Memory and Computation-Efficient Kernel SVM via Binary Embedding and Ternary Model Coefficients

Zijian Lei, Liang Lan
Department of Computer Science, Hong Kong Baptist University, Hong Kong SAR, China
cszjlei@comp.hkbu.edu.hk, lanliang@comp.hkbu.edu.hk

Abstract

Kernel approximation is widely used to scale up kernel SVM training and prediction. However, the memory and computation costs of kernel approximation models are still too high if we want to deploy them on memory-limited devices such as mobile phones, smartwatches, and IoT devices. To address this challenge, we propose a novel memory and computation-efficient kernel SVM model by using binary embedding and ternary model coefficients. First, we propose an efficient way to generate compact binary embedding of the data, preserving the kernel similarity. Second, we propose a simple but effective algorithm to learn a linear classification model with ternary coefficients that can support different types of loss function and regularizer. Our algorithm can achieve better generalization accuracy than existing works on learning ternary coefficients since we allow coefficient to be -1, 0, or 1 during the training stage, and coefficient 0 can be removed during model inference for binary classification. Moreover, we provide a detailed analysis of the convergence of our algorithm and the inference complexity of our model. The analysis shows that the convergence to a local optimum is guaranteed, and the inference complexity of our model is much lower than other competing methods. Our experimental results on five large real-world datasets have demonstrated that our proposed method can build accurate nonlinear SVM models with memory costs less than 30KB.

Introduction

Kernel Support Vector Machine (SVM) is a powerful nonlinear classification model that has been successfully used in many real-world applications. Different from linear SVM, kernel SVM uses kernel function to capture the nonlinear concept. The prediction function of kernel SVM is $f(\mathbf{x}) = \sum_{\mathbf{x}_i} \alpha_i k(\mathbf{x}_i, \mathbf{x})$ where $\mathbf{x}_i$s are support vectors and $k(\mathbf{x}_i, \mathbf{x}_j)$ is a predefined kernel function to compute the kernel similarity between two data samples. Kernel SVM needs to maintain all support vectors explicitly for model inference. Therefore, the memory and computation costs of kernel SVM inference are usually huge, considering that the number of support vectors increases linearly with training data size on noisy data. Many kernel approximation methods (Rahimi and Recht 2008; Le, Sarlós, and Smola 2013; Lan et al. 2019; Hsieh, Si, and Dhillon 2014) have been proposed to reduce the memory and computation costs of kernel SVM in the past decade. The basic idea of these methods is to explicitly construct the nonlinear feature mapping $\mathbf{z} = \Phi(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^p$ such that $\mathbf{z}_i^\top \mathbf{z}_j \approx k(\mathbf{x}_i, \mathbf{x}_j)$ and then apply a linear SVM on $\mathbf{z}$. To obtain good classification accuracy, the dimensionality of nonlinear feature mapping $\mathbf{z}$ needs to be large. On the other hand, due to concerns on security, privacy, and latency caused by performing model inference remotely in the cloud, directly performing model inference on edge devices (e.g., Internet of Things (IoT) devices) has gained increasing research interests recently (Kumar, Goyal, and Varma 2017; Kusupati et al. 2018). Even though those kernel approximation methods can significantly reduce the memory and computation costs of exact kernel SVM, their memory and computation costs are still too large for on-device deployment. For example, the Random Fourier Features (RFF), which is a very popular kernel approximation method for large scale data, requires $\sim d \times p \times 32$ bits for nonlinear feature transformation, $p \times 32$ bits for nonlinear feature representation, and $p \times c \times 32$ bits (c is the number of classes) for classification model to predict the label of a single input data sample. The memory cost of RFF can easily exceed several hundred megabytes (MB), which could be prohibitive for on-device model deployment.

Recently, binary embedding methods (Yu et al. 2017; Needell, Saab, and Woolf 2018) have been widely used for reducing memory cost for data retrieval and classification tasks. The nice property of binary embedding is that each feature value can be efficiently stored using a single bit. Therefore, binary embedding can reduce the memory cost by 32 times compared with storing full precision embedding. The common way for binary embedding is to apply sign$(\cdot)$ function to the famous Johnson-Lindenstrauss embedding: random projection embedding justified by the Johnson-Lindenstrauss Lemma (Johnson, Lindenstrauss, and Schechtman 1986). It obtains the binary embedding by $\mathbf{z} = \text{sign}(\mathbf{Rx})$ where $\mathbf{R} \in \mathbb{R}^{d \times p}$ is a random Gaussian matrix (Needell, Saab, and Woolf 2018; Ravi 2019) or its variants (Yu et al. 2017; Gong et al. 2012; Shen et al. 2017) and sign$(\cdot)$ is the element-wise sign function. However, most of them focus on data retrieval, and our interest is on-device nonlinear classification. The memory cost of the full-precision dense matrix $\mathbf{R}$ is high for on-device model deployment. Besides, even though sign$(\mathbf{Rx})$ provides a nonlinear mapping because of the sign$(\cdot)$ function, the theoretical analysis on binary mapping sign$(\mathbf{Rx})$ can

only guarantee that the angular distance among data samples in the original space is well preserved. It cannot well preserve the kernel similarity among data samples, which is crucial for kernel SVM.

In this paper, we first propose a novel binary embedding method that can preserve the kernel similarity of the shift-invariant kernel. Our proposed method is built on Binary Codes for Shift-invariant kernels (BCSIK) (Raginsky and Lazebnik 2009) but can significantly reduce the memory and computation costs of BCSIK. Our new method can reduce the memory cost of BCSIK from $O(dp)$ to $O(p)$ and the computational cost of BCSIK from $O(dp)$ to $O(plogd)$. Second, in addition to binary embedding, we propose to learn the classification model with ternary coefficients. The classification decision function is reduced to a dot product between a binary vector and a ternary vector, which can be efficiently computed. Also, the memory cost of the classification model will be reduced by 32 times. Unlike existing works (Shen et al. 2017; Alizadeh et al. 2019) on learning ternary coefficients, we allow the model coefficient to be $\{-1, 0, 1\}$ during the training stage. This additional 0 can help to remove uncorrelated binary features and improve the generalization ability of our model. The 0 coefficients and corresponding transformation column vectors in matrix $\mathbf{R}$ can be safely removed during the model inference for binary classification problems. A simple but effective learning algorithm that can support different types of loss function and regularizer is proposed to learn ternary model coefficients from data. Third, we provide a detailed analysis of our algorithm's convergence and our model's inference complexity. The analysis shows that the convergence to a local optimum is guaranteed, and the memory and computation costs of our model inference are much lower than other competing methods. We compare our proposed method with other methods on five real-world datasets. The experimental results show that our proposed method can significantly reduce the memory cost of RFF while achieve good accuracy.

Methodology

Binary Embedding for Shift-Invariant Kernels

Preliminaries on Binary Codes for the Shift-Invariant Kernels (BCSIK). BCSIK (Raginsky and Lazebnik 2009) is a random projection based binary embedding method which can well preserve the kernel similarity defined by a shift-invariant kernel (e.g., Gaussian Kernel). It works by composing random Fourier features with a random sign function. Let us use $\mathbf{x} \in \mathbb{R}^d$ to denote an input data sample with d features. BCSIK encodes $\mathbf{x}$ into a p-dimensional binary representation $\mathbf{z}$ as

$$\mathbf{z} = \text{sign}(\cos(\mathbf{R}^\top \mathbf{x} + \mathbf{b}) + \mathbf{t}), \tag{1}$$

where each column $\mathbf{r}$ in matrix $\mathbf{R} \in \mathbb{R}^{d\times p}$ is randomly drawn from a distribution corresponding to an underlying shift-invariant kernel. For example, for Gaussian kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{2\sigma^2})$, each entry in $\mathbf{r}$ is drawn from a normal distribution $\mathcal{N}(0, \sigma^{-2})$. $\mathbf{b} \in \mathbb{R}^p$ is a column vector where each entry is drawn from a uniform distribution from $[0, 2\pi]$. $\mathbf{t} \in \mathbb{R}^p$ is a column vector where each entry is drawn from a uniform distribution from $[-1, 1]$. $\cos(\cdot)$ and $\text{sign}(\cdot)$ are the element-wise cosine and sign functions. As can be seen, $\Phi(\mathbf{x}) = \cos(\mathbf{R}^\top \mathbf{x} + \mathbf{b})$ in (1) is the random Fourier features for approximating kernel mapping which has the theoretical guarantee $\mathbb{E}[\Phi(\mathbf{x}_1)^\top \Phi(\mathbf{x}_2)] = k(\mathbf{x}_1, \mathbf{x}_2)$ for shift-invariant kernel (Rahimi and Recht 2008), where $\mathbb{E}$ is the statistical expectation. $\text{sign}(\Phi(\mathbf{x}) + \mathbf{t})$ uses a random sign function to convert the full-precision mapping $\Phi(\mathbf{x})$ into binary mapping $\mathbf{z}$. Each entry in $\mathbf{z}$ can be stored efficiently using a single bit. Therefore, compared with RFF, BCSIK can reduce the memory cost of storing approximated kernel mapping by 32 times.

Besides memory saving, a great property of BCSIK is that the normalized hamming distance between the binary embedding of any two data samples sharply concentrates around a well-defined continuous function of the kernel similarity between these two data samples shown in the Lemma 1.

Lemma 1 (Johnson-Lindenstrauss Type Result on BCSIK (Raginsky and Lazebnik 2009))**.** *Define the functions* $h_1(u) \triangleq \frac{4}{\pi^2}(1-u)$ *and* $h_2(u) \triangleq min\{\frac{1}{2}\sqrt{1-u}, \frac{4}{\pi^2}(1-\frac{2}{3}u)\}$*, where* $u \in [0, 1]$*. Fix* $\epsilon, \delta \in (0, 1)$*. For any finite dataset* $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ *of* n *data samples in* $\mathbb{R}^d$*, the following inequality about the normalized Hamming distance on the binary embedding between any two data samples* $\mathbf{z}_i$ *and* $\mathbf{z}_j$ *holds true with probability* $\geq 1 - \epsilon$ *with* $p \geq \frac{1}{2\delta^2} log(\frac{n^2}{\epsilon})$

$$h_1(k(\mathbf{x}_i, \mathbf{x}_j)) - \delta \leq \frac{1}{p} d_H(\mathbf{z}_i, \mathbf{z}_j) \leq h_2(k(\mathbf{x}_i, \mathbf{x}_j) + \delta. \tag{2}$$

Note that the hamming distance between $\mathbf{z}_i$ and $\mathbf{z}_j$ can be expressed as $d_H(\mathbf{z}_i, \mathbf{z}_j) = \frac{1}{2}(p - \mathbf{z}_i^\top \mathbf{z}_j)$. The bounds in Lemma 1 indicate that the binary embedding $\mathbf{z} \in \{-1, 1\}^p$ as defined in (1) well preserves the kernel similarity obtained from the underlying shift-invariant kernel.

Reduce the Memory and Computation Costs of BCSIK. When considering on-device model deployment, the bottleneck of obtaining binary embedding is the matrix-vector multiplication $\mathbf{R}^\top \mathbf{x}$ as shown in (1). It requires $O(dp)$ time and space. By considering p is usually several times larger than d for accurate nonlinear classification, the memory cost of storing $\mathbf{R}$ could be prohibitive for on-device model deployment. Therefore, we propose to generate the Gaussian random matrix $\mathbf{R}$ using the idea of Fastfood (Le, Sarlós, and Smola 2013). The core idea of Fastfood is to reparameterize a Gaussian random matrix by a product of Hadamard matrices and diagonal matrices. Assuming $d = 2^{q1}$ and q is any positive integer, it constructs $\mathbf{V} \in \mathbb{R}^{d\times d}$ as follows to reparameterize a $d \times d$ Gaussian random matrix,

$$\mathbf{V} = \frac{1}{\sigma\sqrt{d}} \mathbf{SHG\Pi HB}, \tag{3}$$

where

- $\mathbf{S}, \mathbf{G}$ and $\mathbf{B}$ are diagonal matrices. $\mathbf{S}$ is a random scaling matrix, $\mathbf{G}$ has random Gaussian entries and $\mathbf{B}$ has elements are independent random signs $\{-1, 1\}$. The memory costs of $\mathbf{S}, \mathbf{G}$ and $\mathbf{B}$ are $O(d)$.
- $\mathbf{\Pi} \in \{0, 1\}^{d\times d}$ is a random permutation matrix and also has $O(d)$ space complexity

[1] We can ensure this by padding zeros to original data

- $\mathbf{H} \in \mathbb{R}^{d\times d}$ is the Walsh-Hadamard matrix defined recursively as: $\mathbf{H}_d = \begin{bmatrix} \mathbf{H}_{d/2} & \mathbf{H}_{d/2} \\ \mathbf{H}_{d/2} & -\mathbf{H}_{d/2} \end{bmatrix}$ with $\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$; The fast Hadamard transform allows us to compute $\mathbf{Hx}$ in $O(d \log d)$ time using the Fast Fourier Transform (FFT) operation.

When reparameterizing a $d \times p$ Gaussian random matrix ($p \gg d$), Fastfood replicates (3) for p/d independent random matrices $\mathbf{V}_i$ and stack together as

$$\tilde{\mathbf{R}}^\top = [\mathbf{V}_1; \mathbf{V}_2; \ldots, \mathbf{V}_{p/d}]^\top, \tag{4}$$

until it has enough dimensions. Then, we can generate the binary embedding in a memory and computation-efficient way as follows,

$$\mathbf{z} = \text{sign}(\cos(\tilde{\mathbf{R}}^\top \mathbf{x} + \mathbf{b}) + \mathbf{t}). \tag{5}$$

Note that each column in $\tilde{\mathbf{R}}$ is a random Gaussian vector from $\mathcal{N}(0, \sigma^{-2}\mathbf{I}_d)$ as proved in (Le, Sarlós, and Smola 2013), therefore the Lemma 1 still holds when we replace $\mathbf{R}$ in (1) by $\tilde{\mathbf{R}}$ for efficiently generating binary embedding. Note that the Hadamard matrix does not need to be explicitly stored. Therefore, we can reduce the space complexity of (1) in BCSIK from $O(dp)$ to $O(p)$ and reduce the time complexity of (1) from $O(dp)$ to $O(plog(d))$.

Ternary Model Coefficients for Classification

By using (5), each data sample $\mathbf{x}_i \in \mathbb{R}^d$ is transformed to a bit vector $\mathbf{z}_i \in \mathbb{R}^p$. Then, we can train a linear classifier on $\{\mathbf{z}_i, y_i\}_{i=1}^n$ to approximate the kernel SVM. Suppose the one-vs-all strategy is used for multi-class classification problems, the memory cost of the learned classifier is $p \times c \times 32$, where c is the number of classes. Since p usually needs to be very large for accurate classification, the memory cost of the classification model could also be too huge for edge devices, especially when we are dealing with multi-class classification problems with a large number of classes.

Here we propose to learn a classification model with ternary coefficients for reducing the memory cost. Moreover, by using binary model coefficients, the decision function of classification is reduced to a dot product between two binary vectors which can be very efficiently computed. Compare to existing works (Shen et al. 2017; Alizadeh et al. 2019) on learning binary model coefficients which constrain the coefficient to be 1 or -1, we allow the model coefficients to be 1, 0 or -1 during the training stage. The intuition of this operation came from two aspects. First, suppose our data distribute in a hyper-cube and linearly separable, the direction of the classification hyper-plane can be more accurate when we allow ternary coefficients. For example, in a lower projected dimension, the binary coefficients can achieve 2^p directions while the ternary ones can achieve 3^p. This additional value 0 can help to remove uncorrelated binary features and improve the generalization ability of our model as the result of importing the regularization term. In addition, we add a scaling parameter α to prevent possible large deviation between full-precision model coefficients and quantized model coefficients which affects the computation of loss function on training data. Therefore, our objective is formulated as

$$\begin{aligned} \min_{\alpha,\mathbf{w}} & \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \alpha\mathbf{w}^\top \mathbf{z}_i)) + \lambda R(\mathbf{w}) \\ \text{s.t.} \quad & \mathbf{w} \in \{-1, 0, 1\}^p \\ & \alpha > 0. \end{aligned} \tag{6}$$

where p is the dimensionality of $\mathbf{z}_i$ and y_i is the corresponding label for $\mathbf{z}_i$. $\ell(y_i, \alpha\mathbf{w}^\top \mathbf{z}_i)$ in (6) denotes a convex loss function and $R(\mathbf{w})$ denotes a regularization term on model parameter $\mathbf{w}$. λ is a hyperparameter to control the tradeoff between training loss and regularization term.

Learning ternary coefficient with Hinge Loss and l_2-norm Regularizer. For simplicity of presentation, let us assume $y_i \in \{-1, 1\}$. Our model can be easily extended to multi-class classification using the one-vs-all strategy. Without loss of generality, in this section, we show how to solve (6) when hinge loss and l_2-norm regularizer are used. In other words, $\ell(y_i, \alpha\mathbf{w}^\top \mathbf{z}_i)$ is defined as $\max(0, 1 - y_i\alpha\mathbf{w}^\top \mathbf{z}_i)$. $R(\mathbf{w})$ is defined as $\alpha^2 \sum_{j=1}^p w_j^2$. Any other loss function or regularization can also be applied.

By using hinge loss and l_2-norm regularization, then (6) will be rewritten as:

$$\begin{aligned} \min_{\alpha,\mathbf{w}} & \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - \alpha y_i \mathbf{w}^\top \mathbf{z}_i) + \lambda\alpha^2 \sum_{j=1}^{p} w_j^2 \\ \text{s.t.} \quad & \mathbf{w} \in \{-1, 0, 1\}^p \\ & \alpha > 0. \end{aligned} \tag{7}$$

We can use alternating optimization to solve (7): (1) fixing $\mathbf{w}$ and solving α; and (2) fixing α and solving $\mathbf{w}$.

1. Fixing $\mathbf{w}$ and solving α. When $\mathbf{w}$ is fixed, (7) will be reduced to a problem with only one single variable α as follows

$$\begin{aligned} \min_{\alpha} & \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - (y_i \mathbf{w}^\top \mathbf{z}_i)\alpha) + (\lambda \sum_{j=1}^{p} w_j^2)\alpha^2 \\ \text{s.t.} \quad & \alpha > 0. \end{aligned} \tag{8}$$

To solve (8), we propose to replace the constraint $\alpha > 0$ by $\alpha \geq \epsilon$ where ϵ is a small positive number. Then, the optimal α can be obtained by using the projected gradient descent method (Boyd, Boyd, and Vandenberghe 2004). Our experimental results have shown this technique can get good accuracy.

2. Fixing α and solving $\mathbf{w}$. When α is fixed, (7) will change to the following optimization problem,

$$\begin{aligned} \min_{\mathbf{w}} & \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - \alpha y_i \mathbf{w}^\top \mathbf{z}_i) + \lambda\alpha^2 \sum_{j=1}^{p} w_j^2 \\ \text{s.t.} \quad & \mathbf{w} \in \{-1, 0, 1\}^p. \end{aligned} \tag{9}$$

Due to the non-smooth constraints, minimizing (7) is an NP-hard problem and needs $O(3^p)$ time to obtain the global optimal solution. The Straight Through Estimator (STE) (Bengio, Léonard, and Courville 2013) framework which

is popular for learning binary deep neural networks can be used to solve (7). However, by considering that the STE can be unstable near a certain local minimum (Yin et al. 2019; Liu and Mattina 2019), we propose a much simpler but effective algorithm to solve (7). The idea is to update parameter $\mathbf{w}$ bit by bit, i.e., one bit each time for $w_j, j = 1, \ldots, p$, while keeping other $(p-1)$ bits fixed. Let use $\mathbf{w}_{(\neg j)}$ to denote the vector that equals to $\mathbf{w}$ except the j-th entry is set to 0. Therefore, (7) will be decomposed into a series of subproblems. A subproblem of (7) which only involves a single variable w_j can be written as (10).

$$\begin{aligned} \min_{w_j} \; L = & \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i \alpha(\mathbf{w}_{\neg j}^\top \mathbf{z}_i + w_j z_{ij})) \\ & + \lambda \alpha^2 w_j^2 + \lambda \alpha^2 \mathbf{w}_{\neg j}^\top \mathbf{w}_{\neg j} \\ \text{s.t.} \quad & w_j \in \{-1, 0, 1\}. \end{aligned} \tag{10}$$

Since $w_j \in \{-1, 0, 1\}$, objective (10) can be solved by just enumerating all three possible values $\{-1, 0, 1\}$ for w_j and select the one with the minimum objective value. Note that $\mathbf{w}^\top \mathbf{z}_i$ and $\mathbf{w}^\top \mathbf{w}$ can be pre-computed. Then, in each subproblem (10), both $\mathbf{w}_{\neg j}^\top \mathbf{z}_i = \mathbf{w}^\top \mathbf{z}_i - w_j z_{ij}$ and $\mathbf{w}_{\neg j}^\top \mathbf{w}_{\neg j} = \mathbf{w}^\top \mathbf{w} - w_j^2$ can be computed in $O(1)$ time. Therefore, we only need $O(n)$ time to evaluate the $L(w_j = -1)$, $L(w_j = 0)$ and $L(w_j = 1)$ for (10). The optimal solution for each subproblem,

$$w_j^* = \text{argmin}\{L(w_j = -1), L(w_j = 0), L(w_j = 1)\} \tag{11}$$

can be obtained in $O(n)$ time. Then w_j will be updated to w_j^* if it is not equal to w_j^*. This simple algorithm can be easily implemented and applied to other popular loss functions (e.g., logloss, square loss) and regularizers (e.g., l_1 regularizer). Note that for some specific loss function (e.g., square loss), a close form solution for w_j can be derived without using enumeration as shown in (11).

Parameter Initialization. In this section, we propose a heuristic way to initialize α and $\mathbf{w}$ which can help us to quickly obtain a local optimal solution for (7). The idea is we first randomly select a small subset of training data to apply the linear SVM on transformed data $\{\mathbf{z}_i, y_i\}_{i=1}^m$ to get the full-precision solution $\mathbf{w}_{full}$. Then the parameter $\mathbf{w}$ is initialized as

$$\begin{aligned} \mathbf{w} &= \text{sign}(\mathbf{w}_{full}) \\ \alpha &= \frac{\|\mathbf{w}_{full}\|_1}{p}. \end{aligned} \tag{12}$$

Empirically this initialization can lead to a fast convergence to a local optimum and produce better classification accuracy. The results are shown in Figure 2 and will be discussed in detail in the experiments section.

Efficiently Compute Ternary-Binary Dot Product. Once we get the ternary coefficients and the binary feature embedding, the following question is how to efficiently compute $\mathbf{w}^T\mathbf{z}$ since the parameter α only scales the value $\mathbf{w}^T\mathbf{z}$ and will not affect the prediction results. Next, we will discuss efficiently computing $\mathbf{w}^T\mathbf{z}$ using bit-wise operations.

For the binary classification problem, after the training stage, we can remove the coefficient w_js with zero value and also the corresponding columns in matrix $\mathbf{B}$. Therefore, for model deployment, our classification model $\mathbf{w}$ is still a binary vector and only one bit is needed for storing one entry w_j. Our model is different from ternary weight networks (Li, Zhang, and Liu 2016) where the coefficient 0 needs to be explicitly represented for model inference. Our prediction score can be computed as,

$$\mathbf{w}^T\mathbf{z} = 2\text{POPCOUNT}(\mathbf{z} \text{ XNOR } \mathbf{w}) - \text{len}(\mathbf{z}), \tag{13}$$

where XNOR is a bit-wise operation, POPCOUNT$(\mathbf{a})$ returns the number of 1 bits in $\mathbf{a}$ and the len$(\mathbf{z})$ returns the length of vector $\mathbf{z}$.

For the multi-class classification problem, the columns in $\mathbf{B}$ can only be removed if their corresponding coefficients are zero simultaneously in all $\mathbf{w}$ vectors for all c classes. Therefore we will need 2-bits to store the remaining coefficients after dropping the 0 value simultaneously in all classes. However, this 2-bit representation might have little influence on computational efficiency. For a given class j with model parameter $\mathbf{w}$, the original $\mathbf{w}_j$ with ternary values can be decomposed as $\mathbf{w} = \mathbf{w}^p \odot \mathbf{w}^s$ where $\mathbf{w}^s$ and $\mathbf{w}^p$ are defined as

$$w_j^p = \begin{cases} 1 & \text{if } w_j = 1 \\ -1 & \text{otherwise} \end{cases}, \quad w_j^s = \begin{cases} 1 & \text{if } w_j = \pm 1 \\ 0 & w_j = 0 \end{cases}. \tag{14}$$

Then $\mathbf{w}^T\mathbf{z}$ can be computed as:

$$\mathbf{w}^T\mathbf{z} = 2\text{POPCOUNT}(\mathbf{z} \text{ XNOR } \mathbf{w}^p \text{ AND } \mathbf{w}^s) - \text{len}(\mathbf{z}). \tag{15}$$

Note that for 1 in (14) represent logic TRUE and store as 1, while 0 and -1 in (14) will be stored as 0 to represent logic FALSE in model inference.

Algorithm Implementation and Analysis

We summarize our proposed algorithm for memory and computation-efficient kernel SVM in Algorithm 1. During the training stage, step 2 needs $O(np)$ space and $O(nplog(d))$ time to obtain the binary embedding by using FFT. To learn the binary model coefficients (from step 4 to step 21), it requires $O(tnp)$ time where t is the number of iterations. Therefore, the training process can be done efficiently. Our algorithm can be easily implemented and applicable to other loss functions and regularizers. The source code of our implementation is included in the supplementary materials and will be publicly available. Next, we present a detailed analysis of the convergence of our algorithm and the inference complexity of our model.

Lemma 2 (Convergence of Algorithm 1)**.** *The Algorithm 1 will converge to a local optimum of objective (7).*

The proof of Lemma 2 can be done based on the fact that each updating step in solving (8) and (11) will only decrease the objective function (7). By using our proposed parameter initialization method, Our experimental results empirically show that our proposed algorithm can converge to a local optimal solution fast.

Algorithms	Memory Cost (bits)			Computation Cost	
	Transformation	Embedding	Classifier	# of BOPS	# of FLOPs
RFF	$\sim d \times p \times 32$	$p \times 32$	$c \times p \times 32$	$-$	$O(d \times p + p)$
Fastfood	$\sim p \times 5 \times 32$	$p \times 32$	$c \times p \times 32$	$-$	$O(plog(d) + p)$
BJLE	$\sim d \times p \times 32$	p	$c \times p \times 32$	$-$	$O(d \times p + p)$
BCSIK	$\sim d \times p \times 32$	p	$c \times p \times 32$	$-$	$O(d \times p + p)$
Our Proposed	$\sim p \times 5 \times 32$	p	$c \times p$	$O(p)$	$O(plog(d))$

Table 1: Memory and Computation Costs for Different Algorithms

Algorithm 1 Memory and computation-efficient kernel SVM via binary embedding and ternary model coefficients

Training

Input: training data set $D = \{\mathbf{x}_i, y_i\}_{i=1}^n$, new dimension p, regularization parameter λ;

Output: transformation parameter $\tilde{\mathbf{R}} \in \mathbb{R}^{d \times p}$, $\mathbf{b} \in \mathbb{R}^p$, $\mathbf{t} \in \mathbb{R}^p$; classification model with ternary coefficients $\mathbf{w}$;

Generate random Gaussian matrix $\tilde{\mathbf{R}}$ as defined in (4), random vector $\mathbf{b}$ and $\mathbf{t}$.
Compute the binary embedding as defined in (5)
Initialization: $\mathbf{w}$, α as shown in (12)
$r = \mathbf{w}^\top \mathbf{w}$. # Pre-Computing regularizer
for $i = 1$ to n **do**
  $h_i = \mathbf{w}^\top \mathbf{z}_i$ # Pre-Computing predictions
end for
repeat
  solve (8) by fixing $\mathbf{w}$
  repeat {Learning $\mathbf{w}$ by fixing α}
    for $j = 1$ to p **do**
      evaluate $L(w_j = -1)$, $L(w_j = 0)$ and $L(w_j = 1)$
      obtain the optimal solution of w_j^* based on (11)
      if $w_j \neq w_j^*$ **then**
        set $w_j^{old} = w_j$ and update $w_j = w_j^*$
        update $h_i = h_i - (w_j^{old} - w_j) z_{ij}$ for each sample
        update $r = r - (w_j^{old})^2 + (w_j)^2$
      end if
    end for
  until objective (9) converge;
until objective (7) converge;

Prediction

Input: a test sample $\mathbf{x}_i$, ternary coefficient $\mathbf{w}$, transformation parameters $\tilde{\mathbf{R}}, \mathbf{b}, \mathbf{t}$;

Output: predicted label $\hat{y}_i$;

compute binary embedding $\mathbf{z}_i$ as defined in (5)
obtain the predicted label by $\hat{y}_i = \text{sign}(\mathbf{w}^\top \mathbf{z}_i)$;

Inference Complexity of Our Model. The main advantage of our model is that it provides memory and computation-efficient model inference. To compare the memory and computation costs for classifying a single input data sample with other methods, we decompose the memory cost into (1) memory cost of transformation parameters; (2) memory cost of embedding; and (3) memory cost of the classification model. We also decompose the computation cost into (1) number of binary operations (# of BOPS) and (2) number of float-point operations (# of FLOPS). To deploy our model, since we do not need to store the Hadamard matrix $\mathbf{H}$ explicitly, we only need $32 \times 3 \times p$ bits to store $\mathbf{S}, \mathbf{G}, \mathbf{\Pi}$. $\mathbf{B}$ can be stored in p bits, $\mathbf{b}$ and $\mathbf{t}$ can be stored in $2 \times 32 \times p$ bits. Therefore, total $\sim p \times 5 \times 32$ bits are needed for storing transformation parameters. For RFF, BCSIK, and Binary Johnson-Lindenstrauss Embedding (BJLE), they need to maintain a large $d \times p$ Gaussian matrix explicitly. Therefore, their memory cost of transformation parameters is $\sim d \times p \times 32$ bits, which is $\sim d$ times larger than our proposed method. As for storing the transformed embedding, RFF needs $p \times 32$ bits where binary embedding methods (i.e., BJLE, BCSIK, and our proposed method) only need p bits. As for storing the classification model, assume that the number of classes is c, and one-vs-all strategy is used. Then, $c \times p \times 32$ bits are needed for full-precision model coefficients, and $c \times p$ bits are needed for binary model coefficients. With respect to the computation complexity of our model, step 1 in prediction needs $O(dlog(d))$ FLOPs, and step 2 needs $O(p)$ BLOPs. The memory and computation costs for different algorithms are summarized in Table 1. Furthermore, since both $\mathbf{w}$ and $\mathbf{z}_i$ are bit vectors, the dot product between them can be replaced by cheap XNOR and POPCOUNT operations, which have been showing to provide $58\times$ speed-ups compared with floating-point dot product in practice (Rastegari et al. 2016).

Experiments

In this section, we compare our proposed method with other efficient kernel SVM approximation methods and binary embedding methods. We evaluate the performance of the following six methods.

- Random Fourier Features (RFF) (Rahimi and Recht 2008): It approximates the shift-invariant kernel based on its Fourier transform.
- Fastfood kernel (Fastfood) (Le, Sarlós, and Smola 2013): It uses the Hadamard transform to speed up the matrix multiplication in RFF.
- Binary Johnson-Lindenstrauss Embedding (BJLE) : It composes JL embedding with sign function (i.e., $\mathbf{z} = \text{sign}(\mathbf{Rx})$) which is a common binary embedding method.
- Binary Codes for Shift-Invariant Kernels (BCSIK) (Raginsky and Lazebnik 2009): It composes the Random Fourier Features from RFF with random sign function.

	metric	usps-b	usps	covtype	webspam	mnist	fashion-mnist
RFF	accuracy	98.63	94.76	84.19	97.79	96.88	87.5
	memory cost	2064KB	2136KB	448KB	2048KB	6328KB	6328KB
	memory red.	1x	1x	1x	1x	1x	1x
Fastfood	accuracy	98.29	94.34	84.82	97.47	96.48	87.55
	memory cost	40KB	112KB	40KB	40KB	112KB	112KB
	memory red.	51x	19x	11x	51x	57x	57x
BJLE	accuracy	97.53	92.53	80.77	96.39	92.95	84.06
	memory cost	2056KB	2128KB	440KB	2040KB	6320KB	6320KB
	memory red.	1x	1x	1x	1x	1x	1x
BCSIK	accuracy	97.85	93.32	81.91	96.41	93.05	84.01
	memory cost	2056KB	2128KB	440KB	2040KB	6320KB	6320KB
	memory red.	1x	1x	1x	1x	1x	1x
Our Method	accuracy	98.01	93.57	81.68	96.34	93.27	83.29
	memory cost	32KB	104KB	32KB	32KB	104KB	104KB
	memory red.	64x	21x	14x	64x	61x	61x
Our Method-b	accuracy	96.57	92.12	78.12	94.75	92.66	82.07
	memory cost	24KB	29KB	24KB	24KB	29KB	29KB
	memory red.	84x	74x	19x	85x	218x	218x

Table 2: Accuracy and memory cost of different models

- Our proposed method with full-precision model coefficients
- Our proposed method with binary model coefficients

To evaluate the performance of these six methods, we use five real-world benchmark datasets. The detailed information about these datasets is summarized in Table 3. The first three datasets are download from LIBSVM website [2], mnist and fashion-mnist are download from openml. [3]

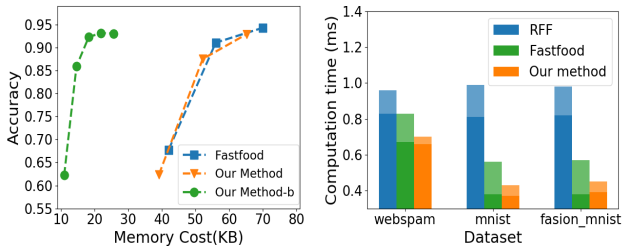
Dataset	class	train size	test size	d
usps-b	2	7,291	2007	256
usps	10	7,291	2007	256
covtype	2	464,810	114,202	54
webspam	2	280,000	70,000	254
mnist	10	60,000	10,000	780
fashion-mnist	10	60,000	10,000	780

Table 3: Experiment datasets

Experiment Results

All the data are normalized by min-max normalization such that the feature values are within the range $[-1, 1]$. The dimension of nonlinear feature mapping p is set to 2048. The σ is chosen from $\{2^{-5}, 2^{-4}, \ldots, 2^{5}\}$. The regularization parameter in both linear SVM and our method is chosen from $\{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$. The prediction accuracy and the memory cost for model inference for all algorithms are reported in Table 2. Memory reduction is also included in Table 2, where 86x means the memory cost is 86 times smaller compared with RFF. As shown in Table 2, RFF gets the best classification accuracy for all five datasets. However, the memory cost for model inference is very high. The binary embedding methods BJLE and BCSIK can only reduce the memory cost by a small amount because the bottleneck is the transformation matrix $\mathbf{R} \in \mathbb{R}^{d \times p}$. Compared with BJLE and BCSIK, Fastfood is memory-efficient for nonlinear feature mapping. However, the memory cost for the classification model could be high for Fastfood, especially when the number of classes is large. Compared with RFF, our proposed method with ternary coefficients can significantly reduce the memory cost from 19x to 218x for model inference.

Next we explore different properties of our proposed algorithm on usps dataset with σ set to 1.



(a) Memory (b) Computation time

Figure 1: Memory and computation efficiency

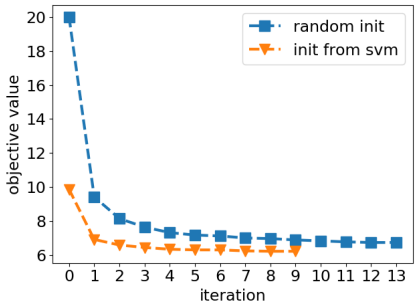
Memory Efficiency of Binary Model. Figure 1(a) illustrates the impact of parameter p. Since RFF, BJLE, BCSIK methods need more than 10 times memory to achieve the same accuracy as Fastfood, so we only compare three memory-efficient methods. As can be seen from it, the accuracy increases as p increases and will converge if p is large enough. Besides, we further compare the memory cost and the prediction accuracy of three Fastfood-based kernel approximation methods. We take the usps dataset as an example.

[2] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

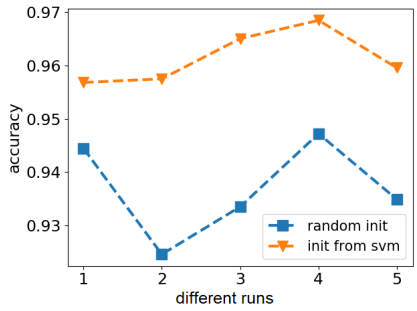
[3] https://www.openml.org/search?type=data

We can set larger p to gain the same performance of the full precision methods shown in Table.2. Furthermore, if we consider the memory cost, our binary model can achieve higher accuracy with the same memory, as shown in 1(a).

Computation Time. We compare the time consumption of our method to process one single sample with the processing time of RFF and the original Fastfood kernel approximation method in Figure 1(b). The dark areas represent the feature embedding time, and the light areas are the prediction time. We show that our method's total computation time can be significantly reduced compared with the other two methods. We have two main observations. First, the Fast Fourier transform in random projection can significantly reduce the projection time, especially when the original data is a high-dimensional dataset (e.g., MNIST and Fashion MNIST). Besides, the binary embedding will add few additional time in feature embedding but will significantly improve the speed in the prediction stage.



(a) Convergence



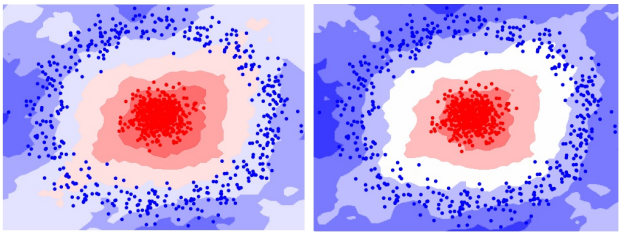
(b) Accuracy

Figure 2: Comparison between random initialization and our proposed initialization method

Convergence of Our Algorithm. In Figure 2(a), we empirically show how our proposed algorithm converges. Here, we compare the two different initialization methods: (1) random initialization; (2) initialization from linear SVM solution. We can observe that using the initialization from a linear SVM solution leads to a slightly lower objective value and converges in a few iterations. Motivated by this observation, we train a linear model on a small subset of data and binarize it as the initial $\mathbf{w}$ for our algorithm in practice.

Effectiveness of SVM Initialization. In Figure 2(b), we further illustrate the effect of our initialization strategy by comparing the prediction accuracy. We can observe that using the initialization from a linear SVM solution leads to higher accuracy and more stable performance compared with the random initialization.

Decision Boundary of Ternary Coefficients. We use the synthetic nonlinear *circle* dataset to illustrate the effect of the ternary coefficients. The *circle* is in two-dimensional space as shown in Figure 3. The blue points in the larger outer circle belong to one class, and the red ones belong to another. We show the decision boundary of using binary and ternary coefficients. As shown in this figure, our proposed binary embedding with linear classifiers can produce effective nonlinear decision boundaries. Besides, as the feature binary embedding might involve some additional noise, the classification model using ternary coefficients can produce better and smoother decision boundary than using binary coefficients.



(a) binary coefficients (b) ternary coefficients

Figure 3: Comparison of the decision boundaries between binary coefficients and ternary coefficients

Conclusion

This paper proposes a novel binary embedding method that can preserve the kernel similarity among data samples. Compared to BCSIK, our proposed method reduces the memory cost from $O(dp)$ to $O(p)$ and the computation cost from $O(dp)$ to $O(plog(d))$ for binary embedding. Besides, we propose a new algorithm to learn the classification model with ternary coefficients. Our algorithm can achieve better generalization accuracy than existing works on learning ternary coefficients since we allow the coefficient to be $\{-1, 0, 1\}$ during the training stage. Our proposed algorithm can be easily implemented and applicable to other types of loss function and regularizer. We also provide a detailed analysis of the convergence of our algorithm and the inference complexity of our model. We evaluate our algorithm based on five large benchmark datasets and demonstrate our proposed model can build accurate nonlinear SVM models with memory cost less than 30KB on all five datasets.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments and valuable suggestions on our paper. This work was supported by NSFC 61906161.

References

Alizadeh, M.; Fernández-Marqués, J.; Lane, N. D.; and Gal, Y. 2019. An Empirical study of Binary Neural Networks' Optimisation. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* .

Boyd, S.; Boyd, S. P.; and Vandenberghe, L. 2004. *Convex optimization*. Cambridge university press.

Gong, Y.; Kumar, S.; Verma, V.; and Lazebnik, S. 2012. Angular quantization-based binary codes for fast similarity search. In *Advances in neural information processing systems*, 1196–1204.

Hsieh, C.-J.; Si, S.; and Dhillon, I. S. 2014. Fast prediction for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 3689–3697.

Johnson, W. B.; Lindenstrauss, J.; and Schechtman, G. 1986. Extensions of Lipschitz maps into Banach spaces. *Israel Journal of Mathematics* 54(2): 129–138.

Kumar, A.; Goyal, S.; and Varma, M. 2017. Resource-efficient machine learning in 2 KB RAM for the internet of things. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1935–1944. JMLR. org.

Kusupati, A.; Singh, M.; Bhatia, K.; Kumar, A.; Jain, P.; and Varma, M. 2018. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*, 9017–9028.

Lan, L.; Wang, Z.; Zhe, S.; Cheng, W.; Wang, J.; and Zhang, K. 2019. Scaling Up Kernel SVM on Limited Resources: A Low-Rank Linearization Approach. *IEEE transactions on neural networks and learning systems* 30(2): 369–378.

Le, Q.; Sarlós, T.; and Smola, A. 2013. Fastfood-computing hilbert space expansions in loglinear time. In *International Conference on Machine Learning*, 244–252.

Li, F.; Zhang, B.; and Liu, B. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711* .

Liu, Z.-G.; and Mattina, M. 2019. Learning low-precision neural networks without straight-through estimator (STE). In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 3066–3072. AAAI Press.

Needell, D.; Saab, R.; and Woolf, T. 2018. Simple classification using binary data. *The Journal of Machine Learning Research* 19(1): 2487–2516.

Raginsky, M.; and Lazebnik, S. 2009. Locality-sensitive binary codes from shift-invariant kernels. In *Advances in neural information processing systems*, 1509–1517.

Rahimi, A.; and Recht, B. 2008. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, 1177–1184.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, 525–542. Springer.

Ravi, S. 2019. Efficient On-Device Models using Neural Projections. In *International Conference on Machine Learning*, 5370–5379.

Shen, F.; Mu, Y.; Yang, Y.; Liu, W.; Liu, L.; Song, J.; and Shen, H. T. 2017. Classification by retrieval: Binarizing data and classifiers. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, 595–604.

Yin, P.; Lyu, J.; Zhang, S.; Osher, S.; Qi, Y.; and Xin, J. 2019. Understanding straight-through estimator in training activation quantized neural nets. In *International Conference on Learning Representations*.

Yu, F. X.; Bhaskara, A.; Kumar, S.; Gong, Y.; and Chang, S.-F. 2017. On binary embedding using circulant matrices. *The Journal of Machine Learning Research* 18(1): 5507–5536.