Sparse Kernel Approximations for Efficient Classification and Detection

Andrea Vedaldi   Andrew Zisserman
Dept. of Engineering Science, University of Oxford, UK
{vedaldi,az}@robots.ox.ac.uk

Abstract

Efficient learning with non-linear kernels is often based on extracting features from the data that "linearise" the kernel. While most constructions aim at obtaining low-dimensional and dense features, in this work we explore high-dimensional and sparse ones. We give a method to compute sparse features for arbitrary kernels, re-deriving as a special case a popular map for the intersection kernel and extending it to arbitrary additive kernels. We show that bundle optimisation methods can handle efficiently these sparse features in learning. As an application, we show that product quantisation can be interpreted as a sparse feature encoding, and use this to significantly accelerate learning with this technique. We demonstrate these ideas on image classification with Fisher kernels and object detection with deformable part models on the challenging PASCAL VOC data, obtaining five to ten-fold speed-ups as well as reducing memory use by an order of magnitude.

1. Introduction

Recent advances in large scale convex optimisation have extended the applicability of linear Support Vector Machines (SVMs) to very large datasets as well as complex regression problem with structured outputs [9, 21, 22, 25]. However, there exist no known general method that can achieve a similar efficiency when working directly with *non-linear kernels*, and these are often necessary to achieve optimal accuracy. This explains the increasing interest in techniques that can efficiently reduce non-linear kernels to linear ones [12, 14, 16–19, 23, 29]. These techniques have been shown to obtain state-of-the-art performance especially in the large scale setting [2, 16].

In more detail, the efficiency of linear SVMs depends on their simple form $f(\mathbf{x};\mathbf{w}) = \langle \mathbf{w}, \mathbf{x} \rangle$ as the inner product of an input data vector $\mathbf{x} \in \mathbb{R}^d$ and a parameter vector $\mathbf{w} \in \mathbb{R}^d$. By contrast, non-linear SVMs $f(\mathbf{x};\boldsymbol{\alpha}) = \sum_{i=1}^n \alpha_i K(\mathbf{x}_i, \mathbf{x})$ are expanded in term of evaluations of a non-linear kernel function $K(\mathbf{x}, \mathbf{x}')$ and are much slower to compute as well as train. However, since all kernels $K(\mathbf{x}, \mathbf{y})$ can be represented as inner products up to a transformation $\Psi(\mathbf{x})$ of the data, *i.e.* $K(\mathbf{x}, \mathbf{y}) = \langle \Psi(\mathbf{x}), \Psi(\mathbf{y}) \rangle$, in principle one can reduce any non-linear SVM to a much faster linear one. Unfortunately, this usually requires features $\Psi(\mathbf{x})$ that are infinite dimensional and/or difficult to compute. This motivates the interest in feature maps $\hat{\Psi}(\mathbf{x}) \in \mathbb{R}^D$ that *approximate* a given kernel, *i.e.* $K(\mathbf{x}, \mathbf{y}) \approx \langle \hat{\Psi}(\mathbf{x}), \hat{\Psi}(\mathbf{y}) \rangle$, while being finite dimensional and computationally cheap.

In order to represent the non-linearity, the dimension D of the feature $\hat{\Psi}(\mathbf{x})$ is usually *larger* than the dimension d of the input data $\mathbf{x}$. While most authors have focused on reducing D (see Sect. 6), a large dimensionality may not be an issue *provided that the features are sparse* [14]. The questions then become: which kernels can be represented by sparse features? How to compute them? And what advantages or disadvantages do these representations offer?

Our first contribution is to derive a general and simple construction for *sparse approximate features* $\hat{\Psi}(\mathbf{x})$ for any kernel $K(\mathbf{x}, \mathbf{x}')$ (Sect. 2.1). The theory relates sparse features and existing dense ones geometrically (Sect. 6). It also includes as a special case the intersection kernel map of Maji and Berg [14] and generalises it to arbitrary additive kernels (Sect. 2.1).

Our second contribution is a fast learning method for the new sparse representations. In contrast to their dense counterparts, the sparse features approximate a kernel by a non-diagonal inner product (Sect. 2.1). We introduce a bundle optimisation method [22] that *learns efficiently with sparse data and non-diagonal regularisers* (Sect. 4). This algorithm is the key in leveraging the sparsity of the features not just as a way of using less memory, but also as a way to improve learning speed.

Our third contribution is to apply the theory to *product quantisation* (PQ) [8]. PQ is a data compression method for large scale learning that was used, for example, by the best entry in the 2011 edition of the ImageNet challenge [2, 7, 19]. We show that PQ can be interpreted as a sparse feature map (Sect. 3) and that this allows an algorithm to *learn on the compressed data directly*, with a five to ten fold speedup on the standard approach (Sect. 5.1) in ad-

978-1-4673-1228-8/12/$31.00 ©2012 IEEE

dition to the usual substantial reduction in storage. We also apply PQ for the first time to deformable part models [6], demonstrating similar benefits (Sect. 5.2).

2. Kernels and approximate feature maps

Learning a linear SVM amounts to fitting a linear function $f(\mathbf{x};\mathbf{w}) = \langle \mathbf{w}, \mathbf{x} \rangle$ to labels $y_1, \dots, y_n \in \{-1, +1\}$ at points $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$, *i.e.* finding a parameter vector $\mathbf{w}$ such that $y_i \approx f(\mathbf{x}_i;\mathbf{w})$ for all i. This is usually formulated as the convex optimisation problem

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{\gamma}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_i(\langle \mathbf{w}, \mathbf{x}_i \rangle) \tag{1}$$

where $\mathcal{L}_i(z)$ is the hinge loss $\max\{0, 1-y_i z\}$. Solvers such as [9,10,21,22] can be used to optimise (1) very quickly, in time linear in the data size n and dimension d.

Linear SVMs are fast but not very expressive. Fortunately, the space of representable functions $f(\mathbf{x};\mathbf{w}) = \langle \mathbf{w}, \Psi(\mathbf{x}) \rangle$ can be easily extended by encoding the data by a non-linear map $\Psi(\mathbf{x}) \in \mathbb{R}^D$. This usually entails *augmenting* the data dimensionality, *i.e.* $D > d$, and therefore increasing by a factor D/d the space and time complexity of the linear SVM. Using a *sparse* encoding may help in reducing this cost.

Speed, however, is not the only concern in the design of a good encoding. Arbitrarily increasing the class of representable functions $f(\mathbf{x};\mathbf{w})$ may in fact *reduce* the performance on the test data due to over-fitting. Fitting the data must be traded off with the regularity of the function $f(\mathbf{x};\mathbf{w})$, as measured by the term $\|\mathbf{w}\|^2$ in the SVM objective (1). The meaning of $\|\mathbf{w}\|^2$ in term of statistical properties of the function $f(\mathbf{x};\mathbf{w})$ is determined by $\Psi(\mathbf{x})$ itself, or, more fundamentally, by the *kernel function* $K(\mathbf{x}, \mathbf{x}') = \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle$. In fact, any two encodings that generate the same kernel learn the same functions. Moreover, all PD functions $K(\mathbf{x}, \mathbf{x}')$ are the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ of an encoding [20], or *feature map*, $\Psi : \mathcal{X} \to \mathcal{H}$ in a suitable inner-product space $\mathcal{H}$, called *feature space* (Fig. 1a):

$$K(\mathbf{x}, \mathbf{x}') = \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle_{\mathcal{H}}. \tag{2}$$

Since a kernel directly captures the statistical properties of the SVM, it is often desirable to *derive* an encoding *from* a given kernel rather than designing it directly. While there exist general methods for constructing a feature space $\mathcal{H}$ for any kernel [20, pag. 33], in calculations one needs to express the features as finite vectors of coordinates, which is harder to obtain, and even impossible if $\mathcal{H}$ is infinite dimensional. This motivates looking for approximate but finite dimensional features, which is the subject of the next section.

2.1. Approximate feature maps

The goal is to approximate a given kernel K by a feature map $\hat{\Psi} : \mathcal{X} \to \mathbb{R}^D$ that is finite dimensional, possibly sparse, and efficient to compute. Approximating the kernel means that $\hat{K}(\mathbf{x}, \mathbf{x}') = \langle \hat{\Psi}(\mathbf{x}), \hat{\Psi}(\mathbf{x}') \rangle$ must be close, in some sense, to $K(\mathbf{x}, \mathbf{x}')$. We give first a standard construction for dense approximate features and then we modify it to yield sparse ones. It may be helpful to refer to Fig. 1 for a geometric interpretation.

Dense approximations. A common feature construction technique is to approximate the exact feature space $\mathcal{H}$ of the kernel K by a subspace $\mathcal{H}' \subset \mathcal{H}$ that is (i) finite dimensional and (ii) has a computable coordinate representation. A simple way to do so is to define $\mathcal{H}'$ as the span of D feature vectors $\Psi(\mathbf{z}_i)$ of representative data points $\mathbf{z}_1, \dots, \mathbf{z}_D \in \mathcal{X}$ [23,29] (Fig. 1b), *i.e.*:

$$\mathcal{H}' = \left\{ \sum_{i=1}^{D} \Psi(\mathbf{z}_i)\Phi_i : \Phi \in \mathbb{R}^D \right\}.$$

Then, for each feature $\Psi(\mathbf{x}) \in \mathcal{H}$, one defines an approximation $\hat{\Psi}(\mathbf{x}) \in \mathcal{H}'$. If the goal is to preserve the kernel between any two points $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, the Cauchy-Schwartz bound

$$(\langle \hat{\Psi}(\mathbf{x}), \Psi(\mathbf{x}') \rangle_{\mathcal{H}} - \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle_{\mathcal{H}})^2 \leq \|\hat{\Psi}(\mathbf{x}) - \Psi(\mathbf{x})\|_{\mathcal{H}}^2 \cdot \|\Psi(\mathbf{x}')\|_{\mathcal{H}}^2, \tag{3}$$

suggests to define $\hat{\Psi}(\mathbf{x})$ as the minimiser the norm of the residual $\|\hat{\Psi}(\mathbf{x}) - \Psi(\mathbf{x})\|_{\mathcal{H}}$. This is the same as defining $\hat{\Psi}(\mathbf{x})$ to be the *orthogonal projection* of the exact feature vector $\Psi(\mathbf{x})$ on the subspace $\mathcal{H}'$.

The coordinates $\Phi(\mathbf{x})$ of the approximation $\hat{\Psi}(\mathbf{x})$ can be computed by minimising the residual analytically:

$$\Phi(\mathbf{x}) = \operatorname*{argmin}_{\Phi\in\mathbb{R}^D} \left\| \Psi(\mathbf{x}) - \sum_{i=1}^{D} \Psi(\mathbf{z}_i)\Phi_i \right\|_{\mathcal{H}}^2. \tag{4}$$

Let $K_{XX'}$ denote the Gram (kernel) matrix calculated at points $X = (\mathbf{x}_1, \dots, \mathbf{x}_m)$ and $X' = (\mathbf{x}'_1, \dots, \mathbf{x}'_p)$, *i.e.* $K_{ij} = K(\mathbf{x}_i, \mathbf{x}'_j)$. Moreover, let $Z = (\mathbf{z}_1, \dots, \mathbf{z}_D)$ be the sequence of the D representative points. Rewriting the residual (4) in matrix form as $K_{\mathbf{xx}} - 2K_{\mathbf{x}Z}\Phi + \Phi^\top K_{ZZ}\Phi$, differentiating w.r.t. Φ, and equating to zero yields the coordinate $\Phi(\mathbf{x})$ of point $\mathbf{x}$

$$\boxed{\Phi(\mathbf{x}) = K_{ZZ}^{\dagger} K_{Z\mathbf{x}}}, \tag{5}$$

where † denotes the pseudoinverse (as the kernel matrix may not be full rank). The functions $\Phi_i : \mathcal{X} \to \mathbb{R}$ are called *coordinate functions*.

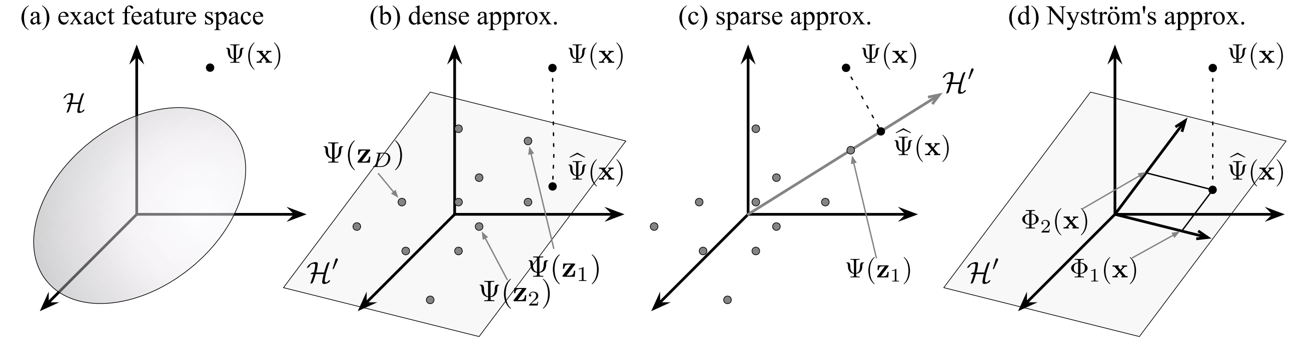

Figure 1: **Geometric interpretation. (a)** The exact feature space, reconstructing the kernel with no approximation. A distribution $p(\mathbf{x})$ on the input data maps to a corresponding density in this space (ellipsoid). **(b)** The dense feature approximation is obtained by projecting the exact feature vector $\Psi(\mathbf{x})$ to the span $\mathcal{H}'$ of representatives $\Psi(\mathbf{z}_1),\dots,\Psi(\mathbf{z}_D)$. **(c)** The sparse feature approximation uses a different subset $\bar{Z}$ of representatives for each encoded point $\mathbf{x}$ (in this case $\bar{Z}=\{\mathbf{z}_1\}$). **(d)** Nyström's approximation (Sect. 6) defines $\mathcal{H}'$ to be the PCA subspace of the the data distribution (a); the coordinates $\Phi_1(\mathbf{x})$, $\Phi_2(\mathbf{x})$ are expressed relative to an orthonormal PCA basis obtained from an eigenvalue problem.

The approximated feature vector is then given by $\hat{\Psi}(\mathbf{x}) = \sum_{i=1}^{D}\Psi(\mathbf{z}_i)\Phi_i(\mathbf{x})$, which generates the kernel $\langle\hat{\Psi}(\mathbf{x}),\hat{\Psi}(\mathbf{x}')\rangle_{\mathcal{H}} = \sum_{i,j=1}^{D}\Phi_i(\mathbf{x})K(\mathbf{z}_i,\mathbf{z}_i)\Phi_j(\mathbf{x}')$, *i.e.*:

$$\boxed{\hat{K}(\mathbf{x},\mathbf{x}') = \Phi(\mathbf{x})^\top K_{ZZ}\Phi(\mathbf{x}')}. \tag{6}$$

This expresses the approximated kernel $\hat{K}$ as the non-diagonal inner product in $\mathbb{R}^D$ given by the PD matrix K_{ZZ}. An equivalent diagonal representation can be obtained by a change of coordinates. For example, let $K_{ZZ} = V^\top\Lambda^2 V$ be the eigen-decomposition of the kernel matrix and define $\bar{\Phi}(\mathbf{x}) = \Lambda V\Phi(\mathbf{x})$. Then $\hat{K}(\mathbf{x},\mathbf{x}') = \bar{\Phi}(\mathbf{x})^\top\bar{\Phi}(\mathbf{x}')$.

Both the coordinate vectors $\Phi(\mathbf{x})$ and $\bar{\Phi}(\mathbf{x})$ are dense in general because the feature vectors $\Psi(\mathbf{x})$ are approximated as *combinations of all the representative points Z*.

Sparse approximations. The key idea to obtain a sparse feature map is to note that each given feature vector $\Psi(\mathbf{x})$ can often be approximated well by a small subset of the D representative points $\mathbf{z}_1,\dots,\mathbf{z}_D$, for example its neighbours (Fig. 1c). Formally, one can restrict the approximation (4) to use only $P < D$ representative points by solving

$$\Phi(\mathbf{x};P) = \min_{\|\Phi\|_0\le P}\left\|\Psi(\mathbf{x}) - \sum_{i=1}^{D}\Psi(\mathbf{z}_i)\Phi_i\right\|_{\mathcal{H}}^2. \tag{7}$$

The constraint $\|\Phi\|_0 \le P$ allows at most P representative points $\bar{Z}\subset Z$ to have non-zero coefficients. These coefficients are still given by the formula (5) by replacing Z with $\bar{Z}$. Denoting by $\Phi(\mathbf{x};\bar{Z})$ the resulting sparse coordinate vector, the problem (7) is reduced to finding the best subset $\bar{Z}_{\mathbf{x}}$:

$$\bar{Z}_{\mathbf{x}} = \operatorname*{argmin}_{\bar{Z}\subset Z,|\bar{Z}|\le P}\left\|\Psi(\mathbf{x}) - \sum_{i=1}^{D}\Psi(\mathbf{z}_i)\Phi_i(\mathbf{x};\bar{Z})\right\|_{\mathcal{H}}^2.$$

This is the feature space equivalent of the sparse encoding/reconstruction problem, and as such can be approximately solved by popular techniques such as matching pursuit, orthogonal matching pursuit, and Lasso. In certain applications it is also possible to specify $\bar{Z}_{\mathbf{x}}$ directly, as in the example below. The idea of choosing the representative as a function of $\mathbf{x}$ can be found in the context of Gaussian processes in [24].

While the vectors $\Phi(\mathbf{x};P)$ are sparse by construction, the kernel approximation is still given by (6), which involves the multiplication by the full matrix K_{ZZ}. Switching to the diagonal representation $\bar{\Phi}(\mathbf{x}) = \Lambda V\Phi(\mathbf{x};P)$ as before eliminates K_{ZZ} but results in a dense coordinate vector. Therefore *the key to efficient learning with sparse features is finding an algorithm that can work efficiently with sparse data and a non-diagonal inner product.*

Example: sparse maps for additive kernels. An *additive kernel* $K(\mathbf{x},\mathbf{x}')$ on $\mathbb{R}^d$ decomposes as a sum $\sum_{j=1}^{d}k(x_j,x_j')$ of 1D kernels $k(x,x')$. Examples include the χ^2 kernel $k(x,x') = 2xx'/(x+x')$, the Hellinger's kernel $\sqrt{xx'}$, and the intersection kernel $\min\{x,x'\}$, which have been shown to perform particularly well for histogram data [30] (bag-of-visual visual words, spatial histograms, colour histograms, HOG, etc.).

Since a feature for the additive kernel $K(\mathbf{x},\mathbf{x}')$ can be obtained by stacking the features for each of the components $k(x_i,x_i')$, it suffices to derive the latter. To construct a sparse feature $\Phi(x)$ for the 1D kernel $k(x,x')$ one can

(i) sample D representative points z_i uniformly on $\mathbb{R}$ and (ii) project each point $x \in \mathbb{R}$ to the two adjacent representatives $\bar{Z} = \{z_i, z_{i+1}\}$ such that $z_i \leq x < z_{i+1}$. Consider for example the intersection kernel $\min\{x, x'\}$ and, without loss of generality, the set of representative points $Z = \{0, 1, 2, \dots, D-1\}$. Evaluating $\Phi(\mathbf{x})$ selects the two points $\bar{Z} = \{i, i+1\}$, $i = \lfloor x \rfloor$ and sets the respective coefficients in $\Phi(x)$ using (5), *i.e.*

$$K^{\dagger}_{\bar{Z}\bar{Z}} K_{\bar{Z}x} = \begin{bmatrix} i & i \\ i & i+1 \end{bmatrix}^{-1} \begin{bmatrix} i \\ x \end{bmatrix} = \begin{bmatrix} i+1-x \\ x-i \end{bmatrix} \tag{8}$$

This equation linearly distributes x to z_i and z_{i+1}. The resulting feature $\Phi(x)$ is identical to the sparse intersection kernel map of [14], which can therefore be interpreted as a sparse projection method, optimal in the sense of (4). In addition to allowing for many variants (*e.g.* using more than two representative points), the theory extends this construction to arbitrary additive kernels, including χ^2, *for which we give the first sparse feature approximation*:

$$K^{\dagger}_{\bar{Z}\bar{Z}} K_{\bar{Z}x} = \frac{2(1+2i)x}{(i+x)(1+i+x)} \begin{bmatrix} i+1-x \\ x-1 \end{bmatrix}. \tag{9}$$

The next section investigates a completely different application of the theory, namely product quantisation.

3. Product quantisation as a sparse feature

This section applies the general theory of Sect. 2.1 to *Product Quantisation* (PQ) [8, 19]. Consider a dataset of high-dimensional descriptors $\mathbf{x}_i \in \mathbb{R}^d$ (*e.g.* spatial histograms [11], Fisher vectors [19]). Storing one descriptor $\mathbf{x}_i$ requires bd bits, where b is the average number of bits per component, 32 if IEEE single precision math is used. The latter is just a particular way of encoding *independently* each descriptor component in space of 2^{32} elements. PQ encodes the data more efficiently by considering *groups* of components instead. In detail, PQ (i) partitions the vector $\mathbf{x}_i$ into M blocks of $G = d/M$ components, (ii) quantises each block into a codebook of 2^{bG} elements by using k-means on sample data, and (iii) stores for each block just the index of the corresponding codeword, for a total of $bGM = bd$ bits.

PQ reduces the memory required to store the dataset by a factor $32/b$. In practice, this factor can be an order of magnitude or more [19], making the technique very useful for large-scale learning, where the uncompressed image descriptors can occupy terabytes. In particular, if compression allows for the entire data to be stored in central memory, this dramatically accelerates learning compared to accessing data from disk. According to [19], however, the PQ compression offers no *intrinsic* speed benefit because the data must be uncompressed on the fly in the solver, as the latter must still operate on the original vectors $\mathbf{x}_i$. We show here that, by interpreting PQ as a sparse feature encoding, it is instead possible to learn directly on the PQ compressed data, achieving a significant speed-up in combination with appropriate solvers (Sect. 4).

In particular, consider learning a linear SVM (1) on PQ compressed data. Assume for simplicity that there is only one PQ block ($M = 1$), as the extension to $M > 1$ is immediate by stacking. PQ approximates each data point $\mathbf{x}$ by the closest element $\mathbf{z}_i$ in a codebook $Z = (\mathbf{z}_1, \dots, \mathbf{z}_D) \in \mathbb{R}^{G \times D}$, where $D = 2^{bG}$ is the number of codewords. This is the same approximation given by the sparse feature construction (7) by imposing that the feature $\Phi(\mathbf{x})$ has exactly one non-zero component ($P = 1$) and that the latter is equal to 1. The latter condition ensures that only the index of the non-zero component must be stored and not its value as well.

The sparse PQ feature map $\Phi(\mathbf{x})$ in combination with the inner product K_{ZZ} results in the approximated kernel (Sect. 2.1)

$$\hat{K}(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^\top K_{ZZ} \Phi(\mathbf{x}) = (\Phi(\mathbf{x})Z)^\top (Z\Phi(\mathbf{x}'))$$

where $Z\Phi(\mathbf{x})$ returns the codeword $\mathbf{z}_i$ that PQ uses to approximate $\mathbf{x}$. Therefore decompressing the data in the solver as in [19] results in exactly the same approximate kernel as the PQ features, and the two learning methods are equivalent (Sect. 2). Handling non-linear (*e.g.* additive) kernels is similar, except that PQ must be modified to use the metric induced by the kernel, rather than the Euclidean one.

The next section introduces a solver that can accelerate learning by using the PQ sparse features.

4. Efficient learning with bundle methods

Modifying the linear SVM learning problem (1) to use the sparse feature $\Phi(\mathbf{x})$ of Sect. 2.1 and the corresponding non-diagonal inner product $\langle \cdot, \cdot \rangle_{K_{ZZ}}$ yields the objective

$$E(\mathbf{w}) = \frac{\lambda}{2} \mathbf{w}^\top K_{ZZ} \mathbf{w} + \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i(\mathbf{w}^\top K_{ZZ} \Phi(\mathbf{x}_i)). \tag{10}$$

To exploit the sparsity of the data $\Phi(\mathbf{x})$ in the calculation of the loss (second term), one can lump together the dense factors in $\mathbf{v} = K_{ZZ}\mathbf{w}$ and consider the equivalent objective

$$E(\mathbf{v}) = \frac{\lambda}{2} \mathbf{v}^\top K^{\dagger}_{ZZ} \mathbf{v} + \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i(\mathbf{v}^\top \Phi(\mathbf{x}_i)). \tag{11}$$

To handle efficiently non-diagonal regulariser $K^{\dagger}_{ZZ}$ as well as the the sparse data we propose to modify the cutting-plane/bundle solver of [9, 22]. This starts by collecting all the individual loss terms in (11) into a single loss function

$$\mathcal{L}(\mathbf{v}^\top \Phi(X)) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i(\mathbf{v}^\top \Phi(\mathbf{x}_i))$$

where $\Phi(X) = [\Phi(\mathbf{x}_1), \dots, \Phi(\mathbf{x}_n)]$, and then solving the convex optimisation problem

$$\min_{\mathbf{v}\in\mathbb{R}^D, \xi\in\mathbb{R}} \frac{\lambda}{2}\mathbf{v}^\top K_{ZZ}^\dagger \mathbf{v} + \xi, \quad \xi \geq \mathcal{L}(\mathbf{v}^\top\Phi(X)). \tag{12}$$

This is called *one-slack formulation* [9] because ξ is a single scalar slack variable capturing the loss averaged over all example data points. The idea is then to construct a sequence of approximated solutions $\mathbf{v}_1, \dots, \mathbf{v}_t, \dots$ corresponding to a progressively more refined piecewise linear lower approximation of the loss function

$$\max_{i=1,\dots,t} b_t - \langle \mathbf{a}_t, \mathbf{v}\rangle \leq \mathcal{L}(\mathbf{v}^\top\Phi(X)),$$

where $(\mathbf{a}_t, b_t)$ are the parameters of the plane added at iteration t. Each plane is tangent to the loss at $\mathcal{L}(\mathbf{v}^\top\Phi(X))$ at $\mathbf{v}_t$:

$$\mathbf{a}_t = -\Phi(X)\nabla\mathcal{L}(\mathbf{v}_t^\top\Phi(X)), \quad b_t = \mathcal{L}(\mathbf{v}_t^\top\Phi(X)) + \mathbf{v}_t^\top\mathbf{a}_t. \tag{13}$$

Note that the planes are under-estimators of the loss due to the convexity of the latter; if the loss is not differentiable, then one simply takes a sub-gradient instead of the gradient in (13).

Given the current solution $\mathbf{v}_t$, the next one $\mathbf{v}_{t+1}$ is found as the minimiser of the convex problem

$$\begin{aligned} &\min_{\mathbf{v}\in\mathbb{R}^D, \xi\in\mathbb{R}} \frac{\lambda}{2}\mathbf{v}^\top K_{ZZ}^\dagger \mathbf{v} + \xi, \\ &\qquad \xi \geq b_i - \langle \mathbf{a}_i, \mathbf{v}\rangle, \quad i = 1, 2, \dots, t. \end{aligned} \tag{14}$$

The dual of this problem is given by

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}_+^t, \|\boldsymbol{\alpha}\|_1 = 1} \mathbf{b}^\top\boldsymbol{\alpha} - \frac{1}{2\lambda}\boldsymbol{\alpha}^\top(A^\top K_{ZZ} A)\boldsymbol{\alpha} \tag{15}$$

where $\boldsymbol{\alpha}$ are the dual variables, $A = [\mathbf{a}_1, \dots, \mathbf{a}_t]$, $\mathbf{b} = [b_1, \dots, b_t]^\top$ and $\mathbf{v} = K_{ZZ}\boldsymbol{\alpha}/\lambda$ at the optimum (strong duality condition). The size of the dual problem is equal to the iteration index t and unrelated to the number of training samples n. Since typically $t \ll n$, this explains the efficiency of the method. Note also that the term $A^\top K_{ZZ} A$ involves multiplying by the kernel matrix K_{ZZ} instead of the pseudo-inverse $K_{ZZ}^\dagger$ that appears in the primal (11).

Assuming that the algorithm converges at $t \ll n$, as typical, the cost of each iteration is $O(Pn + D^2)$, where P is the number of non-zero components in the data. $O(Pn)$ operations are required to generate the new plane $(\mathbf{a}_t, b_t)$ by scanning the dataset, and $O(D^2)$ operations are required to compute the product $\tilde{\mathbf{a}}_t = K_{ZZ}\mathbf{a}_t$.

By comparison, using the equivalent dense features $\bar{\Phi}(\mathbf{x})$ with diagonal regulariser (Sect. 2.1) requires $O(Dn)$ operations. Hence the sparse features use D/P times less memory *and* are faster provided that $Pn + D^2 < Dn$. While the latter is always true for sufficiently large n (since $P \ll D$ by construction), we see next how this complexity can be significantly reduced by exploiting the structure of K_{ZZ}.

Exploiting the structure of K_{ZZ}. The bottleneck in the bundle algorithm is the multiplication $K_{ZZ}\mathbf{a}_i$. For example, for PQ with d/G blocks and $D = 2^{bG}$ codewords per block, this multiplication requires $O(dD^2/G)$ operations (for all the blocks). Fortunately, this calculation can be significantly accelerated by using the factorisation $K_{ZZ}\mathbf{a}_i = Z^\top(Z\mathbf{a}_i)$, as this requires only $O(dD)$ operations. With this improvement, using the sparse PQ features with the bundle solver (which we call *delayed expansion*) uses a fraction

$$\frac{d2^{bG} + dn/G}{dn} = \frac{1}{G} + \frac{2^{bG}}{n} \tag{16}$$

of the operations that would be required by expanding points on the fly (*immediate expansion*) as in [19], or by using directly dense features. Since the last term becomes rapidly negligible for a large number of data points n, this algorithm is roughly G times faster, where G is the size of the PQ blocks.

As a second example, consider the intersection kernel approximation (8). In this case the kernel matrix K_{ZZ} has the form

$$K_{ZZ} = V^\top V, \quad V = \begin{bmatrix} 0 & 1 & 1 & \dots & 1 \\ 0 & 0 & 1 & \dots & 1 \\ & & \dots & & \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}. \tag{17}$$

Since the products $V\mathbf{v}$ and $V^\top(V\mathbf{v})$ are cumulative sums, these can be computed in time $O(D)$ rather than $O(D^2)$.

Comparison with stochastic gradient descent. Stochastic gradient methods such as PEGASOS [21] are sometimes preferred to bundle methods because they offer similar convergence speed but simpler implementation. For example, [14] apply PEGASOS to the optimisation of (11), which results in the simple update equation

$$\mathbf{v} \leftarrow \mathbf{v} - \eta_t\Big(\lambda K_{ZZ}^\dagger\mathbf{v} + \delta_i\Phi(\mathbf{x}_i)\Big) \tag{18}$$

where η_t is the learning rate and $\delta_i = \dot{\mathcal{L}}_i(\mathbf{v}^\top\Phi(\mathbf{x}_i))$ is the derivative of the loss. However, since $K_{ZZ}^\dagger$ is full, the SGD update is not sparse. While this can be alleviated by optimisations such as the use of mini-batches [21], it is still a significant bottleneck, especially if the multiplication by K_{ZZ} requires $O(D^2)$ operations as in the general case.

To summarise, the bundle solver performs a dense operation *only after each complete pass over the data, and therefore seems more suitable to handle sparse represen-*

tations with non-isotropic regularisers.[1] The next section demonstrates applications to the efficient learning of image classifiers and object detectors.

5. Experiments

5.1. PQ for image classification

This section evaluates the PQ compression technique on the PASCAL VOC 2007 classification challenge [3]. The task is to classify twenty object categories and the performance is measured as mean Average Precision (mAP). Each image is represented by the Fisher encoding $\mathbf{x}_i$ obtained as described in [19]: first, dense SIFT features are extracted every three pixels (using the `vl_phow` function of [26]) and compressed from dimension 128 to 80 by using PCA; then a Gaussian mixture model with 256 components is fitted to sample PCA-SIFT descriptors and used to generate a Fisher encoding of the image for 3 × 1 spatial subdivisions [11, 19], for a total of $2 \times 256 \times 80 \times 3 = 40{,}960$ dimensions. The dataset totals about 5,000 training images and occupies about 2GB of memory.

The baseline system works as well as [19] (about 59% mAP) and can be improved slightly by increasing the number of spatial subdivisions (up to about 62% in our experiments). This is a very solid baseline which, combined with PQ, let [19] top the ImageNet classification challenge in 2011 [2, 7]. In fact, PQ allows for a **tenfold reduction of memory usage** (from 2GB to about 100MB in our case, see Fig. 2) with minimal impact on the classification performance (less than 1% mAP).

Our sparse PQ features (delayed expansion) further improve this excellent system by **a 5–10 fold speedup in training**, compared to decompressing the data on the fly (immediate expansion) [19]. As predicted by (16), the gain is larger for larger block sizes G, which also results in better accuracy. As the number of codewords 2^{bG} increases, the term $2^{bG}/n$ starts to dominate and the speedup becomes smaller (16). This indicates that our technique is particularly useful for a very large number n of training samples.

5.2. PQ for object detection

This section proposes a new application area of PQ, namely *deformable parts models* (DPM) [6] for object category detection. A DPM is a collection of spring-connected parts whose appearance is described by HOG [1] templates. Matching a part to an image location requires multiplying the part template by the HOG descriptor extracted at that location. The latter is a collection of $w \times h$ HOG cells, each of which is a d-dimensional feature vector (where $d = 32$ in [6]), so this costs $O(whd)$ operations. Detecting with the model requires matching all parts at all locations and scales, a costly multi-dimensional convolution operation, and is the main bottleneck in testing [15]. Learning a DPM is even more expensive, as this entails testing multiple times on the training data in order to identify the negative support vectors (mining of hard negatives) [6, 27].

PQ can be used to encode HOG cells with a single codeword index (Sect. 3). To test this idea, we reimplemented [6] from scratch, using a bundle solver rather than the original SGD method [6]. This solver has the advantages discussed above (Sect. 5.1) and was found to be as fast as SGD in the standard (uncompressed) case. The code is run on a machine with twelve cores, parallelising operations such as mining for the hard negative examples. Other refinements from [6], such as bounding box regression and contextual rescoring, are not used in our experiments.

Space saving. Space is used to store the hard negative examples (about 1GB) and the pre-computed HOG features for all training images (about 14GB). For $D = 256$ and $D = 512$ PQ codewords a modest drop in detection accuracy (about 3% mAP) was observed. However, encoding each HOG cell with $\log D$ bits rather the $32 \times 32 = 1{,}024$ required by IEEE single precision floats gives a **113 fold reduction of storage** for $D = 512$. Compared to using the common trick of remapping the feature components to bytes, as done in [13] and in Fig. 3 for the baseline results, PQ still results in a 28-fold saving. In order to further simplify our implementation, we simply used 32 bits to encode each HOG codeword index (for $D = 512$ this wastes $32 - 9 = 23$ bits). Even so, the observed storage reduction is more than tenfold, from a dozen GBs to less than one.

Time saving. Testing can be up to $d = 32$ times faster with PQ because convolving a part filter requires now just wh operations rather than whd (because each HOG cell is represented by a $P = 1$ sparse vector). Yet each of the wh operations involves a non-local memory access. Moreover, our sparse convolution code is not as optimised as the routine that we use for the dense case (which uses a fast BLAS implementation). Overall, the sparse convolution is "just" twice as fast than the standard method (Fig. 3). This speedup is orthogonal to others such as [4, 15] and can be combined with them. Since testing is an integral part of training, and in fact is its bottleneck, this speedup transfers to training too. Finally, the bundle solver benefits from using PQ, as indicated by (16), and is three times faster than using the uncompressed data.

6. Related methods

This section summarises existing methods for the construction of approximate feature maps for kernels.

[1] A second problem not discussed in [14] is the choice of a learning rate. While $\eta_t = 1/(\lambda t)$ is optimal for a SVM with isotropic regulariser λ [21] (which is also the difference between PEGASOS and standard SGD), handling (11) is trickier. For example, $K^{\dagger}_{ZZ}$ for the intersection kernel (17) does not have full rank, meaning that the objective function is not even strongly convex. In practice, we found this tuning to be delicate.

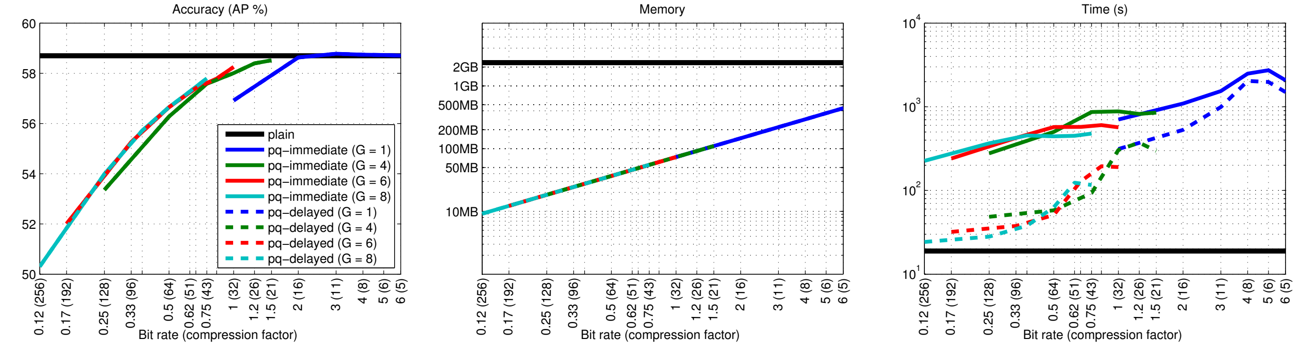

Figure 2: **PQ for classification.** Plain linear SVM (heavy black horizontal line), PQ SVM with immediate expansion (solid lines, [19]) and our PQ SVM with delayed expansion (dashed lines, Sect. 3). From the left to right: mAP classification accuracy on PASCAL VOC 2007, memory, and learning time. PQ allows for a substantial compression ($>$ 40 times) with minimal accuracy loss (1% mAP). Delayed expansion results in a significant speed-up (up to 10 times) compared to the usual immediate expansion method [19]. In principle, delayed expansion is G times faster than the plain (uncompressed) SVM too, but this is not observed due to implementation details (the standard SVM learning code uses an optimised BLAS library, while our delayed expansion code does not). Crucially, however, the plain solver cannot be used efficiently if the uncompressed data does not fit in central memory [19].

	aerop	bicyc	bird	boat	bottl	bus	car	cat	chair	cow	dinin	dog	horse	motor	perso	potte	sheep	sofa	train	tvmon	mean
AP [%]	30.6	58.1	10.0	12.7	21.2	52.1	55.0	20.1	19.4	22.1	20.5	11.3	56.4	43.6	38.8	10.8	14.8	25.6	43.8	43.8	30.5
neg. mining [min]	173	173	96	169	171	174	143	127	115	160	156	96	161	163	42	136	172	152	176	168	146
solver [min]	6.1	6.2	12.5	6.7	6.8	5.4	8.9	11.2	13.9	7.2	8.3	14.0	7.1	9.1	17.9	9.8	6.3	9.5	6.5	8.2	9.1
neg. mined [GB]	1.02	1.09	1.27	1.05	1.06	1.05	1.05	1.10	1.15	1.06	1.10	1.11	1.13	1.10	1.19	1.04	1.03	1.13	1.11	1.09	1.10
data [GB]	14.0	13.6	15.0	13.2	12.5	13.6	17.8	15.8	14.5	13.1	13.6	16.6	15.1	14.3	34.6	13.3	13.1	13.7	14.7	12.6	15.2
AP [%]	27.9	55.2	9.5	10.4	16.4	47.6	52.0	16.0	13.5	18.6	20.7	10.7	53.4	39.7	37.3	10.4	12.7	19.7	41.7	40.9	27.7
neg. mining [min]	97	100	68	75	89	103	84	63	63	75	88	64	90	93	28	58	90	82	104	94	80
solver [min]	2.4	2.1	3.9	2.1	2.8	1.8	2.3	4.2	4.8	2.7	2.7	4.8	2.6	3.9	4.8	3.2	2.0	3.1	2.3	2.3	3.0
neg. mined [GB]	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.05	0.05	0.04	0.05	0.05	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04
data [GB]	0.9	0.9	1.0	0.9	0.8	0.9	1.2	1.1	1.0	0.9	0.9	1.1	1.0	1.0	2.4	0.9	0.9	0.9	1.0	0.8	1.0

Figure 3: **PQ for detection.** Space and time complexity of learning deformable parts model [5] on the PASCAL VOC 2007 data for the twenty classes: the detector accuracy as AP on the test data, the time required to mine the hard negative examples, the time spent in the bundle solver, the space required to store the hard negatives, and the space required to stored the pre-extracted HOG features for the training data. **Top:** standard method. **Bottom:** sparse features from PQ.

Nyström's (PCA) approximations. Most feature constructions are variants of Nyström's approximation [29], whose geometry is also related to the one of our sparse features (Fig. 1). Given a data distribution $p(\mathbf{x})$, this approximation seeks directly the feature map $\Phi : \mathcal{X} \rightarrow \mathbb{R}^D$ that minimises the average reconstruction error of the kernel K. This is given by *projecting the exact feature* $\Psi(\mathbf{x})$ *on the top D principal components of the weighted kernel* $K(\mathbf{x},\mathbf{x}')p(\mathbf{x})p(\mathbf{x}')$ [29] (Fig. 1d).The corresponding coordinate functions (Sect. 2.1) $\Phi_i(\mathbf{x}) = \kappa_i u_i(\mathbf{x})$ are proportional to the eigenfunctions $u_i(\mathbf{x})$ of the kernel with the D largest eigenvalues κ_i^2 [29].

Nyström's construction is often considered a theoretical tool as the density $p(\mathbf{x})$ is not known analytically. An exception is [28] that, by assuming a simple form of $p(\mathbf{x})$, use this construction to derive *closed form* features for kernels such as χ^2 and intersection.

Alternatively, the distribution $p(\mathbf{x})$ can be approximated empirically by the training samples $X = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ (Fig. 1.c). Finding the kernel eigenfunctions reduces to a $n \times n$ discrete eigenproblem, but encoding a new point $\mathbf{x}$ is quite slow as it requires computing the projections $K(\mathbf{x},\mathbf{x}_1), \dots, K(\mathbf{x},\mathbf{x}_n)$. This approximation is equivalent to computing the dense feature map (5) with representative points $Z = X$ followed by PCA. It can be accelerated significantly for additive kernels [16] because in this case the coordinate functions have a scalar argument and can be precomputed.

Sampling. Other popular techniques are based on sampling ideas [18], but the resulting features can be high dimensional and slow to compute in practice (an exception are once more the additive kernels [12]).

7. Summary

We have presented a general method to construct sparse approximate feature maps for arbitrary kernels, relating it to dense constructions based on Nyström's approximation. These representations are based on non-diagonal inner products which can be handled efficiently by bundle optimisation methods. We proposed two applications of the theory: the encoding of additive kernels and accelerating learning with product quantisation. The latter technique is able to reduce significantly the memory needed for large scale learning in classification and detection as well as accelerating optimisation and inference, with only a minor impact on accuracy.

Acknowledgments. This work was supported by the Oxford Violette and Samuel Glasstone Research Fellowships in Science and the ERC grant VisRec no. 228180.

References

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. CVPR*, 2005.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proc. CVPR*, 2009.

[3] M. Everingham, A. Zisserman, C. Williams, and L. V. Gool. The PASCAL visual obiect classes challenge 2007 (VOC2007) results. Technical report, Pascal Challenge, 2007.

[4] P. F. Felzenszwalb, R. Girshick, and D. McAllester. Cascade object detection with deformable part models. In *Proc. CVPR*, 2010.

[5] P. F. Felzenszwalb, R. B. Grishick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *PAMI*, 2009.

[6] P. F. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *Proc. CVPR*, 2008.

[7] http://www.image net.org/challenges/LSVRC/2011/results.

[8] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *PAMI*, 33(1), 2011.

[9] T. Joachims. Training linear SVMs in linear time. In *Proc. KDD*, 2006.

[10] T. Joachims, T. Finley, and C.-N. J. Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 77(1), 2009.

[11] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bag of features: Spatial pyramid matching for recognizing natural scene categories. In *Proc. CVPR*, 2006.

[12] F. Li, C. Ionescu, and C. Sminchisescu. Random fourier approximations for skewed multiplicative histogram kernels. In *Proc. DAGM*, 2010.

[13] D. G. Lowe. Object recognition from local scale-invariant features. In *Proc. ICCV*, 1999.

[14] S. Maji and A. C. Berg. Max-margin additive classifiers for detection. In *Proc. ICCV*, 2009.

[15] M. Pedersoli, A. Vedaldi, and J. Gonzàlez. A coarse-to-fine approach for fast deformable object detection. In *Proc. CVPR*, 2011.

[16] F. Perronnin, J. Sánchez, and Y. Liu. Large-scale image categorization with explicit data embedding. In *Proc. CVPR*, 2010.

[17] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *Proc. ECCV*, 2010.

[18] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Proc. NIPS*, 2007.

[19] J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *Proc. CVPR*, 2011.

[20] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.

[21] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-GrAdient SOlver for SVM. In *Proc. ICML*, 2007.

[22] A. J. Smola, S. V. N. Vishwanathan, and Q. V. Le. Bundle methods for machine learning. In *Proc. NIPS*, 2008.

[23] E. Snelson and Z. Ghaharamani. Sparse Gaussian processes using pseudo-inputs. In *Proc. NIPS*, 2006.

[24] E. Snelson and Z. Ghaharamani. Local and global sparse gaussian process approximations. In *Proc. AISTAT*, 2007.

[25] C. H. Teo, S. V. N. Vishwanathan, A. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 1(55), 2009.

[26] A. Vedaldi and B. Fulkerson. VLFeat – An open and portable library of computer vision algorithms. In *Proc. ACM Int. Conf. on Multimedia*, 2010.

[27] A. Vedaldi and A. Zisserman. Structured output regression for detection with partial occulsion. In *Proc. NIPS*, 2009.

[28] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. In *PAMI*, 2011.

[29] C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Proc. NIPS*, 2001.

[30] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. *IJCV*, 2007.