Power Mean SVM for Large Scale Visual Classification

Jianxin Wu
School of Computer Engineering
Nanyang Technological University, Singapore
jxwu@ntu.edu.sg

Abstract

PmSVM (Power Mean SVM), a classifier that trains significantly faster than state-of-the-art linear and non-linear SVM solvers in large scale visual classification tasks, is presented. PmSVM also achieves higher accuracies. A scalable learning method for large vision problems, e.g., with millions of examples or dimensions, is a key component in many current vision systems. Recent progresses have enabled linear classifiers to efficiently process such large scale problems. Linear classifiers, however, usually have inferior accuracies in vision tasks. Non-linear classifiers, on the other hand, may take weeks or even years to train. We propose a power mean kernel and present an efficient learning algorithm through gradient approximation. The power mean kernel family include as special cases many popular additive kernels. Empirically, PmSVM is up to 5 times faster than LIBLINEAR, and two times faster than state-of-the-art additive kernel classifiers. In terms of accuracy, it outperforms state-of-the-art additive kernel implementations, and has major advantages over linear SVM.

1. Introduction

Classifiers trained with a large set of training examples have played a key role in recent computer vision research. An accurate classifier has been the determining component in object detection [25, 8], object and scene recognition [11, 26], and tracking [3]. It is also important in other vision tasks such as segmentation [20] and retrieval [4].

The availability of ample images and videos, however, poses new challenges for classifier training. With millions of images, training an accurate classifier can take weeks or even years [5, 13]. Recent advances in learning scalable linear classifiers have enabled us to train linear SVM in the order of seconds, even with millions of training examples or millions of feature dimensions, e.g., LIBLINEAR [10] and Pegasos [22].

Unfortunately, in the context of SVM classifiers for computer vision tasks, linear SVM is inferior in terms of accuracy, compared to the kernel versions of SVM. Additive kernels have significantly higher accuracy rates than the dot-product kernel in most vision problems [15, 26, 24, 28].

A kernel is additive if it can be written as the sum of a scalar kernel function for each feature dimension, i.e.,[1]

$$\kappa(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} \kappa(x_i, y_i).$$

Commonly used additive kernels are the histogram intersection kernel $\kappa_{\mathrm{HI}}(x, y) = \min(x, y)$ and the χ^2 kernel $\kappa_{\chi^2}(x, y) = \frac{2xy}{x+y}$. When learning an additive kernel SVM, general purpose SVM solvers may be thousands of times slower than the fast linear solvers. Recent novel algorithms in [15, 26, 24, 18] have bridged the gap. Additive kernel SVM now uses roughly only a few times more training time, compared to the state-of-the-art linear SVM solvers; or even faster than linear SVM in some large vision problems.

However, even these fast solvers may be too slow for modern datasets. LIBLINEAR requires 34 hours to train the 1000 classifiers on the ILSVRC 1000 dataset [1], and existing additive kernel solvers take more than 12 hours to finish (cf. Sec. 5.2.) As we are utilizing more and more images or videos in vision research in order to achieve higher detection / recognition accuracy, it is compulsory to design more efficient solvers for additive kernels.

We propose the Power Mean SVM (PmSVM) algorithm, which consists of the following contributions:

- A novel Power Mean Kernel (Sec. 3.) The power mean kernel is a family of kernels indexed by a parameter p. We show that the popular histogram intersection and χ^2 kernels are both special cases in this family.
- PmSVM, an unified and efficient solver (Sec. 4 and 5.) PmSVM solves the dual SVM formulation using a coordinate descent approach. Existing methods approximate the kernel function and feature mapping. In PmSVM, we approximate the gradient using polynomial regression instead. This novel strategy enables PmSVM to be

[1] We use the same symbol to represent a kernel and its corresponding scalar kernel function. $\mathbf{x} = (x_1, \ldots, x_d)$ and $\mathbf{y} = (y_1, \ldots, y_d)$.

up to 5 times faster than fast linear SVM solvers and two times faster than state-of-the-art additive kernel SVM training methods. The same algorithm is used for all members in the kernel family, which can also be generalized to other additive kernels.

- Accurate classification results (Sec. 5.) The PmSVM algorithm achieves higher classification accuracy rates on large scale benchmark datasets in computer vision. It always outperforms the linear SVM by a large margin, and the accuracy is also slightly but consistently higher than recently proposed additive kernel classifiers.

2. Related Work

Before presenting PmSVM, we briefly review related work in large scale visual classification. We restrict our attention to linear and additive kernel SVM classifiers, as they are the most commonly used in the literature.

Since general purpose kernel SVM solvers (e.g., the SMO algorithm) may take years to complete in large scale datasets, special structures in the linear (dot-product) kernel are exploited to achieve speedup at even a few orders of magnitudes. Two typical examples are the coordinate descent algorithm [10] in LIBLINEAR, and the Pegasos algorithm [22] that performs stochastic gradient descent. In a linear SVM, the gradient (in dual problem) can be computed using only one dot-product evaluation (cf. Algorithm 1), which makes coordinate descent very efficient. Similarly, the optimal boundary has the same number of dimensions as the input, which facilitates stochastic gradient descent.

Linear classifiers have high accuracy rates on very high-dimensional inputs, such as text processing tasks (usually with millions of dimensions). Vision problems, however, mostly have moderate (e.g., a few thousands) feature dimensions. Recently, additive kernels have become popular choices in computer vision because they achieve a tradeoff between high accuracy, moderate training speed, and high testing speed.

An additive kernel has the form $\kappa(\mathbf{x}, \mathbf{y}) = \sum_i \kappa(x_i, y_i)$. The dot-product kernel is in fact an additive kernel, and additive kernel SVM solvers draw upon the efficient linear solvers, which are enabled by explicit or implicit feature mappings. The explicit feature mapping approach in [24] finds an approximate mapping function $\phi(\cdot)$ such that $\kappa(x_i, y_i) \approx \phi(x_i)^T \phi(y_i)$. It is shown that $\phi : \mathbb{R} \mapsto \mathbb{R}^3$ can accurately approximate the kernel function κ. Thus, a d-dimensional non-linear problem becomes a $3d$-dimensional linear one, and fast linear solvers can be directly applied. This approach is applicable to many additive kernels.

If we restrain the values x_i to be integers $\leq \bar{v}$, an obvious mapping for the HIK is $\mathbb{N} \mapsto \mathbb{R}^{\bar{v}}$ where the first x_i mapped values are 1 and the rest being 0. The coordinate descent algorithm is revised in [26] to efficiently handle this mapping indirectly, by using a pre-computation strategy. The approximation in [26] is that the decimal fractions in values x_i are ignored. The mapping in [15] revives the fractions as an additional dimension in the mapping, and leads to a quadratic optimization problem similar to that in normal SVM. The Pegasos algorithm is utilized in [15] to solve it. Both methods are limited to the histogram intersection kernel.

Another approach [18] directly maps the entire vector $\mathbf{x} \in \mathbb{R}^d$ to $\hat{\mathbf{x}} = \phi(\mathbf{x}) \in \mathbb{R}^E$ using kernel PCA, such that $\kappa(\mathbf{x}, \mathbf{y}) \approx \hat{\mathbf{x}}^T \hat{\mathbf{y}}$ and $E \ll d$. The problem then reduces to a linear classification one. The kernel PCA step depends on the training set, and is applicable to many additive kernels.

The aforementioned methods are empirically compared in [24] and [26]. They have similar accuracy rates but the methods in [24, 26] are faster than other ones. In this paper, instead of approximating the feature mapping, we show that we can instead approximate the gradient calculation in the coordinate descent approach. We will present the proposed method in the following section, and will empirically compare it with the linear classifier in LIBLINEAR and the additive kernel SVM in [24, 26].

3. The Power Mean Kernel

3.1. The Power Mean Function and Kernel

In mathematics, a power mean function M_p can be defined for any real number $p \in \mathbb{R}$ and a set of positive numbers $x_1, \ldots, x_n \in \mathbb{R}_+$, as

$$M_p(x_1, \ldots, x_n) = \left(\frac{\sum_{i=1}^n x_i^p}{n} \right)^{1/p} . \tag{1}$$

This function is also well-defined for three special cases: $p = -\infty$, 0, and ∞. Their values are $\min(x_1, \ldots, x_n)$, $\sqrt[n]{\prod_{i=1}^n x_i}$, and $\max(x_1, \ldots, x_n)$, respectively.

One important property of the power mean function is that *many additive kernels are special cases of it*, including:

- The χ^2 kernel, since $M_{-1}(x, y) = \kappa_{\chi^2}(x, y) = \frac{2xy}{x+y}$;
- Histogram Intersection Kernel (abbreviated as HIK), since $M_{-\infty}(x, y) = \kappa_{\text{HI}}(x, y) = \min(x, y)$;
- Hellinger's kernel, since $M_0(x, y) = \sqrt{xy}$.

Thus, we propose a power mean kernel for two vectors $\mathbf{x}$ and $\mathbf{y} \in \mathbb{R}_+^d$ based on the power mean function, which generalizes the three aforementioned additive kernels:

$$M_p(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^d M_p(x_i, y_i) . \tag{2}$$

The power mean kernel is indeed a positive definite kernel when $-\infty \leq p \leq 0$. The special case $p = -\infty$ has already been proved in [26], for the histogram intersection

kernel on $\mathbb{R}_+$. Another special case $p = 0$ is trivial, since $M_p(\mathbf{x}, \mathbf{y}) = (\sqrt{\mathbf{x}})^T(\sqrt{\mathbf{y}})$, where $\sqrt{\mathbf{x}} = (\sqrt{x_1}, \ldots, \sqrt{x_d})$.

Note that the power mean kernel is covered by a broader family of positive definite metric kernels [9]. For the sake of completeness, we still outline a proof here, using the following proposition:

(Proposition 2, [2]) If a kernel κ is conditionally positive definite and negative valued, then $\frac{1}{(-\kappa)^\delta}$ is positive definite for all $\delta \geq 0$.

A function κ is called a conditionally positive definite (cpd) kernel if for all $c_1, \ldots, c_n \in \mathbb{R}$ satisfying $\sum_{i=1}^n c_i = 0$, $\sum_{i,j=1}^n c_i c_j \kappa(x_i, x_j) \geq 0$ holds for arbitrary $x_1, \ldots, x_n$.

Theorem 1. *$M_p(\mathbf{x}, \mathbf{y}) : \mathbb{R}_+^d \times \mathbb{R}_+^d \mapsto \mathbb{R}$ is a positive definite kernel when $-\infty \leq p \leq 0$.*

Proof. We start the proof from the scalar version $M_p(x, y)$ (or, $d = 1$).

It is obvious that $\kappa_1(x, y) = x + y$ is a cpd kernel. We can easily verify that $\sum_{i,j=1}^n c_i c_j (x_i + x_j) = 0$, since $\sum_{i,j} c_i c_j x_i = \sum_i (c_i x_i (\sum_j c_j)) = 0$ and similarly $\sum_{i,j} c_i c_j x_j = 0$ when $\sum_j c_j = 0$.

When we restrict the domain for all variables to positive real numbers, the function $\kappa_2(x, y) = -\frac{x^p + y^p}{2}$ is well-defined and also trivially cpd for any $-\infty < p < 0$.

κ_2 is negative valued, thus the conditions of the proposition are satisfied. Applying the proposition with $\delta = -\frac{1}{p} > 0$, we conclude that

$$M_p(x, y) = \left(\frac{x^p + y^p}{2}\right)^{1/p} = 1/(-\kappa_2(x, y))^\delta$$

is positive definite. Together with the already proven special cases $p = -\infty$ and $p = 0$, $M_p(x, y)$ is positive definite when $-\infty \leq p \leq 0$.

The final step is to generalize from the scalar version $M_p(x, y)$ to the vector version $M_p(\mathbf{x}, \mathbf{y})$. Since $M_p(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^d M_p(x_i, y_i)$, this step is trivial. □

In practice, x^p may not be defined if $x = 0$ and $p < 0$, we will replace all 0^p with ϵ^p where ϵ is a small positive number (e.g., we use $\epsilon = 0.001$ in our implementation).

4. Efficient PmSVM Learning

Since the power mean kernel generalizes many kernels in the additive kernel family, an efficient algorithm for solving power mean kernel SVM also leads to efficient HIK or χ^2 SVM solvers. In this section, we propose PmSVM, an efficient SVM solver for all p values.

Given a power mean kernel M_p, its associated feature mapping ϕ, and a set of training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, we fix our attention to classifiers of the form $\mathrm{sgn}(\mathbf{w}^T\phi(\mathbf{x}))$, i.e., a linear classifier in the feature space without a bias term. We solve the following dual SVM problem (where $\alpha = (\alpha_1, \ldots, \alpha_n)$ are the Lagrange multipliers) [10]:

$$\min_\alpha \; f(\alpha) = \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j M_p(\mathbf{x}_i, \mathbf{x}_j) - \sum_i \alpha_i$$
$$\text{s.t. } 0 \leq \alpha_i \leq C \quad (i = 1, \ldots, n) \; . \tag{3}$$

The classifier boundary is then $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \phi(\mathbf{x}_i)$.

Algorithm 1 The Coordinate Descent Method

1: Given α and the corresponding $\mathbf{w} = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)$.
2: Compute $Q_{ii} = \|\phi(\mathbf{x_i})\|_{\ell_2}^2, \forall\, i = 1, \ldots, n$.
3: **while** α is not optimal **do**
4: **for** $i = 1, \ldots, n$ **do**
5: Compute $G = y_i \mathbf{w}^T \phi(\mathbf{x}_i) - 1$.
6: $\bar{\alpha}_i \leftarrow \alpha_i$.
7: $\alpha_i \leftarrow \min(\max(\alpha_i - G/Q_{ii}, 0), C)$.
8: $\mathbf{w} \leftarrow \mathbf{w} + y_i(\alpha_i - \bar{\alpha}_i)\phi(\mathbf{x}_i)$.
9: **end for**
10: **end while**

4.1. The Coordinate Descent Method

As suggested in [30], Eq. 3 can be efficiently solved by the coordinate descent method in Algorithm 1, by modifying the Algorithm 3 of [30] into a kernel version (and, plus other necessary notation changes).

The key of Algorithm 1 is the ability to efficiently compute the gradient (line 5) and to update the classifier $\mathbf{w}$ (line 8). For non-linear kernels, both tasks are generally very time-consuming.

The gradient G measures the changes in $f(\alpha)$ with respect to α_i (in which i is fixed), which equals

$$G = y_i \mathbf{w}^T \phi(\mathbf{x}_i) - 1 = y_i \sum_{j=1}^n \alpha_j y_j M_p(\mathbf{x}_i, \mathbf{x}_j) - 1 \; ,$$

and requires $O(nd)$ steps to compute. And, when the mapping ϕ is not explicit, the update step is not even feasible.

In this paper, we show that *the gradient can be approximated with only a small cost that is linear in d, i.e., within $O(d)$ steps*. Our approximation strategy also leads to a sufficient statistic for $\mathbf{w}$, and finally contributes to a very efficient power mean kernel SVM solver.

4.2. Gradient Approximation

The main component in line 5 of Algorithm 1 is the term

$$g(\mathbf{x}_i) = \sum_{j=1}^n \alpha_j y_j M_p(\mathbf{x}_i, \mathbf{x}_j) \; . \tag{4}$$

If we denote the j-th dimension of $\mathbf{x}_i$ as $x_{i,j}$, and denote the summation for the j-th dimension as a function g_j with

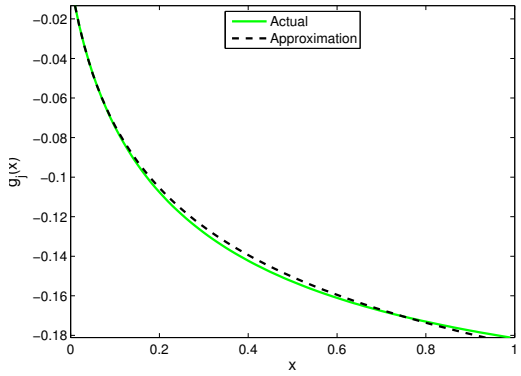

Figure 1. The $g_j(x)$ function and its approximation.

a single scalar input

$$g_j(x) = \sum_{t=1}^{n} \alpha_t y_t M_p(x, x_{t,j}) \ , \tag{5}$$

then obviously $g(\mathbf{x}) = \sum_{j=1}^{d} g_j(x_j)$ for any $\mathbf{x} \in \mathbb{R}_+^d$. Thus, we only need to efficiently calculate (or approximate) such scalar functions $g_j(x)$ for all j.

In fact, for all j, $g_j(x)$ can be accurately approximated in $O(1)$ steps. The green solid curve in Fig. 1 shows an example of g_j in the 15 class scene recognition dataset [11], with a random j, random α (uniform in $[0\ \ 1]$), and $p = -1$. The examples are generated using the bag-of-visual-words implementation in libHIK [26], and then normalized to the range $[0\ \ 1]$. We set the label $y = +1$ for bedroom images and $y = -1$ for all other images. There are far more negative examples than positive ones, so $g_j(x)$ values are negative in this example.

This curve illustrates a nice property of $g_j(x)$: for all j, it is a smooth and monotone function. It also has a limited range $[-1\ \ 1]$ when the input x is normalized. Empirical results on other datasets, α values, p values, and different choices of j also exhibit similar curves and properties.

Consequently, *we can use polynomial regression with very low degree of freedom to accurately approximate this curve.* A degree m polynomial regression model for $g_j(x)$ has parameters $\mathbf{a}_j = (a_{j,0}, \ldots, a_{j,m})$, and it estimates $g_j(x)$ as

$$g_j(x) \approx \sum_{k=0}^{m} a_{j,k} x^k \ . \tag{6}$$

At least $m+1$ distinct points are required to learn the parameters $\mathbf{a}_j$. Here we use exactly $m+1$ points. A vector $\mathbf{c}$ with $m+1$ values $c_0, \ldots, c_m$ within the range $[0\ \ 1]$ are sampled and their corresponding $g_j(c_i)$ are also calculated to form a vector $g_j(\mathbf{c})$. It is well known that the optimal parameters (in the least square sense) are

$$\mathbf{a}_j = X^{-1} g_j(\mathbf{c}) \ , \tag{7}$$

Algorithm 2 Fitting Polynomial Regression Parameters $\mathbf{a}_j$

1: $\mathbf{c} \leftarrow [0.01, 0.06, 0.75]$.
2: Compute $g_j(\mathbf{c})$ using a dataset and a specific j value.
3: $X_{kt} = (\ln(c_k + 0.05))^t, 0 \le k, t \le m = 2$.
4: $\mathbf{a}_j = X^{-1} g_j(\mathbf{c})$.

in which X is a Vandermonde matrix with $X_{kt} = (c_k)^t$, $0 \le k \le m$, $0 \le t \le m$. X is invertible when $\mathbf{c}$ contains distinct values.

As shown in Fig. 1, the shape of $g_j(x)$ resembles a quadratic function. Indeed, a quadratic function (degree two polynomial) can approximate $g_j(x)$ very accurately. Empirically, we use three points $\mathbf{c} = [0.01, 0.06, 0.75]$ in our approximation, which gives satisfactory approximation performance in experiments. Another observation from our experiments is that we should use $\ln(\mathbf{c} + 0.05)$ instead of $\mathbf{c}$ as the input to get better approximation. In summary, the polynomial regression model is learned using Algorithm 2. Note that the matrix X is now defined by $X_{kt} = (\ln(c_k + 0.05))^t$, and this matrix is still invertible.

The black dashed curve in Fig. 1 shows the approximation result of Algorithm 2. It is almost indistinguishable to the green ground-truth curve. In the entire $[0\ \ 1]$ range, the maximum relative difference between the two curves is less than 2%. In other words, Algorithm 2 accurately approximates $g_j(x)$ using only three parameters.

4.3. The PmSVM Algorithm

Suppose the j-th dimension is approximated with parameters $\mathbf{a}_j = [a_{j,0}, a_{j,1}, a_{j,2}]$, the set of values $\{\mathbf{a}_j\}_{j=1}^d$ is a sufficient statistic for the $\mathbf{w}$ in Algorithm 1. It captures the important information of $\mathbf{w}$, and thus, can replace it.

We can calculate the gradient G as follows,

$$G = y_i g(\mathbf{x_i}) - 1 \ , \tag{8}$$

in which

$$g(\mathbf{x}_i) = \sum_{j=1}^{d} g_j(x_{i,j}) = \sum_{j=1}^{d} \sum_{k=0}^{2} a_{j,k} (\ln(x_{i,j} + 0.05))^k \ . \tag{9}$$

For a new example $\mathbf{q}$, we can classify it according to

$$\mathbf{w}^T \phi(\mathbf{q}) = \sum_{j=1}^{n} \alpha_j y_j M_p(\mathbf{q}, \mathbf{x}_j) = g(\mathbf{q}) \ , \tag{10}$$

which can utilize Eq. 9 again.

Eq. 9 is $O(d)$, which means that both testing and gradient computing are very efficient. In order to design an extremely efficient training algorithm, the only missing component is a way to update the set of values $\{\mathbf{a}_j\}$ after updating α_i (i.e., to replace line 8).

Algorithm 3 PmSVM: The Power Mean SVM

$\alpha_i \leftarrow 0, 1 \leq i \leq n$.
$a_{j,k} \leftarrow 0, 1 \leq j \leq d, 0 \leq k \leq 2$.
$Q_{ii} \leftarrow \|\mathbf{x}_i\|_{\ell_1}, 1 \leq i \leq n$.
while α is not optimal **do**
  for $i = 1, \ldots, n$ **do**
    Compute $G = g(\mathbf{x}_i)$ using Eq. 9.
    $\bar{\alpha}_i \leftarrow \alpha_i$.
    $\alpha_i \leftarrow \min(\max(\alpha_i - G/Q_{ii}, 0), C)$.
    $\mathbf{a}_j \leftarrow \mathbf{a}_j + (\alpha_i - \bar{\alpha}_i) y_i X^{-1} M_p(\mathbf{c}, x_{i,j}), \forall\, j$.
  end for
end while
Output. The set of values $\{\mathbf{a}_j\}$.
Classification. For a test example $\mathbf{q} \in \mathbb{R}^d_+$, the classification result is:

$$\mathrm{sgn}(g(\mathbf{q})) = \mathrm{sgn}\left(\sum_{j=1}^{d}\sum_{k=0}^{2} a_{j,k} \left(\ln(q_j + 0.05)\right)^k\right) .$$

Let the changes in α_i be denoted as $\Delta\alpha_i = \alpha_i - \bar{\alpha}_i$. For all $1 \leq j \leq d$, we have

$$\Delta\mathbf{a}_j = X^{-1}\Delta g_j(\mathbf{c}) = \Delta\alpha_i y_i X^{-1} M_p(\mathbf{c}, x_{i,j}) , \quad (11)$$

in which $M_p(\mathbf{c}, x_{i,j})$ is a vector with 3 numbers, formed by concatenating $M_p(c_k, x_{i,j})$, $k = 0, 1, 2$. The two equalities follow from Eq. 7 and 5, respectively. Note that the matrix X^{-1} and the quantity $M_p(c_k, x_{i,j})$ can both be pre-computed and stored, since they only involve either constant values or the training examples. With pre-computed values, Eq. 11 only requires 18 multiplications or summations (since the same $\Delta\alpha_i y_i$ is shared for all j and k, and in our algorithm $m = 2$). As a summary, we propose the PmSVM (Power Mean SVM) method in Algorithm 3.

4.4. Practical Considerations

Both training and testing are very efficient in PmSVM. Although Eq. 11 is a few times more expensive than its counterpart in a linear SVM, we will show by experiments that PmSVM converges using only a tiny fraction of iterations of that of LIBLINEAR. Thus PmSVM are usually much faster than LIBLINEAR in large vision problems (and also other additive kernel SVM solvers). The non-linear additive power mean kernel also leads to significantly higher accuracies than linear SVM in practice.

In PmSVM, $p = -1$ is exactly equivalent to the χ^2 kernel. However, we do not need to use $p = -\infty$ for HIK. In practice, $p = -8$ is accurate enough to simulate the histogram intersection kernel.

It is also important to note that Algorithm 3 is not restricted to be used together with the power mean kernel. In fact, it can be used with any additive kernel so long as the function $g_j(x)$ is smooth and finite.[2] One example beyond the power mean kernel is the Jensen-Shannon's kernel $\kappa_{\mathrm{JS}}(x, y) = \frac{x}{2}\log_2\frac{x+y}{x} + \frac{y}{2}\log_2\frac{x+y}{y}$ [9, 24].

Before presenting experimental results, we conclude this section by considering a few practical issues for implementing PmSVM. Besides $x_{i,j}$, we pre-compute $\ln(x_{i,j} + 0.05)$ and $M_p(c_k, x_{i,j})$ for $k = 1, 2$. We assume $M_p(c_0, x_{i,j}) \approx c_0$ for all i and j to reduce the storage footprint. We use the `float` type (4 bytes) to store a real number. In a sparse format, storing one feature value, its feature index, and pre-computed values requires 20 bytes in total. In contrast, LIBLINEAR uses 16 bytes to store a feature value and feature index pair in a 64 bit computer. So PmSVM uses about 25% more memory than LIBLINEAR. The storage required for $\mathbf{a}_j$ is $3d$ in total, which is negligible.

When the pre-computed values can not be fit into the memory for huge datasets, we can compute $\ln(x_{i,j} + 0.05)$ and $M_p(c_k, x_{i,j})$ during each iteration (with longer training time though.)

5. Experimental Results

5.1. Datasets, Baselines, and Setups

PmSVM is compared with state-of-the-art methods on large scale vision problems, by measuring their training speed and accuracy. We do not present testing time in this paper, since per example testing time (excluding I/O time) is less than 50 milliseconds for all compared methods.

The following methods are compared on a computer with Intel Xeon 5670 CPU (only using one core) and 24G main memory:

PmSVM-χ^2. This is PmSVM with $p = -1$, equivalent to a χ^2 kernel SVM. We set $C = 0.01$.[3]

PmSVM-HI. This is PmSVM with $p = -8$, to simulate an HIK SVM. We set $C = 0.01$. Feature values are normalized to the range $[0\ 1]$ for both PmSVM setups.

LIBLINEAR. This is the linear SVM solver from [10], with default parameter value $C = 1$.

libHIK. This package provides an efficient HIK SVM solver for quantized histogram [26]. Following [26], we set $C = 0.01$ and set the quantization parameter $\bar{v} = 3$.

Feature mapping. We use the feature mapping approach [24] for both χ^2 (**fm-χ^2**) and HIK (**fm-HI**). Following [24], every $x_{i,j}$ is mapped to 3 dimensions during the training process, so the comparison to libHIK is fair. LIBLINEAR is used to classify mapped feature, with $C = 0.01$. We use the C source code from VLFeat for the mappings.

[2] This function does not necessarily need to be monotonic. In a non-monotone case, a larger m value may be needed.

[3] Note that it is not feasible to use cross-validation to choose an optimal C value for the large scale datasets in our experiments. Thus, we use a default C value recommended by respective methods. The source code of PmSVM will be published in our homepage.

Table 1. Results on ILSVRC 1000.

Method	Time (s)	Accuracy	Iterations
PmSVM-χ^2	**23,145**	25.57%	6 - 7
PmSVM-HI	23,299	**25.69**%	6 - 7
LIBLINEAR	122,688	21.13%	31 - 926
libHIK	37,677	18.62%	11 - 100
fm-χ^2	48,173	25.13%	6 - 7
fm-HI	47,781	25.06%	5 - 6

We use 1000 bins to pre-compute a lookup table for feature mapping in the range $[0\ 1]$, so that they do not need to be computed for every feature value during the runtime.

PmSVM aims at solving large scale visual classification problems, for which we choose the following three datasets:

ILSVRC 1000. This dataset has 1000 categories, and every image is encoded as a 21000 dimensional vector. We use the BOW feature set provided by [1], and restrict the training set to have ≤ 900 examples per category, in order to fit the training set into our 24G memory. The total number of training images is 887816. All test examples are used.

Indoor 67. This dataset contains 67 categories of indoor images [19]. We generated a bag-of-visual-words feature set from it using libHIK [26] with the CENTRIST descriptor, 2000 codewords, and parameters "use both, use one-class SVM, and grid step size 4" (62000 dimensional). We follow the train / test split of [19], thus 5360 training examples and 1340 test examples are used.

Caltech 101. This dataset contains 101 categories of object images [7]. We generated a feature set for it using libHIK with the SIFT descriptor, 2000 codewords, and parameters "use both, use one-class SVM, and grid step size 2" (62000 dimensional in total). We used 15 training examples and 20 test examples for every category.

The one-vs.-all strategy is used for all classifiers and we also compare our results with other published accuracy numbers on these datasets. Experimental results on the three datasets are presented in Sec. 5.2, 5.3, and 5.4, respectively. Finally, we show the advantages of different p values in Sec. 5.5.

5.2. ILSVRC 1000

Table 1 shows the results on the ILSVRC 1000 dataset, including training time in seconds, classification accuracy, and the minimum and maximum number of iterations before convergence for all the 1000 binary classifiers.

PmSVM is the fastest solver in Table 1. It only takes about 19% of the training time of LIBLINEAR, although PmSVM solves a non-linear SVM. As analyzed in Sec. 4.4, a single iteration of PmSVM is a little more expensive than LIBLINEAR. However, it uses either 6 or 7 iterations in all 1000 binary classifiers, while most LIBLINEAR classifiers converge in 80 to 400 iterations. PmSVM uses only less than half of the training time of feature mapping approaches. Between these two methods, the differences are caused by the complexity within every iteration.

Table 2. Results on indoor 67.

Method	Time (s)	Accuracy	Iterations
PmSVM-χ^2	**187**	46.20%	6 - 59
PmSVM-HI	213	**47.15**%	6 - 60
LIBLINEAR	698	41.78%	6 - 1000
libHIK	1,172	45.11%	6 - 516
fm-χ^2	318	46.19%	6 - 47
fm-HI	300	46.35%	6 -33
Parts based models [17]		43.1 %	
Pairwise codebook [16]		39.63%	
Object bank [12]		37.6 %	

PmSVM also has the highest accuracy.[4] The relative improvement compared to linear SVM is more than 20%. It has similar but slightly higher accuracies than feature mapping ones. libHIK, although being the second fastest, has the lowest accuracy in this dataset.

We used the feature set provided by the ILSVRC 2010 competition. The same feature set achieved an accuracy of approximately 19% in [5], which is far below the accuracies of PmSVM. Efforts have been made in extracting higher dimensional features (e.g., in [13, 21]), and we expect PmSVM to achieve higher accuracies using such more complex feature sets.

5.3. Indoor 67

Table 2 shows the results for the indoor 67 dataset. The accuracy is the mean value of accuracies in the 67 categories (i.e., average of the diagonal of the confusion matrix).[5]

Similar to results in Table 1, PmSVM-χ^2 is again the fastest SVM solver. It uses about 26% of the training time of LIBLINEAR, and about 60% of that of feature mapping.

PmSVM-HI is again the winner in terms of accuracy. All additive kernel classifiers (PmSVM, libHIK, and feature mapping) improves over linear SVM by a large margin. Comparing within the same kernel group, PmSVM-HI is the most accurate among all classifiers that use the HI kernel, and PmSVM-χ^2 has a very small advantage in χ^2 based classifiers.

In the final part of Table 2, we listed accuracy numbers from three recent researches. Our feature set for this dataset is bag-of-visual-words based. It is generated by the libHIK package, using techniques including the spatial pyramid matching, the codebook generated by histogram intersection kernel clustering, and the CENTRIST visual descriptor [27]. This feature set leads to the high classification accuracy: all additive kernel classifiers (PmSVM, libHIK,

[4] PmSVM-HI accuracy is 25.69%. This is a difficult dataset, and random guess accuracy is only 0.1%.

[5] There are 12 training and 6 testing images whose name ending with "_gif.jpg" that are not readable by OpenCV. We ignore such files.

Table 3. Results on Caltech 101 with 15 training images per class.

Method	Time (s)	Accuracy	Iterations
PmSVM-χ^2	100	72.08%	6 - 9
PmSVM-HI	115	**72.18**%	6 - 10
LIBLINEAR	**77**	68.41%	6 - 11
libHIK	167	71.63%	6 - 9
fm-χ^2	224	72.03%	6 - 12
fm-HI	221	71.94%	6 -10
Graph-matching kernel [6]		**75.3** %	
Localized soft-assignment [14]		74.21%	
NBNN+phow kernels [23]		69.2 %	

and feature mapping) applied to this feature set have 5% to 10% relative improvements compared to previous results.

5.4. Caltech 101

Table 3 shows results for the Caltech 101 dataset. LIBLINEAR has the shortest training time in Table 3. While PmSVM uses 30% to 50% more time than the linear SVM, its training time is less than 50% of that for feature mapping. libHIK is slower than PmSVM, but faster than feature mapping. On this dataset, LIBLINEAR is efficient because it does not use more iterations than other methods. Our conjecture is that Caltech 101 is an "easier to separate" dataset than the other two.

All non-linear classifiers have almost the same accuracy rates. libHIK, although faster than feature mapping and was highly accurate in [26], seems to be constrained by its quantization parameter $\bar{v}$. In some problems, a small $\bar{v}$ (e.g., $\bar{v} = 3$) is enough and libHIK has high accuracy rates (e.g., Caltech 101). In some other problems (e.g., ILSVRC 1000), a small $\bar{v}$ leads to lower accuracies.

Comparing with results using other feature sets at the bottom of Table 3, PmSVM accuracies are lower. The PmSVM classifier, however, may still achieve high accuracy if applied to these feature sets.

In summary, in three benchmark datasets, PmSVM-χ^2 consistently has the shortest training time among non-linear classifiers (and the fastest of all in two datasets). And, PmSVM-HI consistently has the highest accuracy rates among all classifiers. These observations not only verify the effectiveness of PmSVM, but also suggest the effect of p in PmSVM, for which we further study in Sec. 5.5.

5.5. Effect of p

PmSVM has shown consistent patterns in Sec. 5.2 to 5.4: χ^2 ($p = -1$) is the fastest and HIK ($p = -8$) is the most accurate. Using the indoor 67 dataset, Fig. 2 shows the effect of different p values in a larger range: $p = -2^{[-4\ 4]}$ (i.e., $\log_2(-p)$ gradually changes from -4 to 4 with step size 1.)

Training time trend. The training time curve (dotted green) has an obvious pattern if we ignore the point $p = -1$ (i.e., $\log_2(-p) = 0$): it is monotonically decreasing.

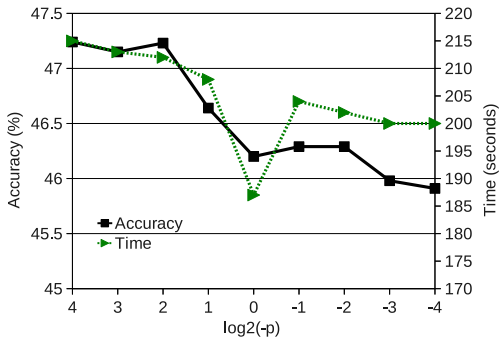

Figure 2. Training time and accuracy with different p values.

PmSVM with $p = -1$, in fact, is faster than $p > -1$ for a special reason. As discussed in Sec. 4.4, we pre-compute $M_p(c_k, x_{i,j})$ in PmSVM. Since M_{-1} involves computing $x^{-1} = 1/x$, M_{-1} is much faster to compute than other M_p values (which requires computing x^p and $x^{1/p}$, $p \neq \pm 1$). If we exclude the pre-computation time, the training time of PmSVM is indeed a monotone function of p.

Classification accuracy trend. The general trend is that accuracy drops while p increases (or equivalently, $\log_2(-p)$ decreases in the x-axis of Fig. 2.) This trend also coincides with results in Tables 1, 2 and 3.

From these limited amount of observations, we are able to recommend a rule of thumb for using PmSVM: choose $p = -1$ for faster speed, and $p = -8$ (or even smaller values) for higher accuracy rates.

6. Conclusions, Limitations and Future Work

In this paper, we propose the power mean kernel and the power mean SVM solver. The power mean kernel family include many popular additive kernels as special cases in this family. We also propose PmSVM, an efficient training method for power mean kernel SVM. PmSVM is not restricted to power mean kernels, but also other additive kernels. PmSVM combines the virtues of linear SVM and non-linear additive kernel SVM: its training time is up to 5 times faster than linear SVM in large vision problems, and it is more accurate than existing additive kernel SVM solvers. It achieves this performance by approximating the gradient computation efficiently through degree 2 polynomial regression, in the coordinate descent framework.

Experiments on PmSVM and other methods are tested on three benchmark problems. The largest one, ILSVRC 1000, has around 1 million examples and 21000 feature dimensions. PmSVM uses only 23 seconds to train a binary classifier on this dataset. However, when the dataset is even bigger and cannot be stored in the main memory, PmSVM will encounter a problem. In the future, we may explore an approach as that in [21]: compress the dataset and handle the compressed data on the fly. An alternative improving

direction is to utilize storage from the hard drive, as in [29].

PmSVM has shown excellent convergence speed in practice. However, we have not theoretically analyzed the effect of gradient approximation to its convergence. We will analyze its asymptotic convergence rate. One more interesting future research is the comparison of various additive kernels. In Sec. 5.5, we observe that HIK consistently outperforms χ^2 in terms of accuracy, on a limited set of benchmarks. It is of practical interest to compare various additive kernels using an extensive benchmark datasets.

Acknowledgements J. Wu would like to thank the anonymous CVPR reviewers for helpful comments, and the NTU startup grant and Singapore MoE AcRF Tier 1 RG 34/09 for support.

References

[1] Large scale visual recognition challenge 2010. http://www.image-net.org/challenges/LSVRC/2010/index.

[2] S. Boughorbel, J.-P. Tarel, and N. Boujemaa. Generalized histogram intersection kernel for image recognition. In *Proc. Int'l Conf. on Image Processing*, volume 3, pages 161–164, 2005.

[3] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. J. V. Gool. Robust tracking-by-detection using a detector confidence particle filter. In *The IEEE Conf. on Computer Vision*, pages 1515–1522, 2009.

[4] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40:5:1–5:60, 2008.

[5] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei. What does classifying more than 10, 000 image categories tell us? In *European Conf. Computer Vision*, pages 71–84, 2010.

[6] O. Duchenne, A. Joulin, and J. Ponce. A graph-matching kernel for object categorization. In *The IEEE Conf. on Computer Vision*, 2011.

[7] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training example: an incremental Bayesian approach tested on 101 object categories. In *CVPR 2004, Workshop on Generative-Model Based Vision*, 2004.

[8] P. F. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2008.

[9] M. Hein and O. Bousquet. Hilbertian metrics and positive denite kernels on probability measures. In *Proc. 10th Int'l Workshop on Articial Intelligence and Statistics*, pages 136–143, 2005.

[10] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Int'l Conf. on Machine Learning*, pages 408–415, 2008.

[11] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, volume II, pages 2169–2178, 2006.

[12] L.-J. Li, H. Su, E. Xing, and L. Fei-Fei. Object bank: A high-level image representation for scene classification & semantic feature sparsification. In *Advances in Neural Information Processing Systems*, pages 1378–1386, 2010.

[13] Y. Lin, F. Lv, S. Zhu, K. Yu, M. Yang, and T. Cour. Large-scale image classification: fast feature extraction and svm training. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2011. to appear.

[14] L. Liu, L. Wang, and X. Liu. In defense of soft-assignment coding. In *The IEEE Conf. on Computer Vision*, 2011.

[15] S. Maji and A. C. Berg. Max-margin additive classifiers for detection. In *The IEEE Conf. on Computer Vision*, pages 40–47, 2009.

[16] N. Morioka and S. Satoh. Building compact local pairwise codebook with joint feature space clustering. In *European Conf. Computer Vision*, LNCS 6311, pages 692–705, 2010.

[17] M. Pandey and S. Lazebnik. Scene recognition and weakly supervised object localization with deformable part-based models. In *The IEEE Conf. on Computer Vision*, 2011.

[18] F. Perronnin, J. Sánchez, and Y. Liu. Large-scale image categorization with explicit data embedding. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 2297–2304, 2010.

[19] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2009.

[20] X. Ren and J. Malik. Learning a classification model for segmentation. In *The IEEE Conf. on Computer Vision*, volume 1, pages 10–17. IEEE, 2003.

[21] J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 1665–1672. IEEE, 2011.

[22] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Int'l Conf. on Machine Learning*, pages 807–817, 2007.

[23] T. Tuytelaars, M. Fritz, K. Saenko, and T. Darrell. The NBNN kernel. In *The IEEE Conf. on Computer Vision*, 2011.

[24] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 3:480–492, 2012.

[25] P. Viola and M. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[26] J. Wu. A fast dual method for HIK SVM learning. In *European Conf. Computer Vision*, pages 552–565, 2010.

[27] J. Wu and J. M. Rehg. CENTRIST: A visual descriptor for scene categorization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 33(8):1489–1501, 2011.

[28] J. Wu, W.-C. Tan, and J. M. Rehg. Efficient and effective visual codebook generation using additive kernels. *Journal of Machine Learning Research*, 12(Nov):3097–3118, 2011.

[29] H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin. Large linear classification when data cannot fit in memory. In *ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pages 833–842, 2010.

[30] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. Recent advances of large-scale linear classification. *Proceedings of IEEE*, 2011.