# Practical Large Scale Classification with Additive Kernels

**Hao Yang** LANCELOT365@GMAIL.COM

**Jianxin Wu** WUJX2001@GMAIL.COM

*Nanyang Technological University, Singapore*

**Editor:** Steven C.H. Hoi and Wray Buntine

## Abstract

For classification problems with millions of training examples or dimensions, accuracy, training and testing speed and memory usage are the main concerns. Recent advances have allowed linear SVM to tackle problems with moderate time and space cost, but for many tasks in computer vision, additive kernels would have higher accuracies. In this paper, we propose the PmSVM-LUT algorithm that employs Look-Up Tables to boost the training and testing speed and save memory usage of additive kernel SVM classification, in order to meet the needs of large scale problems. The PmSVM-LUT algorithm is based on PmSVM (Wu, 2012), which employed polynomial approximation for the gradient function to speedup the dual coordinate descent method. We also analyze the polynomial approximation numerically to demonstrate its validity. Empirically, our algorithm is faster than PmSVM and feature mapping in many datasets with higher classification accuracies and can save up to 60% memory usage as well.

**Keywords:** Large scale classification, Polynomial approximation, Look-Up Table

## 1. Introduction

With the availability of abundant datum, images and videos on the Internet, datasets' size has grown at a rapid rate. As a matter of course, classification of large scale datasets with the support vector machine (SVM) raise more and more attention in the machine learning and computer vision community. Since many vision datasets are generated by the Bag-of-Words method, each training instance may have high dimension and dense features, making the classification even more tricky. Recent advances have allowed us to practically solve large scale linear kernel SVM problem, using tools such as LIBLINEAR (Fan et al., 2008) and Pegasos (Shalev-Shwartz et al., 2007). However, for visual classification tasks, non-linear kernels, especially additive kernels, have significantly higher accuracy than their linear rivalries.

A kernel is additive if the vector form can be written as the sum of the scalar kernel function of each dimension, i.e., if

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

Commonly used additive kernels are the linear kernel itself, the histogram intersection kernel

$$\kappa_{\text{HI}}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} \min(x_i, y_i), \tag{2}$$

and the $\chi^2$ kernel

$$\kappa_{\chi^2}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} \frac{2x_i y_i}{x_i + y_i}. \tag{3}$$

When learning an additive kernel, general purpose solvers like Sequential Minimal Optimization(SMO) (Platt, 1999) can be thousands times slower than fast linear solvers. Recent novel algorithms in Wu (2010); Perronnin et al. (2010); Vedaldi and Zisserman (2012) have bridged the gap. Additive kernel SVM now are only a few time slower than state-of-the-art linear SVM solvers; or even faster than linear SVM in some large vision problems. An particular example is the PmSVM algorithm in (Wu, 2012) to efficiently solve the power mean kernel and other additive kernels learning problem.

The power mean kernel is defined as

$$M_p(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} M_p(x_i, y_i) = \sum_{i=1}^{d} \left( \frac{x_i^p + y_i^p}{2} \right)^{1/p}. \tag{4}$$

It is the general form of many widely used additive kernels, such as the HIK (with $p = -\infty$) and the $\chi^2$ kernel (with $p = -1$). PmSVM combined the dual coordinate descent method with polynomial approximation for the gradient function of each dimension, resulting in an efficient algorithm that can be up to 10 times faster than the LIBLINEAR solver. However, some practical problems such as the polynomial approximation and the memory usage problem, need to be settled, in order for PmSVM to be applied to real large scale problems.

In this paper, we proposed PmSVM-LUT, a modified version of the PmSVM algorithm with Look-Up Table and an analysis on the polynomial approximation. Our main contributions are:

- We proposed PmSVM-LUT, the PmSVM algorithm with Look-Up Table to save memory usage and to further speed up the algorithm. Our algorithm can be up to 5 times faster than the original one using 50% less memory. We show the superiority of the new algorithm by comparing it to state-of-the-art methods on both large scale machine learning datasets and computer vision datasets.

- We demonstrate the approximability of the gradient function by polynomials. We show that by appropriately choosing interpolation nodes, the approximation polynomial function will converge to the gradient function as the polynomial's degree increases. We numerically analyze the approximation error to illustrate this convergence property.

The paper is organized as follows. We briefly review the related work on linear and nonlinear large scale SVM classification in Section 2. In Section 3, we introduce the PmSVM algorithm. In Section 4, we demonstrate the approximability of the gradient function and illustrate the way to choose appropriate interpolation nodes. We give a detailed description of PmSVM-LUT in Section 5 and compare it to PmSVM, feature mapping and LIBLINEAR experimentally in Section 6. The paper is concluded in Section 7.

## 2. Related Work

Before presenting our contributions, we briefly review related work on large scale SVM classification.

Early methods include sequential minimal optimization (Platt, 1999) and $SVM_{light}$ (Joachims, 1999). Both methods decompose a large problem into small sub-problems to reduce the memory usage. However, for large datasets nowadays with millions of training examples, these methods may take years to complete a training task.

There are many new algorithms proposed to speedup the training. For linear SVM problem, typical examples of efficient methods include the dual coordinate descent method in LIBLINEAR (Fan et al., 2008), which focuses on solving the dual problem; and the stochastic gradient descent method in Pegasos (Shalev-Shwartz et al., 2007), which focused on the primal problem. The dual coordinate descent method (see Algorithm 1) maintains $\boldsymbol{w} = \sum_{j=1}^{n} y_j \alpha_j \boldsymbol{x}_j$ so that the calculation of the gradient $\bigtriangledown_i f(\boldsymbol{\alpha}) = y_i \boldsymbol{w}^T \boldsymbol{x}_i - 1 + D_{ii}\alpha_i$ can be done in $O(d)$ time. Typical stochastic gradient descent (like Pegasos) or average stochastic gradient descent algorithm randomly chooses one training instance at a time and update the primal variable $\boldsymbol{w}$ according to the sub-gradient direction. The main advantage of SGD is that the training time is independent of the number of training instances. However, for SGD algorithms, it is not easy to explicitly specify a stopping criteria.

For the non-linear SVM, recent advances includes the Nyström approximation method (Zhang et al., 2008) and the Fourier approximation method (Rahimi and Recht, 2008). Pegasos has also been extended for non-linear kernels. In particular, for additive kernels, the explicit feature mapping method (Vedaldi and Zisserman, 2012), which uses a mapping function $\hat{\phi}(\cdot)$ such that $\kappa(x_i, y_i) \approx \hat{\phi}(x_i)^T \hat{\phi}(y_i)$ to map one dimension of an example into higher dimensions, achieves good experimental results. Experiments show that $\hat{\phi} : \mathbb{R} \mapsto \mathbb{R}^3$ achieves good approximation of the additive kernel function. In this case, a $d$-dimension non-linear problem becomes a $3d$-dimension linear problem and fast linear trainer such as LIBLINEAR can be applied to solve it.

## 3. The Power Mean SVM

Since PmSVM-LUT improves upon PmSVM, we first briefly introduce this algorithm, which is published in Wu (2012).

Given a set of training samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}, \boldsymbol{x}_i \in \mathbb{R}^d$, the dual SVM problem ($\ell_2$-regularized $\ell_1$-loss) without a bias term is defined as

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & f(\boldsymbol{\alpha}) = \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_i \alpha_i \\ \text{s.t.} \quad & 0 \le \alpha_i \le C, \forall i. \end{aligned} \tag{5}$$

Here $\kappa$ is the kernel function and if its associated feature mapping is defined as $\phi$, the classification boundary is then $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \phi(\boldsymbol{x}_i)$.

As suggested by Yuan et al. (2012), (5) can be solved by the dual coordinate descent framework (see Algorithm 1) with only a few modifications on the algorithm in Hsieh et al. (2008).

**Algorithm 1** The Coordinate Descent Method

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

However, unlike the linear SVM problem, which can maintain $\boldsymbol{w}$ efficiently to make the calculation of the gradient function $G$ fast, one can hardly compute $\boldsymbol{w}$ for a non-linear kernel explicitly. We need to address this problem to make it possible to apply this framework to non-linear kernel SVM.

Wu (2012) discovered that we can achieve fast training speed for the power mean kernel and other additive kernels by efficiently evaluating $\boldsymbol{w}^T\phi(\boldsymbol{x}_i)$ in the gradient function $G = y_i\boldsymbol{w}^T\phi(\boldsymbol{x}_i) - 1$. Wu (2012) defines

$$g(\boldsymbol{x}) = \boldsymbol{w}^T\phi(\boldsymbol{x}) = \sum_{t=1}^{n} \alpha_t y_t M_p(\boldsymbol{x}, \boldsymbol{x}_t), \tag{6}$$

and the $j$-th component of $g$ with a scalar input $x$ is then

$$g_j(x) = \sum_{t=1}^{n} \alpha_t y_t M_p(x, x_{t,j}), \tag{7}$$

in which $x_{t,j}$ is the $j$-th component of $\boldsymbol{x}_t$, and $g(\boldsymbol{x}) = \sum_{j=1}^{d} g_j(x_j)$. Wu (2012) showed that since $g_j(x)$ is a smooth function, it can be approximated by a low degree polynomial (degree two used). Thus, we can calculate $g_j(x)$ in constant time and update the gradient function in $O(d)$ steps. The PmSVM algorithm is shown in Algorithm 2.

In Algorithm 2, $\boldsymbol{c} = \{0.01, 0.06, 0.75\}$ are a set of pre-defined approximation nodes, and $X$ is the Vandermonde matrix used for polynomial interpolation:

$$X = \begin{bmatrix} 1 & \ln(c_0 + 0.05) & (\ln(c_0 + 0.05))^2 \\ 1 & \ln(c_1 + 0.05) & (\ln(c_1 + 0.05))^2 \\ 1 & \ln(c_2 + 0.05) & (\ln(c_2 + 0.05))^2 \end{bmatrix}. \tag{8}$$

$\boldsymbol{a}_j$ is the parameters for estimating $g_j$. PmSVM assumes that feature values are in the range $[0, 1]$. $\boldsymbol{a}_j$ is approximated as $\boldsymbol{a}_j = X^{-1}g_j(\boldsymbol{c})$, and is updated by $\Delta\boldsymbol{a}_j = X^{-1}\Delta g_j(\boldsymbol{c}) = \Delta\alpha_i y_i X^{-1} M_p(\boldsymbol{c}, x_{i,j})$ (see Wu (2012) for details.)

Although this algorithm is significantly faster than other state-of-the-art methods, there are also some drawbacks in this method. First of all, though the empirical results were good, Wu (2012) did not systematically analyze the error and convergence of polynomial

**Algorithm 2** The Power Mean SVM Algorithm (Wu, 2012)

```
1:  $\alpha_i \leftarrow 0, 1 \le i \le n$.
    $a_{j,k} \leftarrow 0, 1 \le j \le d, 0 \le k \le 2$.
2:  $Q_{ii} \leftarrow \|\boldsymbol{x}_i\|_{\ell_1}, 1 \le i \le n$.
3:  while $\alpha$ is not optimal do
4:    for $i = 1, ..., n$ do
5:      Compute $G = y_i g(\boldsymbol{x}_i) - 1$ using $g(\boldsymbol{x}_i) = \sum_{j=1}^{d} \sum_{k=0}^{2} a_{j,k} (\ln(x_{i,j} + 0.05))^k$
6:      $\bar{\alpha}_i \leftarrow \alpha_i$
7:      $\alpha_i \leftarrow \min(\max(\alpha_i - G/Q_{ii}, 0), C)$
8:      $\boldsymbol{a}_j \leftarrow \boldsymbol{a}_j + (\alpha_i - \bar{\alpha}_i) y_i X^{-1} M_p(\boldsymbol{c}, x_{i,j}), \forall j$
9:    end for
10: end while
11: Output: The set of values $\boldsymbol{a}_j$.
12: Classification: For a test example $\boldsymbol{q} \in \mathbb{R}_+^d$, the classification result is:
```

$$\operatorname{sgn}(g(\boldsymbol{q})) = \operatorname{sgn}\left(\sum_{j=1}^{d} \sum_{k=0}^{2} a_{j,k} (\ln(q_j + 0.05))^k\right). \tag{9}$$

approximation for the gradient function $g_j(x)$. Secondly, efficient implementation of the algorithm needs to pre-compute and store $\ln(x_{i,j} + 0.05)$ and $M_p(c_k, x_{i,j})$ for each training sample $x_{i,j}$, leading to about 30 percent more memory usage than LIBLINEAR and feature mapping. In this case Wu (2012) uses FLOAT type and the other two use DOUBLE. If all methods are using the DOUBLE, PmSVM may need 80 percent more memory. Last but not least, the logarithm manipulation in (9) and matrix multiplication in line 8 are time consuming and need to be further accelerated.

## 4. Polynomial Approximation of the Gradient

We propose to addresses these three drawbacks. We show the validity of polynomial approximation and the convergence by analyzing its approximation error numerically in this section. In Section 5, we propose PmSVM-LUT to solve the memory usage problem and further speedup the PmSVM algorithm.

### 4.1. Approximability of the gradient

Recall that $g_j(x)$ is defined in (7), which is $\mathbb{R} \mapsto \mathbb{R}$, and we assume that any feature value $x \in [0, 1]$. If the kernel function $\kappa(x, x_{t,j})$ is a continuous function (in the power mean case, it certainly is), by the Weierstrass approximation theorem (Burden and Douglas, 2004; Dzyadyk and Shevchuk, 2008), which states that

**Theorem 1** *If $f$ is a continuous real-valued function on $[a, b]$ and given any $\epsilon > 0$, there exists a polynomial $p$ on [a,b] such that* $\sup_{x \in [a,b]} |f(x) - p(x)| < \epsilon$, we can approximate any continuous function on the closed interval by some polynomials to any degree of accuracy. Therefore, we can approximate $g_j(x)$ by a polynomial, although we do not know how to choose such polynomials or the rate of convergence for now.

Note that although the proposed algorithm is based on the power mean kernel, we can actually approximate *any additive kernel as long as their scalar components are continuous and finite*, such as the Jensen-Shannon's kernel $\kappa_{JS} = \frac{x}{2}\log_2\frac{x+y}{x} + \frac{y}{2}\log_2\frac{x+y}{y}$.

If we define $P_m$ as the set that contains all polynomials with degree $\le m$, and

$$p_m^* = \inf_{p \in P_m} \sup_{x \in [a,b]} |f(x) - p(x)|\,, \tag{10}$$

we conclude from the Weierstrass approximation theorem that $p_m^*$ will converge to $f$ uniformly as $m$ increases. We call such a $p_m^*$ the best approximation polynomial of $f$.

The Chebyshev alternation theorem (Dzyadyk and Shevchuk, 2008) states an important property of the best approximation polynomial:

**Theorem 2** *Assume that a continuous real function $f$ is defined on $[a,b]$. In order that a polynomial $p_m^*$ of degree $\le m$ to be a polynomial of the best approximation of f, it is necessary and sufficient that there exists at least one system of $m+2$ points of $x_i$, $a \le x_1 < x_2 < ... < x_{m+2} \le b$, such that the difference of $f(x) - p_m^*(x) =: r_m(x)$*

- *consecutively takes alternating signs at the points $x_i$.*
- *attains its maximum absolute value on $[a,b]$ at the points $x_i$.*

Combine both theorems, we can get

**Theorem 3** *For any continuous function $f$ on a closed interval $[a,b]$, there exists a sequence of interpolation nodes $\{\boldsymbol{n}_1, \boldsymbol{n}_2, ...\}$ with $\boldsymbol{n}_i$ containing $i+1$ interpolation values, such that the sequence of polynomials $p_i$ interpolating $\boldsymbol{n}_i$ will converge uniformly to $f$.*

**Proof** By the Weierstrass approximation theorem, the sequence of the best polynomial will converge to $f$ uniformly. By the Chebyshev alternation theorem, such best polynomial of degree $m$ must oscillate between the function $m+2$ times, i.e., the best polynomial must have m+1 intersection points with the function. If we take these points as interpolation nodes, by the uniqueness of the polynomial, the interpolated polynomial is then the best polynomial, which will converge to $f$ uniformly. ■

This theorem tells us that if we choose a set of optimal nodes, the polynomials that interpolate these nodes will give us guaranteed approximation result of $g_j(x)$. However, searching for such optimal set of nodes is still one of the most intriguing topics in mathematics. Instead, we use a set of good nodes instead of the optimal ones.

## 4.2. Choosing Interpolation Nodes

PmSVM chose $\{0.01, 0.06, 0.75\}$ as the nodes for degree 2 polynomial interpolation. For the three datasets in Wu (2012), most feature values are in the range $[0.01, 0.10]$, and only a few feature values are above 0.8. However, this choice may not be good enough for other datasets that do not have such a property. Therefore, we want to find out a set of universally good nodes for every dataset, which is defined at any degree $m$ and can be used for higher order interpolation too.

For a function $f \in C^{m+1}[a,b]$, the error $e(x) = |f(x) - p_m(x)|$ satisfies (Burden and Douglas, 2004):

$$e(x) = \frac{f^{(m+1)}(\xi)}{(m+1)!} \left| \prod_{i=1}^{m+1} (x - x_i) \right|, \tag{11}$$

for chosen interpolation nodes $x_1, x_2, ..., x_{m+1}$ and some $\xi$ in $[a,b]$. Thus, it is logical for us to minimize

$$\max \left| \prod_{i=1}^{m+1} (x - x_i) \right|. \tag{12}$$

The Chebyshev nodes are the roots of the Chebyshev polynomial of the first kind $T_{m+1}$, which are defined as

$$x_i = \cos\left(\frac{2i-1}{2(m+1)}\pi\right), i = 1, ..., m+1, \tag{13}$$

for a closed interval $[-1,1]$. We use $m+1$ here because for a degree $m$ polynomial, we need $m+1$ nodes to interpolate. $T_{m+1}$ has the following property (Schwarz, 1989),

**Theorem 4** *Among all polynomials $p_{m+1}(x)$ of degree $m \geq 0$, whose coefficient of $x^{m+1}$ is equal to one, the polynomial $T_{m+1}(x)/2^m$ has the smallest maximum norm of the interval $[-1,1]$, that is, we have*

$$\min_{p_{m+1}(x)} \max_{x\in[-1,1]} |p_{m+1}(x)| = \max_{x\in[-1,1]} \left| \frac{T_{m+1}(x)}{2^m} \right| = \frac{1}{2^m}. \tag{14}$$

Since $\prod_{i=1}^{m+1}(x - x_i)$ is a polynomial whose coefficient of $x^{m+1}$ is equal to one, by Theorem 4, its maximum norm on $[-1,1]$ is a minimum if and only if the $m+1$ nodes coincide with the $m+1$ zeros of $T_{m+1}$. In this case, the polynomial $p_m^*(x)$ whose interpolation nodes are Chebyshev nodes of the $(m+1)$-th Chebyshev polynomial provides the smallest possible bound for the interpolation error

$$|f(x) - p_m^*(x)| \leq \frac{1}{2^m(m+1)!} \max_{\xi\in[-1,1]} \left| f^{(m+1)}(\xi) \right|. \tag{15}$$

This property holds for arbitrary closed interval $[a,b]$, since we only need to do an affine transform for the Chebyshev nodes to get

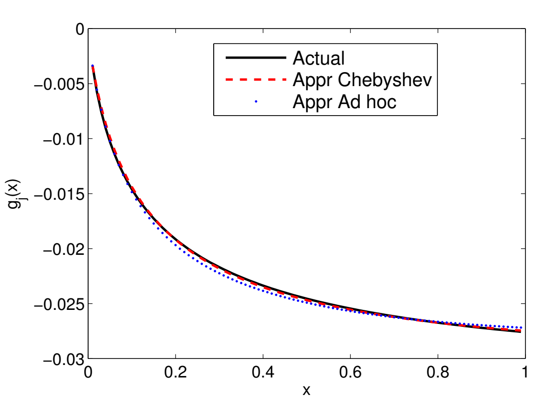
$$\tilde{x}_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2i-1}{2(m+1)}\pi\right), i = 1, ..., m+1. \tag{16}$$

Moreover, the Lebesgue constant $\Lambda_m$ grows only logarithmically if Chebyshev nodes are used, while it grows exponentially if equidistant nodes are used (Smith, 2006). That is,

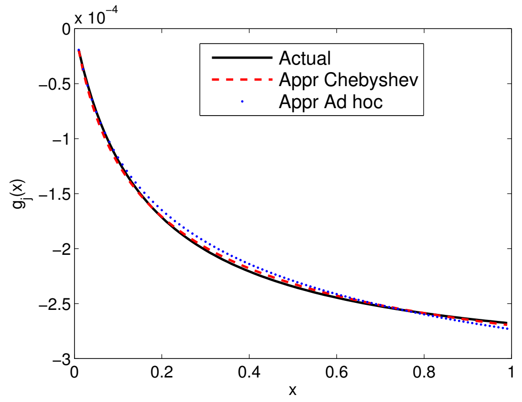
$$\frac{2}{\pi}\log(m+1) + a < \Lambda_m(T) < \frac{2}{\pi}\log(m+1) + 1, \tag{17}$$

for Chebyshev nodes, here $a \approx 0.9625$. It means that using Chebyshev nodes, the approximation polynomial is at most $\frac{2}{\pi}\log(m+1)+2$ worse than the best possible approximation. Overall, the Chebyshev nodes are good choices for polynomial approximation.

In Figure 1, we show the approximation error on some real world datasets with Chebyshev nodes and compare it with error using the ad hoc nodes $\{0.01, 0.06, 0.75\}$. Here we set $p=-1$ in the power mean kernel.



$(a)$



$(b)$

Figure 1: Comparing the approximation quality using Chebyshev nodes and ad hoc nodes $\{0.01, 0.06, 0.75\}$ on MNIST (a) and CALTECH 101 (b).

Table 1: Comparing Interpolation Error of Caltech 101 dataset.

| $j$ | Max Error (C) | Max Error (A) | Mean Error (C) | Mean Error (A) |
|---|---|---|---|---|
| 30 | 5.6e-4 (0.77%) | 8.2e-4 (0.87%) | 3.1e-4 (0.33%) | 4.5e-4 (0.45%) |
| 50 | 1.5e-3 (8.30%) | 2.1e-3 (5.31%) | 4.3e-4 (1.25%) | 1.1e-3 (2.73%) |
| 100 | 3.2e-5 (1.51%) | 5.6e-5 (1.25%) | 1.2e-4 (0.30%) | 2.7e-3 (0.58%) |

Table 2: Comparing Interpolation Error of MNIST dataset.

| $j$ | Max Error (C) | Max Error (A) | Mean Error (C) | Mean Error (A) |
|---|---|---|---|---|
| 30 | 2.2e-5 (1.78%) | 2.9e-5 (2.17%) | 8.8e-6 (0.88%) | 8.3e-6 (0.81%) |
| 50 | 9.3e-6 (2.03%) | 1.5e-5 (2.55%) | 4.7e-6 (1.33%) | 5.9e-6 (1.43%) |
| 100 | 7.8e-4 (1.56%) | 1.1e-3 (1.88%) | 3.4e-4 (0.85%) | 3.6e-4 (0.90%) |

As shown in Figure 1, approximation curves with Chebyshev nodes are obviously closer to the actual curve across the whole range while interpolation with previous ad hoc nodes put more emphasis on the range $[0.01, 0.10]$. Experimental results also show that Chebyshev nodes interpolation has generally smaller maximum error (which is predictable since the Chebyshev nodes provide the minimum bound for the maximum error) and average error than using the ad hoc nodes, as shown in Tables 1 and 2. The numbers in these tables are the actual error, followed by the relative error. (C) and (A) here mean Chebyshev and ad hoc nodes, respectively; and $j$ refers to the dimension of the training example as in $g_j(x)$. Classification accuracy also supports that, approximation with Chebyshev nodes give slightly better result in most cases, as shown in Table 3.

Table 3: Comparing SVM Classification Accuracy.

| | COVTYPE | URL | KDDA | KDDB | RCV |
|---|---|---|---|---|---|
| Chebyshev | **72.09%** | **98.77%** | 89.61% | **90.01%** | **97.92%** |
| Ad hoc | 71.14% | 98.74% | **89.62%** | 90.00% | 97.85% |

## 4.3. Choosing Polynomial Degree

Wu (2012) chooses degree 2 polynomials to approximate $g_j$, and experiments show that this approximation is good enough for some power mean kernels such as $\chi^2$ or HIK. However, we do not know how would the polynomial approximation behave as degree increases. We need to analyze if the polynomial converges to the function $g_j$; if it converges, can we get higher classification accuracy? Analyzing these issues analytically is very difficult, we will numerically analyze the behavior of the polynomial approximation as degree increases.We choose the Chebyshev nodes as interpolation nodes. As shown in Figure 2, as the degree

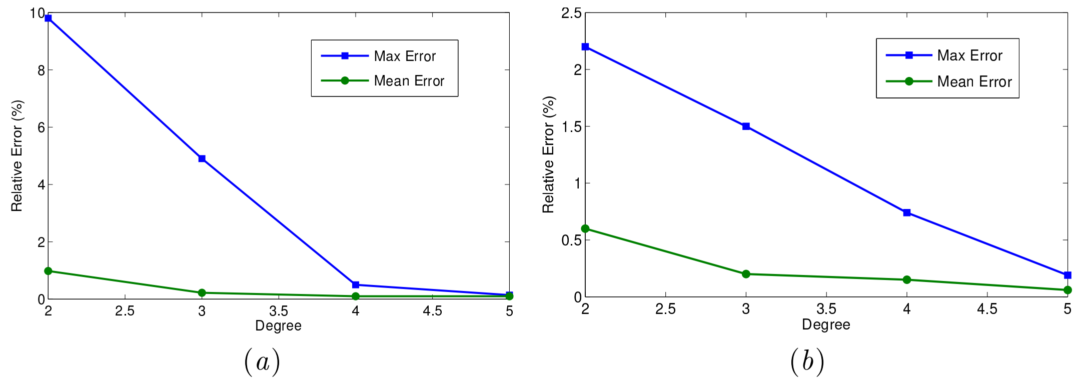


Figure 2: Relative approximation error in percentage with degree 2–5 polynomials on MNIST (a) and Caltech 101 (b).

increases, the maximum error and mean error become smaller. At degree 5, the maximum relative errors tested on 7 datasets are all below 0.5% and the mean relative errors are all below 0.1%. We conclude that, empirically, the approximation polynomial converges to the original function quickly.

Since the worst case error for degree 2 interpolation is already fairly small, further increasing the degree to minimize the error will not affect the accuracy much in many cases, However, the training time will increase as degree goes higher, as shown in Figure 3.

But in some cases, the average accuracy will improve visibly, as shown in Figure 4. In these cases, we may want to use higher order interpolation. Moreover, if the gradient function $g_j$ is not monotone, we also need higher order interpolation. Therefore, we need to improve the algorithm to cope with higher degree polynomial approximation.

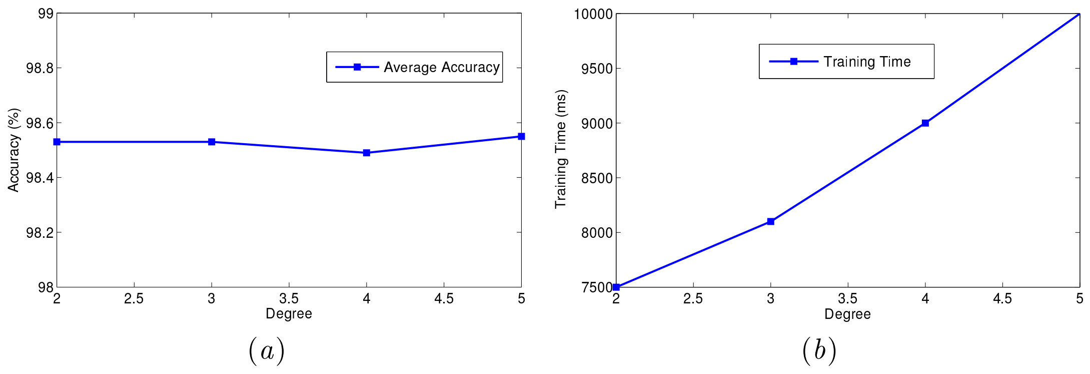


Figure 3: Average SVM classification accuracy (a) and training time (b) for URL as approximation degree increases.

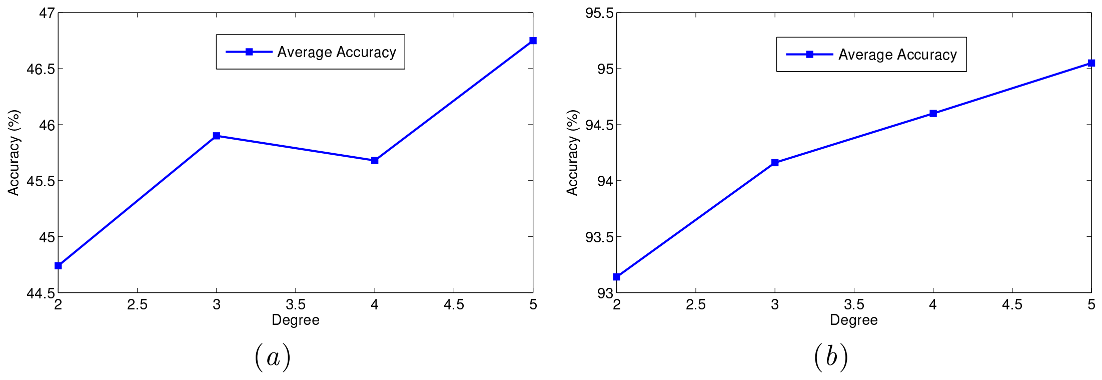


Figure 4: Average SVM Classification accuracy for COVTYPE (a) and WEBSPAM (b) as approximation degree increases.

## 5. Efficient Additive Kernel Learning

To implement Algorithm 2 efficiently, Wu (2012) pre-computes $\ln(x_{i,j}+0.05)$ and $M_p(c_k, x_{i,j})$ for $k = 1, 2$ with the assumption that $M_p(c_0, x_{i,j}) \approx c_0$. Without this pre-computation, the algorithm will be about 5–6 times slower. However, the pre-computation causes some problems. The biggest issue is that it uses 12 bytes more memory per training instance with FLOAT type employed, that is 150% more than not using pre-computation and 25% more than LIBLINEAR, which employs DOUBLE type. If one wants to approximate $g_j(x)$ with a larger degree, the memory usage with also grow linearly. The other issue is that $M_p(c_0, x_{i,j}) \approx c_0$ holds if $c_0 = 0.01$, but if one want to use different sets of nodes, this approximation could be inaccurate.

### 5.1. Coordinate Descent Method with Look-up Table

We use look-up tables to solve these problems and to further accelerate the algorithm. Since $\ln(x_{i,j}+0.05)$ is a scalar function with one variable $x_{i,j}$ and in practice we bound $x_{i,j} \in [0, 1]$, therefore, we can define a look-up table $\boldsymbol{T}_1$ such that each entry $T_{1,i}$, $1 \leq i \leq b$, satisfies:

$$T_{1,i} = \ln(\frac{i}{b} + 0.05), \tag{18}$$

where $b$ is the number of entries in the table. We can then map $x_{i,j}$ to the index of $\boldsymbol{T}_1$ by a trivial hash function $h(x) = \lfloor bx \rfloor$, where $\lfloor \cdot \rfloor$ is the floor function; and $T_{1,h(x_{i,j})}$ is an approximation of $\ln(x_{i,j} + 0.05)$.

Similarly, notice that $X^{-1}M_p(\boldsymbol{c}, x_{i,j})$ in Algorithm 2 line 8 is a $m+1$ dimensional vector for degree $m$ polynomial approximation. Since $X^{-1}M_p(\boldsymbol{c}, x_{i,j})$ only depends on a scalar variable $x_{i,j}$, which is bounded in $[0, 1]$, we can also use a look-up table $\boldsymbol{T}_2$ with $T_{2,i}$ defined as:

$$T_{2,i} = \begin{bmatrix} X^{-1}_{00}M_p(c_0, \frac{i}{b}) + ... + X^{-1}_{0m}M_p(c_m, \frac{i}{b}) \\ \vdots \\ X^{-1}_{m0}M_p(c_0, \frac{i}{b}) + ... + X^{-1}_{mm}M_p(c_m, \frac{i}{b}) \end{bmatrix} \tag{19}$$

to replace it. With the assistance of this look-up table, it is unnecessary to pre-compute and store $M_p(c_k, x_{i,j})$. We will also use the same trivial hash function $h(x)$ to map $x_{i,j}$ to the index in $\boldsymbol{T}_2$.

The worst case search time for hash function lookup is $O(1)$. If we choose $b = 1000$, and worst case errors that are defined as $E_1 = \max \left| T_{1,h(x_{i,j})} - \ln(x_{i,j} + 0.05) \right|$ and $E_2 = \max \left\| T_{2,h(x_{i,j})} - X^{-1}M_p(\boldsymbol{c}, x_{i,j}) \right\|_{\ell_1}$ are both less than 0.001 (0.03%), which is negligible. If we implement these tables with float type, they use $4b(m + 2)$ bytes memory. The time and storage cost for calculating and storing these look-up tables are are $O(mb)$, and can be neglected for large scale learning problems and the error caused by them are also very small. More importantly, calculating and storing the look-up tables are independent of the size of the dataset while the cost of computing and storing $\ln(x_{i,j} + 0.05)$ and $M_p(c_k, x_{i,j})$ will grow with the dataset size linearly. Therefore, with the look-up tables, we only need 8 bytes memory per instance using float type, which is only 40% of Wu's method and 50% of LIBLINEAR in a 64bit OS. Additionally, we also save the time of calculating the logarithm in $g(\boldsymbol{x_i})$ and simplify the matrix operations in calculating $X^{-1}M_p(\boldsymbol{c}, x_{i,j})$, leading to faster training.

Moreover, we can use $\boldsymbol{T}_1$ to accelerate the testing, too. Simply replacing $\ln(q_j + 0.05)$ in (9) of the original algorithm with $T_{1,h(q_j)}$ will save the time of calculating logarithm, leading to faster testing speed.

In summary, we propose PmSVM-LUT in Algorithm 3, which effectively solves the memory usage problem of Algorithm 2 with Look-Up Tables and further accelerates the training and testing speed. These features are extremely important for large scale classification.

### 5.2. Choosing bin number

We need to choose a bin number $b$. Choosing the bin number is important because it not only affects the training time, but also related to training accuracy. Apparently, a small $b$ will lead to large error and low training accuracy. We tested with $b = 10, 100, 1000, 10000$ on several datasets. Experiments show that although $b = 10, 100$ still give meaningful results, the classification accuracies are lower than $b = 1000, 10000$ for about $0.2{-}0.5\%$, and since the

**Algorithm 3** The Power Mean SVM with Look-up Table

```
1:  $\alpha_i \leftarrow 0, 1 \le i \le n$.
    $a_{j,k} \leftarrow 0, 1 \le j \le d, 0 \le k \le m$.
    Calculate $T_1$ and $T_2$ with given bin number $b$ and interpolation degree $m$.
2:  $Q_{ii} \leftarrow \|\boldsymbol{x}_i\|_{\ell_1}, 1 \le i \le n$.
3:  while $\alpha$ is not optimal do
4:    for $i = 1, ..., n$ do
5:      Compute $G = g(\boldsymbol{x}_i)$ using $g(\boldsymbol{x}_i) = \sum_{j=1}^{d}\sum_{k=0}^{m} a_{j,k}(T_{1,h(x_{i,j})})^k$
6:      $\bar{\alpha}_i \leftarrow \alpha_i$
7:      $\alpha_i \leftarrow \min(\max(\alpha_i - G/Q_{ii}, 0), C)$
8:      $\boldsymbol{a}_j \leftarrow \boldsymbol{a}_j + (\alpha_i - \bar{\alpha}_i) y_i T_{2,h(x_{i,j})}, \forall j$
9:    end for
10: end while
11: Output: The set of values $\boldsymbol{a}_j$.
12: Classification: For a test example $\boldsymbol{q} \in \mathbb{R}_+^d$, the classification result is:
```

$$\operatorname{sgn}(g(\boldsymbol{q})) = \operatorname{sgn}\left(\sum_{j=1}^{d}\sum_{k=0}^{m} a_{j,k}(T_{1,h(q_j)})^k\right). \tag{20}$$

approximation error of Look-Up Tables with $b = 1000$ is neglectable (less than 0.03%) and choosing $b = 10000$ or even larger bin number will not improve the accuracy, we just stick to $b = 1000$ in our experiment section. As we have mentioned before, the calculation time of the two LUTs with $b = 1000$ is neglectable (several milliseconds) compared to training time of large scale datasets, there is no need to implement more complex non-uniform LUTs.

## 6. Experimental Results

We compare PmSVM-LUT with state-of-the-art methods on large scale machine learning problems, including both binary and multi-class datasets. In particular, we compare the training time, testing time, memory usage and accuracy. These methods are compared on a computer with Intel Core i7 3930K CPU and 32GB memory. Only one CPU core is used in all experiments.

**PmSVM-LUT**. We set $p = -1$ for $\chi^2$ kernel (**PmSVM-LUT-$\chi^2$**) and $p = -8$ for HIK (**PmSVM-LUT-HI**). For both cases, $C$ is set to be 0.01, following Wu (2012). The interpolation nodes are Chebyshev nodes and degree is set to be 2.

**PmSVM**. We also set $p = -1$ for $\chi^2$ kernel (**PmSVM-$\chi^2$**) and $p = -8$ for HI kernel(**PmSVM-HI**). For both cases, $C$ is set to be 0.01.

**LIBLINEAR**. The linear solver by Fan et al. (2008), with default parameter $C = 1$.

**Feature Mapping with LUT**. We use the feature mapping with look-up table approach for both $\chi^2$ (**fm-$\chi^2$**) and HIK (**fm-HI**) (Vedaldi and Zisserman, 2012). Every $x_{i,j}$ is mapped to 3 dimensions during the training process. Note that we also use look-up table to accelerate the mapping. LIBLINEAR is used to classify the mapped features, with $C = 0.01$.

We choose to test our algorithm on the following large scale datasets, including 5 binary problems from the machine learning community and 3 multi-class datasets from the computer vision area:

**kdda, kkdb, url, rcv1, webspam**[1]: KDDA has 8.4M training instances and 20M features. KDDB contains 19M training examples and 30M features. URL contains 2.4M training examples and 3.2M features. RCV1 is a text categorization dataset. Since the testing set is much larger than training set, we use the testing set as training and vice versa. It then has 677K training examples. WEBSPAM has 350K training examples and 16M features. Since this dataset does not have a testing set, we report 5 fold cross validation result. The training time is just the average of the 5 runs.

**ILSVRC 1000**[2]: This dataset has 1000 categories and over $1,200,000$ images. Each image is encoded as a 21K dimensional vector.

**Indoor 67**: This dataset has 67 categories of indoor images. We follow Wu (2012) to generate a 62000 dimensional vector to represent each image using the CENTRIST (Wu and Rehg, 2011) descriptor and 2000 codewords. We also follow Quattoni and Torralba (2009) to split train/test examples, thus 5360 training examples and 1340 test examples are used.

**Caltech 101**: This datasets contains 101 categories of object images. We follow Wu (2012) to generate a 62000 dimensional vector to represent each image using SIFT descriptor and 2000 codewords. 15 training examples and 20 test examples are used for each category.

## 6.1. Comparing on large scale binary class datasets

Tables 4-7 show the results on 5 large scale binary datasets, including training time and testing time, classification accuracy (overall accuracy) and memory usage.

Table 4: Training time (seconds) on five binary datasets.

| Method | KDDA | KDDB | URL | RCV1 | WEBSPAM |
|---|---|---|---|---|---|
| PmSVM-LUT-$\chi^2$ | **55.0** | **123.0** | **7.5** | **2.2** | **0.9** |
| PmSVM-$\chi^2$ | 66.6 | 148.8 | 11.0 | 3.5 | 1.8 |
| fm-$\chi^2$ | 59.9 | 134.5 | 10.0 | 3.3 | 1.6 |
| PmSVM-LUT-HI | **55.2** | **131.0** | **7.5** | **2.1** | **0.9** |
| PmSVM-HI | 110.7 | 232.3 | 31.2 | 12.3 | 7.2 |
| fm-HI | 66.4 | 148.4 | 11.1 | 3.2 | 1.6 |
| LIBLINEAR | 1481.5 | 3537.8 | 127.2 | 7.1 | 5.8 |

In terms of training time, we can conclude that PmSVM-LUT is the fastest solver here. For $\chi^2$ kernel, the PmSVM-LUT method generally uses 80% to 50% of the training time of the original PmSVM method; for HIK, the PmSVM-LUT method uses only 50% to 12.5% of the training time. For memory usage, as we have analyzed previously, our method only uses about 45% memory of the original PmSVM, and 50% of feature mapping or LIBLINEAR. PmSVM-LUT also takes about 20% of the testing time with slight higher accuracy for most

1. The datasets are downloaded from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
2. The dataset is downloaded from http://www.image-net.org/challenges/LSVRC/2010/

Table 5: Classification accuracy on five binary datasets.

| Method | KDDA | KDDB | URL | RCV1 | WEBSPAM |
|---:|---:|---:|---:|---:|---:|
| PmSVM-LUT-$\chi^2$ | 89.61% | **90.01%** | **98.77%** | **97.85%** | 94.28% |
| PmSVM-$\chi^2$ | **89.62%** | 90.00% | 98.73% | 97.82% | **94.79%** |
| fm-$\chi^2$ | 89.60% | 89.97% | 98.62% | 97.60% | 93.73% |
| PmSVM-LUT-HI | 89.61% | **90.01%** | **98.77%** | **97.85%** | 93.63% |
| PmSVM-HI | **89.62%** | 90.00% | 98.74% | **97.85%** | **94.65%** |
| fm-HI | 89.56% | 89.97% | 98.55% | 97.57% | 93.54% |
| LIBLINEAR | 88.56% | 88.94% | 98.45% | 97.17% | 92.69% |

Table 6: Memory Usage (MB) on five binary datasets.

| Method | KDDA | KDDB | URL | RCV1 | WEBSPAM |
|---:|---:|---:|---:|---:|---:|
| PmSVM-LUT | **3267** | **6077** | **1199** | **530** | **280** |
| PmSVM | 6880 | 13005 | 2812 | 1007 | 504 |
| fm | 5853 | 10950 | 2304 | 828 | 501 |
| LIBLINEAR | 5542 | 10050 | 2253 | 825 | 498 |

Table 7: Testing Time (ms) on five binary datasets.

| Method | KDDA | KDDB | URL | RCV1 | WEBSPAM |
|---:|---:|---:|---:|---:|---:|
| PmSVM-LUT | **75** | **88** | **596** | **7** | N/A |
| PmSVM | 341 | 397 | 2545 | 27 | N/A |

of the binary class datasets. Compared to feature mapping with look-up table, our method is faster and has slightly higher accuracy.

Note that since LIBLINEAR reads one example and then test it, it is hard to separate testing time and I/O time without modifying the code, thus we cannot compare with the testing time of feature mapping and LIBLINEAR here.

PmSVM-LUT accelerate the training in two aspects comparing to PmSVM. First of all, we save the time of pre-computing $\ln(x_{i,j}+0.05)$ and $M_p(c_k, x_{i,j})$ for k = 1,2. For large scale binary class datasets, PmSVM will spend a lot of time on the pre-computation. Secondly, in line 8 of algorithm 1, we save the time of the matrix multiplication that involves 12 summations and multiplications.

## 6.2. Comparing on large scale multi-class datasets

For multi-class large scale classification problems, PmSVM-LUT still shows the virtue of memory saving with more than 50% less memory usage. In term of training time, our algorithm is not necessarily faster than PmSVM but has comparable speed, and is about 2 times faster than feature mapping with LUT.

| Method | (a) ILSVRC | (a) Caltech | (a) indoor | (b) ILSVRC | (b) Caltech | (b) indoor |
|---|---|---|---|---|---|---|
| PmSVM-LUT-$\chi^2$ | 24999 | 78 | 131 | **26.15%** | **72.05%** | **46.18%** |
| PmSVM-$\chi^2$ | **21791** | **71** | **128** | 26.11% | **72.05%** | 46.10% |
| fm-$\chi^2$ | 50687 | 151 | 208 | 25.67% | 72.00% | 45.95% |
| PmSVM-LUT-HI | 24961 | **82** | **144** | **26.30%** | **72.20%** | **46.85%** |
| PmSVM-HI | **22449** | 83 | 148 | 26.23% | **72.20%** | 46.78% |
| fm-HI | 50790 | 151 | 222 | 25.41% | 71.90% | 46.10% |
| LIBLINEAR | 136874 | 53 | 442 | 22.13% | 68.06% | 40.93% |

Table 8: Training time in seconds (a) and average accuracy (b) on vision datasets.

| Method | (a) ILSVRC | (a) Caltech | (a) indoor | (b) ILSVRC | (b) Caltech | (b) indoor |
|---|---|---|---|---|---|---|
| PmSVM-LUT | **12.20** | **0.58** | **0.65** | **254.8** | **10.0** | **2.5** |
| PmSVM | 30.00 | 1.21 | 1.39 | 256.9 | 10.8 | 2.7 |
| fm | 24.30 | 0.94 | 1.22 | | | |
| LIBLINEAR | 24.00 | 0.75 | 1.10 | | | |

Table 9: Memory usage in GB (a) and testing time in seconds (b) on vision datasets.

## 7. Conclusion

In this paper, we proposed the PmSVM-LUT algorithm, an algorithm based on PmSVM (Wu, 2012). PmSVM-LUT accelerates the training and testing speed and save memory usage by employing Look-Up Tables. We also show approximability of the gradient function by polynomials and conclude that the Chebyshev nodes is a good choice to interpolate such polynomials. By numerically analyzing the approximation errors, we demonstrate that for most of the datasets, approximation with Chebyshev nodes has good convergence property as degree increases. Experimental results show that our algorithm can be up to 5 times faster than the original PmSVM in binary-class case and has similar speed in multi-class case with only 50% memory usage. In the future, we may explore the possibility of employing similar approximation on other non-linear kernels, especially the multiplicative kernels.

## References

Ricard I. Burden and Faires J. Douglas. *Numerical Analysisi.* Brooks Cole, 2004.

Vladislav K. Dzyadyk and Igor A. Shevchuk. *Theory of Uniform Approximation of Functions by Polynomials.* Walter de Gruyter, 2008.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the 25th International Conference on Machine Learning*, pages 408–415, 2008.

Thorsten Joachims. Advances in kernel methods. chapter Making large-scale support vector machine learning practical, pages 169–184. MIT Press, Cambridge, MA, USA, 1999.

Florent Perronnin, Jorge Sánchez, and Yan Liu. Large-scale image categorization with explicit data embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2297–2304, 2010.

John C. Platt. Advances in kernel methods. chapter Fast training of support vector machines using sequential minimal optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 413–420, 2009.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pages 1177–1184, 2008.

Hans Rudolf Schwarz. *Numerical Analysis: A Comprehensive Introduction.* John Wiley & Sons, 1989.

Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *Proceedings of the 24th International Conference on Machine Learning*, pages 807–817, 2007.

Simon J. Smith. Lebesgue constants in polynomial interpolation. *Annales Mathematicae et Informaticae*, 33:109–123, 2006.

Andrea Vedaldi and Andrew Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):480–492, 2012.

Jianxin Wu. A fast dual method for HIK SVM learning. In *Proceedings of the 11th European Conference on Computer Vision*, pages 552–565, 2010.

Jianxin Wu. Power mean SVM for large scale visual classification. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2344–2351, 2012.

Jianxin Wu and James M. Rehg. CENTRIST: A visual descriptor for scene categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:1489–1501, 2011.

Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603, 2012.

Kai Zhang, Ivor W. Tsang, and James T. Kwok. Improved Nyström low-rank approximation and error analysis. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1232–1239, 2008.