

A FAST METHOD FOR TRAINING SUPPORT VECTOR MACHINES WITH A VERY LARGE SET OF LINEAR FEATURES

Jochen Maydt and Rainer Lienhart

Intel Labs, Intel Corporation, Santa Clara, CA 95052, USA

Rainer.Lienhart@intel.com

ABSTRACT

Current systems for object detection often use support vector machines (SVMs) as the basic classification algorithm. A rather common case is to compute a small set of linear features and then train the classifier on these features. We present a fast method to train and evaluate SVMs with many linear features and show results for face detection using a set of 210,400 features. The resulting classifier is both more accurate and faster than a classifier trained on raw pixel features, which total up only to 576 features in our case.

1. INTRODUCTION

Any classification algorithm highly depends on the type and quality of the feature set. A feature set should ideally reduce intra-class variance and still be highly discriminative. It is also desirable to use a rather small set of features to avoid the curse of dimensionality and to speed up training and classification.

Due to their simplicity, it is quite common to use linear features as the input to a classifier. There is a variety of powerful analysis methods such as principal component analysis (PCA), fisher discriminant analysis, and fourier transforms. Wavelets, Sobel-gradients [1] and haar-like features [2] also represent popular choices in the domain of object detection and are linear, too.

SVM's strong theoretical foundation and good generalization performance make them suitable for a large variety of classification problems, not only for object detection.

Contribution: We present a fast and novel method to speed up training and evaluation of a classifier with a very large set of linear features. A pre-computation step and a redefinition of the kernel function handle linear feature evaluation implicitly and thus result in a run-time complexity as if no linear features were evaluated at all. We then show results for face detection. A classifier trained with 210,400 linear features has a support vector count and run-time that is 50% lower than a classifier trained on raw pixel inputs (only $24 \times 24 = 576$ in our case) and achieves a comparable classification performance.

More interestingly, this method might also introduce a new class of problem-specific kernels that can improve generalization performance even further.

1.1. Related Work

It is quite common to use linear features for classification, not only for object detection systems. Our feature set is based on the work of P. Viola et al. [2]. They introduce an over-complete set of 45,396 haar-like features for face detection and present a fast evaluation scheme for these features. They use a method based on AdaBoost to pick only a small subset of features.

T. Serre et al. [1] present a face detection system based on SVMs and PCA features of 19×19 image windows. They also select only a small subset of features to speed up classification.

C. Papageorgiou et al. [3] developed a system to detect pedestrians in still images that uses SVMs and 1,326 wavelet features. However, as they only use absolute values of features, the resulting feature set is not linear.

2. FACE DETECTION SYSTEM

We gathered a training set of 2,162 faces and normalized them so that eyes and mouth were roughly at the same position for all faces [7]. This set was then randomly split into 1,652 faces for training and 510 faces for validation. Each face pattern was used to generate several training examples of size 24×24 by randomly mirroring, rotating between $\pm 10^\circ$, scaling by a factor between 0.9 and 1.1, and translating up to half a pixel. This resulted in a training set of 16,520 and a validation set of 10,200 faces. A contrast stretching operation was then used to saturate roughly 5% of the pixels and to achieve some basic lighting correction [8].

Negative training examples were generated from a set of 9,805 images that did not contain faces and then were processed by the same lighting correction method as the positive class. We use 5,000 negative examples for training and a different set of $2 \cdot 10^6$ negative examples for validation.

It is important to realize that lighting correction is a non-linear operation, but feature evaluation occurs after this step and is still linear.

2.1. Haar-like Feature

We use a similar, but even larger set of haar-like features than [2]. The feature prototypes (see figure 1) are scaled independently in x and y direction by integer factors up to a maximum extend of 24×24 . Each resulting feature is then translated to every possible position in a 24×24

window. Feature prototype (a) for example generates 43,200 features. Our complete feature set contains 210,400 features.

Evaluating a feature is rather simple. Let S_w be the sum of pixels corresponding to the white area and A_w be the number of these pixels. Similarly define S_b and A_b for the black area and set $S_0 = S_w + S_b$ and $A_0 = A_w + A_b$. A feature's value is then $f = w_0 S_0 - w_b S_b$ with

$$w_0 = 0.5 \sqrt{\frac{A_b}{A_w A_0}} \quad \text{and} \quad w_b = 0.5 \sqrt{\frac{A_0}{A_w A_b}}.$$

Note that pixels corresponding to the black area are effectively weighted by $(w_0 - w_b)$ as A_0 covers them, too. The weights w_0 and w_b ensure that (see [8] for details):

- a constant added to every pixel does not change f .
- roughly 95% of feature values satisfy $-1 < f < +1$ for images containing random Gaussian noise.

These weights were also chosen for rather practical reasons. First, SVMs require all input features to lie roughly in the same range, e.g. $[-1, +1]$, so that convergence of the QP optimization is assured. Second, it turns out that these weights lead to a good classification performance and speed. Two other choices that maintain the same ratio w_0/w_b , but scale feature values differently are also studied:

$$f' = \sqrt{\frac{A_0}{A_w A_b}} f, \quad f'' = \sqrt{\frac{A_w A_b}{A_0}} f$$

Remember that $A_w \sim A_0$ and $A_b \sim A_0$, hence f' uses less weight for features with a larger support A_0 than f . f'' on the other hand weights larger features even stronger. In effect it compares pixel sums as $f'' = 0.5 * S_0 A_b / A_0 - 0.5 * S_b$.

To speed up feature evaluation for comparison with our method we also implemented a table-driven approach introduced by P. Viola et al. [2]. A feature evaluation then only needs 6 to 9 table lookups and does not depend on the spatial extent of a feature.

2.2. Support Vector Machines

SVMs show good generalization performance even for high dimensional input data and small training sets. This makes them a method of choice for many binary classification tasks. However, without recent extensions such as reduced set methods their high computational demand still poses a major bottleneck, particularly for object detection. A more detailed discussion of the theory and applications of SVMs can be found in [4].

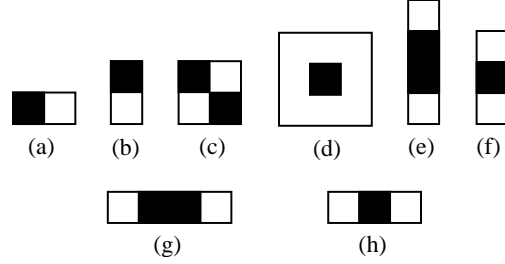


Figure 1 Each feature prototype is scaled in x and y direction by integer factors and then translated to every pixel position of a training example. Feature prototypes (a)-(c), (f) and (h) were introduced in [2].

In short, SVMs solve the following quadratic program:

$$\max_{\alpha_i} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

subject to $\sum_{i=1}^n y_i \alpha_i = 0, \quad 0 \leq \alpha_i \quad \forall i$

where n is the number of training examples, $\mathbf{x}_i \in \mathbf{R}^k$ is training example i and $y_i \in \{-1, +1\}$ is the class of \mathbf{x}_i . Other SVM formulations, e.g. with an L_1 -norm error penalty C , are transparent to the method we present. Common kernel functions $K(\mathbf{x}_i, \mathbf{x}_j)$ are the linear kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$, polynomial kernels $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^d$ of degree d , sigmoid kernels $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\mathbf{x}_i^T \mathbf{x}_j + c)$ and RBF kernels $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / \sigma^2)$ where $\|\mathbf{x}\|^2 = \mathbf{x}^T \mathbf{x}$.

We use a polynomial kernel of degree 2 for face detection as it was successfully used in other publications (see e.g. [1]) and performs very well for object detection.

The predicted class of an example $\mathbf{x} \in \mathbf{R}^k$ is then

$$\text{class}(\mathbf{x}) = \text{sign} \left[\left(\sum_{i=1}^n y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) \right) + b \right] \quad (1)$$

where α_i is the optimal solution of the maximization problem.

3. LINEAR FEATURES FOR SVMs

A linear feature evaluation can be written as $\mathbf{a}^T \mathbf{x}_i$ where $\mathbf{x}_i \in \mathbf{R}^k$ is training example i and $\mathbf{a} \in \mathbf{R}^k$. In our case \mathbf{a} simply represents a rasterized version of a haar-like feature, i.e. $a_i = w_0$ for all white pixels i , $a_i = w_0 - w_b$ for all black pixels i and $a_i = 0$ otherwise. A feature vector with m linear features is then calculated by $\mathbf{A} \mathbf{x}$ where $\mathbf{A} \in \mathbf{R}^{m \times k}$. Note that for some linear features faster computation schemes are possible. Keep also in mind that k is only $24 \times 24 = 576$ in our example, while m is 210,400.

	<i>Training</i>				<i>Classification</i>	
	Memory complexity	Pre-computation Step	Kernel evaluation	Training time	Memory complexity	Classifier evaluation
No Caching	$2m+nk$ 45 MB	-	$2c_f+c_K(m)$ 101,540 μ s	1,846 min	kN_s+2m 7.6 MB	$c_f+N_s(c_f+c_K(m))$ 51.6 s [†]
Caching	nm 16,052 MB	nc_f 500 s [†]	$c_K(m)$ 1,540 μ s	34 min [†]	$m(N_s+1)$ 1,607 MB	$c_f+N_sc_K(m)$ 1.54 s [†]
Our method	$nk+k^2$ 46 MB [‡]	$(m+n/2)k^2$ 50 s	$c_K(k)$ 5 μ s	0.883 min	kN_s 4.4 MB	$N_sc_K(k)$ 0.005 s [†]

[†] estimates based on performance numbers (see below)

[‡] we need temporarily about 300 MB to compute $\mathbf{B}=\mathbf{A}^T\mathbf{A}$, but this is transparent to SVM training

Figure 2 Comparison of memory and computational complexity for $n = 10,000$ training examples, $k = 576$ input pixels, $m = 210,400$ linear features and a resulting classifier with $N_s = 1,000$ support vectors. c_f is the running time for one feature evaluation, and $c_K(x)$ for a dot product of dimension x . We used a Pentium[®] 4 with 2 GHz and measured $c_f \approx 24\mu$ s with the optimized evaluation scheme described in [2] and $c_K(k) \approx 5\mu$ s, $c_K(m) \approx 1540\mu$ s using the platform-optimized math-kernel library from Intel [6].

There are two simple methods to train and evaluate a SVM using linear features. One method caches all feature vectors $\mathbf{z}_i = \mathbf{A}\mathbf{x}_i$, i.e. it pre-computes \mathbf{z}_i and then uses these vectors to calculate kernel elements $K(\mathbf{z}_i, \mathbf{z}_j)$. Evaluation of a classifier then simply transforms an input pattern \mathbf{x} to $\mathbf{z}=\mathbf{A}\mathbf{x}$ and uses $K(\mathbf{z}, \mathbf{z}_i)$ in (1). Unfortunately, even for a moderate number of features it is usually not possible to store all vectors \mathbf{z}_i in memory, neither for training nor for evaluation. A single feature vector in our case uses more than 800 KB (assuming single precision) and training sets with $n > 1,000$ are clearly impossible.

To conserve memory we can also compute $\mathbf{z}_i = \mathbf{A}\mathbf{x}_i$ each time a kernel element $K(\mathbf{z}_i, \mathbf{z}_j)$ is accessed and only store the original training examples \mathbf{x}_i . Evaluating a classifier then computes $\mathbf{z} = \mathbf{A}\mathbf{x}$ and $\mathbf{z}_i = \mathbf{A}\mathbf{x}_i$ for each support vector i . This method is computationally very expensive because training a SVM needs many evaluations of the kernel function. Even with a kernel cache, we need far more than 10^6 kernel evaluations to train a classifier on our training set. A fast scheme for feature evaluation, e.g. [2] for haar-like features, improves performance considerably, but still results in our case in a running time of 1,846 minutes for 10^6 kernel evaluations.

3.1. Speeding up Training

For large feature sets, it is possible to do better than these two methods. A linear kernel evaluation is

$$K(\mathbf{z}_i, \mathbf{z}_j) = \mathbf{z}_i^T \mathbf{z}_j = \mathbf{x}_i^T \mathbf{A}^T \mathbf{A} \mathbf{x}_j = \mathbf{x}_i^T \mathbf{B} \mathbf{x}_j$$

where $\mathbf{B} = \mathbf{A}^T \mathbf{A}$ is symmetric and $\mathbf{B} \in \mathbf{R}^{k \times k}$. A Cholesky factorization of \mathbf{B} results in $\mathbf{U}^T \mathbf{U} = \mathbf{B}$ where $\mathbf{U} \in \mathbf{R}^{k \times k}$ is an upper triangular matrix. If we train a SVM on $\mathbf{x}_i'' = \mathbf{U}\mathbf{x}_i$ instead of $\mathbf{z}_i = \mathbf{A}\mathbf{x}_i$ the results of all kernel evaluations remain unchanged and the solution α_i is identical. However, there are several benefits of using $\mathbf{x}_i'' \in \mathbf{R}^k$ instead of $\mathbf{z}_i \in \mathbf{R}^m$:

- evaluating a feature vector $\mathbf{z}_i = \mathbf{A}\mathbf{x}_i$ is not necessary.
- \mathbf{x}_i'' can usually be stored in memory as it is just as large as the original training data \mathbf{x}_i .
- for over-complete feature sets, i.e. $m > k$, the dot product $\mathbf{x}_i''^T \mathbf{x}_j''$ is of lesser complexity than $\mathbf{z}_i^T \mathbf{z}_j$.

Polynomial and sigmoid kernels also use the dot product $\mathbf{z}_i^T \mathbf{z}_j$ internally, hence we can substitute $K(\mathbf{x}_i'', \mathbf{x}_j'')$ with $K(\mathbf{z}_i, \mathbf{z}_j)$ in this case, too.

Only RBF kernels differ significantly. A simple reformulation of $\|\mathbf{z}_i - \mathbf{z}_j\|^2$ helps:

$$\|\mathbf{z}_i - \mathbf{z}_j\|^2 = \|\mathbf{A}(\mathbf{x}_i - \mathbf{x}_j)\|^2 = (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{A}^T \mathbf{A} (\mathbf{x}_i - \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{U}^T \mathbf{U} \mathbf{x}_i - 2\mathbf{x}_i^T \mathbf{U}^T \mathbf{U} \mathbf{x}_j + \mathbf{x}_j^T \mathbf{U}^T \mathbf{U} \mathbf{x}_j = \|\mathbf{U}\mathbf{x}_i - \mathbf{U}\mathbf{x}_j\|^2$$

and substituting $K(\mathbf{x}_i'', \mathbf{x}_j'')$ with $K(\mathbf{z}_i, \mathbf{z}_j)$ works here, too. As mentioned before the optimal solution is still the same, thus we get a classifier based on the original linear features if we use $K(\mathbf{z}, \mathbf{z}_i)$ in (1) instead of $K(\mathbf{U}\mathbf{x}, \mathbf{x}_i'')$. This might be of particular interest for feature selection based on SVMs with linear kernels (see e.g. [1] or [5]).

3.2. Implementation Issues

There are several difficulties for an implementation of this method. \mathbf{A} might be too large to fit into memory (e.g., \mathbf{A} is $210,400 \times 576$ in our example, while \mathbf{B} is only 576×576). However, we can use a simple blocking scheme and split \mathbf{A} into smaller matrices $\mathbf{A}_1, \dots, \mathbf{A}_p$ with $\mathbf{A}^T = [\mathbf{A}_1^T, \dots, \mathbf{A}_p^T]$. It follows that:

$$\mathbf{A}^T \mathbf{A} = [\mathbf{A}_1^T, \dots, \mathbf{A}_p^T] [\mathbf{A}_1, \dots, \mathbf{A}_p]^T = \mathbf{A}_1^T \mathbf{A}_1 + \dots + \mathbf{A}_p^T \mathbf{A}_p$$

Hence, we can compute \mathbf{B} incrementally and only have to fit \mathbf{B} and one of the smaller matrices \mathbf{A}_i into memory for each step.

A bigger concern is numerical stability. We encountered relative errors of 30% and more for the values of $K(\mathbf{x}_i'', \mathbf{x}_j'')$ if we used single precision for \mathbf{A} and \mathbf{B} . Double

precision provides enough significant digits for our case and was more accurate than $K(\mathbf{z}_i, \mathbf{z}_j)$ using single precision.

The Cholesky factorization $\mathbf{U}^T\mathbf{U} = \mathbf{B}$ might also introduce some numerical inaccuracy. It is possible to avoid it completely with a low additional memory overhead. If we compute $\mathbf{x}_i' = \mathbf{B}\mathbf{x}_i$, $\mathbf{x}_i' \in \mathbf{R}^k$ and keep \mathbf{x}_i in memory, too, we can express every kernel function without referring to \mathbf{U} . More exactly $K(\mathbf{z}_i, \mathbf{z}_j) = \mathbf{x}_i'^T \mathbf{x}_j'$ for linear kernels and a similar result follows for polynomial and sigmoid kernels. For RBF kernels we also store $s_i = \mathbf{x}_i'^T \mathbf{B}\mathbf{x}_i$ and express a kernel evaluation as $K(\mathbf{z}_i, \mathbf{z}_j) = \exp(s_i - 2\mathbf{x}_i'^T \mathbf{x}_j' + s_j)$.

3.3. Speeding up Classification

Classification can use a similar technique to speed up feature calculation. For linear, polynomial and sigmoid kernels, we use $K(\mathbf{z}, \mathbf{z}_j) = \mathbf{x}^T \mathbf{x}_j'$ where $\mathbf{z} = \mathbf{A}\mathbf{x}$ and do not have to evaluate linear features at all. RBF kernels can be evaluated by $K(\mathbf{z}, \mathbf{z}_j) = \exp(s_i - 2\mathbf{x}^T \mathbf{x}_j' + \mathbf{x}^T \mathbf{B}\mathbf{x})$, thus we still have to compute $\mathbf{x}^T \mathbf{B}\mathbf{x}$.

It is important to realize that computational complexity of classification mainly depends on three factors: feature evaluation, the dot product inside the kernel function and the number of support vectors. Clearly, our evaluation scheme does not affect the number of support vectors. For $m \gg k$ classification is significantly faster, because dot product and feature evaluation are of lower complexity. For $m \approx k$ only feature evaluation is faster. This effect is almost negligible for high support vector counts. See figure 2 for a more detailed comparison of complexity for training and evaluation.

4. RESULTS FOR FACE DETECTION

We trained a polynomial classifier of degree 2 for pixel features and for our linear feature set with different feature weights. Classifiers that use our linear feature set achieve a comparable classification performance as pixel features, particularly for detection rates between 80% and 95% (see figure 3). Remember that classification for a polynomial kernel with linear features has the same complexity as for pixel features. As a result, classifiers using linear features are roughly 50% faster due to a lower support vector count.

5. CONCLUSIONS AND FUTURE WORK

We presented a method that makes training and evaluation of SVMs with very large linear feature sets possible. Particularly feature selection approaches such as [5] [1] have to train a classifier with all features first. In combination with our method, it is now possible to study feature selection for SVMs with more than 200,000 features. Together with a cascaded evaluation scheme as in [2] this can speed up classification tremendously.

We also show that different feature weights result in a different generalization performance. Good feature weights might be an interesting alternative to a good

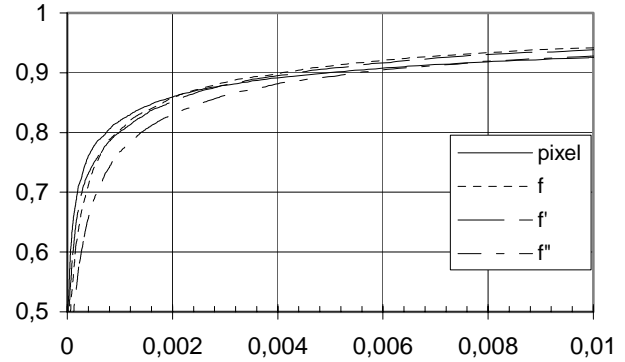


Figure 3 Comparison of ROC curves for pixel features, and different weights of our feature set. Pixel features resulted in a classifier with 2,270 support vectors, f with 1,005, f'' with 1,267 and f''' with 1,011 support vectors.

feature subset, particularly as classification speed for our method does not depend on the number of features.

Finally, the analysis of matrix \mathbf{B} and its structure might lead to new ways to improve generalization performance. In our case, \mathbf{B} has a banded structure that encodes neighboring information of pixels and thus contains domain specific knowledge. Optimizing the elements of \mathbf{B} with an approach as in [5] or an analytical method to integrate domain specific knowledge might prove useful.

6. REFERENCES

- [1] T. Serre, B. Heisele, S. Mukherjee, and T. Poggio, "Feature Selection for Face Detection", *A.I. Memo No. 1697, CBCL Paper No. 192*, M.I.T., Cambridge, MA, September 2000
- [2] P. Viola, and M. Jones, "Robust Real-Time Object Detection", *Cambridge Research Laboratory Technical Report CRL 2001/01*, Compaq CRL, February 2001
- [3] C. Papageorgiou, and T. Poggio, "Trainable Pedestrian Detection", *Proceedings of International Conference on Image Processing (ICIP '99)*, Kobe, Japan, October 1999
- [4] C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition", *Data Mining and Knowledge Discovery 2*, p. 121-167, 1998
- [5] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing Multiple Parameters for Support Vector Machines", *Machine Learning*, 2000
- [6] <http://developer.intel.com/software/products/mkl/>
- [7] H. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection", *IEEE Patt. Anal. Mach. Intell.*, Vol. 20, pp. 22-38, 1998.
- [8] Jochen Maydt, "People Detection in Digital Images: A Component-Based Approach to Object Detection", Master thesis, University of Mannheim and Intel Corporation, Dec. 2001.