

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:

$$\text{sign}(f(x)) \quad \text{where} \quad f(x) = \sum_i f_i(x_i)$$

- ▶ 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:

$$\begin{aligned}h(x) &= \sum_j \alpha_j k(x, v^j) + b \\&= \sum_j \alpha_j \sum_i k_i(x_i, v_i^j) + b \\&= \sum_i \sum_j \alpha_j k_i(x_i, v_i^j) + b \\&= \sum_i f_i(x_i) + b\end{aligned}$$

- ▶ 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 $f^w(x) = \sum_i f_i^{w_i}(x_i)$.
- ▶ 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:

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

- ▶ 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, $\hat{w} = w$.

$$\begin{aligned} f^w(x) &= \sum_i \sum_j w_{i,j} g_{i,j}(x_i) \\ &= \sum_i w_i^T \hat{x}_i \\ &= w^T \hat{x} \end{aligned}$$

Specific case: SVM with histogram intersection kernel

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

$$f_i(x_i) = \sum_j \alpha_j \min(x_i, v_i^j)$$

- ▶ 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

$$w_i = \sum_j \alpha_j \phi(v_i^j)$$

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.

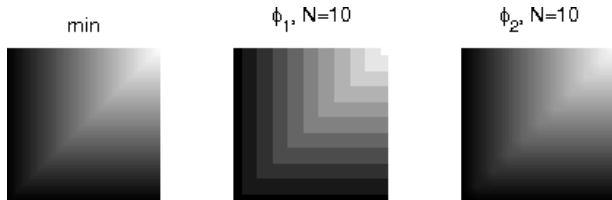


Figure 1. From left to right $\min(x, y)$, $\phi_1(x)\phi_1(y)$ and $\phi_2(x)\phi_2(y)$ with $N = 10$. Note that the ϕ_2 encoding is very close to min.

Quality of min approximation with ϕ_2 is much better.

What I don't get...

- ▶ ϕ serves two purposes:
 - ▶ Approximates min.
 - ▶ Determines form of f_i , since $f_i(x_i) = w_i^T \phi(x_i)$.
- ▶ ϕ_1 causes f_i to be piecewise constant, and ϕ_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 ϕ 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 ϕ_1^s and ϕ_2^s , with only 1 or 2 nonzero values respectively.
- ▶ Also need to modify encoding of w such that $w^{sT} \phi^s(x) = w^T \phi(x)$.
- ▶ With w^s , regularisation becomes non-standard, so custom solver (variant of PEGASOS) developed.

Results: Caltech 101

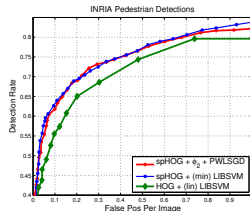
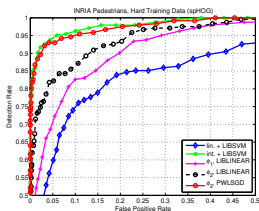
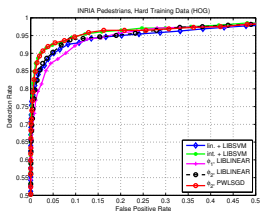
Encoding	Training Algorithm	15 examples		30 examples	
		Training Time(s)	Accuracy(%)	Training Time(s)	Accuracy(%)
identity	LIBLINEAR	18.57 (0.87)	41.19 (0.94)	40.49 (0.80)	46.15 (1.33)
identity	LIBSVM (int kernel)	844.13 (2.10)	50.15 (0.61)	2686.87 (4.30)	56.49 (0.78)
snow= ϕ_1	LIBLINEAR	45.22 (1.17)	46.02 (0.58)	89.68 (0.93)	51.64 (1.02)
ϕ_2	LIBLINEAR	42.31 (1.43)	48.70 (0.61)	101.97 (1.09)	54.79 (1.24)
ϕ_2	PWLSGD	238.98 (2.49)	49.89 (0.45)	291.30 (1.98)	55.35 (0.72)

Results: Daimler Chrysler pedestrians

Encoding	Training Algorithm	Training Time(s)	Accuracy(%)
identity	LIBLINEAR	1.89 (00.10)	72.98 (4.44)
identity	LIBSVM (int. kernel)	363.10 (27.85)	89.05 (1.42)
snow= ϕ_1	LIBLINEAR	2.98 (00.33)	85.71 (1.43)
ϕ_2	LIBLINEAR	1.86 (00.04)	88.80 (1.62)
ϕ_2	PWLSGD	3.18 (00.01)	89.25 (1.58)

Results: INRIA pedestrians

Encoding	Training Algorithm	Training Time (HOG)	Training Time (spHOG)
identity	LIBLINEAR	-	20.12s
identity	LIBSVM (lin. kernel)	>180 min	140 min
identity	LIBSVM (int. kernel)	>180 min	148 min
snow= ϕ_1	LIBLINEAR	35.52s	121.81s
ϕ_2	LIBLINEAR	22.45s	26.76s
ϕ_2	PWLSGD	99.85s	76.12s



► 100 times faster training than CVPR08.