Fast Prediction with SVM Models Containing RBF Kernels

Marc Claesen[1, 2]

Frank De Smet[3, 4]

Johan A.K. Suykens[1, 2]

Bart De Moor[1, 2]

[1]KU Leuven, Department of Electrical Engineering (ESAT), 3000 Leuven, Belgium

STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics

[2]iMinds Medical IT

[3]National Alliance of Christian Mutualities, 1031 Brussels, Belgium

[4]Department of Public Health and Primary Care, KU Leuven, Belgium

Keywords: kernel methods, support vector machines, radial basis function, Maclaurin series approximation

Abstract

We present an approximation scheme for support vector machine models that use an RBF kernel. A second-order Maclaurin series approximation is used for exponentials of inner products between support vectors and test instances. The approximation is applicable to all kernel methods featuring sums of kernel evaluations and makes no assumptions regarding data normalization. The prediction speed of approximated models no longer relates to the amount of support vectors but is quadratic in terms of the number of input dimensions. If the number of input dimensions is small compared to the amount of support vectors, the approximated model is significantly faster in prediction and has a smaller memory footprint. An optimized C++ implementation was made to assess the gain in prediction speed in a set of practical tests. We additionally provide a method to verify the approximation accuracy, prior to training models or during run-time, to ensure the loss in accuracy remains acceptable and within known bounds.

1 Introduction

Kernel methods form a popular class of machine learning techniques for various tasks. An important feature offered by kernel methods is the ability to model complex data through the use of the kernel trick (Schölkopf & Smola, 2002). The kernel trick allows the use of linear algorithms to implicitly operate in a transformed feature space, resulting in an efficient method to construct models which are nonlinear in input space. In practice, despite the computationally attractive kernel trick, the prediction complexity of models using nonlinear kernels may prohibit their use in favor of faster, though less accurate, linear methods.

We present an approach to reduce the computational cost of evaluating predictive nonlinear models based on RBF kernels. This is valuable in situations where model evaluations must be performed in a limited time span. Several applications in the computer vision domain feature such requirements, including object detection (Cao et al., 2008; Maji et al., 2013) and image denoising (Mika, Schölkopf, et al., 1999; Yang et al., 2004).

The widely used Radial Basis Function (RBF) kernel is known to perform well on a large variety of problems. It effectively maps the data onto an infinite-dimensional feature space. The RBF kernel function $\kappa(\cdot, \cdot)$ is defined as follows, with kernel parameter γ:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}. \tag{1}$$

Support vector machines (SVMs) are a prominent class of kernel methods for classification and regression problems (Burges, 1998). The decision functions of SVMs take a similar form for various types of SVM models, including classifiers, regressors and least squares formulations (Cortes & Vapnik, 1995; Suykens & Vandewalle, 1999). For lexical convenience, we will use common SVM terminology throughout the text.

The run-time complexity of kernel methods using an RBF kernel is $O(n_{SV} \times d)$ where n_{SV} is the number of support vectors and d is the input dimensionality. When run-time complexity is crucial and the number of support vectors is large, users are often forced to use linear methods which have $O(d)$ prediction complexity at the cost of reduced accuracy (Maji et al., 2013). We suggest a method which can significantly lower the run-time complexity of models with RBF kernels for many learning tasks.

We propose to approximate the decision function of SVM models that use an RBF

kernel via the second-order Maclaurin series approximation of the exponential function. Using this approximation, prediction speed can be increased significantly when the number of dimensions d is low compared to the number of support vectors n_{SV} in a model. The proposed approximation is applicable to all models using an RBF kernel in popular SVM packages like LIBSVM (Chang & Lin, 2011), SVM^{light} (Joachims, 1999), SHOGUN (Sonnenburg et al., 2010) and LS-SVMlab (De Brabanter et al., 2010).

We will derive the proposed approximation in the context of SVMs but its use easily extends to other kernel methods. The approximation is applicable to all kernel methods that exploit the representer theorem (Schölkopf et al., 2001). This includes methods such as Gaussian processes (Rasmussen & Williams, 2006), RBF networks (Poggio & Girosi, 1990), kernel clustering (Filippone et al., 2008), kernel PCA (Schölkopf, Smola, & Müller, 1998; Suykens et al., 2003) and kernel discriminant analysis (Mika, Ratsch, et al., 1999).

2 Related Work

A large variety of methods exist to increase prediction speed. Two main classes of approaches can be identified: (i) pruning support vectors from models and (ii) approximating the decision function of a given model. Our proposed approach belongs to the latter class. Complementary to those approaches, it is worth mentioning that older research attempted to increase evaluation speed for the exponential function because it represented a significant bottleneck for neural networks (Schraudolph, 1999). Those

techniques can also be applied to RBF kernel evaluations, though they do not change the big-O complexity of prediction.

2.1 Reducing Model Size by Pruning Support Vectors

Pruning support vectors linearly increases prediction speed because the run-time complexity of models with RBF kernels is proportional to the amount of support vectors. Pruning methods have been devised for SVM (Schölkopf, Knirsch, et al., 1998; Liang, 2010) and least squares SVM formulations (Suykens et al., 2002; Hoegaerts et al., 2004).

2.2 Direct Decision Function Approximations

Approaches that focus on approximating the decision function directly typically involve some form of approximation of the kernel function. Such approximations need not retain the structure and interpretation of the original model, provided that the decision function does not change significantly. Kernel approximations may leave out the interpretation of support vectors completely by reordering computations (Herbster, 2001), or by aggregating support vectors into more efficient structures (Cao et al., 2008).

Herbster (2001) devised a method for fast prediction with intersection kernels by reordering computations. Subsequently, further speed enhancements were derived for additive kernels – a superclass of intersection kernels – by approximating dimensions independently or by effectively computing parts of the solution in feature space (Vedaldi & Zisserman, 2010; Maji et al., 2013). Unfortunately, these approaches are not applicable to RBF kernels.

2.3 Second-Order Approximation for RBF Kernel

A second-order approximation of the exponential function for RBF kernels was first introduced by Cao et al. (2008). The basic concept of our paper resembles their work. Cao et al. (2008) make two assumptions regarding normalization in deriving the approximations that may needlessly constrain their applicability. These assumptions are:

1. Feature vectors are normalized to unit length, to simplify κ to $\kappa(\mathbf{x}_i, \mathbf{x}_j) = e^{-2\gamma} e^{2\gamma \mathbf{x}_i^T \mathbf{x}_j}$.

2. Feature values must always be positive such that $0 \leq \mathbf{x}_i^T \mathbf{z} \leq 1$ holds.

We will perform a more general derivation that requires none of these assumptions. Our derivation is agnostic to data normalization and we provide a conservative bound to assess the validity of the approximation during prediction (Eq. (11)). Additionally, we derive the full approximation in matrix-form using the gradient and Hessian of the approximated part of the decision function. This allows the use of highly optimized linear algebra libraries in implementations of our work. Our benchmarks demonstrate that the use of such libraries yields a significant speed-up. Finally, we freely provide our implementation.

3 Approximation

Predicting with SVMs involves computing a linear combination of inner products in feature space between the test instance $\mathbf{z} \in \mathbb{R}^d$ and all support vectors. In subsequent equations, $\mathbf{X} \in \mathbb{R}^{d \times n_{\mathrm{SV}}}$ represents a matrix of n_{SV} support vectors. We will denote the i-th support vector by $\mathbf{x}_i$ (the i-th column of $\mathbf{X}$). Via the representer theorem (Schölkopf

et al., 2001), the decision values are a linear combination of kernel evaluations between the test instance and all support vectors:

$$f(\cdot) : \mathbb{R}^d \to \mathbb{R} : f(\mathbf{z}) = \sum_{i=1}^{n_{\text{SV}}} \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{z}) + b, \tag{2}$$

where b is a bias term, α contains the support values, $\mathbf{y}$ contains the training labels and $\kappa(\cdot, \cdot)$ is the kernel function. Expanding the RBF kernel function (1) in Eq. (2) yields:

$$f(\mathbf{z}) = \sum_{i=1}^{n_{\text{SV}}} \alpha_i y_i e^{-\gamma\|\mathbf{x}_i - \mathbf{z}\|^2} + b = \sum_{i=1}^{n_{\text{SV}}} \alpha_i y_i e^{-\gamma\|\mathbf{x}_i\|^2} e^{-\gamma\|\mathbf{z}\|^2} \underbrace{e^{2\gamma \mathbf{x}_i^T \mathbf{z}}}_{} + b. \tag{3}$$

The exponentials of inner products between support vectors and the test instance – underbraced in Equation (3) – induce prediction complexity $O(n_{\text{SV}} \times d)$. Large models with many support vectors are slow in prediction, because each SV necessitates computing the exponential of an inner product in d dimensions for every test instance $\mathbf{z}$.

We propose a second-order Maclaurin series approximation for these exponentials to bypass the inner products (see the appendix for details on the Maclaurin series). The exponential per test instance $e^{-\gamma\|\mathbf{z}\|^2}$ can be computed exactly in $O(d)$. Before approximating the factors $e^{2\gamma \mathbf{x}_i^T \mathbf{z}}$, we reorder Equation (3) by moving the factor $e^{-\gamma\|\mathbf{z}\|^2}$ in front of the summation:

$$\begin{aligned} f(\mathbf{z}) &= e^{-\gamma\|\mathbf{z}\|^2} \big(\sum_{i=1}^{n_{\text{SV}}} \alpha_i y_i e^{-\gamma\|\mathbf{x}_i\|^2} e^{2\gamma \mathbf{x}_i^T \mathbf{z}} \big) + b, \\ &= e^{-\gamma\|\mathbf{z}\|^2} g(\mathbf{z}) + b, \end{aligned} \tag{4}$$

with:

$$g(\cdot) : \mathbb{R}^d \to \mathbb{R} : g(\mathbf{z}) = \sum_{i=1}^{n_{\text{SV}}} \alpha_i y_i e^{-\gamma\|\mathbf{x}_i\|^2} e^{2\gamma \mathbf{x}_i^T \mathbf{z}}. \tag{5}$$

The exponentials of inner products can be replaced by the following approximation, based on the second-order Maclaurin series of the exponential function (see the ap-

pendix):

$$e^{2\gamma \mathbf{x}_i^T \mathbf{z}} \approx 1 + 2\gamma \mathbf{x}_i^T \mathbf{z} + 2\gamma^2 (\mathbf{x}_i^T \mathbf{z})^2. \tag{6}$$

Approximating the exponentials $e^{2\gamma \mathbf{x}_i^T \mathbf{z}}$ in $g(\mathbf{z})$ (5) via Equation (6) yields:

$$\begin{aligned} \hat{g}(\mathbf{z}) &= \sum_{i=1}^{n_{\mathrm{SV}}} \alpha_i y_i e^{-\gamma \|\mathbf{x}_i\|^2} \big(1 + 2\gamma \mathbf{x}_i^T \mathbf{z} + 2\gamma^2 (\mathbf{x}_i^T \mathbf{z})^2\big), \\ &= c + \mathbf{v}^T \mathbf{z} + \mathbf{z}^T \mathbf{M} \mathbf{z}, \end{aligned} \tag{7}$$

with:

$$\begin{aligned} c \in \mathbb{R} = g(\mathbf{0}^d) &= \sum_{i=1}^{n_{\mathrm{SV}}} \alpha_i y_i e^{-\gamma \|\mathbf{x}_i\|^2}, \\ \mathbf{v} \in \mathbb{R}^d \rightarrow v_j = \nabla g(\mathbf{z}) &= 2\gamma \sum_{i=1}^{n_{\mathrm{SV}}} \alpha_i y_i e^{-\gamma \|\mathbf{x}_i\|^2} X_{j,i}, \\ \mathbf{v} &= \mathbf{X}\mathbf{w}, \\ \mathbf{M} \in \mathbb{R}^{d \times d} \rightarrow M_{j,k} = \frac{\partial^2 g(\mathbf{z})}{\partial z_j \partial z_k} &= 2\gamma^2 \sum_{i=1}^{n_{\mathrm{SV}}} \alpha_i y_i e^{-\gamma \|\mathbf{x}_i\|^2} X_{j,i} X_{k,i}, \\ \mathbf{M} &= \mathbf{X}\mathbf{D}\mathbf{X}^T. \end{aligned}$$

The vector $\mathbf{v}$ and matrix $\mathbf{M}$ represent the gradient and Hessian of g, respectively. Here $\mathbf{w} \in \mathbb{R}^{n_{\mathrm{SV}}}$ is a weighting vector: $w_i = 2\gamma \alpha_i y_i e^{-\gamma \|\mathbf{x}_i\|^2}$ and $\mathbf{D} \in \mathbb{R}^{n_{\mathrm{SV}} \times n_{\mathrm{SV}}}$ is a diagonal scaling matrix: $D_{i,i} = 2\gamma^2 \alpha_i y_i e^{-\gamma \|\mathbf{x}_i\|^2}$ and $D_{i,j} = 0$ if $i \neq j$. Finally, the approximated decision function $\hat{f}(\mathbf{z})$ is obtained by using $\hat{g}(\mathbf{z})$ in Eq. (4):

$$\hat{f}(\mathbf{z}) = e^{-\gamma \|\mathbf{z}\|^2} \hat{g}(\mathbf{z}) + b = e^{-\gamma \|\mathbf{z}\|^2} \big(c + \mathbf{v}^T \mathbf{z} + \mathbf{z}^T \mathbf{M} \mathbf{z}\big) + b. \tag{8}$$

The parameters c, $\mathbf{v}$, $\mathbf{M}$ and b are independent of test points and need only be computed once. The complexity of a single prediction becomes $O(d^2)$ – due to $\mathbf{z}^T \mathbf{M} \mathbf{z}$ – instead of $O(n_{\mathrm{SV}} \times d)$ for an exact RBF kernel.

The model size and prediction complexity of the proposed approximation is independent of the amount of support vectors in the exact model. This is especially interesting for least squares SVM formulations, which are generally not sparse in terms of support vectors (Suykens & Vandewalle, 1999). The RBF approximation loses its appeal when the number of input dimensions grows very large. For problems with high input dimensionality, the feature mapping induced by an RBF kernel often yields little improvement over using the linear kernel anyway (Hsu et al., 2003).

3.1 Approximation Accuracy

The relative error of the second-order Maclaurin series approximation of the exponential function is less than 3.05% for exponents in the interval $[-0.5, 0.5]$ (see Eq. (13) in Appendix A). Adhering to this interval guarantees that the relative error of any given term in the linear combination of $\hat{g}(\mathbf{z})$ is below 3.05%, compared to $g(\mathbf{z})$ (Eqs. (7) and (5), respectively). This translates into the following bound for our approximation:

$$|2\gamma \mathbf{x}_i^T \mathbf{z}| < \frac{1}{2}, \quad \forall i. \tag{9}$$

The inner product can be avoided via the Cauchy-Schwarz inequality:

$$|\mathbf{x}_i^T \mathbf{z}| \leq \|\mathbf{x}_i\| \|\mathbf{z}\|, \quad \forall i. \tag{10}$$

Combining Eqs. (9) and (10) yields a way to assess the validity of the approximation in terms of the support vector $\mathbf{x}_M$ with maximal norm ($\forall i : \|\mathbf{x}_M\| \geq \|\mathbf{x}_i\|$):

$$\|\mathbf{x}_M\|^2 \|\mathbf{z}\|^2 < \frac{1}{16\gamma^2}. \tag{11}$$

Storing $\|\mathbf{x}_M\|^2$ in the approximated model enables checking adherence to the bound in Eq. (11) during prediction, based on the squared norm of the test instance $\|\mathbf{z}\|^2$. Observe that this bound can be verified during prediction at no extra cost because $\|\mathbf{z}\|^2$ must be computed anyway (see Eq. (8)). Our tools can additionally report an upper bound for γ for a given data set prior to training a model. In this case, the upper bound is obtained based on the maximum norm over all instances. The obtained upper bound for γ may be slightly overconservative, because the data instance with maximum norm will not necessarily become a support vector.

3.2 Implementation

In order to benchmark the approximation against exact evaluations, we have made a `C++` implementation to approximate LIBSVM models and predict with the approximated model.[1] The implementation features a set of configurations to do the main computations. The configurations differ in the use of linear algebra libraries and vector instructions. Different configurations have consequences in two aspects: (i) approximating an SVM model and (ii) predicting with the approximated model.

Approximation Speed The key determinant of approximation speed is matrix math. Approximation time is dominated by the computation of $\mathbf{M} = \mathbf{X}\mathbf{D}\mathbf{X}^T$, which involves large matrices if d and n_{SV} are large. The following implementations have been made:

1. `LOOPS`: uses simple loops to implement matrix math (default).

2. `BLAS`: uses the Basic Linear Algebra Subprograms (BLAS) for matrix math

[1] Our implementation is available at `https://github.com/claesenm/approxsvm`.

(Blackford et al., 2001). The BLAS are usually available by default on modern Linux installations (in `libblas`). This default version is typically not heavily optimized.

3. `ATLAS`: uses the Automatically Tuned Linear Algebra Software (ATLAS) routines for matrix math (Whaley et al., 2000). ATLAS provides highly optimized versions of the BLAS for the platform on which it is installed. The performance of ATLAS is comparable to vendor-specific linear algebra libraries such as Intel's Math Kernel Library (Eddelbuettel, 2010).

Prediction Speed The main factor in prediction speed for approximated models is evaluating $\mathbf{z}^T\mathbf{M}\mathbf{z}$ where $\mathbf{M}$ is a symmetric $d \times d$ matrix. This simple operation can exploit Single Instruction Multiple Data (SIMD) instruction sets if the platform supports them. The use of vector instructions can be enabled via compiler flags. We observed no significant gains in prediction speed when using the BLAS or ATLAS.

4 Classification Benchmarks

To illustrate the speed and accuracy of the approximation, we used it for a set of classification problems. The exact models were always obtained using LIBSVM (Chang & Lin, 2011). We investigated the amount of labels that differ between exact and approximated models as well as speed gains.

The accuracies are listed in Table 1. We report the accuracy of the exact model and the percentage of labels which differ between the exact model and the approximation (note that not all differences are misclassifications). Table 2 reports the results of our

speed measurements. Before discussing these results, we briefly summarize the data sets we used.

4.1 Data Sets

To facilitate verification of our results, we used data sets that are freely available in LIBSVM format at the website of the LIBSVM authors.[2] We used all the data sets as they are made available, without extra normalization or preprocessing. We used the following classification data sets:

- `a9a`: the Adult data set, predict who has a salary over $\$50.000$, based on various information (Platt, 1998). This data set contains two classes, $d = 123$ features (most are binary dummy variables) and $32.561/16.281$ training/testing instances.

- `mnist`: handwritten digit recognition (LeCun et al., 1998). This data set contains 10 classes – we classified class 1 versus others, $d = 780$ features and $60.000/10.000$ training/testing instances.

- `ijcnn1`: used for the IJCNN 2001 neural network competition (Prokhorov, 2001). There are 2 classes, $d = 22$ features and $49.990/91.701$ training/testing instances.

- `sensit`: SensIT Vehicle (combined), vehicle classification (Duarte & Hen Hu, 2004). This data set contains 3 classes – we classified class 3 versus others, $d = 100$ features and $78.823/19.705$ training/testing instances.

[2] Available here: `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

- `epsilon`: used in the Pascal Large Scale Learning Challenge.[3] This data set contains 2 classes, $d = 2000$ features and $400.000/100.000$ training/testing instances. To reduce training time, we switched the training and test set.

4.2 Accuracy

The accuracies we obtained in our benchmarks are listed in Table 1. This table contains the key parameters per data set: number of dimensions d and the maximum value that should be used for γ in order to guarantee validity of the approximation (γ_{MAX}). Here γ_{MAX} is obtained via Eq. (11) after data normalization. The last column shows that only a very minor number of labels are predicted differently by the exact and approximated models.

Some of the listed results do not adhere to the bound, e.g. $\gamma > \gamma_{MAX}$. We used these parameters to illustrate that even though the accuracy of some terms in the linear combination may be inaccurate (e.g. relative error larger than 3%), the overall accuracy may still remain very good. In practice, we always recommend to adhere to the bound which guarantees high accuracy. Ignoring this bound is equivalent to abandoning all guarantees regarding approximation accuracy, because it is impossible to assess the approximation error which increases exponentially (shown in Figure 1 in the appendix).

When the bound was satisfied, the fraction of erroneous labels was consistently less than 1% (`a9a`, `mnist` and `ijcnn1`). In the last experiment for `a9a` we used a value for γ that is over five times larger than γ_{MAX} and still get only 3.5% of erroneous labels. These results demonstrate that the approximation is very acceptable in terms of

[3] Website: `http://largescale.ml.tu-berlin.de/instructions/`.

accuracy.

The experiments on `sensit` and `epsilon` illustrate that a large number of dimensions d safeguards the validity of the approximation to some extent, even when γ becomes too large. The fraction γ/γ_{MAX} is larger for `epsilon` than it is for `sensit` but due to the higher number of dimensions in `epsilon`, the fraction of erroneous labels remains lower (0.53% for `epsilon` versus 0.95% for `sensit`). This occurs because the Cauchy-Schwarz inequality (Equation (10)) is a worst-case upper bound for the inner product. When d grows large, it is less likely for $|\mathbf{x}_i^T\mathbf{z}|$ to approach $\|\mathbf{x}_i\|\|\mathbf{z}\|$. In other words, the bound we use – based on the Cauchy-Schwarz inequality – is more conservative for larger input dimensionalities.

data set	d	γ_{MAX}	γ	n_{test}	n_{SV}	acc (%)	$diff$ (%)
adult (a9a)	122	0.018	0.01	16.281	11.834	84.8	0.2
adult (a9a)	122	0.018	0.02	16.281	11.674	84.9	1.3
adult (a9a)	122	0.018	0.10	16.281	11.901	85.0	3.5
mnist	780	10^{-3}	10^{-4}	10.000	2.174	99.3	0.08
ijcnn1	22	0.064	0.05	91.701	4.044	97.7	0.46
sensit	100	0.0025	0.003	19.705	25.722	86.5	0.95
epsilon	2000	0.25	0.35	400.000	36.988	89.2	0.53

Table 1: Experiment summary: data set name, dimensionality, maximum value for γ, actual value for γ, number of testing points, number of SV in the model, accuracy of the (exact) model, percentage of erroneous labels in approximation.

4.3 Speed Measurements

Timings were performed on a desktop running Debian Wheezy. We used the default BLAS that are prebundled with Debian, which appear to be somewhat optimized, but not as much as ATLAS. We ran benchmarks on an Intel i5-3550K, which supports the Advanced Vector Extensions (AVX) instruction set for SIMD operations.

Table 2 contains timing results of prediction speed between exact models and their approximations. The speed increase for the approximation is evident: ranging from 7 to 137 times when the time to approximate is disregarded, or 4.4 to 69 times when it is accounted for. We can see that the speed increase also holds for a large number of dimensions (2000 for the `epsilon` data set). The model for `mnist` contains few SVs compared to the number of dimensions, which causes a smaller speed increase in favor of the approximated model.

In terms of approximation speed, the impact of specialized linear algebra libraries is apparant as shown in columns 3 and 4 of Table 2. ATLAS consistently outperforms BLAS and both are orders of magnitude faster than the naive implementation, particularly when the matrix $\mathbf{X}$ gets large (over $100\times$ faster for `epsilon`, where $\mathbf{X}$ is 2.000×36.988).

The impact of vector instructions is clear, with gains of up to 25% in prediction speed when they are used (cfr. `mnist` results). Note that most of the time is spent on file IO for these benchmarks, which may give a pessimistic misinterpretation of the speed increase of vector instructions.

Table 2: Comparison of prediction speed of an exact model vs. approximations. Approximations are classified based on use of math libraries and vector instructions. Times listed are prediction time and approximation time. Timings were performed in high-priority mode (using nice -3 in Linux) on an Intel i5-3550K at 3.30 GHz. ATLAS timings are not reported when its speed was comparable to BLAS. The last two columns contain the relative increase in prediction speed of the approximation compared to exact predictions, with and without accounting for the time needed to approximate the exact model.

$\star$: time in minutes for the `epsilon` data set.

data set	approach	math	t_{approx} (s)	SIMD	t_{pred} (s)	ratio 1	ratio 2
`a9a`	exact	/	/	/	$\mathbf{13.75 \pm 0.060}$	1	1
	approx	`BLAS`	0.05 ± 0.002	$\times$	0.160 ± 0.002	86	65
		`LOOPS`	0.56 ± 0.021	✓	0.146 ± 0.003	94	19
	optimal	`BLAS`	0.05 ± 0.002	✓	0.146 ± 0.003	94	70
`mnist`	exact	/	/	/	$\mathbf{12.81 \pm 0.016}$	1	1
	approx	`BLAS`	0.036 ± 0.001	$\times$	1.757 ± 0.008	7.3	7.1
		`LOOPS`	1.480 ± 0.005	✓	1.405 ± 0.006	9.1	4.4
	optimal	`BLAS`	0.036 ± 0.001	✓	1.405 ± 0.006	9.1	8.9
`ijcnn1`	exact	/	/	/	$\mathbf{15.87 \pm 0.012}$	1	1
	approx	`BLAS`	0.010 ± 0.000	$\times$	0.679 ± 0.012	23	23
		`LOOPS`	0.010 ± 0.000	✓	0.667 ± 0.016	24	23
	optimal	`BLAS`	0.010 ± 0.000	✓	0.667 ± 0.016	24	23
`sensit`	exact	/	/	/	$\mathbf{79.62 \pm 0.127}$	1	1
	approx	`BLAS`	0.670 ± 0.000	$\times$	0.590 ± 0.000	134	63
		`LOOPS`	1.437 ± 0.036	✓	0.581 ± 0.012	137	39
	optimal	`ATLAS`	0.565 ± 0.005	✓	0.581 ± 0.012	137	69
`epsilon`	exact	/	/	/	$\mathbf{622.1 \pm 0.165}$	1	1
$\star$	approx	`BLAS`	1.161 ± 0.003	$\times$	10.78 ± 0.110	58	52
		`LOOPS`	43.98 ± 0.495	✓	9.68 ± 0.03	64	12
	optimal	`ATLAS`	0.442 ± 0.029	✓	9.68 ± 0.03	64	61

5 Applications

The most straightforward applications of the proposed approximation are those which require fast prediction. This includes many computer vision applications such as object detection, which require a large amount of predictions, potentially in real-time (Cao et al., 2008; Maji et al., 2013).

Complementary to featuring faster prediction, the approximated kernel models are often smaller than exact models. The approximated models consist of three scalars (b, c and γ), a dense vector ($\mathbf{v} \in \mathbb{R}^d$) and a dense, symmetric matrix ($\mathbf{M} \in \mathbb{R}^{d \times d}$). When the number of dimensions is small compared to the number of SVs, these approximated models are significantly smaller than their exact counterparts. We included Table 3 to illustrate this property: it shows the model sizes per classification data set. In our experiments the approximated models are smaller for all data sets except one. The biggest compression ratio we obtained was 290 times (for the `sensit` data set). If we would approximate least squares SVM models, the compression ratios would be even larger due to the larger amount of SVs in least squares SVM models (Suykens & Vandewalle, 1999).

Finally, a subtle side effect of our method is the fact that training data is obfuscated in approximated models. Data obfuscation is a technique used to hide sensitive data (Bakken et al., 2004). Training data may be proprietary and/or contain sensitive information that cannot be exposed in contexts such as biomedical research (Murphy & Chueh, 2002). In standard SVM models, the support vectors are exact instances of the training set. This renders SVM models unusable when data dissemination is an issue. The approximated models consist of complicated combinations of the support vectors

data set	d	n_{SV}	exact	approximation	ratio
`a9a`	122	11.834	830 KB	111 KB	7.5
`mnist`	780	2.174	3.2 MB	3.7 MB	0.86
`ijcnn1`	22	4.044	628 KB	4.2 KB	150
`sensit`	100	25.722	32 MB	113 KB	290
`epsilon`	2.000	36.988	1.1 GB	42 MB	27

Table 3: Comparison of model sizes (both types are stored in text format).

(and typically $d \ll n_{\mathrm{SV}}$), which makes it very challenging to reverse-engineer parts of the original data from the model. The approximation can be considered a surrogate one-way function of the support vectors (Levin, 2000). As such, these approximations may allow SVMs to be used in situations where they are currently not considered (Bakken et al., 2004).

Conclusion

We have derived an approximation for SVM models with RBF kernels, based on the second-order Maclaurin series approximation of the exponential function. The applicability of the approximation is not limited to SVMs: it can be used in a wide variety of kernel methods. The proposed approximation has been shown to yield significant gains in prediction speed.

Our work generalizes the approximation proposed by Cao et al. (2008). The derivation we performed made no implicit assumptions regarding data normalization. An easily verifiable bound was established which can be used to guarantee that the relative

error of individual terms in the approximation remains low.

Our benchmarks have shown that a minor loss in accuracy can result in very large gains in prediction speed. We have listed some example applications for such approximations. In addition to applications in which low run-time complexity is desirable, applications that require compact or data-hiding models benefit from our approach.

Acknowledgements

Marc Claesen is a research assistant at KU Leuven, funded by IWT grant number 111065. Johan Suykens is a professor at KU Leuven. Bart De Moor is a full professor at KU Leuven, Belgium. Research supported by: Research Council KUL: ProMeta, GOA MaNet, KUL PFV/10/016 SymBioSys, START 1, OT 09/052 Biomarker, several PhD/postdoc & fellow grants; Flemish Government: IOF: IOF/HB/10/039 Logic Insulin, FWO: PhD/postdoc grants, projects: G.0871.12N research community MLDM; G.0733.09; G.0824.09; IWT: PhD Grants; TBM-IOTA3, TBM-Logic Insulin; FOD: Cancer plans; Hercules Stichting: Hercules III PacBio RS; EU-RTD: ERNSI; FP7-HEALTH CHeartED; COST: Action BM1104, Action BM1006: NGS Data analysis network; ERC AdG A-DATADRIVE-B.

A Approximation of exponential function

The Maclaurin series for the exponential function and its second-order approximation are:

$$\begin{aligned} e^x &= \sum_{k=0}^{\infty} \frac{1}{k!} x^k, \\ &\approx 1 + x + \frac{1}{2} x^2. \end{aligned} \tag{12}$$

Figure 1 illustrates the absolute relative error of the second-order Maclaurin series approximation. The relative error is smaller than $\pm 3\%$ when the absolute value of the exponent x is small enough, e.g. $|x| < 0.5$:

$$|x| < \frac{1}{2} \quad \Rightarrow \quad \left|\frac{e^x - 1 - x - 0.5x^2}{e^x}\right| < 0.0305. \tag{13}$$

This can be used to verify the validity of the approximation.

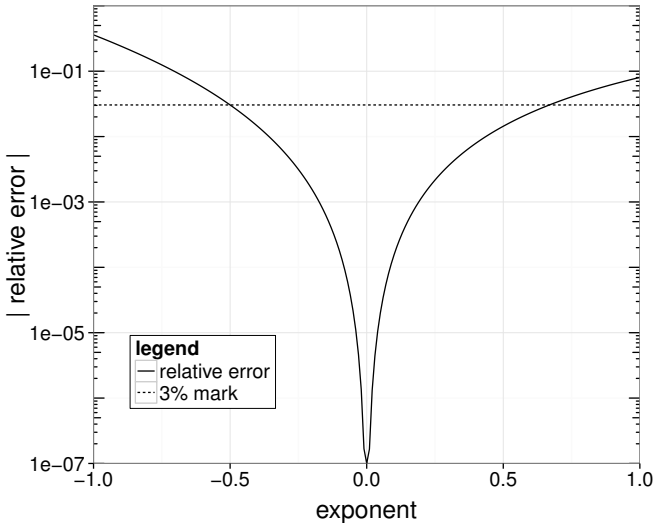


Figure 1: Absolute relative error of the second-order Maclaurin series approximation: $\left|\frac{e^x - 1 - x - 0.5x^2}{e^x}\right|$.

References

Bakken, D., Rarameswaran, R., Blough, D., Franz, A., & Palmer, T. (2004). Data obfuscation: anonymity and desensitization of usable data sets. *Security Privacy, IEEE*, *2*(6), 34-41.

Blackford, L. S., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., et al. (2001). An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, *28*, 135–151.

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, *2*(2), 121–167.

Cao, H., Naito, T., & Ninomiya, Y. (2008). Approximate RBF Kernel SVM and Its Applications in Pedestrian Classification. In *The 1st International Workshop on Machine Learning for Vision-based Motion Analysis - MLVMA'08.* Marseille, France. Available from `http://hal.inria.fr/inria-00325810`

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, *2*, 27:1–27:27. (Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`)

Cortes, C., & Vapnik, V. (1995, September). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

De Brabanter, K., Karsmakers, P., Ojeda, F., Alzate, C., De Brabanter, J., Pelckmans, K., et al. (2010). *LS-SVMlab toolbox user's guide version 1.8* (Tech. Rep. No. 10-146). Leuven, Belgium: ESAT-SISTA, KU Leuven.

Duarte, M. F., & Hen Hu, Y. (2004). Vehicle classification in distributed sensor networks. *Journal of Parallel and Distributed Computing*, *64*(7), 826–838.

Eddelbuettel, D. (2010). *Benchmarking single-and multi-core BLAS implementations and GPUs for use with R.* Mathematica.

Filippone, M., Camastra, F., Masulli, F., & Rovetta, S. (2008, January). A survey of kernel and spectral methods for clustering. *Pattern Recognition*, *41*(1), 176–190.

Herbster, M. (2001). Learning additive models online with fast evaluating kernels. In *Computational learning theory* (pp. 444–460).

Hoegaerts, L., Suykens, J. A., Vandewalle, J., & De Moor, B. (2004). A comparison of pruning algorithms for sparse least squares support vector machines. In *Neural information processing* (pp. 1247–1253).

Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al. (2003). *A practical guide to support vector classification.*

Joachims, T. (1999). Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, & A. J. Smola (Eds.), *Advances in kernel methods - support vector learning* (pp. 169–184). Cambridge, MA, USA: MIT Press.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. (MNIST database available at `http://yann.lecun.com/exdb/mnist/`.)

Levin, L. A. (2000). The tale of one-way functions. *CoRR*, *cs.CR/0012023*.

Liang, X. (2010). An effective method of pruning support vector machine classifiers. *Neural Networks, IEEE Transactions on*, *21*(1), 26–38.

Maji, S., Berg, A. C., & Malik, J. (2013). Efficient classification for additive kernel SVMs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(1), 66-77.

Mika, S., Ratsch, G., Weston, J., Scholkopf, B., & Muller, K. (1999). Fisher discriminant analysis with kernels. In *Neural networks for signal processing ix, 1999. proceedings of the 1999 ieee signal processing society workshop.* (p. 41-48).

Mika, S., Schölkopf, B., Smola, A., Müller, K.-R., Scholz, M., & Rätsch, G. (1999). Kernel PCA and de-noising in feature spaces. *Advances in neural information processing systems*, *11*(1), 536–542.

Murphy, S., & Chueh, H. (2002). A security architecture for query tools used to access large biomedical databases. *Proc AMIA Symp*.

Platt, J. C. (1998). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, & A. J. Smola (Eds.), *Advances in kernel methods – support vector learning.* Cambridge, MA, USA: MIT Press.

Poggio, T., & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE*, *78*(9), 1481–1497.

Prokhorov, D. (2001). IJCNN 2001 neural network competition. *Slide presentation in IJCNN*.

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning*. The MIT Press.

Schölkopf, B., Herbrich, R., & Smola, A. J. (2001). A generalized representer theorem. In *Computational learning theory* (pp. 416–426).

Schölkopf, B., Knirsch, P., Smola, A., & Burges, C. (1998). Fast approximation of support vector kernel expansions, and an interpretation of clustering as approximation in feature spaces. In *Mustererkennung 1998* (pp. 125–132). Springer.

Schölkopf, B., Smola, A., & Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, *10*(5), 1299–1319.

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization and beyond*. the MIT Press.

Schraudolph, N. N. (1999). A fast, compact approximation of the exponential function. *Neural Computation*, *11*(4), 853–862.

Sonnenburg, S., Rätsch, G., Henschel, S., Widmer, C., Behr, J., Zien, A., et al. (2010, June). The SHOGUN machine learning toolbox. *Journal of Machine Learning Research*, *11*, 1799–1802. Available from `http://www.shogun-toolbox.org`

Suykens, J. A. K., De Brabanter, J., Lukas, L., & Vandewalle, J. (2002). Weighted least squares support vector machines: robustness and sparse approximation. *Neurocomputing*, *48*(1), 85–105.

Suykens, J. A. K., & Vandewalle, J. (1999, June). Least squares support vector machine classifiers. *Neural Processing Letters*, *9*(3), 293–300.

Suykens, J. A. K., Van Gestel, T., Vandewalle, J., & De Moor, B. (2003). A support vector machine formulation to PCA analysis and its kernel version. *Neural Networks, IEEE Transactions on*, *14*(2), 447–450.

Vedaldi, A., & Zisserman, A. (2010). Efficient additive kernels via explicit feature maps. In *Computer vision and pattern recognition (cvpr), 2010 ieee conference on* (pp. 3539–3546).

Whaley, C., Petitet, A., & Dongarra, J. J. (2000). Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, *27*, 2001.

Yang, J., Zhang, D., Frangi, A. F., & Yang, J.-y. (2004). Two-dimensional PCA: a new approach to appearance-based face representation and recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *26*(1), 131–137.