Kernel SVM Based on Binary Embedding and Optimized Random Fourier Features

Junhong Zhang

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 to preserve 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, the experimental results on real-world datasets have demonstrated that our proposed method can build accurate nonlinear SVM models.

Keywords: kernel approximation, binary embedding, .

1 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. 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 on large datasets. To address this problem, the kernel approximation methods was proposed to reduce the the memory and computation costs of kernel SVM in the past decade [1]. It aims to construct the nonlinear feature mapping $\mathbf{z} = \phi(\mathbf{x}) : \mathbb{R}^d \mapsto \mathbb{R}^p$ such that

$$\mathbf{z}_i^\top \mathbf{z}_j \approx k(\mathbf{x}_i, \mathbf{x}_j). \tag{1}$$

where $k(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ denotes kernel function. Then we can apply linear SVM to implement nonlinear classification. To obtain promising classification accuracy, the dimensionality of nonlinear feature mapping $\mathbf{z}$ needs to be large. Therefore, 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 or IoT device deployment since it must store high-dimensional features in float point format. Recently, binary embedding methods have been widely used to reduce the memory burden in 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 at least 32 time compared with storing full precision embedding. The common way for binary embedding is to apply sign$(\cdot)$ function to obtain the binary code with only $+1$ and -1. Based on this, we intent to capture the binary representations $\mathbf{z} \in \mathbb{R}^p$ for kernel similarity preserving, i.e., the approximation (1) can be reached.

Reference [2] explore a novel binary embedding method with shift-invariant kernel approximation. The proposed method is built on Binary Codes for Shift-invariant kernels (BCSIK) [3], but the memory and computation costs are significantly reduced. In short, the computation burden and memory costs are significantly reduced. This report will explain the technical details of [2] and present improvements. We implement the algorithms and further evaluate the classification performance.

2 Related works

In this section, we will briefly review the kernel approximation method, i.e., Random Fourier Feature (RFF), and the binary embedding based kernel approximation method, i.e., Binary Codes for the Shift-Invariant Kernels (BCSIK).

2.1 Random Fourier Feature

Reference [1] proposed a randomized feature learning method to approximate shift-invariant kernel. Consider a shift-invariant kernel $k(\mathbf{x},\mathbf{y}) = k(\mathbf{x}-\mathbf{y})$ and its Fourier transform:

$$k(\mathbf{x}-\mathbf{y}) = \int_{\mathbb{R}^d} p(\mathbf{w}) \exp\left(i\mathbf{w}^\top(\mathbf{x}-\mathbf{y})\right) \mathrm{d}\mathbf{w} = \mathbb{E}_{\mathbf{w}\sim p(\mathbf{w})}[\zeta_\mathbf{w}(\mathbf{x})\bar{\zeta}_\mathbf{w}(\mathbf{y})]. \tag{2}$$

where i is the imaginary unit and $\zeta_\mathbf{w}(\mathbf{x}) = \exp(i\mathbf{w}^\top\mathbf{x})$. Obviously, $\zeta_\mathbf{w}(\mathbf{x})\bar{\zeta}_\mathbf{w}(\mathbf{y})$ is the unbiased estimate of $k(\mathbf{x},\mathbf{y})$. For standard Gaussian kernel, i.e., $k(\mathbf{x},\mathbf{y}) = \exp(-\|\mathbf{x}-\mathbf{y}\|_2^2/2)$, we have

$$p(\mathbf{w}) = \frac{1}{(2\pi)^{p/2}} \exp\left(-\frac{\|\mathbf{w}\|_2^2}{2}\right). \tag{3}$$

That is to say, $\mathbf{w} \sim \mathcal{N}(0,\mathbf{I})$. Based on this idea, we can construct the randomized featrue of the Gaussian kernel as

$$\mathbf{z}(\mathbf{x}) = \sqrt{\frac{2}{p}} \cos\left(\mathbf{W}^\top\mathbf{x}+\mathbf{b}\right), \quad \mathbf{W} \in \mathbb{R}^{d\times p},\ \mathbf{b} \in \mathbb{R}^p. \tag{4}$$

where $\mathbf{w}_i \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ and $b_i \sim$ Uniform$(0, 2\pi)$ and cos$(\cdot)$ is element-wise operator. [1] have proved that $\mathbf{z}(\mathbf{x})^\top\mathbf{z}(\mathbf{y}) \approx k(\mathbf{x},\mathbf{y})$, which is concluded as the following theorem.

Theorem 1. *Let $\mathcal{M}$ be a compact subset of $\mathbb{R}^d$ with diamter* diam$(\mathcal{M})$*. Then we have*

$$\Pr\left[\sup_{\mathbf{x},\mathbf{y}\in\mathcal{M}} |\mathbf{z}(\mathbf{x})^\top\mathbf{z}(\mathbf{y}) - k(x,y)| \geq \epsilon\right] \leq 2^8 \left(\frac{\sigma_p^2 \mathrm{diag}(\mathcal{M})}{\epsilon}\right)^2 \exp\left(-\frac{p\epsilon^2}{4(d+2)}\right) \tag{5}$$

where $\sigma_p^2 = \mathbb{E}_{\mathbf{w}\sim p(\mathbf{w})}(\mathbf{w}^\top\mathbf{w})$ is the second moment of Fourier transform of $k(\cdot)$.

2.2 Binary Codes for the Shift-Invariant Kernels

Reference [3] proposed the following way to construct the binary representation to approximate the implicit mapping of the Gaussian kernel:

$$\mathbf{z}(\mathbf{x}) = \mathrm{sign}(\Phi(\mathbf{x})+\mathbf{t}) = \mathrm{sign}(\cos(\mathbf{W}^\top\mathbf{x}+\mathbf{b})+\mathbf{t}). \tag{6}$$

where $w_{ij} \sim \mathcal{N}(0, \sigma^{-2})$, $b_i \sim \text{Uniform}(0, 2\pi)$, and $t_i \sim \text{Uniform}(-1, 1)$. It is obvious that $\Phi(\mathbf{x}) = \cos(\mathbf{W}^\top \mathbf{x} + \mathbf{b})$ is the RFF and BCSIK is the signed version of RFF (with a random bias). Besides, the kernel similarity can be well preserved with (6), which is guaranteed by the following lemma.

Lemma 1. *Define*

$$h_1(u) = \frac{4}{\pi^2}(1-u), \quad h_2(u) = \min\left\{\frac{1}{2}\sqrt{1-u^2}, \frac{4}{\pi^2}\left(1 - \frac{2}{3}u\right)\right\}, \quad u \in [0, 1]. \tag{7}$$

Fixing $\delta, \epsilon \in (0, 1)$, for any dataset $\{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$, with probability at least $1 - \epsilon$ and $p \geq \log(n^2/\epsilon)/(2\delta^2)$, we have

$$h_1(k(\mathbf{x}_i, \mathbf{x}_j)) - \delta \leq \frac{1}{p} d_H(\mathbf{z}_i, \mathbf{z}_j) \leq h_2(k(\mathbf{x}_i, \mathbf{x}_j)) + \delta. \tag{8}$$

where $d_H(\mathbf{z}_i, \mathbf{z}_j) = (p - \mathbf{z}_i^\top \mathbf{z}_j)/2$ is the Hamming distance between $\mathbf{z}_i$ and $\mathbf{z}_j$.

3 Methodology

In this section, we present the method proposed in [2]. Then we will present a novel strategy to improve this method.

3.1 Reduced-memory BCSIK (RM-BCSIK)

For on-device model deployment, the bottleneck of obtaining binary embedding is the matrix-vector multiplication of $\mathbf{W}^\top \mathbf{x}$, which requires $O(dp)$ time and space. Since we usually set $p \gg d$ to pursue better nonlinear classification performance, the computation and memory costs are expensive for the embedded devices or IoT devices. To address this problem, we propose to generate the Gaussian random matrix via the techniques in Fastfood [4], i.e, Reduced-memory BCSIK. Assume $d = 2^q$ with q is the positive integer (we can always ensure this by zero paddings), then we can construct a $d \times d$ Gaussian random matrix via

$$\mathbf{V}^\top = \frac{1}{\sigma\sqrt{d}} \mathbf{S}\mathbf{H}\mathbf{G}\mathbf{\Pi}\mathbf{H}\mathbf{B}. \tag{9}$$

where $\mathbf{S}, \mathbf{G}$ and $\mathbf{B}$ are diagonal matrices, and $s_{ii} \sim \text{Uniform}(0, 1)$, $g_{ii} \sim \mathcal{N}(0, 1)$, $b_{ii} \sim \text{Rad}$, and Rad denotes the Rademacher distribution:

$$p(\sigma = -1) = p(\sigma = -1) = \frac{1}{2}. \tag{10}$$

$\mathbf{\Pi}$ is the random permutation matrix, and $\mathbf{H}$ is the Walsh-Hadamard matrix defined as

$$\mathbf{H}_d = \begin{bmatrix} \mathbf{H}_{d/2} & \mathbf{H}_{d/2} \\ \mathbf{H}_{d/2} & -\mathbf{H}_{d/2} \end{bmatrix}, \quad \mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{11}$$

Consider that for any vector $\mathbf{x} \in \mathbb{R}^d$, computing $\mathbf{H}\mathbf{x}$ is equivalent to compute Fast Walsh-Hadamard Transform (FWHT), which costs time of $O(d \log d)$. Besides, FWHT does not requires storing the Hadamard matrix, which means the computation can be done with $O(1)$ spaces.

Based on the above method, we can create the $d \times p$ random Gaussian matrix by stacking them together.

$$\mathbf{W} = [\mathbf{V}_1, \mathbf{V}_2, \cdots, \mathbf{V}_{p/d}]. \tag{12}$$

Algorithm 1 Reduced-memory BCSIK

```
Require: $\mathbf{x} \in \mathbb{R}^d, p > 0$.
Ensure: $\mathbf{z}(\mathbf{x}) = \mathrm{sign}(\cos(\mathbf{W}^\top \mathbf{x} + \mathbf{b}) + \mathbf{t})$.
 1: Initialize $\mathbf{z} = []$.
 2: for $l = 1 : p/d$ do
 3:     $\tilde{\mathbf{x}} \leftarrow \mathbf{x}$
 4:     for $i = 1 : d$ do
 5:         $\tilde{x}_i \leftarrow b\tilde{x}_i$, where $b \sim$ Rad.
 6:     end for
 7:     $\tilde{\mathbf{x}} \leftarrow \mathrm{fwht}(\tilde{\mathbf{x}})$.
 8:     $\tilde{\mathbf{x}} \leftarrow \mathrm{shuffle}(\tilde{\mathbf{x}})$.
 9:     for $i = 1 : d$ do
10:         $\tilde{x}_i \leftarrow g\tilde{x}_i$, where $g \sim \mathcal{N}(0, 1)$.
11:     end for
12:     $\tilde{\mathbf{x}} \leftarrow \mathrm{fwht}(\tilde{\mathbf{x}})$.
13:     for $i = 1 : d$ do
14:         $x_i \leftarrow s\tilde{x}_i$, where $s \sim$ Uniform$(0, 1)$.
15:     end for
16:     $\mathbf{z} \leftarrow \mathrm{concat}(\mathbf{z}, \tilde{\mathbf{x}})$.
17: end for
18: for $i = 1 : p$ do
19:     $z_i \leftarrow z_i + t$, where $t \sim$ Uniform$(-1, 1)$.
20: end for
21: $\mathbf{z} \leftarrow \mathrm{sign}(\mathbf{z}/(\sigma\sqrt{d}))$
```

Therefore, we can compute

$$\mathbf{z}^{(i)} = \mathbf{V}_i^\top \mathbf{x} = \frac{1}{\sigma\sqrt{d}} \mathbf{SHG\Pi HBx}. \tag{13}$$

and the random feature can be obtained via

$$\mathbf{W}^\top \mathbf{x} = [\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \cdots, \mathbf{z}^{(p/d)}]. \tag{14}$$

We can summarize the above procedures as Algorithm 1. Based on the above analysis, we can see that the proposed algorithm reduce the space complexity of BCSIK from $O(dp)$ into $O(p)$, and reduce the time complexity from $O(dp)$ into $O(p \log d)$.

3.2 Improvements

The above methods have limitations that ignore the label information of the dataset. Moreover, the construction of $\mathbf{W}$ and $\mathbf{b}$ are data-independent. Therefore, the learned feature might not be optimal for classification. Inspired by [5], we intent to optimize the random binary embedding to align the label information. That is to say, we find a distribution that aligns the feature mapping with label $\mathbf{y}$. We first consider binary-classification prblem, where $y_i \in \{-1, 1\}$. Define a kernel k_Q as

$$k_Q(\mathbf{x}, \mathbf{x}') = \int \phi(\mathbf{x}; \mathbf{w})\phi(\mathbf{x}'; \mathbf{w})q(\mathbf{w})\mathrm{d}\mathbf{w}. \tag{15}$$

where $q(\mathbf{w})$ is the probability density function of distribution Q. The desired kernel should be matches the label information, which leads to the following kernel alignment problem.

$$\max_{Q \in \mathcal{P}} \sum_{i,j} k_Q(\mathbf{x}_i, \mathbf{x}_j) y_i y_j. \tag{16}$$

where $\mathcal{P}$ denotes

$$\mathcal{P} = \{Q : D_f(Q||P_0) \leq \rho\}, \quad D_f(Q||P_0) = \int f\left(\frac{p(\mathbf{w})}{q(\mathbf{w})}\right) q(\mathbf{w})\mathrm{d}\mathbf{w} \tag{17}$$

where P_0 is the original distribution of Fourier random features, and $f(\cdot)$ is the divergence function satisfying $f(1) = 0$. For simplication, we use χ^2-divergence $f(t) = t^2 - 1$ in the following discussion. However, (16) is intractable since it is infinite dimensional. The alternative way is to approximate the integral operator with discrete sum over $\mathbf{w}_i \sim P_0$. Define $\bar{\mathcal{P}} = \{q : D_f(q||\mathbf{1}/N_w) \leq \rho\}$, and $\mathbf{1}/N_w$ is the empirical approximation of P_0. Therefore, we have the following empirical version of (16):

$$\max_{\mathbf{q}\in\bar{\mathcal{P}}} \sum_{i,j} y_i y_j \sum_{l=1}^{N_w} q_l \phi(\mathbf{x}_i; \mathbf{w}_l)\phi(\mathbf{x}_j; \mathbf{w}_l). \tag{18}$$

Note that the objective function can be rewritten as

$$\begin{aligned} \sum_{i,j} y_i y_j \sum_{l=1}^{N_w} q_l \phi(\mathbf{x}_i; w_l)\phi(\mathbf{x}_j; w_l) &= \sum_{l=1}^{N_W} q_l \left(\sum_{i=1}^{n} y_i \phi(\mathbf{x}_i; w_l)\right)^2 & (19) \\ &= \mathbf{q}^\top \left((\Phi\mathbf{y}) \odot (\Phi\mathbf{y})\right). & (20) \end{aligned}$$

where $\phi_{il} = \phi(\mathbf{x}_i; w_l)$ and $\odot$ denotes Hardamard product. Then we can write the partial Lagrangian of (18)

$$L(\mathbf{q}, \lambda) = \mathbf{q}^\top \left((\Phi\mathbf{y}) \odot (\Phi\mathbf{y})\right) - \lambda(D_f(\mathbf{q}||\mathbf{1}/N_w) - \rho) \tag{21}$$

By solving the dual problem $\min_{\lambda\geq 0} \max_{\mathbf{q}\in\Delta} L(\mathbf{q}, \lambda)$, where $\Delta = \{\mathbf{q} \in \mathbb{R}_+^{N_w} : \mathbf{q}^\top\mathbf{1} = 1\}$, we can obtain the optimal λ, and furhter compute the optimal $\mathbf{q}$ via

$$\max_{\mathbf{q}\in\Delta} \mathbf{q}^\top\mathbf{v} - \frac{\lambda}{N_w} \sum_{l=1}^{N_w} (N_w q_l)^2, \tag{22}$$

where $\mathbf{v} = ((\Phi\mathbf{y}) \odot (\Phi\mathbf{y}))$. The solution of this problem is given by

$$q_l = \left(\frac{v_l}{\lambda N_w} + \tau\right), \tag{23}$$

where τ is chosen so that $\sum_{l=1}^{N_w} q_l = 1$. With the optimal $\mathbf{q}$, we can sample the optimized weight matrix from Q, i.e., $\tilde{w}_{ij} \sim Q$. Therefore, we can construct the random binary features as

$$\mathbf{z}(\mathbf{x}) = \mathrm{sign}(\cos(\tilde{\mathbf{W}}^\top\mathbf{x} + \mathbf{b}) + \mathbf{t}). \tag{24}$$

where $\mathbf{b}, \mathbf{t}$ are defined similarly with the previous section. The optimized distribution $\mathbf{q}$ is always sparse, which means the random features corresponding to the zero probability of $\mathbf{q}$ are eliminated. This property naturally decreases the dimension of the random features and significantly reduce the time and space costs of the later computation.

4 Implementation details

In this section, we will present the key codes of the implementation as well as the improvements of the original paper.

4.1 The Implementation of RM-BCSIK

We implemented RM-BCSIK algorithm in MATLAB (version: R2021b or later). The source codes of constructing matrix **W** and **b** are shown as follows.

```matlab
function [W, b] = rmbcsik(d, sigma, p)
% Binary Codes for Shift-invariant kernels (Gaussian kernel)
%
%    Input:
%        d: the original dimension of sample.
%    sigma: the bandwidth of Gaussian kernel.
%        p: the dimension of random features
%
%   Output:
%        W: the weight matrix, Wij ~ N(0,sigma^{-2})
%        b: the bias vector  bij ~ Uniform(0,2*pi).
%
%   Usage:
%      [n, d] = size(X);   % each row of X is a sample.
%      [W, b] = rmbcsik(d);
%      Z = makeRFF(d, W, b, X);
%
%   Written by Junhong Zhang, 2022.11.4

d = 2 ^ ceil(log(d)/log(2));
W = zeros(d, p);

ptr = 1;
for ii = 1:floor(p/d)
    S = rand(d, 1);
    G = randn(d, 1);
    B = sign(2*rand(d, 1)-1); % rademacher random var

    V = fwht(diag(B)) * d; % Fast Hadamard Transform
    V = V(randperm(d), :); % re-order Z
    V = G .* V;
    V = fwht(V) * d;
    V = S .* V / (sigma * sqrt(d));

    if p - ptr >= d
        W(:, ptr:ptr+d-1) = V;
    else
        W(:, ptr:p) = V(:, 1:p-ptr+1);
    end
    ptr = ptr + d;
end
b = rand(1,p) * 2*pi;
end
```

Then we call the interface `makeRFF()` to create the random features, which is defined as

```matlab
function Z = makeRFF(W, b, X)
    D = size(W, 2);
    Z = sqrt(2/D)*cos(bsxfun(@plus,X*W,b));
end
```

4.2 Kernel optimization

The codes of kernel optimization [5] are available on GitHub[1]. However, they only consider the case of binary-class classification. To overcome this limitation, we improve the codes to support multi-class classification. Note that for multi-class dataset with $y_i = \{1, 2, \cdots, c\}$, we can set the following label

[1]https://github.com/KaikaiZhao/LearningKernelsWithRandomFeatures

matrix $\bar{\mathbf{Y}}$:

$$\bar{y}_{ij} = \begin{cases} 1, & y_i = j, \\ -1, & \text{otherwise.} \end{cases} \tag{25}$$

Hence the objective function of kernel alignment can be re-defined as

$$\sum_{i,j} \bar{\mathbf{y}}_i^\top \bar{\mathbf{y}}_j \sum_{l=1}^{N_w} q_l \phi(\mathbf{x}_i; w_l)\phi(\mathbf{x}_j; w_l) = \sum_{l=1}^{N_w} q_l \left\| \sum_{i=1}^{n} \bar{\mathbf{y}}_i \phi(\mathbf{x}_i; w_l) \right\|^2 \tag{26}$$

$$= \mathbf{q}^\top \mathbf{v}. \tag{27}$$

where $v_l = \|\sum_{i=1}^{n} \phi(\mathbf{x}_i; w_l)\bar{\mathbf{y}}_i\|^2$. This objective function shares the same format with (18), so we can use the same method to obtain the optimal $\mathbf{q}$. Therefore, we present the codes of creating the optimal kernel distribution, i.e., Q in section 3.2.

```matlab
function [W_opt, b_opt, q, q_distrib] = optimizeGaussianKernel(Xtrain, ytrain, Nw, rho, tol, sigma)
    d = size(Xtrain, 1);

    % generate standard Gaussian random features
    W = randn(d, Nw) / sigma;
    b = rand(1,Nw)*2*pi;

    % set up/solve the problem using the chi-square divergence
    Phi = cos(bsxfun(@plus,Xtrain*W, b));

    uy = unique(ytrain);
    if numel(uy) == 2     % binary classification
        yy = (2*ytrain - (uy(1)+uy(2))) / (uy(2)-uy(1)); % scale to {-1,+1}
    else
        yy = ind2vec(ytrain); % one-hot label matrix
        yy = 2*yy - 1;       % construct \bar{Y}.
    end
    Ks = Phi*yy;
    Ks = sum(Ks.^2, 2);
    q_temp = linear_chi_square(-Ks, 1/Nw*ones(Nw,1), rho/Nw, tol);

    % eliminate the zero probability features.
    idx = q_temp > eps;
    q = q_temp(idx);
    W_opt = W(:,idx);
    b_opt = b(idx);
    q_distrib = cumsum(q/sum(q));
end
```

5 Experiments

In this section, we evaluate the performance of the implemented methods, which is compared with other closely related kernel SVM methods.

5.1 Experimental setup

The experiments are conducted on four real-world datasets. The statistical properties are shown in Table 1. We randomly split each dataset into training set and testing set with proportion of 6:4, and repeated the experiments 10 times to compute the average accuracy as well as the standard deviation. In the experiments, the proposed RM-BCSIK method will be compared with full-precision RFF, the binary embedding with kernel optimization (i.e., the improvement in Section 3.2), and the exact kernel SVM. We use LIBLINEAR [6] and LIBSVM [7] to implement linear SVM and kernel SVM, respectively.

Table 1: The basic information of the used datasets.

Dataset	#class	#feature	#sample
Sonar	2	60	208
DNA	3	180	2000
BinaryAlpha	36	320	1404
USPS	10	256	7291

We implement MATLAB interface for learning different random fourier features, so we can use a single function to generate different type of random features.

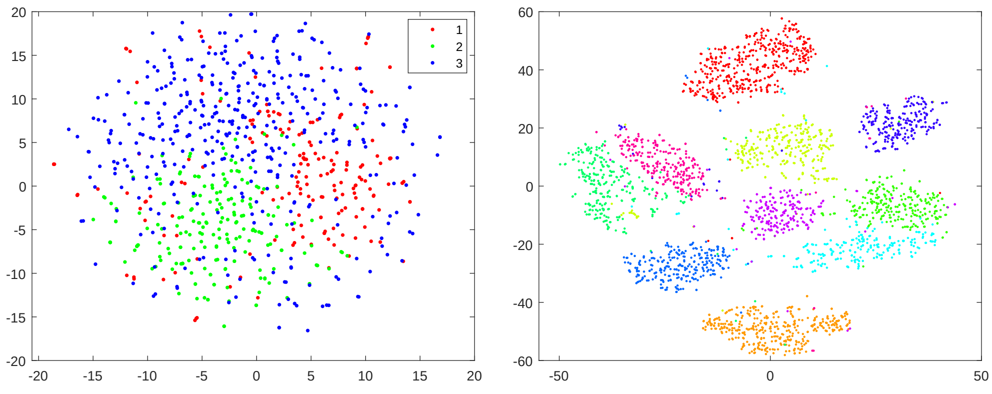
```matlab
function model = rff_svm(Xtrain, Ytrain, param, varargin)
%rff_svm: SVM with random fourier features.
%   Input:
%      Xtrain: training data, each row is a sample.
%      Ytrain: label of data, a col vector.
%      param: the hyper-parameters.
%      options (varargin):
%             FeatureType: could be RFF/RM-BCSIK/Optimized. Corresponds to
%                         different type of random features.
%                    sign: could be true/false. Generate binary code or not.
%
%   Output: learned model.
%
%   Written by Junhong Zhang, 2022.11.10.
p = inputParser;
p.KeepUnmatched(true);
p.addParameter("FeatureType", "Optimized", ...
    @(x) assert(ismember(lower(x), ["rff", "rm-bcsik", "optimized"]), ...
               "FeatureType␣should␣be␣RFF/RM-BCSIK/Optimized."));
p.addParameter("sign", true, ...
    @(x) assert(islogical(x), ...
               "The␣sign␣flag␣should␣be␣true␣or␣false."));
parse(p, varargin{:});

opt = p.Results;
switch lower(opt.FeatureType)
    case "rff"
        W = randn(size(Xtrain,2), param.dim) / param.sigma;
        b = rand(1, param.dim) * 2*pi;
        Ztrain = makeRFF(W, b, Xtrain);
        D = param.dim;
        model.name = "rf-ff";
    case "rm-bcsik"
        m = size(Xtrain, 2);
        [W, b] = rmbcsik(m, param.sigma, param.dim);
        W = W(1:m, :);                % zero paddings
        Ztrain = makeRFF(W, b, Xtrain);
        model.name = "rf-rmbcsik";
        D = param.dim;
    case "optimized"
        [W, b, alpha, alpha_distrib] = optimizeGaussianKernel(Xtrain, Ytrain, ...
            param.dim, param.rho, param.tol, param.sigma);
        D = length(alpha);
        [D, W, b] = createOptimizedGaussianKernelParams(D, W, b, alpha_distrib);
        Ztrain = makeRFF(W, b, Xtrain);
        model.name = "rf-optimized";
end
if opt.sign
    Ztrain = sign(Ztrain);
end

svm = liblineartrain(Ytrain, sparse(Ztrain), '-s␣3␣-q');
model.svm = svm;
model.rffW = W; model.rffb = b; model.rffD = D;
end
```

Table 2: The classification performance of different nonlinear SVM on different datasets.

Datasets	RFF+SVM	this paper [2]	ours (Section 3.2)	exact kernel SVM
Sonar	82.89±4.99	80.84±5.45	83.49±2.90	82.05±5.37
DNA	93.48±1.94	91.84±3.92	93.62±0.63	94.25±0.55
BinaryAlpha	68.79±2.93	66.69±3.68	70.07±1.14	70.89±1.66
USPS	96.54±0.29	95.36±0.39	95.73±0.25	96.44±0.24



(a) DNA (b) USPS

Figure 1: Visualization of the binary codes of our method on DNA and USPS datasets.

5.2 Results and analysis

The classification accuracy of different methods are shown in Table 2. Just for reminder, in this table, "RFF+SVM" denotes the nonlinear SVM which uses the full-precision random Fourier feature, "this paper" denotes the method proposed in [2], i.e., the algorithm discussed in Section 3.1, "ours" denotes our improved method in Section 3.2, and "exact kernel SVM" denotes nonlinear SVM directly implemented via kernel trick [7]. Beside, we should point out that "this paper" and "ours" are both based on random binary embedding. As shown in Table 2, the method in [2] can obtain promising classification performance, and our method always reach better performance than [2]. Note that the performance of our method has no significant difference with the exact kernel SVM, which suggests that our method can still reach promising performance without full-precision data representation. Therefore, the proposed method not only significantly reduce the memory burden, but preserve the discriminative information in the original data.

We also visualize the learned binary embedding of our method by T-SNE algorithm. As shown in Figure 1, the points with different color denotes the data from different classes. We can see that the class separability of the data is well preserved after the binary embedding. This also suggests the effectiveness of the proposed improved method.

6 Conclusion and future work

We implement a novel binary embedding method based on random Fourier feature for kernel approximation and nonlinear classification [2]. Based on this, an improved method is presented in this report to enhance the classification performance, which adjust the random features via kernel alignment strategy. The numerical experiments show that the proposed method could obtain good classification performance.

References

[1] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," *Advances in Neural Information Processing Systems*, vol. 20, 2007.

[2] Z. Lei and L. Lan, "Memory and computation-efficient kernel svm via binary embedding and ternary model coefficients," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 8316–8323, 2021.

[3] M. Raginsky and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels," *Advances in Neural Information Processing Systems*, vol. 22, 2009.

[4] Q. Le, T. Sarlós, A. Smola, *et al.*, "Fastfood-approximating kernel expansions in loglinear time," in *Proceedings of the International Conference on Machine Learning*, vol. 85, p. 8, 2013.

[5] A. Sinha and J. C. Duchi, "Learning kernels with random features," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[6] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *the Journal of Machine Learning research*, vol. 9, pp. 1871–1874, 2008.

[7] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, pp. 1–27, 2011.