Max-Margin Additive Classifiers for Detection

Subhransu Maji and Alexander C. Berg

Sam Hare
VGG Reading Group
October 30, 2009
Introduction

▶ CVPR08: SVMs with additive kernels can be evaluated efficiently.
▶ This work: SVMs with additive kernels can be trained efficiently.
Additive classifiers

- Classifiers of the form:

\[ \text{sign}(f(x)) \text{ where } f(x) = \sum_i f_i(x_i) \]

- Sum of coordinate functions \( f_i \).

- An SVM with an additive kernel is an additive classifier.
Additive kernels

When $k(x, y) = \sum_i k_i(x_i, y_i)$, SVM decision function becomes:

$$h(x) = \sum_j \alpha_j k(x, v^j) + b$$

$$= \sum_j \alpha_j \sum_i k_i(x_i, v^j_i) + b$$

$$= \sum_i \sum_j \alpha_j k_i(x_i, v^j_i) + b$$

$$= \sum_i f_i(x_i) + b$$

- CVPR08: Approximate each $f_i$ as piecewise constant or piecewise linear.
- Cost of evaluating decision function now independent of number of support vectors (like for a linear SVM).
Learning $f_i$ directly

Previously, kernelised SVM trained as usual, and then $f_i$ approximated. Now want to learn $f_i$ directly.

- Assume parametric decision function $f^w(x) = \sum_i f_i^w(x_i)$.
- Encode parameters $w$ as $\hat{w}$ and data points $x$ as $\hat{x}$ such that $f^w(x) \approx \hat{w}^T \hat{x}$.
- Solve for optimal decision function in large-margin sense:

$$f^w_* = \arg\min_{f^w} \left\{ R(\hat{w}) + \frac{1}{n} \sum_k \max(0, 1 - y^k \hat{w}^T \hat{x}^k) \right\}$$

- Basically a standard linear SVM objective (depending on form of $R$).
Learning $f_i$ directly (cont.)

- When $f^w$ is additive and $f_i^{w_i}$ are a linear combination of a finite number of basis functions, $\hat{w} = w$.

$$f^w(x) = \sum_i \sum_j w_{i,j} g_{i,j}(x_i)$$

$$= \sum_i w_i^T \hat{x}_i$$

$$= w^T \hat{x}$$
Specific case: SVM with histogram intersection kernel

- Given support vectors \( \{ v^j \} \) and coefficients \( \{ \alpha_j \} \), coordinate function is:

\[
f_i(x_i) = \sum_j \alpha_j \min(x_i, v^j_i)
\]

- By encoding \( \min(x, y) \approx \phi(x)^T \phi(y) \), can rewrite \( f_i \) in desired form:

\[
f_i(x_i) \approx w_i^T \phi(x_i)
\]

where

\[
w_i = \sum_j \alpha_j \phi(v^j_i)
\]
Encoding \( \min(x, y) \approx \phi(x)^T \phi(y) \)

Two options proposed in paper (assume \( x \in [0, 1] \)):

- \( \phi_1(x) = \frac{1}{\sqrt{N}} U(R(Nx)) \) where e.g. \( U(3) = (1, 1, 1, 0, 0, 0) \), \( R \) is a rounding function, and \( N \) determines discretisation scale.
- \( \phi_2(x) = \frac{1}{\sqrt{N}} U'(Nx) \) where e.g. \( U'(3.5) = (1, 1, 1, 0.5, 0, 0) \), i.e. no rounding.

Figure 1. From left to right \( \min(x, y) \), \( \phi_1(x)\phi_1(y) \) and \( \phi_2(x)\phi_2(y) \) with \( N = 10 \). Note that the \( \phi_2 \) encoding is very close to \( \min \).

Quality of \( \min \) approximation with \( \phi_2 \) is much better.
What I don’t get...

- $\phi$ serves two purposes:
  - Approximates min.
  - Determines form of $f_i$, since $f_i(x_i) = w_i^T \phi(x_i)$.

- $\phi_1$ causes $f_i$ to be piecewise constant, and $\phi_2$ causes $f_i$ to be piecewise linear, both with uniformly spaced breaks.
  - These were the same approximations for $f_i$ considered in CVPR08 work.
  - Presumably this is intentional, but paper didn’t really make the connection clear.
Sparsity

- In principle, could now apply $\phi$ to training data and use off-the-shelf linear SVM solver.
- But impractical as encoding dense.
- Modify encoding slightly (see details in paper) to give $\phi_1^s$ and $\phi_2^s$, with only 1 or 2 nonzero values respectively.
- Also need to modify encoding of $w$ such that $w^s T \phi^s(x) = w^T \phi(x)$.
- With $w^s$, regularisation becomes non-standard, so custom solver (variant of PEGASOS) developed.
## Results: Caltech 101

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Training Algorithm</th>
<th>15 examples</th>
<th>30 examples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training Time(s)</td>
<td>Accuracy(%)</td>
<td>Training Time(s)</td>
</tr>
<tr>
<td>identity</td>
<td>LIBLINEAR</td>
<td>18.57 (0.87)</td>
<td>41.19 (0.94)</td>
</tr>
<tr>
<td>identity</td>
<td>LIBSVM (int kernel)</td>
<td>844.13 (2.10)</td>
<td>50.15 (0.61)</td>
</tr>
<tr>
<td>snow=φ₁</td>
<td>LIBLINEAR</td>
<td>45.22 (1.17)</td>
<td>46.02 (0.58)</td>
</tr>
<tr>
<td>φ₂</td>
<td>LIBLINEAR</td>
<td>42.31 (1.43)</td>
<td>48.70 (0.61)</td>
</tr>
<tr>
<td>φ₂</td>
<td>PWLSGD</td>
<td>238.98 (2.49)</td>
<td>49.89 (0.45)</td>
</tr>
</tbody>
</table>
Results: Daimler Chrysler pedestrians

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Training Algorithm</th>
<th>Training Time(s)</th>
<th>Accuracy(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity</td>
<td>LIBLINEAR</td>
<td>2.98 (0.003)</td>
<td>85.71 (1.43)</td>
</tr>
<tr>
<td>identity</td>
<td>LIBLINEAR</td>
<td>1.86 (0.004)</td>
<td>88.80 (1.62)</td>
</tr>
<tr>
<td>snow=φ₁</td>
<td>LIBLINEAR</td>
<td>3.18 (0.001)</td>
<td>89.25 (1.58)</td>
</tr>
<tr>
<td>φ₂</td>
<td>LIBLINEAR</td>
<td>2.98 (0.003)</td>
<td>85.71 (1.43)</td>
</tr>
<tr>
<td>φ₂</td>
<td>PWLSGD</td>
<td>3.18 (0.001)</td>
<td>89.25 (1.58)</td>
</tr>
<tr>
<td></td>
<td>LIBSVM (int. kernel)</td>
<td>363.10 (27.85)</td>
<td>89.05 (1.42)</td>
</tr>
</tbody>
</table>
Results: INRIA pedestrians

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Training Algorithm</th>
<th>Training Time (HOG)</th>
<th>Training Time (spHOG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity</td>
<td>LIBLINEAR</td>
<td>-</td>
<td>20.12s</td>
</tr>
<tr>
<td>identity</td>
<td>LIBSVM (lin. kernel)</td>
<td>&gt;180 min</td>
<td>140 min</td>
</tr>
<tr>
<td>identity</td>
<td>LIBSVM (int. kernel)</td>
<td>&gt;180 min</td>
<td>148 min</td>
</tr>
<tr>
<td>snow=ϕ₁</td>
<td>LIBLINEAR</td>
<td>35.52s</td>
<td>121.81s</td>
</tr>
<tr>
<td>ϕ₂</td>
<td>LIBLINEAR</td>
<td>22.45s</td>
<td>26.76s</td>
</tr>
<tr>
<td>ϕ₂</td>
<td>PWLSGD</td>
<td>99.85s</td>
<td>76.12s</td>
</tr>
</tbody>
</table>

- 100 times faster training than CVPR08.