

Face Detection with Support Vector Machines and a Very Large Set of Linear Features

Jochen Maydt

Max-Planck-Institute for Computer Science

Pestalozzistr. 4
69469 Weinheim
+49 6201 57046

Jochen.Maydt@gmx.de

Rainer Lienhart

Intel Labs

2200 Mission College Blvd.
Santa Clara, CA 95052, USA
+1 (408) 765-3450

Rainer.Lienhart@intel.com

ABSTRACT

This paper presents a fast and novel method to speed up training and evaluation of support vector machine (SVM) classifiers 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 train a classifier for face detection on a set of 210,400 linear features. The resulting classifier has a support vector count and running time that is 50% lower than a classifier trained on raw pixel features, but still maintains a comparable classification performance. A recent feature weighting approach is adapted for large linear feature sets and used to improve the classification performance of both classifiers even further. Again, the classifier based on our linear feature set outperforms a comparable classifier based on pixel inputs.

Categories and Subject Descriptors

I.4.8 [Image Processing and Computer Vision]: Scene Analysis – *object recognition*.

General Terms

Algorithms, Performance.

Keywords

Face Detection, Support Vector Machines, Haar-like Features, Feature Weighting.

1. INTRODUCTION

Object detection and particularly face detection has already been studied for many years. Detection systems for face detection in digital still images, i.e. without any motion or stereo information, already approach a high level of maturity. Classification

performance leaves only little space for improvement (see e.g. [9] [10]), but classification speed still needs to be improved.

Many current object detection systems rely on general-purpose classification algorithms instead of handcrafted models. Support vector machines (SVMs) for instance have a strong theoretical foundation and show excellent results with respect to generalization performance for a large number of applications. Unfortunately, SVM classifiers usually have a relatively low classification speed.

Another important part of a classification system is the feature set as it determines type and quality of the classifier's input. A feature set should ideally reduce intra-class variance and still be highly discriminative. Due to their simplicity, linear features are quite common 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 [5] and haar-like features [12] also represent popular choices in the domain of object detection and are linear, too.

Combining SVMs with linear features is quite a common case. B. Heisele et al. [5] train several SVMs for face detection. They compare the performance of pixel, gradient and wavelet features and conclude that pixel features work best for their system. They also use both, pixel and gradient/wavelet features, but results are no better than for pixel features alone. To improve the classification speed of their system they use a feature selection approach and train a SVM with only a few PCA features. T. Serre et al. [11] also present a face detection system based on SVMs and PCA features on 19x19 image windows. Again, they select only a small subset of features to speed up classification. C. Papageorgiou et al. [8] 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. P. Viola et al. [12] present a very different approach to face detection. Based on a set of 45,396 haar-like features and a fast evaluation scheme they construct a cascade of more and more complex classifiers. A face is only detected if every classifier in the cascade detects a face. Simple classifiers at the beginning of the cascade only use a few features and are very fast to evaluate. These classifiers can effectively reject already a majority of background windows. It is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM Multimedia '02, December 1-6, 2002, Juan Les Pins, France.

Copyright 2000 ACM 1-58113-000-0/00/0000...\$5.00.

interesting that their feature selection strategy is very specific to the classification algorithm (AdaBoost) that they use.

We introduce a method that facilitates training and evaluation of SVMs with large linear feature sets. It is very effective for over-complete feature sets where the input data has a much smaller dimensionality than the resulting feature vectors. We use an extended version of the haar-like feature set presented in [12] and train a classifier that is 50% faster than a classifier based on pixel features, but still has a comparable classification performance. Moreover, we believe that our method provides a good foundation for a fast object detection framework as in [12]. It might also introduce a new class of problem-specific kernels that can improve generalization performance considerably.

We then show that not only the feature set, but also the weight (or normalization) of each feature plays an important role. A recent feature weighting approach [3] is used to improve generalization performance even further. Again, our haar-like feature set outperforms pixel features with respect to speed.

2. A 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 [9]. 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 24x24 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 in each direction. This resulted in a training set of 16,520 and a validation set of 10,200 faces. Negative training examples were generated from a set of 9,805 images that do not contain faces. Our validation set for the negative class consists of $2 \cdot 10^6$ negative examples that were not used during training.

2.1 Lighting Correction

To lower the effect of different lighting conditions we use a simple contrast stretching operation on every 24x24 window:

$$\Gamma(x, y) = \text{saturate} \left[\frac{0.5}{\sigma} (I(x, y) - \mu) \right]$$

where (σ, μ) are variance and average of all pixels of a 24x24 window and $I(x, y)$ and $\Gamma(x, y)$ is the intensity of a pixel at position (x, y) before and after contrast stretching respectively. This saturates roughly 5% of all pixels of a training example [7]. Remember that even though lighting correction is a non-linear operation in our case, feature evaluation occurs after this step and is still linear.

2.2 Haar-like Features

We use a similar, but even larger set of haar-like features than [12]. The idea is to evaluate feature prototypes (see Figure 1) at different scales and at various positions in a detection window so that a large number of specific features is created. As a result, it is possible to describe a face coarsely with only a few features. For example, a rather large feature (h) in the upper middle of a detection window could be an indicator for the eyes, as they are usually darker than the area in between.

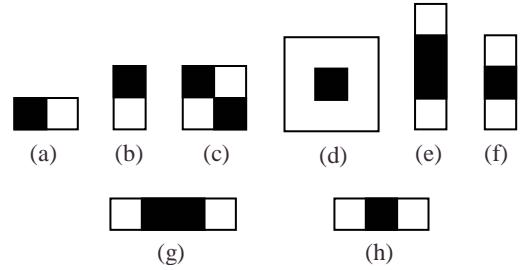


Figure 1. Each feature prototype is scaled in x and y direction by integer factors and then translated to every pixel position of a training example. Feature prototypes (a)-(c), (f) and (h) were introduced in [12] and are similar to a gradient operator.

Generating features from prototypes proceeds as follows. Feature prototypes are scaled independently in x and y direction by integer factors up to a maximum extent of 24x24. Each resulting feature is then translated to every possible position in a 24x24 window. Consider feature prototype (a) with its initial size of 2x1: for this size it generates a total of $(24-2) \cdot (24-1) = 506$ features. If we also take different sizes into account, prototype (a) generates 43,200 features. Our complete feature set contains 210,400 features.

2.2.1 Feature Evaluation

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

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

Note that pixels corresponding to the black area are effectively weighted by $(w_0 - w_b)$ as A_0 covers these pixels, too.

2.2.2 Normalizing Features

In short, these weights w_0 and w_b ensure that [7]:

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

If we think about the relation to haar features it might seem more intuitive to use simpler weights, e.g. $w_b = w_w = 1/A_0$. However, there exist features that have an unequal number of black and white pixels (e.g. feature (d)). Using weights with $w_b = w_w$ would result in a behavior which differs from that of haar features, i.e. the feature's value would no longer compare the average intensity of the black and white area, but would change if we increased every pixel by a constant. To satisfy condition a) we have to maintain the ratio w_b/w_0 of the weights presented above.

Now consider a single feature p and its feature value f_p . We can still choose a different scaling factor or weight s_p and use $s_p f_p$ as an input to a classifier without breaking condition a). One method to determine s_p evaluates feature p for all training examples and sets s_p so that $-1 < s_p f_p < +1$ [5]. However, an outlier

can effect s_p severely. Even worse, if the training set grows larger it is more and more likely that we observe every feature p at least once with its maximum value. Features with a large support usually have a weak response on natural images and consequently need larger weights s_p . Thus we think it is more appropriate to estimate the distribution of f_p and then set s_p so that $-1 < s_p f_p < +1$ for almost all f_p . To avoid evaluating features on the complete training set, we derived the weights w_b and w_0 analytically so that $-1 < f_p < +1$ for roughly 95% of the feature values.

In addition to $s_p=1$ we also study two other choices for s_p :

$$s_p' = \sqrt{\frac{A_0}{A_w A_b}} \quad \text{and} \quad s_p'' = \sqrt{\frac{A_w A_b}{A_0}}$$

and define $f''=s_p' f$ and $f'''=s_p'' f$. Remember that $A_w \sim A_0$ and $A_b \sim A_0$ for all scales of one feature prototype, because the relative size of black and white area do not change. It follows that $s_p'' \sim \sqrt{A_0}$ and f''' weights features with a larger support A_0 even stronger than f . More exactly, $f'''=0.5 * S_0 A_b / A_0 - 0.5 * S_b$, thus f''' directly compares pixel sums of the black and white area. f'' on the other hand results in less weight for large features and compares the average intensities instead.

2.2.3 Fast Feature Evaluation

We also implemented a table-driven approach introduced by P. Viola et al. [12] that can speed up feature evaluation dramatically. A feature evaluation then only needs 6 to 9 table lookups and does not depend on the spatial extent of a feature.

2.3 Support Vector Machines

SVMs show good generalization performance even for high dimensional input data and small training sets. This makes them well suited for many binary classification tasks. A more detailed discussion of the theory and applications of SVMs can be found in [1].

In short, a hard margin SVM solves the following quadratic program (QP1):

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

subject to $\sum_{i=1}^n y_i \alpha_i = 0$, $0 \leq \alpha_i$ for all 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 . 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 $\|\mathbf{x}\|^2 = \mathbf{x}^T \mathbf{x}$. They all can be rewritten so that they use the dot product $\mathbf{x}_i^T \mathbf{x}_j$ as the most expensive operation. Each kernel function results in a different type of decision boundary. A polynomial kernel for example uses a polynomial curve of degree d to represent the decision boundary, while the linear kernel simply uses a plane in the input space.

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

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

where α_i is the optimal solution of (QP1). Training examples \mathbf{x}_i that correspond to an $\alpha_i > 0$ are called support vectors (SVs). Non-support vectors are not necessary for classification as $\alpha_i = 0$.

There exist several modifications for hard margin SVMs, such as the L_1 - and L_2 -norm formulations. Both of them introduce an error term to the goal function to allow training errors (i.e. misclassified training examples). However, they are transparent to our method for speeding up training and evaluation of SVMs.

Our face detection system uses a hard margin SVM with a polynomial kernel of degree 2. This kernel has already been used in many other object detection systems [5] [11] [8] and gives very good results. We use LIBSVM [2] to train the SVM as it was simple to adapt to our special problem setting.

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 $\mathbf{A} \in \mathbf{R}^{m \times k}$. However, there exist much faster computation schemes for some linear transforms, such as wavelet or fourier transforms. Keep also in mind that k is only $24 * 24 = 576$ in our case, while $m = 210,400$.

3.1 Brute Force Methods

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 $K(\mathbf{z}_i, \mathbf{z}_j)$. Evaluating 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 smaller number of linear 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 and many feature evaluations. 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. as in [12] for haar-like features, can improve performance, but still results in a running time of 1,846 minutes for 10^6 kernel evaluations in our case.

Not only feature evaluation is a bottleneck. Both methods also suffer from an increased dimensionality of the dot product $\mathbf{z}_i^T \mathbf{z}_j$ for each kernel evaluation.

Table 1. 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 $c_f \approx 24\mu\text{s}$ with the table-driven evaluation scheme described in [2] and $c_K(k) \approx 5\mu\text{s}$, $c_K(m) \approx 1540\mu\text{s}$ using the platform-optimized math-kernel library from Intel [6].

	<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 μs	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 μs	34 min [†]	$m(N_s+1)$ 1,607 MB	$c_f+N_s c_K(m)$ 1.54 s [†]
Our method	$nk+k^2$ 46 MB [‡]	$(m+n/2)k^2$ 50 s	$c_K(k)$ 5 μs	0.883 min	kN_s 4.4 MB	$N_s c_K(k)$ 0.005 s [†]

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

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

3.2 Speeding up Training

For large feature sets, it is possible to do better than these two methods. We can write a linear kernel evaluation as

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

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

We can also substitute $\mathbf{K}(\mathbf{x}_i', \mathbf{x}_j')$ with $\mathbf{K}(\mathbf{z}_i, \mathbf{z}_j)$ for polynomial and sigmoid kernels, because they use the dot $\mathbf{z}_i^T \mathbf{z}_j$ internally. Only RBF kernels differ significantly. A simple reformulation of $\|\mathbf{z}_i - \mathbf{z}_j\|^2$ helps:

$$\begin{aligned} \|\mathbf{z}_i - \mathbf{z}_j\|^2 &= \|\mathbf{A}(\mathbf{x}_i - \mathbf{x}_j)\|^2 = (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{A}^T \mathbf{A} (\mathbf{x}_i - \mathbf{x}_j) = \\ &(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{U}^T \mathbf{U} (\mathbf{x}_i - \mathbf{x}_j) = \|\mathbf{U}(\mathbf{x}_i - \mathbf{x}_j)\|^2 = \|\mathbf{x}_i' - \mathbf{x}_j'\|^2 \end{aligned}$$

and substitution of $\mathbf{K}(\mathbf{x}_i', \mathbf{x}_j')$ with $\mathbf{K}(\mathbf{z}_i, \mathbf{z}_j)$ 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 $\mathbf{K}(\mathbf{z}_i, \mathbf{z}_j)$ in (1) instead of $\mathbf{K}(\mathbf{U}\mathbf{x}_i, \mathbf{U}\mathbf{x}_j)$.

3.3 Implementation Issues

There are several difficulties for an implementation of this method. \mathbf{A} might be too large to fit into memory. In our case, \mathbf{A} is $210,400 \times 576$, 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:

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

and we can compute \mathbf{B} incrementally. We only have to keep \mathbf{B} and one of the smaller matrices \mathbf{A}_i in memory for each step.

A bigger concern is numerical stability. We encountered relative errors of 30% and more for the values of $\mathbf{K}(\mathbf{x}_i', \mathbf{x}_j')$ 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 $\mathbf{K}(\mathbf{z}_i, \mathbf{z}_j)$ 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 also keep \mathbf{x}_i in memory, we can express every kernel function without referring to \mathbf{U} . More exactly $\mathbf{z}_i^T \mathbf{z}_j = \mathbf{x}_i^T \mathbf{x}_j'$ and a linear kernel evaluation is $\mathbf{K}(\mathbf{z}_i, \mathbf{z}_j) = \mathbf{x}_i^T \mathbf{x}_j'$. A similar result follows for polynomial and sigmoid kernels. For RBF kernels we also have to store $v_i = \mathbf{x}_i^T \mathbf{B} \mathbf{x}_i$ and express a kernel evaluation as $\mathbf{K}(\mathbf{z}_i, \mathbf{z}_j) = \exp(v_i - 2\mathbf{x}_i^T \mathbf{x}_j' + v_j)$.

3.4 Speeding up Classification

We can use a similar technique to speed up classification of an unknown example \mathbf{x} and its feature vector $\mathbf{z}=\mathbf{A}\mathbf{x}$. As $\mathbf{z}'=\mathbf{x}^T \mathbf{x}_j'$ it is not necessary to compute $\mathbf{z}=\mathbf{A}\mathbf{x}$ and we can avoid feature evaluation completely for linear, polynomial and sigmoid kernels. For RBF kernels we still have to compute $\mathbf{x}^T \mathbf{B} \mathbf{x}$ as $\mathbf{K}(\mathbf{z}, \mathbf{z}_j) = \exp(v_j - 2\mathbf{x}^T \mathbf{x}_j' + \mathbf{x}^T \mathbf{B} \mathbf{x})$.

Computational complexity of classification depends on three factors:

- the number of support vectors
- speed of feature evaluation
- the dot product $\mathbf{z}^T \mathbf{z}_j$ or $\mathbf{x}^T \mathbf{x}_j'$ inside the kernel function

Clearly, the evaluation scheme itself does not change the number of support vectors. However, feature evaluation is not necessary; only for RBF kernels we have to compute $\mathbf{x}^T \mathbf{B} \mathbf{x}$. The dot product usually has the biggest impact on speed and leads to a significant

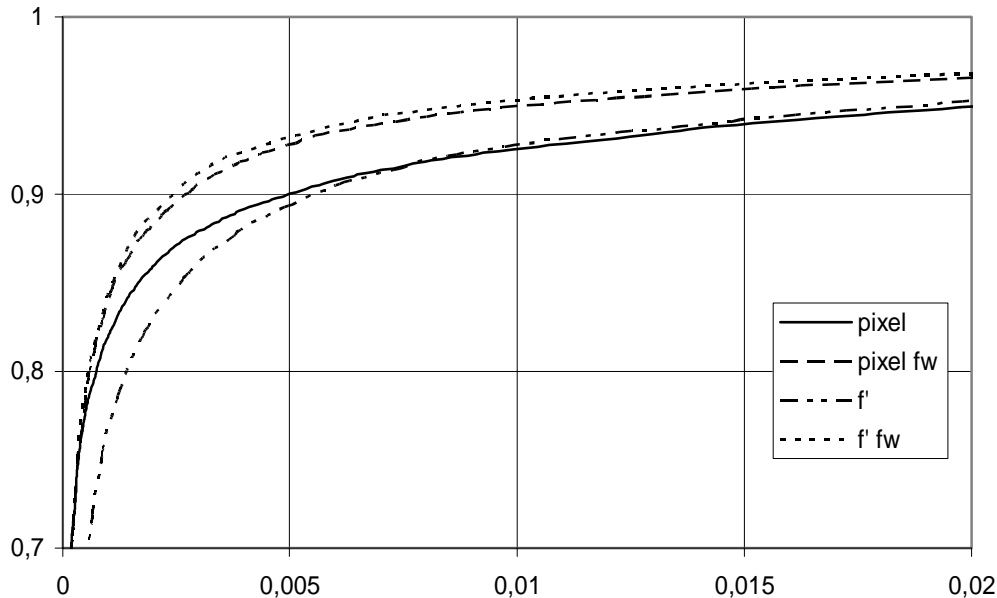


Figure 2. ROC curves for pixel features and different weights for our feature set. Pixel features resulted in a classifier with 2,270 support vectors, f with 1,005, f' with 1,011 and f'' with 1,267 support vectors.

speedup for $m \gg k$. For $m = k$ only feature evaluation is faster, but this effect is almost negligible for high support vector counts.

Table 1 summarizes memory and computational complexity of the different algorithms for training and evaluation.

3.5 Results

We trained a SVM with a polynomial kernel of degree 2 for pixel features and for our linear feature set with different feature weights. All positive and 5,000 negative training examples were used. The points of ROC curves were generated by varying b of equation (1) from $-\infty$ to $+\infty$ and classifying the validation set again.

Classifiers that use our linear feature set achieve a comparable classification performance as pixel features, particularly for detection rates of 80% and above (see Figure 2). The value of b found by the SVM actually results in a detection rate between 95% and 97% for all feature sets. 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.

To get a better classifier and a more representative set of negative training examples, we used a bootstrapping procedure. First, we trained a classifier with 10,000 negative and all positive training examples. Then, we replaced all non-support vectors corresponding to negative training examples with misclassified negative training examples and trained the SVM again. We repeated this step once more to get our final classifier with haar-like features (4,768 SVs) and pixel features (5,039 SVs) and applied it to the CMU test set (only Test A + Test B so far, results for the full test set are in work). The classifier with haar-like features has a detection rate of 81% and produces 90

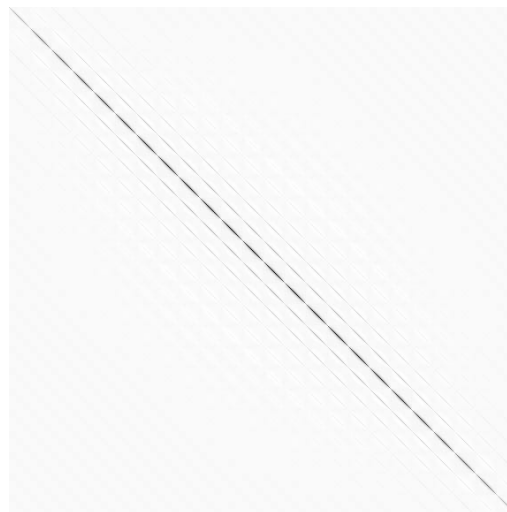


Figure 3. Matrix \mathbf{B} of size 576x576 for feature weights $s_p=1$ with elements between $-4 \cdot 10^{-5}$ and $1.4 \cdot 10^{-3}$. Elements are negatively scaled so that the largest elements of \mathbf{B} are black and the smallest are white.

false alarms. The classifier based on pixel features detects 77% of the faces and has 61 false alarms.

The analysis of matrix \mathbf{B} reveals an interesting structure. It seems to be a key for understanding the improved generalization performance. In our case, \mathbf{B} has a banded structure where each band corresponds to pixels in the neighborhood of the diagonal (see Figure 3).

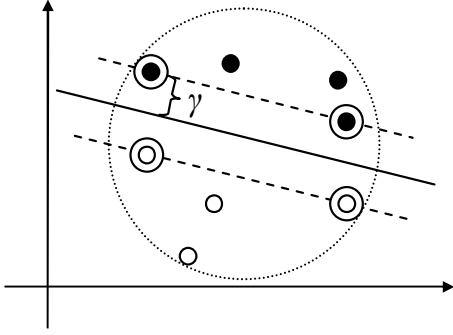


Figure 4. A linear SVM for separable data. Circled dots represent support vectors and $\gamma=1/(2W)$ is the margin. R is the radius of the circle containing all training examples.

4. FEATURE WEIGHTING

Feature weights can have a significant impact on classification performance and support vector count (see Figure 2). So far we have derived these weights s_p for each feature p analytically and even only considered $s_p=1$ for pixel features. However, there are methods to optimize s_p that can further improve generalization performance.

Chapelle et al. [3] perform a gradient descent to minimize a bound on the generalization error. More exactly, they compute the derivative of this bound with respect to each feature weight s_p and update these weights with a gradient step. As an effect, the classification performance slowly improves as they step along the gradient, at least if the bound has a minimum close to the real minimum of the generalization error. The R^2W^2 bound performs quite well for this purpose, even though it is a rather loose bound.

4.1 R^2W^2 Bound Minimization

The R^2W^2 bound is

$$\tau = \frac{1}{n} R^2 W^2$$

where n is the number of training examples. For a linear kernel with separable training data, R is the radius of the smallest sphere that contains all training examples and $\gamma=1/(2W)$ measures the smallest distance of a training example from the decision boundary (see Figure 4). A similar result follows for other kernel functions for the kernel feature space that they induce. To understand the R^2W^2 bound imagine that all points in Figure 4 move away from the decision boundary, but still stay inside the sphere, i.e. γ increases. Then it is more likely that other examples also lie further away from the decision boundary and hence the classifier probably will make fewer mistakes. Shrinking the radius R has a similar effect.

For our purposes, it is easier to represent the feature weights s_p as a part of the kernel function $K(\mathbf{D}\mathbf{x}_i, \mathbf{D}\mathbf{x}_j)$ where \mathbf{D} is a diagonal matrix with s_p as its elements. For convenience we will refer to $K(\mathbf{D}\mathbf{x}_i, \mathbf{D}\mathbf{x}_j)$ as K_{ij} . W^2 is then the objective function's value of (QP1), i.e.

Table 2. Different kernel functions and their derivative w.r.t. s_p . Kernel functions for haar-like features are very similar. They use \mathbf{h}_i instead of \mathbf{x}_i and $\mathbf{h}_i(p)$ instead of $\mathbf{x}_i(p)$, where $\mathbf{x}_i(p)$ is the p -th element of training example i and $\mathbf{h}_i(p)$ is the p -th element of the corresponding haar feature vector \mathbf{h}_i .

	$K_{ij} =$	$\frac{\partial K_{ij}}{\partial s_p} =$
linear	$\mathbf{x}_i^T \mathbf{D}^T \mathbf{D} \mathbf{x}_j$	$2s_p \mathbf{x}_i(p) \mathbf{x}_j(p)$
poly	$(\mathbf{x}_i^T \mathbf{D}^T \mathbf{D} \mathbf{x}_j + 1)^d$	$2s_p d \mathbf{x}_i(p) \mathbf{x}_j(p) * (\mathbf{x}_i^T \mathbf{D}^T \mathbf{D} \mathbf{x}_j + 1)^{d-1}$
RBF	$\exp\left(\frac{\mathbf{D}(\mathbf{x}_i - \mathbf{x}_j)^2}{\sigma^2}\right)$	$\frac{2s_p (\mathbf{x}_i(p) - \mathbf{x}_j(p))^2}{\sigma^2} \exp\left(\frac{\mathbf{D}(\mathbf{x}_i - \mathbf{x}_j)^2}{\sigma^2}\right)$

$$W^2 = \max_{\alpha_i} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j \alpha_i \alpha_j K_{ij}$$

subject to $\sum_{i=1}^n \alpha_i = 0$, $0 \leq \alpha_i$ for all i .

The radius R is defined in a similar way by (QP2):

$$R^2 = \max_{\beta_i} \sum_{i=1}^n \beta_i K_{ii} - \sum_{i=1}^n \sum_{j=1}^n \beta_i \beta_j K_{ij}$$

subject to $\sum_{i=1}^n \beta_i = 1$, $0 \leq \beta_i$ for all i .

Computing R^2 is fortunately even faster than computing W^2 and can be done with LIBSVM, too. In the following, we will refer to α_i^o and β_i^o as the solution of (QP1) and (QP2).

4.2 Gradient of the R^2W^2 Bound

Chapelle et al. proved that the derivative of τ is

$$\frac{\partial \tau}{\partial s_p} = \frac{1}{n} \left(R^2 \frac{\partial W^2}{\partial s_p} + W^2 \frac{\partial R^2}{\partial s_p} \right)$$

where

$$\frac{\partial W^2}{\partial s_p} = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i^o \alpha_j^o y_i y_j \frac{\partial K_{ij}}{\partial s_p} \quad (2)$$

$$\frac{\partial R^2}{\partial s_p} = \sum_{i=1}^n \beta_i^o \frac{\partial K_{ii}}{\partial s_p} - \sum_{i=1}^n \sum_{j=1}^n \beta_i^o \beta_j^o \frac{\partial K_{ij}}{\partial s_p} \quad (3)$$

Both (2) and (3) are very similar, thus we will only describe how to compute (2) in detail. Table 2 gives an overview of the derivatives of different kernel functions. We will limit ourselves to the polynomial kernel in the following. Linear and sigmoid kernels can be handled similarly.

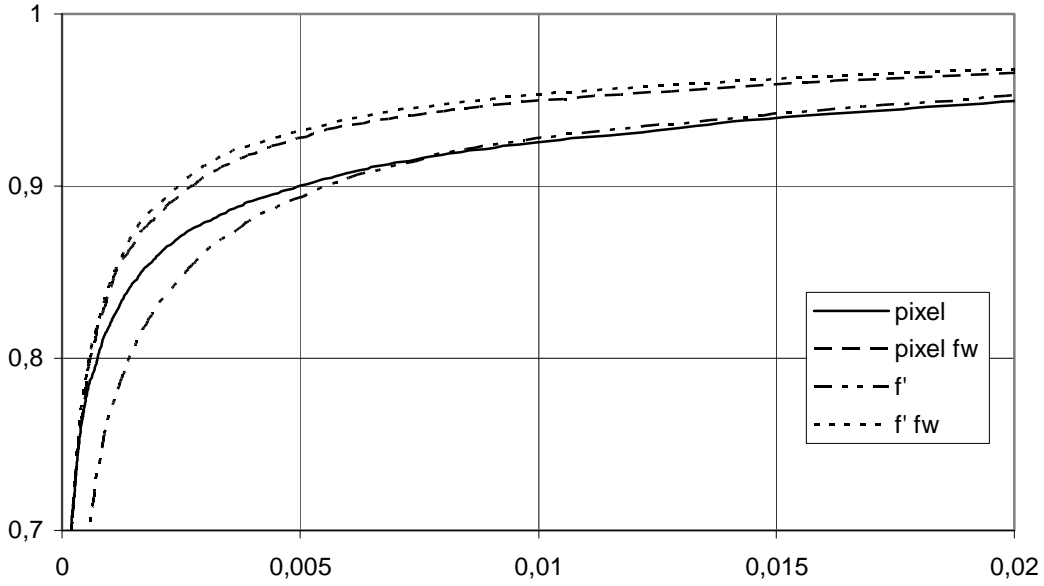


Figure 4. ROC curves for classifiers trained with and without feature weighting. Feature weighting has clearly a strong effect on generalization performance, particularly for our haar-like feature set. The f' classifier with feature weighting has 1,153 and the pixel classifier has 1,640 support vectors.

4.3 Gradient for Pixel Features

For pixel features it follows that

$$\frac{\partial W^2}{\partial s_p} = -s_p d \sum_{i=1}^n \sum_{j=1}^n \alpha_i^o \alpha_j^o y_i y_j \mathbf{x}_i(p) \mathbf{x}_j(p) (\mathbf{x}_i^T \mathbf{D}^T \mathbf{D} \mathbf{x}_j + 1)^{d-1}$$

for all p , where $\mathbf{x}_i(p)$ is the p -th pixel of training example \mathbf{x}_i .

We can use several simple optimizations to speed up the evaluation of this expression. Indices i and j that correspond to non-support vectors can be eliminated, because $\alpha_i^o=0$ or $\alpha_j^o=0$. The summation term is also symmetric w.r.t. i and j , i.e. its value does not change if i and j are exchanged. However, caching the outer derivative $(\mathbf{x}_i^T \mathbf{D}^T \mathbf{D} \mathbf{x}_j)^{d-1}$ of the kernel function has a much larger impact on performance. Kernel evaluations are very compute intense and do not depend on p .

Finally, it is also possible to rewrite this expression using matrix-vector notation. If we set $\mathbf{x}(p)=[\mathbf{x}_1(p), \dots, \mathbf{x}_n(p)]^T$ and define

$$\bar{K}_{ij} = \alpha_i^o \alpha_j^o y_i y_j (\mathbf{x}_i^T \mathbf{D}^T \mathbf{D} \mathbf{x}_j + 1)^{d-1}$$

as the elements of a symmetric matrix $\bar{\mathbf{K}}$, it follows that

$$\frac{\partial W^2}{\partial s_p} = -s_p d \mathbf{x}(p)^T \bar{\mathbf{K}} \mathbf{x}(p)$$

and a math library such as [6] can significantly speed up gradient construction. As we encountered numerical instabilities for high support vector counts ($N_s \geq 1,000$), we used row and column permutations to make the matrix-vector multiplication more stable.

4.4 Gradient for Haar-like Features

Consider the derivative of a polynomial kernel function for haar-like features

$$\frac{\partial K_{ij}}{\partial s_p} = 2s_p d \mathbf{h}_i(p) \mathbf{h}_j(p) (\mathbf{x}_i^T \mathbf{A}^T \mathbf{D}^T \mathbf{D} \mathbf{A} \mathbf{x}_j + 1)^{d-1}$$

where \mathbf{h}_i is the feature vector of \mathbf{x}_i and $\mathbf{h}_i(p)$ denotes its p -th element. Evaluating features is therefore an important part of gradient construction. The outer derivative $(\mathbf{x}_i^T \mathbf{A}^T \mathbf{D}^T \mathbf{D} \mathbf{A} \mathbf{x}_j + 1)^{d-1}$ can be computed with the method described in 3.2.

The gradient with respect to a feature weight s_p is

$$\frac{\partial W^2}{\partial s_p} = -s_p d \sum_{i=1}^n \sum_{j=1}^n \alpha_i^o \alpha_j^o y_i y_j \mathbf{h}_i(p) \mathbf{h}_j(p) (\mathbf{x}_i^T \mathbf{A}^T \mathbf{D}^T \mathbf{D} \mathbf{A} \mathbf{x}_j + 1)^{d-1}$$

If we define $\mathbf{h}(p)=[\mathbf{h}_1(p), \dots, \mathbf{h}_n(p)]^T$ and

$$\bar{K}_{ij} = \alpha_i^o \alpha_j^o y_i y_j (\mathbf{x}_i^T \mathbf{A}^T \mathbf{D}^T \mathbf{D} \mathbf{A} \mathbf{x}_j + 1)^{d-1}$$

we get a similar result as for pixel features, i.e.

$$\frac{\partial W^2}{\partial s_p} = -s_p d \mathbf{h}(p)^T \bar{\mathbf{K}} \mathbf{h}(p).$$

It is important to speed up gradient construction, because the gradient is computed many times during the course of a gradient descent algorithm. Particularly the sheer size of our feature set poses a problem, as gradient construction is $210,400/576 \approx 365$ times slower than for pixel features. An implementation for gradient construction that caches all kernel elements, uses the symmetry of the summation term and only loops over indices

corresponding to support vectors took 36 minutes for our linear feature set and 1,000 support vectors. Matrix-vector notation and Intel's Math Kernel Library [6] lowered the running time to 12 minutes.

4.5 Results

We stepped along the gradient for the classifiers of 3.5 until the R^2W^2 bound did not change anymore. Remember that we are performing a gradient descent in 210,400 dimensional space for our haar-like feature set. The probability to find only a local minimum is quite high and in fact we could observe different results for all initial feature weights s_p , s_p' and s_p'' . We obtained the best result for s_p' , i.e. feature weights that favor large features. ROC curves are given in Figure 4.

We initially hoped that a procedure as in [3], which iteratively performs feature weighting and then discards features with smaller weights, could be a substitute for feature selection. This failed probably due to the highly correlated nature of the features. Generalization performance decreased considerably as we discarded more and more features. A different feature selection strategy such as [4] [13] might be more successful.

5. CONCLUSION AND FUTURE WORK

We presented a method that makes training and evaluation of SVMs with very large linear feature sets possible. Feature selection and weighting approaches such as [3] [4] [11] [13] have to train a classifier with all features first and can benefit from this method. Hence, it is now possible to study feature selection and weighting for SVMs with more than 200,000 features.

We also introduce an algorithm to construct the gradient of the R^2W^2 bound very quickly, both for pixel and for linear feature sets. We then show that good feature weights result in a much better generalization performance. Good feature weights might be an interesting alternative to a good feature subset, as classification speed for our method does not depend on the number of features.

Of particular interest is the structure of \mathbf{B} that indirectly represents domain specific knowledge (i.e. in our case the notion of a pixel neighborhood). Optimizing the elements of \mathbf{B} directly instead of a set of feature weights, or analytically studying the effect of its structure might be of interest. Finally, our method might serve as a building block for an approach similar to [12] that has to select a few features for rapid object detection.

6. ACKNOWLEDGEMENTS

We would like to thank Chih-Jen Lin for his very helpful comments about LIBSVM.

7. REFERENCES

- [1] C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition", *Data Mining and Knowledge Discovery* 2, p. 121-167, 1998
- [2] C. Chang and C. Lin, "LIBSVM: a Library for Support Vector Machines (Version 2.3)", 2001, software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>
- [3] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing Multiple Parameters for Support Vector Machines", *Machine Learning*, 2000
- [4] I. Guyon, J. Weston, S. Barnhill and V. Vapnik, "Gene Selection for Cancer Classification using Support Vector Machines", in *Machine Learning*, 2000
- [5] B. Heisele, T. Poggio and M. Pontil, "Face Detection in Still Gray Images", A.I. Memo No. 1687, Center for Biological and Computational Learning Paper No. 187, M.I.T., Cambridge, MA, May 2000
- [6] Intel's Math Kernel Library, <http://developer.intel.com/software/products/mkl/>
- [7] J. Maydt, "People Detection in Digital Images: A Component-Based Approach to Object Detection", Master thesis, University of Mannheim and Intel Corporation, Dec. 2001.
- [8] C. Papageorgiou, and T. Poggio, "Trainable Pedestrian Detection", *Proceedings of International Conference on Image Processing (ICIP '99)*, Kobe, Japan, October 1999
- [9] H. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection", *IEEE Patt. Anal. Mach. Intell.*, Vol. 20, pp. 22-38, 1998.
- [10] H. Schneiderman, "A Statistical Approach to 3D Object Detection Applied to Faces and Cars", Technical Report CMU-RI-TR-00-06, Carnegie Mellon University, May 2000
- [11] 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
- [12] P. Viola, and M. Jones, "Robust real-time face detection", *Cambridge Research Laboratory Technical Report CRL 2001/01*, Compaq CRL, February 2001, available at <http://citeseer.nj.nec.com/viola01robust.html>
- [13] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio and V. Vapnik, "Feature Selection for SVMs", in *Advances in Neural Information Processing Systems 13*, Cambridge, MA, 2001