A Study on Machine Learning Algorithms Using Feature Interactions

Kyohei Atarashi

Division of Computer Science and Information Technology
Graduate School of Information Science and Technology
Hokkaido University
January, 2021

Contents

Acknowledgments

My acknowledgments have to begin with my supervisor, Satoshi Oyama. He has given me opportunities to research freely and many insightful comments on the research. I would not have finished any research in this dissertation without his helpful advice and patient guidance. I am very glad to learn and research as his student and I would like to express my sincere gratitude to him.

I would also like to express my special gratitude to Masahito Kurihara. Through discussions with him, I have learned a lot of things, especially, how to consider research topics, advance research, and write research papers.

I would like to thank Masahito Yamamoto, Hidenori Kawamura, and Tetsuo Ono for their comments on my presentation.

At the end of my master's course, I visited the University of Massachusetts Amherst (UMass). The main results in Chapter 4 were developed at that time. I am grateful for Subhransu Maji who is an associate professor at UMass and was my mentor. His careful advice was very helpful and the collaboration with him was valuable for me. I appreciate the support of the Global Station for Big Data and Cybersecurity, especially Masaharu Yoshioka and Hiroki Arimura, to my UMass visiting and daily research. It is a project of the Global Institution for Collaborative Research and Education at Hokkaido University.

I am also grateful for Kazune Furudo, Gota Gando, Keiki Zen, Tomoumi Takase, and Jing Song. Discussions with them have helped and motivated me. I also want to thank all other members of the Intelligence Software Laboratory. I would not have enjoyed my laboratory life without them.

Finally, I want to thank my friends and family.

This work was partially supported by JSPS KAKENHI Grant Number JP20J13620.

Chapter 1

Introduction

1.1 Background

Machine learning is the study for automatically extracting rules from data and using such rules to predict future data or decision making and machine learning methods have been used in many real-world applications, for example, image classification, text classification, recommender systems, image generation, visual-question answering, video games, self-driving, and so on [45, 6, 26]. Mathematically, data is typically represented as a set of vectors called *feature vectors* or a set of tuples of a feature vector and a scalar, and the extraction of (learning) a rule is formulated as a function approximation problem, i.e., a numerical optimization problem. Recently, it is required for machine learning methods not only to perform accurately but also to be interpretable (explainable) [1]. Unfortunately, it is difficult to learn both an interpretable and accurate model: accurate models are often complicated.

There are many machine learning predictive models: linear models, linear basis function basis models, the kernel regression model, trees and tree-based ensemble models, neural networks, and so on [6, 45, 26]. Among them, models based on *feature interactions* (combinations) (e.g., $x_i x_j$, where $\boldsymbol{x}$ is a vector that contains some information on an input instance wished to predict), e.g, polynomial regression models, kernel regression models, and factorization machines (FMs) [60, 61], have been used in many real-world applications and recently have attracted attention because of their high interpretability and performance. For example, consider an academic paper classification problem. This problem is just a text classification problem and hence a basic vector representation of data is bag-of-words: $x_i \in \mathbb{N}$ is an occurrence of a word in the title. Feature interactions are easy to interpret: in this case, $x_i x_j > 0$ implies both i-th word and j-th word are included in the title of the paper. Feature interactions can be useful for accurate prediction, for example, a paper including both "kernel" and "learning" in its title is surely a machine learning paper although a Linux kernel research paper can include "kernel" in its title and an educational research paper can include "learning" in its title.

However, there are some issues in methods based on feature interactions:

1. **Scalability.** From the point of view of computational costs, it is difficult to use the existing methods based on feature interactions when both the number of observed (training) instances and the number of features are large. For example, a second-order polynomial regression model requires $O(d^2)$ time for evaluation and $O(Nd^4 + d^6)$ time for learning, where N is the number of observed instances and d is the number of features. A kernel regression model (with polynomial kernels) requires $O(Nd)$

time for evaluation and $O(N^2d + N^3)$ time for learning. FMs require $O(dk)$ time for evaluation, where $k \in \mathbb{N}_{>0}$, $k \ll d$ is a hyperparameter, so they can be used for both N and d are large. However, their optimization problem is a non-convex optimization problem, so a global optimum is not obtained in general.

2. **Interpretability and accuracy in high-dimensional applications.** In the existing methods, the number of feature interactions grows in a polynomial or exponential order w.r.t the number of features d. Fortunately, in many models based on feature interactions, the importance of each feature interaction can be computed efficiently. However, it is difficult to obtain higher-level knowledge when d is large. For example, in order to find important feature interactions such that a machine learning user does not regard them as important but a learned model does, it is basically required to enumerate all the used feature interactions. In the existing methods, the enumeration of all the used feature interactions might be prohibitive from the point of view of the computational cost. Moreover, even if the enumeration can be done, it might be impossible to find unexpectedly important feature interactions for a machine learning user (e.g., when $d = 100,000$, even the number of second-order feature interactions is $4,999,950,000$). Furthermore, the existing methods typically use all feature interactions or all specific-order feature interactions. Such interactions can include some (or many) irrelevant interactions for prediction and the use of such interactions makes the performance of a learned model poor.

1.2 Contributions and Organization

The goal of this paper is to develop more efficient, accurate, and interpretable machine learning algorithms (models and learning algorithms) using feature interactions than the existing algorithms. The contributions of this paper are as follows.

- We propose accurate models for feature-based link prediction (Chapter 3). Feature-based link prediction is the computational problem of determining whether two given objects are linked or not from feature vectors of two objects, and it includes many real-world applications: recommender systems, face verification, protein-protein interaction prediction, author name disambiguation, and so on. In some applications (e.g, protein-protein interaction prediction and author name disambiguation), feature interactions from the same object are irrelevant. The proposed methods use higher-order feature interactions only across two objects and therefore can be more accurate than the existing methods using only second-order feature interactions across two objects or using feature interactions not only across objects but also from the same objects. We also extend the proposed methods to deep-neural-network-based methods for more accurate prediction.

- We propose a method to learn machine learning predictive models based on feature interactions efficiently (Chapter 4). The proposed method is a random feature map for the itemset kernel, which is a generalization of some kernels using feature interactions. We also provide some theoretical analyses on the proposed method. Furthermore, we propose some faster and more memory efficient methods. The proposed methods enable to learn models using feature interactions more efficiently

when both the number of observed (training) instances and the number of features are large.

Organization. This dissertation is organized as follows. Chapter 2 introduces some basic notations/definitions and presents some basic existing methods. Chapter 3 presents models based on higher-order feature interactions across objects for feature-based link prediction. We propose random feature maps for kernel functions using feature interactions in Chapter 4, which enable us to learn predictive models based on feature interactions efficiently. We conclude our dissertation in Chapter 5.

Chapter 2

Preliminaries

2.1 Notation

We use $[M : N]$ to denote the set $\{M, M+1, \ldots, N-1, N\}$ and use $[N]$ when $M = 1$. We use $\circ$ for the element-wise product (a.k.a Hadamard product) of vectors, matrices, and tensors. We denote the ℓ_p norm for a vector and matrix as $\|\cdot\|_p$. Given a matrix $\boldsymbol{X}$, we use $\boldsymbol{x}_i$ for the i-th row vector and $\boldsymbol{x}_{:,i}$ for the i-th column vector. Given a matrix $\boldsymbol{X} \in \mathbb{R}^{n \times m}$, we denote the ℓ_q norm of the vector $(\|\boldsymbol{x}_1\|_p, \ldots, \|\boldsymbol{x}_n\|_p)^\top$ by $\|\boldsymbol{X}\|_{p,q} \coloneqq \left(\sum_{i=1}^n \|\boldsymbol{x}_i\|_p^q\right)^{1/q}$ and call it $\ell_{p,q}$ norm. We use the terms $\tilde{\ell}_{p,q}$ norm for $\ell_{p,q}$ norm for the transpose matrix, i.e., $\left\|\boldsymbol{P}^\top\right\|_{p,q}$, respectively. For the number of non-zero elements in a vector and matrix, we use $\mathrm{nnz}(\cdot)$. We define $\mathrm{supp}(\boldsymbol{x})$ as the indices of non-zero elements in $\boldsymbol{x} \in \mathbb{R}^d$: $\{i \in [d] : x_i \neq 0\}$. We use $\mathbf{I}_d$ for the (d, d) identity matrix. We define $\mathrm{abs}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^n \mapsto (|x_1|, \ldots, |x_n|)^\top$. Given $\boldsymbol{x} \in \mathbb{R}^d$, we use $\boldsymbol{x}_{\neg j}$ to denote the $d-1$ dimensional vector with x_j removed. We use $\boldsymbol{x}_{i:j}$ to denote the subvector of $\boldsymbol{x}$ that consists from the x_i to x_j : $\boldsymbol{x}_{i:j} = (x_i, x_{i+1}, \ldots, x_j)^\top$. We use $\langle \cdot, \cdot \rangle$ to denote the standard inner (dot) product for the vectors, matrices, and tensors. Given $\boldsymbol{a} \in \mathbb{R}^{d_1}$ and $\boldsymbol{b} \in \mathbb{R}^{d_2}$, we use $\boldsymbol{a} \otimes \boldsymbol{b} \in \mathbb{R}^{d_1 \times d_2}$, where $(\boldsymbol{a} \otimes \boldsymbol{b})_{i,j} = a_i b_j$, to denote the tensor (outer) product of $\boldsymbol{a}$ and $\boldsymbol{b}$. We use $(\boldsymbol{a}; \boldsymbol{b})$ to denote the concatenation of $\boldsymbol{a}$ and $\boldsymbol{b}$: $(\boldsymbol{a}; \boldsymbol{b}) = (a_1, \ldots, a_{d_1}, b_1, \ldots, b_{d_2})^\top \in \mathbb{R}^{d_1 + d_2}$. We use $\boldsymbol{e}_j^d \in \{0, 1\}^d$ for the d-dimensional standard basis vector whose j-th element is one and the others are zero. If a function f parameterized by Θ, we denote it as $f(\cdot; \Theta)$. However, for simplicity, we sometimes denote it as $f(\cdot)$.

2.2 Problem Setting

In this doctoral thesis, we mainly consider machine learning models and algorithms for a *supervised learning problem*. Given an input domain $\mathcal{X}$ and output domain $\mathcal{Y}$, our goal is to obtain an *accurate* function $f^* : \mathcal{X} \to \mathcal{Y}$ minimizing the (expected or true) risk

$$R(f) = \mathbb{E}_P[\ell^*(f(\boldsymbol{x}), y)], \tag{2.1}$$

where P is a joint probability distribution over $\mathcal{X} \times \mathcal{Y}$ that generates input-output pairs and $\ell^* : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ is a loss function that measures how wrong two arguments are, i.e., $\ell^*(f(\boldsymbol{x}), y)$ measures the wrongness of the prediction $f(\boldsymbol{x})$ w.r.t the true output y. For example, $\ell^*(y_1, y_2) = \frac{1}{2}(y_1 - y_2)^2$ is called squared loss (typically $\mathcal{Y} = \mathbb{R}$), $\ell^*(y_1, y_2) = |y_1 - y_2|$ is called absolute loss (typically $\mathcal{Y} = \mathbb{R}$), and $\ell^*(y_1, y_2) = 0$ if $y_1 = y_2$ otherwise

1 is called 0-1 loss ($\mathcal{Y} = \{0, 1\}$ or $\{-1, 1\}$). We call a function $f : \mathcal{X} \to \mathcal{Y}$ a *predictive model*. Unfortunately, we do not know the true distribution P in general and thus it is impossible to learn the optimal predictive model f^*. In supervised learning, we assume that there is a training dataset $\mathcal{D} = \{(\boldsymbol{x}_n, y_n) : \boldsymbol{x}_n \in \mathcal{X}, y_n \in \mathcal{Y}, n \in [N]\} \in (\mathcal{X} \times \mathcal{Y})^N$, where training instances are independently and identically distributed to the P. Then, we obtain an accurate function $\hat{f} : \mathcal{X} \to \mathbb{R}$ on P by

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{n=1}^{N} \ell(f(\boldsymbol{x}_n), y_n) + \Omega(f), \tag{2.2}$$

where $\mathcal{F}$ is a subset of all functions from $\mathcal{X}$ to $\mathbb{R}$, $\ell : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ is a *surrogate* loss function, and $\Omega : \mathcal{F} \to \mathbb{R}_{\geq 0}$. $\mathcal{F}$ is a subset of all predictive models and represents our (machine learning users') assumption or bias to predictive models. Ω measures how complex a predictive model is. We call $\mathcal{F}$ and Ω hypothesis sets and regularization term (or regularizer), respectively. The first reason why $\mathcal{F}$ and Ω are introduced is to avoid *overfitting*. Overfitting is a phenomenon such that a learned predictive model fits the training datasets $\mathcal{D}$ well but its performance on (unknown) P is poor. Our true goal is just to learn a predictive function that minimizes (2.1), not to learn one that fits only the training dataset. The second one is to make the optimization problem easily solvable. ℓ is introduced to make the optimization problem easily solvable too.

Hereinafter, we assume that $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}$ or $\{0, 1\}$ (or $\{1, -1\}$), and ℓ is convex and μ-smooth. The followings are examples of such loss functions.

- Squared loss: $\ell(y', y) = \frac{1}{2}(y' - y)^2$.
- Logistic loss: $\ell(y', y) = \log(1 + \exp(-y'y))$.
- Squared hinge loss: $\ell(y', y) = \max(0, 1 - y'y)^2$.

We call $\boldsymbol{x} \in \mathcal{X}$ a feature vector and each element a feature.

2.3 Linear Model

Linear models are one of the simplest and most classical predictive models used in statistics and machine learning. Linear models predict the output of $\boldsymbol{x}$ as

$$f_{\mathrm{LM}}(\boldsymbol{x}; \boldsymbol{w}, b) \coloneqq \langle \boldsymbol{x}, \boldsymbol{w} \rangle + b, \tag{2.3}$$

where $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are learnable parameters, i.e., $\mathcal{F} = \{f(\cdot) = \langle \cdot, \boldsymbol{w} \rangle + b : \boldsymbol{w} \in \mathbb{R}^d, w \in \mathbb{R}\}$. For simplicity, we omit the intercept b thereafter. The optimization problem of linear models is

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{n=1}^{N} \ell(f_{\mathrm{LM}}(\boldsymbol{x}_n; \boldsymbol{w}), y_n) + \Omega(\boldsymbol{w}). \tag{2.4}$$

The advantages of the linear model are as follows.

1. **Fast to evaluate and train**. Linear models can be evaluated in $O(\mathrm{nnz}\,(\boldsymbol{x}))$ time. Moreover, when $\ell(\cdot, \cdot)$ is convex w.r.t first argument and Ω is convex, (2.4) is convex optimization problem and can be solved efficiently by using a gradient descent (GD), coordinate descent (CD), or stochastic gradient descent (SGD) method. In addition, when ℓ is the squared loss and Ω is ℓ_2^2 norm, then the optimal solution to (2.4) can be computed analytically.

2. **Easy to interpret**. The magnitude of j-th element in $\boldsymbol{w}$, $|w_j|$, can be interpreted as *importance* of j-th feature.

A popular example of Ω is ℓ_2^2 norm $\Omega(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|_2^2$, where $\lambda > 0$ is a regularization-strength hyperparameter. It is convex and differentiable, and makes the optimization problem λ-strongly convex. Another popular example is ℓ_1 norm $\Omega(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|_1$. It induces *sparsity* on $\boldsymbol{w}$: the optimal solution $\boldsymbol{w}^*$ tends to have elements of zero. $w_j^* = 0$ implies j-th feature is not used in the linear model $f_{\mathrm{LM}}(\cdot, \boldsymbol{w}^*)$, and therefore it is used for *feature selection*.

Unfortunately, the relationship between features and outputs are not necessarily linear in real-world problems. In real-world applications, non-linear models often perform better than linear models.

2.4 Polynomial Regression and Linear Basis Function Models

The *Polynomial regression* model (PR) is an extension of linear models. The output of the M-order PR is defined as

$$f_{\mathrm{PR}}^M(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{W}^{(2)}, \dots, \boldsymbol{W}^{(M)}) \coloneqq \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{m=2}^{M} \sum_{j_1 < \cdots < j_m} w_{j_1, \dots, j_m}^{(m)} x_{j_1} \cdots x_{j_m}, \tag{2.5}$$

where $\boldsymbol{w} \in \mathbb{R}^d$ and $\boldsymbol{W}^{(m)} \in \mathbb{R}^{\overbrace{d \times \cdots \times d}^{m}}$ for all $m \in [2, M]$ are learnable parameters. We especially call the 2-order PR the quadratic regression (QR) and denote it as f_{QR}:

$$f_{\mathrm{QR}}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{W}) \coloneqq \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{j_2 > j_1} x_{j_1} x_{j_2} w_{j_1, j_2}. \tag{2.6}$$

In addition, we call $\boldsymbol{W}$ feature interaction matrix.

The PR is essentially a linear model. Let $\phi_{\mathrm{poly}}^m : \mathbb{R}^d \to \mathbb{R}^{\binom{d}{m}}$ be

$$\phi_{\mathrm{poly}}^m(\boldsymbol{x}) \coloneqq (x_1 x_2 \cdots x_{m-1} x_m, x_1 x_2 \cdots x_{m-1} x_{m+1}, \dots, x_{d+1-m} x_{d+2-m} \cdots x_d)^\top. \tag{2.7}$$

Then, the M-order PR is equivalent to the linear model with $(\boldsymbol{x}; \phi_{\mathrm{poly}}^2(\boldsymbol{x}); \cdots; \phi_{\mathrm{poly}}^M(\boldsymbol{x}))^\top$ as a feature vector. Therefore, the PR is optimized in the same way as linear models. Generally, we call a map $\phi : \mathcal{X} \to \mathbb{R}^D$ a *basis function* (or *feature map*) and linear models with a basis function, $f_{\mathrm{LM}}(\phi(\cdot))$, *linear basis function models*. Because the PR is essentially equivalent to the linear model with (2.7), its objective function is

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{n=1}^{N} \ell(f_{\mathrm{PR}}^M(\boldsymbol{x}_n), y_n) + \Omega(\boldsymbol{w}, \boldsymbol{W}^{(2)}, \dots, \boldsymbol{W}^{(M)}), \tag{2.8}$$

and that of the linear basis function model with $\phi : \mathcal{X} \to \mathbb{R}^D$ is obviously

$$\min_{\boldsymbol{w} \in \mathbb{R}^D} \frac{1}{N} \sum_{n=1}^{N} \ell(f_{\mathrm{LM}}(\phi(\boldsymbol{x}_n)), y_n) + \Omega(\boldsymbol{w}), \tag{2.9}$$

Clearly, the PR uses feature interactions (and of course the QR). The PR is easy to interpret and it can capture a non-linear relationship between inputs and outputs due to feature interactions. However, it requires $O\left(\mathrm{nnz}\left(\boldsymbol{x}\right)^M\right)$ computational cost for evaluation. Moreover, weights for unobserved interactions from the training dataset are learned as 0. Therefore, the PR can be used only when a dataset is sparse from the point of view of computational cost but it cannot estimate parameters well when a dataset is sparse.

2.5 Kernel Methods

The *kernel regression* model (KR) is defined by

$$f_K(\boldsymbol{x}; \boldsymbol{\alpha}, \{\boldsymbol{x}_n\}) \coloneqq \sum_{n=1}^{N} \alpha_n K(\boldsymbol{x}, \boldsymbol{x}_n), \tag{2.10}$$

where $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function called *kernel function* and $\boldsymbol{\alpha} \in \mathbb{R}^N$ is a learnable parameter. The objective function of the KR is

$$\frac{1}{N} \sum_{n=1}^{N} \ell(f_K(\boldsymbol{x}; \boldsymbol{\alpha}, \{\boldsymbol{x}_m\}), y_n) + \Omega_K(\boldsymbol{\alpha}; \{\boldsymbol{x}_m\}). \tag{2.11}$$

The KR with positive (semi) definite kernels (defined in bellow) can be regarded as a *dual* of linear basis function models. Let $\phi : \mathcal{X} \to \mathbb{R}^D$ and $K(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle$. Then, (2.10) can be written as

$$f_K(\boldsymbol{x}; \boldsymbol{\alpha}, \{\boldsymbol{x}_n\}) = \left\langle \sum_{n=1}^{N} \alpha_n \phi(\boldsymbol{x}_n), \phi(\boldsymbol{x}) \right\rangle. \tag{2.12}$$

Namely, the KR with $K(\boldsymbol{x}, \boldsymbol{y}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle$ is equivalent to linear basis function models with $\boldsymbol{w} \in \mathrm{span}\{\phi(\boldsymbol{x}_n) : n \in [N]\}$. Here, we consider the orthonormal decomposition of $\mathbb{R}^D$: for all $\boldsymbol{w}^* \in \mathbb{R}^D$, there exists $\boldsymbol{w} \in \mathrm{span}\{\phi(\boldsymbol{x}_n) : n \in [N]\}$ and $\boldsymbol{w} \in \mathbb{R}^D$ such that $\langle \boldsymbol{w}, \boldsymbol{w}' \rangle = 0$ and $\boldsymbol{w}^* = \boldsymbol{w} + \boldsymbol{w}'$. By construction, $\langle \boldsymbol{w}^*, \phi(\boldsymbol{x}_n) \rangle = \langle \boldsymbol{w}, \phi(\boldsymbol{x}_n) \rangle$. Therefore, the optimal solution (2.9) can be written as (2.12) if $\Omega(\boldsymbol{w} + \boldsymbol{w}') \geq \Omega(\boldsymbol{w})$ for all $\boldsymbol{w} \in \mathrm{span}\{\phi(\boldsymbol{x}_n) : n \in [N]\}$ and $\boldsymbol{w}' \in \mathbb{R}^D$ such that $\langle \boldsymbol{w}, \boldsymbol{w}' \rangle = 0$. Note that ℓ_2^2 satisfies the condition $\Omega(\boldsymbol{w} + \boldsymbol{w}') \geq \Omega(\boldsymbol{w})$ for all $\boldsymbol{w} \in \mathrm{span}\{\phi(\boldsymbol{x}_n) : n \in [N]\}$ and $\boldsymbol{w}' \in \mathbb{R}^D$ such that $\langle \boldsymbol{w}, \boldsymbol{w}' \rangle = 0$.

For more precise discussion, we present some technical terms and some well-known important results [17].

Definition 2.1 (Reproducing kernel Hilbert space)**.** A Hilbert space (complete inner product space) $\mathcal{H}$ of functions $f : \mathcal{X} \to \mathbb{R}$ is called *reproducing kernel Hilbert space* (RKHS) if $\delta_{\boldsymbol{x}} : \mathcal{H} \to \mathbb{R}$, $\delta_{\boldsymbol{x}}(f) = f(\boldsymbol{x})$ is continuous for all $\boldsymbol{x} \in \mathcal{X}$.

Definition 2.2 (Positive definite functions)**.** A symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is *positive (semi) definite* if for all

$$\sum_{n_1=1}^{N} \sum_{n_2=1}^{N} \alpha_{n_1} \alpha_{n_2} K(\boldsymbol{x}_{n_1}, \boldsymbol{x}_{n_2}) \tag{2.13}$$

for all $\alpha_n \in \mathbb{R}$, $\boldsymbol{x}_n \in \mathcal{X}$, $n \in [N]$, $N \geq 1$.

Definition 2.3 (Reproducing kernel)**.** Let $\mathcal{H}$ be a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$ and $\langle \cdot, \cdot \rangle_{\mathcal{H}} : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$ is its inner product. A function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a reproducing kernel of $\mathcal{H}$ if

$$K(\cdot, \boldsymbol{x}) \in \mathcal{H} \ \forall \boldsymbol{x} \in \mathcal{X}, \tag{2.14}$$

$$\langle f, K(\cdot, \boldsymbol{x}) \rangle = f(\boldsymbol{x}) \ \forall \boldsymbol{x} \in \mathcal{X}, \ \forall f \in \mathcal{H}. \tag{2.15}$$

Lemma 2.1. *Let $\mathcal{H}$ be any Hilbert space and $\phi : \mathcal{X} \to \mathcal{H}$. Then, $H(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \mapsto \langle \phi(\cdot), \phi(\cdot) \rangle_{\mathcal{H}}$ is positive (semi) definite.*

Theorem 2.2. *For a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$, $\mathcal{H}$, if the reproducing kernel exists, then it is unique. Moreover, if $\mathcal{H}$ is an RKHS if and only if it has a reproducing kernel.*

Theorem 2.3 (Moore-Aronszajn)**.** *Let $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a positive (semi) definite function. Then, there exists a unique RKHS $\mathcal{H}$ with reproducing kernel K.*

Thus, for a given basis function ϕ, there exists an RKHS $\mathcal{H}$ with reproducing kernel $\langle \phi(\cdot), \phi(\cdot) \rangle_{\mathcal{H}}$. Moreover, for a positive (semi) definite kernel K, there exists an RKHS with reproducing kernel K and $K(\boldsymbol{x}, \cdot)$ can be regarded as a feature map of $\boldsymbol{x}$. Hence, designing a feature map ϕ is equivalent to designing a positive (semi) definite function K. In the next subsection, we introduce some kernel function based on feature interactions.

The optimization problem of the KR corresponding to (2.9) with $\Omega(\cdot) = \lambda \|\cdot\|_2^2$ is

$$\min_{\boldsymbol{\alpha} \mathbb{R}^N} \frac{1}{N} \sum_{n=1}^{N} \ell(f_K(\boldsymbol{x}; \boldsymbol{\alpha}, \{\boldsymbol{x}_m\}), y_n) + \lambda \sum_{n_1=1}^{N} \sum_{n_2=1}^{N} \alpha_{n_1} \alpha_{n_2} K(\boldsymbol{x}_{n_1}, \boldsymbol{x}_{n_2}), \tag{2.16}$$

and it is convex optimization problem. Therefore, the KR requires $O(N^2 T)$ and $O(NT)$ time complexity for train and evaluation, respectively, where T is the computational cost for evaluating a kernel function. Even if D is too large, a kernel function can be evaluated efficiently (e.g., $O(d)$). Therefore, the advantage of the KR is what its computational cost is independent of D and its disadvantage is scalability w.r.t N.

2.5.1 Kernels Using Feature Interactions

The polynomial kernel is one of the most well-known kernel function. The m-order polynomial kernel is defined by

$$K_{\text{poly}}^m(\boldsymbol{x}, \boldsymbol{y}; c) \coloneqq (\langle \boldsymbol{x}, \boldsymbol{y} \rangle + c)^m, \tag{2.17}$$

where $c > 0$ is a hyperparameter. The m-order polynomial kernel (we assume $c = 0$ for simplicity) can be written as the inner product of two vectors with the following feature map:

$$\phi : \mathbb{R}^d \mapsto \left(x_1^m, \sqrt{m} x_1^{m-1} x_2, \ldots, \sqrt{m(m-1)} x_1^{m-2} x_2 x_3, \ldots, x_d^m \right)^\top \in \mathbb{R}^{(d+m-1)!/((d-1)!m!)}, \tag{2.18}$$

so polynomial kernels use feature interactions.

The ANOVA kernel [9, 64, 10] is similar to the polynomial kernel. The definition of an m-order ANOVA kernel between $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ is

$$K_{\mathrm{A}}^m(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \sum_{j_1 < \cdots < j_m}^{d} x_{j_1} \cdots x_{j_m} y_{j_1} \cdots y_{j_m}, \tag{2.19}$$

where $2 \leq m \leq d \in \mathbb{N}$ is the order of the ANOVA kernel. For convenience, 0/1-order ANOVA kernels are often defined as $K_{\mathrm{A}}^0(\boldsymbol{x}, \boldsymbol{y}) = 1$ and $K_{\mathrm{A}}^1(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$. The difference between the ANOVA kernel and the polynomial kernel is that the ANOVA kernel does not use feature interactions that include the same feature (e.g., $x_1 x_1$, $x_2^2 x_3$) while the polynomial kernel does. Although the evaluation of the m-order ANOVA kernel involves $O(d^m)$ terms, it can be computed in $O(dm)$ time using dynamic programming [9, 64]. Precisely, from the following well-known recursion of the ANOVA kernel [64, 9, 10], $K_{\mathrm{A}}^t(\boldsymbol{p}, \boldsymbol{x})$ and the FM and HOFM are clearly multi-linear w.r.t $p_1, \ldots, p_d, x_1, \ldots, x_d$:

$$K_{\mathrm{A}}^m(\boldsymbol{p}, \boldsymbol{x}) = K_{\mathrm{A}}^m(\boldsymbol{p}_{\neg j}, \boldsymbol{x}_{\neg j}) + p_j x_j K_{\mathrm{A}}^{m-1}(\boldsymbol{p}_{\neg j}, \boldsymbol{x}_{\neg j}). \tag{2.20}$$

In some applications, ANOVA-kernel-based models have achieved better performance than polynomial-kernel-based models [9, 10]. We discuss these models later in this section.

While the ANOVA kernel uses only m-order different feature interactions, the all-subsets kernel [9] K_{all} uses all different feature interactions and is defined as

$$K_{\mathrm{all}}(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \prod_{j=1}^{d} (1 + x_j y_j). \tag{2.21}$$

Clearly, evaluation of the all-subsets kernel takes only $O(d)$ time.

For a given family of itemsets $\mathcal{S} \subseteq 2^{[d]}$, the itemset kernel [29] on $\mathcal{S}$ is defined as

$$K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \sum_{V \in \mathcal{S}} \prod_{j \in V} x_j y_j = \langle \phi_{\mathcal{S}}(\boldsymbol{x}), \phi_{\mathcal{S}}(\boldsymbol{y}) \rangle . \tag{2.22}$$

The itemset kernel clearly uses feature interactions in the family of itemsets $\mathcal{S}$ and can be regarded as an extension of the ANOVA kernel, all-subsets kernel, and standard dot product. For example, when $\mathcal{S} = 2^{[d]}$, $K_{2^{[d]}}$ clearly uses all feature interactions and hence is equivalent to the all-subsets kernel K_{all} in (2.21). When $\mathcal{S} = \binom{[d]}{m} \coloneqq \{V \subseteq [d] \mid |S| = m\}$, the itemset kernel $K_{\mathcal{S}}$ is equivalent to m-order ANOVA kernel K_{A}^m. Furthermore, when $\mathcal{S} = \{\{1\}, \ldots, \{d\}\}$, the itemset kernel $K_{\mathcal{S}}$ clearly represents the standard dot product.

2.6 Factorization Machines and Polynomial Networks

In this section, we introduce *factorization machines* (FMs) and related models.

2.6.1 Factorization Machines

FMs [60, 61] are models based on second-order feature interactions. FMs predict the target of $\boldsymbol{x}$ as

$$\begin{aligned} f_{\mathrm{FM}}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{P}) &\coloneqq \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{j_2 > j_1} \langle \boldsymbol{p}_{j_1}, \boldsymbol{p}_{j_2} \rangle \, x_{j_1} x_{j_2} \\ &= \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \frac{1}{2} \sum_{j_1=1}^{d} \sum_{j_2 \in [d] \setminus \{j_1\}} \langle \boldsymbol{p}_{j_1}, \boldsymbol{p}_{j_2} \rangle \, x_{j_1} x_{j_2}, \end{aligned} \tag{2.23}$$

where $\boldsymbol{w} \in \mathbb{R}^d$ and $\boldsymbol{P} \in \mathbb{R}^{d\times k}$ are learnable parameters, and $k \in \mathbb{N}_{>0}$ is the rank hyperparameter. The first term in (2.23) represents the linear relationship, and the second term represents the second-order polynomial relationship between the input and target. For a given training dataset $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, the objective function of the FM is

$$L_{\mathrm{FM}}(\boldsymbol{w}, \boldsymbol{P}; \lambda_w, \lambda_p) \coloneqq \frac{1}{N}\sum_{n=1}^{N} \ell(f_{\mathrm{FM}}(\boldsymbol{x}_n), y_n) + \lambda_w \left\|\boldsymbol{w}\right\|_2^2 + \lambda_p \left\|\boldsymbol{P}\right\|_2^2, \tag{2.24}$$

where $\lambda_w, \lambda_p \geq 0$ are the regularization-strength hyperparameters.

The inner product of the j_1-th and j_2-th row vectors in $\boldsymbol{P}$, $\langle \boldsymbol{p}_{j_1}, \boldsymbol{p}_{j_2} \rangle$, corresponds to the weight for the interaction between the j_1-th and j_2-th features in the FM. Thus, FMs are equivalent to the QR with factorization of the feature interaction matrix $\boldsymbol{W} = \boldsymbol{P}\boldsymbol{P}^\top$:

$$f_{\mathrm{QR}}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{P}\boldsymbol{P}^\top) = f_{\mathrm{FM}}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{P}). \tag{2.25}$$

The computational cost for evaluating FMs is $O(\mathrm{nnz}\,(\boldsymbol{x})\, k)$, i.e., it is linear w.r.t the dimension of feature vector d, because the second term in (2.23) can be rewritten as

$$\sum_{j_2 > j_1} \langle \boldsymbol{p}_{j_1}, \boldsymbol{p}_{j_2} \rangle\, x_{j_1} x_{j_2} = \sum_{s=1}^{k} (\langle \boldsymbol{p}_{:,s}, \boldsymbol{x} \rangle^2 - \langle \boldsymbol{p}_{:,s} \circ \boldsymbol{p}_{:,s}, \boldsymbol{x} \circ \boldsymbol{x} \rangle)/2. \tag{2.26}$$

On the other hand, the QR clearly requires $O\left(\mathrm{nnz}\,(\boldsymbol{x})^2\right)$ time and $O\left(d^2\right)$ space for storing $\boldsymbol{W}$, which is prohibitive for a high-dimensional case. Moreover, this factorized representation enables FMs to learn the weights for unobserved feature interactions but the QR (PR) does not learn such weights [60].

The objective function in (2.24) is differentiable, so Rendle [60] developed the SGD algorithm for minimizing (2.24). Although the objective function is non-convex w.r.t $\boldsymbol{P}$, it is multi-convex w.r.t $\boldsymbol{p_j}$ for all $j \in [d]$. It can thus be efficiently minimized by using the CD algorithm [62, 10]. Both the SGD and CD algorithms require $O(\mathrm{nnz}\,(\boldsymbol{X})\, k)$ time per epoch (using all instances at one time in the SGD algorithm and updating all parameters at one time in the CD algorithm), where $\boldsymbol{X} \in \mathbb{R}^{N\times d}$ is the design matrix. It is linear w.r.t both the number of training examples N and the dimension of feature vector d.

2.6.2 Polynomial Networks and Convex Factorization Machines

Livni et al. [41] proposed polynomial networks (PNs), which are depth-2 neural networks with a polynomial as the activation function:

$$f_{\mathrm{PN}}^2(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{P}, \boldsymbol{\alpha}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{s=1}^{k} \alpha_s \langle \boldsymbol{p}_{:,s}, \boldsymbol{x} \rangle^2 = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{s=1}^{k} \alpha_s K_{\mathrm{poly}}^2(\boldsymbol{x}, \boldsymbol{p}_{:,s}; 0), \tag{2.27}$$

where $\boldsymbol{w} \in \mathbb{R}^d$, $\boldsymbol{P} \in \mathbb{R}^{d\times k}$, and $\boldsymbol{\alpha} \in \mathbb{R}^k$ are the learnable parameters, and $k \in \mathbb{N}_{>0}$ is the number of hidden units (hyperparameter). $\boldsymbol{P}$ is the weight matrix between the input and hidden layers, and $\boldsymbol{\lambda}$ is the weight vector between the hidden layer and the output layer. They showed that PNs can approximate neural networks with a sigmoidal activation function.

In addition, they also provided an efficient convergence-guaranteed greedy learning algorithm based on solving an eigenvalue problem. At each iteration, their algorithm adds

a new hidden unit and learns only its weight vector. Mathematically, at t-th iteration, their algorithm finds $\boldsymbol{p}_{:,t}$ that minimizes the following first order approximation:

$$\begin{aligned}
&\frac{1}{N}\sum_{n=1}^{N}\ell\left(f_{\mathrm{PN}}^{2}\left(\boldsymbol{x}_n;\boldsymbol{w},(\boldsymbol{P}^{(t)},\boldsymbol{p}_t),(\boldsymbol{\alpha}^{(t)},\alpha_t)\right),y_n\right)\\
&=\frac{1}{N}\sum_{n=1}^{N}\ell(f^{(t)}(\boldsymbol{x}_n)+\alpha_t\left\langle\boldsymbol{x}_n,\boldsymbol{p}_{:,t}\right\rangle^{2},y_n)\\
&\simeq\frac{1}{N}\sum_{n=1}^{N}\left[\ell\left(f^{(t)}(\boldsymbol{x}_n),y_n\right)+\alpha\left\langle\boldsymbol{x}_n,\boldsymbol{p}_{:,t}\right\rangle^{2}\ell'\left(f^{(t)}(\boldsymbol{x}_n),y_n\right)\right],
\end{aligned}\tag{2.28}$$

where $f^{(t)}(\boldsymbol{x}_n)=f_{\mathrm{PN}}^{2}(\boldsymbol{x};\boldsymbol{w},\boldsymbol{P}=(\boldsymbol{p}_{:,1},\ldots,\boldsymbol{p}_{:,t-1}),\boldsymbol{\alpha}=(\alpha_1,\ldots,\alpha_{t-1})^{\top})$. The first term in (2.28) is independent of $\boldsymbol{p}_{:,s}$ and we can assume that $\left\|\boldsymbol{p}_{:,t}\right\|_2=1$ with out loss of generality since $\alpha_t\left\langle\boldsymbol{x},\boldsymbol{p}_{:,t}\right\rangle^{2}=\alpha_t\left\|\boldsymbol{p}_{:,t}\right\|_2^{2}\left\langle\boldsymbol{x},\boldsymbol{p}_{:,t}/\left\|\boldsymbol{p}_{:,t}\right\|_2\right\rangle^{2}$. Therefore the optimization problem w.r.t $\boldsymbol{p}_{:,t}$ can be written as

$$\max_{\boldsymbol{p}_{:,s}\in\mathbb{R}^d,\left\|\boldsymbol{p}_{:,s}\right\|_2=1}\left|\boldsymbol{p}_{:,t}^{\top}\left[\sum_{n=1}^{N}\ell'\left(f^{(t)}(\boldsymbol{x}_n),y_n\right)\boldsymbol{x}_n\boldsymbol{x}_n^{\top}\right]\boldsymbol{p}_{:,t}\right|,\tag{2.29}$$

so the solution to (2.29) is the dominating eigenvector of $\sum_{n=1}^{N}\ell'\left(f^{(t)}(\boldsymbol{x}_n),y_n\right)\boldsymbol{x}_n\boldsymbol{x}_n$, and it can be computed efficiently, e.g., by using the power method. When $\boldsymbol{p}_{:,1},\ldots,\boldsymbol{p}_{:,t}$ are fixed, the optimization problem of $\boldsymbol{\alpha}$ (and $\boldsymbol{w}$) is equivalent to that of linear models. They also extended second-order PNs to third-order ones, which comprise a subset of depth-3 PNs. Blondel et al. [8], Yamada et al. [74] proposed convex factorization machines, it is almost the same as PNs.

Moreover, Blondel et al. [10] proposed a *lifted* formulation of PNs:

$$f_{\mathrm{PN}'}^{2}(\boldsymbol{x};\boldsymbol{w}\in\mathbb{R}^d,\boldsymbol{U},\boldsymbol{V}\in\mathbb{R}^{d\times k},\boldsymbol{\alpha}\in\mathbb{R}^k)=\left\langle\boldsymbol{w},\boldsymbol{x}\right\rangle\sum_{s=1}^{k}\alpha_s\left\langle\boldsymbol{u}_{:,s},\boldsymbol{x}\right\rangle\left\langle\boldsymbol{v}_{:,s},\boldsymbol{x}\right\rangle.\tag{2.30}$$

Lifted PNs are multi-linear w.r.t their parameters whereas original PNs (2.27) are not. Thus, lifted PNs can be optimized efficiently by using the CD algorithm.

2.6.3 Higher-order FMs and All-subsets Model

Blondel et al. [9] proposed higher-order FMs (HOFMs), which use not only second-order feature interactions but also higher-order feature interactions. M-order HOFMs predict the target of $\boldsymbol{x}$ as

$$f_{\mathrm{HOFM}}^{M}(\boldsymbol{x};\boldsymbol{w},\boldsymbol{P}^{(2)},\ldots,\boldsymbol{P}^{(M)})\coloneqq\left\langle\boldsymbol{x},\boldsymbol{w}\right\rangle+\sum_{m=2}^{M}\sum_{s=1}^{k}K_{\mathrm{A}}^{m}\left(\boldsymbol{x},\boldsymbol{p}_{:,s}^{(m)}\right),\tag{2.31}$$

where $\boldsymbol{P}^{(2)},\ldots,\boldsymbol{P}^{(M)}\in\mathbb{R}^{d\times k}$ are learnable parameters for $2,\ldots,M$-order feature interactions, respectively. M-order HOFMs clearly use from second to M-order feature interactions. Although the evaluation of HOFMs (2.31) seems to take $O\left(d^m k\right)$ time at first glance, it can be completed in $O\left(dkM^2\right)$ time since m-order ANOVA kernels can be evaluated in $O(dm)$ time by using dynamic programming [9, 64]. Blondel et al. [9] also proposed efficient CD and SGD-based algorithms.

Blondel et al. [9] also proposed the all-subsets model, which uses all feature interactions. The output of the all-subsets model is defined by

$$f_{\mathrm{all}}(\boldsymbol{x}; \boldsymbol{P}) \coloneqq \sum_{s=1}^{k} K_{\mathrm{all}}(\boldsymbol{x}, \boldsymbol{p}_{:,s}) = \sum_{m=0}^{d} \sum_{s=1}^{k} K_{\mathrm{A}}^{m}(\boldsymbol{p}_{:,s}, \boldsymbol{x}), \tag{2.32}$$

where $\boldsymbol{P} \in \mathbb{R}^{d \times k}$ is the learnable parameter. The all-subsets model uses all 2^d feature interactions. The all-subsets model is also multi-linear w.r.t its parameter, and therefore it is optimized by the CD algorithm efficiently. Practically, the all-subsets model tends to have lower performance than HOFMs [9].

2.6.4 Deep-Neural-Networks-based FMs

In the last decade, deep neural networks (DNNs) have achieved state-of-the-art performances in many tasks [22]. With these success, several researches have proposed DNN-based FMs [27, 58, 77, 24, 38, 70, 65, 14]. These researches introduced new layers that use feature interactions like FMs and proposed DNN-based models using them. He and Chua [27] proposed *neural factorization machines* (NFMs) that use second-order feature interactions in the *bi-interaction layer*, which is a hidden layer using second-order feature interactions. Each unit in the bi-interaction layer is a second-order ANOVA kernel and NFMs are DNNs using the bi-interaction layer as the first hidden layer. The main idea of other researches [58, 77, 24] are almost the same as that of He and Chua [27] and these DNN-based FMs achieved better performance than that of the original FMs.

Chapter 3

Algorithms for Feature-based Link Prediction Using Higher-order Feature Interactions across Objects

3.1 Introduction

Link prediction is the computational problem of determining whether two given objects are linked. In a common setting, an adjacency matrix with missing values for an objective network is given, and the task is predicting the missing values. In this chapter, we consider a feature-based link prediction problem in which the feature vectors of two objects are given. Feature-based link prediction is used in a more general setting because feature-based link prediction can be applied not only to the common adjacency-matrix-based (in other words, index-based) link prediction problem by regarding the indices of objects as features but also to many other identification-related problems: face verification by using two facial images, disambiguation of two author names in two different papers by using words in titles and names of co-authors, link prediction for a social network by using member features, and so on. While index-based link prediction methods cannot predict links between an unknown (completely new) object and unknown or known objects, feature-based link prediction methods can do it as long as the feature vectors are given.

Models using second-order feature interactions are effective for feature-based link prediction [51, 5, 72, 60, 43, 46, 61]. The higher-order factorization machine (HOFM) [9], which is an extension of the factorization machine (FM) [60, 61] that enables higher-order feature interactions, outperforms models using second-order feature interactions. It has thus been attracting the attention of many machine learning researchers. However, in feature-based link prediction, the HOFM uses higher-order feature interactions not only across the two objects but also from the same object. Feature interactions from the same object are irrelevant to major link prediction problems such as predicting identity (face verification, author name disambiguation, and so on) because it is not reasonable to determine whether two objects are the same from the features of only one object.

This chapter is based on [4], by the same authors, which appeared in IEICE Transactions on Information and Systems, Copyright(c) 2020 IEICE. The material in this paper was presented in part at the IEICE Transactions on Information and Systems [4], and all the figures of this chapter are reused from [4] under the permission of the IEICE.

As an efficient method for computing feature interactions only across two objects is not available, a model is needed that uses feature interactions only across two objects. We have developed models that use higher-order feature interactions only across two objects.

The contributions of this chapter are as follows:

- We derive an algorithm for efficiently computing the sum of higher-order feature interactions only across two objects.
- We present a model that uses feature interactions only across two objects and can be efficiently evaluated using the proposed algorithm.
- We present an efficient CD algorithm. Keys of our algorithm's efficiency make the CD algorithm for the HOFM faster than the original CD algorithm proposed by Blondel et al. [9].
- We also propose deep-neural-network-extensions of our proposed models.
- We describe and discuss the relationships among proposed models and existing models on the basis of representabilities of these models.

In Section 3.2, we explain existing methods [51, 5, 72, 43, 46, 60, 61, 9, 27, 58, 77, 24, 38, 41]. We present our contributions described above in Section 3.3. Experimental results are shown in Section 3.4 and finally we conclude in Section 3.5.

3.1.1 Problem Formulation and Notation

Feature-based link prediction is the computational problem of determining whether two objects, $\boldsymbol{a} \in \mathbb{R}^{d_1}$ and $\boldsymbol{b} \in \mathbb{R}^{d_2}$, are linked or not. Therefore, our goal is to obtain a classifier $f : \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \to \mathbb{R}$ such that $f(\boldsymbol{a}, \boldsymbol{b}) \geq 0$ if two objects are linked and $f(\boldsymbol{a}, \boldsymbol{b}) < 0$ otherwise. Supervised learning approaches have been used to obtain *accurate* classifier f. With these approaches, the user first collects a training set, i.e., a set of labeled pairs of objects $\mathcal{D} = \{(\boldsymbol{a}_1, \boldsymbol{b}_1, t_1), \ldots, (\boldsymbol{a}_N, \boldsymbol{b}_N, t_N)\}$, where $t_i \in \{-1, 1\}$ is the label of the i-th pair of objects, $t_i = 1$ means the two objects are linked, and $t_i = -1$ means they are not linked. The machine learning user next inputs $\mathcal{D}$ into the supervised learning algorithm to obtain classifier f.

3.2 Existing Methods Using Feature Interactions Across Objects

3.2.1 Feature Interactions Across Objects

As described above, classifier f is obtained by using a supervised learning algorithm after collecting a training set. However, the feature vectors of pairs are needed in order to use conventional supervised learning algorithms and models. Because only feature vectors for each object are given in our feature-based link prediction setting, design feature vectors of the pairs are required for common algorithms and models. However, designing appropriate feature vectors is a difficult problem in general. Fortunately, second-order feature interactions across objects work effectively [51, 5, 72, 43, 46].

The vector representing feature interactions across $\boldsymbol{a}$ and $\boldsymbol{b}$ can be written as the tensor product of $\boldsymbol{a}$ and $\boldsymbol{b}$:

$$\mathrm{vec}(\boldsymbol{b} \otimes \boldsymbol{a}) = (a_1 b_1, \ldots, a_1 b_{d_2}, \ldots, a_{d_1} b_{d_2})^\top. \tag{3.1}$$

It is possible to obtain f enabling second-order feature interactions across object by using this feature vector as the feature vector of a pair, but the computational cost of computing this vector is $O(d_1 d_2)$, which is too high.

3.2.2 Kernel Method

Oyama and Manning [51] and Ben-Hur and Noble [5] proposed the following kernel function for second-order feature interactions across objects:

$$\mathcal{P}^2((\boldsymbol{a}, \boldsymbol{b}), (\boldsymbol{a}', \boldsymbol{b}')) \coloneqq \sum_{i,j} a_i a'_i b_j b'_j. \tag{3.2}$$

We call this kernel a *pairwise kernel*. Clearly it enables second-order feature interactions across objects and can be computed in $O(d)$ time by $\langle \boldsymbol{a}, \boldsymbol{a}' \rangle \cdot \langle \boldsymbol{b}, \boldsymbol{b}' \rangle$. Hence, when the number of training data N is not so large, using a model (e.g., support vector machines (SVMs)) along with the pairwise kernel $\mathcal{P}^2$ enables the use of second-order feature interactions across objects without computing the feature vector of (3.1) efficiently.

Although using a second-order polynomial kernel for concatenating the feature vectors of two objects $\langle (\boldsymbol{a}; \boldsymbol{b}), (\boldsymbol{a}'; \boldsymbol{b}') \rangle^2$ enables the use of second-order feature interactions, then interactions from the same object, e.g., $a_1 a_2 a'_1 a'_2$, are also included. They are irrelevant for such applications as predicting identity because it is not reasonable to determine whether two objects are the same from the features of only one object. Indeed, SVMs with a pairwise kernel outperformed ones with polynomial kernels in author name disambiguation [51].

3.2.3 Matrix Factorization Method

The linear regression model with $\mathrm{vec}(\boldsymbol{a} \otimes \boldsymbol{b})$ can be written in matrix form:

$$f_{\mathrm{BM}}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{W}) \coloneqq \boldsymbol{a}^\top \boldsymbol{W} \boldsymbol{b} = \sum_{i=1}^{d_1} \sum_{j=1}^{d_2} w_{i,j} a_i b_j, \tag{3.3}$$

where $\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}$ is the learnable parameter. We call this a *bilinear model* (BM). The size of the BM and the computational cost of evaluating the BM are $O(d_1 d_2)$, and these may be prohibitive. One proposed solution is factorization of $\boldsymbol{W}$ [72, 43, 46]:

$$f_{\mathrm{FBM}}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{U}, \boldsymbol{V}) \coloneqq \boldsymbol{a}^\top \boldsymbol{U} \boldsymbol{V}^\top \boldsymbol{b} = f_{\mathrm{BM}}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{U} \boldsymbol{V}^\top), \tag{3.4}$$

where $\boldsymbol{U} \in \mathbb{R}^{d_1 \times k}$ and $\boldsymbol{V} \in \mathbb{R}^{d_2 \times k}$ are the learnable parameters and $k \in \mathbb{N}$ is the rank hyper-parameter. We call this the *factorized bilinear model* (FBM). The size of the FBM and the computational cost of evaluating the FBM are $O(k(d_1 + d_2))$, which are acceptable.

3.2.4 FMs and HOFMs for Link Prediction

HOFMs and FMs have been used in recommender systems in which the feature vectors of each user and item are given [60, 10] and in feature-based link prediction [9] by using $(\boldsymbol{a}; \boldsymbol{b})$ as $\boldsymbol{x}$. In this case, the FM not only uses feature interactions across objects but also uses different-feature interactions from the same object: $\{a_i a_j \mid i \neq j\} \cup \{b_i b_j \mid i \neq j\}$, so they are irrelevant to some problems like predicting identity.

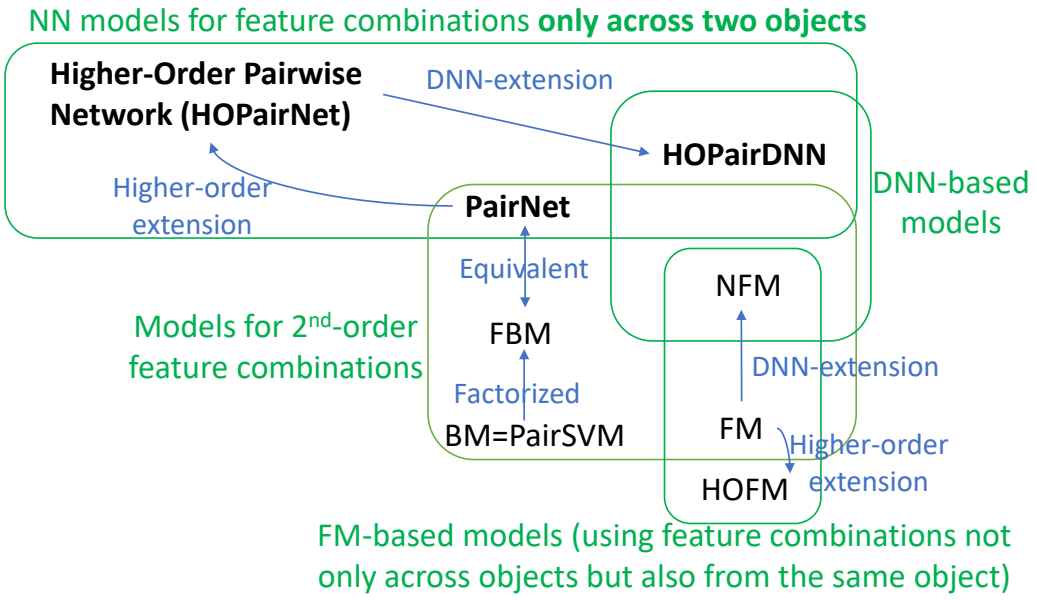


Figure 3.1: Relationships among proposed methods (HOPairNet, PairNet, HOPairDNN) and some existing methods.

3.3 Higher-Order Feature Interactions Across Two Objects

3.3.1 Basic Idea of Our Research

As mentioned above, the use of second-order feature interactions across objects is an effective approach in feature-based link prediction [51, 5, 72, 43, 46]. Although the HOFM [9] using higher-order feature interactions achieved better performance than the second-order FM [60, 61], the HOFM and FM use feature interactions from the same object, and these interactions are irrelevant to such problems as predicting identity. Therefore, models using higher-order feature interactions only across the two objects should outperform these models described above. Fig. 3.1 summarizes relationships among the proposed methods: *pairwise networks* (PairNets), *higher-order pairwise networks* (HOPairNets), and *higher-order pairwise deep neural networks* (HOPairDNNs) that will be presented later and some existing methods.

3.3.2 Higher-Order Pairwise Kernel

We define the higher-order feature interactions across $\boldsymbol{a} \in \mathbb{R}^{d_1}$ and $\boldsymbol{b} \in \mathbb{R}^{d_2}$ as

$$\bigcup_{t=1}^{m-1} \{a_{i_1} \cdots a_{i_t} b_{j_1} \cdots b_{j_{m-t}} \mid i_{t_1} < i_{t_2}, j_{t_1} < j_{t_2} \; \forall t_1 < t_2\},$$

that is, m-order feature interactions including at least one feature of both objects, and not including feature interactions from the same object. We also define a kernel using

Algorithm 1 DP algorithm for evaluating $K_{\mathrm{A}}^m(\boldsymbol{p}, \boldsymbol{x})$ in $O(md)$ time and $O(m)$ memory

Input: $\boldsymbol{p}, \boldsymbol{x} \in \mathbb{R}^d$
  $a_0 \leftarrow 1, a_t \leftarrow 0 \, \forall t \in [m]$;
  for $j \leftarrow 1, \ldots, d$ **do**
    for $t \leftarrow m, \ldots, 1$ **do**
      $a_t \leftarrow a_t + p_j x_j a_{t-1}$;
    end for
  end for
Output: $K_{\mathrm{A}}^m(\boldsymbol{p}, \boldsymbol{x}) = a_m$

higher-order feature interactions across objects:

$$\begin{aligned}&\mathcal{P}^m((\boldsymbol{u}, \boldsymbol{v}), (\boldsymbol{a}, \boldsymbol{b})) \\ &= \sum_{t=1}^{m-1} \sum_{i_1 < \cdots < i_t} \sum_{j_1 < \cdots < j_{m-t}} \left(\prod_{t'=1}^{t} u_{i_{t'}} a_{i_{t'}} \right) \left(\prod_{t'=1}^{m-t} b_{j_{t'}} v_{j_{t'}} \right), \end{aligned} \tag{3.5}$$

where $\boldsymbol{u} \in \mathbb{R}^{d_1}$ and $\boldsymbol{v} \in \mathbb{R}^{d_2}$. When $m = 2$, this equation is equivalent to the pairwise kernel in (3.2), so we call this kernel an m-order pairwise kernel.

Naïve computation of an m-order pairwise kernel takes $O\left(\sum_{t=1}^{m-1} d_1^t d_2^{m-t}\right)$ time, which may be prohibitive. However, a clue to efficiently computing a higher-order pairwise kernel can be obtained from the following transformation:

$$\mathcal{P}^m((\boldsymbol{u}, \boldsymbol{v}), (\boldsymbol{a}, \boldsymbol{b})) = \sum_{t=1}^{m-1} K_{\mathrm{A}}^t(\boldsymbol{u}, \boldsymbol{a}) K_{\mathrm{A}}^{m-t}(\boldsymbol{v}, \boldsymbol{b}). \tag{3.6}$$

This equation shows that an m-order pairwise kernel can be computed in $O(m)$ time when the series of the ANOVA kernels, $K_{\mathrm{A}}^t(\boldsymbol{u}, \boldsymbol{a})$ and $K_{\mathrm{A}}^t(\boldsymbol{v}, \boldsymbol{b})$ for $t \in [m-1]$, are given. The ANOVA kernels $K_{\mathrm{A}}^t(\boldsymbol{u}, \boldsymbol{a})$ and $K_{\mathrm{A}}^t(\boldsymbol{v}, \boldsymbol{b})$ for $t \in [m-1]$ are computed in $O(md_1)$ and $O(md_2)$ time and memory by previous algorithm [9]. Therefore, an m-order pairwise kernel can be computed in $O(m(d_1 + d_2))$ time and memory. Fortunately, the DP algorithm for evaluating the ANOVA kernel based on recursion (2.20) can actually be run in only $O(m)$ memory, and an m-order pairwise kernel can be computed in $O(m(d_1 + d_2))$ time and $O(m)$ memory. We show the procedure of this efficient algorithm for computing ANOVA kernels in Algorithm 1. The final value of a_t in Algorithm 1 is $K_{\mathrm{A}}^t(\boldsymbol{p}, \boldsymbol{x})$ for $t \in [m]$. Therefore, not only $K_{\mathrm{A}}^m(\boldsymbol{p}, \boldsymbol{x})$ but also $K_{\mathrm{A}}^t(\boldsymbol{p}, \boldsymbol{x})$ for $t \in [m-1]$ is obtained in $O(md)$ time and $O(m)$ memory. This insight for the memory efficiency of recursion (2.20) can be useful for the optimization and it will be described in Section 3.3.4.

There are two important properties of the higher-order pairwise kernel. The first one is multi-linearity. It is derived from the multi-linearity of the ANOVA kernels and (3.6). The second one is homogeneity. Let $\lambda \in \mathbb{R}$ and $m \in \mathbb{N}_{\geq 2}$. Then,

$$\lambda^m \mathcal{P}^m((\boldsymbol{u}, \boldsymbol{v}), (\boldsymbol{a}, \boldsymbol{b})) = \mathcal{P}^m((\lambda \boldsymbol{u}, \lambda \boldsymbol{v}), (\boldsymbol{a}, \boldsymbol{b})). \tag{3.7}$$

When $m = 2$, the following equation is also satisfied:

$$\lambda \mathcal{P}^2((\boldsymbol{u}, \boldsymbol{v}), (\boldsymbol{a}, \boldsymbol{b})) = \mathcal{P}^2((\lambda \boldsymbol{u}, \boldsymbol{v}), (\boldsymbol{a}, \boldsymbol{b})). \tag{3.8}$$

It is derived from the homogeneousity of the ANOVA kernel [10]: $\lambda^m K_{\mathrm{A}}^m(\boldsymbol{p}, \boldsymbol{x}) = K_{\mathrm{A}}^m(\lambda\boldsymbol{p}, \boldsymbol{x})$. From (3.6) and homogeneousity of the ANOVA kernel, (3.7) is obtained:

$$\begin{aligned}\lambda^m \mathcal{P}^m((\boldsymbol{u}, \boldsymbol{v}), (\boldsymbol{a}, \boldsymbol{b})) &= \sum_{t=1}^{m-1} \lambda^t K_{\mathrm{A}}^t(\boldsymbol{u}, \boldsymbol{a}) \lambda^{m-t} K_{\mathrm{A}}^{m-t}(\boldsymbol{v}, \boldsymbol{b}) \\ &= \sum_{t=1}^{m-1} K_{\mathrm{A}}^t(\lambda\boldsymbol{u}, \boldsymbol{a}) K_{\mathrm{A}}^{m-t}(\lambda\boldsymbol{v}, \boldsymbol{b}) = \mathcal{P}^m((\lambda\boldsymbol{u}, \lambda\boldsymbol{v}), (\boldsymbol{a}, \boldsymbol{b})). \end{aligned} \tag{3.9}$$

(3.8) is obtained in similar way: $\lambda\mathcal{P}^2((\boldsymbol{u}, \boldsymbol{v}), (\boldsymbol{a}, \boldsymbol{b})) = \lambda K_{\mathrm{A}}^1(\boldsymbol{u}, \boldsymbol{a}) K_{\mathrm{A}}^1(\boldsymbol{v}, \boldsymbol{b}) = K_{\mathrm{A}}^1(\lambda\boldsymbol{u}, \boldsymbol{a}) K_{\mathrm{A}}^1(\boldsymbol{v}, \boldsymbol{b}) = \mathcal{P}^2((\lambda\boldsymbol{u}, \boldsymbol{v}), (\boldsymbol{a}, \boldsymbol{b}))$. It is used to discuss the representability of our proposed models.

3.3.3 Higher-Order Pairwise Network and Pairwise Network

We first propose a *higher-order pairwise network* (HOPairNet) that uses higher-order feature interactions only across the two objects. It is based on the definition of the HOFM and PN [41, 10]. The model formula for this model is given by

$$\begin{aligned}&f_{\mathrm{HOPairNet}}^m(\boldsymbol{a}, \boldsymbol{b}; \mathcal{U}^{(m)}, \mathcal{V}^{(m)}, \Lambda^{(m)}) \\ &\coloneqq \sum_{t=2}^{m} \sum_{s=1}^{k} \lambda_s^{(t)} \mathcal{P}^t \left(\left(\boldsymbol{u}_{:,s}^{(t)}, \boldsymbol{v}_{:,s}^{(t)} \right), (\boldsymbol{a}, \boldsymbol{b}) \right), \end{aligned} \tag{3.10}$$

where $\boldsymbol{U}^{(2)}, \ldots, \boldsymbol{U}^{(m)} \in \mathbb{R}^{d_1 \times k}$, $\boldsymbol{V}^{(2)}, \ldots, \boldsymbol{V}^{(m)}, \boldsymbol{\lambda}^{(2)} \in \mathbb{R}^{d_2 \times k}$, and $\boldsymbol{\lambda}^{(2)}, \ldots, \boldsymbol{\lambda}^{(m)} \in \mathbb{R}^k$ are the learnable parameters, and $\mathcal{U}^{(m)}, \mathcal{V}^{(m)}$, and $\Lambda^{(m)}$ are sets of them. An m-order HOPairNet clearly enables the use of from second to m-order feature interactions only across the two objects. We call a second-order HOPairNet a *PairNet*:

$$f_{\mathrm{Pair}}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{\lambda}, \boldsymbol{U}, \boldsymbol{V}) \coloneqq \sum_{s=1}^{k} \lambda_s \mathcal{P}^2 \left((\boldsymbol{u}_{:,s}, \boldsymbol{v}_{:,s}), (\boldsymbol{a}, \boldsymbol{b}) \right). \tag{3.11}$$

The most important property of the HOPairNet and PairNet is multi-linearity w.r.t $\lambda_s^{(t)}$, $u_{j_1,s}^{(t)}$, and $v_{j_2,s}^{(t)}$. It is easily derived from the multi-linearity of the higher-order pairwise kernel. It makes the objective function of the HOPairNet model multi-convex, enabling it to be efficiently optimized by using the CD algorithm, as with the HOFM.

With regards to the representability of the PairNet, [10] showed that when m is odd, $\boldsymbol{\lambda} = 1$ can be fixed without loss of generality in the FM and PN. Similar results are obtained with a model using a higher-order pairwise kernel.

Lemma 3.1. *Let* $f_{\mathcal{P}^m}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{\lambda}, \boldsymbol{U}, \boldsymbol{V}) \coloneqq \sum_{s=1}^k \lambda_s \mathcal{P}^m((\boldsymbol{u}_{:,s}, \boldsymbol{v}_{:,s}), (\boldsymbol{a}, \boldsymbol{b}))$, $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_{\geq 0}$ *be a convex loss function, and* $L_{\mathcal{P}^m}(\mathcal{D}, \boldsymbol{\lambda}, \boldsymbol{U}, \boldsymbol{V}) \coloneqq \frac{1}{N} \sum_{i=1}^N \ell(f_{\mathcal{P}^m}(\boldsymbol{a}_i, \boldsymbol{b}_i), t_i)$. *Then, if* m *is even that is greater than 2,*

$$\min_{\boldsymbol{\lambda}, \boldsymbol{U}, \boldsymbol{V}} L_{\mathcal{P}^m}(\boldsymbol{\lambda}, \boldsymbol{U}, \boldsymbol{V}) \leq \min_{\boldsymbol{U}, \boldsymbol{V}} L_{\mathcal{P}^m}(\boldsymbol{1}, \boldsymbol{U}, \boldsymbol{V}), \tag{3.12}$$

and otherwise (that is, if m *is odd or 2)*

$$\min_{\boldsymbol{\lambda}, \boldsymbol{U}, \boldsymbol{V}} L_{\mathcal{P}^m}(\boldsymbol{\lambda}, \boldsymbol{U}, \boldsymbol{V}) = \min_{\boldsymbol{U}, \boldsymbol{V}} L_{\mathcal{P}^m}(\boldsymbol{1}, \boldsymbol{U}, \boldsymbol{V}). \tag{3.13}$$

Proof. With the use the homogeneity of the higher-order pairwise kernel, Lemma 3.1 is derived in the same way as the result for the PN and FM ([10], Lemma 4). □

Because $f_{\mathcal{P}^2}$ is equivalent to the PairNet, $\boldsymbol{\lambda} = 1$ can be fixed without loss of generality in the PairNet. Lemma 3.1 says that introducing $\boldsymbol{\lambda}^{(t)}$ for even $t \geq 4$ improves the representability of the HOPairNet.

We next show that the PairNet is equivalent to the FBM from Lemma 3.1 and from the result of transformation to the matrix form of (3.11).

Lemma 3.2. *Equivalence of the PariNet and the FBM.*

For every PairNet f_{Pair}, there exist an FBM f_{FBM} such that $f_{\mathrm{Pair}}(\boldsymbol{a}, \boldsymbol{b}) = f_{\mathrm{FBM}}(\boldsymbol{a}, \boldsymbol{b})$ for all $\boldsymbol{a} \in \mathbb{R}^{d_1}, \boldsymbol{b} \in \mathbb{R}^{d_2}$.

Proof. We first show the matrix form of the model equation of the PairNet, which is given by

$$f_{\mathrm{Pair}}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{\lambda}, \boldsymbol{U}, \boldsymbol{V}) = \boldsymbol{a}^\top \boldsymbol{U} \mathrm{diag}(\boldsymbol{\lambda}) \boldsymbol{V}^\top \boldsymbol{b}. \tag{3.14}$$

From Lemma 3.1, $\boldsymbol{\lambda} = 1$ can be fixed in the PairNet without loss of generality. When $\boldsymbol{\lambda} = 1$, the PairNet is equivalent to the FBM: $f_{\mathrm{Pair}}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{1}, \boldsymbol{U}, \boldsymbol{V}) = \boldsymbol{a}^\top \boldsymbol{U} \boldsymbol{V}^\top \boldsymbol{b} = f_{\mathrm{FBM}}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{U}, \boldsymbol{V})$. □

Therefore, the HOPairNet can be also regarded as an higher-order generalization of the FBM. Furthermore, the following result for regularization in the BM, the PairNet, and the FBM is derived from Lemma 3.2, (3.4), and previous results of regularization in the PN and FM ([10], Theorem 2).

Theorem 3.3. *Equivalence of regularized problems. Let $\|\cdot\|_*$ be the nuclear norm. Then,*

$$\min_{\boldsymbol{W}} \frac{1}{N} \sum_{i=1}^{N} \ell\left(t_i, f_{\mathrm{BM}}\right) + \beta \left\|\boldsymbol{W}\right\|_* \tag{3.15}$$

$$= \min_{\boldsymbol{\lambda}, \boldsymbol{U}, \boldsymbol{V}} \frac{1}{N} \sum_{i=1}^{N} \ell(f_{\mathrm{Pair}}, t_i) + \frac{\beta}{2} \sum_{s=1}^{k} |\lambda_s| \Omega(\boldsymbol{u}_{:,s}, \boldsymbol{v}_{:,s}), \tag{3.16}$$

$$= \min_{\boldsymbol{U}, \boldsymbol{V}} \frac{1}{N} \sum_{i=1}^{N} \ell(f_{\mathrm{FBM}}, t_i) + \frac{\beta}{2} \left(\|\boldsymbol{U}\|_2^2 + \|\boldsymbol{V}\|_2^2 \right), \tag{3.17}$$

where $\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}$, $\boldsymbol{U} \in \mathbb{R}^{d_1 \times k}$, $\boldsymbol{V} \in \mathbb{R}^{d_2 \times k}$, $\boldsymbol{\lambda} \in \mathbb{R}^k$, $\mathrm{rank}(\boldsymbol{W}^*) \leq k$, $\Omega(\boldsymbol{u}, \boldsymbol{v}) \coloneqq \|\boldsymbol{u}\|_2^2 + \|\boldsymbol{v}\|_2^2$, and $\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} \frac{1}{N} \sum_{i=1}^{N} \ell(f_{\mathrm{BM}}(\boldsymbol{a}_i, \boldsymbol{b}_i; \boldsymbol{W}), t_i) + \beta \|\boldsymbol{W}\|_*$ (and we omit the input vectors for each model).

Proof. These results can be obtained in the same way as in ([10], Theorem 2). Evaluation First, the value of loss term in (3.15) is equal to that in (3.17) when $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^\top$. Then, (3.15)=(3.17) is derived from the relationship between the nuclear norm and the Frobenius norm of the factorized matrix $\|\boldsymbol{W}\|_* = \min_{\boldsymbol{U}, \boldsymbol{V} \text{s.t.} \boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^\top} (\|\boldsymbol{U}\|_2^2 + \|\boldsymbol{V}\|_2^2)/2$. (3.17)=(3.16) is derived from the following transformation of (3.14). Let $\tilde{\boldsymbol{\lambda}} = \left(\sqrt{|\lambda_0|}, \ldots, \sqrt{|\lambda_k|}\right)^\top$ and $\boldsymbol{\lambda}_{\mathrm{sign}} = (\mathrm{sign}(\lambda_1), \ldots, \mathrm{sign}(\lambda_s))^\top$. Then,

$$\begin{aligned} & f_{\mathrm{Pair}}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{\lambda}, \boldsymbol{U}, \boldsymbol{V}) = \boldsymbol{a}^\top \boldsymbol{U} \mathrm{diag}(\boldsymbol{\lambda}) \boldsymbol{V}^\top \boldsymbol{b} \\ & = \boldsymbol{a}^\top \boldsymbol{U} \mathrm{diag}\left(\tilde{\boldsymbol{\lambda}}\right) \mathrm{diag}\left(\boldsymbol{\lambda}_{\mathrm{sign}}\right) \mathrm{diag}\left(\tilde{\boldsymbol{\lambda}}\right) \boldsymbol{V}^\top \boldsymbol{b}. \end{aligned} \tag{3.18}$$

Let $\tilde{\boldsymbol{U}} = \boldsymbol{U}\mathrm{diag}\left(\tilde{\boldsymbol{\lambda}}\right)\mathrm{diag}\left(\boldsymbol{\lambda}_{\mathrm{sign}}\right)$ and $\tilde{\boldsymbol{V}} = \mathrm{diag}\left(\tilde{\boldsymbol{\lambda}}\right)\boldsymbol{V}$. Then, substituting $\tilde{\boldsymbol{U}}$ and $\tilde{V}$ as $\boldsymbol{U}$ and $\boldsymbol{V}$ in (3.17) results (3.16). □

Nuclear norm regularization has been used for obtaining low-rank solutions [19, 30], so a low-rank solution is expected for problem (3.16) for the PairNet. Although we cannot derive a theoretical result for the HOPairNet (i.e., the higher-order case), we can use the straightforward extension of (3.16) as the objective function,

$$\frac{1}{N}\sum_{i=1}^{N}\ell(y_i, t_i) + \frac{\beta}{2}\sum_{t=2}^{m}\sum_{s=1}^{k}\left|\lambda_s^{(t)}\right|\Omega(\boldsymbol{u}_{:,s}^{(t)}, \boldsymbol{v}_{:,s}^{(t)}), \tag{3.19}$$

where $y_i = f_{\mathrm{HOPair}}^{m}(\boldsymbol{a}_i, \boldsymbol{b}_i)$, because problem (3.19) can be regarded as the least absolute shrinkage and selection operator (LASSO) [66] for $\boldsymbol{\lambda}^{(t)}$ for $t \in [2:m]$ when $\boldsymbol{U}^{(t)}$ and $\boldsymbol{V}^{(t)}$ are fixed. Therefore, sparse solutions for $\boldsymbol{\lambda}^{(t)}$ can be expected, and obtaining sparse solutions can be regarded as the selection of bases; that is, a low-rank solution can be expected. To be more precise, because $\boldsymbol{\lambda}^{(t)} = \boldsymbol{1}$ can be fixed when t is odd or 2 from Lemma 3.1, we only fit $\boldsymbol{\lambda}^{(t)}$, which has even t greater than 2. We use the CD algorithm with proximal operation [53] for optimizing such $\lambda^{(t)}$. It is easily done by caching $\Omega\left(\boldsymbol{u}_{:,s}^{(t)}, \boldsymbol{v}_{:,s}^{(t)}\right)$ and $\mathcal{P}^{(t)}\left(\left(\boldsymbol{u}_{:,s}^{(t)}, \boldsymbol{v}_{:,s}^{(t)}\right), (\boldsymbol{a}_i, \boldsymbol{b}_i)\right)$ for all $s \in [k]$ and $i \in [n]$.

3.3.4 CD Algorithm for HOPairNets

As described above, optimization problems (3.16) and (3.19) are multi-convex optimization problems. Hence, the two models proposed above can be efficiently optimized by using the CD algorithm. Here we describe its use for the HOPairNet that includes the PairNet (when $m = 2$). We assume that loss function l is convex and μ-smooth function. Similar to the CD algorithm for the HOFM [9], the update rule for $u_{j,s}^{(m)}$ is $u_{j,s}^{(m)} \leftarrow u_{j,s}^{(m)} - \eta_{j,s}^{-1}\partial L/\partial u_{j,s}^{(m)}$, where L is the objective function in (3.19) and $\eta_{j,s} = \mu\sum_{i=1}^{N}\left(\partial y_i/\partial u_{j,s}^{(m)}\right)^2/N + \beta|\lambda_s^{(t)}|$. The update rule for $v_{j,s}^{(t)}$ is easily derived in a similar manner. For updating, one obviously must compute the partial gradient

$$\frac{\partial y_i}{\partial u_{j,s}^{(m)}} = \lambda_s^{(m)}\sum_{t=1}^{m-1}\frac{\partial K_{\mathrm{A}}^{t}\left(\boldsymbol{u}_{:,s}^{(m)}, \boldsymbol{a}_i\right)}{\partial u_{j,s}^{(m)}}K_{\mathrm{A}}^{m-t}\left(\boldsymbol{v}_{:,s}^{(m)}, \boldsymbol{b}_i\right),$$

and it requires $\partial K_{\mathrm{A}}^{t}\left(\boldsymbol{u}_{:,s}^{(m)}, \boldsymbol{a}_i\right)/\partial u_{j,s}^{(m)}$ for $t \in [m-1]$. One can compute it by existing algorithm proposed by Blondel et al. [9]. Then, the computational cost for updating all coordinates of $\boldsymbol{U}^{(m)}$ and $\boldsymbol{V}^{(m)}$ once is $O\left(m^2k(n_z(\boldsymbol{A}) + n_z(\boldsymbol{B}))\right)$ time and $O(Nm)$ memory, where $\boldsymbol{A}$ and $\boldsymbol{B}$ are matrices in which the i-th row vector is $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$, respectively.

Here, we present a more efficient CD algorithm for the HOPairNet that takes only $O(mk(n_z(\boldsymbol{A}) + n_z(\boldsymbol{B})))$ time for updating all the coordinates of $\boldsymbol{U}^{(m)}$ and $\boldsymbol{V}^{(m)}$ once. It is based on the insight for the memory efficiency of recursion (2.20) (i.e., Algorithm 1) and the following new recursion for calculating the partial gradient of the ANOVA kernel:

$$\begin{aligned}\frac{\partial K_{\mathrm{A}}^{m}(\boldsymbol{p}, \boldsymbol{x})}{\partial p_j} &= x_j K_{\mathrm{A}}^{m-1}(\boldsymbol{p}_{\neg j}, \boldsymbol{x}_{\neg j}) \\ &= x_j\left(K_{\mathrm{A}}^{m-1}(\boldsymbol{p}, \boldsymbol{x}) - p_j\frac{\partial K_{\mathrm{A}}^{m-1}(\boldsymbol{p}, \boldsymbol{x})}{\partial p_j}\right).\end{aligned} \tag{3.20}$$

Algorithm 2 Algorithm for computing $\partial K_{\mathrm{A}}^m(\boldsymbol{p}, \boldsymbol{x})/\partial p_j$ in $O(m)$ time and memory

Input: $p_j, x_j, K_{\mathrm{A}}^t(\boldsymbol{p}, \boldsymbol{x}) \, \forall t \in [m-1]$
  $\tilde{p}_1 \leftarrow x_j$;
  for $t \leftarrow 2, \ldots, m$ **do**
    $\tilde{p}_t \leftarrow x_j \left(K_{\mathrm{A}}^{t-1}(\boldsymbol{p}, \boldsymbol{x}) - p_j \tilde{p}_{t-1} \right)$;
  end for
Output: $\frac{\partial K_{\mathrm{A}}^m(\boldsymbol{p}, \boldsymbol{x})}{\partial p_j} = \tilde{p}_m$

The advantage of recursion (3.20) is its reduced time complexity. When $K_{\mathrm{A}}^t(\boldsymbol{p}, \boldsymbol{x})$ for $t \in [m]$ are given, $\partial K_{\mathrm{A}}^t(\boldsymbol{p}, \boldsymbol{x})/\partial p_j$ for $t \in [m]$ can be computed in $O(m)$ time. The algorithm used for computing them using recursion (3.20) is shown in Algorithm 2. $K_{\mathrm{A}}^t(\boldsymbol{p}, \boldsymbol{x})$ for $t \in [m]$ can be computed in $O(md)$ time and $O(m)$ memory with Algorithm 1. With Algorithm 1 and Algorithm 2, $K_{\mathrm{A}}^t(\boldsymbol{p}, \boldsymbol{x})$ and $\partial K_{\mathrm{A}}^t(\boldsymbol{p}, \boldsymbol{x})/\partial p_j$ for $t \in [m]$ can be computed in $O(md)$ time and $O(m)$ memory while an algorithm proposed by Blondel et al. [9] takes $O\left(md + m^2\right)$ time.

For an efficient implementation of the CD algorithm, we need a method for efficiently synchronizing prediction. Fortunately, the prediction is also synchronized in $O(m)$ time. Let $p_j^{\mathrm{new}} = p_j - \Delta$ be the value after updating and $\boldsymbol{p}^{\mathrm{new}} = (p_1, \ldots, p_{j-1}, p_j^{\mathrm{new}}, p_{j+1}, \ldots, p_d)^\top$. Then,

$$K_{\mathrm{A}}^m\left(\boldsymbol{p}^{\mathrm{new}}, \boldsymbol{x}\right) = K_{\mathrm{A}}^m(\boldsymbol{p}, \boldsymbol{x}) - \Delta \frac{\partial K_{\mathrm{A}}^{m-1}(\boldsymbol{p}, \boldsymbol{x})}{\partial p_j}. \tag{3.21}$$

From these results about the complexities of the computing partial gradient and synchronizing ANOVA kernels and predictions, using proposed algorithms can reduce the computational cost of the CD algorithm for the HOPairNet from $O\left(m^2 k \left(n_z\left(\boldsymbol{A}\right) + n_z\left(\boldsymbol{B}\right)\right)\right)$ to $O\left(mk\left(n_z\left(\boldsymbol{A}\right) + n_z\left(\boldsymbol{B}\right)\right)\right)$ time. This improvement is easily applied to the CD algorithm for the HOFM.

3.3.5 Symmetrization

For some applications such as predicting identity, symmetry must be ensured; that is, $f(\boldsymbol{a}, \boldsymbol{b}) = f(\boldsymbol{b}, \boldsymbol{a})$ must be satisfied. In such applications, the domains of $\boldsymbol{a}$ and $\boldsymbol{b}$ are the same, so $d_1 = d_2 = d$. We discuss here the method for ensuring symmetry in our proposed models. We first consider the symmetry of the PairNet. The most straightforward way of ensuring symmetry is parameter sharing: $\boldsymbol{U} = \boldsymbol{V} = \boldsymbol{P}$. However, this parameter sharing does not preserve the multi-linearity of proposed models and thus we cannot optimize them efficiently.

Here, we use the relationship between the PairNet and the BM (Lemma 3.2 and (3.4)). The $f_{\mathrm{BM}}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{W})$ clearly satisfies the symmetry requirement when $\boldsymbol{W}$ is a symmetric matrix. Because the PairNet can be regarded as a BM with $\boldsymbol{W} = \boldsymbol{U}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^\top$, the model $f_{\mathrm{BM}}(\boldsymbol{a}, \boldsymbol{b}; (\boldsymbol{U}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^\top + \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{U}^\top)/2)$ can be regarded as the symmetric PairNet. This technique is called the *symmetrization trick* [10] and preserves the multi-linearity of the model. A method for ensuring HOPairNet symmetry can be easily derived from the

following transformation for the symmetric PairNet:

$$\begin{aligned}&\boldsymbol{a}^\top \frac{1}{2}\left(\boldsymbol{U}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^\top + \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{U}^\top\right)\boldsymbol{b}\\ &= \frac{1}{2}\left(f_{\mathrm{Pair}}(\boldsymbol{a},\boldsymbol{b};\boldsymbol{\lambda},\boldsymbol{U},\boldsymbol{V}) + f_{\mathrm{Pair}}(\boldsymbol{b},\boldsymbol{a};\boldsymbol{\lambda},\boldsymbol{U},\boldsymbol{V})\right).\end{aligned} \tag{3.22}$$

From this, a symmetrization method for the HOPairNet is derived: $\frac{1}{2}(f^m_{\mathrm{HOPair}}(\boldsymbol{a},\boldsymbol{b}) + f^m_{\mathrm{HOPair}}(\boldsymbol{b},\boldsymbol{a}))$.

3.3.6 Problem of DNNs in Symmetric Link Prediction

As described Chapter 2, DNNs are attracting attention in the field of machine learning. Since DNNs can automatically obtain feature representations, one might consider that DNNs with the concatenated vector $(\boldsymbol{a};\boldsymbol{b})$ should be able to outperform existing methods even in link prediction. However, concatenated vectors do not satisfy symmetry ($(\boldsymbol{a};\boldsymbol{b}) \neq (\boldsymbol{b};\boldsymbol{a})$). Therefore, for symmetric link prediction, a DNN with symmetry $f_{\mathrm{DNN}}((\boldsymbol{a};\boldsymbol{b})) = f_{\mathrm{DNN}}((\boldsymbol{b};\boldsymbol{a}))$ should be used. Bishop [7] classified the approaches to making a NN invariant into four approaches

1. The training set is augmented using replicas of the training patterns, transformed in accordance with the desired invariance.

2. A regularization term is added to the error function that penalizes changes in the model output when the input is transformed.

3. Invariance is built into the pre-processing by extracting features that are invariant under the required transformations.

4. Invariance is built into the structure of the NN.

In symmetric link prediction, the NN should be invariant with respect to the order of the data values in a pair; i.e., it should have symmetry. Approach 1, called data augmentation, incurs extra computational costs. Approach 2 cannot be used in pairwise classification because transformation that changes the data order is not continuous. Approach 3 is problematic because designing suitable features is difficult. Therefore, we took Approach 4. If the values of the hidden layer connected with the input layer satisfy symmetry, the NN output satisfies symmetry. Namely, let $\boldsymbol{W}$ be the weight matrix between the input and the hidden layer. The NN output satisfies symmetry if the following equation holds:

$$\boldsymbol{W}\begin{pmatrix}\boldsymbol{a}\\ \boldsymbol{b}\end{pmatrix} = \boldsymbol{W}\begin{pmatrix}\boldsymbol{b}\\ \boldsymbol{a}\end{pmatrix}. \tag{3.23}$$

Let $\boldsymbol{W}_1$ be the weight matrix between the partial input layer for the first object and the entire hidden layer, $\boldsymbol{W}_2$ be the weight matrix between remaining input layer for the second object and the entire hidden layer. Since $\boldsymbol{W} = (\boldsymbol{W}_1, \boldsymbol{W}_2)$, (3.23) can be rewritten as $\boldsymbol{W}_1(\boldsymbol{a}-\boldsymbol{b}) = \boldsymbol{W}_2(\boldsymbol{a}-\boldsymbol{b})$. For arbitrary $\boldsymbol{a}$ and $\boldsymbol{b}$, this equation holds if $\boldsymbol{W}_1 = \boldsymbol{W}_2$. However, there is actually a problem here. If this equation holds, the following equation holds.

$$\boldsymbol{W}(\boldsymbol{a};\boldsymbol{b}) = \hat{\boldsymbol{W}}(\boldsymbol{a}+\boldsymbol{b}), \tag{3.24}$$

where $\hat{\boldsymbol{W}}$ is equal to $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. From (3.24), it follows that the addition of feature vectors of two objects is input to the DNN. However, the addition of feature vectors of two objects is not suitable for pairwise classification because the information about the features of each object is lost. For example, in the author matching problem, it cannot be determined which words appear in which papers.

3.3.7 Higher-Order Pairwise Deep Neural Networks

Finally, we present DNN-based models that enable higher-order feature conjunctions across objects. The proposed models use higher-order feature conjunction across objects explicitly. Moreover, the proposed models with symmetry do not lost the information about the features of each object.

We first give the interpretation of the PairNet as a neural network (NN). (3.11) can be rewritten:

$$f_{\mathrm{Pair}}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{\lambda}, \boldsymbol{U}, \boldsymbol{V}) = \sum_{s=1}^{k} \lambda_s \left\langle \mathrm{vec}(\boldsymbol{b} \otimes \boldsymbol{a}), \mathrm{vec}(\boldsymbol{v}_{:,s} \otimes \boldsymbol{u}_{:,s}) \right\rangle . \tag{3.25}$$

Therefore, the PairNet can be regarded as a depth-2 NN with $\mathrm{vec}(\boldsymbol{b} \otimes \boldsymbol{a})$ as input, an identity function as the activation function, $\mathrm{vec}(\boldsymbol{v}_{:,s} \otimes \boldsymbol{u}_{:,s})$ as the weight between the input and s-th units in the hidden layer, $\boldsymbol{\lambda}$ as the weight between the hidden layer and the output unit, and k is the number of hidden units. We propose a *pairwise deep neural network* (PairDNN):

$$f_{\mathrm{PairDNN}}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{\Theta}) \coloneqq f_{\mathrm{DNN}}(\sigma(\boldsymbol{z}); \boldsymbol{\Theta}), \tag{3.26}$$

where $\sigma(\boldsymbol{z}) = (\sigma(z_1), \ldots, \sigma(z_k))$, $z_s = \mathcal{P}^2\left((\boldsymbol{u}_{:,s}, \boldsymbol{v}_{:,s}), (\boldsymbol{a}, \boldsymbol{b})\right)$ for $s \in [k]$, is the input of $f_{\mathrm{DNN}}(\cdot)$, $\sigma : \mathbb{R} \to \mathbb{R}$ is an element-wise activation function, $f_{\mathrm{DNN}} : \mathbb{R}^k \to \mathbb{R}$ is a DNN, and $\boldsymbol{\Theta}$ is the set of parameters for the DNN (we do not specify the architecture (form) of f_{DNN}). This modeling is an analogy of some existing DNN-extensions of FMs [27, 58, 77, 24, 38]. A PairDNN can be regarded as a DNN with $\mathrm{vec}(\boldsymbol{b} \otimes \boldsymbol{a})$ as the input and $\sigma(\cdot)$ as the activation function in the first hidden layer. The reason we introduce $\sigma(\cdot)$ is that the activation function in DNNs is a commonly used non-linear function. If $\sigma(\cdot)$ is an identity function and $\boldsymbol{U} = \boldsymbol{V}$, the PairDNN is equivalent to the *Pairwise DNN* [2]. The PairNet, PairDNN, and Pairwise DNN enable the use of second-order feature interactions across objects without computing it directly and the same technique recently proposed by other researchers [37, 50]. While the DNN can extract useful feature representation, introducing the layer using feature interactions explicitly improves the performance of the DNN. Indeed, the DNN-based models using feature interactions outperformed simple DNNs in some applications [28, 39, 37].

We also propose a *higher-order pairwise deep neural network* (HOPairDNN) by defining $z_s^{(m)} = \sum_{t=2}^{m} \mathcal{P}^t\left(\left(\boldsymbol{u}_{:,s}^{(t)}, \boldsymbol{v}_{:,s}^{(t)}\right), (\boldsymbol{a}, \boldsymbol{b})\right)$ for $s \in [k]$ in (3.26). The HOPairDNN can be regarded as a DNN with from 2 to m-order feature interactions across objects as input. We call the layer that computes $\boldsymbol{z}^{(m)}$ *higher-order pairwise interaction layer*.

3.3.8 Relationship between Proposed Methods and Existing Methods for Index-based Link Prediction

As described in Section 3.1, while index-based link prediction methods cannot predict links between an unknown object and unknown or known objects, feature-based link prediction

methods can do it as long as the feature vectors are given. Moreover, methods for feature-based link prediction can be applied to index-based link prediction by regarding the indices of objects as features; given indices of nodes a_{ind} and b_{ind}, one-hot encoding vectors of indices can be used as feature vectors of nodes. Then, d_1 and d_2 correspond to the number of nodes. The model equation of the latent factor (feature) model [35] [1], which is a well-known method for index-based link prediction, $f_{\text{latent}}(a_{\text{ind}}, b_{\text{ind}}) : [d_1] \times [d_2] \mapsto \langle \boldsymbol{u}_{a_{\text{ind}}}, \boldsymbol{v}_{b_{\text{ind}}} \rangle + w^{(a)}_{a_{\text{ind}}} + w^{(b)}_{b_{\text{ind}}}$, where $\boldsymbol{u}_i \in \mathbb{R}^k$ $(i \in [d_1])$, $\boldsymbol{v}_j \in \mathbb{R}^k$ $(j \in [d_2])$, $\boldsymbol{w}^{(a)} \in \mathbb{R}^{d_1}$ and $\boldsymbol{w}^{(b)} \in \mathbb{R}^{d_2}$ are learnable parameters. Parameters $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ are called *latent factors*, and $\boldsymbol{w}^{(a)}$ and $\boldsymbol{w}^{(b)}$ are called *biases*. It is known that the FM for index-based link prediction (i.e., using one-hot encoding vectors) is equivalent to this latent factor model and the FM using both indices and additional features of objects outperformed the latent factor model in the recommendation task [60]. The PairNet for index-based link prediction is equivalent to the latent factor model without biases: $f_{\text{Pair}}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{1}, \boldsymbol{U}, \boldsymbol{V}) = \langle \boldsymbol{u}_{a_{\text{ind}}}, \boldsymbol{v}_{b_{\text{ind}}} \rangle$. Furthermore, from this result, it is clear that the second-order pairwise network layer for one-hot encoding vectors is equivalent to the generalized matrix factorization layer, which is used in the *neural collaborate filtering* [28] that is a DNN-extension of the latent factor model.

In index-based link prediction setting, one can obtain some feature representations by some graph embedding methods [23]. Several researches showed that using both indices and features of objects could improve the performances [60, 43, 78]. Moreover, some researches showed the effectiveness of the bilinear pooling, which uses second-order feature interactions between extracted (embedded) feature vectors [39, 37]. Thus, it seems promising to use feature-based link prediction methods that use feature interactions (e.g., proposed methods) with both indices and embedded features of objects in index-based link prediction. We leave the investigation of it for future work.

There have been many other index-based link prediction methods and recently the method using graph NN [73] has been proposed [76]. Since this is also a method for index-based link prediction, this method cannot predict links for unknown nodes although they can leverage additional features of nodes. On the other hand, our methods (and other methods for feature-based link prediction) can learn models without indices of objects and predict links for unknown objects.

3.4 Experiments

We evaluated the performance of our proposed models using three feature-based link prediction tasks: author disambiguation, co-author link prediction and recommendation.

3.4.1 Datasets

- **DBLP.** For the author disambiguation task, we extracted 3,384 papers in which there were 729 unique author names from the DBLP dataset. Each paper was considered an object. If two papers had an author in common, we gave that pair of papers a positive label. We used all the words in the titles, the coauthor names, and the publication venues for creating bag-of-words feature vectors.

- **NIPS.** For the co-author link prediction task, we obtained a dataset of co-author graphs from the first 12 editions of the Neural Information Processing Systems

[1] This method is also called *matrix factorization* in the context of the index-based link prediction.

Table 3.1: Datasets for evaluation.

Dataset	d_1	d_2	N_{train}	N_{valid}	N_{test}
DBLP	9,264		22,416	1,000	21,264
NIPS	13,649		3,134	134	3,000
ML100K	49	29	21,200	1,000	20,200

Conference [63]. There were 2,037 authors in this dataset. If two authors had collaborated, we gave that pair of authors a positive label. Each object (author) was represented by a bag-of-words feature vector, which used words in their publications.

- **ML100K.** For the recommendation task, we obtained a dataset from the MovieLens 100K (ML100K) dataset [25] that contained data for 943 users and 1682 movies ($\boldsymbol{a}$ represented a user, and $\boldsymbol{b}$ represented a movie). For features and labels, we generally followed the practice of Novikov et al. [49] but we did not use user and movie indices.

The number of positive and negative pairs in all datasets were equalized by under-sampling the negative pairs similarly in [51]. For all datasets, we created a validation set by randomly sampling the test set after creating the training and test sets. The details for the three datasets are summarized in Table 3.1. Intuitively, proposed methods seem suitable for the DBLP and NIPS dataset. On the other hand, proposed methods do not seem suitable for the ML100K dataset since feature interactions from the same object can be effective (e.g., there may be movies that tend to receive high ratings).

3.4.2 Comparison with HOPairNets and Existing Models

We first compared our proposed HOPairNet with these existing models:

- **HOPairNet**. Our proposed HOPairNet defined in (3.10). We minimized the objective function (3.19) by using the CD algorithm. We introduced $\boldsymbol{\lambda}$ when $m \geq 4$ because of the Lemma 3.1. For the DBLP and NIPS datasets, we used the symmetrization method described in Section 3.3.5.

- **HOFM**. The higher-order factorization machine [60, 9] defined in (2.31). It was optimized by using the CD algorithm.

- **PairSVM**. We used the SVM with a second-order pairwise kernel proposed by Oyama and Manning [51], Ben-Hur and Noble [5]:
  $y = \sum_{i=1}^{N} \alpha_i \mathcal{P}^2((\boldsymbol{a}_i, \boldsymbol{b}_i), (\boldsymbol{a}, \boldsymbol{b}))$.

For the DBLP and NIPS datasets, we set $k = 30$ for the **HOPairNet** and **HOFM** following [10]. For the ML100K dataset, we set $k = 10$ for the **HOPairNet** and **HOFM**. For the **HOPairNet** and **HOFM** we set β (a regularization hyper-parameter) to $10^{-7}, 10^{-6}$, or 10^{-5} on the basis of the accuracy for the validation set. We set the regularization hyper-parameter in the objective function of the PairSVM to $10^{-7}, 10^{-6}, \ldots, 10^{7}$. We set $\ell(\cdot, \cdot)$ as the logistic loss for the **HOPairNet**, **HOFM**. We compared the accuracies of these models for the three test sets. Note that this is an experiment of feature-based link prediction task and hence we did not use indices of objects (namely, the adjacency-matrix/graph). Hence, we cannot compare these methods and other methods for index-based link prediction that require indices of objects, e.g., latent factor models [35] and the neural collaborate

Table 3.2: Comparison of accuracies of PairNets with those of existing models.

Model	DBLP	NIPS	ML100K
FBM ($m = 2$)	0.7639	0.8567	0.5951
HOPairNet ($m = 3$)	**0.7827**	**0.8570**	**0.6239**
HOPairNet ($m = 4$)	0.7761	0.8443	0.607
HOFM ($m = 2$)	0.7037	0.7840	0.6318
HOFM ($m = 3$)	0.7470	0.7858	**0.6362**
HOFM ($m = 4$)	0.7412	0.7840	0.6225
PairSVM	0.7290	**0.9171**	0.5972

filtering [28], the graph neural networks [76]. Results are shown in Table 3.2. For the DBLP dataset, the third-order **HOPairNet** achieved the best accuracy. The accuracy of the **HOPairNet** was more robust than that of the **HOFM** with respect to the order of feature interactions. Higher-order feature interactions across objects can be effective as shown by the better performance of the third- and fourth-order **HOPairNet** compared to the second-order **HOPairNet** (which is a PairNet equivalent to the FBM) for all datasets. The better performance of the **HOPairNet** compared to the **HOFM** for the DBLP and NIPS datasets shows that using feature interactions only across objects is important for such problems as predicting identity.

While the **PairSVM** achieved the best performance for the NIPS dataset, its performance for the DBLP dataset was not good. The proportion of non-zero values in the DBLP and NIPS datasets was 0.1% and 7.5%. Hence, if the data are not very sparse and the number of training instances is not so large, the **PairSVM** is a good choice.

For the ML100K dataset, the **HOFM** outperformed the **HOPairNet** and **PairSVM** because the **HOFM** use feature interactions from the same object and the ML100K dataset is designed for recommender systems as described above. Because the ML100K dataset is designed for recommender systems, use of feature interactions from the same object can be effective (e.g., there may be movies that tend to receive high ratings).

3.4.3 Comparison with HOPairDNN and Existing DNN-based Models

Next we compared our proposed HOPairDNN with a DNN-based FM/HOFM for the DBLP dataset.

- **HOPairDNN**. Our proposed HOPairDNN is described in Section 3.3.7. We minimized the logistic loss without regularization terms by using the Adam stochastic optimization method [33]. We set the learning rate for $\boldsymbol{\Theta}$ to the default value (0.001), and for $\boldsymbol{U}^{(t)}, \boldsymbol{V}^{(t)},\ t \in [2:m]$ to $0.001, 0.002, \ldots$, or 0.009. We used the identity function as an activation function in the higher-order pairwise interaction layer $\sigma(\cdot)$. We used the relu function as an activation function in f_{DNN}.

- **HONFM**. The **HONFM** is the higher-order extension of the NFM [27]. We also tuned the **HONFM** as for the **HOPairDNN**.

- **Concat**. A DNN with $(\boldsymbol{a}; \boldsymbol{b})$ as input. We used Adam with the learning rate $0.001, 0.002, \ldots,$ or 0.009. We used relu activation function in all hidden layers.

Table 3.3: Comparison of accuracies of DNN-based models for DBLP dataset with those of existing models. For comparison, some results from Table 3.2 are shown.

Model	Accuracy	Model	Accuracy
HOPairDNN ($m = 2$)	0.8527	**FBM** ($m = 2$)	0.7639
HOPairDNN ($m = 3$)	**0.8547**	**HOPairNet** ($m = 3$)	0.7827
HOPairDNN ($m = 4$)	0.8483	**HOPairNet** ($m = 4$)	0.7761
HONFM ($m = 2$)	0.8144	**HOFM** ($m = 2$)	0.7037
HONFM ($m = 3$)	0.8142	**HOFM** ($m = 3$)	0.7470
HONFM ($m = 4$)	0.8124	**HOFM** ($m = 4$)	0.7412
Concat	0.7600	**PairSVM**	0.7290

Table 3.4: Comparison of ROC-AUCs, precisions, and recalls for imbalanced DBLP dataset.

Model	ROC-AUC	Precision	Recall
HOPairDNN ($m = 2$)	0.9064	0.0822	0.7629
HOPairDNN ($m = 3$)	0.9057	0.1392	0.7154
HOPairDNN ($m = 4$)	**0.9124**	0.1873	0.6993
HOPairNet ($m = 2$)	0.8350	0.0800	0.5659
HOPairNet ($m = 3$)	0.8393	0.0822	0.5925
HOPairNet ($m = 4$)	0.8384	0.0792	0.6075
HONFM ($m = 2$)	0.8700	0.0297	**0.7841**
HONFM ($m = 3$)	0.8676	0.0359	0.7487
HONFM ($m = 4$)	0.8670	0.0364	0.7456
HOFM ($m = 2$)	0.7892	0.0097	0.3708
HOFM ($m = 3$)	0.8144	0.0417	0.6221
HOFM ($m = 4$)	0.8128	0.0425	0.6158
Concat	0.8331	0.0923	0.5379
PairSVM	0.8811	**0.2047**	0.5223

The number of hidden layers was four in all models. We used the Dropout [68] for regularization in all models. We adjusted the number of hidden units to make the number of parameters almost the same as that for the simple DNN that has four hidden layers with 1,000 units and whose input vectors are the addition of $\boldsymbol{a}$ and $\boldsymbol{b}$: $\boldsymbol{a} + \boldsymbol{b}$. We ran the experiment five times using different initial values and compared the average values.

As shown in Table 3.3, the **HOPairDNN**, especially the third-order one, achieved the best performance. Note that the differences in performance between the second-, third-, and fourth-order **HOPairDNN** were smaller than those for the **HOPairNet**. Since DNN-based models are strongly non-linear, the effect of higher-order feature interactions may be smaller. Although the results of the **HONFM**, which are the DNN-extension of the **HOFM**, were better than those of **HOFM** and PairNets, they were inferior to those of **HOPairDNN**. Both **HONFM** and **HOPairNet** outperformed the **Concat** and thus explicitly using feature interactions are effective.

3.4.4 Comparison on Imbalanced Setting

Next, we compared our proposed methods and existing methods on the *imbalanced* DBLP dataset. Although we undersampled the negative pairs for training, validation, and test

data in Section 3.4.2 and Section 3.4.3, we undersampled the negative pairs for only training data in this experiment. The number of pairs in validation and test data were $284,099$ and $1,136,398$, respectively. We used the same training data as in Section 3.4.2 and Section 3.4.3. We used the area under the receiver operating characteristic curve (ROC-AUC), precision, and recall as the evaluation metrics since the number of negative pairs was one hundred times more than that of positive pairs in imbalanced DBLP dataset. We tuned the hyper-parameters of the proposed and existing methods as in Section 3.4.2 and Section 3.4.3 with ROC-AUC as the evaluation metric.

As shown in Table 3.4, the **HOPairDNN**, especially fourth-order one, achieved the best ROC-AUC, and our proposed **HOPairDNN** outperformed **HOFM**. Although the precisions of all models were low, that of the **PairSVM** was higher than those of other models and thus the ROC-AUC of the **PairSVM** was higher than those of the **HOPairDNN**, **HOFM**, and **HONFM**. In our hyper-parameter tuning scenario, the highest recall of the **PairSVM** was 0.5247, it was lower than those of proposed models in Table 3.4. We note that the **HOPairDNN** achieved the best precision and F-measure when we tuned the hyper-parameters of all models on the basis of the F-measure for the validation set; the highest F-measure and precision were 0.4024 and 0.9077, respectively. Therefore, our experimental results suggested that a machine learning user should use the **HOPairDNN** or **PairSVM** if the precision is more important, otherwise, normally use the **HOPairDNN**.

3.5 Conclusion

We have presented models using higher-order feature interactions only across the two objects being compared in Chapter 3. Our proposed model, HOPairNet, can be regarded as a higher-order generalization of the factorized bilinear model or pairwise extension of the higher-order factorization machine. We have also presented an algorithm for efficiently computing higher-order feature interactions only across two objects. Moreover, we have proposed an efficient CD algorithm for proposed models. Furthermore, we have proposed the HOPairDNN, which is a DNN-extension of the HOPairNet. In addition, we have presented the relationships among proposed methods, existing methods for feature-based link prediction and for index-based link prediction. Experimental results demonstrated the effectiveness of our proposed models.

Chapter 4

Random Feature Maps for Efficiently Using Feature Interactions

4.1 Introduction

Kernel methods enable learning in high, possibly infinite-dimensional feature spaces without explicitly expressing them. In particular, kernels that model feature combinations such as polynomial kernels, the ANOVA kernel, and the all-subsets kernel [9, 64] have been shown to be effective for a number of tasks in computer vision and natural language understanding [39, 21]. However their scalability remains a challenge; support vector machines (SVMs) with non-linear kernels require $O(N^2)$ time and $O(N^2)$ memory for training and $O(N)$ time and memory for evaluation, where N is the number of training instances [12].

To address this issue several researchers have proposed *random feature maps* $Z : \mathbb{R}^d \to \mathbb{R}^D$ for kernels $K(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ that satisfy

$$\mathbb{E}[\langle Z(\boldsymbol{x}), Z(\boldsymbol{y})\rangle] = K(\boldsymbol{x}, \boldsymbol{y}). \tag{4.1}$$

The idea is to perform classification, regression, or clustering on a corresponding high-dimensional feature space approximately but efficiently using linear models in a low-dimensional space by mapping the data points using $Z(\cdot)$. Examples include *random Fourier feature maps* that approximate shift-invariant kernels: $K(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y})$ [59]; *random Maclaurin feature maps* that approximate dot product kernels [32]: $K(\boldsymbol{x}, \boldsymbol{y}) = k(\langle \boldsymbol{x}, \boldsymbol{y}\rangle)$; *tensor sketching* for polynomial kernels: $K^m_{\text{poly}}(\boldsymbol{x}, \boldsymbol{y}; c) \coloneqq (\langle \boldsymbol{x}, \boldsymbol{y}\rangle + c)^m$ [56], and so on [40, 55, 42, 67, 40].

FMs [60, 61] and variants [9, 10, 49] also model feature combinations without explicitly computing them, similar to kernel methods, but have better scalability during evaluation. These methods can be thought of as a two-layer neural network with polynomial activations with a fixed number of learnable parameters (see (2.23)). However, unlike kernel methods, their optimization problem is generally non-convex and difficult to solve. But due to their efficiency during evaluation FMs are attractive for large-scale problems and have been successfully applied to applications such as link prediction and recommender systems. This work analyzes the relationship between polynomial kernel models and factorization machines in more detail.

In this chapter, we present a random feature map for the *itemset kernel* that takes into account all feature combinations within a family of itemsets $\mathcal{S} \subseteq 2^{[d]}$. To the best of our knowledge, the random feature map for the itemset kernel is novel. The itemset

kernel includes the ANOVA kernel, all-subsets kernel, and standard dot product, so linear models using this map are an alternative to the ANOVA or all-subsets kernel SVMs, FMs, and all-subsets model. They scale well with the size of the training dataset, unlike kernel methods, and their optimization problem is convex and easy to solve, unlike that of FMs. We also present theoretical analyses of the proposed random feature map and discuss the relationship between linear models trained on these features and factorization machines. Furthermore, we present a faster and more memory-efficient random feature map for the ANOVA kernel based on the signed circulant matrix technique [20]. Finally, we evaluate the effectiveness of the feature maps on several datasets.

This chapter is organized as follows. In Section 4.2, we describe random feature maps methodology and some existing methods. We present our basic algorithm and some theoretical guarantees in Section 4.3. We introduce some extensions of the itemset kernel and how to modify our algorithm for such extension in Section 4.4. Section 4.5 and Section 4.6 present faster and more memory-efficient algorithms than our basic algorithm described in Section 4.2. We demonstrate the proposed methods on synthetic and real-world datasets in Section 4.7 and Section 4.8. Most of the proofs are presented in Section 4.10.

4.2 Random Feature Maps and Related Work

Rahimi and Recht [59] proposed the random Fourier feature map $Z_{\mathrm{RF}} : \mathbb{R}^d \to \mathbb{R}^D$ for shift-invariant kernels $K(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y})$. Their method is based on Bochner's theorem.

Theorem 4.1 (Bochner [59])**.** *A continuous kernel $K(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y})$ on $\mathbb{R}^d$ is positive (semi) definite if and only if k is the Fourier transform of a non-negative measure.*

From this theorem, for a shift-invariant kernel k, we have

$$k(\boldsymbol{x} - \boldsymbol{y}) = \frac{1}{Z} \int p(\boldsymbol{\omega}) \exp(i \langle \boldsymbol{\omega}, \boldsymbol{x} - \boldsymbol{y} \rangle) d\boldsymbol{\omega} \tag{4.2}$$

$$= \frac{1}{Z} \mathbb{E}_{\boldsymbol{\omega} \sim p}[\langle (\cos(\langle \boldsymbol{\omega}, \boldsymbol{x} \rangle), \sin(\langle \boldsymbol{\omega}, \boldsymbol{x} \rangle)), (\cos(\langle \boldsymbol{\omega}, \boldsymbol{y} \rangle), \sin(\langle \boldsymbol{\omega}, \boldsymbol{y} \rangle)) \rangle] \tag{4.3}$$

$$= \frac{1}{Z} \mathbb{E}_{\boldsymbol{\omega} \sim p, b \sim \mathcal{U}(0, 2\pi)}[\sqrt{2} \cos(\langle \boldsymbol{\omega}, \boldsymbol{x} \rangle + b) \sqrt{2} \cos(\langle \boldsymbol{\omega}, \boldsymbol{y}) + b \rangle], \ \exists Z > 0, \tag{4.4}$$

where $Z > 0$ and p is the scaled Fourier transform of k such that $\int p(\boldsymbol{\omega}) d\boldsymbol{\omega} = 1$ and $\mathcal{U}$ is a uniform distribution. Therefore, the map $Z_{\mathrm{RF}} : \mathbb{R}^d \to \mathbb{R}^D$

$$Z_{\mathrm{RF}}(\boldsymbol{x}) \coloneqq \sqrt{\frac{2}{ZD}} \cos(\boldsymbol{\Omega} \boldsymbol{x} + \boldsymbol{b}), \tag{4.5}$$

where $\boldsymbol{\Omega} \in \mathbb{R}^{D \times d}$ and $\boldsymbol{b} \in [0, 2\pi]^D$ such that $\boldsymbol{\omega}_j \sim p$ and $b_j \sim \mathcal{U}(0, 2\pi)$, and cos is element-wise, approximates the feature space induced by k in the sense of the dot product:

$$\mathbb{E}_{\boldsymbol{\omega} \sim p, b \sim \mathcal{U}(0, 2\pi)} \left[\langle Z_{\mathrm{RF}}(\boldsymbol{x}), Z_{\mathrm{RF}}(\boldsymbol{y}) \rangle \right] = k(\boldsymbol{x} - \boldsymbol{y}) = K(\boldsymbol{x}, \boldsymbol{y}). \tag{4.6}$$

The dot product of two random Fourier feature maps $\langle Z_{\mathrm{RF}}(\boldsymbol{x}), Z_{\mathrm{RF}}(\boldsymbol{y}) \rangle$ can be interpreted as a Monte Carlo approximation of $K(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y})$. Linear models with Z_{RF} approximates the KR model with K and scale well w.r.t N.

Based on their idea, many random feature maps for various kernel functions have been proposed [32, 56, 40, 55, 42, 67, 40]. In this section, we introduce the existing random feature maps for dot product kernels and polynomial kernels [32, 56].

Algorithm 3 Random Maclaurin Map

Input: $\boldsymbol{x} \in \mathbb{R}^d$, $p > 0$, $a_n = k^{(n)}(0)/n!$ for all $n \in \mathbb{N}_{\geq 0}$

1: **for** $s \leftarrow 1, \ldots, D$ **do**
2: Generate non-negative integer n with distribution $p(n) \propto 1/(p^{n+1})$;
3: Generate n Rademacher vectors $\boldsymbol{\omega}_{s,1}, \ldots, \boldsymbol{\omega}_{s,n} \in \{-1, +1\}^d$;
4: Compute $Z_s = \sqrt{a_n p^{n+1}} \prod_{j=1}^{n} \langle \boldsymbol{\omega}_{s,j}, \boldsymbol{x} \rangle$;
5: **end for**

Output: $Z_{\mathrm{RM}}(\boldsymbol{x}) = (Z_1, \ldots, Z_D)^\top / \sqrt{D}$

Algorithm 4 Tensor Sketching for m-order Polynomial Kernel

Input: $\boldsymbol{x} \in \mathbb{R}^d$

1: Compute m different count sketches of $\boldsymbol{x}$: $\boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(m)} \in \mathbb{R}^D$;
2: Compute FFT of each count sketch: $\boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(m)} \leftarrow \mathrm{FFT}(\boldsymbol{c}^{(1)}), \ldots, \mathrm{FFT}(\boldsymbol{c}^{(m)})$;
3: Compute element-wise product: $\boldsymbol{z} \leftarrow \boldsymbol{c}^{(1)} \circ \cdots \circ \boldsymbol{c}^{(m)}$;

Output: $Z_{\mathrm{TS}}(\boldsymbol{x}) = \mathrm{FFT}^{-1}(\boldsymbol{z})$

4.2.1 Random Feature Maps for Polynomial Kernels

The random Maclaurin (RM) feature map [32] is for dot product kernels: $K(\boldsymbol{x}, \boldsymbol{y}) = k(\langle \boldsymbol{x}, \boldsymbol{y} \rangle)$. It uses the Maclaurin expansion of $k(\cdot)$: $k(x) = \sum_{n=0}^{\infty} a_n x^n$, where $a_n = k^{(n)}(0)/n!$ is the n-th coefficient of the Maclaurin series. It uses two distributions: $p_{\mathrm{order}}(N = n) \propto 1/p^{n+1}$, where $p > 1$, and the Rademacher distribution (a fair coin distribution). The RM map procedure is shown in Algorithm 3. Its computational cost is $O\left(\sum_{s=1}^{D} N_s d\right)$ time and memory, where N_s ($s \in [D]$) is the order of the s-th randomized feature, especially $O(Ddm)$ time and memory when the objective kernel is the homogeneous polynomial kernel: $K^m_{\mathrm{HP}}(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle^m$.[1]

The tensor sketching (TS) [56] is a random feature map for the homogeneous polynomial kernel $K^m_{\mathrm{poly}}(\boldsymbol{x}, \boldsymbol{y}; 0)$. Because polynomial kernels $K^m_{\mathrm{poly}}(\boldsymbol{x}, \boldsymbol{y}; c) = (c + \langle \boldsymbol{x}, \boldsymbol{y} \rangle)^m$ can be written as K^m_{HP} by concatenating $\sqrt{c}$ to each vector, a TS can approximate K^m_{poly}. Although an RM feature map can also approximate polynomial kernels, the TS can approximate them more efficiently. The TS is based on a fast algorithm to compute a count sketch [13] of an outer product of two vectors [52]. The count sketch is a method to estimate the frequency of all items in a stream [13], and the machine learning community uses it as a dimensionality reduction method [71] since it preserves the standard dot products in the sense of expectation. Although the count sketch of an outer product of two vectors approximates polynomial kernels, it is prohibitive to compute an outer product of two vectors naïvely from the point of view of computational cost. Therefore, Pham and Pagh [56] showed that for two vectors the convolution of two count sketches is a count sketch of an outer product of two vectors, and the former can be computed efficiently. This idea is originally proposed to approximate the matrix multiplications of two matrices [52] and based on an efficient algorithm for the convolution of polynomials using a fast Fourier transform (FFT) and an inverse FFT (IFFT). Algorithm 4 shows the procedure of the TS for the m-order polynomial kernel. Their tensor sketch algorithm takes $O(m(d + D \log D))$ time and $O(md \log D)$ memory and is thus more efficient than the random Maclaurin

[1] When the objective kernel is a homogeneous polynomial kernel, one can fix $n = m$ and $p_{\mathrm{order}}(N = m) = 1$ otherwise 0; that is, do not sample n.

Algorithm 5 Random Kernel Feature Map

Input: $\boldsymbol{x} \in \mathbb{R}^d$, $\mathcal{S} \subseteq 2^{[d]}$

1: Generate D Rademacher vectors $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D \in \{-1, +1\}^d$;

2: Compute D itemset kernels $K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}_s)$ for all $s \in [D]$;

Output: $Z(\boldsymbol{x}) = \frac{1}{\sqrt{D}} \left(K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}_1), \ldots, K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}_D)\right)^\top$;

algorithm.

Linear models using the TS or RM feature map are a good alternative to polynomial kernel SVMs and PNs [32, 56]. Similarity, although linear models using a random feature map that approximates the itemset kernel would be a good alternative for the ANOVA or all-subsets kernel SVMs, FMs, and all-subsets models, such a map has not yet been reported.

4.3 Random Feature Map for Itemset Kernel

In this section, we propose a random feature map for the itemset kernel, showed some theoretical analyses, and presented a faster and more memory-efficient algorithm especially for the ANOVA kernel. We recall that the itemset kernel for a given family of itemsets $\mathcal{S} \subseteq 2^{[d]}$ is defined as

$$K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \sum_{V \in \mathcal{S}} \prod_{j \in V} x_j y_j = \langle \phi_{\mathcal{S}}(\boldsymbol{x}), \phi_{\mathcal{S}}(\boldsymbol{y}) \rangle \,. \tag{2.22}$$

As shown in Algorithm 5, the proposed *random kernel (RK) map* is simple: (1) generate D Rademacher vectors from a Rademacher distribution and (2) compute D itemset kernels between the original feature vector and each Rademacher vector. Mathematically, the RK feature map $Z_{\mathrm{RK}} : \mathbb{R}^d \to \mathbb{R}^D$ for the itemset kernel $K_{\mathcal{S}}$ (2.22) is defined as

$$Z_{\mathrm{RK}}(\boldsymbol{x}; \mathcal{S}, \boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D) \coloneqq \frac{1}{\sqrt{D}} (K_S(\boldsymbol{x}, \boldsymbol{\omega}_1), \ldots, K_S(\boldsymbol{x}, \boldsymbol{\omega}_D))^\top, \tag{4.7}$$

where $\boldsymbol{\omega}_s \in \{-1, 1\}^d$ is the vector sampled from the Rademacher distribution. The RK map algorithm is shown in Algorithm 5. The following proposition states that the RK feature map approximates the itemset kernel.

Proposition 4.2. *Let $Z_{\mathrm{RK}} : \mathbb{R}^d \to \mathbb{R}^D$ be the random kernel (RK) feature map in Algorithm 5. Then, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and $\mathcal{S} \subseteq 2^{[d]}$,*

$$\mathbb{E}_{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D} [\langle Z_{\mathrm{RK}}(\boldsymbol{x}; \mathcal{S}, \boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D), Z_{\mathrm{RK}}(\boldsymbol{y}; \mathcal{S}, \boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D) \rangle] = K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}). \tag{4.8}$$

Hence, linear models using the proposed RK feature map can use feature combinations efficiently and are a good alternative to FMs and all-subsets models.

4.3.1 Analyses

We next present some theoretical results for the RK feature map. We first analyze the precision of the RK feature map. Let $\mathcal{E}(\boldsymbol{x}, \boldsymbol{y})$ be the approximation error: $\mathcal{E}(\boldsymbol{x}, \boldsymbol{y}) \coloneqq$

$\langle Z_{\mathrm{RK}}(\boldsymbol{x}), Z_{\mathrm{RK}}(\boldsymbol{y})\rangle - K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y})$. We assume that the ℓ_1 norm of the feature vector is bounded: $\|\boldsymbol{x}\|_1 \leq R$, where $R \in \mathbb{R}_{>0}$. This assumption is the same as the one used in previous research [32, 56, 59]. For convenience, we use the same notation as Kar and Karnick [32]: $\mathcal{B}_p(\mathbf{0}, R) = \{\boldsymbol{x} \mid \|\boldsymbol{x}\|_p \leq R\}$. With this notation, the assumption above is written as $\boldsymbol{x} \in \mathcal{B}_1(\mathbf{0}, R)$. Then, we have the following useful absolute error bound.

Lemma 4.3. *For all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}_1(\mathbf{0}, R) \subseteq \mathbb{R}^d$, and $\mathcal{S} \subseteq 2^{[d]}$,*

$$p(|\mathcal{E}(\boldsymbol{x}, \boldsymbol{y})| \geq \varepsilon) \leq 2 \exp\left(\frac{-D\varepsilon^2}{2e^{4R}}\right). \tag{4.9}$$

This upper bound does not depend on the family of itemsets $\mathcal{S}$ or on the dimension of the original feature vectors d. This result comes from the assumption that data points are restricted in $\mathcal{B}_1(\mathbf{0}, R)$.

Next, we consider the uniform bound on the absolute error of the RK feature map. Kar and Karnick [32] derived the uniform bound on the absolute error of the RM feature map and we follow their approach. Let the domain of feature vectors $\mathcal{B} \subseteq \mathcal{B}_1(\mathbf{0}, R)$ be the compact subset of $\mathbb{R}^d$. Then, $\mathcal{B}$ can be covered by a finite number of balls [15], and one can obtain the following uniform bound.

Lemma 4.4. *Let $\mathcal{B} \subseteq \mathcal{B}_1(\mathbf{0}, R)$ be a compact subset of $\mathbb{R}^d$. Then, for all $\mathcal{S} \subseteq 2^{[d]}$,*

$$p\left(\sup_{\boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}} |\mathcal{E}(\boldsymbol{x}, \boldsymbol{y})| \geq \varepsilon\right) \leq 2\left(\frac{32R\sqrt{d}e^{2R}}{\varepsilon}\right)^{2d} \exp\left(-\frac{D\varepsilon^2}{8e^{4R}}\right). \tag{4.10}$$

This uniform bound says that, by taking $D = \Omega\left(\frac{de^{4R}}{\varepsilon^2} \log\left(\frac{R\sqrt{d}e^{2R}}{\varepsilon\delta}\right)\right)$, the absolute error is uniformly lower than ε with a probability of at least $1 - \delta$. This uniform bound also does not depend on the family of itemsets $\mathcal{S}$; it depends only on ε, the dimension of random feature map D, the dimension of the original feature vectors d, and the upper bound on the ℓ_1 norm of the original feature vectors R. The behavior of this uniform bound w.r.t d, ε, and δ is expressed in the form of $D = \Omega\left(\frac{d}{\varepsilon^2} \log\left(\frac{\sqrt{d}}{\varepsilon\delta}\right)\right)$. This is the same as for the RM feature map [32].

We have discussed the upper bounds of the RK feature map for the itemset kernel. Next, we consider the absolute error bound for $K_{\mathcal{S}} = K_{\mathrm{A}}^m$ (that is, $\mathcal{S} = \binom{[d]}{m}$). Here, we also assume that $\boldsymbol{x} \in \mathcal{B}_1(\mathbf{0}, R)$.

Lemma 4.5. *Let $\mathcal{S} = \binom{[d]}{m}$. Then, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}_1(\mathbf{0}, R) \subseteq \mathbb{R}^d$,*

$$p(|\mathcal{E}(\boldsymbol{x}, \boldsymbol{y})| \geq \varepsilon) \leq 2 \exp\left(-\frac{D\varepsilon^2}{2R^{4m}}\right). \tag{4.11}$$

The absolute error bound of Lemma 4.5 is the same as the absolute error bound of the Tensor Sketching [56].

As described above, the algorithm of the proposed RK feature map uses the Rademacher distribution for random vectors. Here, we discuss the generalized RK feature map, which allows the use of other distributions.

Proposition 4.6. *If the distribution of $\boldsymbol{\omega}_s$ for all $s \in [D]$ in Algorithm 1 has (i) a mean of 0 and (ii) a variance of 1, the RK feature map approximates the itemset kernel.*

There are many distributions with a mean of 0 and a variance of 1: the standard Gaussian distribution $\mathcal{N}(0,1)$, the uniform distribution $\mathcal{U}(-\sqrt{3},\sqrt{3})$, the Laplace distribution $\mathrm{Laplace}\left(0,1/\sqrt{2}\right) = \frac{1}{\sqrt{2}}\exp\left(-\sqrt{2}|\omega|\right)$, and so on. Which distribution should be used? The next lemma says that the Rademacher distribution should be used.

Lemma 4.7. *Let $\mathfrak{P}_{0,1}$ be the set of all distributions with a mean of 0 and a variance of 1, and let $p^* \in \mathfrak{P}_{0,1}$ be the Rademacher distribution. Then, for all $p \in \mathfrak{P}_{0,1}$ and $\mathcal{S} \subseteq 2^{[d]}$,*

$$\begin{aligned}&\sup_{\boldsymbol{x},\boldsymbol{y}\in\mathcal{B}_\infty(0,R)} \mathbb{V}_{\boldsymbol{\omega}_1,\dots,\boldsymbol{\omega}_D\sim p^*}[\langle Z_{\mathrm{RK}}(\boldsymbol{x}), Z_{\mathrm{RK}}(\boldsymbol{y})\rangle] \\ &\leq \sup_{\boldsymbol{x},\boldsymbol{y}\in\mathcal{B}_\infty(0,R)} \mathbb{V}_{\boldsymbol{\omega}_1,\dots,\boldsymbol{\omega}_D\sim p}[\langle Z_{\mathrm{RK}}(\boldsymbol{x}), Z_{\mathrm{RK}}(\boldsymbol{y})\rangle]. \qquad (4.12)\end{aligned}$$

That is, a Rademacher distribution achieves the minimax optimal variance for the RK feature map among the valid distributions.

Finally, we discuss the computational complexity of the RK feature map in two special cases. When $K_{\mathcal{S}}(\cdot,\cdot) = K_{\mathrm{A}}^m(\cdot,\cdot)$, a D-dimensional RK feature map takes $O(Ddm)$ time and $O(Dd)$ memory because an m-order ANOVA kernel can be computed in $O(dm)$ time and $O(m)$ memory by using dynamic programming [9, 64]. This is the same as the computational cost for an RM feature map for an m-order polynomial kernel. For $K_{\mathcal{S}}(\cdot,\cdot) = K_{\mathrm{all}}(\cdot,\cdot)$, a D-dimensional RK feature map can be computed in $O(Dd)$ time and $O(Dd)$ memory.

4.3.2 Loglinear Time RK Feature Map for ANOVA Kernel

As described above, the computational cost of the proposed RK feature map in Algorithm 5 clearly depends on the computational cost of the itemset kernel $K_{\mathcal{S}}$. This is a drawback of the RK feature map. The computational cost of the RK feature map for an m-order ANOVA kernel is $O(Ddm)$ time. This cost is the same as that of the RM feature map for an m-order polynomial kernel and larger than that for the TS ($O(m(d + D\log D))$). The number of parameters for the proposed method for an m-order ANOVA kernel is $O(Dd)$, which is also larger than that of the TS ($O(md\log D)$) because $m \ll d < D$ in most cases.

While the random Fourier (RF) feature map, which does not have the order parameter m ($Z_{\mathrm{RF}}(\boldsymbol{x}) = \sqrt{2/D}\cos(\boldsymbol{\Pi}\boldsymbol{x} + \boldsymbol{b})$, where $\boldsymbol{\Pi} \in \mathbb{R}^{D\times d}, \boldsymbol{b} \in \mathbb{R}^d$), also takes $O(Dd)$ time and $O(Dd)$ memory, methods have recently been proposed that take $O(D\log d)$ time and $O(D)$ memory [20, 36]. In this section, we present a faster and more memory efficient RK feature map for the ANOVA kernel based on these recently proposed methods, especially that of Feng et al., which takes $O(mD\log d)$ time and $O(D)$ memory.

First we explain *signed circulant random feature* (SCRF) [20]. The $O(Dd)$ time complexity of the RF feature map is caused by the computation of $\boldsymbol{\Pi}\boldsymbol{x}$. The SCRF reduced it to $O(D\log d)$ time without loss of the key property of the RF feature map; approximating the shift-invariant kernel. In the SCRF, without loss of generality, it is assumed that D is divisible by d ($D/d \coloneqq T$) and that $\boldsymbol{\Pi}$ is replaced by the concatenation of T projection matrices: $\tilde{\boldsymbol{\Pi}} = (\boldsymbol{P}^{(1)}; \boldsymbol{P}^{(2)}; \cdots; \boldsymbol{P}^{(T)})$. $\boldsymbol{P}^{(t)} \in \mathbb{R}^{d\times d}$, $t \in [T]$, is called a *signed circulant random matrix*, which is a variant of the circulant matrix: $\boldsymbol{P}^{(t)} = \mathrm{diag}(\boldsymbol{\sigma}_t)\mathrm{circ}(\boldsymbol{\omega}_t)$, where $\boldsymbol{\sigma}_t \in \{-1,+1\}^d$ is a Rademacher vector, $\boldsymbol{\omega}_t \in \mathbb{R}^d$ is a random vector generated from an appropriate distribution (e.g., the Gaussian distribution for the radial basis function

kernel), and $\mathrm{circ}(\boldsymbol{\omega}_t) \in \mathbb{R}^{d \times d}$ is a circulant matrix in which the first column is $\boldsymbol{\omega}_t$. This formulation clearly reduces the memory required for the RF feature map from $O(Dd)$ to $O(2Td) = O(2D)$. Moreover, the product of $\tilde{\boldsymbol{\Pi}}$ and $\boldsymbol{x}$ surprisingly can be converted into FFT, IFFT, and the element-wise product of vectors which means that time complexity can be reduced from $O(Dd)$ to $O(D \log d)$.

Unfortunately, it is difficult to apply the SCRF technique to the RK feature map because the computation of the itemset kernel does not require the product of a random projection matrix and a feature vector in general. Fortunately, the ANOVA kernel, which is a special case of the itemset kernel, can be computed efficiently [10] by using recursion:

$$K_{\mathrm{A}}^{m}(\boldsymbol{\omega}, \boldsymbol{x}) = \frac{1}{m} \sum_{t=1}^{m} (-1)^{t+1} K_{\mathrm{A}}^{m-t}(\boldsymbol{\omega}, \boldsymbol{x}) \left\langle \boldsymbol{\omega}^{\circ t}, \boldsymbol{x}^{\circ t} \right\rangle, \tag{4.13}$$

where $\boldsymbol{x}^{\circ p}$ represents the p-times element-wise product of $\boldsymbol{x}$. Hence, the RK feature map for the ANOVA kernel can be written in matrix form:

$$Z_{\mathrm{RK}}(\boldsymbol{x}) = \frac{1}{m\sqrt{D}} \sum_{t=1}^{m} (-1)^{t+1} \boldsymbol{a}^{m-t} \circ (\boldsymbol{\Omega}^{\circ t} \boldsymbol{x}^{\circ t}), \tag{4.14}$$

where $\boldsymbol{\Omega} \coloneqq (\boldsymbol{\omega}_1^\top; \cdots; \boldsymbol{\omega}_D^\top) \in \mathbb{R}^{D \times d}$ is the matrix in which each row is the random vector of the RK map, and $\boldsymbol{a}^t \coloneqq (K_{\mathrm{A}}^t(\boldsymbol{\omega}_1, \boldsymbol{x}), \ldots, K_{\mathrm{A}}^t(\boldsymbol{\omega}_D, \boldsymbol{x}))^\top \in \mathbb{R}^D$ is the vector of the t-order ANOVA kernels (clearly, $\boldsymbol{a}^t$ can be regarded as an RK feature of the t-order ANOVA kernel). Although computing $\boldsymbol{\Omega}^{\circ t}$ in (4.14) seems costly, it is actually trivial when each random vector $\boldsymbol{\omega}_s$ for all $s \in [D]$ is generated from a Rademacher distribution. In this case, $\boldsymbol{\Omega}^{\circ t} = \boldsymbol{\Omega}$ if t is odd; otherwise, it is an all-ones matrix. Therefore, the SCRF technique can be applied to the RK feature map for the ANOVA kernel. Doing this reduces the computational cost of $\boldsymbol{\Omega}^{\circ t} \boldsymbol{x}^{\circ t}$ from $O(Dd)$ to $O(D \log d)$ and thus that of the RK feature map for the m-order ANOVA kernel from $O(mDd)$ time and $O(Dd)$ memory to $O(mD \log d)$ time and $O(D)$ memory. We call a random kernel feature map with the signed circulant random feature a *signed circulant random kernel (SCRK) feature map*.

Although the original SCRF for the RF feature map introduces $\boldsymbol{\sigma}$, resulting in a low variance estimator for the shift-invariant kernel, when order m is even, $\boldsymbol{\sigma}$ is unfortunately meaningless in the proposed RK feature map for the m-order ANOVA kernel case because $K_{\mathrm{A}}^{m}(-\boldsymbol{\omega}, \boldsymbol{x}) = K_{\mathrm{A}}^{m}(\boldsymbol{\omega}, \boldsymbol{x})$. Therefore, the SCRK feature map for an even-order ANOVA kernel may not be effective.

4.3.3 Relationship between FMs and RK Map for ANOVA Kernel

The equation for linear models using the RK feature map for the second-order ANOVA kernel $Z_{\mathrm{RK}}(\boldsymbol{x})$ is:

$$f_{\mathrm{LM}}(Z_{\mathrm{RK}}(\boldsymbol{x}); \boldsymbol{w}) = \frac{1}{\sqrt{D}} \sum_{s=1}^{D} w_s K_{\mathrm{A}}^{2}(\boldsymbol{\omega}_s, \boldsymbol{x}), \tag{4.15}$$

where $\boldsymbol{w} \in \mathbb{R}^D$ is the weight vector for the RK feature map $Z_{\mathrm{RK}}(\boldsymbol{x})$. Hence, linear models using the RK feature map can be regarded as FMs with $\boldsymbol{\lambda} = \boldsymbol{w}/\sqrt{D}$ and only one learnable parameter $\boldsymbol{\lambda}$ and without the linear term. Therefore, theoretical results that guarantee the generalization error of linear models using the RK map can be applied to the theoretical

Algorithm 6 Random Kernel Map for the Weighted Itemset Kernel

Input: $\boldsymbol{x} \in \mathbb{R}^d$, $\mathcal{S} \subseteq 2^{[d]}$, $\{w_V\}_{V \in \mathcal{S}}$

1: **for** $s \leftarrow 1, \ldots, D$ **do**
2: Generate Rademacher vector $\boldsymbol{\omega}_s \in \{-1, +1\}^d$;
3: Compute $Z_s = K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}_s; \{\sqrt{w}_V\}_{V \in \mathcal{S}})$;
4: **end for**

Output: $Z_{\mathrm{RK}}(\boldsymbol{x}; \mathcal{S}, \{w_V\}_{V \in \mathcal{S}}) = (Z_1, \ldots, Z_D)^\top / \sqrt{D}$

analysis of that of FMs. We leave this to future work. The same relationship holds between linear models using the RK feature map for the all-subsets kernel and the all-subsets model. Interestingly, it also holds between linear models using the RM feature map for the polynomial kernel and lifted PNs, which are multi-convex formulation models of PNs [10].

4.4 Extensions of Itemset Kernel

In this section, we extend the itemset kernel (2.22) to the *weighted* itemset kernel and item-*multiset* kernel, and then fix Algorithm 5 for these extensions.

4.4.1 Weighted Itemset Kernel

We first extend the itemset kernel (2.22) to weighted itemset kernel as

$$K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}; \{w_V\}_{V \in \mathcal{S}}) \coloneqq \sum_{V \in \mathcal{S}} w_V \prod_{j \in V} x_j y_j, \tag{4.16}$$

where $w_V \in \mathbb{R}_{\geq 0}$ for all $V \in \mathcal{S}$ is the weight for itemset V. Clearly, (4.16) is equivalent to (2.22) when $w_V = 1$ for all $V \in \mathcal{S}$. Hereinafter, we refer the weighted itemset kernel as the itemset kernel.

Fortunately, Algorithm 5 can be applied to the weighted itemset kernel with a simple modification. Algorithm 6 shows the procedure of the RK map for the weighted itemset kernel.

Proposition 4.8. *Let $Z_{\mathrm{RK}} : \mathbb{R}^d \to \mathbb{R}^D$ be the RK feature map in Algorithm 6. Then, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, $\mathcal{S} \subseteq 2^{[d]}$, and $\{w_V \in \mathbb{R}_{\geq 0}\}_{V \in \mathcal{S}}$*

$$\mathbb{E}_{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D}[\langle Z_{\mathrm{RK}}(\boldsymbol{x}), Z_{\mathrm{RK}}(\boldsymbol{y}) \rangle] = K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}; \{w_V\}_{V \in \mathcal{S}}). \tag{4.17}$$

4.4.2 Item-multiset Kernel and Redundant Feature Augmentation

In this section, we first introduce the *item-multiset kernel*, which is a generalization of the (weighted) itemset kernel. The item-multiset kernel includes not only the itemset kernel but also dot product kernels, so a linear model using a random feature map for the item-multiset kernel can approximate many kernel machines and other related models like FMs. We then introduce *redundant feature augmentation* (RFA), which enables random feature maps to be constructed for the item-multiset kernel.

Algorithm 7 Redundant Feature Augmentation

Input: $\boldsymbol{x} \in \mathbb{R}^d$, $\mathcal{M} \subseteq \mathbb{N}^d$, $\{w_{\boldsymbol{m}}\}_{\boldsymbol{m}\in\mathcal{M}}, w_{\boldsymbol{m}} \in \mathbb{R}_{\geq 0}$;

$\tilde{\mathcal{S}} \leftarrow \{\}$;

$d_j \leftarrow \max_{\boldsymbol{m}\in\mathcal{M}} m_j$ for all $j \in [d]$, $\tilde{d} \leftarrow \sum_{j=1}^d d_j$;

$J_j \leftarrow \{1 + \sum_{i=1}^{j-1} d_i, \dots, d_j + \sum_{i=1}^{j-1} d_i\}$ for all $j \in [d]$;

$\tilde{\boldsymbol{x}} \leftarrow (\underbrace{x_1, \dots, x_1}_{d_1}, \underbrace{x_2, \dots, x_2}_{d_2}, x_3, \dots, \underbrace{x_d, \dots, x_d}_{d_d}) \in \mathbb{R}^{\tilde{d}}$; ▷ $\tilde{x}_k = x_j \ \forall k \in J_j$

for each $\boldsymbol{m} \in \mathcal{M}$ **do** ▷ Converts item-multiset to itemset

  $V \leftarrow \{\}$;

  for $j \leftarrow 1, \dots, d$ **do**

    Pick $k_1, \dots, k_{m_j} \in J_j$ and append them to $V : V \leftarrow V \cup \{k_1, \dots, k_{m_j}\}$;

  end for

  $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}} \cup \{V\}, w_V \leftarrow w_{\boldsymbol{m}}$;

end for

Output: $\tilde{\boldsymbol{x}}, \tilde{\mathcal{S}}, \{w_V\}_{V\in\tilde{\mathcal{S}}}$

Item-multiset Kernel

For given $\mathcal{M} \subseteq \mathbb{N}_{\geq 0}^d$ and $\{w_{\boldsymbol{m}} \in \mathbb{R}_{\geq 0}\}_{\boldsymbol{m}\in\mathcal{M}}$, we define (weighted) item-multiset kernel as

$$K_{\mathcal{M}}^{\text{multi}}(\boldsymbol{x}, \boldsymbol{y}; \{w_{\boldsymbol{m}}\}_{\boldsymbol{m}\in\mathcal{M}}) \coloneqq \sum_{\boldsymbol{m}\in\mathcal{M}} w_{\boldsymbol{m}} \prod_{j=1}^{d} x_j^{m_j} y_j^{m_j}. \tag{4.18}$$

We call a non-negative integer vector $\boldsymbol{m} \in \mathcal{M}$ an item-multiset because it corresponds to the multiset of items (m_j corresponds to the multiplicity of j), and this is the reason we call $K_{\mathcal{M}}^{\text{multi}}$ the item-multiset kernel. The item-multiset kernel is clearly a generalization of the itemset kernel (2.22): the item-multiset kernel is equivalent to the itemset kernel when each $\boldsymbol{m}$ is a binary vector. The item-multiset kernel uses feature combinations that include combinations of the same features, e.g., x_1^2, that are not used in the itemset kernel. The item-multiset kernel thus includes polynomial kernels. Moreover, it includes the dot product kernels: given a dot product kernel $K_{\text{dot}}(\cdot, \cdot; \{a_n\}_{n=0}^{\infty})$, we can represent it as the item-multiset kernel $K_{\mathcal{M}}^{\text{multi}}(\cdot, \cdot; \{w_{\boldsymbol{m}}\}_{\boldsymbol{m}\in\mathcal{M}})$ by taking $\mathcal{M} = \mathbb{N}_{\geq 0}^d$ and $w_{\boldsymbol{m}} = a_n \cdot |\boldsymbol{m}|!/(m_1! \cdots m_d!)$.

Because the item-multiset kernel includes the itemset kernel and dot product kernels, many kernel machines and related models can be approximated by using a random feature map for the item-multiset kernel if there is one. At a glance, the item-multiset kernel can be approximated by the RK map in Algorithm 5 since the item-multiset kernel is closely similar to the itemset kernel. Unfortunately, the RK map cannot approximate the item-multiset kernel. A simple counterexample is a polynomial kernel $\langle \cdot, \cdot \rangle^m$ with $m > 1$. We show the proof of this in the Appendix although it requires only elementary calculation.

4.4.3 Redundant Feature Augmentation

Our basic idea for constructing random feature maps for an item-multiset kernel is (1) converting an item-multiset kernel to an equivalent itemset kernel and then (2) using the RK map for such converted itemset kernel. We here propose a method converting

an item-multiset kernel to an itemset kernel and call it redundant feature augmentation (RFA). The RFA procedure is shown in Algorithm 7. The following proposition states that RFA can convert an item-multiset kernel $K_{\mathcal{M}}^{\text{multi}}(\cdot,\cdot;\{w_{\boldsymbol{m}}\}_{\boldsymbol{m}\in\mathcal{M}})$ on $\mathbb{R}^d$ to an equivalent itemset kernel $K_{\tilde{\mathcal{S}}}(\cdot,\cdot;\{w_V\}_{V\in\tilde{\mathcal{S}}})$ on $\mathbb{R}^{\tilde{d}}$.

Proposition 4.9. *Given an item-multiset kernel $K_{\mathcal{M}}^{\text{multi}}(\cdot,\cdot;\{w_{\boldsymbol{m}}\})$ (i.e., a family of item-multiset $\mathcal{M}$ and weights $\{w_{\boldsymbol{m}}\}$) and feature vectors $\boldsymbol{x},\boldsymbol{y}\in\mathbb{R}^d$, RFA outputs a family of itemsets $\tilde{\mathcal{S}}$ and $\tilde{\boldsymbol{x}}$, $\tilde{\boldsymbol{y}}\in\mathbb{R}^{\tilde{d}}$ such that*

$$K_{\tilde{\mathcal{S}}}(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{y}};\{w_V\}) = K_{\mathcal{M}}^{\text{multi}}(\boldsymbol{x},\boldsymbol{y};\{w_{\boldsymbol{m}}\}). \tag{4.19}$$

Proof. In lines 2-4 (Algorithm 7), RFA converts the input feature vector $\boldsymbol{x}\in\mathbb{R}^d$ to the augmented feature vector $\tilde{\boldsymbol{x}}\in\mathbb{R}^{\tilde{d}}$:

$$\tilde{\boldsymbol{x}} = (\underbrace{x_1,\dots,x_1}_{d_1},\underbrace{x_2,\dots,x_2}_{d_2},\dots,\underbrace{x_d,\dots,x_d}_{d_d}), \tag{4.20}$$

where $d_j=\max_{\boldsymbol{m}\in\mathcal{M}} m_j$ is the maximum multiplicity of j-th feature in $\mathcal{M}$ (obviously, $\sum_{j=1}^d d_j=\tilde{d}$). $J_j=\{1+\sum_{i=1}^{j-1}d_i,\dots,d_j+\sum_{i=1}^{j-1}d_i\}\subset[\tilde{d}]$ in line 3 is the set of indices such that $\tilde{x}_k=x_j,\ \forall k\in J_j$. Then, in lines 5-11, RFA converts each item-multiset $\boldsymbol{m}$ in $\mathcal{M}$ to the itemset V such that

$$\prod_{j\in V}\tilde{x_j}\tilde{y_j}=\prod_{j=1}^{d}(x_jy_j)^{m_j} \tag{4.21}$$

for all $\boldsymbol{x},\boldsymbol{y}\in\mathbb{R}^d$. In line 8, m_j indices $\{k_1,k_2,\dots,k_{m_j}\}$ are chosen from J_j. Because J_j is the set of indices such that $\tilde{x}_k=x_j,\ \forall k\in J_j$, $\prod_{i=1}^{m_j}\tilde{x}_{k_i}\tilde{y}_{k_i}=(x_jy_j)^{m_j}$ holds. Therefore, for the outputs of RFA, Equation (4.19) holds, i.e., RFA converts an item-multiset kernel to an equivalent itemset kernel. □

An example result is shown below.

Example 4.1. Let $\boldsymbol{x}=(x_1,x_2,x_3)^\top\in\mathbb{R}^3$, $\mathcal{M}=\{(1,3,0),(2,2,1),(0,0,4)\}$, and $w_{\boldsymbol{m}}=1$ for all $\boldsymbol{m}\in\mathcal{M}$. Then, since the maximum multiplicity of each feature (i.e., $\max_{\boldsymbol{m}\in\mathcal{M}}m_j$) is 2, 3, and 4, RFA outputs

$$\tilde{\boldsymbol{x}}=(x_1,x_1,x_2,x_2,x_2,x_3,x_3,x_3,x_3)^\top\in\mathbb{R}^9, \tag{4.22}$$

$$\tilde{\mathcal{S}}=\{\{1,3,4,5\},\{1,2,3,4,6\},\{6,7,8,9\}\}, \tag{4.23}$$

$$\{w_V\}_{V\in\tilde{\mathcal{S}}}=\{1,1,1\} \tag{4.24}$$

with $J_1=\{1,2\}$, $J_2=\{3,4,5\}$, and $J=\{6,7,8,9\}$. Clearly, for redundant augmented feature vectors $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ and converted family of itemsets $\tilde{\mathcal{S}}$, $K_{\mathcal{M}}^{\text{multi}}(\boldsymbol{x},\boldsymbol{y})=K_{\tilde{\mathcal{S}}}(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{y}})$ holds.

Since the item-multiset kernel $K_{\mathcal{M}}^{\text{multi}}(\cdot,\cdot;\{w_{\boldsymbol{m}}\}_{\boldsymbol{m}\in\mathcal{M}})$ on $\mathbb{R}^d$ can be converted to an equivalent itemset kernel $K_{\tilde{\mathcal{S}}}(\cdot,\cdot;\{w_V\}_{V\in\tilde{\mathcal{S}}})$ on $\mathbb{R}^{\tilde{d}}$ by using RFA, the RK map with RFA,

$$Z_{\text{RK}}(\boldsymbol{x};\mathcal{M},\{w_{\boldsymbol{m}}\}_{\boldsymbol{m}\in\mathcal{M}})=Z_{\text{RK}}(\tilde{\boldsymbol{x}};\tilde{\mathcal{S}},\{w_V\}_{V\in\mathcal{S}}), \tag{4.25}$$

where $\tilde{\boldsymbol{x}},\tilde{\mathcal{S}}$, and $\{w_V\}_{V\in\mathcal{S}}$ are the outputs of RFA for $\boldsymbol{x}$, $\mathcal{M}$, and $\{w_{\boldsymbol{m}}\}_{\boldsymbol{m}\in\mathcal{M}}$, can approximate the item-multiset kernel $K_{\mathcal{M}}^{\text{multi}}(\cdot,\cdot;\{w_{\boldsymbol{m}}\}_{\boldsymbol{m}\in\mathcal{M}})$. The RK map with RFA procedure

Algorithm 8 Random Kernel Map with Redundant Feature Augmentation

Input: $\boldsymbol{x} \in \mathbb{R}^d$, $\mathcal{M} \subseteq \mathbb{N}^d$, $\{w_{\boldsymbol{m}}\} \in \mathbb{R}^{\mathcal{M}}_{\geq 0}$

1: Compute $\tilde{\boldsymbol{x}}$, $\tilde{\mathcal{S}}$, and $\{w_V\}_{V \in \tilde{\mathcal{S}}}$ by using RFA in Algorithm 7;
2: **for** $s = 1, \ldots, D$ **do** ▷ Applies the RK map to $\tilde{\boldsymbol{x}}$, $\tilde{\mathcal{S}}$, and $\{w_V\}_{V \in \tilde{\mathcal{S}}}$
3: Generate Rademacher vector $\boldsymbol{\omega}_s \in \{-1, +1\}^{\tilde{d}}$;
4: Compute $Z_s = K_{\tilde{\mathcal{S}}}(\tilde{\boldsymbol{x}}; \boldsymbol{\omega}_s, \{\sqrt{w}_V\}_{V \in \tilde{\mathcal{S}}})$;
5: **end for**

Output: $Z_{\mathrm{RK}}(\boldsymbol{x}; \mathcal{M}, \{w_{\boldsymbol{m}}\}_{\boldsymbol{m} \in \mathcal{M}}) = (Z_1, \ldots, Z_D)^\top / \sqrt{D}$

for the item-multiset kernel $K^{\mathrm{multi}}_{\mathcal{M}}(\cdot, \cdot; \{w_{\boldsymbol{m}}\}_{\boldsymbol{m} \in \mathcal{M}})$ is shown in Algorithm 8. In principle, any random feature map for the itemset kernel with RFA can be used for the item-multiset kernel. Therefore, we mainly consider the itemset kernel hereinafter.

Unfortunately, Algorithm 8 cannot be used for all $\mathcal{M} \in \mathbb{N}^d_{\geq 0}$. While the cardinality of the family of itemset $\mathcal{S} \subseteq 2^{[d]}$ is always finite, that of the family of item-multiset $\mathcal{M} \subseteq \mathbb{N}^d$ can be countable. If it is, RFA cannot be applied because it requires explicit computation of a countable infinite-dimensional vector $\tilde{\boldsymbol{x}}$. In Section 4.6, we show the subsampled RK map, which generates sparse random features when the original feature vector $\boldsymbol{x}$ is sparse. Interestingly, the subsampled RK map with RFA solves the issue of not being able to use RFA when $\mathcal{M}$ is countable.

The method used for construction of $\tilde{\mathcal{S}}$ in Algorithm 7 (lines 10–17) is not unique. We introduce another construction method in Section 4.6. The subsampled RK map with RFA based on it includes the RM map as a special case.

4.5 Sparse Random Kernel Map

Although random feature maps overcome the scalability issue of canonical kernel methods, it is hard to use random feature maps for very-large-scale sparse datasets, which can reside in memory due to their sparsity. Most random feature maps generate dense random features and thus cause memory explosion when the dataset is very large and sparse. For example, consider a dataset $\boldsymbol{X}$ with $N = 10^7$, $d = 10^5$, and sparsity of 99.99%. The number of non-zero entries in this dataset is $\mathrm{nnz}(\boldsymbol{X}) = 0.0001 \times ND = 10^8$, so this dataset requires $10^8 \times 64 \times 3$ bit $= 800 \times 3$ MB even if it is represented as a sparse matrix in coordinate format, which represents a sparse matrix as a set of lists of values, row indices, and column indices of the non-zero entries. If a random feature map is applied to this dataset with $D = 2d$, the number of non-zero entries of random feature matrix $\boldsymbol{Z} \in \mathbb{R}^{N \times D}$ is $\mathrm{nnz}(\boldsymbol{Z}) = ND = 2 \times 1/(1 - 0.9999) \times \mathrm{nnz}(\boldsymbol{X})$, so $2 \times 10,000 \times 800$ MB $= 16$ TB memory is required, which would be prohibitive in most cases. One might consider that using a stochastic solver such as stochastic gradient descent [11] can solve this memory issue by not computing random feature maps of all instances before optimization but only computing that of one instance in each optimization iteration. Unfortunately, the computational cost of each optimization iteration is dominated by that of computing a random feature map. Given a random feature vector, the computational cost of each optimization iteration of a stochastic solver is typically $O(D)$. On the other hand, the computational cost of computing a random feature is typically $O(Dd)$. There have been several researches for efficient online kernel learning with random features [16, 47]. They have achieved good prediction performances with small D. However, their computational cost at each

iteration is also dominated by that of computing a random feature map. Although several researchers have proposed structured random feature maps [36, 20, 75], which run more efficiently than non-structured random feature maps (typically $O(D \log d)$ time), their computational costs also dominate that of each stochastic optimization iteration.

In this section, we propose a variant of the RK map. Like the SCRK map, this algorithm is faster and more memory efficient than the RK map. Moreover, this algorithm produces sparse random features when original feature is sparse, i.e., this algorithm is applicable to large-sclae sparse datasets.

Each element of the RK map $Z_{\mathrm{RK}}(\boldsymbol{x}; \mathcal{S}, \{w_V\}_{V \in \mathcal{S}})_s$ is the itemset kernel between $\boldsymbol{x}$ and the sampled random vector $\boldsymbol{\omega}_s$: $K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}_s; \{\sqrt{w}_V\}_{V \in \mathcal{S}})$. From the definition of the itemset kernel in (2.22), $K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}_s) = 0$ if $\mathrm{supp}(\boldsymbol{x}) \cap \mathrm{supp}(\boldsymbol{\omega}_s) = \emptyset$. Therefore, if random basis vectors $\boldsymbol{\omega}_s$ for all $s \in [D]$ are sparse, the RK map generates sparse random features for sparse original feature $\boldsymbol{x}$. Proposition 4.6 states that all distributions with a mean of zero and a variance of one can be used for the RK map. Thus, we propose using the following distribution, which maintains the approximation property of the RK map and generates sparse random features for a sparse original feature vector:

$$\omega = \begin{cases} \frac{-1}{\sqrt{1-p}} & \text{with probability} \quad \frac{1-p}{2}, \\ 0 & \text{with probability} \quad p, \\ \frac{1}{\sqrt{1-p}} & \text{with probability} \quad \frac{1-p}{2}, \end{cases} \tag{4.26}$$

where $p \in [0, 1)$ is the sparsity parameter. We call this distribution a *sparse Rademacher distribution* and call an RK map with a sparse Rademacher distribution a *sparse RK map*. The mean and variance of the sparse Rademacher distribution are clearly zero and one, respectively, so it can be used as the distribution of the random basis vectors for the RK map. The larger p, the sparser the random basis vectors and hence the sparser the random features. Moreover, sparse basis vectors require less memory than dense random basis vectors. Furthermore, $Z_{\mathrm{RK}}(\boldsymbol{x}; \mathcal{S}, \{w_V\}_{V \in \mathcal{S}})_s = K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}_s; \{\sqrt{w}_V\}_{V \in \mathcal{S}})$ can be computed more efficiently when $\boldsymbol{x}$ and/or $\boldsymbol{\omega}_s$ are/is sparse. Therefore, a sparse RK map runs faster and uses less memory than a canonical RK map based on the Rademacher distribution. To be more precise, a sparse RK map requires $O(dD(1-p))$ memory for random basis vectors in expectation. For an m-order ANOVA kernel, a well-known example of the itemset kernel, a sparse RK map runs in $O(mdD(1-p))$ time in expectation.

4.5.1 Efficient Sampling Algorithm from Sparse Rademacher Distribution

Next, we present an efficient algorithm for sampling random basis vectors from a sparse Rademacher distribution. It runs in linear time w.r.t the number of non-zero entries in the sampled basis vectors, $O(\mathrm{nnz}\,(\boldsymbol{\Omega}))$ (i.e, $O(Dd(1-p))$ in expectation) time whereas a naïve algorithm requires $O(Dd)$ time. The procedure of the efficient sampling algorithm is shown in Algorithm 9 [2]. The key components of our algorithm are Walker's alias method [69], which is a sampling method for discrete distributions, and the Fisher-Yates (FY) shuffle algorithm [34], a method for sampling a permutation in linear time. Instead of naïvely

[2]In an actual implementation, $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D$ are represented as a sparse matrix such as in coordinate format, compressed sparse row format, or compressed sparse column format. The sampling algorithm outputs not only the value of the non-zero entries but also several lists of integers that represent the indices of the non-zero entries. The lists created depend on the format of the sparse matrix. We thus simply denote "Output: $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D$" in Algorithm 9.

Algorithm 9 Efficient Sampling Algorithm for Sparse Rademacher Distribution

Input: $p \in [0, 1)$, d, D.

1: Preprocess of Walker's alias method for a binomial distribution $\mathrm{Bin}(1-p, d)$;
2: $a_j \leftarrow j$ for all $j \in [d]$;
3: **for** $s \leftarrow 1, \ldots, D$ **do**
4: Sample the number of non-zero elements d_s in $\boldsymbol{\omega}_s$ from $\mathrm{Bin}((1-p), d)$ by using Walker's alias method; ▷ $O(1)$
5: **for** $i \leftarrow 1, \ldots, d_s$ **do** ▷ Determines indices of non-zero elements;
6: Sample u from $[d-i+1]$ uniformly;
7: Shuffle: $a_i, a_{u+i-1} \leftarrow a_{u+i-1}, a_i$;
8: **end for**
9: Generate $\omega_{s,a_i} \in \{-1, 1\}$ from Rademacher distribution for all $i \in [d_s]$;
10: $\omega_{s,a_i} \leftarrow \omega_{s,a_i}/\sqrt{1-p}$ for all $i \in [d_s]$;
11: **end for**

Output: $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D$ ▷ Regards non-sampled entries as zeros

sampling $\omega_{s,j}$ from a sparse Rademacher distribution Dd times, Algorithm 9 repeats the following for all $s \in [D]$.

1. The number of non-zero elements d_s in $\boldsymbol{\omega}_s$ is sampled from the binomial distribution $\mathrm{Bin}(1-p, d)$ (line 4).

2. The indices of non-zero elements $a_1, \ldots, a_s$ (lines 5-8) are sampled by using a portion of the FY shuffle algorithm. Because the proposed algorithm requires not a permutation of $[d]$ but only a subset of $[d]$ with d_s elements, the procedure from line 5 to line 8 is repeated only d_s times (this procedure is repeated d times in the original FY shuffle algorithm).

3. The values of non-zero elements $\omega_{s,a_1}, \ldots, \omega_{s,a_{d_s}}$ are sampled.

The following proposition guarantees the efficiency and correctness of Algorithm 9.

Proposition 4.10. *Algorithm 9 runs in* $O(\mathrm{nnz}\,(\boldsymbol{\Omega}))$ *and the distribution of its output is the sparse Rademacher distribution* (4.26).

Proof. We first show that Algorithm 9 runs in $O(\mathrm{nnz}\,(\boldsymbol{\Omega}))$. The computational cost of each component in Algorithm 9 are as follows:

- The preprocess for Walker's alias method: $O(d)$ (line 1).
- The preprocess for the FY shuffle: $O(d)$ (line 2).
- Sampling the number of non-zero elements d_s in $\boldsymbol{\omega}_s$: $O(1)$ (line 4).
- Sampling the indices of non-zero elements $a_1, \ldots, a_s$ in $\boldsymbol{\omega}_s$ by using a portion of the FY shuffle algorithm: $O(d_s)$ (lines 5-8).
- Sampling the values of non-zero elements $\omega_{s,a_1}, \ldots, \omega_{s,a_s}$ in $\boldsymbol{\omega}_s$: $O(d_s)$ (line 9 and 10).

Algorithm 10 Subsampled Random Kernel Map

Input: $\boldsymbol{x} \in \mathbb{R}^d$, $\mathcal{S} \subseteq 2^{[d]}$, $\{w_V\}$, $\{\mathcal{S}_\lambda\}_{\lambda\in\Lambda} \subseteq 2^{\mathcal{S}}$, $\{\alpha_\lambda\}_{\lambda\in\Lambda} \in \mathbb{R}_{\geq 0}^{|\Lambda|}$, s.t. $\sum_{\lambda\in\Lambda} \alpha_\lambda K_{\mathcal{S}_\lambda}(\cdot,\cdot) = K_{\mathcal{S}}(\cdot,\cdot)$, and $\{p_\lambda\}_{\lambda\in\Lambda} \in \Delta^{|\Lambda|-1}$ s.t. $p_\lambda > 0\ \forall\lambda \in \Lambda$.

1: **for** $s \leftarrow 1, \ldots, D$ **do**
2: Sample index of sub-family of itemsets: $\lambda_s \in \Lambda$ with probability p_{λ_s};
3: Generate Rademacher vector $\boldsymbol{\omega}_s \in \{-1, 1\}^d$;
4: $Z_s \leftarrow \sqrt{\alpha_{\lambda_s}/p_{\lambda_s}} K_{\mathcal{S}_{\lambda_s}}(\boldsymbol{x}, \boldsymbol{\omega}_s; \{\sqrt{w}_V\}_{V\in\mathcal{S}_{\lambda_s}})$;
5: **end for**

Output: $Z(\boldsymbol{x}) = (Z_1, \ldots, Z_D)^\top/\sqrt{D}$

Therefore, the computational cost of Algorithm 9 is $O(d + \sum_{s=1}^D d_s) = O(\mathrm{nnz}\,(\boldsymbol{\Omega}))$, which is linear w.r.t the number of non-zero elements of the sampled random basis vectors. Next, we show that the distribution of $\omega_{s,j}$ is the sparse Rademacher distribution (4.26). The probability of $\omega_{s,j} \neq 0$ in Algorithm 9 is

$$p(\omega_{s,j} \neq 0) = \sum_{k=1}^{d_s} p(j \text{ is included in } a_1, \ldots, a_{d_s} \mid d_s = k) p(d_s = k) \tag{4.27}$$

$$= \sum_{k=1}^{d} \frac{\binom{d-1}{k-1}}{\binom{d}{k}} \binom{d}{k} p^{d-k}(1-p)^k \tag{4.28}$$

$$= (1-p) \sum_{k'=0}^{d-1} \binom{d-1}{k'} p^{d-1-k'}(1-p)^{k'} = 1 - p. \tag{4.29}$$

Thus, both the probability of $\omega_{s,j} = 1/\sqrt{1-p}$ and $\omega_{s,j} = -1/\sqrt{1-p}$ are $(1-p)/2$. It means that $\omega_{s,j}$ is governed by the sparse Rademacher distribution. The independence of each element can be derived in a similar calculation, so we omit it. □

Note that not only the Rademacher distribution but also other distributions with a mean of zero and a variance of one can be used in Algorithm 9 (line 9 and 10), In other words, although we use a sparse version of the Rademacher distribution, one can use not only a sparse version of the Rademacher distribution but also a sparse version of the standard Gaussian distribution $\mathcal{N}(0,1)$, the uniform distribution $\mathcal{U}(-\sqrt{3}, \sqrt{3})$, the Laplace distribution $\mathrm{Laplace}\left(0, 1/\sqrt{2}\right) = \frac{1}{\sqrt{2}} \exp\left(-\sqrt{2}|\omega|\right)$, etc. This is easily seen from Proposition 4.6. The reason we use a sparse Rademacher distribution is Lemma 4.7: the Rademacher distribution achieves the minimax optimal variance of the approximation error of the RK map among the distributions with a mean of zero and a variance of one.

4.6 Subsampled Random Kernel Map

The next proposed method *subsampled RK map* also generates sparse random features when the original feature vector $\boldsymbol{x}$ is sparse. Moreover, the subsampled RK map solves the issue of RFA described in Section 4.4.3: RFA cannot be used for a countable family of item-multiset $\mathcal{M}$.

The subsampled RK map procedure is shown in Algorithm 10. The subsampled RK map has some hyperparameters: Λ, $\{\mathcal{S}_\lambda\}_{\lambda\in\Lambda} \subseteq 2^{\mathcal{S}}$, $\{p_\lambda > 0\}_{\lambda\in\Lambda}$, and $\{\alpha_\lambda > 0\}_{\lambda\in\Lambda}$. $\{\mathcal{S}_\lambda\}_{\lambda\in\Lambda}$ is a family of $\mathcal{S}$, i.e., a set of subset of $\mathcal{S}$. $\{p_\lambda\}_{\lambda\in\Lambda}$ is a probability distribution on

$\{\mathcal{S}_\lambda\}$. $\{\alpha_\lambda\}_{\lambda\in\Lambda}$ is a set of weights for $\{\mathcal{S}_\lambda\}$, which is introduced to satisfy the requirement of the approximation in (4.1). Λ is an index set and introduced for convenience. Each feature of the canonical RK feature vector is $K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}_s; \{\sqrt{w}_V\}_{V\in\mathcal{S}})$ and it uses all itemsets $\mathcal{S}$ as in (4.7). On the other hand, each feature of the subsampled RK feature vector $Z_{\mathrm{SubRK}}(\boldsymbol{x})_s$ does not use all itemsets $\mathcal{S}$ but use a subset $\mathcal{S}_\lambda$ of $\mathcal{S}$:

$$Z_{\mathrm{SubRK}}(\boldsymbol{x}; \mathcal{S}, \{w_V\}_{V\in\mathcal{S}})_s = \sqrt{\alpha_{\lambda_s}/p_{\lambda_s}} K_{\mathcal{S}_{\lambda_s}}(\boldsymbol{x}, \boldsymbol{\omega}_s; \{\sqrt{w}_V\}_{V\in\mathcal{S}_{\lambda_s}}). \tag{4.30}$$

S_{λ_s} is sampled from $\{\mathcal{S}_\lambda\}$ in accordance with $\{p_\lambda\}$ (in line 2).

The following proposition states that the subsampled RK map approximates the item-multiset kernel.

Proposition 4.11. *For all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, $\mathcal{S} \subseteq 2^{[d]}$, and $\{w_V\}_{V\in\mathcal{S}}$, the subsampled RK map approximates the itemset kernel; i.e.,*

$$\mathbb{E}[\langle Z_{\mathrm{SubRK}}(\boldsymbol{x}), Z_{\mathrm{SubRK}}(\boldsymbol{y})\rangle] = K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}; \{w_V\}) \tag{4.31}$$

holds if the inputs of the subsampled RK map $\{\mathcal{S}_\lambda \subseteq \mathcal{S}\}_{\lambda\in\Lambda}$, $\{\alpha_\lambda \in \mathbb{R}_{\geq 0}\}_{\lambda\in\Lambda} \subseteq$ and $\{p_\lambda\}_{\lambda\in\Lambda} \in \Delta^{|\Lambda|-1}$ satisfy

$$\sum_{\lambda\in\Lambda} \alpha_\lambda K_{\mathcal{S}_\lambda}(\cdot,\cdot) = K_{\mathcal{S}}(\cdot,\cdot),\ p_\lambda > 0\ \forall\lambda \in \Lambda. \tag{4.32}$$

Proof. It is sufficient to prove $\mathbb{E}[Z_{\mathrm{SubRK}}(\boldsymbol{x})_s \cdot Z_{\mathrm{SubRK}}(\boldsymbol{y})_s] = K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y})$:

$$\begin{aligned}
\mathbb{E}[Z_{\mathrm{SubRK}}(\boldsymbol{x})_s \cdot Z_{\mathrm{SubRK}}(\boldsymbol{y})_s] &= \sum_{\lambda\in\Lambda} p_\lambda \mathbb{E}\left[\sqrt{\frac{\alpha_\lambda}{p_\lambda}} K_{\mathcal{S}_\lambda}(\boldsymbol{x}, \boldsymbol{\omega}) \sqrt{\frac{\alpha_\lambda}{p_\lambda}} K_{\mathcal{S}_\lambda}(\boldsymbol{y}, \boldsymbol{\omega})\right] && (4.33)\\
&= \sum_{\lambda\in\Lambda} p_\lambda \frac{\alpha_\lambda}{p_\lambda} \mathbb{E}\left[K_{\mathcal{S}_\lambda}(\boldsymbol{x}, \boldsymbol{\omega}) K_{\mathcal{S}_\lambda}(\boldsymbol{y}, \boldsymbol{\omega})\right] && (4.34)\\
&= \sum_{\lambda\in\Lambda} \alpha_\lambda K_{\mathcal{S}_\lambda}(\boldsymbol{x}, \boldsymbol{y}) = K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}). && (4.35)
\end{aligned}$$

(4.35) follows from the condition (4.32) and the approximation property of the canonical RK map. □

The idea of the subsampled RK map is similar to that of the sparse RK map. Each element in the RK map is an itemset kernel between a feature vector and a sampled random base $\boldsymbol{\omega}_s$, as described above. From the definition of the itemset kernel (2.22), the itemset kernel can be *zero* if the original feature vector $\boldsymbol{x}$ is sparse and the family of itemsets $\mathcal{S}$ is small. For example, $K_{\mathcal{S}'}(\boldsymbol{x}, \cdot) = 0$ if $\mathcal{S}' = \{V : V \in \mathcal{S}, V \ni 1\}$ for all $\boldsymbol{x}$ such that $x_1 = 0$. Therefore, the RK map can be made to generate a sparse random feature vector when the original feature vector is sparse by modifying the algorithm of the RK map: (1) sampling $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \ll |\mathcal{S}|$ from a family of $\mathcal{S}$: $\{\mathcal{S}_\lambda\}_{\lambda\in\Lambda}$ and (2) using sampled $\mathcal{S}'$ instead of all itemsets $\mathcal{S}$.

4.6.1 Choice of Family of $\mathcal{S}$

As described above, the subsampled RK map has some hyperparameters: an index set Λ (it can be considered a subset of $2^{2^{[d]}}$ or $2^{\mathcal{S}}$), a family of $\mathcal{S}$: $\{\mathcal{S}_\lambda\}_{\lambda\in\Lambda}$, probabilities $\{p_\lambda\}_{\lambda\in\Lambda}$, and weights $\{\alpha_\lambda\}_{\lambda\in\Lambda}$. We must specify these hyperparameters such that the

Algorithm 11 Subsampled Random Kernel Map with RFA

Input: $\boldsymbol{x} \in \mathbb{R}^d$, $\mathcal{M} \subseteq \mathbb{N}_{\geq 0}^d$, $\{w_{\boldsymbol{m}}\}$, $\{\mathcal{M}_\lambda\}_{\lambda\in\Lambda} \subseteq 2^{\mathcal{M}}$, $\{\alpha_\lambda\}_{\lambda\in\Lambda} \in \mathbb{R}_{\geq 0}^{|\Lambda|}$, s.t. $\sum_{\lambda\in\Lambda} \alpha_\lambda K_{\mathcal{M}_\lambda}(\cdot,\cdot) = K_{\mathcal{M}}(\cdot,\cdot)$, and $\{p_\lambda\}_{\lambda\in\Lambda} \in \Delta^{|\Lambda|-1}$ s.t. $p_\lambda > 0\ \forall\lambda \in \Lambda$.

1: **for** $s \leftarrow 1, \ldots, D$ **do**
2: Sample index of sub-family of ite-multimsets: $\lambda_s \in \Lambda$ with probability p_{λ_s};
3: Compute $\tilde{\mathcal{S}}_{\lambda_s}, \tilde{\boldsymbol{x}}$, and $\{w_V\}_{V\in\mathcal{S}}$ by using RFA for $\mathcal{M}_{\lambda_s}$ and $\boldsymbol{x}$;
4: Generate Rademacher vector $\boldsymbol{\omega}_s \in \{-1, 1\}^{\tilde{d}}$;
5: $Z_s \leftarrow \sqrt{\alpha_{\lambda_s}/p_{\lambda_s}} K_{\mathcal{S}_{\lambda_s}}(\tilde{\boldsymbol{x}}, \boldsymbol{\omega}_s; \{\sqrt{w}_V\}_{V\in\mathcal{S}_{\lambda_s}})$;
6: **end for**

Output: $Z_{\mathrm{SubRK}}(\boldsymbol{x}; \mathcal{M}, \{w_{\boldsymbol{m}}\}_{\boldsymbol{m}\in\mathcal{M}}) = (Z_1, \ldots, Z_D)^\top/\sqrt{D}$

condition (4.32) is satisfied. Fortunately, it is not hard to construct only valid ones. If $\{S_\lambda\}$ is a *partition* of $\mathcal{S}$ and $\alpha_\lambda = 1$ for all $\lambda \in \Lambda$, the condition (4.32) holds for any $\{p_\lambda > 0\}$. However, the choice of these hyperparameters can greatly affect the sparsity and approximation performances of the subsampled RK map.

From the point of view of sparsity, we propose a family of $\mathcal{S}$ such that the features used in each family of itemsets is restricted:

$$\Lambda = \binom{[d]}{k}, \mathcal{S}_\lambda = \{V \in \mathcal{S} : V \subseteq \lambda\}, \tag{4.36}$$

where $k \in \mathbb{N}_{>0}$ is the number of features used in each small family of itemsets, $\mathcal{S}_\lambda$. In this case, $\mathcal{S}_\lambda \subseteq \mathcal{S}$ is the family of itemsets that use only the features included in λ. The following is an example for the choice of the family of $\mathcal{S}$.

Example 4.2. Let $d = 3$, $\mathcal{S} = \{\{1, 2\}, \{1, 3\}, \{2\}\}$, and $k = 2$. Then,

$$\Lambda = \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}, \{\mathcal{S}_\lambda\}_{\lambda\in\Lambda} = \{\{\{1, 2\}, \{2\}\}, \{\{1, 3\}\}, \{\{2\}\}\}. \tag{4.37}$$

If this construction of $\{\mathcal{S}_\lambda\}_{\lambda\in\Lambda}$ is used for the subsampled RK map with a small k, the subsampled RK map generates sparse random features (when the original feature vector is sparse). However, this construction cannot be used for all $\mathcal{S}$ and k. When $k < \max_{V\in\mathcal{S}} |V|$, there are no valid $\{\mathcal{S}_\lambda\}$ and $\{\alpha_\lambda\}$ because $\bigcup_{\lambda\in\Lambda} \mathcal{S}_\lambda \neq \mathcal{S}$. Moreover, if k is set to a large value for constructing a valid $\{\mathcal{S}_\lambda\}$ and $\{\alpha_\lambda\}$, $|\mathcal{S}_\lambda|$ becomes large, so the random feature vector is not sparse. There are, however, several advantages.

1. This construction method with $k \geq m$ is valid for the m-order ANOVA kernel, which is a well-known example of the itemset kernel.

2. The subsampled RK map with this construction method runs efficiently. Only the ω_j for all $j \in \lambda_s$ need to be sampled because ω_j for all $j \notin \lambda_s$ are not used in $Z_{\mathrm{SubRK}}(\boldsymbol{x})_s = K_{\mathcal{S}_{\lambda_s}}(\boldsymbol{x}, \boldsymbol{\omega}_s)$. To be more precise, the subsampled RK map with this construction method requires not $O(Dd)$ but $O(Dk)$ time and space for sampling $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D$.

4.6.2 Random Feature Maps for Item-multiset Kernel with Countable $\mathcal{M}$

As described in Section 4.4.2, an RK map with RFA can approximate an item-multiset kernel with a finite-cardinality $\mathcal{M}$ but cannot approximate one with a countable $\mathcal{M}$.

Fortunately, this issue can be solved by using a subsampled RK map instead of a canonical RK map. Let us consider the family of $\mathcal{M}$ such that the cardinalities of all elements are finite: $\{\mathcal{M}_\lambda \subseteq \mathcal{M}\}_{\lambda\in\Lambda}$ s.t. $|\mathcal{M}_\lambda| < \infty$ for all $\lambda \in \Lambda$. Then, Λ is a countable set because $\mathcal{M}$ is countable, and the subsets of $\mathbb{N}_{\geq 0}$ with finite cardinality can be numbered as the sum of their elements and dictionary order. Therefore, one can sample the $\mathcal{M}_\lambda$ by giving each λ a distinct non-negative integer n_λ and sampling it with $p_{n_\lambda} = 1/2^{n_\lambda+1}$. Because each $\mathcal{M}_\lambda$ is a finite set, RFA can be used for the $K^{\mathrm{multi}}_{\mathcal{M}_\lambda}$. Thus, the subsampled RK map can approximate the item-multiset kernel with countable $\mathcal{M}$. The procedure of the subsampled RK map with RFA is shown in Algorithm 11.

4.6.3 Relationship between Subsampled Random Kernel Map and Random Maclaurin Map

Finally, we discuss the relationship between the proposed subsampled RK map and RM map. The RM map is a special case of the subsampled RK map with RFA. For a given dot product kernel $K_{\mathrm{dot}}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{n=0}^{\infty} a_n \langle \boldsymbol{x}, \boldsymbol{y}\rangle^n$, there exists a family of item-multisets and their weights s.t. $K_{\mathrm{dot}}(\boldsymbol{x}, \boldsymbol{y}) = K^{\mathrm{multi}}_{\mathcal{M}}(\boldsymbol{x}, \boldsymbol{y}; \{w_{\boldsymbol{m}}\})$, as described in Section 4.4.2. Consider the following hyperparameter setting:

- $\mathcal{M} = \mathbb{N}^d_{\geq 0}, \Lambda = \mathbb{N}_{\geq 0}$,
- $\{\mathcal{M}_n = \{\boldsymbol{m} \in \mathcal{M} = \mathbb{N}^d_{\geq 0} : |\boldsymbol{m}| = n\}\}_{n\in\Lambda}$,
- $\{\alpha_n = a_n\}_{n\in\Lambda}$,
- $\{p_n = 1/2^{n+1}\}$,
- $w_V = 1$ for all $V \in \tilde{\mathcal{S}}_{n_s}$,

where n_s corresponds to λ_s in Algorithm 11. Then, for s-th feature in the output random feature vector, we have

$$Z_{\mathrm{SubRK}}(\boldsymbol{x})_s = \sqrt{\frac{\alpha_{n_s}}{D p_{n_s}}} K_{\tilde{\mathcal{S}}_{n_s}}\left(\tilde{\boldsymbol{x}}, \boldsymbol{\omega}; \{1\}_{V\in\tilde{\mathcal{S}}_{n_s}}\right) \text{ and} \tag{4.38}$$

$$\tilde{\boldsymbol{x}} = (\underbrace{x_1, \ldots, x_1}_{n_s}, \underbrace{x_2, \ldots, x_2}_{n_s}, \ldots, x_d, \ldots, x_d) \in \mathbb{R}^{n_s d} \tag{4.39}$$

since the maximum multiplicity of j-th feature is n_s for all $j \in [d]$. Moreover, we modify the conversion of the family of item-multiset $\mathcal{M}_{n_s}$ into the family of itemset $\tilde{\mathcal{S}}_{n_s}$ in Algorithm 7 as follows:

$$\tilde{\mathcal{S}}_{n_s} \coloneqq \{V \subseteq [n_s d] : |V| = n_s,\ i \not\equiv j \mod n_s \text{ for all } i, j \in V\} \tag{4.40}$$

$$= \prod_{t=1}^{n_s} \{t, t + n_s, \ldots, t + (d-1)n_s\}. \tag{4.41}$$

Clearly, $\left|\tilde{\mathcal{S}}_{n_s}\right| = d^{n_s} = |\tilde{M}_{n_s}|$ and

$$K^{\mathrm{multi}}_{\mathcal{M}_{n_s}}(\boldsymbol{x}, \boldsymbol{y}; \{1_{\boldsymbol{m}}\}_{\boldsymbol{m}\in\mathcal{M}_{n_s}}) = K_{\tilde{\mathcal{S}}_{n_s}}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}; \{1_V\}_{V\in\tilde{\mathcal{S}}_{n_s}}) \tag{4.42}$$

Table 4.1: Datasets used in Section 4.7

Datasets	d	N_{train}	N_{valid}	N_{test}
ML100K	78	$21,200$	$1,000$	$20,202$
phishing	30	$9,000$	555	$1,500$
IJCNN	22	$35,000$	14,900	$91,701$

holds, i.e., the above hyperparameter setting is valid. Moreover, the following holds for $Z_{\text{SubRK}}(\boldsymbol{x})_s$:

$$Z_{\text{SubRK}}(\boldsymbol{x})_s = \sqrt{\frac{\alpha_{n_s}}{Dp_{n_s}}} K_{\tilde{\mathcal{S}}_{n_s}}\left(\tilde{\boldsymbol{x}}, \boldsymbol{\omega}; \{1\}_{V\in\tilde{\mathcal{S}}_{n_s}}\right) = \sqrt{\frac{\alpha_{n_s}}{Dp_{n_s}}}\left(\sum_{V\in\tilde{\mathcal{S}}_{n_s}}\prod_{j\in V}\tilde{x}_j\omega_j\right) \tag{4.43}$$

$$= \sqrt{\frac{\alpha_{n_s}}{Dp_{n_s}}}\prod_{t=1}^{n_s}\langle\tilde{\boldsymbol{x}}[t], \boldsymbol{\omega}[t]\rangle = \sqrt{\frac{\alpha_{n_s}}{Dp_{n_s}}}\prod_{t=1}^{n_s}\langle\boldsymbol{x}, \boldsymbol{\omega}[t]\rangle = Z_{\text{RM}}(\boldsymbol{x})_s, \tag{4.44}$$

where $\boldsymbol{\omega}[t]$ is a d-dimensional sub-vector of $\boldsymbol{\omega}$ such that

$$\boldsymbol{\omega}[t] = (\omega_t, \omega_{n_s+t} \ldots, \omega_{(d-1)n_s+t})^\top \tag{4.45}$$

for all $t \in [n_s]$, and similarly for $\tilde{\boldsymbol{x}}[t]$. That is, the RM map is a special case of the subsampled RK map (with RFA). Clearly, the algorithm of the RM map is much simpler than that of the subsampled RK map for the dot product kernel. Therefore, the RM map can be regarded as a subsampled RK map with techniques that make the algorithm simple. For dot product kernels, sparse random features can be generated by our proposed subsampled RK map with the construction of family of $\mathcal{S}$ in (4.36) while sparse random features are not generated by the RM map, that is, by setting $\Lambda = \mathbb{N}_{\geq 0}$ and $\{\mathcal{M}_n = \{\boldsymbol{m} \in \mathcal{M} : |\boldsymbol{m}| = n\}\}_{n\in\Lambda}$.

4.7 Experiments for RK Map and SCRK Map

In this section, we demonstrate the effectiveness of the RK map and SCRK map.

4.7.1 Datasets

We used three datasets: MovieLens 100K (ML100K) [25], phishing [44] and IJCNN dataset [57]. We normalized each feature vector by their ℓ_1 norm. Table 4.1 shows the overview of datasets.

4.7.2 Accuracy of Approximation

We first evaluated the accuracy of our proposed RK feature map. We calculated the absolute error of the approximation of ANOVA kernels ($m = 2$ or 3) and all-subsets kernel on the training datasets. Each feature vector was normalized by its ℓ_1 norm. Only $10,000$ instances were used. We calculated the mean absolute errors for these instances for 100 trials using Rademacher, Gaussian, Uniform, and Laplace distributions in the RK feature maps and compared the results. For the ANOVA kernels, we also compared them with the SCRK feature map. We varied the dimension of the random features: $2, 4, 8$ and 16

Table 4.2: Absolute errors of RK feature maps for second-order ANOVA kernel, third-order ANOVA kernel, and all-subsets kernel using different distributions for ML100K dataset.

(a) Second-order ANOVA kernel

Method	$D = 2d$	$D = 4d$	$D = 8d$	$D = 16d$
RK (Rademacher)	**6.53e-4 ± 3.86e-5**	**4.62e-4 ± 2.19e-5**	**3.29e-4 ± 1.26e-5**	**2.33e-4 ± 1.02e-5**
RK (Gaussian)	7.31e-4 ± 6.82e-5	5.22e-4 ± 3.71e-5	3.73e-4 ± 1.83e-5	2.62e-4 ± 1.06e-5
RK (Uniform)	6.85e-4 ± 4.96e-5	4.92e-4 ± 2.90e-5	3.50e-4 ± 1.68e-5	2.47e-4 ± 1.05e-5
RK (Laplace)	8.29e-4 ± 1.36e-4	6.16e-4 ± 8.30e-5	4.39e-4 ± 4.03e-5	3.11e-4 ± 2.00e-5
SCRK	7.22e-4 ± 2.13e-4	5.01e-4 ± 9.74e-5	3.60e-4 ± 8.46e-5	2.54e-4 ± 4.34e-5

(b) Third-order ANOVA kernel

Method	$D = 2d$	$D = 4d$	$D = 8d$	$D = 16d$
RK (Rademacher)	**2.26e-5 ± 1.74e-6**	**1.64e-5 ± 8.69e-7**	**1.17e-5 ± 4.70e-7**	**8.35e-6 ± 2.29e-7**
RK (Gaussian)	2.67e-5 ± 3.89e-6	1.97e-5 ± 2.35e-6	1.45e-5 ± 1.17e-6	1.05e-5 ± 6.06e-7
RK (Uniform)	2.40e-5 ± 2.58e-6	1.77e-5 ± 1.46e-6	1.30e-5 ± 8.25e-7	9.27e-6 ± 3.93e-7
RK (Laplace)	3.09e-5 ± 8.56e-6	2.44e-5 ± 5.08e-6	1.80e-5 ± 3.01e-6	1.31e-5 ± 1.54e-6
SCRK	2.29e-5 ± 4.93e-6	1.65e-5 ± 2.28e-6	1.19e-5 ± 1.50e-6	8.40e-6 ± 6.73e-7

(c) All-subsets kernel

Method	$D = 2d$	$D = 4d$	$D = 8d$	$D = 16d$
RK (Rademacher)	4.24e-2 ± 1.14e-2	2.94e-2 ± **7.07e-3**	2.01e-2 ± **4.79e-3**	1.49e-2 ± 4.94e-3
RK (Gaussian)	4.25e-2 ± 1.23e-2	3.07e-2 ± 8.23e-3	2.12e-2 ± 5.29e-3	1.54e-2 ± 4.79e-3
RK (Uniform)	4.32e-2 ± 1.11e-2	2.96e-2 ± 7.61e-3	**1.99e-2** ± 5.05e-3	**1.45e-2 ± 3.93e-3**
RK (Laplace)	**4.15e-2 ± 1.04e-2**	**2.89e-2** ± 7.34e-3	2.00e-2 ± 5.12e-3	1.49e-2 ± 4.17e-3

Table 4.3: Absolute errors of RK feature maps for second-order ANOVA kernel, third-order ANOVA kernel, and all-subsets kernel using different distributions for phishing dataset.

(a) Second-order ANOVA kernel

Method	$D = 2d$	$D = 4d$	$D = 8d$	$D = 16d$
RK (Rademacher)	**9.31e-5 ± 1.89e-5**	**6.71e-5 ± 9.60e-6**	**4.89e-5 ± 8.01e-6**	**3.51e-5 ± 5.11e-6**
RK (Gaussian)	1.03e-4 ± 2.74e-5	7.41e-5 ± 1.70e-5	5.19e-5 ± 8.56e-6	3.84e-5 ± 6.79e-6
RK (Uniform)	9.59e-5 ± 2.05e-5	6.99e-5 ± 1.00e-5	5.09e-5 ± 7.20e-6	3.71e-5 ± 7.07e-6
RK (Laplace)	1.12e-4 ± 4.07e-5	8.01e-5 ± 2.05e-5	5.97e-5 ± 1.59e-5	4.30e-5 ± 7.61e-6
SCRK	1.03e-4 ± 3.73e-5	7.84e-5 ± 2.54e-5	5.42e-5 ± 1.48e-5	3.87e-5 ± 8.71e-6

(b) Third-order ANOVA kernel

Method	$D = 2d$	$D = 4d$	$D = 8d$	$D = 16d$
RK (Rademacher)	**1.10e-6 ± 3.69e-7**	**8.64e-7 ± 1.92e-7**	**6.73e-7 ± 1.32e-7**	**5.18e-7** ± 1.24e-7
RK (Gaussian)	1.26e-6 ± 5.42e-7	9.80e-7 ± 3.25e-7	7.49e-7 ± 1.79e-7	5.76e-7 ± 1.48e-7
RK (Uniform)	1.17e-6 ± 4.50e-7	9.23e-7 ± 2.22e-7	7.23e-7 ± 1.36e-7	5.68e-7 ± 1.64e-7
RK (Laplace)	1.44e-6 ± 8.81e-7	1.13e-6 ± 4.78e-7	9.18e-7 ± 3.87e-7	6.92e-7 ± 2.06e-7
SCRK	1.17e-6 ± 6.43e-7	9.86e-7 ± 3.99e-7	7.28e-7 ± 2.12e-7	5.27e-7 ± **8.97e-8**

(c) All-subsets kernel

Method	$D = 2d$	$D = 4d$	$D = 8d$	$D = 16d$
RK (Rademacher)	3.42e-2 ± 1.47e-2	2.36e-2 ± 9.75e-3	**1.64e-2** ± 6.95e-3	**1.15e-2 ± 4.16e-3**
RK (Gaussian)	3.48e-2 ± 1.33e-2	**2.23e-2 ± 7.85e-3**	1.72e-2 ± 6.29e-3	1.16e-2 ± 4.39e-3
RK (Uniform)	3.22e-2 ± 1.21e-2	2.33e-2 ± 1.00e-2	**1.64e-2** ± 6.34e-3	1.25e-2 ± 4.87e-3
RK (Laplace)	**3.21e-2 ± 1.17e-2**	2.27e-2 ± 9.53e-3	1.65e-2 ± **5.71e-3**	1.28e-2 ± 5.33e-3

Table 4.4: Absolute errors of RK feature maps for second-order ANOVA kernel, third-order ANOVA kernel, and all-subsets kernel using different distributions for IJCNN dataset.

(a) Second-order ANOVA kernel

Method	$D = 2d$	$D = 4d$	$D = 8d$	$D = 16d$
RK (Rademacher)	**1.97e-3 ± 1.85e-4**	**1.39e-3 ± 1.14e-4**	**9.87e-4 ± 9.01e-5**	**6.94e-4 ± 5.65e-5**
RK (Gaussian)	2.67e-3 ± 7.27e-4	1.88e-3 ± 3.77e-4	1.36e-3 ± 1.91e-4	9.68e-4 ± 1.20e-4
RK (Uniform)	2.21e-3 ± 2.79e-4	1.59e-3 ± 1.49e-4	1.13e-3 ± 1.19e-4	7.98e-4 ± 7.30e-5
RK (Laplace)	2.76e-3 ± 1.11e-3	2.21e-3 ± 9.26e-4	1.73e-3 ± 6.67e-4	1.25e-3 ± 3.49e-4
SCRK	2.00e-3 ± 3.37e-4	1.43e-3 ± 2.35e-4	1.02e-3 ± 1.47e-4	7.04e-4 ± 9.42e-5

(b) Third-order ANOVA kernel

Method	$D = 2d$	$D = 4d$	$D = 8d$	$D = 16d$
RK (Rademacher)	**1.57e-5 ± 9.14e-7**	**1.11e-5 ± 4.68e-7**	**7.89e-6 ± 3.12e-7**	**5.57e-6 ± 1.81e-7**
RK (Gaussian)	2.49e-5 ± 8.37e-6	1.89e-5 ± 5.43e-6	1.43e-5 ± 3.72e-6	1.06e-5 ± 1.90e-6
RK (Uniform)	2.13e-5 ± 3.48e-6	1.56e-5 ± 1.93e-6	1.11e-5 ± 1.00e-6	7.93e-6 ± 5.96e-7
RK (Laplace)	2.74e-5 ± 1.70e-5	2.35e-5 ± 1.23e-5	2.01e-5 ± 1.34e-5	1.63e-5 ± 8.08e-6
SCRK	1.59e-5 ± 1.74e-6	1.12e-5 ± 1.04e-6	8.03e-6 ± 6.86e-7	5.64e-6 ± 4.22e-7

(c) All-subsets kernel

Method	$D = 2d$	$D = 4d$	$D = 8d$	$D = 16d$
RK (Rademacher)	1.30e-1 ± 3.98e-2	9.77e-2 ± 3.19e-2	6.94e-2 ± 2.16e-2	4.65e-2 ± 1.13e-2
RK (Gaussian)	1.33e-1 ± **3.12e-2**	9.18e-2 ± 1.99e-2	6.39e-2 ± 1.43e-2	4.57e-2 ± 1.11e-2
RK (Uniform)	**1.29e-1** ± 3.14e-2	**8.87e-2 ± 1.69e-2**	**6.06e-2 ± 1.24e-2**	**4.37e-2 ± 9.43e-3**
RK (Laplace)	1.31e-1 ± 3.47e-2	8.99e-2 ± 1.78e-2	6.13e-2 ± **1.24e-2**	4.44e-2 ± 9.92e-3

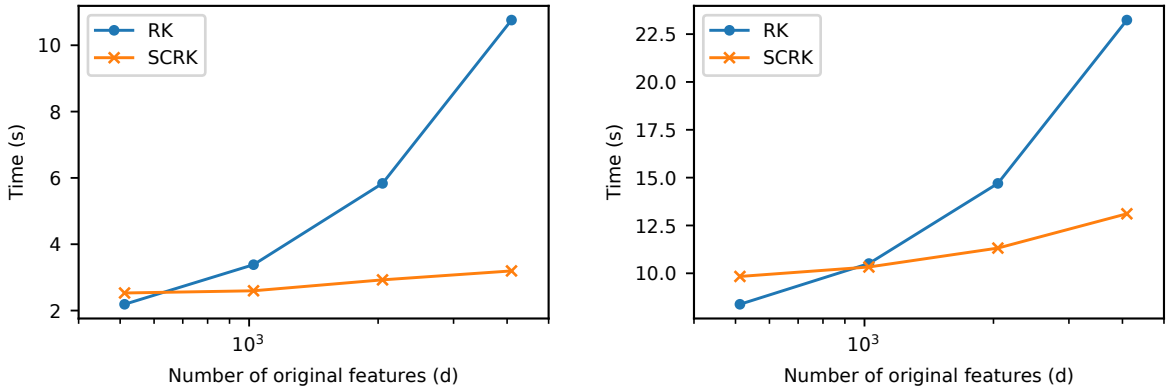


Figure 4.1: Comparisons of mapping times of RK and SCRK feature maps for second-order ANOVA kernel (left) and third-order ANOVA kernel (right) with different dimensions of original feature vector for synthetic dataset (d is shown in log scale).

times that of the original feature vectors. We used Scipy [31] implementations of FFT and IFFT (scipy.fftpack) in the SCRK and TS feature maps.

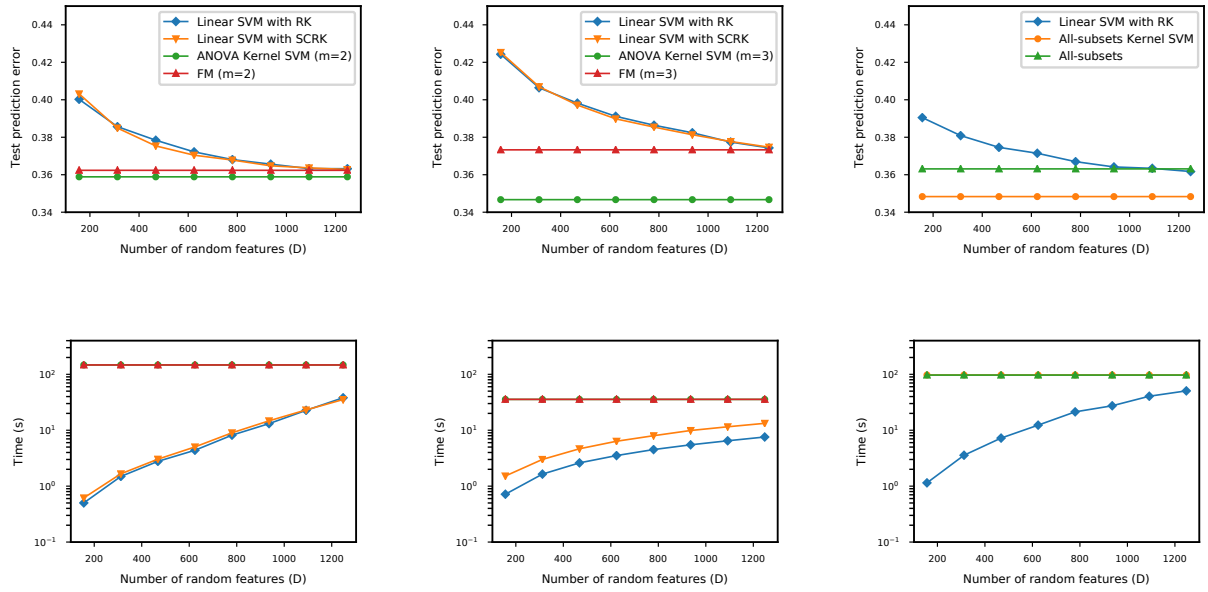
As shown in Section 4.7.2 Table 4.3, and Table 4.4, the RK feature map with the Rademacher distribution had the lowest absolute error and variance for the second- and third-order ANOVA kernels. In contrast, the differences in the absolute errors between the distributions were small for the all-subsets kernel. The variances were large even for $D = 16d$, so the RK feature map for the all-subsets kernel requires a larger D. For the third-order ANOVA kernel, the performance of the SCRK feature map was as good as that of the RK feature map with the Rademacher distribution. However, for the second-order ANOVA kernel, that of the SCRK feature map was not good. As described above, the SCRK feature map is not efficient when order m is even because $\boldsymbol{\sigma}$ is meaningless.

We next evaluated the effectiveness of the SCRK feature map, which is more time and memory efficient than the RK one w.r.t the dimension of the original feature vector. We created synthetic data with various dimensions of the original features and compared the mapping times of the SCRK and RK feature maps for the second-order ANOVA kernel. We used $\mathcal{N}(0, 1)$ as the distribution of original features and changed the dimension of the original features: $d = 512,\ 1,024,\ 2,048$ and $4,096$. We set $D = 8,092$ for all d.

As shown in Fig. 4.1, when the dimension of the original feature vector d was large, the SCRK feature map was more efficient. Although the running time of the RK feature map increased linearly w.r.t d, that of the SCRK feature map increased logarithmically. However, when $d = 512$, the RK feature map was faster than the SCRK feature map. This may be because of the following reasons. First, the difference between d and $\log d$ is small, if d is small. Furthermore, the SCRK feature map requires FFT and IFFT, and hence its dropped constants in Big-O notation are larger than that of the RK feature map.

4.7.3 Performance Comparison on Supervised Learning Setting

We next evaluated the performance of linear models using our proposed RK/SCRK feature maps on the ML100K, phishing, and IJCNN datasets. For the ML100K dataset, We converted the recommender system problem to a binary classification problem. We binarized the original ratings (from 1 to 5) by using 5 as a threshold. We varied the random

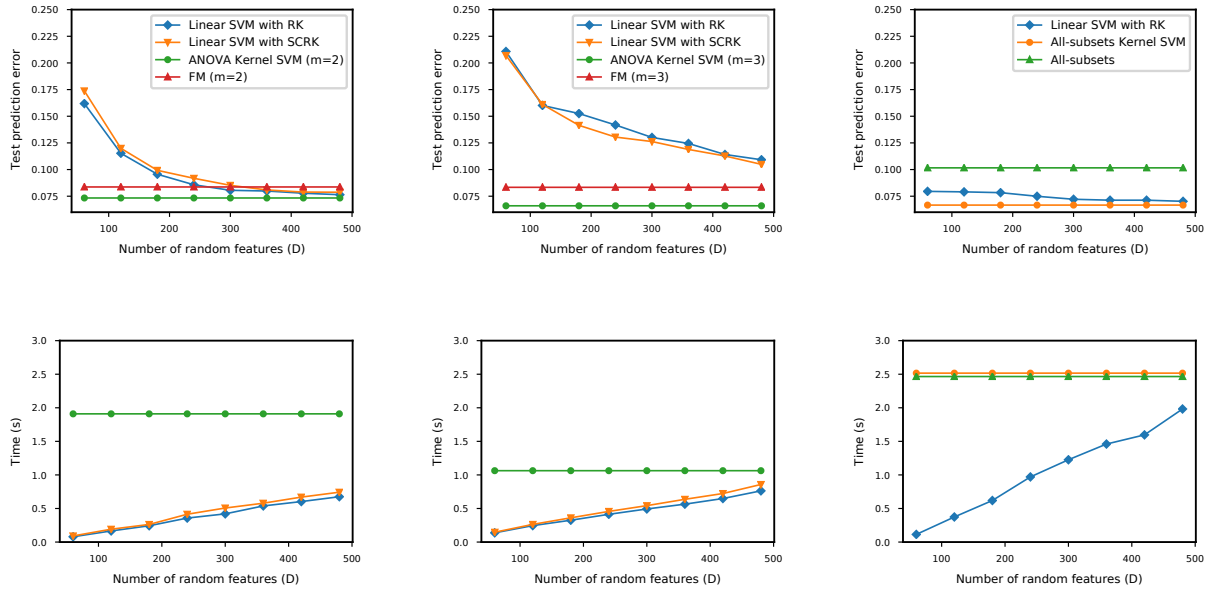


(a) Second-order ANOVA kernel (b) Third-order ANOVA kernel (c) All-subsets kernel

Figure 4.2: Test prediction errors and times for linear SVM using RK feature map approximating (a) second-order ANOVA kernel, (b) third-order ANOVA kernel, and (c) all-subsets kernel and for two existing methods on ML100K dataset. Upper graphs show test prediction errors; lower ones show training and test times (time is shown in log scale).

features dimension in a manner similar to that used in the first evaluation. We compared the prediction errors and learning and testing times for linear SVMs using the proposed RK feature map for the ANOVA/all-subsets kernel, for linear SVMs using the SCRK feature map for the ANOVA kernel, for non-linear SVMs with the ANOVA/all-subsets kernel, and for m-order FMs, and for the all-subsets model. Although there was a linear term in the original FMs, we ignored it for simplicity. All the methods have a regularization hyperparameter, which we set on the basis of the validation test prediction error of the non-linear SVMs. For the linear SVMs using random feature maps, we ran ten trials with a different random seed for each trial and calculated the mean of the values. We used a Rademacher distribution for the random vectors. For the FMs and all-subsets model, we also ran ten trials and calculated the mean of the values. We used coordinate descent [9] as the optimization method. Because this optimization requires many iterations and much time, we ran the optimization process for the same length of time used for the non-linear SVMs. For the rank hyperparameter, we followed Blondel et al. [9] and set it to 30. We used LinearSVC and SVC in scikit-learn [54] as implementations of linear SVMs and non-linear SVMs. LinearSVC used liblinear [18] and SVC used libsvm [12]. For the implementation of FMs, we used FactorizationMachineClassifier in polylearn [48].

As shown in Fig. 4.2, when the number of random features $D = 1,248 = 16d$, the prediction errors of the linear SVMs using the proposed RK feature map were as good as those of the non-linear SVMs, FMs, and all-subsets model. Furthermore, even though $D = 1,248$, their training and testing times were 2–5 times less than those of the non-linear SVMs, FMs, and all-subsets model. Because the dimension of the original feature vector was small, the running times of the linear SVMs using the SCRK feature map were longer than those of the linear SVMs using the RK feature map when $m = 3$. The prediction



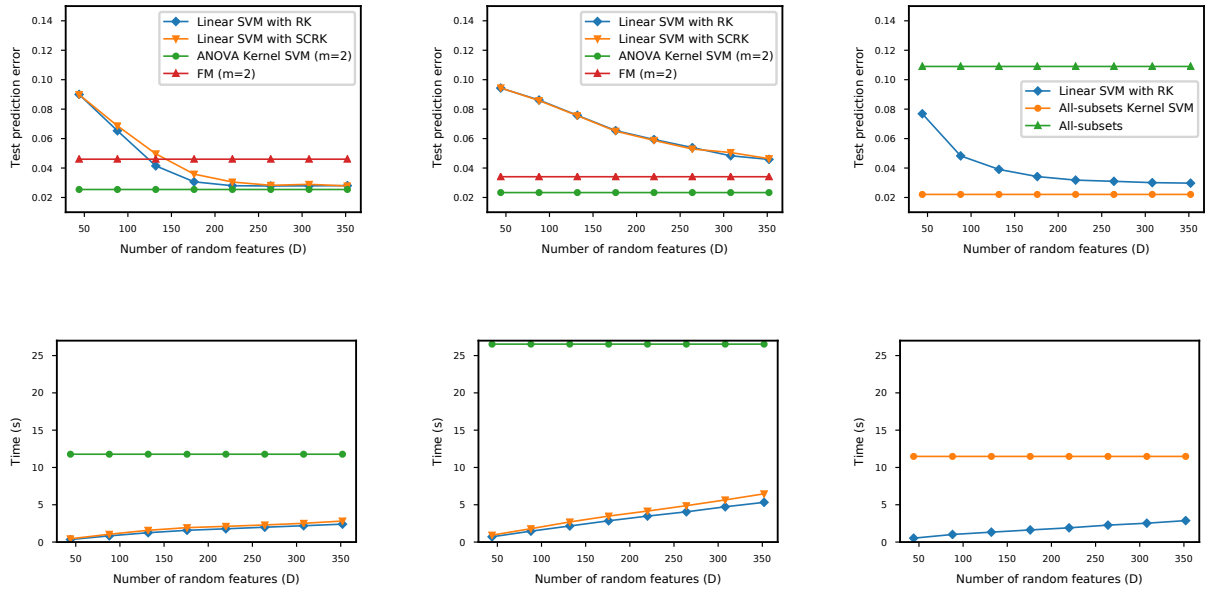
(a) Second-order ANOVA kernel (b) Third-order ANOVA kernel (c) All-subsets kernel

Figure 4.3: Test prediction errors and times for linear SVM with RK feature map approximating (a) second-order ANOVA kernel, (b) third-order ANOVA kernel and (c) all-subsets kernel, kernel SVMs and FMs on phishing dataset. Upper graphs show test prediction errors; lower ones show training and test times.

errors of the linear SVMs using the SCRK feature map were as good as those of the linear SVMs using the RK feature map, and the SCRK feature map required only $O(D \log d)$ time.

We also compared the prediction errors and learning and testing times among random-feature-based methods for the polynomial-like kernel: linear SVMs using the proposed RK/SCRK feature map for the ANOVA kernel, TS feature map, and the RM feature map for the polynomial kernel. Similar to the evaluation above, we set the regularization parameter on the basis of the validation test prediction error of the non-linear SVMs (we also ran the polynomial kernel SVMs). We again ran ten trials with a different random seed for each trial and calculated the mean of the values.

As shown in Fig. 4.5 Fig. 4.6, and Fig. 4.7, when the number of random features D is small, the prediction errors of linear SVMs using the TS/RM feature map were better than those of linear SVMs using the RK feature map. However, when the numbers were larger, the prediction errors of linear SVMs using the RK feature map were as good as those of linear SVMs using the TS feature map. The linear SVMs using the RM feature map achieved the best performance. However, their running times were clearly longer compared to those of the other methods. Moreover, the RM feature map is not space-efficient: it requires $O(Ddm)$ memory for the m-order polynomial kernel while the proposed RK/SCRK feature map for an m-order ANOVA kernel requires only $O(Dd)/O(D)$ memory. The training and testing times of linear SVMs using the RK feature map were the lowest among all methods.



(a) Second-order ANOVA kernel (b) Third-order ANOVA kernel (c) All-subsets kernel

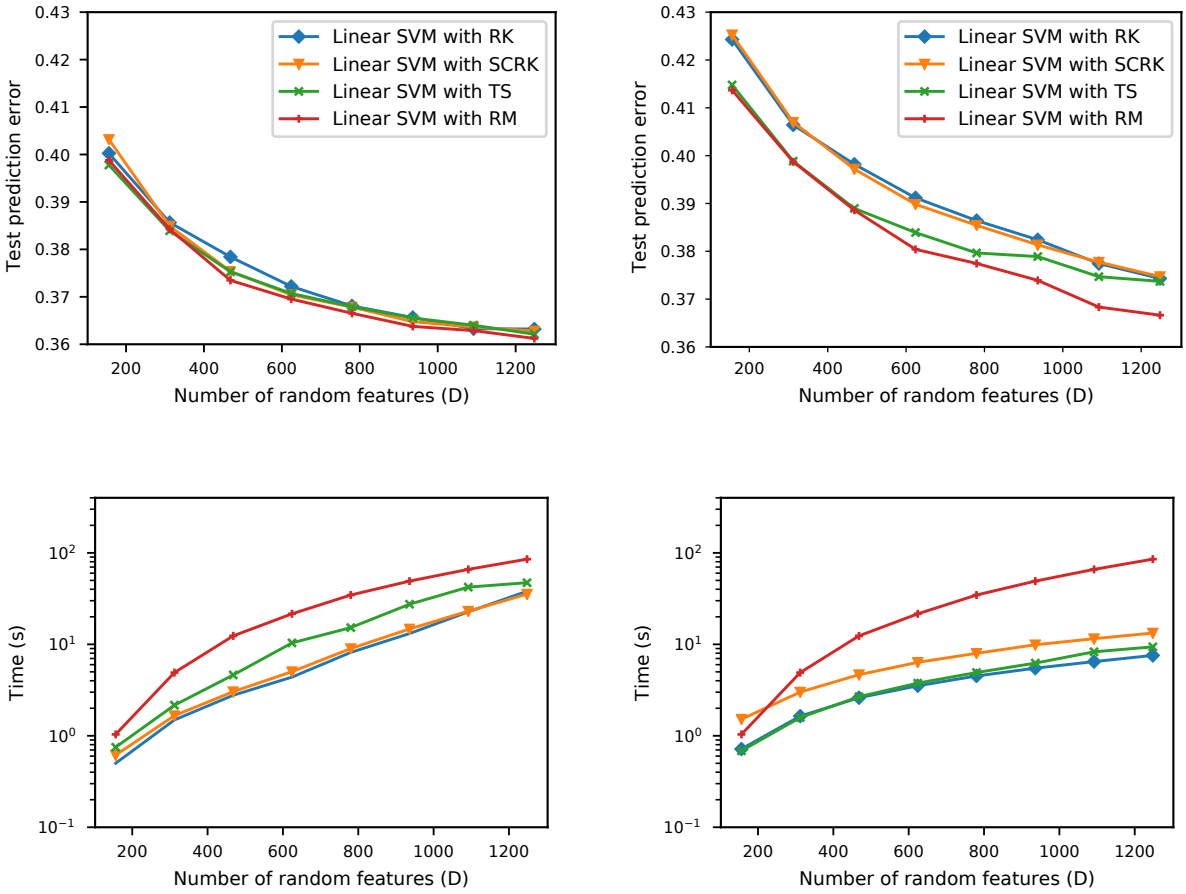
Figure 4.4: Test prediction errors and times for linear SVM with RK feature map approximating (a) second-order ANOVA kernel, (b) third-order ANOVA kernel and (c) all-subsets kernel, ANOVA kernel SVMs, and FMs on IJCNN dataset. Upper graphs show test prediction errors; lower ones show training and test times.

4.8 Experiments for Sparse RK Map and Subsampled RK Map

In this section, we demonstrate the effectiveness of the Sparse RK map and Subsampled map on sparse datasets.

4.8.1 Datasets

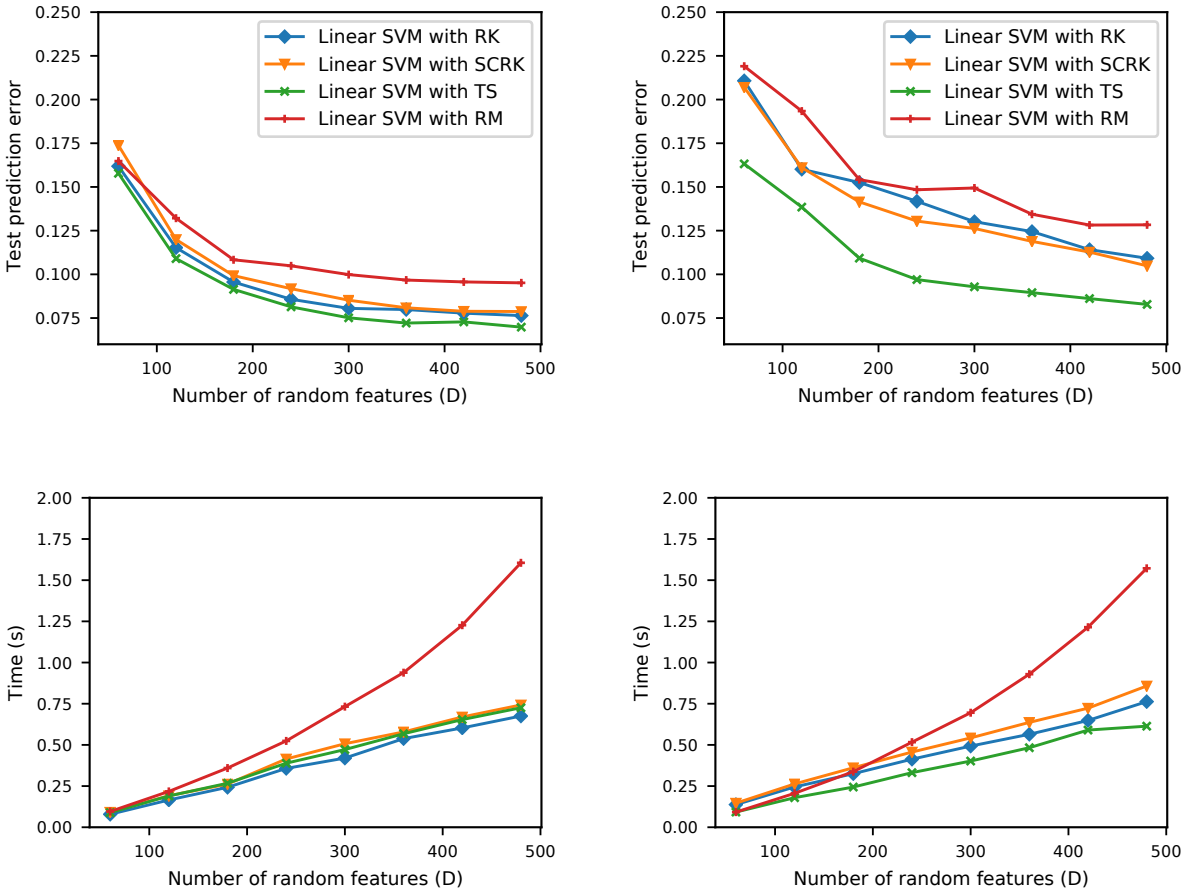
We evaluated the performance of linear models using our sparse/subsampled RK map on the RCV1 dataset and the MovieLens 1M (ML1M) dataset, which are large-scale sparse datasets for the news document classification task and the movie recommendation task, respectively. In particular, we used the RCV1 binary dataset available at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#rcv1.binary`. The number of training, validation, and testing instances were $558,112$, $69,764$, and $69,765$. The density (i.e., $\mathrm{nnz}\left(\boldsymbol{X}\right)/Nd$) of the RCV1 dataset was 1.55×10^{-3}. The number of features (i.e., d) was $47,236$ for the RCV1 dataset. For the ML1M dataset, unlike the ML100K dataset, we used the id of users and movies, occupation, gender, age, zip-code, and genres information as features. We converted the recommendation task to a binary classification problem by binarizing the original ratings (there were from 1 to 5 ratings, and we used 4 as a threshold). The raw dataset was available at `https://grouplens.org/datasets/movielens/1m/`. The dimension of the feature vector was $13,410$ for the ML1M dataset. The number of training, validation, and testing instances were $800,167$, $100,022$, and $100,022$. The density of the ML1M dataset was 6.04×10^{-4}. For both datasets, we scaled all feature vectors by their ℓ_1 norm.



(a) Second-order ANOVA kernel

(b) Third-order ANOVA kernel

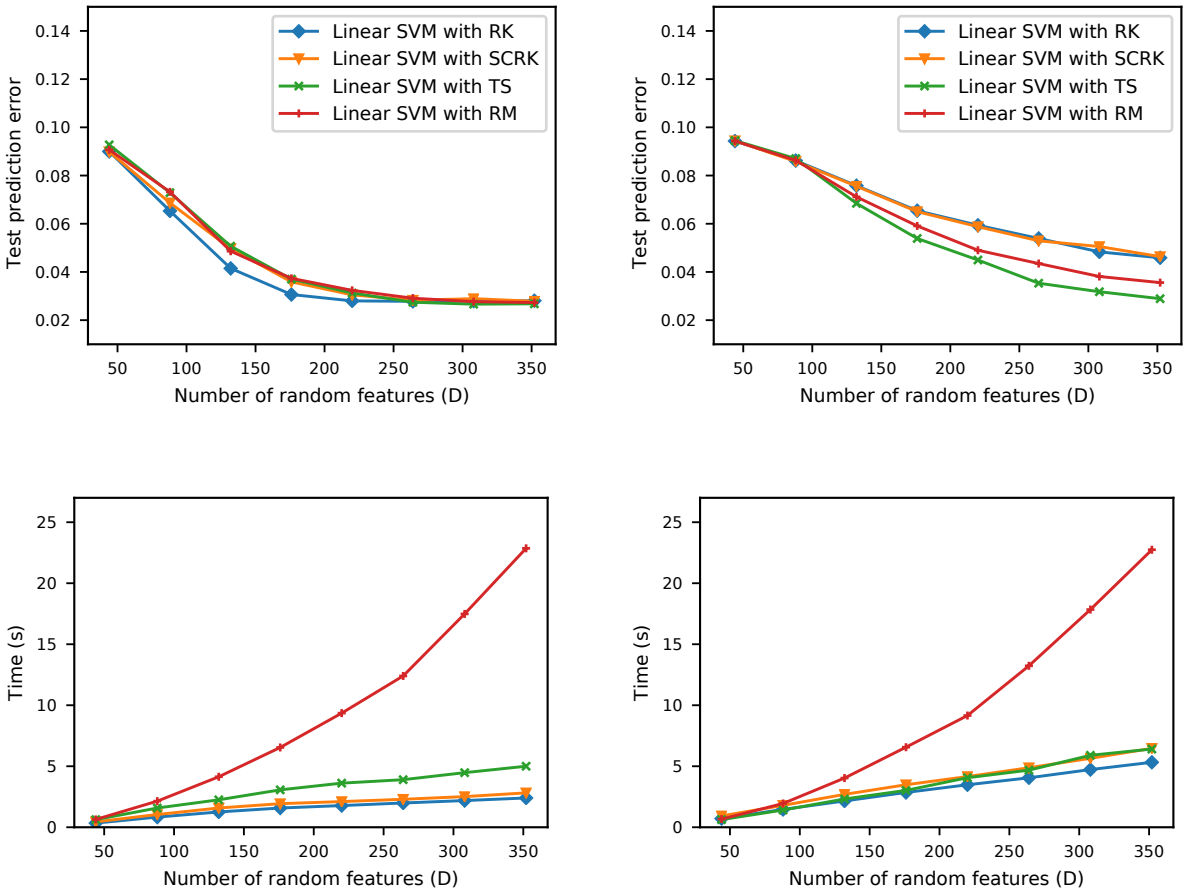
Figure 4.5: Test prediction errors and times for linear SVM with RK/SCRK feature map approximating (a) second-order ANOVA kernel and (b) third-order ANOVA kernel and linear SVM with TS/RM approximating (a) second-order polynomial kernel and (b) third-order polynomial kernel on ML100K dataset. Upper graphs show test prediction errors; lower ones show training and test times.



(a) Second-order ANOVA kernel

(b) Third-order ANOVA kernel

Figure 4.6: Test prediction errors and times for linear SVM with RK/SCRK feature map approximating (a) second-order ANOVA kernel and (b) third-order ANOVA kernel and linear SVM with TS/RM approximating (a) second-order polynomial kernel and (b) third-order polynomial kernel on phishing dataset. Upper graphs show test prediction errors; lower ones show training and test times.



(a) Second-order ANOVA kernel

(b) Third-order ANOVA kernel

Figure 4.7: Test prediction errors and times for linear SVM with RK/SCRK feature map approximating (a) second-order ANOVA kernel and (b) third-order ANOVA kernel and linear SVM with TS/RM approximating (a) second-order polynomial kernel and (b) third-order polynomial kernel on IJCNN dataset. Upper graphs show test prediction errors; lower ones show training and test times.

Table 4.5: Datasets used in Section 4.8.

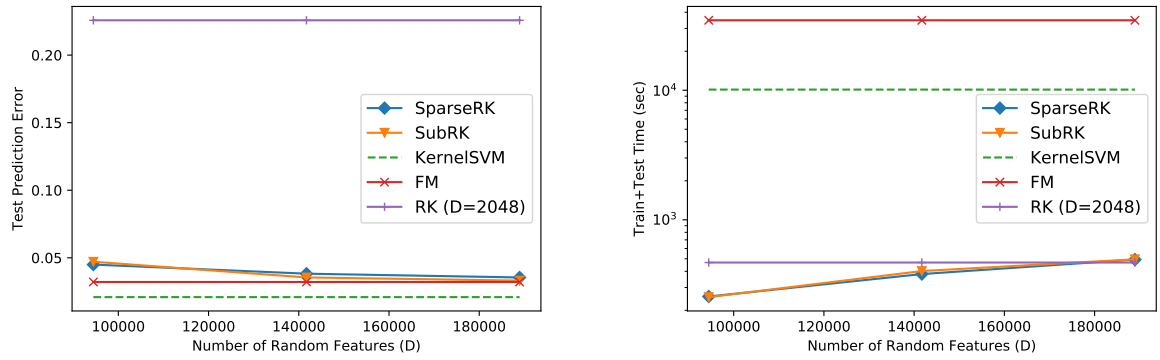
Dataset	d	N_{train}	N_{valid}	N_{test}
RCV1	$47,236$	$558,112$	$69,764$	$69,765$
ML1M	$13,410$	$800,167$	$100,022$	$100,022$

4.8.2 Performance Comparison on Supervised Learning Setting

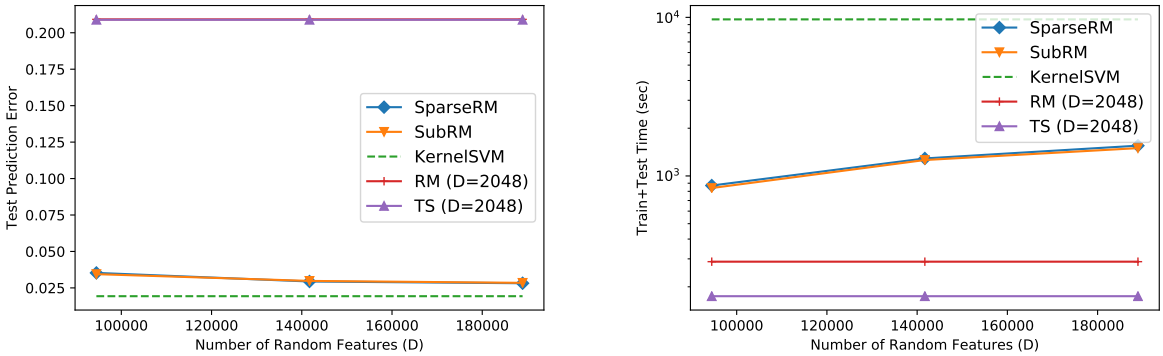
We compared the training times, which included sampling random basis vectors time and transforming data points time for random feature methods, and testing times and the classification error rates for the prediction of testing data for following nine methods:

- **SparseRK**: linear support vector machines (SVMs) using sparse RK map for ANOVA kernel.
- **SubRK**: linear SVMs using subsampled RK map with the construction of family of $\mathcal{S}$ in (4.36) for ANOVA kernel.
- **SparseRM**: linear SVMs using RM map for polynomial kernel with sparse Rademacher distribution.
- **SubRM**: linear SVMs using subsampled RK map with the construction of family of $\mathcal{S}$ in (4.36) for polynomial kernel.
- **RK**: linear SVMs using canonical RK map for ANOVA kernel.
- **RM**: linear SVMs using canonical RM map for polynomial kernel.
- **TS**: linear SVMs using tensor sketching (TS), which is a random feature map for the polynomial kernel [56].
- **KernelSVM**: SVMs with ANOVA or polynomial kernel.
- **FM**: factorization machines without the linear term [61].

We set the order of the ANOVA kernel and the polynomial kernel to 2 for all methods. For **SparseRK**, we set the sparsity parameter p in the sparse Rademacher distribution to 0.999 for the RCV1 dataset, and 0.996 for ML1M dataset. For **SubRK**, we used (4.36) to construct the family of $\mathcal{S}$ and set the sampling probability of each λ as $p_\lambda = 1/|\Lambda|$ for all $\lambda \in \Lambda$. We show the derivation of the valid $\{\alpha_\lambda\}$ in the Appendix. The number of sub-features k was set to $d(1-p)$ for both datasets. For both the **SparseRK** and **SubRK**, we set the the number of random features D to $2d = 94,472$, $3d = 141,708$, and $4d = 188,944$ for the RCV1 dataset, and $2d = 26,820$, $4d = 53,640$, $6d = 80,460$, and $8d = 107,280$ for the ML1M dataset. The settings for **SparseRM** and **SubRM** were similar, respectively, to those of **SparseRK** and **SubRK** described above. For the **SubRM**, we also show the derivation of the valid $\{\alpha_\lambda\}$ in the Appendix. For **RK**, **RM**, and **TS**, we set D to $2,048$ because they generate dense random features for the RCV1 dataset, and $1,048$ for the ML1M dataset. The memories required by the random feature matrix of the canonical random feature methods were almost the same as those required by the random feature methods (**SparseRK**, **SubRK**, **SparseRM**, and **SubRM**). For **FM**, we set the rank-hyperparameter to 30 by following [9]. Table 4.5 shows the summary of the datasets and hyperparameter settings. Similarly in Section 4.7, we used LinearSVC



(a) Second-order ANOVA kernel

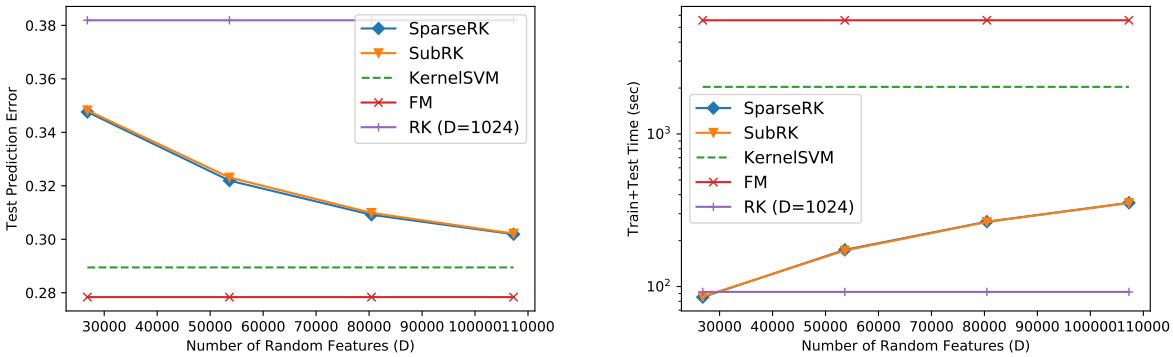


(b) Second-order polynomial kernel

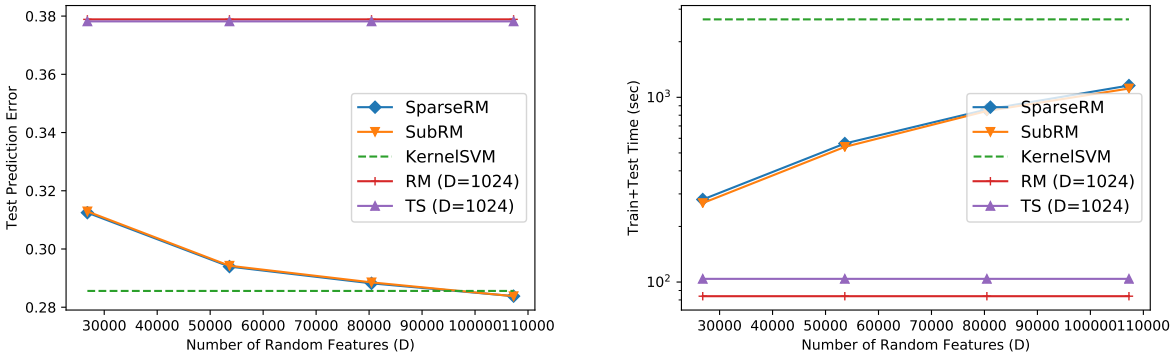
Figure 4.8: Test prediction errors and times for methods using second-order ANOVA kernel (upper) and second-order polynomial kernel (lower) on RCV1 dataset. Left graph shows prediction errors; right one shows sum of training time, which includes sampling random basis vectors time and transforming data points time for random feature method, learning linear model time, and test time (time is shown in log scale).

in scikit-learn [54] as the implementation of the linear SVM for the linear SVMs with random feature maps. For **KernelSVM**, we sampled 160,000 samples from the training instances and used only those instances because using all instances would have required an enormous amount of time for training and testing **KernelSVM**. We used SVC in scikit-learn [54] as the implementation of nonlinear SVMs. We set the size of the cache used in **KernelSVM** to 16 GB. For the FM implementation, we used FactorizationMachineClassifier in polylearn [48]. All the methods have a regularization hyperparameter, which we set on the basis of the validation classification errors of **KernelSVM**. For all methods, we ran the experiment ten times with different random seeds.

The experimental results are shown in Fig. 4.8 for the RCV1 dataset and Fig. 4.9 for the ML1M dataset. We compared the methods among those using a second-order ANOVA kernel and those using a second-order polynomial kernel, respectively. For both the test prediction error and the training and testing time, lower is better. The canonical random feature maps with a small D (**RK**, **RM**, and **TS**) were faster but their performances were worse. Although **KernelSVM** and **FM** performed well, they required a large amount of training and testing time. The proposed sparse random feature methods (**SparseRK**, **SubRK**, **SparseRM**, and **SubRM**) performed as well as **FM**



(a) Second-order ANOVA kernel



(b) Second-order polynomial kernel

Figure 4.9: Test prediction errors and times for methods using second-order ANOVA kernel (upper) and second-order polynomial kernel (lower) on ML1M dataset. Left graph shows prediction errors; right one shows sum of training time, which includes sampling random basis vectors time and transforming data points time for random feature method, learning linear model time, and prediction time (time is shown in log scale).

and **KernelSVM**. Furthermore, their training and testing times were 2–130 times less than those of **FM** and **KernelSVM**. In proposed **SparseRK**, **SubRK**, **SparseRM**, and **SubRM**, the times for training linear models were small; the computation of random features took most of the times. For the ML1M datasets, **FM** achieved the best performance because the ML1M dataset extremely sparse and parameters of FMs can be estimated well for large-scale sparse datasets [61] as described in Section 2.6. Therefore, when the dataset is extremely sparse and long training time is acceptable, using FMs is a good choice. We must note that the hyperparameter tuned on **KernelSVM** was used for the other methods, which further improved the performances of the proposed method.

Next, we investigated the effect of sparsity parameter p (and $k = d(1-p)$) on the performance of the linear model by comparing the proposed methods (i.e., **SparseRK** and **SubRK**) with the canonical method (i.e., **RK**) by setting the same number of random features (D) and changing the sparsity parameters. For this comparison, we used the protein dataset because the **RK** required an enormous amount of memory and time for the RCV1 dataset and the ML1M dataset. The protein dataset is available at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html`. The number of features was 357 and the number of training, validation, and testing instances

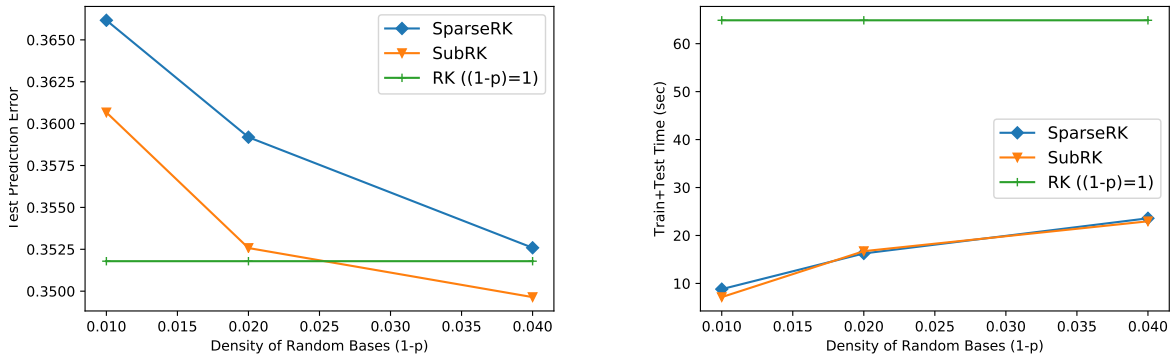


Figure 4.10: Test prediction errors and times for methods using second-order ANOVA kernel on protein dataset. Left graph shows prediction errors; right one shows sum of training time, which includes sampling random basis vectors time and transforming data points time for random feature method, learning linear model time, and test time.

were $14,895$, $2,871$, and $6,621$, respectively. For all methods, we set $D = 16d = 5,712$. For **SparseRK**, we set the sparsity parameter p to 0.99, 0.98, and 0.96. For **SubRK**, we set the number of sub-features k to $d(1-p)$, i.e., 3, 7, and 14. In this experiment, we also ran and evaluated each method ten times with different random seeds.

The experimental results are shown in Fig. 4.10. The x-axes represent $1-p$, i.e., the density of the random basis vectors. When $p = 0.96$ (i.e., $(1-p) = 0.040$), **SparseRK** and **SubRK** performed as well as **RK**. Even when sparsity parameter was high ($p = 0.99$), the differences of the test prediction error between proposed **SparseRK**/**SubRK** and **RK** were small; 0.014 and 0.009, respectively. Moreover, **SparseRK** and **SubRK** ran 3–6 times faster than **RK** while they performed as well as **RK**.

4.9 Conclusion

In this chapter, we have presented a random feature map that approximates the itemset kernel. Although the itemset kernel depends on $\mathcal{S}$, the error bound we have presented does not depend on it or the original dimension d. Moreover, we have showed that the proposed random kernel feature can be used not only with the Rademacher distribution but also with other distributions with zero mean and unit variance. Furthermore, we have showed that the Rademacher distribution achieves the minimax optimal variance both theoretically and empirically. We have also showed how to efficiently compute the random kernel feature map for the ANOVA kernel by using a signed circulant matrix projection technique. Moreover, we have proposed the sparse random kernel map and the subsampled random kernel map, which generate sparse random features and therefore can be applied to a large-scale sparse dataset. In addition, we have extended our methods to the item-multiset kernel, which is a generalization of the itemset kernel. Our evaluation showed that linear models using the proposed random kernel feature map are good alternatives to factorization machines and kernel methods for several classification tasks.

4.10 Proofs

4.10.1 Proof of Proposition 4.2

Proof. Let us consider the product of itemset kernels $K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega})$ and $K_{\mathcal{S}}(\boldsymbol{y}, \boldsymbol{\omega})$, where $\boldsymbol{\omega} \in \{-1, +1\}^d$ is a Rademacher vector:

$$K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}) K_{\mathcal{S}}(\boldsymbol{y}, \boldsymbol{\omega}) = \left(\sum_{V_1 \in \mathcal{S}} \prod_{j_1 \in V_1} x_{j_1} \omega_{j_1} \right) \left(\sum_{V_2 \in \mathcal{S}} \prod_{j_2 \in V_2} y_{j_2} \omega_{j_2} \right) \tag{4.46}$$

$$= \sum_{V_1 \in \mathcal{S}} \sum_{V_2 \in \mathcal{S}} \prod_{j_1 \in V_1} x_{j_1} \omega_{j_1} \prod_{j_2 \in V_2} y_{j_2} \omega_{j_2}. \tag{4.47}$$

Then, $\omega_j^2 = 1$ for all j because $\omega_j \in \{-1, +1\}$. Hence, (4.47) can be rewritten as

$$\sum_{V_1 \in \mathcal{S}} \sum_{V_2 \in \mathcal{S}} \prod_{j_1 \in V_1} x_{j_1} \omega_{j_1} \prod_{j_2 \in V_2} y_{j_2} \omega_{j_2} \tag{4.48}$$

$$= \sum_{V_1 \in \mathcal{S}} \sum_{V_2 \in \mathcal{S}} \prod_{j_4 \in V_1 \cap V_2} \omega_{j_4}^2 \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2} \tag{4.49}$$

$$= \sum_{V_1 \in \mathcal{S}} \sum_{V_2 \in \mathcal{S}} \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2}, \tag{4.50}$$

where $V_1 \triangle V_2$ is the symmetric difference between V_1 and V_2: $V_1 \triangle V_2 = (V_1 \cup V_2) \setminus (V_1 \cap V_2)$. Furthermore, $V_1 \triangle V_2 = \emptyset$ if and only if $V_1 = V_2$. Therefore, one can separate (4.50) as

$$\sum_{V_1 \in \mathcal{S}} \sum_{V_2 \in \mathcal{S}} \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2}$$

$$= \sum_{V_1 = V_2} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2} + \sum_{V_1 \neq V_2} \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2} \tag{4.51}$$

$$= \sum_{S \in \mathcal{S}} \prod_{j \in S} x_j y_j + \sum_{V_1 \neq V_2} \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2}. \tag{4.52}$$

The first term in (4.52) is $K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y})$. The expectation over $\boldsymbol{\omega}$ of the second term is 0 because this term always contains ω_j, and each ω_j is sampled from a Rademacher (fair coin) distribution. Therefore, $\mathbb{E}_{\boldsymbol{\omega}}[K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}) K_{\mathcal{S}}(\boldsymbol{y}, \boldsymbol{\omega})] = K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y})$ and hence

$$\mathbb{E}_{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D}[\langle Z_{\mathrm{RK}}(\boldsymbol{x}), Z_{\mathrm{RK}}(\boldsymbol{y}) \rangle] = \mathbb{E}\left[\sum_{s=1}^{D} \frac{1}{\sqrt{D}} K(\boldsymbol{x}, \boldsymbol{\omega}_s) \frac{1}{\sqrt{D}} K(\boldsymbol{y}, \boldsymbol{\omega}_s) \right] \tag{4.53}$$

$$= K(\boldsymbol{x}, \boldsymbol{y}). \tag{4.54}$$

□

4.10.2 Proofs for Analyses (Section 4.3.1)

Proofs of Lemma 4.3

Proof. The all-subsets kernel $K_{\mathrm{all}}(\cdot, \cdot)$ uses all feature combinations. Hence, $\sup K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}) \leq \sup K_{\mathrm{all}}(\boldsymbol{x}, \boldsymbol{\omega})$ and $\inf K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}) \geq -\sup K_{\mathrm{all}}(\boldsymbol{x}, \boldsymbol{\omega})$ holds for all $\mathcal{S} \subseteq 2^{[d]}$, that is, $|K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega})| \leq \sup K_{\mathrm{all}}(\boldsymbol{x}, \boldsymbol{\omega})$. Therefore, we consider here $\sup K_{\mathrm{all}}(\boldsymbol{x}, \boldsymbol{\omega})$ in order to derive an upper

bound of approximation error. For all $d \geq 1$, $\boldsymbol{x} \in \mathbb{R}^d$ and $\boldsymbol{\omega} \in \{-1, +1\}^d$, the following inequality holds under the assumption that $\|\boldsymbol{x}\|_1 \leq R$:

$$\sup_{\boldsymbol{x},\boldsymbol{\omega}} K_{\mathrm{all}}(\boldsymbol{x}, \boldsymbol{\omega}) = \sup_{\boldsymbol{x},\boldsymbol{\omega}} \prod_{j=1}^{d} (1 + x_j \omega_j) = \sup_{\boldsymbol{x}} \prod_{j=1}^{d} (1 + |x_j|) \tag{4.55}$$

$$= \left(1 + \frac{R}{d}\right)^d < \lim_{d \to \infty} \left(1 + \frac{R}{d}\right)^d = e^R. \tag{4.56}$$

Therefore, $|K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}) K_{\mathcal{S}}(\boldsymbol{y}, \boldsymbol{\omega})| < e^{2R}$, and then (4.9) can be easily obtained from Hoeffding's inequality. □

Proof of Lemma 4.4

Lemma 4.3 gives the absolute error bound of any $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}_1(\boldsymbol{0}, R)$. We show the uniform error bound of the RK map by following the analysis in [32]. We first derive the upper bound of $\|\nabla_{\boldsymbol{x}} \mathcal{E}(\boldsymbol{x}, \boldsymbol{y})\|_2$ and $\|\nabla_{\boldsymbol{y}} \mathcal{E}(\boldsymbol{x}, \boldsymbol{y})\|_2$, that is, the Lipschitz constant of $\mathcal{E}(\cdot, \cdot)$ because their method requires it.

Lemma 4.12. *For all $\mathcal{S} \subseteq 2^{[d]}$,*

$$\sup_{\boldsymbol{x},\boldsymbol{y} \in \mathcal{B}_1(\boldsymbol{0},R)} \|\nabla_{\boldsymbol{x}} \mathcal{E}(\boldsymbol{x}, \boldsymbol{y})\|_2 = \sup_{\boldsymbol{x},\boldsymbol{y} \in \mathcal{B}_1(\boldsymbol{0},R)} \|\nabla_{\boldsymbol{y}} \mathcal{E}(\boldsymbol{x}, \boldsymbol{y})\|_2 \leq \sqrt{d} e^{2R}. \tag{4.57}$$

Proof. From the proof of Proposition 4.2,

$$\mathcal{E}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{D} \sum_{s=1}^{D} \sum_{V_1 \in \mathcal{S}} \sum_{V_2 \neq V_1} \prod_{j_3 \in V_1 \triangle V_2} \omega_{s,j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2}. \tag{4.58}$$

From the triangle inequality,

$$\|\nabla_{\boldsymbol{x}} \mathcal{E}(\boldsymbol{x}, \boldsymbol{y})\|_2 \leq \frac{1}{D} \sum_{s=1}^{D} \left\| \nabla_{\boldsymbol{x}} \sum_{V_1 \in \mathcal{S}} \sum_{V_2 \neq V_1} \prod_{j_3 \in V_1 \triangle V_2} \omega_{s,j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2} \right\|_2. \tag{4.59}$$

Clearly,

$$\sup_{\boldsymbol{x},\boldsymbol{y} \in \mathcal{B}_1(\boldsymbol{0},R)} \|\nabla_{\boldsymbol{x}} \mathcal{E}(\boldsymbol{x}, \boldsymbol{y})\|_2 \leq \sup_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{\omega}} \left\| \nabla_{\boldsymbol{x}} \sum_{V_1 \in \mathcal{S}} \sum_{V_2 \neq V_1} \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2} \right\|_2. \tag{4.60}$$

Because the ℓ_2 norm of $\nabla_{\boldsymbol{x}} \mathcal{E}(\boldsymbol{x}, \boldsymbol{y})$ is $\left(\sum_{j=1}^{d} \{\partial \mathcal{E}(\boldsymbol{x}, \boldsymbol{y}) / \partial x_j\}^2\right)^{\frac{1}{2}}$, let us consider $\partial \mathcal{E}(\boldsymbol{x}, \boldsymbol{y}) / \partial x_j$:

$$\frac{\partial \mathcal{E}(\boldsymbol{x}, \boldsymbol{y})}{\partial x_j} = \partial \left(\sum_{V_1 \in \mathcal{S}} \sum_{V_2 \in \mathcal{S}, V_2 \neq V_1} \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2} \right) / \partial x_j \tag{4.61}$$

$$= \partial \left(x_j \sum_{V_1 \ni j} \sum_{V_2 \neq V_1} \prod_{j_3} \omega_{j_3} \prod_{j_1 \in V_1, j_1 \neq j} x_{j_1} \prod_{j_2 \in V_2} y_{j_2} + \sum_{V_1 \not\ni j} \sum_{V_2 \neq V_1} \prod_{j_3} \omega_{j_3} \prod_{j_1 \in V_1,} x_{j_1} \prod_{j_2 \in V_2} y_{j_2} \right) / \partial x_j \tag{4.62}$$

$$= \sum_{V_1 \ni j} \sum_{V_2 \neq V_1} \prod_{j_3} \omega_{j_3} \prod_{j_1 \in V_1, j_1 \neq j} x_{j_1} \prod_{j_2 \in V_2} y_{j_2}. \tag{4.63}$$

From (4.60), (4.63), and the inequality $K_{\mathcal{S}}(|\boldsymbol{x}|,\mathbf{1}) \leq K_{\mathrm{all}}(|\boldsymbol{x}|,\mathbf{1}) < e^R$, where $\mathbf{1}$ is a d-dimensional vector in which all coordinates are one,

$$\sup_{\boldsymbol{x},\boldsymbol{y}\in\mathcal{B}_1(\mathbf{0},R)} \|\nabla_{\boldsymbol{x}}\mathcal{E}(\boldsymbol{x},\boldsymbol{y})\|_2 \tag{4.64}$$

$$\leq \sup_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{\omega}} \left[\sum_{j=1}^{d}\left\{\sum_{V_1\ni j}\sum_{V_2\neq V_1}\prod_{j_3}\omega_{j_3}\prod_{j_1\in V_1, j_1\neq j}x_{j_1}\prod_{j_2\in V_2}y_{j_2}\right\}^2\right]^{\frac{1}{2}} \tag{4.65}$$

$$\leq \sup_{\boldsymbol{x},\boldsymbol{y}} \left[\sum_{j=1}^{d}\left\{\sum_{V_1\ni j}\sum_{V_2\neq V_1}\prod_{j_1\in V_1, j_1\neq j}|x_{j_1}|\prod_{j_2\in V_2}|y_{j_2}|\right\}^2\right]^{\frac{1}{2}} \tag{4.66}$$

$$\leq \sup_{\boldsymbol{x},\boldsymbol{y}} \left[\sum_{j=1}^{d}\left\{\left(\sum_{V_1\in\mathcal{S}}\prod_{j_1\in V_1}|x_{j_1}|\right)\left(\sum_{V_2\in\mathcal{S}}\prod_{j_2\in V_2}|y_{j_2}|\right)\right\}^2\right]^{\frac{1}{2}} \tag{4.67}$$

$$= \sup_{\boldsymbol{x},\boldsymbol{y}} \left[\sum_{j=1}^{d}\left\{K_{\mathcal{S}}(|\boldsymbol{x}|,\mathbf{1})K_{\mathcal{S}}(|\boldsymbol{y}|,\mathbf{1})\right\}^2\right]^{\frac{1}{2}} \tag{4.68}$$

$$< \left[\sum_{j=1}^{d}\left\{e^R\cdot e^R\right\}^2\right]^{\frac{1}{2}} = \sqrt{d}e^{2R}. \tag{4.69}$$

Similarly, $\sup_{\boldsymbol{x},\boldsymbol{y}\in\mathcal{B}_1(\mathbf{0},R)}\|\nabla_{\boldsymbol{y}}\mathcal{E}(\boldsymbol{x},\boldsymbol{y})\|_2 \leq \sqrt{d}e^{2R}$ can be shown. □

Finally, we show the proof of Lemma 4.4 by using Lemma 4.3, Lemma 4.12, and the results in [32].

Proof. It is well known that a d-dimensional compact set $\mathcal{B}$ can be covered at most $T = (4\mathrm{diam}(\mathcal{B})/r)^d$ balls of radius r by constituting ε-net [15]. We assume that $\boldsymbol{x},\boldsymbol{y} \in \mathcal{B} \subseteq \mathcal{B}_1(\mathbf{0},R) \subseteq \mathcal{B}_2(\mathbf{0},R)$ and hence $\mathrm{diam}(\mathcal{B})$ is at most $2R$. Let $\mathcal{T}$ be the ε-net with radius r (set of the centers of the balls). Then, for all $\boldsymbol{x},\boldsymbol{y}\in\mathcal{B}$, there exists $\boldsymbol{x}'$ and $\boldsymbol{y}'$ that satisfy $|\boldsymbol{x}-\boldsymbol{x}'| \leq r, |\boldsymbol{y}-\boldsymbol{y}'| \leq r$, and

$$\sup_{\substack{\boldsymbol{x}\in\mathcal{B}_2(\boldsymbol{x}',r)\cap\mathcal{B},\\ \boldsymbol{y}\in\mathcal{B}_2(\boldsymbol{y}',r)\cap\mathcal{B}}} |f(\boldsymbol{x},\boldsymbol{y}) - f(\boldsymbol{x}',\boldsymbol{y}')| \leq 2Lr \tag{4.70}$$

for a L-Lipschitz bivariate function $f : \mathcal{B}\times\mathcal{B}\to\mathbb{R}$ (Lemma 5 in [32] (Lemma 9 in its arXiv version)). Furthermore, if $\mathcal{T}$ provides an $(\varepsilon/2)$-close approximation to $K_{\mathcal{S}}$, that is, $\sup_{\boldsymbol{x}',\boldsymbol{y}'\in\mathcal{T}}|\mathcal{E}(\boldsymbol{x}',\boldsymbol{y}')| \leq \varepsilon/2$, applying (4.70) by $f=\mathcal{E}$ derives

$$\sup_{\boldsymbol{x},\boldsymbol{y}\in\mathcal{B}}|\mathcal{E}(\boldsymbol{x},\boldsymbol{y})| = \sup_{\boldsymbol{x},\boldsymbol{y}\in\mathcal{B}}|\mathcal{E}(\boldsymbol{x},\boldsymbol{y}) - \mathcal{E}(\boldsymbol{x}',\boldsymbol{y}') + \mathcal{E}(\boldsymbol{x}',\boldsymbol{y}')| \leq \varepsilon/2 + 2Lr \tag{4.71}$$

because there exists $\boldsymbol{x}',\boldsymbol{y}'\in\mathcal{T}$ such that $\boldsymbol{x}\in\mathcal{B}_2(\boldsymbol{x}',r)$ and $\boldsymbol{y}\in\mathcal{B}_2(\boldsymbol{y}',r)$. Therefore, by choosing $r=\varepsilon/4L$, $\sup_{\boldsymbol{x},\boldsymbol{y}\in\mathcal{B}}|\mathcal{E}(\boldsymbol{x},\boldsymbol{y})| \leq \varepsilon$ if $\sup_{\boldsymbol{x}',\boldsymbol{y}'\in\mathcal{T}}|\mathcal{E}(\boldsymbol{x}',\boldsymbol{y}')| \leq \varepsilon/2$. Then, $|\mathcal{T}| \leq (32RL/\varepsilon)^d$, and following inequality can be obtained from Lemma 4.3 and Lemma 4.12 :

$$p\left(\sup_{\boldsymbol{x},\boldsymbol{y}\in\mathcal{B}}|\mathcal{E}(\boldsymbol{x},\boldsymbol{y})| > \varepsilon\right) \leq p\left(\sup_{\boldsymbol{x}',\boldsymbol{y}'\in\mathcal{T}}|\mathcal{E}(\boldsymbol{x}',\boldsymbol{y}')| > \frac{\varepsilon}{2}\right) \tag{4.72}$$

$$\leq 2\left(\frac{32RL}{\varepsilon}\right)^{2d}\exp\left(-\frac{D\varepsilon^2}{8e^{4R}}\right) = 2\left(\frac{32R\sqrt{d}e^{2R}}{\varepsilon}\right)^{2d}\exp\left(-\frac{D\varepsilon^2}{8e^{4R}}\right). \tag{4.10}$$

□

Proof of Lemma 4.5

Proof. If $|K_{\mathrm{A}}^m(\boldsymbol{x}, \boldsymbol{\omega})| \leq R^m$ this lemma clearly holds from Hoeffding's inequality, similar to Lemma 4.3. For the ANOVA kernel, there is a recursion [9]:

$$K_{\mathrm{A}}^m(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{m} \sum_{t=1}^{m} (-1)^{t+1} K_{\mathrm{A}}^{m-t}(\boldsymbol{x}, \boldsymbol{y}) \mathcal{D}^t(\boldsymbol{x}, \boldsymbol{y}), \tag{4.73}$$

where $\mathcal{D}^t(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \sum_{j=1}^{d} x_j^t y_j^t = \langle \boldsymbol{x}^{\circ t}, \boldsymbol{y}^{\circ t} \rangle$. We prove $|K_{\mathrm{A}}^m(\boldsymbol{x}, \boldsymbol{\omega})| \leq R^m$ by induction based on this recursion. For $m = 1$, $|K_{\mathrm{A}}^1(\boldsymbol{x}, \boldsymbol{\omega})| = |\langle \boldsymbol{x}, \boldsymbol{\omega} \rangle| \leq R$. Now suppose that this inequality is true for $m - 1 \geq 1$. Then,

$$K_{\mathrm{A}}^m(\boldsymbol{x}, \boldsymbol{\omega}) = \frac{1}{m} \sum_{t=1}^{m} (-1)^{t+1} K_{\mathrm{A}}^{m-t}(\boldsymbol{x}, \boldsymbol{\omega}) \mathcal{D}^t(\boldsymbol{x}, \boldsymbol{\omega}) \leq \frac{1}{m} \sum_{t=1}^{m} R^{m-t} \left| \sum_{j=1}^{d} x_j^t \omega_j^t \right| \tag{4.74}$$

$$\leq \frac{1}{m} \sum_{t=1}^{m} R^{m-t} \sum_{j=1}^{d} |x_j|^t \leq \frac{1}{m} \sum_{t=1}^{m} R^{m-t} \|\boldsymbol{x}\|_1^t \leq R^m. \tag{4.75}$$

We can also obtain $-R^m \leq K_{\mathrm{A}}^m(\boldsymbol{x}, \boldsymbol{w})$. Therefore, $|K_{\mathrm{A}}^m(\boldsymbol{x}, \boldsymbol{w}) K_{\mathrm{A}}^m(\boldsymbol{y}, \boldsymbol{w})| \leq R^{2m}$ holds, meaning that Lemma 4.5 can be obtained in a manner similar to that for Lemma 4.3. □

Proof of Proposition 4.6

Proof. From the proof of Proposition 4.2, in the general case

$$K_{\mathcal{S}}(\boldsymbol{\omega}, \boldsymbol{x}) K_{\mathcal{S}}(\boldsymbol{\omega}, \boldsymbol{y}) = \sum_{S \in \mathcal{S}} \prod_{j \in S} \omega_j^2 x_j y_j + \sum_{V_1 \neq V_2} \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2}. \tag{4.76}$$

From (4.76), clearly $\mathbb{E}[K_{\mathcal{S}}(\boldsymbol{\omega}, \boldsymbol{x}) K_{\mathcal{S}}(\boldsymbol{\omega}, \boldsymbol{y})] = K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y})$ for all $\boldsymbol{x}, \boldsymbol{y}$ if and only if $\mathbb{E}[\omega_j] = 0$ and $\mathbb{V}[\omega_j] = \mathbb{E}[\omega_j^2] - \mathbb{E}[\omega_j]^2 = \mathbb{E}[\omega_j^2] = 1$ for all $j \in [d]$. □

Proof of Lemma 4.7

Before proving Lemma 4.7, we present a lemma for the relationship between the second and fourth moments.

Lemma 4.13. *For the second and fourth moments of any probabilistic distribution $p(x)$,*

$$\mathbb{E}[x^4] \geq 2\mathbb{E}[x^2] - 1. \tag{4.77}$$

Proof.

$$\mathbb{E}[x^4] - \mathbb{E}[x^2] = \int_{-\infty}^{\infty} p(x) x^2 (x^2 - 1) dx \tag{4.78}$$

$$= \int_{-\infty}^{-1} p(x) x^2 (x^2 - 1) dx + \int_{-1}^{1} p(x) x^2 (x^2 - 1) dx + \int_{1}^{\infty} p(x) x^2 (x^2 - 1) dx \tag{4.79}$$

$$\geq \int_{-\infty}^{-1} p(x) (x^2 - 1) dx + \int_{-1}^{1} p(x) (x^2 - 1) dx + \int_{1}^{\infty} p(x) (x^2 - 1) dx \tag{4.80}$$

$$= \int_{-\infty}^{\infty} p(x) (x^2 - 1) dx = \mathbb{E}[x^2] - 1. \tag{4.81}$$

□

Finally, we prove Lemma 4.7.

Proof. The variance of the dot product of random kernel maps, $\mathbb{V}[\langle Z_{\mathrm{RK}}(\boldsymbol{x}), Z_{\mathrm{RK}}(\boldsymbol{y})\rangle]$, can be written as

$$\mathbb{V}[\langle Z_{\mathrm{RK}}(\boldsymbol{x}), Z_{\mathrm{RK}}(\boldsymbol{y})\rangle] = \frac{1}{D}\left\{\mathbb{E}\left[\left(K_{\mathcal{S}}(\boldsymbol{\omega}, \boldsymbol{x})K_{\mathcal{S}}(\boldsymbol{\omega}, \boldsymbol{y})\right)^2\right] - K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y})^2\right\}. \tag{4.82}$$

By simple expansion, without loss of generality, $\mathbb{E}\left[\left(K_{\mathcal{S}}(\boldsymbol{\omega}, \boldsymbol{x})K_{\mathcal{S}}(\boldsymbol{\omega}, \boldsymbol{y})\right)^2\right] - K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y})^2$ can be written as

$$\begin{aligned}
&\mathbb{E}\left[\left(K_{\mathcal{S}}(\boldsymbol{\omega}, \boldsymbol{x})K_{\mathcal{S}}(\boldsymbol{\omega}, \boldsymbol{y})\right)^2\right] - K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y})^2 \\
&= \sum_{V_1, V_2, V_3, V_4, V_5, V_6 \in 2^{[d]}} a_{V_1, V_2, V_3, V_4, V_5, V_6} \\
&\qquad \times \prod_{j_1 \in V_1} \mathbb{E}[\omega_{j_1}^4] x_{j_1}^2 y_{j_1}^2 \\
&\qquad \times \prod_{j_2 \in V_2} \mathbb{E}[\omega_{j_2}^3] x_{j_2}^2 y_{j_2} \\
&\qquad \times \prod_{j_3 \in V_3} \mathbb{E}[\omega_{j_3}^3] x_{j_3} y_{j_3}^2 \prod_{j_4 \in V_4} x_{j_4} y_{j_4} \\
&\qquad \times \prod_{j_5 \in V_5} x_{j_5}^2 \prod_{j_6 \in V_6} y_{j_6}^2 \\
&+ \sum_{V_1, V_2 \in \mathcal{S}} \prod_{j \in V_1 \cap V_2} (\mathbb{E}[\omega_{j_1}^4] - 1) x_{j_1}^2 y_{j_1}^2 \prod_{j_2 \in V_1 \triangle V_2} x_{j_2} y_{j_2},
\end{aligned} \tag{4.83}$$

where $a_{V_1, V_2, V_3, V_4, V_5, V_6} \in \mathbb{N}_{\geq 0}$ $(V_1, V_2, V_3, V_4, V_5, V_6 \subseteq 2^{[d]})$ is coefficient that depends on only $\mathcal{S}$. Although $\mathbb{E}[\omega_j^4] x_j^2 y_j^2, x_j^2$ and y_j^2 are always non-negative for all $j \in [d], \boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}_\infty(0, R)$, and $p \in \mathfrak{P}_{0,1}$ (distribution for $\boldsymbol{\omega}$), $\mathbb{E}[\omega_j^3] x_j^2 y_j, \mathbb{E}[\omega_j^3] x_j y_j^2$, and $x_j y_j$ can be negative. This makes it difficult to compare the variances of the dot products of random kernel maps with different distributions for $\boldsymbol{\omega}$.

Fortunately, the maximum variances can be compared relatively easily. For all $\mathcal{S}$, p and $(\boldsymbol{x}^*, \boldsymbol{y}^*) \in \arg\max_{\boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}_\infty(0,R)} \mathbb{V}_{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D \sim p}[\langle Z_{\mathrm{RK}}(\boldsymbol{x}), Z_{\mathrm{RK}}(\boldsymbol{y})\rangle]$, all terms in (4.83) are non-negative. For example, in the case $\mathbb{E}[\omega^3] \geq 0$, $\mathbb{E}[\omega^3] x_j^2 y_j, \mathbb{E}[\omega^3] x_j y_j^2$ and $x_j y_j$ are clearly non-negative if $x_j \geq 0$ and $y_j \geq 0$ for all $j \in [d]$. Similarly, for $\mathbb{E}[\omega^3] \leq 0$, $\mathbb{E}[\omega^3] x_j^2 y_j, \mathbb{E}[\omega^3] x_j y_j^2$ and $x_j y_j$ are non-negative if $x_j \leq 0$ and $y_j \leq 0$ for all $j \in [d]$. Therefore, obviously $x_j^* y_j^* \geq 0$ for all $j \in [d]$. Furthermore, from Lemma 4.13, $\mathbb{E}[\omega^4] - 1 \geq 0$. Therefore, the last term in (4.83) is also non-negative for all $\mathcal{S}$ and p, all $(\boldsymbol{x}^*, \boldsymbol{y}^*)$.

Hence, a distribution $\tilde{p} \in \mathfrak{P}_{0,1}$ with the third and fourth moments of 0 clearly achieves minimax optimal variances, and it is just the Rademacher distribution. □

4.10.3 Proof of Proposition 4.8

Proof. Recall the definition of the weighted itemset kernel:

$$K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}; \{w_V\}_{V \in \mathcal{S}}) = \sum_{V \in \mathcal{S}} w_V \prod_{j \in V} x_j y_j, \tag{2.22}$$

where $\mathcal{S} \subseteq 2^{[d]}$ is the family of itemsets and $w_V \in \mathbb{R}_{\geq 0}$ for all $V \in \mathcal{S}$ is the weight for itemset V.

It is sufficient to prove

$$\mathbb{E}[K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}; \{\sqrt{w_V}\}) \cdot K_{\mathcal{S}}(\boldsymbol{y}, \boldsymbol{\omega}; \{\sqrt{w_V}\})] = K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}; \{w_V\}) \tag{4.84}$$

because $\langle Z_{\mathrm{RK}}(\boldsymbol{x}), Z_{\mathrm{RK}}(\boldsymbol{y})\rangle = \sum_{s=1}^{D} K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}_s; \{\sqrt{w_V}\}) \cdot K_{\mathcal{S}}(\boldsymbol{y}, \boldsymbol{\omega}_s; \{\sqrt{w_V}\})/D$ and $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D$ are sampled independently. The inside term of the expectation in (4.84) is

$$\begin{aligned}
&K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{\omega}; \{\sqrt{w_V}\}) K_{\mathcal{S}}(\boldsymbol{y}, \boldsymbol{\omega}; \{\sqrt{w_V}\}) && (4.85)\\
&= \left(\sum_{V_1 \in \mathcal{S}} \sqrt{w_{V_1}} \prod_{j_1 \in V_1} x_{j_1} \omega_{j_1}\right)\left(\sum_{V_2 \in \mathcal{S}} \sqrt{w_{V_2}} \prod_{j_2 \in V_2} y_{j_2} \omega_{j_2}\right) && (4.86)\\
&= \sum_{V_1 \in \mathcal{S}} \sum_{V_2 \in \mathcal{S}} \sqrt{w_{V_1}} \sqrt{w_{V_2}} \prod_{j_1 \in V_1} x_{j_1} \omega_{j_1} \prod_{j_2 \in V_2} y_{j_2} \omega_{j_2}. && (4.87)
\end{aligned}$$

Then, $\omega_j^2 = 1$ for all j because $\omega_j \in \{-1, +1\}$. Hence, (4.105) can be rewritten as

$$\begin{aligned}
&\sum_{V_1 \in \mathcal{S}} \sum_{V_2 \in \mathcal{S}} \sqrt{w_{V_1}} \sqrt{w_{V_2}} \prod_{j_1 \in V_1} x_{j_1} \omega_{j_1} \prod_{j_2 \in V_2} y_{j_2} && (4.88)\\
&= \sum_{V_1 \in \mathcal{S}} \sum_{V_2 \in \mathcal{S}} \sqrt{w_{V_1}} \sqrt{w_{V_2}} \prod_{j_4 \in V_1 \cap V_2} \omega_{j_4}^2 \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2} && (4.89)\\
&= \sum_{V_1 \in \mathcal{S}} \sum_{V_2 \in \mathcal{S}} \sqrt{w_{V_1}} \sqrt{w_{V_2}} \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2}, && (4.90)
\end{aligned}$$

where $V_1 \triangle V_2$ is the symmetric difference between V_1 and V_2: $V_1 \triangle V_2 = (V_1 \cup V_2) \setminus (V_1 \cap V_2)$. Furthermore, $V_1 \triangle V_2 = \emptyset$ if and only if $V_1 = V_2$. Therefore, one can separate (4.90) as

$$\begin{aligned}
&\sum_{V_1 \in \mathcal{S}} \sum_{V_2 \in \mathcal{S}} \sqrt{w_{V_1}} \sqrt{w_{V_2}} \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2}\\
&= \sum_{V_1 = V_2} \sqrt{w_{V_1}} \sqrt{w_{V_1}} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2}\\
&\quad + \sum_{V_1 \neq V_2} \sqrt{w_{V_1}} \sqrt{w_{V_2}} \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2} && (4.91)\\
&= \sum_{V \in \mathcal{S}} w_V \prod_{j \in S} x_j y_j + \sum_{V_1 \neq V_2} \sqrt{w_{V_1}} \sqrt{w_{V_2}} \prod_{j_3 \in V_1 \triangle V_2} \omega_{j_3} \prod_{j_1 \in V_1} x_{j_1} \prod_{j_2 \in V_2} y_{j_2}. && (4.92)
\end{aligned}$$

The first term in (4.92) is $K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}; \{w_V\})$ and it is surely the weighted itemset kernel. The expectation over the $\boldsymbol{\omega}$ of the second term is 0 because this term always contains ω_j, and each ω_j is sampled from the Rademacher (fair coin) distribution. Therefore, we have (4.84). □

4.10.4 RK Map Cannot Approximate Polynomial Kernels

Although the item-multiset kernel is closely similar to the itemset kernel, the RK map cannot approximate the item-multiset kernel. We show that the RK map cannot approximate the second-order polynomial kernel, which is an example of the item-multiset kernel.

We assume $d > 1$, where d is the dimension of the feature vector. It is sufficient to prove

$$\mathbb{E}\left[K_{\mathcal{M}}^{\mathrm{multi}}(\boldsymbol{x}, \boldsymbol{\omega}; \{\sqrt{w_{\boldsymbol{m}}}\}) \cdot K_{\mathcal{M}}^{\mathrm{multi}}(\boldsymbol{y}, \boldsymbol{\omega}; \{\sqrt{w_{\boldsymbol{m}}}\})\right] \neq K_{\mathcal{M}}^{\mathrm{multi}}(\boldsymbol{x}, \boldsymbol{y}; \{w_{\boldsymbol{m}}\}) \ \exists \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d. \tag{4.93}$$

For the second-order polynomial kernel case, $\mathcal{M} = \{\boldsymbol{m} \in \mathbb{N}_{\geq 0}^d : \|\boldsymbol{m}\|_1 = 2\}$, and $w_{\boldsymbol{m}} = \|\boldsymbol{m}\|_0$. Then, because $\omega_j^2 = 1$ for all $j \in [d]$ (as shown in the above section),

$$K_{\mathcal{M}}^{\text{multi}}(\boldsymbol{x}, \boldsymbol{\omega}; \{\sqrt{w_{\boldsymbol{m}}}\}) = \sum_{j=1}^{d} \omega_j^2 x_j^2 + \sqrt{2} \sum_{j_1 > j_2} \omega_{j_1} \omega_{j_2} x_{j_1} x_{j_2} \tag{4.94}$$

$$= \sum_{j=1}^{d} x_j^2 + \sqrt{2} \sum_{j_1 > j_2} \omega_{j_1} \omega_{j_2} x_{j_1} x_{j_2}, \tag{4.95}$$

and similarly for $K_{\mathcal{M}}^{\text{multi}}(\boldsymbol{y}, \boldsymbol{\omega}; \{\sqrt{w_{\boldsymbol{m}}}\})$. For the product of them, we have

$$\mathbb{E}\left[K_{\mathcal{M}}^{\text{multi}}(\boldsymbol{x}, \boldsymbol{\omega}; \{\sqrt{w_{\boldsymbol{m}}}\}) \cdot K_{\mathcal{M}}^{\text{multi}}(\boldsymbol{y}, \boldsymbol{\omega}; \{\sqrt{w_{\boldsymbol{m}}}\})\right] \tag{4.96}$$

$$= \left(\sum_{j=1}^{d} x_j^2\right)\left(\sum_{j=1}^{d} y_j^2\right) + 2 \sum_{j_1 > j_2} x_{j_1} x_{j_2} y_{j_1} y_{j_2} \tag{4.97}$$

$$= K_{\mathcal{M}}^{\text{multi}}(\boldsymbol{x}, \boldsymbol{y}; \{w_{\boldsymbol{m}}\}) + \sum_{j_1 \neq j_2}^{d} x_{j_1}^2 y_{j_2}^2. \tag{4.98}$$

Therefore, (4.93) holds.

4.10.5 Valid Parameters of Subsampled RK Map

Here we derive the valid parameters of the subsampled RK Map using the construction method of the family of the $\mathcal{S}$ described in (4.36) for the ANOVA kernel and polynomial kernels, which are used in **SubRK** and **SubRM**. Recall the proposed construction of family of $\mathcal{S}$ in (4.36):

$$\Lambda = \binom{[d]}{k}, \mathcal{S}_\lambda = \{V \in \mathcal{S} : V \subseteq \lambda\}, \tag{4.36}$$

where $k \in \{l, \ldots, d\}$ is the number of subfeatures (hyperparameter). The subsampled family of itemsets $\mathcal{S}_\lambda$ is the set of itemsets that use features only in λ. As desribed in Section 5, we must set the $\{\alpha_\lambda\}_{\lambda \in \Lambda}$ such that

$$\sum_{\lambda \in \Lambda} \alpha_\lambda K_{\mathcal{S}_\lambda}(\boldsymbol{x}, \boldsymbol{y}) = K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}). \tag{4.99}$$

ANOVA Kernel Case. We first derive the valid parameters for the n-order ANOVA kernel case. For any $V \in \binom{[d]}{n}$, there exists $\binom{d-n}{k-n}$ index sets that include V, i.e., $|\{\lambda \in \Lambda : \lambda \supseteq V\}| = \binom{d-n}{k-n}$. Therefore, we have

$$\sum_{\lambda \in \Lambda} K_{\mathcal{S}_\lambda}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j_1 < \cdots < j_n} \binom{d-n}{k-n} \prod_{i=1}^{n} x_{j_i} y_{j_i} = \binom{d-l}{k-l} \sum_{j_1 < \cdots < j_n} \prod_{i=1}^{n} x_{j_i} y_{j_i} \tag{4.100}$$

$$= \binom{d-n}{k-n} K_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}), \tag{4.101}$$

and thus by setting $\alpha_\lambda = 1/\binom{d-n}{k-n}$ for all $\lambda \in \Lambda$, the subsampled RK map can approximate the n-order ANOVA kernel.

Polynomial Kernel Case. For **SubRM**, we do not implement the subsampled RK map with RFA in Algorithm 7, naïvely. To derive our **SubRM** algorithm, we first present following lemma.

Lemma 4.14. *For given $n \leq k \leq d \in \mathbb{N}_{>0}$, let $\Lambda = \binom{[d]}{k}$, and $\boldsymbol{\omega}_1, \dots \boldsymbol{\omega}_n \in \{-1, 1\}^d$ are the random vectors sampled from the Rademacher distribution. Then,*

$$\sum_{\lambda_1 \in \Lambda} \cdots \sum_{\lambda_n \in \Lambda} \mathbb{E}_{\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_n} \left[\prod_{t=1}^{n} \left(\sum_{j \in \lambda_t} x_j \omega_{t,j} \right) \left(\sum_{j \in \lambda_t} y_j \omega_{t,j} \right) \right] = \binom{d-1}{k-1}^n \langle \boldsymbol{x}, \boldsymbol{y} \rangle^n . \tag{4.102}$$

Proof. We first fix $\lambda_1, \dots, \lambda_n$. Then, we have

$$\mathbb{E}_{\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_n} \left[\prod_{t=1}^{n} \left(\sum_{j \in \lambda_t} x_j \omega_{t,j} \right) \left(\sum_{j \in \lambda_t} y_j \omega_{t,j} \right) \right] \tag{4.103}$$

$$= \prod_{t=1}^{n} \mathbb{E}_{\boldsymbol{\omega}_t} \left[\left(\sum_{j \in \lambda_t} x_j \omega_{t,j} \right) \left(\sum_{j \in \lambda_t} y_j \omega_{t,j} \right) \right] \tag{4.104}$$

$$= \prod_{t=1}^{n} \left(\sum_{j \in \lambda_t} x_j y_j \right) . \tag{4.105}$$

Next, we consider the summation of (4.105) with respect to the $\lambda_1, \dots, \lambda_n$. Let $S_{\lambda_t} \coloneqq \sum_{j \in \lambda_t} x_j y_j$. For any $j \in [d]$, there exists $\binom{d-1}{k-1}$ itemsets that include $j \in [d]$ in $\Lambda = \binom{[d]}{k}$, i.e., $|\{V : V \in \Lambda, j \ni V\}| = \binom{d-1}{k-1}$. Therefore, $\sum_{\lambda_t \in \Lambda} S_{\lambda_t} = \binom{d-1}{k-1} \sum_{j=1}^{d} x_j y_j$ and we have

$$\sum_{\lambda_1 \in \Lambda} \cdots \sum_{\lambda_n \in \Lambda} \prod_{t=1}^{n} \left(\sum_{j \in \lambda_t} x_j y_j \right) = \sum_{\lambda_1 \in \Lambda} \cdots \sum_{\lambda_n \in \Lambda} \prod_{t=1}^{n} S_{\lambda_t} \tag{4.106}$$

$$= \prod_{t=1}^{n} \left(\sum_{\lambda_t \in \Lambda} S_{\lambda_t} \right) = \prod_{t=1}^{n} \left(\binom{d-1}{k-1} \sum_{j=1}^{d} x_j y_j \right) \tag{4.107}$$

$$= \binom{d-1}{k-1}^n \langle \boldsymbol{x}, \boldsymbol{y} \rangle^n . \tag{4.108}$$

□

From this Lemma 4.14, we propose Algorithm 12, which approximates n-order polynomial kernels. This algorithm can be regarded as the subsampled RM map with the construction in (4.36). In Algorithm 12, $\alpha = \left(1/\binom{d-1}{k-1}\right)^n$ corresponds to α_λ and $p = \left(1/\binom{d}{k}\right)^n$ corresponds to p_λ in the subsampling RK map (Algorithm 6/7). We use it as the **SubRM** in our experiments.

Algorithm 12 Subsampled Random Maclaurin Map for n-order Polynomial Kernel

Input: $\boldsymbol{x} \in \mathbb{R}^d$, $k \in \mathbb{N}_{\geq n}$

$p \leftarrow \left(1/\binom{d}{k}\right)^n$;

$\alpha \leftarrow \left(1/\binom{d-1}{k-1}\right)^n$;

$c \leftarrow \alpha/p = (d/k)^n$;

for $s = 1, \ldots, D$ **do**

    Generate n Rademacher vectors $\boldsymbol{\omega}_{s,1}, \ldots, \boldsymbol{\omega}_{s,n} \in \{-1, +1\}^d$;

    Generate n itemsets $\lambda_{s,1}, \ldots, \lambda_{s,n} \in \binom{[d]}{n}$ uniformly and independently;

    Compute $Z_s = \sqrt{c} \prod_{t=1}^{n} \sum_{j \in \lambda_{s,t}} \omega_{s,j}, x_j$;

end for

Output: $Z_{\mathrm{RM}}(\boldsymbol{x}) = (Z_1, \ldots, Z_D)^\top / \sqrt{D}$

Chapter 5

Conclusion

In this doctoral dissertation, we have studied machine learning algorithms using feature interactions. Especially, we have developed models using higher-order feature interactions across objects for feature-based link prediction and efficient learning methods for predictive models based on feature interactions.

For feature-based link prediction, we have presented models based on higher-order feature combinations only across the two objects being compared in Chapter 3. Our proposed model, HOPairNet, can be regarded as a higher-order generalization of the factorized bilinear model or pairwise extension of the higher-order factorization machine. We have also presented an algorithm for efficiently computing higher-order feature combinations only across two objects. Moreover, we have proposed an efficient CD algorithm for the proposed model. Furthermore, we have proposed the HOPairDNN, which is a DNN-extension of the HOPairNet. In addition, we have also presented the relationships among proposed methods, existing methods for feature-based link prediction, and for index-based link prediction.

We have tackled the scalability issue of kernel methods in Chapter 4. We have presented a random feature map, random kernel map, that approximates the itemset kernel as well as some theoretical analyses on the proposed method. By using the proposed method, machine learning users can apply the kernel machines using feature interactions to a large-scale dataset. We have also shown how to efficiently compute the random kernel feature map for the ANOVA kernel by using a signed circulant matrix projection technique. Moreover, we have proposed the sparse random kernel map and the subsampled random kernel map, which generate sparse random features and therefore can be applied to a large-scale sparse dataset. These methods are faster and more memory efficient than the canonical random kernel map and useful for a large-scale sparse dataset since they use a sparse random feature matrix and can generate sparse random features. In addition, we have extended our methods to the item-multiset kernel, which is a generalization of the itemset kernel.

In future work, I plan to develop a method to make predictive models using feature interactions more interpretable. To tell the truth, I have already developed a feature interaction selection method of FMs that can improve the interpretability of FMs. For more details, please see our preprint [3].

Bibliography

[1] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018.

[2] Kyohei Atarashi, Satoshi Oyama, Masahito Kurihara, and Kazune Furudo. A deep neural network for pairwise classification: Enabling feature conjunctions and ensuring symmetry. In *PAKDD*, pages 83–95, 2017.

[3] Kyohei Atarashi, Satoshi Oyama, and Masahito Kurihara. Factorization machines with regularization for sparse feature interactions. *arXiv preprint arXiv:2010.09225*, 2020.

[4] Kyohei Atarashi, Satoshi Oyama, and Masahito Kurihara. Link prediction using higher-order feature combinations across objects. *IEICE Transactions on Information and Systems*, E103.D(8):1833–1842, 2020.

[5] Asa Ben-Hur and William Stafford Noble. Kernel methods for predicting protein–protein interactions. *Bioinformatics*, 21(suppl 1):i38–i46, 2005.

[6] Christopher Bishop. *Patern Recognition and Machine Learning*. Springer, 2006.

[7] Christopher Bishop. *Patern Recognition and Machine Learning*, pages 261–267. Springer, 2006.

[8] Mathieu Blondel, Akinori Fujino, and Naonori Ueda. Convex factorization machines. In *ECML-PKDD*, pages 19–35. Springer, 2015.

[9] Mathieu Blondel, Akinori Fujino, Naonori Ueda, and Masakazu Ishihata. Higher-order factorization machines. In *NeurIPS*, pages 3351–3359, 2016.

[10] Mathieu Blondel, Masakazu Ishihata, Akinori Fujino, and Naonori Ueda. Polynomial networks and factorization machines: New insights and efficient training algorithms. In *ICML*, pages 850–858, 2016.

[11] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[12] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27, 2011.

[13] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *ICALP*, pages 693–703. Springer, 2002.

[14] Weiyu Cheng, Yanyan Shen, and Linpeng Huang. Adaptive factorization network: Learning adaptive-order feature interactions. In *AAAI*, 2020.

[15] Felipe Cucker and Steve Smale. On the mathematical foundations of learning. *Bulletin of the American Mathematical Society*, 39(1):1–49, 2002.

[16] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina F Balcan, and Le Song. Scalable kernel methods via doubly stochastic gradients. In *NeurIPS*, pages 3041–3049, 2014.

[17] Arthur Gretton Dino Sejdinovic. What is an rkhs?, 2014.

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

[19] Maryam Fazel. *Matrix rank minimization with applications*. PhD thesis, PhD thesis, Stanford University, 2002.

[20] Chang Feng, Qinghua Hu, and Shizhong Liao. Random feature mapping with signed circulant matrix projection. In *IJCAI*, pages 3490–3496, 2015.

[21] Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. In *EMNLP*, pages 457–468, 2016.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, pp.443–485, 2016. URL `http://www.deeplearningbook.org`.

[23] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016.

[24] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. In *IJCAI*, pages 1725–1731, 2017.

[25] F Maxwell Harper and Joseph A Konstan. The movielens datasets: history and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4):19, 2016.

[26] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[27] Xiangnan He and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. In *SIGIR*, pages 355–364, 2017.

[28] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *WWW*, pages 173–182, 2017.

[29] Masakazu Ishihata and Matiheu Blondel. Itemset factorization machines. In *JSAI*, 2017.

[30] Martin Jaggi, Marek Sulovsk, et al. A simple algorithm for nuclear norm regularized problems. In *ICML*, pages 471–478, 2010.

[31] Eric Jones, Travis Oliphant, and Pearu Peterson. Scipy: Open source scientific tools for python, 2001.

[32] Purushottam Kar and Harish Karnick. Random feature maps for dot product kernels. In *AISTATS*, pages 583–591, 2012.

[33] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[34] Donald E Knuth. *The Art of Computer Programming, volume 2: Seminumerical Algorithms*. Addison-Wesley Professional, 2014.

[35] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[36] Quoc Le, Tamás Sarlós, and Alex Smola. Fastfood-approximating kernel expansions in loglinear time. In *ICML*, pages 244–252, 2013.

[37] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Factorized bilinear models for image recognition. In *ICCV*, 2017.

[38] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1754–1763, 2018.

[39] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *ICCV*, pages 1449–1457, 2015.

[40] Fanghui Liu, Xiaolin Huang, Yudong Chen, and Johan AK Suykens. Random features for kernel approximation: A survey in algorithms, theory, and beyond. *arXiv preprint arXiv:2004.11154*, 2020.

[41] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *NeurIPS*, pages 855–863, 2014.

[42] Subhransu Maji and Alexander C Berg. Max-margin additive classifiers for detection. In *ICCV*, pages 40–47. IEEE, 2009.

[43] Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. In *ECML-PKDD*, pages 437–452, 2011.

[44] Rami M Mohammad, Fadi Thabtah, and Lee McCluskey. An assessment of features related to phishing websites using an automated technique. In *ICITST*, pages 492–497, 2012.

[45] Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

[46] Nagarajan Natarajan and Inderjit S Dhillon. Inductive matrix completion for predicting gene–disease associations. *Bioinformatics*, 30(12):60–68, 2014.

[47] Tu Dinh Nguyen, Trung Le, Hung Bui, and Dinh Q Phung. Large-scale online kernel learning with random feature reparameterization. In *IJCAI*, pages 2543–2549, 2017.

[48] Vlad Niculae. A library for factorization machines and polynomial networks for classification and regression in python. https://github.com/scikit-learn-contrib/polylearn/, 2016.

[49] Alexander Novikov, Mikhail Trofimov, and Ivan Oseledets. Exponential machines. *ICLR Workshop*, 2016.

[50] Kyoung Woon On, Jin-hwa Kim, Jeonghee Kim, and Jung-woo Ha. Hadamard product for low-rank bilinear pooling. In *ICLR*, 2017.

[51] Satoshi Oyama and Christopher D. Manning. Using feature conjunctions across examples for learning pairwise classifiers. In *ECML*, pages 322–333, 2004.

[52] Rasmus Pagh. Compressed matrix multiplication. *ACM Transactions on Computation Theory*, 5(3):9, 2013.

[53] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014.

[54] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[55] Jeffrey Pennington, Felix Xinnan X Yu, and Sanjiv Kumar. Spherical random features for polynomial kernels. In *NeurIPS*, pages 1846–1854, 2015.

[56] Ninh Pham and Rasmus Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *KDD*, pages 239–247, 2013.

[57] Danil Prokhorov. IJCNN 2001 neural network competition. Slide Presentation in IJCNN, 2001.

[58] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. Product-based neural networks for user response prediction. In *ICDM*, pages 1149–1154. IEEE, 2016.

[59] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NeurIPS*, pages 1177–1184, 2008.

[60] Steffen Rendle. Factorization machines. In *ICDM*, pages 995–1000, 2010.

[61] Steffen Rendle. Factorization machines with libFM. *ACM Transactions on Intelligent Systems and Technology*, 3(3):57, 2012.

[62] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *SIGIR*, pages 635–644, 2011.

[63] Sam Roweis. https://cs.nyu.edu/ roweis/data.html, 2002.

[64] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[65] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *CIKM*, pages 1161–1170, 2019.

[66] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[67] Andrea Vedaldi and Andrew Zisserman. Efficient additive kernels via explicit feature maps. *IEEE transactions on pattern analysis and machine intelligence*, 34(3):480–492, 2012.

[68] Stefan Wager, Sida Wang, and Percy S Liang. Dropout training as adaptive regularization. In *NeurIPS*, pages 351–359, 2013.

[69] Alastair J Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software*, 3(3):253–256, 1977.

[70] Ruoxi Wang, Rakesh Shivanna, Derek Z Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed H Chi. Dcn-m: Improved deep & cross network for feature cross learning in web-scale learning to rank systems. *arXiv preprint arXiv:2008.13535*, 2020.

[71] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *ICML*, pages 1113–1120, 2009.

[72] Wei Wu, Zhengdong Lu, and Hang Li. Learning bilinear model for matching queries and documents. *Journal of Machine Learning Research*, 14(1):2519–2548, 2013.

[73] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[74] Makoto Yamada, Wenzhao Lian, Amit Goyal, Jianhui Chen, Kishan Wimalawarne, Suleiman A Khan, Samuel Kaski, Hiroshi Mamitsuka, and Yi Chang. Convex factorization machine for toxicogenomics prediction. In *KDD*, pages 1215–1224, 2017.

[75] Felix Xinnan Yu, Ananda Theertha Suresh, Krzysztof Choromanski, Daniel Holtmann-Rice, and Sanjiv Kumar. Orthogonal random features. In *NeurIPS*, pages 1975–1983, 2016.

[76] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *NeurIPS*, pages 5165–5175, 2018.

[77] Weinan Zhang, Tianming Du, and Jun Wang. Deep learning over multi-field categorical data. In *ECIR*, pages 45–57, 2016.

[78] He Zhao, Lan Du, and Wray Buntine. Leveraging node attributes for incomplete relational data. In *ICML*, pages 4072–4081, 2017.