}^{CD}) > w^T f(\text{"It was red"}, \hat{x}_{AB}^{DF})$$

$$w^T f(\text{"It was red"}, \hat{x}_{AB}^{CD}) > w^T f(\text{"It was red"}, \hat{x}_{AB}^{CD})$$

$$\ldots$$

$$w^T f(\text{"It was red"}, \hat{x}_{AB}^{CD}) > w^T f(\text{"It was red"}, \hat{x}_{EF}^{GH})$$

a lot!
Alignment example

- **We want:**

\[
\arg \max_y \ w^\top f(\text{'What is the'}, y) = \begin{array}{c}
1 \\
2 \\
3
\end{array} \quad \begin{array}{c}
1 \\
2 \\
3
\end{array}
\]

- **Equivalently:**

\[
w^\top f(\text{'What is the'}, \begin{array}{c}
1 \\
2 \\
3
\end{array}) > w^\top f(\text{'Quel est le'}, \begin{array}{c}
1 \\
2 \\
3
\end{array})
\]

\[
w^\top f(\text{'What is the'}, \begin{array}{c}
1 \\
2 \\
3
\end{array}) > w^\top f(\text{'Quel est le'}, \begin{array}{c}
1 \\
2 \\
3
\end{array})
\]

\[\ldots\]

\[
w^\top f(\text{'What is the'}, \begin{array}{c}
1 \\
2 \\
3
\end{array}) > w^\top f(\text{'Quel est le'}, \begin{array}{c}
1 \\
2 \\
3
\end{array})
\]

\[\text{a lot!}\]
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:
  \[ \forall y \quad w^T f_i(y^*_i) \geq w^T f_i(y) + \ell_i(y) \]
  \[ \arg \max_y w^T f_i(y) + \ell_i(y) \]
- Add this constraint and resolve the (non-structured) QP (e.g. with SMO or other QP solver)
Some issues:

- Can easily spend too much time solving QPs
- Doesn’t exploit shared constraint structure
- In practice, works pretty well; fast like perceptron/MIRA, more stable, no averaging
Likelihood, Structured

\[ L(w) = -k||w||^2 + \sum_i \left( w^\top f_i(y^*_i) - \log \sum_y \exp(w^\top f_i(y)) \right) \]

\[ \frac{\partial L(w)}{\partial w} = -2kw + \sum_i \left( f_i(y^*_i) - \sum_y P(y|x_i)f_i(y) \right) \]

- **Structure needed to compute:**
  - Log-normalizer
  - Expected feature counts
    - E.g. if a feature is an indicator of DT-NN then we need to compute posterior marginals \( P(DT-NN|sentence) \) for each position and sum

- **Also works with latent variables (more later)**
Comparison
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

[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]
M3Ns

- Another option: express all constraints in a packed form
  - Maximum margin Markov networks [Taskar et al 03]
  - Integrates solution structure deeply into the problem structure

- Steps
  - Express inference over constraints as an LP
  - Use duality to transform minimax formulation into min-min
  - Constraints factor in the dual along the same structure as the primal; alphas essentially act as a dual “distribution”
  - Various optimization possibilities in the dual
Example: Kernels

- Quadratic kernels

\[
K(x, x') = (x \cdot x' + 1)^2
\]

\[
= \sum_{i,j} x_i x_j x'_i x'_j + 2 \sum_i x_i x'_i + 1
\]

\[
K(y, y') = (f(y)^\top f(y') + 1)^2
\]
Non-Linear Separators

- Another view: kernels map an original feature space to some higher-dimensional feature space where the training set is (more) separable

\[ \Phi: y \rightarrow \varphi(y) \]
Why Kernels?

- Can’t you just add these features on your own (e.g. add all pairs of features instead of using the quadratic kernel)?
  - Yes, in principle, just compute them
  - No need to modify any algorithms
  - But, number of features can get large (or infinite)
  - Some kernels not as usefully thought of in their expanded representation, e.g. RBF or data-defined kernels [Henderson and Titov 05]

- Kernels let us compute with these features implicitly
  - Example: implicit dot product in quadratic kernel takes much less space and time per dot product
  - Of course, there’s the cost for using the pure dual algorithms...