UNIVERSITY OF CALIFORNIA SAN DIEGO

# Linear Kernel Alignment for Domain Generalisation

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Cognitive Science

by

Shuai Tang

2021

University of California San Diego

2021

## DEDICATION

To people who are always looking for effective and efficient solutions.

# EPIGRAPH

*All models are wrong, but some are useful.*

—George E. P. Box (1919 - 2013) and other uncredited statisticians

## TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

## ACKNOWLEDGEMENTS

I want to sincerely thank Virginia R. de Sa, my advisor, for her endless support and incredible patience to me. Virginia helped me to become an independent researcher, and also helped a lot with my self-esteem. I am not particulatly good at being told what to do, so she tailored her way around it by encouraging me to find a mutual ground for both of us where I am still the lead. It is quite lucky for me to have her as an advisor for the past few years.

During my time at UC San Diego, I spent most of my time in between CogSci and CSE department mostly because of my research, made friends and learnt from everyone I know. During the first three years, the support from my cohort was tremendous. To Michael Allen, Amy Rae Fox, Kevin Jensen, Reina Mizrahi, Eric Morgan, and Tricia Ngoon, you all are my heros. To Qin Yao, Saining Xie, Long Jin and Justin Lazarow, we had really good times. To Richard Gao, Tom Donoghue, Dalin Guo, and Weiqi Zhao, thanks for being willing to keep an eye on me, and checking in with me. To Mengting Wan, Jianmo Ni, Wang-Cheng Kang, well, I guess not all of us went to Google in the end. To Mahta Mousavi, Kueida Liao, Xiaojing Xu, Ollie D'Amico and Vijay Veerabadran, thanks for having to listen and having to deal with my very harsh critiques. To everyone whom I have worked with before, thank you. To everyone who have asked for my opinions on their projects, sorry for being so harsh.

I am also very thankful for the working experience I had over the years in the industry. Not that it shaped my decision in joining the industry, it is more about working with other interns with various background and diverse interests for months, and the learning on balancing one's work and life. At Adobe, Hailin Jin was my manager for two times, and along with others, including Chen Fang, Zhaowen Wang, Long Mai, Zhe Lin and Jon Brandt, we worked on interesting research topics that had a deep influence on Adobe's products. At Microsoft Research, I had the chance to work with Paul Smolensky for a summer and continued our collaboration for a year. At Amazon, I was working very closely with Andreas Damianou, Tom Diethe and Wooyoung Moon, and it turned out that we, as a group of people who have very different opinions towards certain research topics, could still work together smoothly and productively.

Obviously I can't miss people I met during those amazing internships. Many thanks to Yingwei Li, Yufei Wang, Longqi Yang, Chenxi Liu, Chuhang Zou (Adobe Research 2016), Zhengqin Lin, Tao Zhou, Zhixin Shu, Chongruo Wu, Zexiang Xu (Adobe Research 2017), Elizabeth Clark, Yoojung Choi, Xin Wang, Shi Feng, Dinghan Shen, Antoine Bosselut, Amr Sharaf, Robin Jia, Asli Celikyilmaz, Richard Shih, Even Pu (MSR 2018), Wesley J. Maddox, Charlie Dickens, Eric Meissner, Pablo Garcia Moreno, Hisham Husain, Eero Siivola, Virginia Aglietti, Xiaoyu Lu (Amazon 2019), Thomas A. Drake, Gabe Pratt and Selvan Senthivel (AWS 2020). I definitely have missed many of you who contributed to my incredible internship experience, apologies.

My journey at UC San Diego wouldn't have happened if Zhuowen Tu hadn't contacted me three days before Apr. 15, 2015. Zhuowen helped me schedule a few interviews with Virginia de Sa, Brad Voytek and Saining Xie. Thanks to all of them for making this happen. The journey of being a grad student will probably end here with five committe members, Virginia de Sa, Ben Bergen, Eran Mukamel, Ndapa Nakashole and Lawrence Saul. I happened to have taken their courses or have attended conferences together, and it makes sense to have them on my committee. Thanks very much.

At the end, I want to thank my parents for loving me all these years. Also thanks to friends who loved me and who probably will still love me in the future.

Chapter 2, in part, is currently being prepared for submission for publication of the material. Tang, Shuai; Maddox, Wesley J.; Dickens, Charlie; Diethe, Tom; Damianou, Andreas. "Sketched CKA: An Efficient Tool for Neural Network Representations". The dissertation author was the primary investigator and author of this material. A preprint version of the material is available on ArXiv.

Chapter 3, in part, is currently being prepared for submission for publication of the material. Tang, Shuai; de Sa, Virginia R. "Deep Transfer Learning with Ridge Regression". The dissertation author was the primary investigator and author of this material. A preprint version of the material is available on ArXiv.

Chapter 4, in part, is currently being prepared for submission for publication of the material. Tang, Shuai; de Sa, Virginia R. “Ensemble Pruning with Linear Kernel Alignment”. The dissertation author was the primary investigator and author of this material.

## VITA

|  |  |
|---|---|
| 2015 | B.Eng. in Information Science and Communication Engineering, Zhejiang University, Hangzhou, China |
| 2016 | Deep Learning Research Intern, Adobe Research Lab, San Jose, USA |
| 2017 | Deep Learning Research Intern, Adobe Research Lab, San Jose, USA |
| 2018 | Deep Learning Research Intern, Microsoft Research, Redmond, USA |
| 2019 | Applied Scientist Intern, Amazon Inc., Cambridge, UK |
| 2020 | Applied Scientist Intern, Amazon Web Services, Virtual, USA |
| 2021 | Ph. D. in Cognitive Science, University of California San Diego |

## PUBLICATIONS

Patrick W Gallagher, Shuai Tang, Zhuowen Tu, "What happened to my dog in that network: Unraveling top-down generators in convolutional neural networks", *ArXiv*, 2015

Shuai Tang, Hailin Jin, Chen Fang, Zhaowen Wang, Virginia R. de Sa, "Rethinking skip-thought: A neighborhood based approach", *2nd Workshop on Representation Learning for Natural Language Processing at ACL2017*, 2017

Shuai Tang, Hailin Jin, Chen Fang, Zhaowen Wang, Virginia R. de Sa, "Trimming and improving skip-thought vectors", *ArXiv*, 2017

Shuai Tang, Hailin Jin, Chen Fang, Zhaowen Wang, Virginia R. de Sa, "Speeding up context-based sentence representation learning with non-autoregressive convolutional decoding", *3nd Workshop on Representation Learning for Natural Language Processing at ACL2018*, 2018

Shuai Tang, Virginia R. de Sa, "Improving Sentence Representations with Consensus Maximisation", *Workshop on Interpretability and Robustness in Audio, Speech, and Language at NeurIPS 2018*, 2018

Shuai Tang, Paul Smolensky, Virginia R. de Sa, "A Simple Recurrent Unit with Reduced Tensor Product Representations", *Workshop on Interpretability and Robustness in Audio, Speech, and Language at NeurIPS 2018*, 2018

Shuai Tang, Mahta Mousavi, Virginia R. de Sa, "An Empirical Study on Post-processing Methods for Word Embeddings", *ArXiv*, 2019

Yinghao Li, Shuai Tang, Virginia R. de Sa, "Supervised Spike Sorting Using Deep Convolutional Siamese Network and Hierarchical Clustering", *BioArXiv*, 2019

Shuai Tang, Virginia R. de Sa, "Exploiting Invertible Decoders for Unsupervised Sentence Representation Learning", *Proceedings of the 57th Conference of the Association for Computational Linguistics*, 4050-4060, 2019

Shuai Tang, Wesley J. Maddox, Charlie Dickens, Tom Diethe, Andreas Damianou, "Sketched CKA: An Efficient Tool for Neural Network Representations", *ArXiv*, 2020

Shuai Tang, Virginia R. de Sa, "Deep Transfer Learning with Ridge Regression", *ArXiv*, 2020

Wesley J. Maddox, Shuai Tang, Pablo Garcia Moreno, Andrew Gordon Wilson, Andreas Damianou, "Fast Adaptation with Linearized Neural Networks", *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, 2021

ABSTRACT OF THE DISSERTATION

**Linear Kernel Alignment for Domain Generalisation**

by

Shuai Tang

Doctor of Philosophy in Cognitive Science

University of California San Diego, 2021

Professor Virginia R. de Sa, Chair

Kernel Alignment has been developed and analysed in the field of multiple kernel learning in the past few decades. Recent studies have shown that kernel alignment with linear kernels can be used to measure the similarity between two sets of high-dimensional vector representations produced by neural networks. Given its theoretical guarantees in learning and practical implication in analysing machine learning models, this thesis examines linear kernel alignment and the algorithm for learning kernel alignment in domain generalisation mainly in three aspects. Firstly, with the help of sketching techniques, we demonstrate that linear kernel alignment is a decent proxy for transferability, which is defined as a score that implies how well a pretrained model would perform on a downstream task. Secondly, in the setting of transferring learnt knowledge from a pretrained neural network to a downstream task, instead of finetuning the top layer or the whole network to adapt to the new task, we propose to accumulate feature vectors from a certain number of layers for making final predictions. The layer selection is done through an algorithm for seeking an optimal convex combination of linear kernels from individual layers. The optimised combination gives as good performance as combining all layers, and the performance is better than simply finetuning the top layer of a pretrained neural network. Thirdly, in ensemble learning, by using the algorithm for learning kernels with linear kernels constructed from individual predictors, the optimal convex combination drastically prunes the predictors required for the inference whilst boosting the performance of ensemble learning methods when all predictors are used. Through these three chapters, we demonstrate the simplicity and practicality of linear kernel alignment in domain generalisation.

# Chapter 1

# Introduction

Domain generalisation is a research topic that tries to address the potential of transferring knowledge from a single source domain with ample data or from multiple source domains to the target domain of our interest. With the increasing amount of unlabelled data available online and the drastically increasing number of parameters, deep learning models have shown promising results on this research topic [YCBL14]. The conventional method is to pretrain a large neural network on an enormous dataset — billions of data samples — collected from a domain that is relevant to the target one, and then fine-tune the network to a small dataset from the target domain through backpropagation. It is the simplest and most straight-forward way of conducting domain generalisation, however, the fine-tuning step can still be time-consuming and costly when the pretrained neural network is large. When multiple pretrained models are given, fine-tuning on all of them becomes incredibly expensive.

Prior to the current deep learning era, kernel machines dominated the theoretical and empirical machine learning research fields due to their simplicity and interpretability. These methods make predictions based on the pair-wise relationshp of dta samples, and the relationship is defined through a manually selected kernel function that computes a score for two data samples. Kernel machines also enjoyed a great success in understanding other machine learning algorithms and in application domains, including text classification [Joa98], protein sequence classification [WLI$^+$05], etc. However, specialised kernel functions were designed for individual application domains. Later on, neural networks were brought to the centre of the stage of the machine learning research due to their flexibility provided by the learnable parameters which adapt themselves to individual applications automatically, and due to their impressive performance in various domains. Researchers are interested in augmenting kernel machines with the same level of flexibility.

Kernel-Target Alignment [CKEST06] was proposed to adjust directions in a kernel matrix to maximally align with the target kernel that is a linear kernel constructed with the target vector. It helps kernel functions to adapt to different datasets accordingly, as opposed to staying fixed throughout the inference regardless of the individual difference among datasets. Built upon the idea of aligning the kernel constructed from data and the target kernel, [CMR12] proposed an algorithm which learns to combine multiple kernels with different hyperparameters. We are interested in adapting the algorithm for learning kernels, which finds an optimal convex combination of kernel functions to adapt to a given datasets, with linear kernels in domain generalisation.

The thesis builds the connection between algorithms for multiple kernel learning and domain generalisation using linear kernels. Specifically, we consider a practical situation when multiple pretrained models or extracted sets of features for the target dataset are given, and instead of directly using all of them to make predictions, a feature selection step is applied with the algorithm for learning kernels ( with only linear kernels ) to reduce the number of models or the number of the sets of features for making predictions. Fig. 1.1 visualises the functionality of the three chapters in this thesis in connecting the kernel machines and domain generalisation.

In Chapter 2, we consider the scenario when many pretrained neural networks are given, and the goal is to efficiently identify the best-performing model among all on the target dataset. We define a transferability index that can be served as a predictive signal of the accuracy of a certain model on a dataset, and we show that the cheaply computed alignment score with only

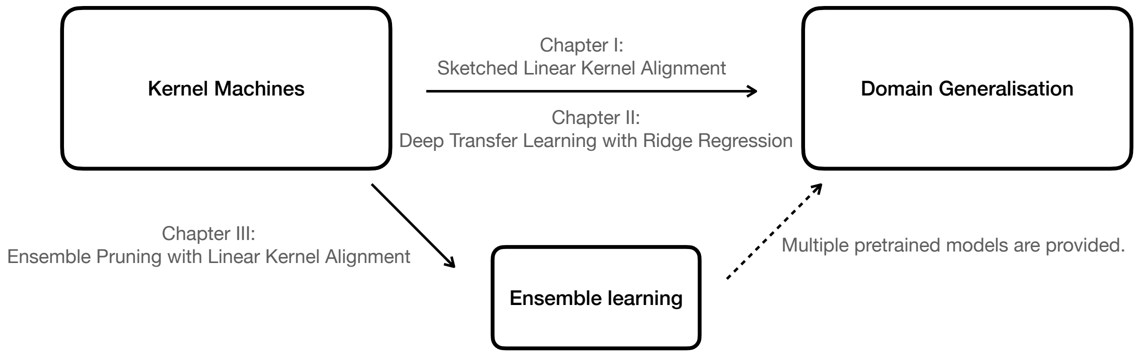

Kernel Machines
Chapter I:
Sketched Linear Kernel Alignment
Chapter II:
Deep Transfer Learning with Ridge Regression
Domain Generalisation
Chapter III:
Ensemble Pruning with Linear Kernel Alignment
Multiple pretrained models are provided.
Ensemble learning

**Figure 1.1**: Three chapters are structured to visualise our contribution in connecting kernel machines and domain generalisation. Specifically, we use the idea of learning kernels with multiple kernel functions to the situation where multiple pretrained models are provided in transfer learning.

linear kernels can be a proxy for the index that we are interested in. As the computation for the transferability needs to be considerably faster than the computation for obtaining the final accuracy, sketching techniques [Woo14] are applied to speed up the computation and reduce the memory cost of Linear Kernel Alignment. When multiple pretrained models are provided, our proposed method can quickly identify which model is the best-performing one on a given target dataset without fine-tuning. Simulations on randomly generated tasks from real-world datasets with pretrained neural networks show a strong linear relationship between the alignment score and the accuracy.

In Chapter 3, we address the question of how well a pretrained neural network performs on a target dataset without fine-tuning, and our potential solution is to accumulate features produced from multiple layers of a neural network. A common hypothesis, which is supported by empirical observations, is that different layers encode different levels of abstraction of the input data. As described in [TPB00], bottom layers have high mutual information with the input but low mutual information with the labels, and top layers show high mutual information with the labels as they are very close to the loss function. Technically, all layers can be accumulated for making predictions, but the computational cost is unbearable as intermediate layers produce feature vectors in very high dimensional spacs. We propose to use Linear Kernel Alignment to determine the minimum number of layers to accumulate for making predictions. Firstly, we can show that accumulating only selected layers performs as well as using all layers, but the former is much faster than the latter. Secondly, our proposed method provides higher accuracy for six selected classification tasks than fine-tuning only the top layer, and, with the same time budget, our method achieves better performance than fine-tuning the top three layers and fine-tuning the whole neural network.

In previous chapters, provided models are neural networks, and they are able to generate vector representations of the input data. Generated vectors are sent into Linear Kernel Alignment for selecting the most necessary features. In Chapter 4, we consider a more restricted but more generic scenario where we only have access to the predictions without the vector representations produced by models. and we examine the possibility of applying Linear Kernel Alignment for pruning provided models so that, during testing, only the remaining models are used, which leads to faster test runtime compared against using all pretrained models. To simulate the environment, bagging and boosting are used to create pretrained models, and the base learners include decision trees, k-nearest neighbours and multi-layer perceptrons respectively. Linear Kernel Alignment is applied on top of the predictors provided from all models on the training set for selecting predictions from only a small set of pretrained models so that the combined prediction gives the maximum alignment score with the target kernel. Experiments indicate that our proposed pruning method boosts the performance of the original ensemble learning method in most cases, and maintains the same performance in the remaining cases, whilst pruning a large number of predictors.

Next, we briefly introduce the definition of Kernel Alignment and its simplified form with only linear kernels. Then we discuss how it is applied in each chapter of the thesis.

Given a kernel matrix $\boldsymbol{K} \in \mathbb{R}^{N \times N}$, where $N$ is the number of data samples, each entry $\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ can be considered as the relationship between a pair of data samples $(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and the relationship is defined through the kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. When two kernel matrices are provided $\boldsymbol{K}_1 \in \mathbb{R}^{N \times N}$ and $\boldsymbol{K}_2 \in \mathbb{R}^{N \times N}$, where $k_1 : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and $k_2 : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ are the corresponding kernel functions, the definition of Kernel Alignment is as follows:

$$\rho(\boldsymbol{K}_1, \boldsymbol{K}_2) = \frac{\langle \boldsymbol{K}_1, \boldsymbol{K}_2 \rangle_F}{||\boldsymbol{K}_1||_F ||\boldsymbol{K}_2||_F} \in [0, 1] \tag{1.1}$$

where $||\cdot||_F$ is the Frobenius norm, and $\langle \boldsymbol{K}_1, \boldsymbol{K}_2 \rangle_F = \mathrm{Tr}(\boldsymbol{K}_1 \boldsymbol{K}_2^\top)$ is the Frobenius inner product. If we vectorise both kernel matrices and denote them as $\mathrm{vec}(\boldsymbol{K}_1)$ and $\mathrm{vec}(\boldsymbol{K}_2) \in \mathbb{R}^{N^2}$, Eq. 1.1 can be written as:

$$\rho(\boldsymbol{K}_1, \boldsymbol{K}_2) = \frac{\mathrm{vec}(\boldsymbol{K}_1)^\top \mathrm{vec}(\boldsymbol{K}_2)}{||\mathrm{vec}(\boldsymbol{K}_1)||_2 ||\mathrm{vec}(\boldsymbol{K}_2)||_2}, \tag{1.2}$$

which is equivalent to computing the cosine similarity between two vectorised matrices. Since in this thesis, we mainly consider linear kernel functions $k_1(\boldsymbol{x}, \boldsymbol{y}) = k_2(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\top \boldsymbol{y}$, where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, the definition of Kernel Alignment can be simplified in the following way:

$$\rho(\boldsymbol{K}_1, \boldsymbol{K}_2) = \frac{\langle \boldsymbol{X}\boldsymbol{X}^\top, \boldsymbol{Y}\boldsymbol{Y}^\top \rangle_F}{||\boldsymbol{X}\boldsymbol{X}^\top||_F ||\boldsymbol{Y}\boldsymbol{Y}^\top||_F} = \frac{||\boldsymbol{X}^\top \boldsymbol{Y}||_F^2}{||\boldsymbol{X}\boldsymbol{X}^\top||_F ||\boldsymbol{Y}\boldsymbol{Y}^\top||_F}, \tag{1.3}$$

where $\boldsymbol{X} \in \mathbb{R}^{N \times d_x}$ and $\boldsymbol{Y} \in \mathbb{R}^{N \times d_y}$.

The algorithm proposed in [CMR12] considers the situation when multiple kernel functions are provided and it finds a convex combination of kernels so that the combined kernel has the maximum kernel alignment with the target kernel. To translate the situation in our case, there are a set of features produced from different models for the same dataset $\{\boldsymbol{X}_i\}_{i=1}^p$ , where $\boldsymbol{X}_i \in \mathbb{R}^{N \times d_i}$, $\forall i \in \{1, 2, \dots, p\}$ and $p$ is the number of models. The associated target is given is a vector $\boldsymbol{y} \in \mathbb{R}^N$. Now, we aim to find a set of $\{\mu_i\}_{i=1}^p$ so that the combined kernel $\boldsymbol{K} = \sum_{i=1}^p \mu_i \boldsymbol{X}_i \boldsymbol{X}_i^\top$ aligns with the target kernel $\boldsymbol{K}_Y = \boldsymbol{y}\boldsymbol{y}^\top$ maximally. The optimisation problem can be formalised as follows:

$$\begin{aligned}
\boldsymbol{\mu}^\star &= \underset{\{\boldsymbol{\mu}>0 \wedge ||\boldsymbol{\mu}||=1\}}{\arg\max} \frac{\langle \boldsymbol{K}, \boldsymbol{K}_Y \rangle_F}{||\sum_{i=1}^{p} \mu_i \boldsymbol{K}_i||_F ||\boldsymbol{K}_Y||_F} \\
&= \underset{\{\boldsymbol{\mu}>0 \wedge ||\boldsymbol{\mu}||=1\}}{\arg\max} \frac{\langle \sum_{i=1}^{p} \mu_i \boldsymbol{X}_i \boldsymbol{X}_i^\top, \boldsymbol{y}\boldsymbol{y}^\top \rangle_F}{||\sum_{i=1}^{p} \mu_i \boldsymbol{K}_i||_F} \\
&= \underset{\{\boldsymbol{\mu}>0 \wedge ||\boldsymbol{\mu}||=1\}}{\arg\max} \frac{\sum_{i=1}^{p} \mu_i ||\boldsymbol{X}_i^\top \boldsymbol{y}||_2^2}{||\sum_{i=1}^{p} \mu_i \boldsymbol{X}_i \boldsymbol{X}_i^\top||_F}, \qquad (1.4)
\end{aligned}$$

where $\boldsymbol{\mu} > 0$ is the non-negative constraint imposed on individual $\mu_i$s and $||\boldsymbol{\mu}|| = 1$ is a norm constraint. When the set of features $\{\boldsymbol{X}_i\}_{i=1}^{p}$ are not pair-wise linearly independent from each other, the non-negative constraint performs feature selection.

Chapter 2 discuss the application of Linear Kernel Alignment in Eq. 1.3 as a proxy for transferability, which in our case is defined as a predictive signal for the accuracy of a pretrained model on a downstream task. The equation is essentially the same as in Eq. 1.3 with $\boldsymbol{X}$ containing the feature vectors produced by a pretrained neural network on a new task, and $\boldsymbol{Y}$ containing the one-hot encoded target vectors for the classification labels. Since computing the alignment score requires us to store both $\boldsymbol{X}$ and $\boldsymbol{Y}$ at the same time, with increasing number of data samples, the memory cost and the computational complexity increase. To alleviate the issue, we propose to sketch both matrices with the same random projection matrix $\boldsymbol{S} \in \mathbb{R}^{k \times N}$, and the projection dimension $k$ is fixed prior to the computation. The sketched alignment score is then computed as :

$$\rho_{\boldsymbol{S}}(\boldsymbol{K}_1, \boldsymbol{K}_2) = \frac{||\boldsymbol{X}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{Y}||_F^2}{||\boldsymbol{S}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{S}^\top||_F ||\boldsymbol{S}\boldsymbol{Y}\boldsymbol{Y}^\top \boldsymbol{S}^\top||_F}, \qquad (1.5)$$

To further speed up the computation, instead of using $\boldsymbol{S}$ filled with samples from a Gaussian random variable, we use the hashing-based Count-Sketch method [CW13] which has only one non-zero entry per column in $\boldsymbol{S}$. Through sketching, we improve the practicality of the alignment score in domain generalisation.

Chapter 3 explores the possible optimal performance of a pretrained neural network on a downstream task without fine-tuning. Our potential solution is to combine features from multiple layers on the same dataset, and use the algorithm for kernel alignment to determine layers to accumulate. Given a pretrained neural network with $L$ layers and a dataset $\{\boldsymbol{X}, \boldsymbol{Y}\}$ where $\boldsymbol{X} \in \mathbb{R}^{N\times d}$ and $\boldsymbol{Y} \in \mathbb{R}^{N\times C}$, and $C$ is the number of classes. By forwarding the dataset into the pretrained neural network, each layer produces a feature matrix $\boldsymbol{X}_l \in \mathbb{R}^{N\times d_l}$. Then the following optimisation problem similar to Eq. 1.4 is solved:

$$\boldsymbol{\mu}^\star = \arg\max_{\{\boldsymbol{\mu}>0\}} \frac{\sum_{l=1}^{L} \mu_l ||\boldsymbol{X}_l^\top \boldsymbol{Y}||_F^2}{||\sum_{l=1}^{L} \mu_l \boldsymbol{X}_l \boldsymbol{X}_l^\top||_F}, \tag{1.6}$$

Features produced from layers with non-zero $\mu_l$s are accumulated for making final predictions. An obvious issue is that features from intermediate layers $\boldsymbol{X}_l \in \mathbb{R}^{N\times d_l}$ are in very high-dimensional spaces, storing them for the optimisation problem is memory-intense. Empirical evidence has shown that features produced from neural networks are approximately low-rank, which means that we only need a few directions to approximate the covariance structure. Therefore, we consider the Nyström method to approximate the linear kernel at each layer, and we set the maximum approximation dimension $k$. The features sent to the optimisation problem are the low-rank approximations which have maximum size $\mathbb{R}^{N\times k}$. As mentioned at the beginning of this chapter, fine-tuning is still the simplest and most straight-forward way for domain generalisation, but our method can serve as a reliable and cheap alternative.

Chapter 4 describes a two-step method to prune the predictors in ensemble learning. The predictors learnt in ensemble learning are not all necessary for making predictions at the test time, as summarised in [TPV09], and pruning predictors leads to fast inference for new data samples and sometimes better performance. We propose to first run the algorithm for learning kernels on top of predictions generated from a large number of models provided by an ensemble learning method, either bagging or boosting with a certain base learner. The objective function in the first step is similar to Eq. 1.4, and it is defined as follows:

$$\boldsymbol{\mu}^{\star} = \underset{\{\boldsymbol{\mu}>0\}}{\arg\max} \frac{\sum_{i=1}^{p} \mu_i (\boldsymbol{y}_i^{\top} \boldsymbol{y})^2}{||\sum_{i=1}^{p} \mu_i \boldsymbol{y}_i \boldsymbol{y}_i^{\top}||_F} = \underset{\{\boldsymbol{\mu}>0\}}{\arg\max} \frac{\sum_{i=1}^{p} \mu_i (\boldsymbol{y}_i^{\top} \boldsymbol{y})^2}{\left(\sum_{i=1}^{p} \sum_{j=1}^{p} \mu_i \mu_j (\boldsymbol{y}_i^{\top} \boldsymbol{y}_j)^2\right)^{\frac{1}{2}}}, \tag{1.7}$$

where $\boldsymbol{y}_i \in \mathbb{R}^N$ contains predictions on the training set produced by the $i$-th predictor. After solving the optimisation problem, predictors with non-zero $\boldsymbol{\mu}$ remain and the rest are pruned. The test runtime is shortened due to pruning. Our method stands out among other methods when the base learner used in the ensemble learning is relatively low-variance. The experiments show that not only does our method keep a very small portion of the predictors, it also boosts the performance on top of the ensemble learning in most cases.

Individual chapters of the thesis have their own domain-specific introduction, related work and conclusion. Technical details are left to the last section in each chapter to avoid breaking the continuity of content in each chapter. In the final chapter, we summarise our contributions.

# Chapter 2

# Sketched Centered Kernel Alignment

## 2.1 Introduction

Neural networks exhibit significant redundancy in learnt representations presented in hidden units. The capability of learning high-dimensional features helps neural networks to generalise well, as has been extensively studied in the literature of overparameterised models [BG19]. However, this characteristic results in intolerable time and memory complexity for methods that study the learnt representations as a probe for further understanding of neural networks, and their usage in transfer learning. With the empirical observations that learnt feature vectors of a neural network are practically low-rank, the bottlenack on the complexity can be handled with approaches which drastically reduce the redundancy whilst maintaining learnt structures within the representations. Previously proposed pruning methods can achieve this goal by cutting uninformative connections between neurons, although it is challenging to come up with a one-fits-all algorithm that guarantees good generalisation and low-memory cost. In this paper, we explore techniques from the area of random projections, which we find to be particularly suited for exploring the low-rank properties of feature representations for the purposes of analysing trained neural networks within transfer learning scenarios.

Empirical findings have shown that the feature representations are approximately low-rank in most cases, including the spatial feature maps in convolutional neural networks [JVZ19] and the attention maps in transformers [WLK$^+$20]. Therefore, it permits low-rank approximation techniques to be engaged in analysing and studying neural networks. Rather than directly storing and analysing feature representations themselves, one can hope to get a low-rank approximation of the original feature matrix, or obtain a data summary that requires less memory to store but still maintains information, such as principal directions themselves, and the variance explained on each. This motivates us to apply random projections in analysing and studying the encoded information in feature vectors in neural networks, and guide us on how to transfer neural networks.

Random projection is a tool for mainly reducing the dimension of data points in Euclidean space with theoretical guarantees on maintaining the pairwise relationship of data points. Alternatively, one can create data summaries with random projections that preserve the spectral properties, so that the number of data samples can be reduced tremendously. Recent research also showed that random projections can be applied to construct explicit kernels [PP13] or kernel approximations [RR$^+$07], and it is potentially helpful in transfer learning as well. Due to the memory limit of existing GPUs, one can only fit a batch of data samples per iteraction and use neural networks to produce features, which is similar to the streaming setting of data processing, therefore, ideally we want random projections with subspace embedding guarantees which are also efficient on data streams.

In this paper, we propose to sketch feature vectors at individual layers to reduce the memory complexity of storing them, and the computational complexity in computing the alignment in between feature vectors using Centered Kernel Alignment (CKA) [CMR12]. We theoretically and empirically show that, with sketching, the CKA results are still reliable for illustrating properties of neural networks but can be obtained much faster, and can be served as an indicator of transferability of a neural network. Then we combine the sketching techniques with an algorithm for learning kernels [CMR12] in transfer learning setting to optimally combine a large number of pretrained models efficiently.

## 2.2 Related Work

Advanced tools to better understand neural networks and their learning dynamics have been recently developed. A majority of them approach the problem through either feature or gradient vectors [KNLH19]. The former methods consider neural networks as feature extractors. Therefore, by studying the properties of feature vectors, one can gain insights on the input-output relations the neural network is approximating. The latter methods take functional analysis algorithms, and through studying the Jacobian matrix w.r.t. parameters defined by a neural network, they provide insights into the geometry of functions that a wired neural network is capable of approximating. For our work, we focus on the feature vectors, as given a dataset, the forward computation that produces feature maps is easily parallelisable in batch mode in most deep learning frameworks, however, the parallelisation of gradient computations is more involved.

Kernel approximations by subsampling for the popular Nyström method have been studied extensively [GM16]. As empirically illustrated [KMT12], the sampling methods on rows or columns play an important role in the success of the approximation. As in our case, the feature vectors are produced per batch of samples given a dataset. Therefore, the row of the feature matrix $\boldsymbol{A} \in \mathbb{R}^{N \times d}$ are observed sequentially. This situation limits the number of subsampling methods we can select from. However, since we aim to directly get a data summary of $\boldsymbol{A}$, it becomes a task that requires row-wise subsampling for approximating a linear kernel.

A straightforward option is uniform sampling of the data, which has been predominantly applied in materialising a data summary in previous studies of neural networks' features [RGYSD17, MRB18, KNLH19]. The main issue is that uniform sampling can be arbitrarily bad in making a query of any quantity w.r.t. the full matrix [CLM+15], thus it may cause invalid observations. In order to balance efficient streaming capability with strong approximation guarantees [DG03], we propose using the sparse Johnson-Lindenstrauss Transform (SJLT) to summarise the feature matrix $\boldsymbol{A}$ for further studies of neural networks.

Besides accelerating the computation for analysing neural networks, one can certainly take advantages of data summaries and make predictions on a downstream task directly with a linear model. When many pretrained neural networks are provided, we show that by taking the data summaries of the top layer, it becomes efficient to take advantages of all models and improve the predictive accuracy in transfer learning.

## 2.3 Method

[KNLH19] has shown the potential use case of CKA in comparing neural networks' features. We now briefly reiterate the concept and discuss the complexity of applying it in large scale. Then we introduce our approach to make CKA for neural networks comparison computationally efficient.

### 2.3.1 Centred Kernel Alignment

The definition is given as:

$$\rho(\boldsymbol{K}_1, \boldsymbol{K}_2) = \frac{\langle \boldsymbol{K}_1, \boldsymbol{K}_2 \rangle_F}{||\boldsymbol{K}_1||_F ||\boldsymbol{K}_2||_F} \tag{2.1}$$

where $\boldsymbol{K}_1, \boldsymbol{K}_2 \in \mathbb{R}^{N \times N}$. For linear kernels $\boldsymbol{K}_1 = \boldsymbol{X}\boldsymbol{X}^\top$ and $\boldsymbol{K}_2 = \boldsymbol{Y}\boldsymbol{Y}^\top$, the definition can be formalised as

$$\rho(\boldsymbol{K}_1, \boldsymbol{K}_2) = \frac{||\boldsymbol{X}^\top \boldsymbol{Y}||_F^2}{||\boldsymbol{X}\boldsymbol{X}^\top||_F ||\boldsymbol{Y}\boldsymbol{Y}^\top||_F}, \tag{2.2}$$

where $\boldsymbol{X} \in \mathbb{R}^{N \times d_x}$ and $\boldsymbol{Y} \in \mathbb{R}^{N \times d_y}$ are zero-centred feature vectors.

As CKA is designed for measuring the normalised alignment between two zero-centred kernel matrices, it is natually invariant to rotation in the feature space and isotropic scaling of the kernel functions. The fact that it is not invariant to invertible linear transformations makes it suitable for comparing neural networks as discussed and empirically illustrated in [KNLH19], since a similarity measure that is invariant to invertible linear transformations, including $R^2$ statistics and CCA, gives a similarity score 1 when one of $\boldsymbol{X}$ and $\boldsymbol{Y}$ has full row rank, due to overfitting.

**Applications.** CKA measures the alignment between two sets of samples given two kernel functions. For simplicity, we stick to linear kernels in our following sections. There are numerous applications of CKA. Besides measuring the similarity between two sets of feature vectors from the same neural network or two different ones, CKA can also be used to study the alignment between feature vectors and targets as its original purpose - learning kernels. Theorem 13 and 15 in [CMR12] showed the error of a binary classification task is upper bounded by $1-\rho(\boldsymbol{X}\boldsymbol{X}^\top, \boldsymbol{Y}\boldsymbol{Y}^\top)/\Gamma$, where $\Gamma$ is the maximum output of the classifier.

The first application helps us understand the grouping of layers $\{\boldsymbol{A}_i\}_{i=1}^L$ within a neural network by the CKA scores, and how individual layers evolve during learning. The second one helps us to illustrate the alignment between individual layers $\boldsymbol{A}_i \in \mathbb{R}^{N\times d_l}$ and the targets $\boldsymbol{Y}\boldsymbol{Y}^\top$. Though it only gives a lower bound on the accuracy, we show that CKA tracks the accuracy closely, and helps model selection.

**Complexity.** Unfortunately, storing the full kernel matrix of individual layers would require $O(N^2)$ memory whilst matrix multiplication to compute the similarity indices would require $O(N(d_x+d_y))$ in Eq. 2.2 or $O(N^2)$ in Eq. 2.1 computational time, where $N$ is the number of samples.

### 2.3.2 Summarising Data

The numerator of CKA is the Hilbert Schmidt Independence Criterion (HSIC) [GHS$^+$05] scaled by the squared number of samples, and HSIC is a successful test of nonlinear dependency between two sets of samples $\boldsymbol{X}$ and $\boldsymbol{Y}$ in RKHS $\mathcal{H}_1 \times \mathcal{H}_2$ induced by $k_1(\cdot,\cdot) \times k_2(\cdot,\cdot)$. When $k_1$ and $k_2$ are characteristic, $\text{HSIC}(\mathcal{H}_1,\mathcal{H}_2,\boldsymbol{X},\boldsymbol{Y}) = 0$ iff $\boldsymbol{X}$ and $\boldsymbol{Y}$ are independent. Technically, as long as the number of samples are the same in two datasets, HSIC or CKA can be applied to compute a proxy score as the similarity between two datasets with carefully chosen kernel functions [GBSS05, MFSS17].

From the perspective of testing independence of two datasets $\boldsymbol{X}$ and $\boldsymbol{Y}$ in the product kernel space, HSIC can be interpreted as measuring the distance between the joint distribution $p_{\boldsymbol{XY}}$ and the product of marginal distributions $p_{\boldsymbol{X}} \otimes p_{\boldsymbol{Y}}$ [MFSS17]. Therefore, HSIC can be rewritten as

$$\text{HSIC}(\mathcal{H}_1,\mathcal{H}_2,\boldsymbol{X},\boldsymbol{Y}) = ||\mathcal{C}_{\boldsymbol{XY}}||^2_{\text{HS}} = ||\mu_{p_{\boldsymbol{XY}}} - \mu_{p_{\boldsymbol{X}} \otimes p_{\boldsymbol{Y}}}||_{\mathcal{H}_1 \times \mathcal{H}_2} \tag{2.3}$$

where $\mathcal{C}_{\boldsymbol{XY}}$ is the covariance operator and $\mu_{\boldsymbol{XY}} = \mathbb{E}_{\boldsymbol{XY}}[\boldsymbol{X}(X) \otimes \boldsymbol{Y}(Y)] \in \mathbb{R}^{d_x \times d_y}$. As HSIC is the squared norm of $\mathcal{C}_{\boldsymbol{XY}}$ in $\mathcal{H}_1 \times \mathcal{H}_2$, and $\mathcal{C}_{\boldsymbol{XY}}$ is an expected value, it encourages us to apply sketching techniques to get a summary of data samples in each dataset prior to computing HSIC, which is the numerator of CKA. Summarising data samples reduces both the memory complexity for storing the matrix and the time complexity for computing CKA.

### 2.3.3 Empirical Evidence

It has been shown that big data matrices are approximately low-rank [UT19]. However, we aim to show empirically that feature maps from layers in a neural network have a similar property. This will motivate using small-space summaries that preserve the directions manifest by the feature map. Due to noise, real-world data matrices can be high (algebraic) rank, so we instead compute the stable (or numerical) rank of a feature matrix defined as

$$\text{srank}(\boldsymbol{A}) = ||\boldsymbol{A}||_F^2 / ||\boldsymbol{A}||_2^2 \leq \text{rank}(\boldsymbol{A}). \tag{2.4}$$

Stable rank is less sensitive to small singular directions so is more robust than algebraic rank; it measures the speed of spectral decay of the data. [CNW15] show that $\boldsymbol{A}$ can be approximately reconstructed with $O(\text{srank}(\boldsymbol{A}) \log(\text{srank}(\boldsymbol{A})/\delta)/\varepsilon^2)$ directions. A matrix with a small stable rank indicates that one can use a few directions to approximate the matrix without losing much information.

To show that the learnt feature maps at individual layers $\{\boldsymbol{A}_i\}_{i=1}^L$ of a pretrained ImageNet neural networks are approximately low-rank, we forward four datasets, CIFAR10, CIFAR100, STL10 and SVHN to ResNet-18 and ResNet-34, and compute the stable rank value at each layer. Fig. 2.1 presents the results. It is clear that, even though the dimension of feature vectors is $O(10^5)$, the maximum stable rank of all layers of two models is less than 40. The observation serves as a piece of empirical evidence that feature matrices are approximately low-rank, thus it is not necessarily required to store the full feature matrix at each layer.

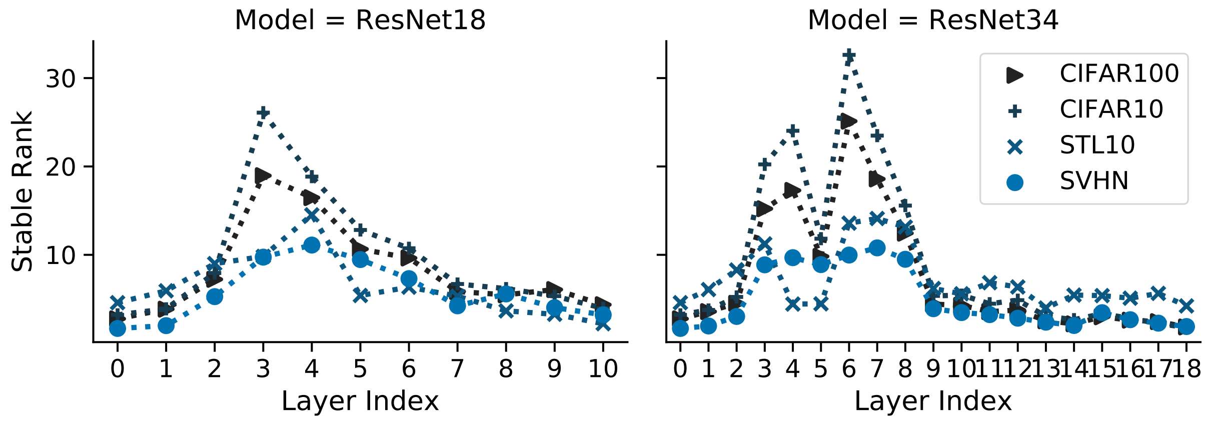


**Figure 2.1**: **Stable Rank.** The stable rank of 4 datasets in 2 networks consistently has stable rank $\ll d = O(10^5)$.

### 2.3.4 Sketching the Kernel Matrix

We wish to ease the computational burden by using a low-rank approximation to the feature matrices. By approximating two feature matrices with the same rank, sketching provides a way around needing original data samples when comparing models as in [KNLH19]. We apply a version of the CountSketch (also known as CWT) of [CW13] which is extremely sparse and can be applied efficiently by *streaming* through the input, which has time complexity that is linear in number of total data samples. The algorithm is presented in Alg. 1. The CWT can be seen as applying a sign function to the rows of input $\boldsymbol{X}$ followed by hashing the rows to $M$ buckets uniformly at random. A detailed definition is given in Sec. 2.5.1.

**Algorithm 1:** Feature Sketching

```
Data: a neural network $\phi$, a dataset $D = \{\boldsymbol{x}_i\}_{i=1}^{N}$, number of buckets $M$, and batch
      size $bs$
Result: Data Summary $\tilde{\boldsymbol{A}}$
$\tilde{\boldsymbol{A}} = \mathbf{0} \in \mathbb{R}^{M\times d}$, $t = 0$ while $t \leq \lceil \frac{N}{bs} \rceil$ do
    sample a batch of data $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=bs\times t+1}^{bs\times(t+1)}$, ;
    compute features $\boldsymbol{A} = \phi(\boldsymbol{X}) \in \mathbb{R}^{bs\times d}$, for $j = 1;\ j \leq bs;\ j++$ do
        sample a binary value $s$, ;
        uniformly sample an index $k \in \{1, 2, ..., M\}$ $\tilde{\boldsymbol{A}}[k,:] = \tilde{\boldsymbol{A}}[k,:] + s\boldsymbol{A}[j,:]$
    end
    $t = t + 1$
end
```

After sketching the two matrices being compared in CKA, we have individual summaries $\tilde{\boldsymbol{K}}_1 = \boldsymbol{S}\boldsymbol{X}\boldsymbol{X}^\top\boldsymbol{S}^\top$ and $\tilde{\boldsymbol{K}}_2 = \boldsymbol{S}\boldsymbol{Y}\boldsymbol{Y}^\top\boldsymbol{S}^\top$, where $\boldsymbol{S}$ is a random projection matrix that satisfies subspace embedding requirements. It is important to know how sketching affects the accuracy of similarities. We provide a bound on the difference between Sketched CKA (SCKA) and CKA The first step shows how much HSIC changes with sketching, and the second step shows how much the SCKA differs from CKA.

**Lemma 1** With probability $1-\delta$, $\left| ||\boldsymbol{X}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{Y}||_F - ||\boldsymbol{X}^\top \boldsymbol{Y}||_F \right| \leq \varepsilon ||\boldsymbol{X}^\top \boldsymbol{Y}||_F$ .

**Lemma 2** With probability $1-\delta$, $|\rho_S - \rho| \leq \frac{4\varepsilon}{(1-\varepsilon)^2}\rho$,

where $\rho_S = \rho(\tilde{\boldsymbol{K}}_1, \tilde{\boldsymbol{K}}_2) = \rho(\boldsymbol{S}\tilde{\boldsymbol{K}}_1\boldsymbol{S}^\top, \boldsymbol{S}\tilde{\boldsymbol{K}}_2\boldsymbol{S}^\top)$, and $\rho = \rho(\boldsymbol{K}_1, \boldsymbol{K}_2)$. Proofs of each lemma are in Sec. 2.5.4.

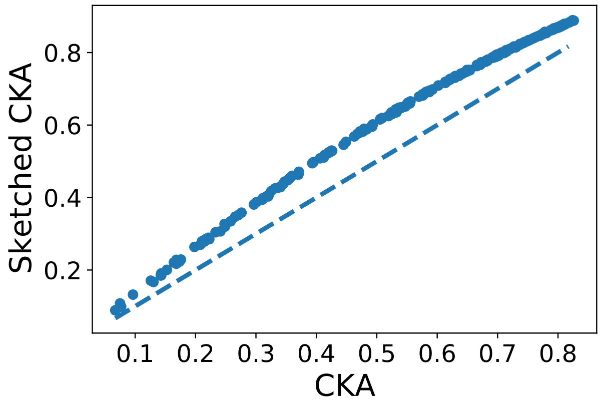


**Figure 2.2**: **The comparison between Sketched CKA and CKA values.** The scatter plot shows 250 pairs of CKA and Sketched CKA values, and the dashed line presents where the value pairs are equal ($y = x$). It empirically verifies that the absolute difference between CKA and Sketched CKA is small.

Lemmas 1 & 2 show that sketching approximately preserves the learnt representations. The lemmas hold for arbitrary subspace embeddings so $\boldsymbol{S}$ can be sampled from a family with better failure probability dependency than the CountSketch (e.g SJLT).

To empirically illustrate the difference between $\rho_{\text{SCKA}}(\boldsymbol{X}, \boldsymbol{Y})$ and $\rho_{\text{CKA}}(\boldsymbol{X}, \boldsymbol{Y})$. A data matrix $\boldsymbol{X} \in \mathbb{R}^{10,000 \times 512}$ is generated, and 250 affine transformations $\{\boldsymbol{T}_i\}_{i=1}^{250}$ are sampled to construct $\{\boldsymbol{Y}_i\}_{i=1}^{250}$ with $\boldsymbol{Y}_i = \boldsymbol{X}\boldsymbol{T}_i$. It results in 250 $\rho_{\text{CKA}}(\boldsymbol{X}, \boldsymbol{X}\boldsymbol{T}_i)$ values that fall uniformly in $[0,1]$. We apply an SJLT, $\boldsymbol{S}_i \in \mathbb{R}^{512 \times 10,000}$, to $\boldsymbol{X}$ in each pair to compute $\rho_{\text{SCKA}}(\boldsymbol{X}, \boldsymbol{X}\boldsymbol{T}_i) = \rho_{\text{CKA}}(\boldsymbol{S}_i\boldsymbol{X}, \boldsymbol{S}_i\boldsymbol{X}\boldsymbol{T}_i)$. Fig. 2.2 shows 250 pairs of Sketched CKA and CKA values, and it is clear that value pairs are very close to each other with Sketched CKA values being slightly larger than CKA ones.

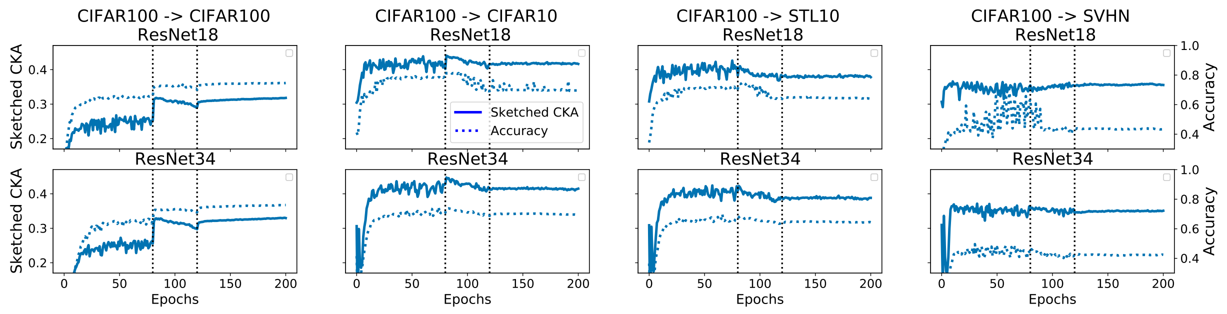


**Figure 2.3**: **SCKA between the top layer features and targets vs. Accuracy.** Each solid line describes the change of our proposed Sketched CKA between the top layer features of a ResNet model being trained on CIFAR100 evaluated on a target task. Each dotted line presents the change of accuracy using the same features on the target task. Dashed vertical lines indicate two steps of decreasing the learning rate.
It is clear that Sketched CKA, which provides a lower bound on the accuracy, actually tracks the change of performance of the top layer features on four tasks presented in the plots. Once the learning rate decays, SCKA and Accuracy both increase when the target task is CIFAR100 itself, however, both decrease when the target task isn't the same as the training task. It implies that decreasing the learning rate helps the network to excel on the same dataset, but decreases the transferability to other tasks.

### 2.3.5 Our Method in a Nutshell

Given a neural network $\phi$ and a dataset $D$ with $N$ samples, our method sketches each data sample into $M$ buckets. This step can be parallelised in the batch mode in deep learning frameworks. Then, the inner product of the summary matrix and itself gives the kernel matrix $\tilde{\boldsymbol{K}} \in \mathbb{R}^{M\times M}$.

## 2.4 Results

We present four sets of experiments to show that 1) our proposed way of sketching data samples or feature vectors to a fixed size still demonstrates the expected behaviours of neural networks in terms of similarity scoring between layers, 2) the Sketched CKA value can serve as a measure of transferability of top layer features of a neural network given a target task. The detailed experimental design and results are presented in each subsection. Since a majority of experiments are conducted on ResNet models, we use the layer index $l$ to denote the index of a

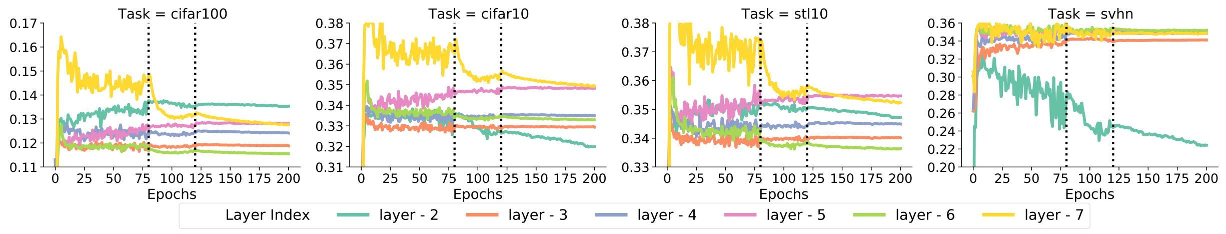


**Figure 2.4**: **SCKA between layers and targets.** $\{\rho_S(\boldsymbol{A}_l\boldsymbol{A}_l^\top, \boldsymbol{Y}\boldsymbol{Y}^\top)\}_{l=2}^7$ and $l = 8$ is presented in Fig. 2.3 as it is the top layer. Plots presents the convergence of each layer during the training of a ResNet-18 model on CIFAR100 evaluated with 4 probing datasets. The observation is relatively consistent with prior work that layer 1-6 converge faster than layer 7 and 8 when the probing task comes from a similar domain. Whilst on SVHN, the alignment between layer 2 and targets decreases quickly across training on CIFAR100, and higher layers converges quicker.

residual block which often contains two or three convolutional layers.

CKA takes two kernel matrices $\tilde{\boldsymbol{K}}_1$ and $\tilde{\boldsymbol{K}}_2$ as inputs and outputs an alignment score.

### 2.4.1 Training ResNet Models from Scratch

Two Residual Networks (ResNet) [HZRS15], ResNet-18 and 34, are trained on CIFAR100 for 200 epochs. Stochastic Gradient Descent (SGD) is used. The initial learning rate is 0.1, and is decreased by a factor of 10 at epoch 80 and 120 as recommended in [HZRS16]. The weight decay is set to be $5e-4$. Details can be found in the publicly available implementations[1]. We denote first 80 epochs as stage 1, 80 - 120 as stage 2, and the rest stage 3.

During training, we evaluate SCKA $\rho_S(\boldsymbol{A}_L\boldsymbol{A}_L^\top, \boldsymbol{Y}\boldsymbol{Y}^\top)$ and accuracy of the top layer features on predicting targets, with a simple linear model, on CIFAR100, CIFAR10, [Kri09], STL10 [CNL11] and SVHN [NWC+11]. The results are shown in Fig. 2.3. In stage 1, both SCKA between the top layer features and the targets and the accuracy of using the same feature vectors for predicting the targets are increasing gradually. At the beginning of stage 2, both values have a significant increase on CIFAR100, and a small increase on CIFAR10. It implies that lowering the learning rate helps a model become specialised on the training dataset CIFAR100, and it helps the top layer features generalise on datasets from a similar domain.

[1]https://github.com/kuangliu/pytorch-cifar

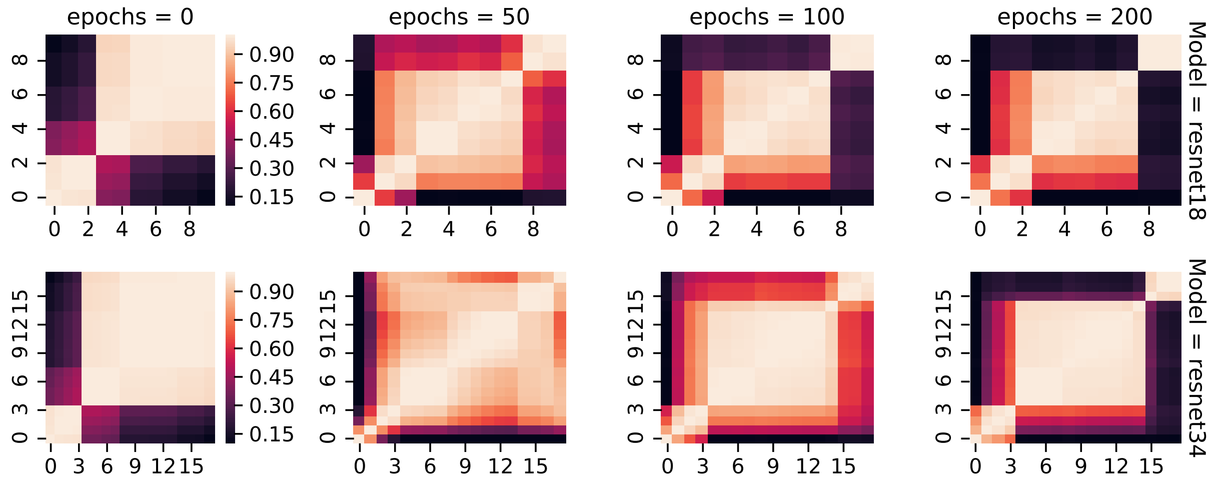


**Figure 2.5**: **Layerwise similarity, illustrated by SCKA, of a ResNet model (ResNet18 or ResNet34) during training on CIFAR100.** At the beginning, the layers are generally grouped into two sets by the SCKA values. During the course of training, we observe that the grouping becomes more prominent as individual layers gradually become specialised. It takes longer for the relatively deeper model, ResNet34, to reach the final grouping, whilst the shallower one, ResNet18, reaches a stable grouping very early. It implies that deeper models may take longer to converge, but may be capable of exploring more solutions.

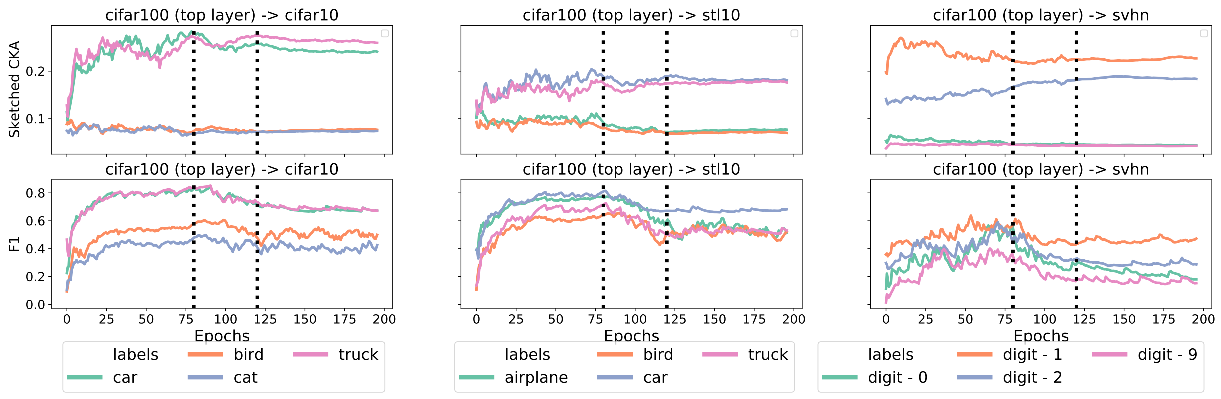


**Figure 2.6**: **SCKA vs. F1 score of one-vs-all classification tasks on each dataset.** Each dataset is decomposed into one-vs-all binary classification tasks, and both SCKA and F1 score are illustrated during the course of training a ResNet18 model on CIFAR100. The set illustrated labels of each dataset contains two with highest SCKA values at the end of training and two lowest.
We can observe that, the higher the SCKA value is, the higher the F1 score is. In addition, the network trained on CIFAR100 helps the network to separate out cars and trucks easily on both CIFAR10 and STL10.

In Stage 3, the SCKA and the accuracy reach a higher level than they are in stage 1 on CIFAR100, and a similar level on CIFAR10, but a lower level on STL10. Our interpretation is that STL10 data comes from downsampled images from ImageNet [DDS$^+$09], while CIFAR10 and CIFAR100 are from a similar data domain. Both values stay at a low level on SVHN as the dataset is drastically different from other datasets.

The SCKA values between individual layers and the targets $\rho_S(\boldsymbol{A}_l\boldsymbol{A}_l^\top, \boldsymbol{Y}\boldsymbol{Y}^\top)$, where $l \in \{1,2,...,L\}$, are plotted in Fig. 2.4. The general observation is that higher layers are more aligned with the targets than the lower layers are, but exceptions also exist. Lower layers tend to converge faster than the higher layers do on CIFAR100, CIFAR10, and STL10. An unexpected observation is that, on SVHN, the SCKA value between features from layer-2 with the target gradually decays across the training. A potential explanation for this observation is that, even though bottom layers are considered to be learning more general filters than abstract concepts, the training dataset still matters in determining the generality of learnt filters.

Fig. 2.5 presents cross-layer similarity scores measured by $\rho_S(\boldsymbol{A}_i\boldsymbol{A}_i^\top, \boldsymbol{A}_j\boldsymbol{A}_j^\top)$, where $i, j \in \{1,2,...,L\}$. Both ResNet-18 and ResNet-34 models group layers in a certain fashion, and the grouping becomes prominent during the training. This observation is consistent with [KNLH19]. However, at the 50-th epoch, ResNet-34 model still expresses high alignment scores between bottom layers and top layers, whilst ResNet-18 already reaches a stable grouping of layers that is similar to the grouping at the end of training. It suggests that a model with more layers tends to stablise slower, but it is potentially capable of exploring more diverse solutions before the training reaches the end.

One can consider a multi-class classification task as a set of one-vs-all binary classification tasks, and analysing the progress of each task during training the multi-class task provides insights on the difficulty of each binary classification task. Fig. 2.6 shows, during the training on CIFAR100, the change of SCKA and F1 score of each binary task from CIFAR10, STL10 and SVHN. Consistently, on CIFAR10, and STL10, cars and trucks are easier to classify compared to

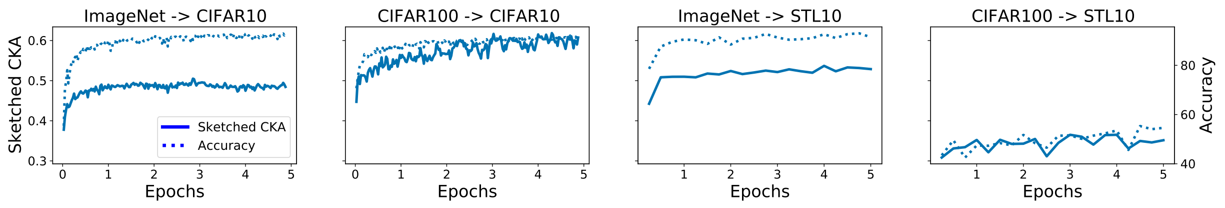


**Figure 2.7**: **SCKA of the top layer features vs. Accuracy during fine-tuning.** The network pretrained on ImageNet or CIFAR100 is fine-tuned on either CIFAR10 and STL10 for five epochs as the performance becomes stabilised. It is observable that SCKA tracks Accuracy during fine-tuning, which supports the argument that SCKA is indeed a decent and reliable predictor of the Accuracy.

other classes as the binary tasks have higher F1 scores, which our proposed SCKA is insync with. It helps us understand which classes the neural network chooses to learn from first, and which classes are intrincially different.

### 2.4.2 Fine-tuning from Pretrained Models

Pretrained models from either ImageNet or CIFAR100 are fine-tuned on CIFAR10 and STL10 for five epochs. During the fine-tuning, both SCKA of the top layer features and targets, $\rho_S(\boldsymbol{A}_L\boldsymbol{A}_L^\top, \boldsymbol{Y}\boldsymbol{Y}^\top)$, and the accuracy of each task are recorded. Fig. 2.7 presents the curves. It is clear that our proposed SCKA keeps track of the trend of accuracy closely. We also measured SCKA along with F1 scores of binary classification tasks with labels selected in the previous section. FIg. 2.8 shows that our SCKA is still capable of distinguishing easy tasks from hard ones.

### 2.4.3 Sketched CKA as Transferability

Previous experiments show that our proposed SCKA is insync with the accuracy of a given task during training or fine-tuning of a model. To further illustrate that the SCKA can serve as a measure for transferability of the top layer features of a neural network on a larger scale, we conduct a similar experiment as proposed in [NHSA20]. We sample a subset of classes from

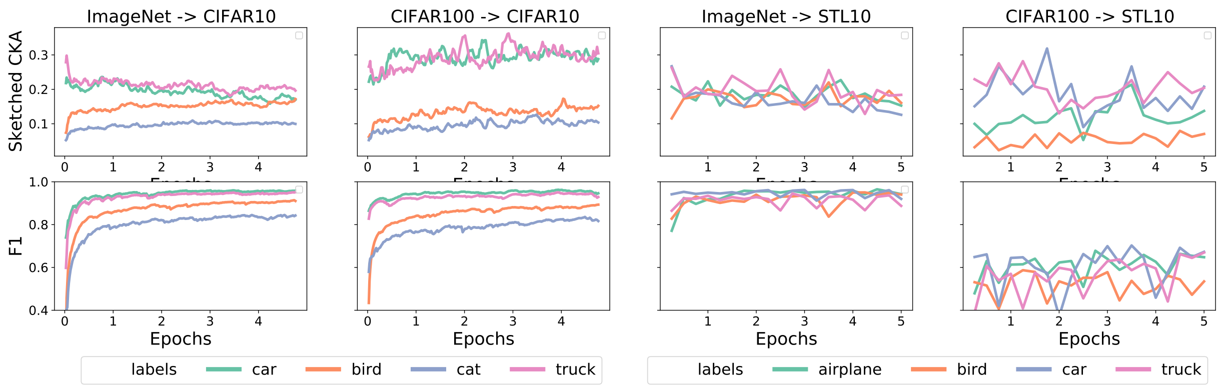


**Figure 2.8**: Same labels as the ones in Fig. 2.6 are selected, and the changes of SCKA and F1 of binary classification tasks using the top layer features during fine-tuning a pretrained model are presented. When the downstream task is CIFAR10, our SCKA on both models pretrained on ImageNet and CIFAR100 are capable of differentiating easier tasks from the harder ones. On STL10, as it is a subset of downsampled ImageNet images, the ImageNet pretrained model performs overall better than the CIFAR100 pretrained one. Our SCKA indicates that individual classes are categorised equally well on ImageNet, but the difference appears when the training data is CIFAR100.

one of the three tasks used in our experiments, including CIFAR10, CIFAR100 and STL10, to compose a new task, and then use one of the publicly available ResNet models pretrained on ImageNet in PyTorch [PGM$^+$19], including ResNet-18, -34, -50 and -101, to extract features for the sampled task.

Fig. 2.9 shows the linear relationship between the $\log \rho_S(\boldsymbol{A}_L\boldsymbol{A}_L^\top, \boldsymbol{Y}\boldsymbol{Y}^\top)$ and the accuracy of using the top layer features for making predictions using a simple linear model. It implies that CKA can be regarded as a reliable predictor of transferability, and our proposed SCKA is a computationally efficient substitute.

Our proposed SCKA achieves similar functionality as LEEP [NHSA20]. However, the LEEP score fails when the target dataset is only a shift of the source dataset when all samples are on the one side of the decision boundary, whilst ours doesn't [TN20].

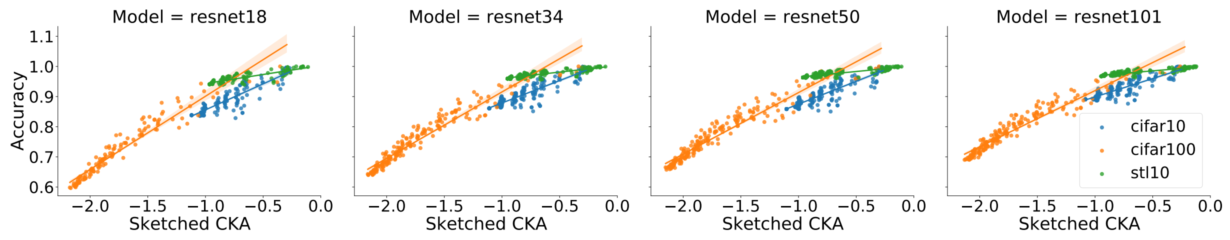


**Figure 2.9**: **SCKA vs. Accuracy of randomly subsampled tasks.** A task is composed of randomly subsampled classes from one the three tasks, including CIFAR10, CIFAR100, and STL10, and the accuracy and the logarithm of SCKA value of the top layer features and the targets using a pretrained ImageNet model is visualised as a dot in the plots. It shows the linear relationship between the log of SCKA and the accuracy using a pretrained ImageNet model on a downstream task. As the pretrained model goes deeper with the number of layers (from ResNet18-34-50-101), the cluster of dots gradually move upwards w.r.t. the same log-SCKA value. It matches our expectation that deeper models generalises better with the ResNet architecture.

**Table 2.1**: **Results of Transfer Learning.** Two ImageNet models, 150 models with random subsets of data from CIFAR10, CIFAR100, and SVHN are used as feature extractors, none of which have seen the full training sets. SCKA leverages the property of the algorithm [CMR12] for selecting models, and when irrelevant feauture extractors are presented, the algorithm is capable of pruning them for making predictions. Sketching helps create a fixed-size data summary for each dataset given each model which is independent from the number of data samples in each task, and it helps boost the speed of the algorithm for learning kernels.

| Pretraining Tasks / Transfer Tasks | STL10 | CIFAR10 | CIFAR100 | SVHN | FashionMNIST |
|---|---|---|---|---|---|
| ImageNet | 96.06 | 86.48 | 67.31 | 47.97 | 85.68 |
| subsets of CIFAR10 | 82.17 | 95.02 | 49.22 | 56.70 | 84.81 |
| subsets of CIFAR100 | 75.91 | 79.80 | 79.08 | 57.65 | 86.77 |
| subsets of SVHN | 53.00 | 58.49 | 28.82 | 97.00 | 86.16 |
| All models | 96.14 | 94.90 | 79.12 | 97.02 | 86.86 |

### 2.4.4 Transfer Learning

Given a large number of pretrained models, we now demonstrate that our Sketched CKA can be combined with the learning kernel algorithm in [CMR12] to leverage all pretrained models efficiently. The learning kernel algorithm aims to find an optimal set of nonnegative weights on individual kernel matrices so that the weighted sum of kernel matrices gives the maximum CKA score with the target, and it can be formally written as

$$\boldsymbol{\mu}^\star = \underset{\{||\boldsymbol{\mu}||=1 \wedge \boldsymbol{\mu} \geq 0\}}{\arg\max} \ \langle \boldsymbol{K}, \boldsymbol{Y}\boldsymbol{Y}^\top \rangle_F / ||\boldsymbol{K}||_F, \tag{2.5}$$

where $\boldsymbol{K} = \sum_{k=1}^{p} \mu_k \boldsymbol{K}_k$. Given Proposition 9 in [CMR12], the aforementioned optimisation problem is equivalent to the following one with normalisation.

$$\boldsymbol{v}^\star = \arg\min_{\boldsymbol{v} \geq 0} \boldsymbol{v}^\top \boldsymbol{M} \boldsymbol{v} - 2\boldsymbol{v}^\top \boldsymbol{a}, \tag{2.6}$$

where $\boldsymbol{M}_{k,l} = \langle \boldsymbol{K}_k, \boldsymbol{K}_l \rangle_F$, $\boldsymbol{a}_k = \langle \boldsymbol{K}_k, \boldsymbol{Y}\boldsymbol{Y}^\top \rangle_F$ and $p$ is the number of kernel matrices. The memory complexity of this algorithm is $O(Nd)$, where $d = \sum_{k=1}^{p} d_k$ and $d_k$ is the dimension of feature vectors produced from the $k$-th pretrained model, and the time complexity for a forward computation is at least $O(pN^2)$.

When a large number of pretrained models are presented, one can regard each one of them as a function that gives a fixed dimensional vector for a given sample. Combined with our proposed SCKA, $\tilde{\boldsymbol{M}}_{k,l} = \langle \boldsymbol{S}\boldsymbol{K}_k\boldsymbol{S}^\top, \boldsymbol{S}\boldsymbol{K}_l\boldsymbol{S}^\top \rangle_F = ||\boldsymbol{X}_k^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{X}_l||_F^2$ and $\tilde{\boldsymbol{a}}_k = \langle \boldsymbol{K}_k, \boldsymbol{Y}\boldsymbol{Y}^\top \rangle_F = ||\boldsymbol{X}_k^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{Y}||_F^2$. As proposed in the previous section, if $\boldsymbol{S}$ is implemented as a stack of $q$ CountSketch methods with $M$ sketches, the time complexity becomes $O(qNd + pM^2)$. The memory complexity becomes $O(Md)$. When $N \gg d_k$, sketching helps to reduce both the computational and memory complexity.

In our experiments, the model zoo contains 152 pretrained model, among which two are released ResNet-18 and ResNet-34 models pretrained on ImageNet, and 50 models are pretrained on subsets of one the three datasets, CIFAR10, CIFAR100, and SVHN. Beside these three datasets, STL10 and FashionMNIST are included for evaluating our transfer learning approach. To avoid any trivial solution, we make sure that none of the pretrained models have seen the full training set of any dataset in our study, so that during transfer, each model has to handle unseen data samples.

The results are presented in Table 2.1. SCKA combined with the learning kernel alignment algorithm can select feature vectors from useful ones as the performance of using all models is similar to that of using only models pretrained with data subsampled from the same dataset. The contribution of SCKA is the drastic reduction on the memory cost of storing features from models, and the computation of $\tilde{\boldsymbol{M}}$ and $\tilde{\boldsymbol{a}}$.

## 2.5 Details of Concepts

### 2.5.1 Properties of sketches

**Definition** *CountSketch [Woo14]*: Initialise $\boldsymbol{S} = \boldsymbol{0} \in \mathbb{R}^{m \times n}$. For every column $i$ of $\boldsymbol{S}$ choose a row $h(i)$ uniformly at random. Set $\boldsymbol{S}_{h(i),i}$ to either $+1$ or $-1$ with equal probability. Then $\boldsymbol{S} = \boldsymbol{BD}$ with $\boldsymbol{B} \in \mathbb{R}^{m \times n}$ is a matrix selecting the hash buckets assigned to a row of the input and $D$ is a diagonal matrix of Rademacher random variables.

We also require the following definitions:

**Definition** *Johnson-Lindenstrauss Transform (JLT)*: A matrix $\boldsymbol{S} \in \mathbb{R}^{k \times n}$ is a JLT with parameters $\varepsilon$, $\delta$, $f$, written $JLT(\varepsilon, \delta, f)$ if, with probability at least $1 - \delta$, for any $V \subset \mathbb{R}^n$ of size $f$:

$$\forall \boldsymbol{v}, \boldsymbol{v}' \in V : |\langle \boldsymbol{v}, \boldsymbol{v}' \rangle - \langle \boldsymbol{Sv}, \boldsymbol{Sv}' \rangle| \leq \varepsilon \|\boldsymbol{v}\| \|\boldsymbol{v}'\| \tag{2.7}$$

Our bound applies for any sketch which is a $JLT(\varepsilon, \delta, f)$. However, in practice, we instead implement the CWT as it is extremely fast to apply. Technically, the CWT does not ensure a $JLT(\varepsilon, \delta, f)$ in the worst case. This is because it preserves the norm of $f = 2^{\Omega(d)}$ points lying in a $d$-dimensional subspace rather than an arbitrary set of $f$ points. Further details can be found on pages 16-17 of [Woo14]. One sparse transform which achieves our bound is the *Sparse Johnson-Lindentrauss Transform* of [NN13] which can be thought of as $s$ stacked CWT matrices of height $m/s$ for a projection dimension $m$. Our implementation explored several settings for $s$, however, $s = 1$ performed well-enough and was extremely fast to apply. This is the reason we use the CWT in practice but prove our bounds for general JLT matrices.

In spite of this, there remain some strong theoretical reasons why we expect the CWT to perform favourably. This is because it still preserves all directions which make up the column space of the input matrix, as formalised below.

**Definition** *Subspace Embedding*: A matrix $\boldsymbol{S}$ which ensures:

$$(1-\varepsilon)\|\boldsymbol{Ax}\|_2^2 \leq \|\boldsymbol{SAx}\|_2^2 \leq (1+\varepsilon)\|\boldsymbol{Ax}\|_2^2 \tag{2.8}$$

is called a $(1 \pm \varepsilon)$ *subspace embedding* for the column space of $A$. In addition, this requirement gives a pointwise guarantee on approximate singular values.

The main result we need is from [Woo14]:

**Theorem.** Let $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ be a matrix of full rank. Then for any $\delta \in [0,1]$, $\boldsymbol{S} \in \mathbb{R}^{m \times n}$ is a $(1 \pm \varepsilon)$-subspace embedding for the column space of $\boldsymbol{A}$ with probability $1-\delta$ provided $m = O(d^2/\delta\varepsilon^2)$. Furthermore, $\boldsymbol{SA}$ can be computed in time $O(\mathrm{nnz}(\boldsymbol{A}))$.

Note that the CountSketch has weak failure probability dependence on $\delta$ but we find in our applications this is not problematic. Also, despite the input matrices being dense, it is still quicker to stream through the input and hash the rows rather than invoking other sketches which are applied through more expensive methods such as explicit matrix product or Fast Fourier Transforms.

### 2.5.2 Training details of models

We followed the publicly available training procedure of Residual Network, and the details include that (1) the initial learning rate is 0.1, and it decays by a factor of 10 every 80 epochs; (2) each model is trained for 200 epochs; (3) the optimizer is stochastic gradient descent with batch size 128 and weight decay coefficient $10^{-4}$ for training each model[2].

Model training is done on V100 GPUs, and the main results in our paper including model comparisons and the study on the norm of kernel mean embedding are done on a single Titan 1080 GPU.

### 2.5.3 Hyperparameters in sketching

There are two main hyperparameters in sketching, the batch size *bs* and the sketching size $M$, which is named as the number of buckets in our main paper. The batch size doesn't have an impact on the results of sketching so we set it to be the largest possible to fit in a single Titan 1080.

For visualisation purposes, we found that $M = \{512, 2048\}$ presents similar trend. For transfer learning purposes, the performance increases as $\boldsymbol{M}$ becomes larger, but in our experiments, we set $M = 2048$.

### 2.5.4 Proof of the bound in Lemma 1

We slightly alter the notations to match the convention used in [Woo14]. Here, we set $\boldsymbol{X} \in \mathbb{R}^{N \times d_1}$, $\boldsymbol{Y} \in \mathbb{R}^{N \times d_2}$ and $\boldsymbol{S} \in \mathbb{R}^{M \times N}$. Let $\boldsymbol{S}$ be a $JLT(\varepsilon, \delta, f)$.

Given in [Woo14] that

$$\Pr\left[\left|\langle \boldsymbol{S}(\boldsymbol{X})_{\cdot i}, \boldsymbol{S}(\boldsymbol{Y})_{\cdot j}\rangle - \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle\right| \leq \varepsilon ||(\boldsymbol{X})_{\cdot i}||_2 ||(\boldsymbol{Y})_{\cdot j}||_2\right] \geq 1 - \delta \qquad (2.9)$$

[2]https://github.com/kuangliu/pytorch-cifar

The above error bound can be written as:

$$\langle \boldsymbol{S}(\boldsymbol{X})_{\cdot i}, \boldsymbol{S}(\boldsymbol{Y})_{\cdot j}\rangle = \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle \pm \varepsilon ||(\boldsymbol{X})_{\cdot i}||_2 ||(\boldsymbol{Y})_{\cdot j}||_2 \tag{2.10}$$

$$\langle \boldsymbol{S}(\boldsymbol{X})_{\cdot i}, \boldsymbol{S}(\boldsymbol{Y})_{\cdot j}\rangle^2 = \left(\langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle \pm \varepsilon ||(\boldsymbol{X})_{\cdot i}||_2 ||(\boldsymbol{Y})_{\cdot j}||_2\right)^2 \tag{2.11}$$

$$\begin{aligned} &= \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle^2 + \varepsilon^2 ||(\boldsymbol{X})_{\cdot i}||_2^2 ||(\boldsymbol{Y})_{\cdot j}||_2^2 \\ &\quad \pm 2\varepsilon \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle ||(\boldsymbol{X})_{\cdot i}||_2 ||(\boldsymbol{Y})_{\cdot j}||_2 \end{aligned} \tag{2.12}$$

According to the Cauchy-Schwartz inequality, we have $\langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle \leq ||(\boldsymbol{X})_{\cdot i}||_2 ||(\boldsymbol{Y})_{\cdot j}||_2$. Then the upper bound can be simplified as

$$\begin{aligned} \langle \boldsymbol{S}(\boldsymbol{X})_{\cdot i}, \boldsymbol{S}(\boldsymbol{Y})_{\cdot j}\rangle^2 \leq& \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle^2 + \varepsilon^2 ||(\boldsymbol{X})_{\cdot i}||_2^2 ||(\boldsymbol{Y})_{\cdot j}||_2^2 \\ &+ 2\varepsilon \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle ||(\boldsymbol{X})_{\cdot i}||_2 ||(\boldsymbol{Y})_{\cdot j}||_2 \end{aligned} \tag{2.13}$$

$$\leq \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle^2 + \varepsilon^2 ||(\boldsymbol{X})_{\cdot i}||_2^2 ||(\boldsymbol{Y})_{\cdot j}||_2^2 + 2\varepsilon ||(\boldsymbol{X})_{\cdot i}||_2^2 ||(\boldsymbol{Y})_{\cdot j}||_2^2 \tag{2.14}$$

$$\leq \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle^2 + (\varepsilon^2 + 2\varepsilon) ||(\boldsymbol{X})_{\cdot i}||_2^2 ||(\boldsymbol{Y})_{\cdot j}||_2^2 \tag{2.15}$$

Similarly, the lower bound can be simplified as:

$$\begin{aligned} \langle \boldsymbol{S}(\boldsymbol{X})_{\cdot i}, \boldsymbol{S}(\boldsymbol{Y})_{\cdot j}\rangle^2 \geq& \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle^2 + \varepsilon^2 ||(\boldsymbol{X})_{\cdot i}||_2^2 ||(\boldsymbol{Y})_{\cdot j}||_2^2 \\ &- 2\varepsilon \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle ||(\boldsymbol{X})_{\cdot i}||_2 ||(\boldsymbol{Y})_{\cdot j}||_2 \end{aligned} \tag{2.16}$$

$$\geq \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle^2 + \varepsilon^2 ||(\boldsymbol{X})_{\cdot i}||_2^2 ||(\boldsymbol{Y})_{\cdot j}||_2^2 - 2\varepsilon ||(\boldsymbol{X})_{\cdot i}||_2^2 ||(\boldsymbol{Y})_{\cdot j}||_2^2 \tag{2.17}$$

$$\geq \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle^2 + (\varepsilon^2 - 2\varepsilon) ||(\boldsymbol{X})_{\cdot i}||_2^2 ||(\boldsymbol{Y})_{\cdot j}||_2^2 \tag{2.18}$$

Now we have:

$$\langle \boldsymbol{S}(\boldsymbol{X})_{\cdot i}, \boldsymbol{S}(\boldsymbol{Y})_{\cdot j}\rangle^2 = \langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle^2 + (\varepsilon^2 \pm 2\varepsilon) ||(\boldsymbol{X})_{\cdot i}||_2^2 ||(\boldsymbol{Y})_{\cdot j}||_2^2 \tag{2.19}$$

We sum over $d_1$ columns in $\boldsymbol{X}$ and $d_2$ columns in $\boldsymbol{Y}$

$$\sum_{i=1}^{d_1}\sum_{j=1}^{d_2}\langle \boldsymbol{S}(\boldsymbol{X})_{\cdot i}, \boldsymbol{S}(\boldsymbol{Y})_{\cdot j}\rangle^2 = \sum_{i=1}^{d_1}\sum_{j=1}^{d_2}\langle (\boldsymbol{X})_{\cdot i}, (\boldsymbol{Y})_{\cdot j}\rangle^2 + \sum_{i=1}^{d_1}\sum_{j=1}^{d_2}(\varepsilon^2 \pm 2\varepsilon)||(\boldsymbol{X})_{\cdot i}||_2^2||(\boldsymbol{Y})_{\cdot j}||_2^2 \tag{2.20}$$

$$||\boldsymbol{X}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{Y}||_F^2 = ||\boldsymbol{X}^\top \boldsymbol{Y}||_F^2 + (\varepsilon^2 \pm 2\varepsilon)||\boldsymbol{X}||_F^2||\boldsymbol{Y}||_F^2 \tag{2.21}$$

Since $||\boldsymbol{X}||_F^2||\boldsymbol{Y}||_F^2 \geq ||\boldsymbol{X}^\top \boldsymbol{Y}||_F^2$, the bound can be tightened by

$$||\boldsymbol{X}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{Y}||_F^2 = (1 \pm \varepsilon)^2||\boldsymbol{X}^\top \boldsymbol{Y}||_F^2 \tag{2.22}$$

Therefore,

$$\Pr\left[\Big| ||\boldsymbol{X}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{Y}||_F - ||\boldsymbol{X}^\top \boldsymbol{Y}||_F \Big| \leq \varepsilon ||\boldsymbol{X}^\top \boldsymbol{Y}||_F\right] \geq 1 - \delta \tag{2.23}$$

### 2.5.5 Proof of the bound in Lemma 2

By setting $\boldsymbol{X} = \boldsymbol{Y} = \Psi$ we can easily derive

$$\Pr\left[\Big| ||\Psi^\top \boldsymbol{S}^\top \boldsymbol{S}\Psi||_F - ||\Psi^\top \Psi||_F \Big| \leq \varepsilon ||\Psi^\top \Psi||_F\right] \geq 1 - \delta \tag{2.24}$$

Then we have:

$$||\boldsymbol{X}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{X}||_F||\boldsymbol{Y}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{Y}||_F = (1 \pm \varepsilon)^2||\boldsymbol{X}^\top \boldsymbol{X}||_F||\boldsymbol{Y}^\top \boldsymbol{Y}||_F \tag{2.25}$$

Therefore,

$$\left(\frac{1-\varepsilon}{1+\varepsilon}\right)^2 \rho \leq \rho_S \leq \left(\frac{1+\varepsilon}{1-\varepsilon}\right)^2 \rho \tag{2.26}$$

$$\text{where } \rho_S = \frac{||\boldsymbol{X}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{Y}||_F^2}{||\boldsymbol{X}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{X}||_F ||\boldsymbol{Y}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{Y}||_F}$$

$$\text{and } \rho = \frac{||\boldsymbol{X}^\top \boldsymbol{Y}||_F^2}{||\boldsymbol{X}^\top \boldsymbol{X}||_F ||\boldsymbol{Y}^\top \boldsymbol{Y}||_F}$$

Then the absolute difference between the Sketched CKA and the original CKA is bounded as shown below:

$$|\rho_S - \rho| \leq \max\{\frac{4\varepsilon}{(1+\varepsilon)^2}, \frac{4\varepsilon}{(1-\varepsilon)^2}\}\rho = \frac{4\varepsilon}{(1-\varepsilon)^2}\rho \tag{2.27}$$

Formally,

$$\Pr\left[|\rho_S - \rho| \leq \frac{4\varepsilon}{(1-\varepsilon)^2}\rho\right] \geq 1 - \delta \tag{2.28}$$

## 2.6 Acknowledgements

Chapter 2, in part, is currently being prepared for submission for publication of the material. Tang, Shuai; Maddox, Wesley J.; Dickens, Charlie; Diethe, Tom; Damianou, Andreas. “Sketched CKA: An Efficient Tool for Neural Network Representations”. The dissertation author was the primary investigator and author of this material. A preprint version of the material is available on ArXiv.

We thank Hisham Husain for helpful discussions and Jordan Massiah and Pablo Garcia Moreno for helping preparing the codebase.

# Chapter 3

# Deep Transfer Learning with Ridge Regression

## 3.1 Introduction

Besides scoring high on tasks they are trained on, neural networks have also excelled on tasks where datasets are collected from similar domains. Prior work [YCBL14] showed that filters/parameters learnt in DNNs pretrained on ImageNet generalise better with slight fine-tuning than those learnt from random initialisations. Since then, applications in Computer Vision have had major breakthroughs by initialising DNNs with pretrained parameters and fine-tuning them to adapt to new tasks. Similarly, Natural Language Processing (NLP) welcomed its "ImageNet Era" with large and deep pretrained language models including Bert [DCLT19], and performance on downstream NLP tasks has achieved state-of-the-art on a daily basis by employing more data and deeper models during pretraining, and using smarter methods for fine-tuning.

These advances in transfer learning using pretrained DNNs and fine-tuning, however, come with a large computational cost. An essential step to boost the performance on a given new task is to fine-tune the pretrained DNN until it converges, which is computationally intense since these models tend to have hundreds of millions of parameters. An alternative approach is to freeze the parameters and treat the pretrained DNN model as a feature extractor that produces abstracted vector representations which can be used to train a simple classifier that benefits from the pretraining knowledge. But as the parameters are not adapted to the new task, the latter approach provides inferior performance to fine-tuning.

We here propose a new way of augmenting the latter approach without fine-tuning the DNN. Our approach is to take an accumulation of feature vectors produced at different individual layers which encode different aspects of the data. Since feature vectors are highly correlated with each other, as they are generated from a single DNN, only a few are needed to make predictions. We adopt the alignment maximisation algorithm for combining kernels [CMR12], in which we first find a convex combination of linear kernels constructed from individual layers that gives maximal alignment with the target kernel constructed from one-hot encoding of the labels. Then, we take the ensemble of feature vectors of layers selected by non-zero elements in the sparse combination, and make predictions using kernel ridge regression (KRR) with approximation.

## 3.2 Related Work

Transfer learning with classical machine learning methods has been studied for a couple of decades [PY10], including boosting [DYXY07], support vector machines [MBS13], ridge regression [CM11], etc. These methods benefit from the transparency of classical machine learning models, and from their capacity as universal function approximators with strong theoretical guarantees However, it is not easy to incorporate structural priors, such as human knowledge, into regularising the learning process. This information is crucial in advancing machine learning systems.

Neural networks are also universal function approximators [HSW89], and learnt vectorised representations are generalisable across tasks, with recent advances in architecture designs specifically for individual types of inputs, including convolutional layers for image recognition [LBBH98], recurrent layers [Elm90, HS97] and transformer modules [VSP$^+$17], etc. Recent research has demonstrated that deep models pretrained on large amounts of data generalise well on unseen data sampled from relevant domains [YCBL14] by fine-tuning. With growing depth of networks, the cost for fine-tuning becomes non-negligible. Efforts in knowledge distillation from deep models to shallow ones [BC14, HVD15], [FH17] showed that neural networks can be simplified after learning, although the learnt transferable features can be potentially detrimented during distillation.

Our approach takes the best of both worlds by using feature vectors produced from multiple layers of a pretrained neural network without explicit fine-tuning, and makes predictions with KRRs for a new task. With help from low-rank approximations, our approach only requires passing the training data once through a neural network without backpropagation.

## 3.3 Method

The key concept is to apply KRR with a few layers of feature vectors produced from a pretrained neural network to make predictions, classification in our case, on a downstream task. Our notation includes: $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ is the data matrix with $N$ samples with each sample in $d$-dimensional space, $\boldsymbol{Y} \in \mathbb{R}^{N \times c}$ is the corresponding labels with one-hot encoding, $\boldsymbol{X}_l \in \mathbb{R}^{N \times d_l}$ contains flattened feature vectors as rows produced at the $l$-th layer of a neural network, $\boldsymbol{S} \in \mathbb{R}^{M \times N}$ is a random projection matrix that meets the requirement of subspace embedding with $M \ll N$, $\boldsymbol{I}_p \in \mathbb{R}^{p \times p}$ is an identity matrix, $L$ is the number of layers in a pretrained neural network, and $\alpha$ is the regularisation term in ridge regression. Other notation will be introduced as needed.

### 3.3.1 Low-rank Approximation at Layers

Flattened feature vectors generated from neural networks are generally high-dimensional and redundant, [1] Theoretical work [UT19] showed that big data matrices are generally low rank, and they can be approximated with a few directions without much information loss. Therefore, we use random projections to obtain low-rank approximations of feature vectors with many fewer dimensions. Given that the Nyström method is well-studied in approximating large-scale kernel matrices [GM16], we follow the formula to approximate a linear kernel $\boldsymbol{X}_l\boldsymbol{X}_l^\top$ as

$$\boldsymbol{X}_l\,(\boldsymbol{S}\boldsymbol{X}_l)^\top\,(\boldsymbol{S}\boldsymbol{X}_l\boldsymbol{X}_l^\top\boldsymbol{S}^\top)^\dagger\,(\boldsymbol{S}\boldsymbol{X}_l)\,\boldsymbol{X}_l^\top \tag{3.1}$$

where $\dagger$ is the pseudo-inverse of a square matrix. If $M \ll d$, which is mostly the case for feature vectors generated from neural networks, and the eigendecomposition is written as $(\boldsymbol{S}\boldsymbol{X}_l\boldsymbol{X}_l^\top\boldsymbol{S}^\top)^\dagger = \boldsymbol{Q}_l\Lambda_l\boldsymbol{Q}_l^\top$, then the low-rank approximation of $\tilde{\boldsymbol{X}}_l$ can be obtained by

$$\tilde{\boldsymbol{X}}_l = \boldsymbol{X}_l\,(\boldsymbol{S}\boldsymbol{X}_l)^\top\,\boldsymbol{Q}_l\Lambda_l^{-0.5} \tag{3.2}$$

with each sample in at most $M$-dimensional space. As we aim to conduct layer-wise low-rank approximations, it is preferable to apply sparse random projections instead of dense ones. Therefore, we consider a stack of $s$ independent CountSketch [CW13] which is equivalent to the sparse Johnson-Lindenstrauss Transformation [Woo14]. Compared to a safe dense random projection matrix filled with samples from a Gaussian random variable which requires a matrix multiplication for reducing the dimension, the time complexity of CountSketch is independent from the projection dimension $M$, and this desired property a significant amount of computation on top of the forward pass of a neural network.

In CountSketch, the random projection matrix $\boldsymbol{S}$ is considered as a hash table that uni-

[1] Fig. 3.8 shows the output dimensions of individual residual blocks in four ResNet models.

formly hashes $N$ samples into $M$ buckets with a binary value randomly sampled from $\{+1,-1\}$ so there is no need to materialise $\boldsymbol{S}$. Successful applications of CountSketch including polynomial kernel approximation [PP13] and large-scale regressions are due to its scalability with theoretical guarantees when few hash tables are used [Woo14]. Generally, larger $s$ leads to better approximations, yet the performance improvement becomes marginal. Prior work [Jag19] showed empirically that $s=4$ works well on real-world datasets; thus, we set $s=4$ which drastically reduces the cost for low-rank approximations at individual layers. The time complexity of Nyström is $O((sN+M^2+NM)d_l+(M+N)M^2)$.

With limited GPU memory, producing feature vectors for a downstream task given a pretrained neural network is often done in batches of samples. CountSketch is also well-suited in this situation as, technically, the approximation can be done in only one forward pass. At the end of this stage, one can obtain a set of low-rank approximations of feature vectors at individual layers $\{\tilde{\boldsymbol{X}}_l\}_{l=1}^L$, the memory complexity is at most $O(LNM)$. We now use the alignment maximisation algorithm to determine a smaller set of layers to use for the task.

### 3.3.2 Convex Combination of Layers

As feature vectors at individual layers arise from consecutive transformations from previous layers, the same input data will have many redundant features across layers. Thus we aim to select only a few layers that give the maximum alignment with the target. Specifically, a vector $\boldsymbol{\mu}=[\mu_1,\mu_2,...,\mu_L]^\top$ is optimised to maximise the following alignment [CMR12]:

$$\boldsymbol{\mu}^\star = \underset{\{||\boldsymbol{\mu}||=1\wedge\boldsymbol{\mu}\geq 0\}}{\arg\max} \langle \boldsymbol{K},\boldsymbol{Y}\boldsymbol{Y}^\top\rangle_F/||\boldsymbol{K}||_F \tag{3.3}$$
$$\text{where } \boldsymbol{K}=\textstyle\sum_{l=1}^L \mu_l \tilde{\boldsymbol{X}}_l \tilde{\boldsymbol{X}}_l^\top$$

Proposition 9 in [CMR12] showed that it is equivalent to the quadratic programming problem: $\boldsymbol{v}^\star = \arg\min_{\boldsymbol{v}\geq 0} \boldsymbol{v}^\top\boldsymbol{M}\boldsymbol{v}-2\boldsymbol{v}^\top\boldsymbol{a}$, where $\boldsymbol{a}_l=||\tilde{\boldsymbol{X}}_l^\top\boldsymbol{Y}||_F^2$ and $\boldsymbol{M}_{k,l}=||\tilde{\boldsymbol{X}}_k^\top\tilde{\boldsymbol{X}}_l||_F^2$, then $\boldsymbol{\mu}^\star=\boldsymbol{v}^\star/||\boldsymbol{v}^\star||$.

Intuitively, $\boldsymbol{a}_l$ measures the linear alignment between the low-rank feature vectors $\tilde{\boldsymbol{X}}_l$ and the targets $\boldsymbol{Y}$, and $\boldsymbol{M}_{k,l}$ indicates the linear alignment between two sets of feature vectors produced at the $k$-th and $l$-th layer.

Non-zero entries in $\boldsymbol{\mu}^\star$ provide a weighted sparse combination of feature vectors from a few layers that gives the highest linear alignment with targets. Note that there is no specific regularisation to control the sparsity of $\boldsymbol{\mu}^\star$, but due to the associations across layers, the resulting vector is extremely sparse as illustrated in our experiments. The time complexity is dominated by materialising $\boldsymbol{M} \in \mathbb{R}^{L\times L}$, which is $O(L^2M^2N)$ at worst.

The kernel $\boldsymbol{K}$ induces an embedding space which is a concatenation of feature vectors weighted by $\mu_l^{1/2}$, then the optimisation problem in Eq. 3.4 can be written in a weight-space perspective:

$$\begin{aligned} \boldsymbol{\mu}^\star &= \underset{\{||\boldsymbol{\mu}||=1,\boldsymbol{\mu}\geq 0\}}{\arg\max} \; ||\boldsymbol{X}_\phi^\top \boldsymbol{Y}||_F^2 / ||\boldsymbol{X}_\phi^\top \boldsymbol{X}_\phi||_F \\ \text{where } \boldsymbol{X}_\phi &= [\mu_1^{\frac{1}{2}}\tilde{\boldsymbol{X}}_1, \mu_2^{\frac{1}{2}}\tilde{\boldsymbol{X}}_2, ..., \mu_L^{\frac{1}{2}}\tilde{\boldsymbol{X}}_L] \end{aligned} \tag{3.4}$$

It is worth noting that the objective is not the "goodness-of-fit" measure for linear regression, $R^2$ statistics $||\boldsymbol{X}_\phi^\top \boldsymbol{Q}_Y||_F^2/||\boldsymbol{X}_\phi||_F^2$, where $\boldsymbol{Q}_Y$ contains eigenvectors of $\boldsymbol{Y}\boldsymbol{Y}^\top$. Optimising $\boldsymbol{\mu}$ to maximise $R^2$ will lead to drastic overfitting by accumulating all layers, and subsequently meaningless $\boldsymbol{\mu}$. One can equate maximising $R^2$ as setting the off-diagonal terms in $\boldsymbol{M}$ to 0, and optimising the same objective. Therefore, it considers individual layers to be independent from each other, and it encourages the algorithm to select all layers to maximise $R^2$, which leads to overfitting.

The objective in Eq. 3.4 finds a convex combination of features that maximises the alignment between the subspace spanned by the concatenated features and that by the one-hot encoded label space, so it prevents $\boldsymbol{X}_\phi$ from accumulating more feature vectors once an optimal subset is obtained. Therefore, the alignment-based objective prevents overfitting to a certain degree.

### 3.3.3 Kernel Ridge Regression

Since we aim to use the closed-form solution in ridge regression for transfer learning, to increase the capacity of our model, an RBF kernel function is applied on top of the selected features from the neural network.

We denote $L_s \ll L$ as the number of layers with positive $\mu_l$. Since, in the end, the predictions are made by kernel ridge regression, if $N$ is of a manageable order, then there is no need to conduct kernel approximation through Nyström. However, low-rank approximation can potentially help reduce the noise in data, which leads to a better generalisation compared to computing the exact kernel function.

| condition | prediction $\boldsymbol{y}$ |
|---|---|
| $N \gg M_s$ | $\tilde{\boldsymbol{x}}_\psi(\tilde{\boldsymbol{X}}_\psi^\top \tilde{\boldsymbol{X}}_\psi + \alpha \boldsymbol{I}_{M_s})^{-1}\tilde{\boldsymbol{X}}_\psi^\top \boldsymbol{Y}$ |
| $N \ll M_s$ | $\tilde{\boldsymbol{x}}_\psi \tilde{\boldsymbol{X}}_\psi^\top(\tilde{\boldsymbol{X}}_\psi \tilde{\boldsymbol{X}}_\psi^\top + \alpha \boldsymbol{I}_N)^{-1}\boldsymbol{Y}$ |

We consider approximating an RBF kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-||\boldsymbol{x}_i - \boldsymbol{x}_j||^2/2\sigma^2)$ with the Nyström method using the same subsampling in Sec. 3.3.1, CountSketch, to further promote fast computation on accumulated feature vectors $\boldsymbol{X}_\phi$. We denote the number of buckets in $m$ hash functions as $M_s$, then the time complexity of this step is $O((m + M_s)NM + 2M_s^2 M + M_s^3)$. Since $N \gg M$ and $N \gg M_s$, the dominating term in the complexity is $M_s NM$. The hyperparameter $\sigma^2$ is heuristically set to $\max\{||\boldsymbol{x}_i||^2/2 : i = 1, 2, ..., N\}$. One could cross-validate $\sigma^2$ as well, however, for the sake of reducing the complexity of transfer learning, we stick to the heuristic value. The approximated low-rank feature map of an RBF function is denoted as $\tilde{\boldsymbol{X}}_\psi \in \mathbb{R}^{N \times M_s}$.

**Table 3.1**: **Dataset details.** Individual cell indicates (# Training Samples / # Test Samples / [# Classes]).

| Training | In-domain Transfer | | | Out-of-domain Transfer | | |
|---|---|---|---|---|---|---|
| ImageNet | CIFAR10 | CIFAR100 | STL10 | SVHN | CUB200 | Kuzushiji49 |
| 1.2m / - [1000] | 50k / 10k [10] | 50k / 10k [100] | 5k / 8k [10] | 73k / 26k [10] | 6k / 6k [200] | 116k / 38k [49] |

**Table 3.2**: **Results of supervised transfer learning.** Median accuracy of five trials is reported in each cell, and each cell has two accuracy terms of transferring from [ResNet-18 / ResNet-34]. Except for CUB200, our method outperforms LogReg significantly since the variance of five trials is very small as presented in figures.

| | CIFAR10 [In] | CIFAR100 [In] | STL10 [In] | CUB200 [Out] | SVHN [Out] | Kuzushiji49 [Out] |
|---|---|---|---|---|---|---|
| LogReg | 87.45 / 89.94 | 69.08 / 72.76 | 95.08 / 96.55 | 60.80 / 61.60 | 64.36 / 59.47 | 74.56 / 71.08 |
| Ours | 90.77 / 92.31 | 71.31 / 74.63 | 96.30 / 97.31 | 58.78 / 61.70 | 88.76 / 88.53 | 88.12 / 88.00 |

Given a new data sample $\tilde{\boldsymbol{x}}_{\psi}$, the prediction is given the closed-form solution of ridge regression in the table above.

Then the label of a test sample is the index of the maximum value in predicted $\boldsymbol{y}$. The time complexity of ridge regression is determined by the inverse of a square matrix and the matrix multiplication that gives the square matrix, and it is $\min(N^3 + M_sN^2, M_s^3 + NM_s^2)$.

In summary, our proposed method has three steps including 1) **CountSketch** to obtain low-rank feature vectors at individual layers to a manageable size, 2) **convex combination** to take weighted accumulation of feature vectors, and 3) **KRR** with Nyström approximation to make predictions. Compared to multiple forward and backward passes required in fine-tuning or training classifiers, our method drastically reduces the computational cost.

## 3.4 Experiments

We demonstrate the effectiveness of our method through experiments on transfering ResNet-based models [HZRS15, HZRS16] pretrained on the ImageNet dataset [DDS+09, RDS+15] to downstream tasks, including three in-domain datasets, CIFAR-10, CIFAR-100 [Kri09], STL10 [CNL11], and three out-of-domain ones, Street View House Number (SVHN) [NWC+11], Caltech-UCSD-200 (CUB200) [WBW+10], Kuzushiji49 [CBIK+18][2]. Basic statistics of each dataset are presented in Table 3.1, and their descriptions are included in Sec. 3.7.1. Experiments are done in PyTorch [PGC+17].

**Table 3.3**: **Performance of three methods using the same time budget.** "Time / seconds" row reports the running time of our method including hyperparameter tuning and test runtime on 4 large datasets with 2 pretrained models. Three bottom rows report the accuracy of each method using the same time budget. It needs to be noted that only half of the running time of our method is on GPU (Titan1080), and the rest is on CPU (i9-7900X @ 3.30GHz). In comparison, we allow all three other fine-tuning methods to use the full time period. The value in the parentheses is the test runtime.

| | CIFAR10 (In-domain) | | CIFAR100 (In) | | SVHN (Out) | | Kuzushiji49 (Out-of-domain) | |
|---|---|---|---|---|---|---|---|---|
| ResNet-layer | 18 | 34 | 18 | 34 | 18 | 34 | 18 | 34 |
| Running Time / seconds | 456.5 | 487.8 | 485.3 | 517.8 | 522.5 | 573.2 | 1474.83 | 1911.32 |
| Accuracy (Ours) | 90.77 (15.4) | **92.31** (16.6) | 71.31 | **74.63** | **88.76** | **88.53** | **88.12** (52.55) | **88.00** (61.11) |
| Acc (top 1 layer) | 85.50 | 87.15 | 64.75 | 69.36 | 53.95 | 50.10 | 55.98 | 51.24 |
| Acc (top 3 layers) | **91.70** | 90.09 | **73.18** | 70.26 | 79.92 | 67.70 | 80.75 | 69.43 |
| Acc (all layers) | 85.38(12.66) | 85.12(16.12) | 58.13 | 45.52 | 85.94 | 83.71 | 86.24(47.94) | 81.86(60.32) |

**Hyperparameter Settings**: We report results with $M = \{512, 1024, 2048\}$, and $M_s = 2M$. ResNet-18 and ResNet-34 pretrained on ImageNet are selected as base models to transfer from. We only hash feature vectors from every residual block in a model. The regularisation strength $\alpha$ is cross-validated on the training set of the downstream task with values ranging from $\{1e-1, 1e-2, 1e-3, 1e-4\}$.

**Comparison Partner**: **Fine-tuning the top layer on each downstream task with soft-**

[2]The full Kuzushiji49 dataset has 232k training images, whick takes too long to cross-validate hyperparameters for LogReg. Thus, the same half of the dataset is used.

**max regression.** Models are fine-tuned for 30 epochs with Adam optimiser, and the learning rate decays by a factor of 2 every 10 epochs. Cross validation is conducted to optimise the following hyperparameters and their associated values: data augmentation=$\{$with, without$\}$, weight decay rate=$\{1e-3, 1e-4, 1e-5\}$, initial learning rate=$\{1e-3,\ 2.5e-4\}$. Note that fine-tuning with data augmentation tremendously increases the training time as the neural network needs to be kept during fine-tuning, while for other methods where data augmentation is disabled, one can store feature vectors from the last layer prior to fine-tuning. Results are marked with **LogReg** in the following tables and figures.

**Trials**: Since our method involves random projections and comparison partners require initialisation, for fair comparison, we run each method five times with different random seeds, and each marker in each plot presents the mean of five trials along with a vertical bar indicating the standard deviation. It is noticeable that vertical bars are often invisible as hyperparameters of each method are cross-validated on the training set. The main results are presented in Tab. 3.2.

### 3.4.1 Time-Constrained Comparison

Since measuring the time complexity of fine-tuning a pretrained deep learning model on a downstream task is non-trivial, we empirically compare the performance of our method against fine-tuning different numbers of layers using the same time budget, and it is the total clock time on a single computing machine used by our method. It is worth noting that only the first half of our method is required on a GPU, and the rest is done on a CPU. However, for the simplicity of comparisons, we allow the fine-tuning methods to take the full time on a GPU. The results are presented in 3.3. It is interesting that, under the same time budget, our method outperforms fine-tuning on four out of six cases, which shows that our method is capable of reducing the cost of transfer learning by reducing the GPU runtime.

It is indeed that fine-tuning is the most straightforward and simple method for transfer learning, but the caveat is that fine-tuning also involves many hyparameters, including learning rate, batch size, regularisation, etc. Meanwhile, our method only has two hyparameters, projection dimension $M$ and regularisation in ridge regression $\alpha$. We do not claim that our proposed method is absolutely superior to fine-tuning as we aim to provide an alternative approach to spark new directions for transfer learning, even without fine-tuning.

### 3.4.2 Supervised Transfer Learning

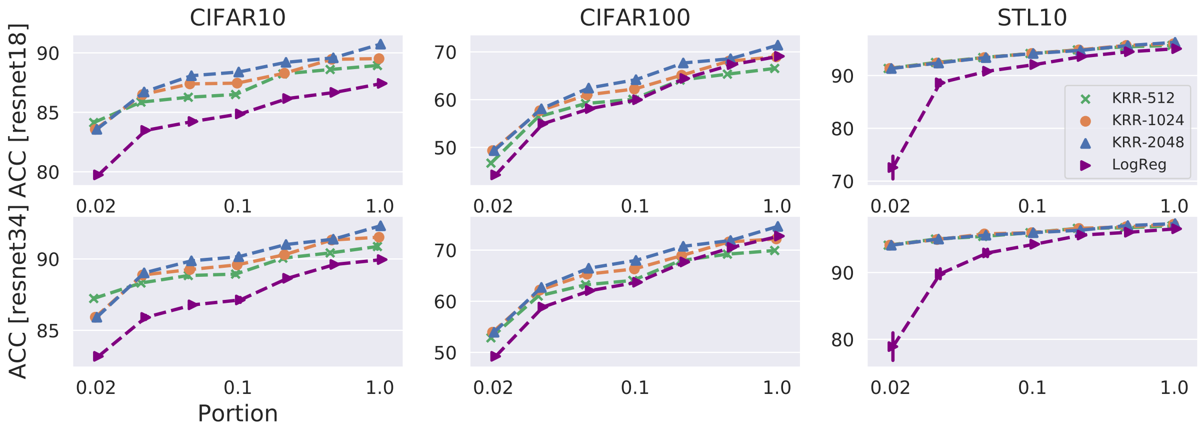
Since individual downstream tasks have ample samples in the training set, it encourages us to study our method and comparison methods when varying the portion of training samples. Specifically, the kept portion of training samples varies from 2% to 100%, and the interval is determined linearly in the log-space. The results are presented in Fig. 3.1.

Our method for $M = \{1024, 2048\}$ outperforms fine-tuning the top layer on five out of six transfer tasks with different portions of training samples, and only performs relatively similar to fine-tuning on CUB200, which is a finegrained bird species recognition task.

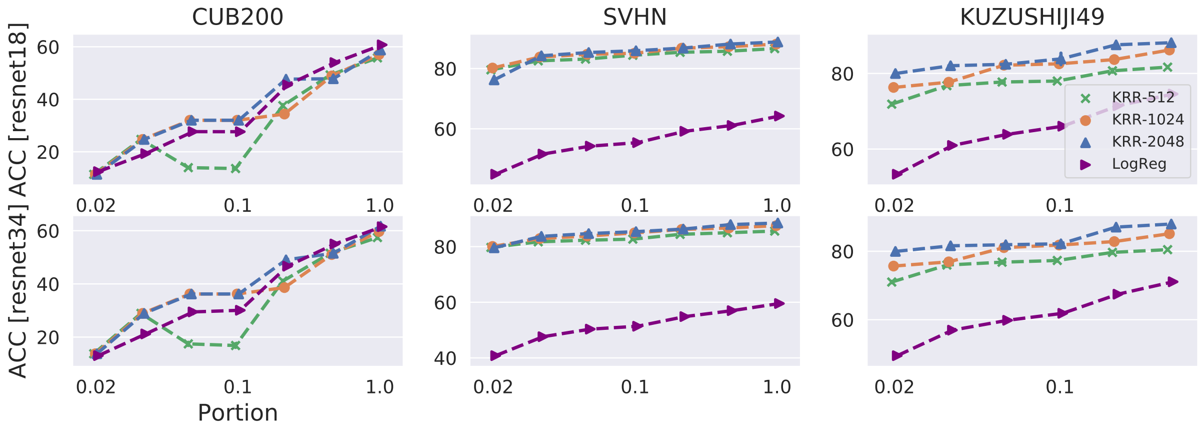
Fig. 3.2 presents results of our method on two variants of ResNet models, including ResNeXt [XGD$^+$17] and Wide-ResNet [ZK16]. Our method can leverage the generalisation ability of a pretrained neural network and scale well as the number of layers increases.

Our method adopted the Nyström method for low-rank approximation of feature vectors at individual layers, which involves hashing $N$ data samples into $M$ buckets first, then solving a linear system, and the hash functions can be applied across all layers. A more direct approach is to hash individual features in each feature vector into $M$ buckets as in $\boldsymbol{X}_l\boldsymbol{S}_l \in \mathbb{R}^{N\times M}$. This approach eliminates the step of solving a linear system, which reduces the time complexity to $O(sMd_l)$ for each layer. The comparison between Nyström and random projection is presented in Fig. 3.3.

Overall, Nyström provides better accuracy across all six tasks than random projection. However, it is noticeable that the difference between the two is smaller on in-domain transfer tasks. The observation also serves as a piece of supporting evidence that our method is relatively



(a) In-domain transfer tasks



(b) Out-of-domain transfer tasks

**Figure 3.1**: **Supervised transfer with varying portions of training samples from the transfer task.** Except for CUB200, our method with all three $M$'s generalises better than LogReg does (purple lines in plots) when the portion of training samples varies from 2% to 100%, and the observation is consistent across two different depths of ImageNet models.

consistent when different low-rank approximation schemes are applied.

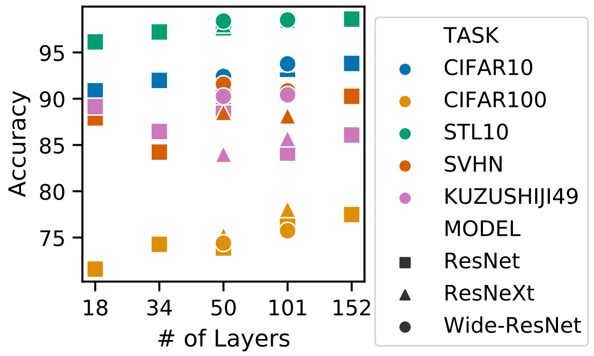
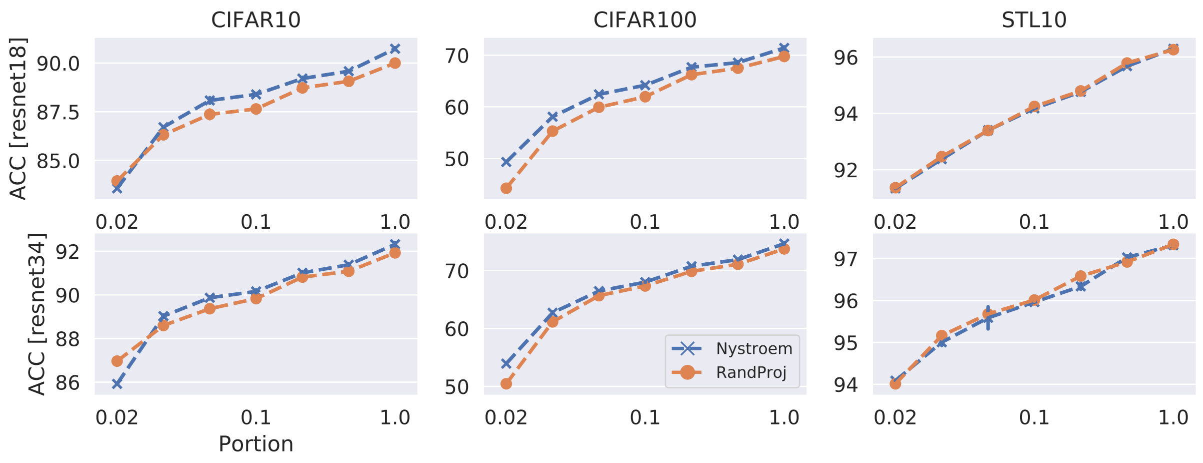
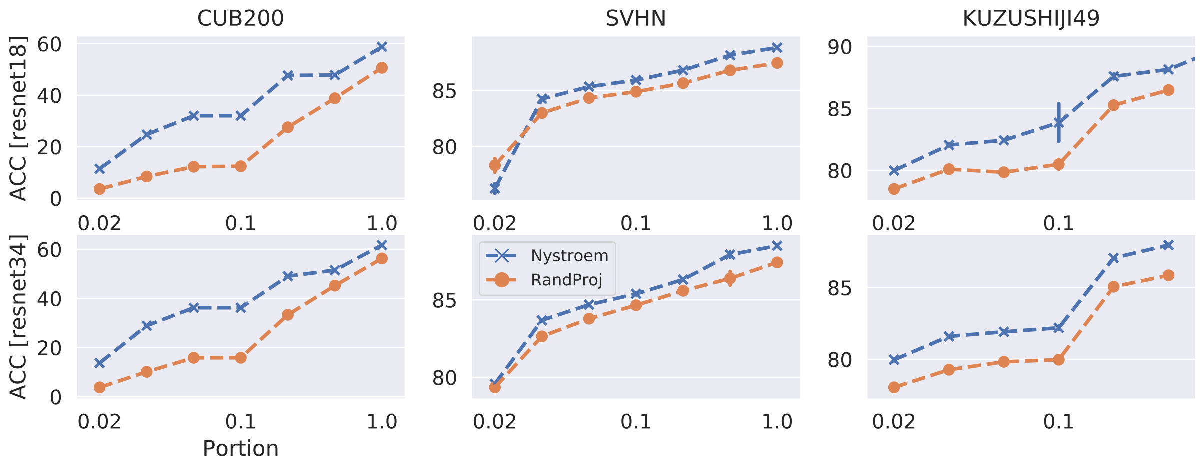


**Figure 3.2**: Our method on three variants of ResNet models with various numbers of layers.

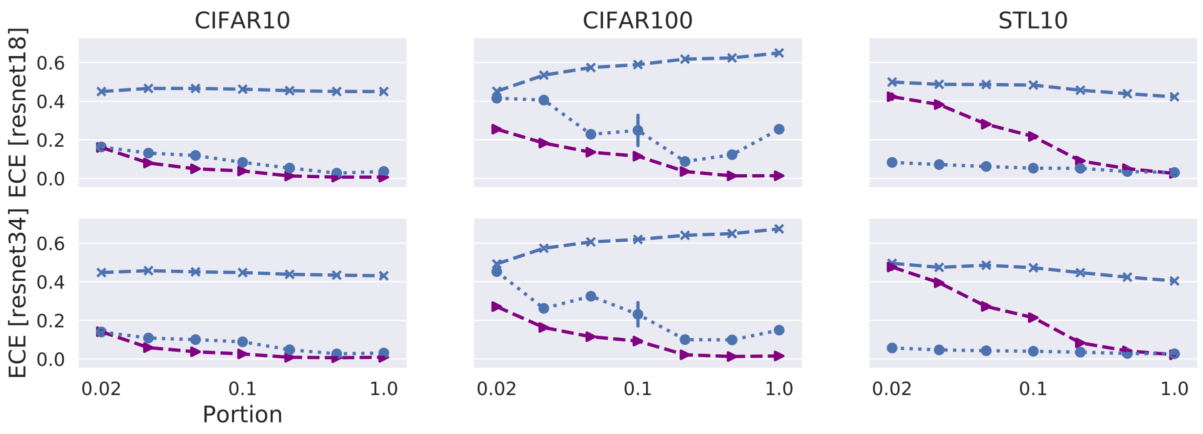


(a) in-domain transfer tasks

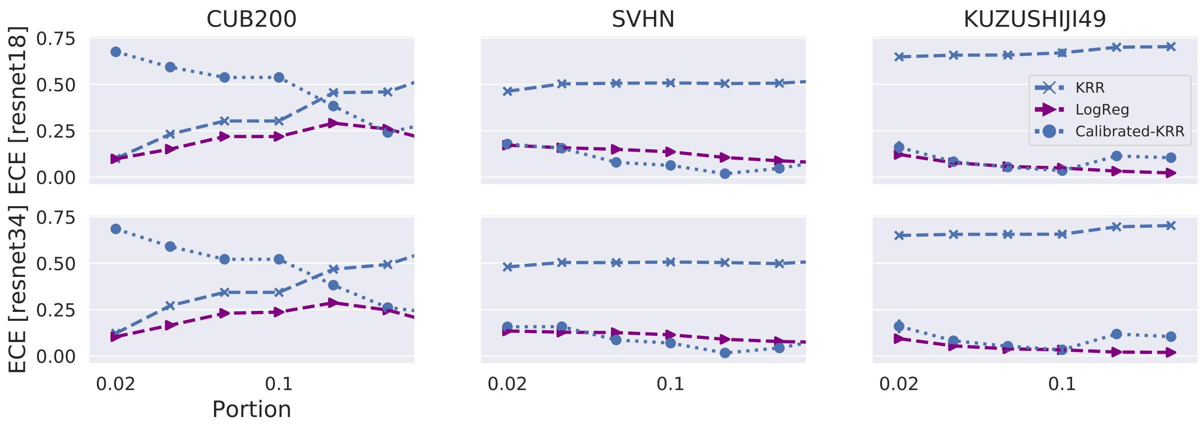


(b) out-of-domain transfer tasks

**Figure 3.3**: **Nyström vs. Random Projection in the first step of our method.** The performance improvement of Nyström over random projection is relatively larger on out-of-domain transfer tasks that it is on in-domain ones, and the observation is consistent across varying portions of training data.



(a) in-domain transfer tasks



(b) out-of-domain transfer tasks

**Figure 3.4**: **Expected Calibration Error (lower the better) with varying portions of training data.** LogReg provides better calibrated models than ours does. However, a posthoc adjustment by *Temperature Scaling* helps our method to match the calibration performance with LogReg.

Although our method provides both speed up and accuracy improvement in transfer learning, we are also interested in how well-calibrated our learnt classifier is compared to fine-tuning the top layer. It is expected that KRR, SVM and tree-based boosted classifiers are not well-calibrated as the objective is not probabilistic [NMC05].Expected Calibration Error (ECE) [GPSW17] is computed for our method, and baseline models - fine-tuning the top layer with logistic regression. The formula of ECE is given as ECE $= \sum_{c=1}^{C} \frac{B_c}{N} |\text{acc}(B_c) - \text{conf}(B_c)|$, where $N$ is the number of samples, and $C$ is the number of bins in the estimation.

The results shown in Fig. 3.4 validate our expectation that our method gives worse calibration on test set compared to logistic regression. A simple cure is **Temperature Scaling** [GPSW17], which optimises a parameter $t$ to rescale the output from the classifer to reduce the ECE on training data, and doesn't change the predicted labels. In our case, $t$ can be simply cross-validated efficiently.

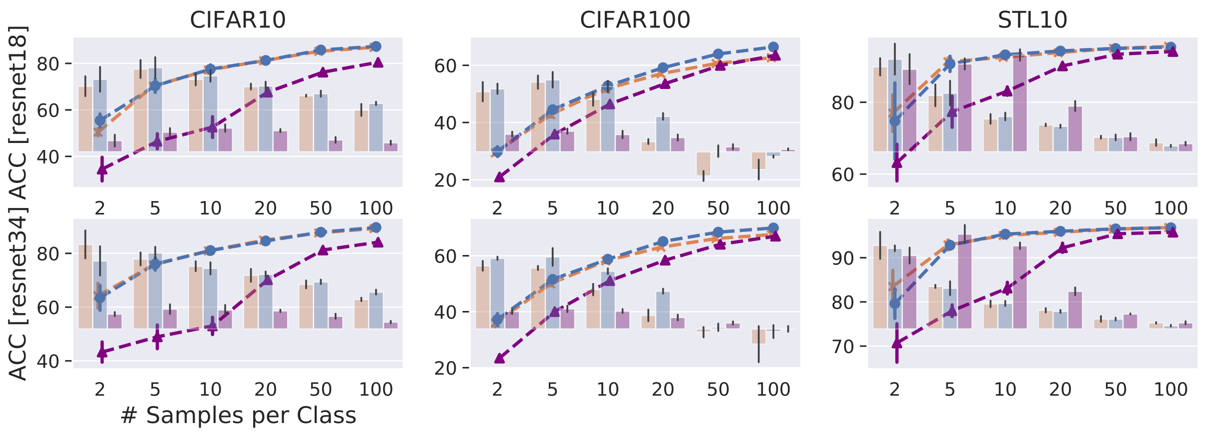
### 3.4.3 Semi-supervised Transfer Learning

There are many ways of incorporating unlabelled data into kernel ridge regression, including manifold regularisation [BNS06] and transductive learning [CM06]. Since manifold regularisation requires exact computation or an approximation of the Laplacian matrix on labelled and unlabelled samples, which leads to increased learning time, we adopted the transductive learning method for regression problems to leverage unlabelled data when extremely limited labelled training samples $\{\tilde{\boldsymbol{X}}_\psi, \boldsymbol{Y}\}$ and a large amount of unlabelled samples $\{\tilde{\boldsymbol{X}}'_\psi\}$ are provided. The solution of transductive ridge regression [CM06] is $\boldsymbol{W} = (\boldsymbol{A} + \boldsymbol{I}_{M_s})^{-1}\boldsymbol{B}$, where

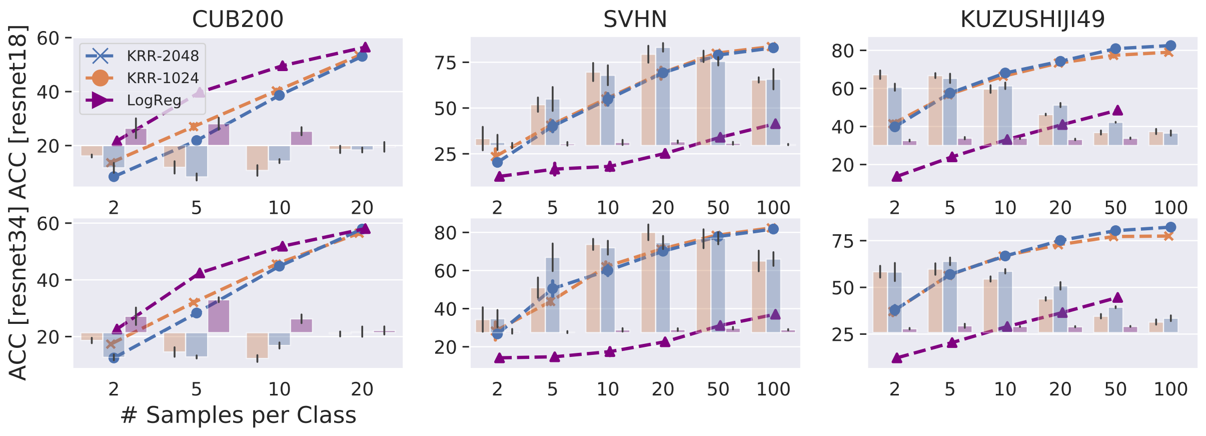
$$\boldsymbol{A} = \beta' \tilde{\boldsymbol{X}}_\psi'^\top \tilde{\boldsymbol{X}}'_\psi + \beta \tilde{\boldsymbol{X}}_\psi^\top \tilde{\boldsymbol{X}}_\psi, \boldsymbol{B} = \beta' \tilde{\boldsymbol{X}}_\psi'^\top \boldsymbol{Y}' + \beta \tilde{\boldsymbol{X}}_\psi^\top \boldsymbol{Y}$$

$\beta'$ and $\beta$ are hyparameters that control the contribution from unlabelled data and labelled data, which can be cross-validated on the labelled data. $\boldsymbol{Y}'$ comes from the ridge regression model learnt only on labelled data, and is given as $\boldsymbol{Y}' = g(\tilde{\boldsymbol{X}}'_\psi(\tilde{\boldsymbol{X}}_\psi^\top \tilde{\boldsymbol{X}}_\psi + \alpha \boldsymbol{I}_{M_s})^{-1} \tilde{\boldsymbol{X}}_\psi^\top \boldsymbol{Y})$, where $\alpha$ is a hyperparameter, and $g(\cdot)$ sets the maximum value of the vector prediction of a data sample to 1 and the rest to 0. For fine-tuning the top layer, we use the supervised classifier trained on labelled samples to annotate unlabelled data samples, and incorporate these samples into the training set and retrain the classifier with cross-validation.

We simulate a semi-supervised learning environment by keeping 2, 5, 10, 20, 50, or 100 labelled training samples per class, and leave the rest as unlabelled samples. Accuracy of semi-supervised transfer learning on the testset is reported in lineplots in Fig. 3.5, and the relative improvement against supervised transfer learning is reported in barplots in the same figure.



(a) in-domain transfer tasks



(b) out-of-domain transfer tasks

**Figure 3.5**: **Accuracy of semi-supervised learning with varying number of labelled samples per class.** The number of labelled examples changes from 1 to 100 per class and the rest are left unlabelled for semi-supervised learning. Left y-axis for line plots refers to the accuracy of semi-supervised learning, and right y-axis for bar plots refers to the relative improvement brought by unlabelled data. **(I)** Our method gives better performance than LogReg overall expect for CUB200. **(II)** Our method is also better at leveraging unlabelled samples for learning as indicated by taller bars for ours than LogReg expect for STL10 and CUB200.

Our method outperforms fine-tuning the last layer on five out of six transfer tasks indicated by the lineplots. Our method also gives significant relative improvement when unlabelled samples are incorporated through transductive regression on these five tasks as well, while unlabelled samples don't improve the performance for the top layer fine-tuning method. Negative results are concentrated on CUB200, where unlabelled samples become detrimental to our method while helpful for fine-tuning.

## 3.5 Insights provided by $\mu_l$

The solution to Eq. 3.4 indicates the number of accumulated layers and their weights. We plot a heatmap with y-axis indicating the index of layers, x-axis indicating the portion of training samples, and gradient colour scheme presenting the value of $\mu_l$ in Fig. 3.6. As shown

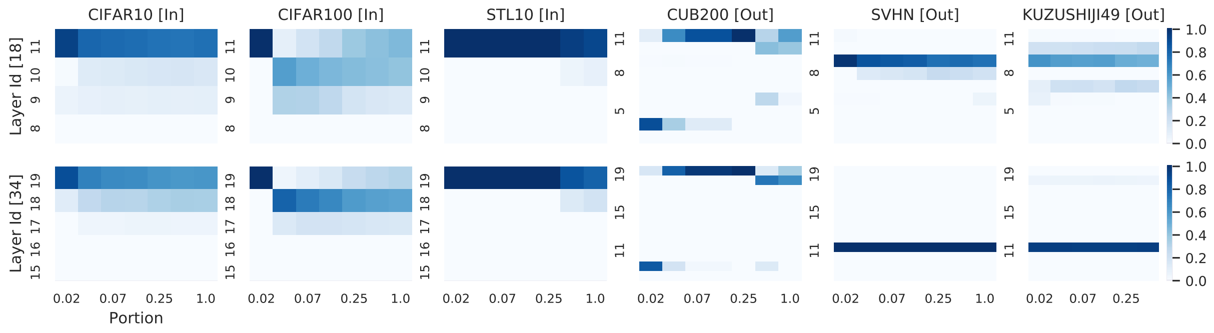


**Figure 3.6**: **Convex combination $\mu$ of layers vs. Varying portions of training samples.** In-domain transfer tasks assign higher values to top few layers, and out-of-domain ones prefer slightly lower layers.

in [HZRS16], the penultimate layer (index 11 for resnet18 and 19 for resnet34) of a ResNet removes all spatial information by averaging outputs. As illustrated in Fig. 3.6, layers before the penultimate layer have been assigned non-zero $\mu_l$'s across six tasks confirming that preserving spatial information helps in transfer learning.

For in-domain transfer tasks, it turns out that the top few layers are the most useful, and the improvement of our method is brought by the ability of identifying and accumulating these layers. STL10 contains images from the ImageNet dataset but with lower resolution, so the penultimate layer provides adequately abstract information of the images, which explains the observation that our method assigns a very dominant $\mu$ towards the penultimate layer.

For SVHN and Kuzushiji49, clearly, the selected layers don't include the feature vectors generated from the penultimate layer, and lower layers give higher $\mu$, which results in better performance than fine-tuning the top linear layer. However, our method doesn't provide better performance compared to fine-tuning the last layer on CUB200. A potential explanation comes from the fact that kernel ridge regression learns one-vs-all classifiers, and it is suitable when many classes are presented. This is a limitation of our method, but also a research direction for future study.

### 3.5.1 Accumulated vs Individual Layers

As $\boldsymbol{X}_\phi$ in our method is a weighted concatenation of feature vectors from layers with non-zero $\mu_l$, it is important to conduct a sanity check on the effectiveness of accumulating layers compared to using these layers alone. Therefore, we gradually accumulate layers sorted by their $\mu_l$, and plot the performance curve versus the number of accumulated layers. Then these layers are applied individually to make predictions as a comparison. We use the full training dataset in this subsection. The results are shown in Fig. 3.7.

For in-domain transfer tasks, we see that the performance improves as our method accumulates layers, while the trend is not obvious/significant for out-of-domain transfer tasks. Overall, accumulating a few layers provides better performance than making predictions based on individual layers.

### 3.5.2 Accumulating All Layers

One can technically accumulate features from all layers without running the alignment maximisation algorithm described in Sec. 3.3.2 to select layers. However, the computational cost increases drastically as the total number of layers in a neural network is much larger than the number of selected layers with non-zero $\mu_l$.

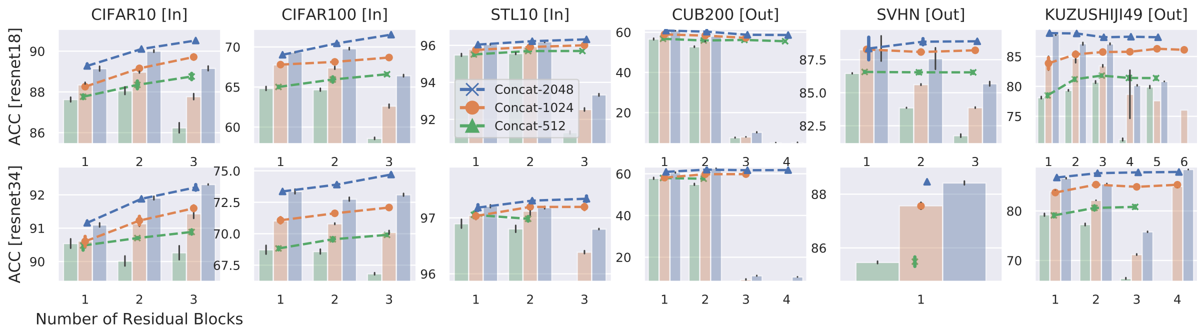


**Figure 3.7**: **Accuracy of accumulating Residual Blocks vs. that of individual Ones.** Line plots indicate accuracy of accumulating blocks until the exhaustion of non-zero $\mu_l$, and bar plots indicates the performance of these blocks separately. **(I)** In-domain transfer tasks demonstrate increasing accuracy when blocks are accumulated gradually, so does Kuzushiji49, which validates that accumulating layers helps. **(II)** Note that values $\{\mu_l\}_{l=1}^{L}$ don't directly imply the importance of layers, and that explains why the bar plots don't have a monotonic trend. When two layers are highly correlated with each other, the algorithm in Sec. 3.3.2 will only select the one with a smaller norm value. Even though both are important to making predictions, the algorithm learns to prune highly correlated layers.

The comparison between accumulating all layers and accumulating only selected layers is presented in Tab. 3.4. As discussed in the previous sections that the redundancy of information not only exists within individual layers, it also appears cross various layers. Therefore, using all layers provides similar performance as using a few selected layers, but requires longer running time.

**Table 3.4**: **Performance of KRR with low-rank features produced from all layers.** The value in each parenthesis indicates the ratio between the running time of using all layers and that of using only selected layers. As shown, the performance is similar to that of accumulating a few layers selected by the alginment maximisation algorithm, but accumulating all layers takes much longer to run.

| Tasks | CIFAR10 | CIFAR100 | STL10 |
|---|---|---|---|
| resnet18 | 90.5 (x3.8) | 71.2 (x3.7) | 95.5 (x1.4) |

## 3.6 Conclusion

We provided a promising Ridge Regression based transfer learning scheme for deep learning models. It doesn't require fine-tuning, which simplifies the transfer learning problem to simple regressions, and it is capable of identifying a few layers to accumulate for making better predictions.

We evaluated our method on supervised transfer with varying portions of training data, and handle semi-supervised transfer learning problems via transductive regression. Both show significant improvement compared to fine-tuning only the last layer. Discussions addressed the issue of calibration by Temperature Scaling, and demonstrate the superiority of Nyström over plain random projections.

Through the lense provided by $\{\mu_l\}_{l=1}^{L}$, we can see that target-related information has large overlap across layers so that accumulating selected layers is able to performance as well as using all layers, and ours provides higher computational efficiency.

## 3.7 Details of Experimental Design

### 3.7.1 Dataset Descriptions

**CIFAR10** [Kri09] consists of 60$k$ images, each of size $32 \times 32$. The train/test split is made available, and the training set contains 50$k$ images and the test set contains the rest. Each image has an object at the center, and the total 10 object categories are similar to ones in ImageNet dataset.

**CIFAR100** [Kri09] has 60$k$ images as well, each of size $32 \times 32$. The ratio of training images and test ones is the same as in CIFAR10. Each image has an object at the center, and in total, there are 100 object categories, which makes the task harder than CIFAR10.

**STL10** [CNL11] has 500 images for training and 800 images for testing per class, each of size $96 \times 96$ and in total, there are 10 classes. Since images of this dataset come from labelled samples from ImageNet but with lower resolution, models pretrained on ImageNet are expected to generalise well.

**SVHN** [NWC+11] consists of real-world images obtained from house numbers in Google Street View images, therefore, there are 10 categories. The training set contains $73,257$ images, and the test set contains $26,032$. The resolution of images is $32 \times 32$. The dataset also provides a set of $531,131$ unlabelled images, and we didn't make use of it in our study.

**CUB200** [WBW+10] is an image dataset with photos of 200 bird species (mostly North American). The total number of training images is $6,033$, therefore, each class has around 30 training examples. The task itself is considered to difficult as it requires the model to pay attention to details of the bird presented in each image, which makes it a fine-grained classification problem.

**Kuzushiji49** [CBIK+18] is a Japanese character recognition task, which contains 48 Hiragana characters and one Hiragana iteration mark. The dataset itself is much larger than aforementioned ones, and contains only gray-scale images. The training set contains $232,365$ images, and in our study, we only used half of the whole set. The test set contains $38,547$ images, which is used to evaluate the effectiveness of our method and other methods. The size of images is $28 \times 28$.

### 3.7.2 Results: RBF Baselines

RBF kernels as universal kernels are widely used in many research domains. Since we used features produced by neural network models learnt on the ImageNet dataset as inputs to an RBF kernel, it is reasonable to compare to the method that takes an ensemble of RBF kernels with various bandwidths and directly takes the vectorised images as inputs. Nyström approximation is applied to reduce the memory complexity.

Individual RBF kernels are selected as follows, and learning kernel alignment [CMR12] is also applied to find the optimal combination of RBF kernels with different bandwidths.

$$\begin{aligned} k_{ij,p} &= \exp(-||\boldsymbol{x}_i - \boldsymbol{x}_j||^2/(2^p\gamma)), \\ \text{where } \gamma &= \text{median}_{i,j\in\{1,2,...,N\}}||\boldsymbol{x}_i - \boldsymbol{x}_j||^2 \\ \text{and } p &\in \{-2,-1,...,10\} \end{aligned} \tag{3.5}$$

The results are presented in Tab. 3.5. Since RBF kernels are directly operating on pixels of images without neural networks, the performance is worse than our method or fine-tuning the top layer (LogReg). It serves as supporting evidence that inductive biases (prior knowledge) introduced by convolutional layers are important in image recognition tasks.

**Table 3.5**: Results of LogReg, our method and RBF kernels.

| Methods | CIFAR10 [In] | CIFAR100 [In] | STL10 [In] |
|---|---|---|---|
| LogReg | 87.45 / 89.94 | 69.08 / 72.76 | 95.08 / 96.55 |
| RBF | 51.40 | 21.76 | 43.58 |
| Ours | 90.77 / 92.31 | 71.31 / 74.63 | 96.30 / 97.31 |

### 3.7.3 Results: Transferring within In-domain tasks

We have three in-domain transfer tasks, and train a model for each task then evaluate its performance on other tasks using our method. The results are presented in Tab. 3.6. Overall, our method provides reasonable performance across tasks, and it doesn't involve fine-tuning the models. Specifically, for STL10, as the dataset itself has very few images in the training set, models trained on CIFAR10 and CIFAR100 give better generalisation on STL10 than those trained on STL10 itself.

**Table 3.6**: Transferring within In-domain tasks.

| Tasks for Pretraining | Transfer Tasks | | |
|---|---|---|---|
| | CIFAR10 | CIFAR100 | STL10 |
| CIFAR10 | 94.61 | 57.16 | 82.81 |
| CIFAR100 | 87.85 | 76.21 | 80.63 |
| STL10 | 73.72 | 43.65 | 67.85 |

### 3.7.4 Dimensionality

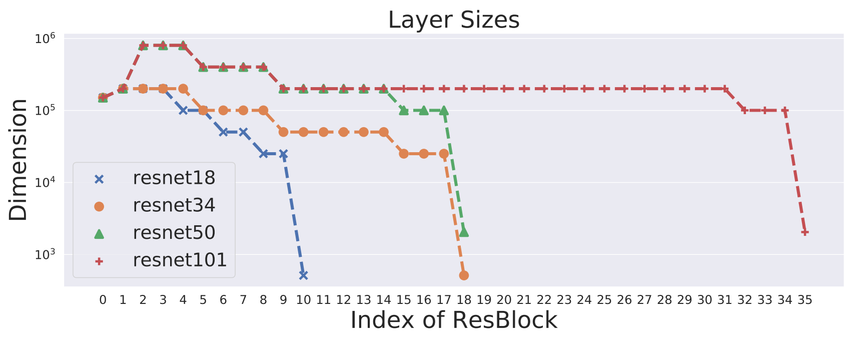


**Figure 3.8**: Dimensionality of each layer in ResNet models. Note that y-axis is on the log scale.

## 3.8 Acknowledgements

Chapter 3, in part, is currently being prepared for submission for publication of the material. Tang, Shuai; de Sa, Virginia R.. “Deep Transfer Learning with Ridge Regression”. The dissertation author was the primary investigator and author of this material. A preprint version of the material is available on ArXiv.

We gratefully thank Charlie Dickens and Wesley J. Maddox for fruitful discussions, and appreciate Mengting Wan and Shi Feng for comments on our draft.

# Chapter 4

# Ensemble Pruning with Linear Kernel Alignment

## 4.1 Introduction

Ensemble learning algorithms have enjoyed a great success prior to the era of deep learning, including bagging with bootstrapping [Bre96], boosting [FS97, Fri01] and random forests [Bre01]. Although many ensemble learning algorithms with traditional machine learning models as base learners have been surpassed by neural networks in certain application domains, they are still widely applied in analysing tabular data and in predictive modelling. In addition, the idea of taking an ensemble of a bag of deep learning predictors [SEZ+14, SZ15, HZRS15, XGD+17] is used extensively in previous champion and runner-up models for the ImageNet challenge [DDS+09]. Given the popularity of ensemble learning in modelling tabular data and its application in improving performance using multiple predictors, it is worth analysing their performance with respect to the number of predictors required.

We mainly discuss two ensemble learning techniques, which are bagging and boosting. Bagging makes predictions for a new sample by averaging predictions from multiple learners, and they are either trained from various sources that are relevant to the target or learnt with bootstrapped subsets of the same dataset so that individual learners' profiles are diversified. Specifically, in the second case, learners that are low-bias and high-variance are preferred. The effect of bagging on variance reduction makes it promising in achieving low regression error. Boosting iteratively fits weak learners — high-bias, but with higher-than-chance performance — to the dataset, and readjusts each learnt predictor's contribution at the end of each iteraction. The bias term in the Mean Squared Error is reduced by greedy fitting on the dataset, and the variance is also reduced by ensemble of all learnt predictors. In practice, both Gradient Boosting and AdaBoost have demonstrated strong performance.

Technically bagging and boosting take linear combinations of the learnt predictors to make final predictions on the test set, which naturally leads to an investigation on the number of predictors required to maintain the original performance. Reducing the number of predictors after learning accelerates the inference speed, and also leads to a reduction on the storage cost as pruned predictors are no longer needed and there is no need to compute and store their predictions. A simple way of pruning learnt predictors is to directly learn a set of non-negative weights on individual predictors so that the regression error is minimised, and this method strongly corresponds to Stacked Generalisation [Wol92] and Ensemble of Kernel Predictors [CMR11].

We propose that, in between learning the predictors and the non-negative weights on top of them, a kernel learning step is conducted. We consider each predictor as a function and use the predictions to construct a linear kernel. The optimisation step tries to find a convex combination of linear kernels so that the combined kernel has the maximum alignment with the target kernel. The idea originates from the algrithm for learning kernel alignment [CMR12] with non-negative constraints on the coefficients of kernels. In our experiments with bagging and boosting, we show that our proposed two step method achieves higher sparsity than the single-step does whilst improving the performance on top of the ensemble methods in most cases.

## 4.2 Method

Consider predictors $\{f_i\}_{i=1}^p$, each as a function that maps a sample $\boldsymbol{x}$ in the input space $\mathcal{X}$ to the target space $\mathcal{Y}$, are already learnt on the provided dataset $D = \{\boldsymbol{x}_n, y_n\}_{n=1}^N$, we consider the algorithm for learning kernels with Centered Kernel Alignment (CKA) [CM06] for pruning the predictors in order to obtain acceleration at inference time.

### 4.2.1 Centered Kernel Alignment

Given multiple kernel functions $\{k_i\}_{i=1}^p$, with each inducing a mapping $\phi_i$ in the RKHS, and $k_i(\boldsymbol{x}_n, \boldsymbol{x}_m) = \langle \phi_i(\boldsymbol{x}_n), \phi_i(\boldsymbol{x}_m) \rangle_{\mathcal{H}_i}$, one can evaluate the empirical kernel matrix $\boldsymbol{K}_i \in \mathbb{R}^{N \times N}$ on the same dataset $D$. The algorithm [CMR12] finds a set of weights $\{\mu_i\}_{i=1}^p$ so that the combined kernel matrix $\boldsymbol{K} = \sum_{i=1}^p \mu_i \boldsymbol{K}_i$ has the maximum CKA value with the target kernel $\boldsymbol{K}_Y = \boldsymbol{y}\boldsymbol{y}^\top$. Specifically, CKA is defined as follows:

$$\rho = \frac{\langle \boldsymbol{HKH}, \boldsymbol{K}_Y \rangle_F}{||\boldsymbol{HKH}||_F ||\boldsymbol{K}_Y||_F} \tag{4.1}$$

where $\boldsymbol{H}$ is the centering matrix. For simplicity, we assume that kernel matrices are centered, therefore, the objective function is defined as :

$$\boldsymbol{\mu}^\star = \arg\max_{\boldsymbol{\mu} \geq 0} \frac{\langle \boldsymbol{K}, \boldsymbol{K}_Y \rangle_F}{||\boldsymbol{K}||_F} = \arg\max_{\boldsymbol{\mu} \geq 0} \frac{\sum_{i=1}^p \mu_i \langle \boldsymbol{K}_i, \boldsymbol{K}_Y \rangle_F}{||\sum_{i=1}^p \mu_i \boldsymbol{K}_i||_F} \tag{4.2}$$

### 4.2.2 Linear Kernels and Pruning

To apply the algorithm for learning kernels on the problem of pruning ensemble models, one can directly set $\phi_i = f_i$, $\forall i \in \{1, 2, ..., p\}$, with linear kernels, therefore, $\boldsymbol{K}_i = \hat{\boldsymbol{y}}_i \hat{\boldsymbol{y}}_i^\top$, where $\hat{\boldsymbol{y}}_i \in \mathbb{R}^N$ contains predictions for training samples provided by the predictor $f_i$. The objective can be rewritten as :

$$\boldsymbol{\mu}^{\star} = \arg\max_{\boldsymbol{\mu}\geq 0} \frac{||\Pi\hat{\boldsymbol{Y}}^{\top}\boldsymbol{y}||_2^2}{||\Pi^{\top}\hat{\boldsymbol{Y}}^{\top}\hat{\boldsymbol{Y}}\Pi||_F} = \arg\max_{\boldsymbol{\mu}\geq 0} \frac{\sum_{i=1}^{p}\mu_i(\hat{\boldsymbol{y}}_i^{\top}\boldsymbol{y})^2}{\left(\sum_{i=1}^{p}\sum_{j=1}^{p}\mu_i\mu_j(\hat{\boldsymbol{y}}_i^{\top}\hat{\boldsymbol{y}}_j)^2\right)^{\frac{1}{2}}} \tag{4.3}$$

where $\hat{\boldsymbol{Y}} = [\hat{\boldsymbol{y}}_1, \hat{\boldsymbol{y}}_2, ..., \hat{\boldsymbol{y}}_p] \in \mathbb{R}^{N\times p}$ is a concatenation of predictions from all predictors, and $\Pi$ is a diagonal matrix with entries $\mu_i^{\frac{1}{2}}$. After optimisation, only predictors with non-zero weights are selected for making predictions, and predictors with zero weights are discarded.

### 4.2.3 Additional Pruning

Technically, once the set of $\{\mu_i\}_{i=1}^{p}$ is determined, one can directly use the combined kernel $\boldsymbol{K}$ for making predictions using kernel ridge regression with its dual form

$$\boldsymbol{y}_{\star} = \hat{\boldsymbol{y}}_{\star}\Pi\Pi^{\top}\hat{\boldsymbol{Y}}^{\top}(\hat{\boldsymbol{Y}}\Pi\Pi^{\top}\hat{\boldsymbol{Y}}^{\top} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}, \tag{4.4}$$

where $\hat{\boldsymbol{y}}_{\star}$ contain predictions from predictors with non-zero $\mu$s. By applying the Woodbury Matrix Identity , one can obtain the primal view of the kernel ridge regression solution,

$$\beta^{\star} = (\Pi^{\top}\hat{\boldsymbol{Y}}^{\top}\hat{\boldsymbol{Y}}\Pi + \lambda\boldsymbol{I})^{-1}\Pi^{\top}\hat{\boldsymbol{Y}}^{\top}\boldsymbol{y}, \tag{4.5}$$

which is effectively optimising the following objective with the remaining predictors:

$$\boldsymbol{\beta}^{\star} = \arg\min_{\boldsymbol{\beta}} ||\hat{\boldsymbol{Y}}\Pi\boldsymbol{\beta} - \boldsymbol{y}||_2^2 + \lambda||\hat{\boldsymbol{Y}}\Pi\boldsymbol{\beta}||_2^2 \tag{4.6}$$

where the diagonal terms of $\Pi$ are the solutions $\{\mu_i^{\star}\}_{i=1}^{p}$ to the optimisation problem defined in Eq. 4.3, which indicates which predictors to use in this step. To achieve further acceleration in the inference time, one can enforce $\beta$s to be non-negative, which results in the following optimisation problem:

$$\boldsymbol{\beta}^{\star} = \arg\max_{\boldsymbol{\beta}\geq 0} \frac{\boldsymbol{\beta}^\top \Pi^\top \hat{\boldsymbol{Y}}^\top \boldsymbol{y}}{||\hat{\boldsymbol{Y}}\Pi\boldsymbol{\beta}||_2^2} \tag{4.7}$$

The resulting number of predictors is equal to the number of non-zero entries in $\boldsymbol{\beta}^{\star}$, which is upper bounded by the number of non-zero entries in $\boldsymbol{\mu}^{\star}$. One can consider the first step — optimising $\boldsymbol{\mu}$ — as selecting predictors, and the second step — optimising $\boldsymbol{\beta}$ — as adjusting the contribution of each selected predictor with the potential of further reducing the overall number of selected predictors.

## 4.3 Analysis

The option of directly applying the aforementioned second step for selecting predictors comes naturally, and it corresponds to previous proposed ideas, including **Stacked Generalisation** [Wol92] and **Ensemble of Kernel Predictors** [CMR11]. The objective function is simply:

$$\boldsymbol{\mu}^{\star} = \arg\max_{\boldsymbol{\mu}\geq 0} \frac{\boldsymbol{\mu}^\top \hat{\boldsymbol{Y}}^\top \boldsymbol{y}}{||\hat{\boldsymbol{Y}}\boldsymbol{\mu}||_2^2} = \arg\max_{\boldsymbol{\mu}\geq 0} \frac{\sum_{i=1}^{p} \mu_i \hat{\boldsymbol{y}}_i^\top \boldsymbol{y}}{\sum_{i=1}^{p}\sum_{j=1}^{p} \mu_i \mu_j \hat{\boldsymbol{y}}_i^\top \hat{\boldsymbol{y}}_j} \tag{4.8}$$

For better understanding, Tab 4.1 presents three objective functions w.r.t. the inner-product values $\{\hat{\boldsymbol{y}}_i^\top \hat{\boldsymbol{y}}_j\}_{i,j=1}^{p}$ and $\{\hat{\boldsymbol{y}}_i^\top \boldsymbol{y}\}_{i=1}^{p}$. By substituting the inner-product values with their squared terms in the objective of Stacked Generalisation, one can recover the objective in the Linear Kernel Alignment. The two objective functions in Eq. 4.3 and Eq. 4.8 differ when predictors are producing similar predictions for the same set of data, which is a result of applying low-variance machine learning models as base learners. In this case, the squared inner-product values in Eq. 4.3 signify the individual differences, which results in a smaller number of predictors after learning than that produced by Stacked Generalisation. In the other case, when the selected base learner for ensemble learning is high-variance, as the individual differences are large enough, the selected predictors by both optimisation functions are similar.

**Table 4.1**: Optimisation Objectives.

| Linear Kernel Alignment | Stacked Generalisation | $R^2$ Maximisation |
|---|---|---|
| $\frac{\sum_{i=1}^{p}\mu_i(\hat{\boldsymbol{y}}_i^\top\boldsymbol{y})^2}{\left(\sum_{i=1}^{p}\sum_{j=1}^{p}\mu_i\mu_j(\hat{\boldsymbol{y}}_i^\top\hat{\boldsymbol{y}}_j)^2\right)^{\frac{1}{2}}}$ | $\frac{\sum_{i=1}^{p}\mu_i\hat{\boldsymbol{y}}_i^\top\boldsymbol{y}}{\sum_{i=1}^{p}\sum_{j=1}^{p}\mu_i\mu_j\hat{\boldsymbol{y}}_i^\top\hat{\boldsymbol{y}}_j}$ | $\frac{\sum_{i=1}^{p}\mu_i(\hat{\boldsymbol{y}}_i^\top\boldsymbol{y})^2}{\sum_{i=1}^{p}\mu_i\hat{\boldsymbol{y}}_i^\top\hat{\boldsymbol{y}}_i}$ |

Another option is to optimise the $R^2$-statistics, which is usually considered as the "goodness-of-fit" measure for linear regression models. The objective function is defined as:

$$\boldsymbol{\mu}^\star = \underset{\boldsymbol{\mu}\geq 0}{\arg\max}\frac{||\Pi^\top\hat{\boldsymbol{Y}}^\top\boldsymbol{y}||_2^2}{||\hat{\boldsymbol{Y}}\Pi||_F^2||\boldsymbol{y}||_2^2} = \underset{\boldsymbol{\mu}\geq 0}{\arg\max}\frac{||\Pi^\top\hat{\boldsymbol{Y}}^\top\boldsymbol{y}||_2^2}{||\hat{\boldsymbol{Y}}\Pi||_F^2} = \underset{\boldsymbol{\mu}\geq 0}{\arg\max}\frac{\sum_{i=1}^{p}\mu_i(\hat{\boldsymbol{y}}_i^\top\boldsymbol{y})^2}{\sum_{i=1}^{p}\mu_i\hat{\boldsymbol{y}}_i^\top\hat{\boldsymbol{y}}_i} \tag{4.9}$$

By comparing Eq. 4.9 and Eq. 4.3, it is clear that the numerators of both objectives are the same, however, the difference in the denominators plays an important role on the sparsity of $\boldsymbol{\mu}$. According to the Cauchy-Schwartz inequality, we have

$$\sum_{i=1}^{p}\mu_i\boldsymbol{y}_i^\top\boldsymbol{y}_i = ||\hat{\boldsymbol{Y}}||_F^2 \leq ||\hat{\boldsymbol{Y}}^\top\hat{\boldsymbol{Y}}||_F = \left(\sum_{i=1}^{p}\sum_{j=1}^{p}\mu_i\mu_j(\boldsymbol{y}_i^\top\boldsymbol{y}_j)^2\right)^{\frac{1}{2}}, \tag{4.10}$$

which means that the objective function in Eq. 4.3 penalises $\boldsymbol{\mu}$ stronger than that in Eq. 4.9. Given the other non-negativity constraint we emposed on $\boldsymbol{\mu}$, with the numerators in both objective functions having the same value, Eq. 4.3 selects fewer predictors than Eq. 4.9 does. Furthermore, directly maximising the $R^2$ score could lead to overfitting when the number of predictors presented is more than that of the data samples, especially when the combined predictions $\hat{\boldsymbol{Y}}$ is full row-rank.

Suppose that not all $\boldsymbol{\mu}$ are zeros, the equality of the aforementioned inequality holds in two cases. The first case is when $\hat{\boldsymbol{y}}_i = \hat{\boldsymbol{y}}$, $\forall i \in \{1,2,...,p\}$, and the denominators of both Eq. 4.3 and 4.9 reduce to $c||\boldsymbol{\mu}||_1$, where $c = ||\hat{\boldsymbol{y}}||_2^2$. In this case, all predictors are making the exact same predictions for training samples. Then both would select one predictor to make predictions.

The second case is when $\hat{\boldsymbol{y}}_i^\top \hat{\boldsymbol{y}}_j = 0, \forall i \neq j$ and $i, j \in \{1, 2, ..., p\}$. In this case, the optimisation problem is equivalent to projecting the target $\boldsymbol{y}$ into the space spanned by basis vectors $\{\frac{\boldsymbol{y}_i}{||\boldsymbol{y}_i||}\}_{i=1}^{p}$. However, the second case can only exist when the number of data samples presented is more than the number of predictors. In addition, the second case means that all learnt predictors are making pair-wise linearly independent predictions for the same dataset, which happens rarely. To summarise, as long as the predictors are not making the exact same predictions or completely uncorrelated predictions, our method gives a sparser solution which keeps fewer predictions after learning.

Through analysing the optimisation functions of comparison partners presented in Tab. 4.1, one can conclude that our proposed method has the potential of reaching a solution with higher sparsity, which reduces the number of remaining predictors and accelerates the inference. We mainly compare our method Eq. 4.3 and Stacked Generalisation in Eq. 4.8.

To illustate the effectiveness of applying the aforementioned algorithm for pruning ensemble models, we consider two main ensemble learning algorithms, including bagging and boosting.

## 4.4 Bagging

Bagging [Bre96] works when the base learner is an estimator with low bias and high variance. By averaging predictions produced by individual predictors trained with different subsets of the same dataset, the overall variance is reduced, which results in a reduction of the Mean Squared Error.

There are two main contributing factors to the success of bagging, including choosing a proper low-bias high-variance estimator as the base learner, and the bootstrap step that creates subsets of the same dataset. In practice, the unpruned decision tree [BFOS83] is usually considered as a good base learner, as well as the $k$-nearest neighbours (kNNs) algorithm [Alt92] with a very small $k$.

Bagging fails to improve over individual predictors when the variance of the chosen learner changes slowly or the bias changes drastically as the provided training samples vary [BY02]. Classic examples include pruned trees and kNNs with a large $k$.

Our proposed method is guaranteed to improve over the individual learners at least on the training set as it first selects only a few predictors, and corrects predictions by a linear regression model. Therefore, on the training set, it, at worst, performs as well as the best-performing predictor. In our case, even if the chosen learner doesn't meet the requirement of having low bias and high variance, our method can still improve over the best-performing predictor. In addition, since only a few predictors are selected, our method reduces the inference time.

### 4.4.1 Datasets

It is difficult to analyse the generalisation bound as the bound often tells us the worst performance of a learnt model on the unseen test set. Therefore, we conducted experiments with varying number of predictors $p \in \{2^3, 2^4, ..., 2^{10}\}$ and varying bootstrap ratio $\alpha \in \{0.1, 0.2, 0.5\}$. Experiments are run 10 times with different random seeds, and results are mainly presented in scatterplots.

We simulated a single-target regression task and a three-way classification task as the benchmark datasets to compare our proposed method with others. The regression task is based on the function provided in [Fri91]. Each data sample has 10 features, and the corresponding target is generated as following:

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5) + 10x_4 + 5x_5 + \varepsilon \tag{4.11}$$

where $\varepsilon \sim N(0,1)$, and $x_i \sim U(0,1)$, $\forall i \in \{1,2,...,10\}$. Only the first five features are used for generating the target, and the rest are independent from the target.

For the classification task, we followed the recipe in [HRZZ09] and generated a dataset for classification. Each data sample has 10 features, and the class label is determined by the length of the data sample.

$$c = \begin{cases} 0 & \sum_{i=1}^{10} x_i^2 \leq \mathcal{X}_{10,1/3}^2 \\ 1 & \mathcal{X}_{10,1/3}^2 < \sum_{i=1}^{10} x_i^2 \leq \mathcal{X}_{10,2/3}^2 \\ 2 & \sum_{i=1}^{10} x_i^2 \leq \mathcal{X}_{10,3/3}^2 \end{cases} \tag{4.12}$$

where $x_i \sim N(0,1)$, $i \in \forall\{1,2,...,10\}$, and $\mathcal{X}_{10,k/3}^2$ is the $(k/3)100\%$ quantile of the $\mathcal{X}_{10}^2$ distribution.

For both tasks, 3000 data samples are generated for training, and 1000 for testing. We mainly compare our method with bagging and stacked generalisation. We first bootstrap the dataset with replacement to generate multiple subsets. Then base learners are trained on individual subsets. For bagging, the predictions on the test set produced by individual predictors are directly averaged to make final predictions. For ours and stacked generalisation, predictions on the training set are collected from predictors, and optimisation problems described in Eq. 4.3 and Eq. 4.8 are solved respectively to produce weights on individual learners, and the predictions on the test set are computed accordingly with the weights.

### 4.4.2 Base Learner

**Decision Tree:** We consider three types, including unpruned decision trees, three-layer trees and single-layer tree stumps. The bias of decision tree models decreases and the variance increases as the more layers are added to the tree.

**K-Nearest Neighbors:** Four different numbers of neighbours are experimented respectively, includng 1, 5, 10, and 25. The bias of the kNN algorithm increases and the variance decreases as more neighbours are included in making the final decision.

**Neural Network:** This is a family of powerful learners, and they are capable of producing abstracted vector representations of the input data, with which the problem simplifies to one that can be solved with linear models. We design two neural networks as base learners, in which one has two hidden layers with 5 and 2 hidden units respectively, and the other has three hidden layers with 100 hidden units each. For the classification task, the activation function is Rectified Linear Unit [FM82], and the Hyperbolic Tangent Function [AS64] is applied for the regression task.

### 4.4.3 Results

We present results in two different fashions, and each reflects a unique aspect of the learning algorithm. Firstly, we do hope that the two-stage optimisation scheme that we proposed doesn't lead to a significant increase for the training time. The results of our proposed method, namely Linear Kernel Alignment, Stacked Generalisation, and Bagging, on the regression task are presented in Fig. 4.1, 4.3 and 4.5 with the base learner as decision trees, k-nearest neighbors algorithm, and neural networks, respectively. Classification results are presented in Fig. 4.2, 4.4 and 4.6

**Error vs. Training Time:** As both LKA and Stacked Generalisation learn additional weights on individual predictors, it is reasonable to expect that, in most cases, both algorithms should outperform Bagging, and the advantage gradually becomes significant as the bias of the base learner increases. Among different base learners, large neural networks achieve the lowest error rate on both classification and regression task without cross-validation.

An interesting observation is that, the issue of overfitting occurs for both Stacked Generalisation and LKA when the large neural networks are used as base learners, shown in Fig. 4.5 for regression and Fig. 4.6 for classification, and each network has sufficient number of bootstrapped samples to learn from. In that case, directly averaging predictions from models already reaches the lower bound of the MSE on the test set, which is the irreducible error, therefore, learning additional weights causes overfitting. However, since both Stacked Generalisation and LKA impose strong sparsity on the weights, the issue of overfitting isn't severe.

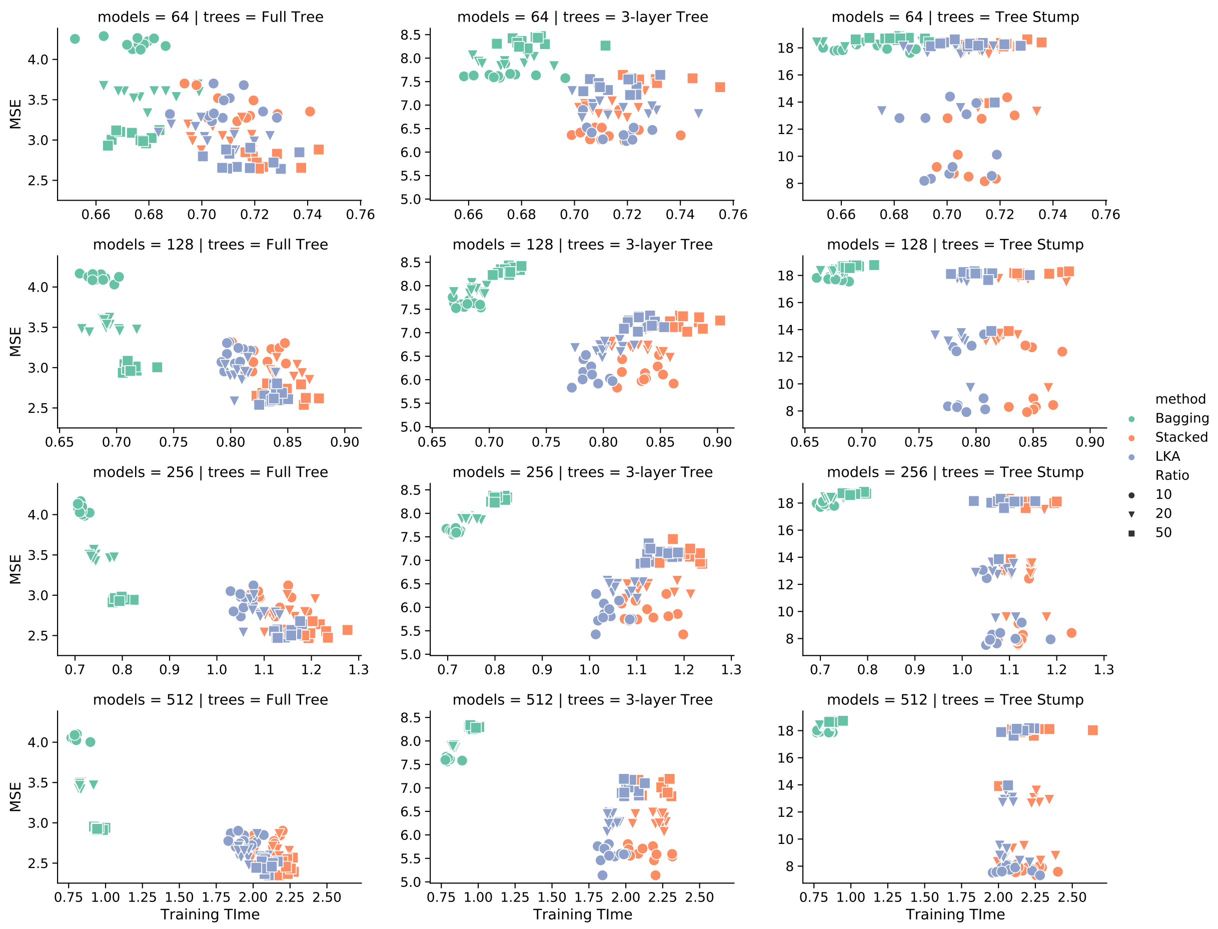


**Figure 4.1**: **Mean Squared Error on Test Set vs. Training Time of Bagging with Decision Trees.** Both Stacked Generalisation and Linear Kernel Alignment improve the performance by readjusting weights on individual predictors, and the observation is consistent across three different tree types, three bootstrap ratios $\{10\%, 20\%, 50\%\}$, and three numbers of initial learners. However, since an additional optimisation step is involved in both, the training time is increased.

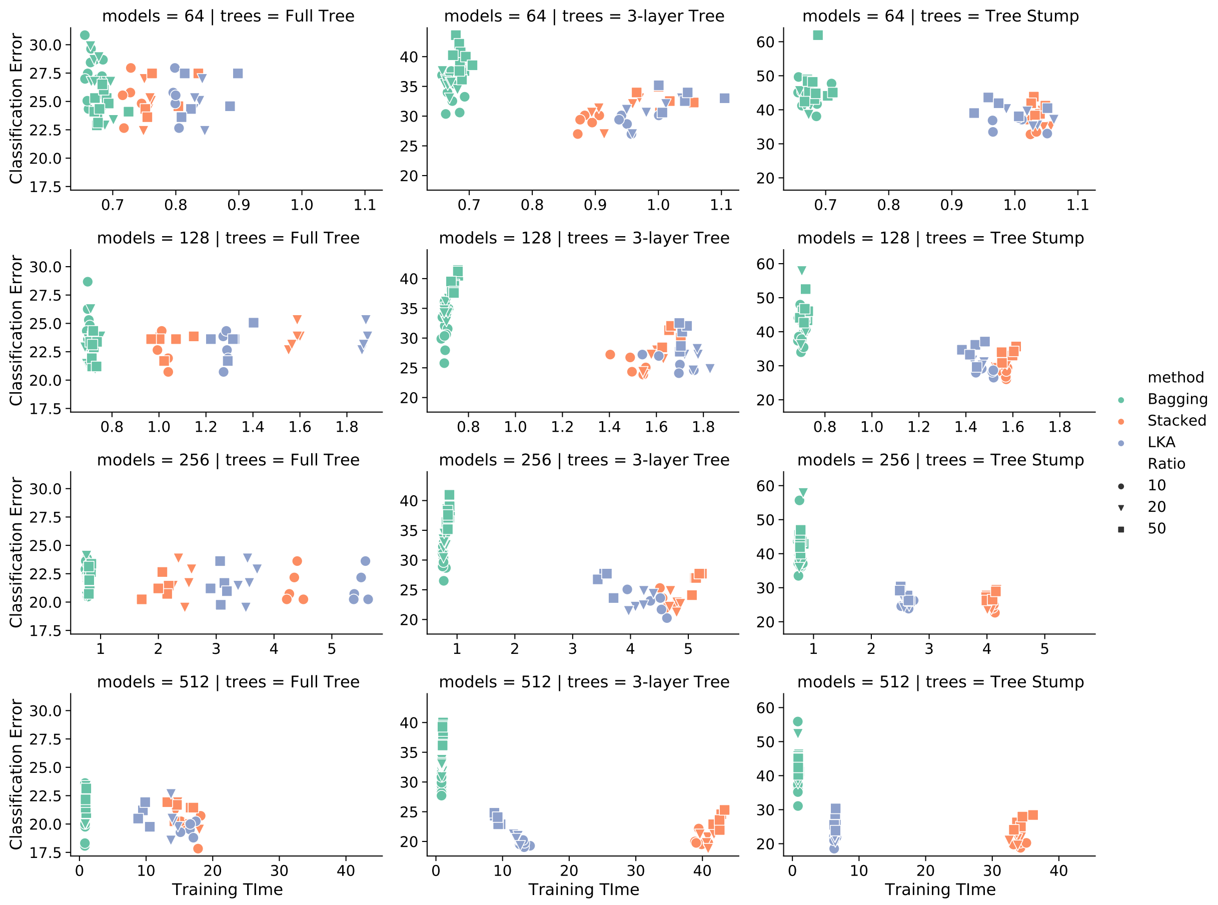


**Figure 4.2**: **Classification Error on Test Set vs. Training Time of Bagging with Decision Trees.** The observation on the classification task is relatively similar to that on the regression task. Both algorithms boost the performance whilst LKA has a modest increase of the training time.

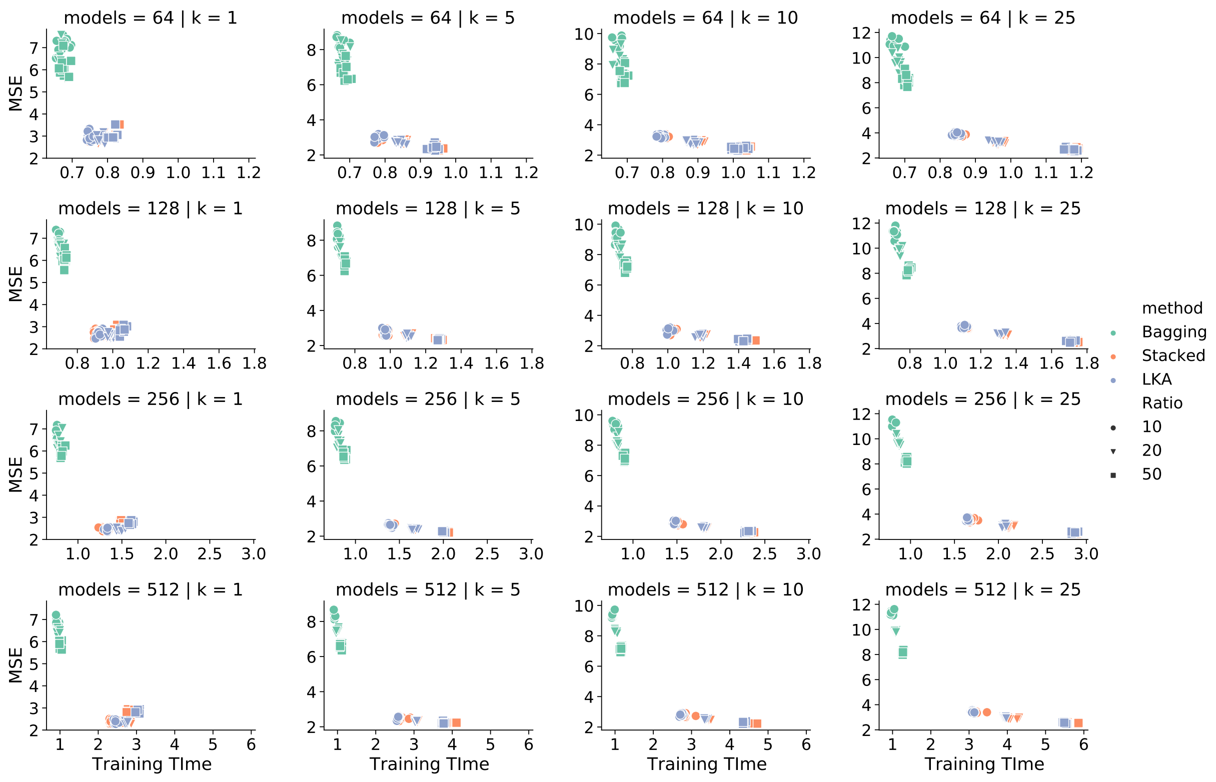


**Figure 4.3**: **Mean Squared Error on Test Set vs. Training Time of Bagging with K-Nearest Neighbours.** It is clear that both Stacked Generalisation and LKA reach smaller error rates than Bagging does in all cases. The increased training time comes from the additional optimisation step in both Stacked Generalisation and Linear Kernel Alignment, and there is no significant difference between two methods in terms of the training time.

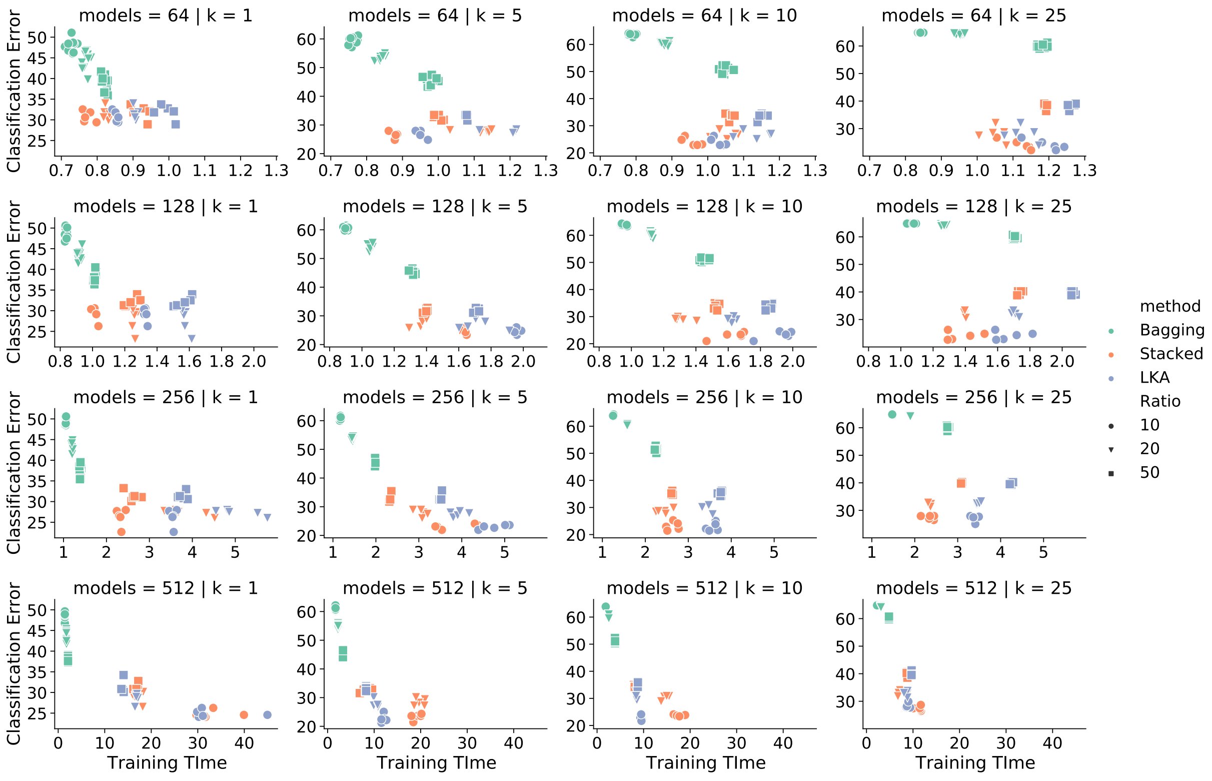


**Figure 4.4**: **Classification Error on Test Set vs. Training Time of Bagging with K-Nearest Neighbors.** CLassification errors are further reduced by readjusting weights on individual predictors through LKA or Stacked Generalisation with a fixed bootstrap ratio. Interestingly, the optimal performance of both LKA and Stacked Generalisation is achieved with the smallest bootstrap ratio 10% in our experiments, but Bagging performs better when the bootstrap ratio is larger.

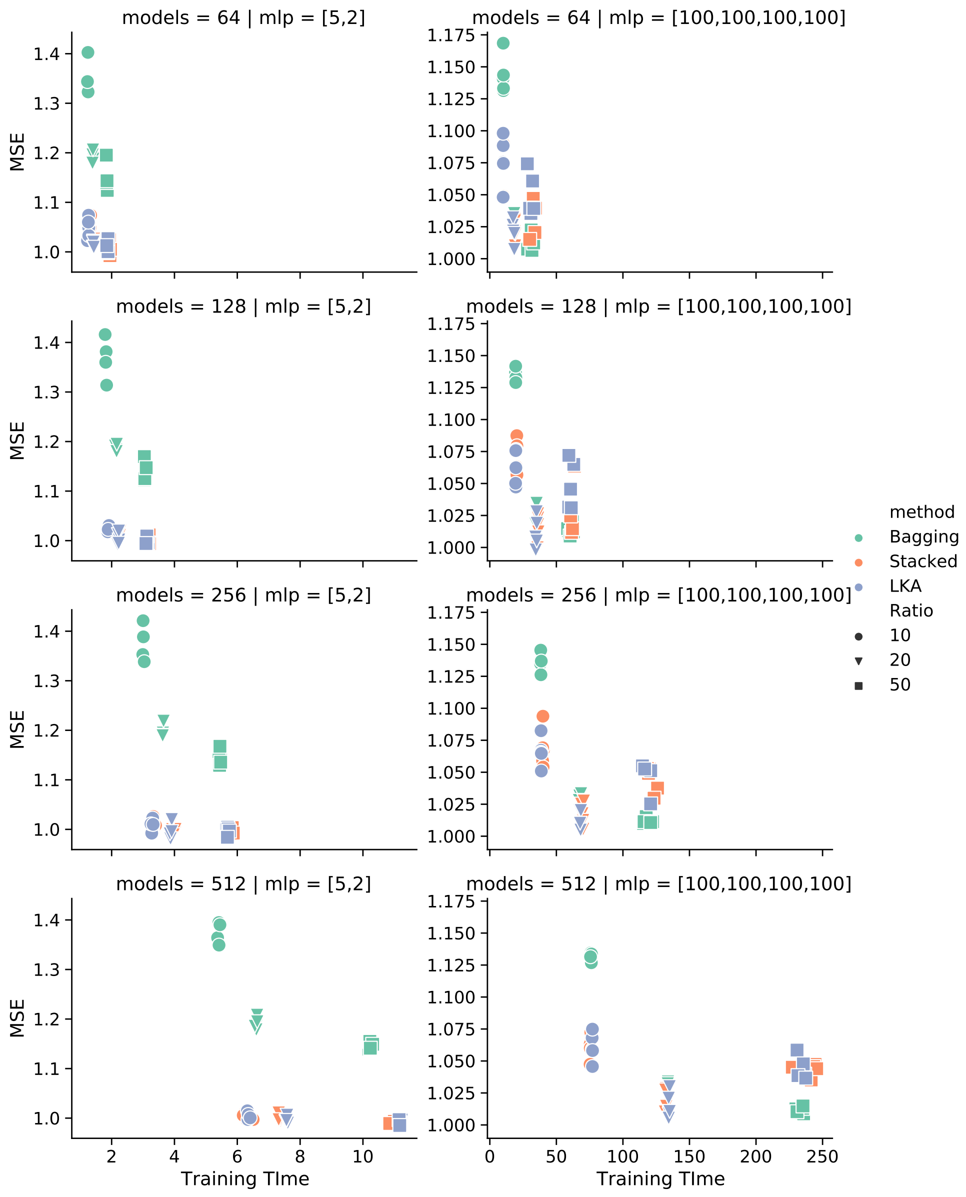


**Figure 4.5**: **MSE on Test Set vs. Training Time of Bagging with Neural Networks.** Both algorithms improve the performance over bagging overall. However, the performance slightly deteriorates for the regression task when the bootstrap ratio is 50% and large neural networks are used as base learners, which might be a result of overfitting as both algorithms learn additional weights on top of individual predictors.

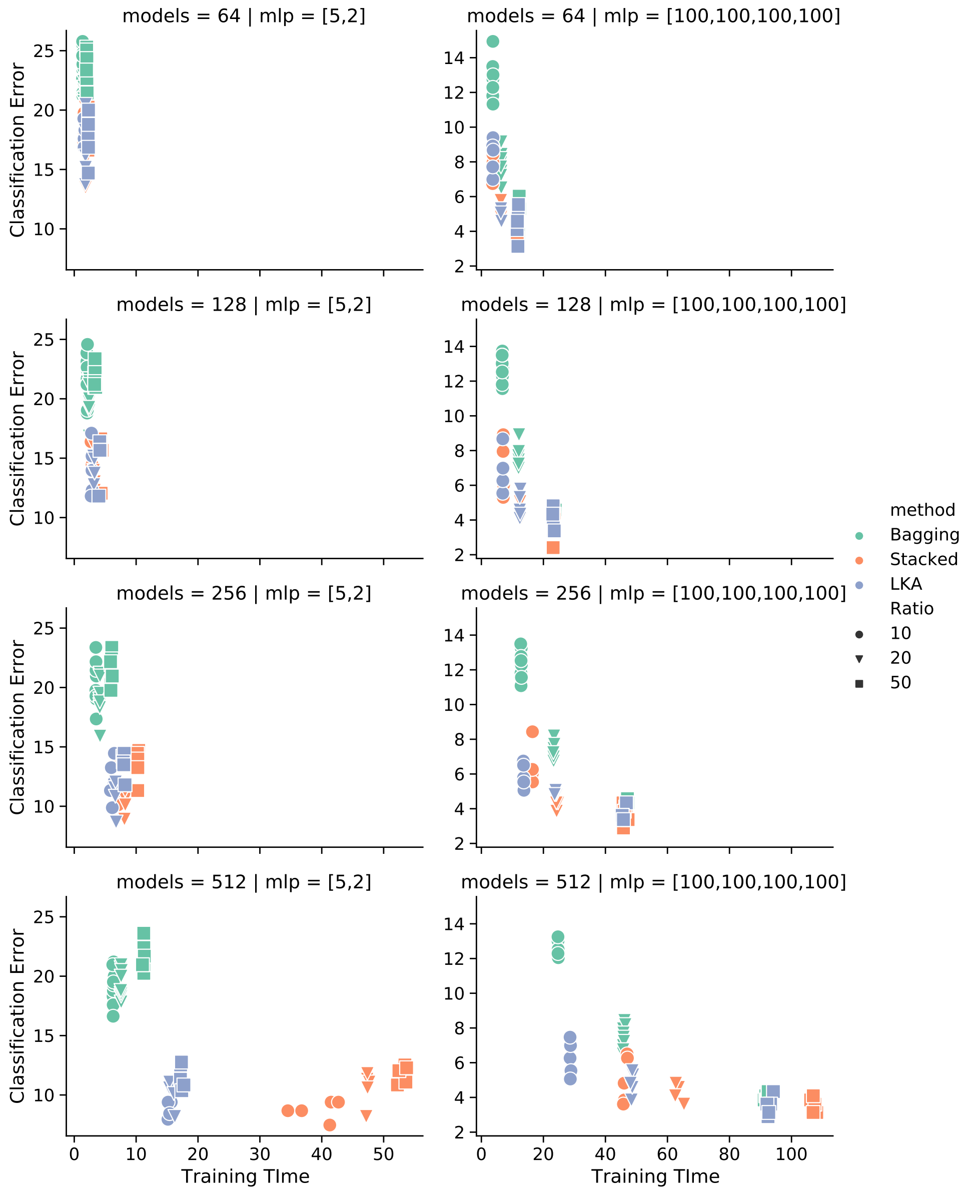


**Figure 4.6**: **MSE on Test Set vs. Training Time of Bagging with Neural Networks.** With a fixed bootstrap ratio, LKA and Stacked Generalisation perform better than Bagging does in most cases, and LKA takes slightly less time during training than Stacked Generalisation does.

**Comparison on the number of remaining predictors:** Given that the performance is improved, we are interested in the number of remaining predictors in Stacked Generalisation and LKA. Fig. 4.7, 4.9, and 4.11 present the comparison between the number of remaining predictors produced by Stacked Generalisation and that resulting from LKA in each condition on the regression task. For the same comparison on the classification task, Fig. 4.8, 4.10, and 4.12 present results.

Given our analysis above, when predictors are producing similar predictions with each other, and the similarity is measured by the pair-wise inner-product values $\{\hat{\boldsymbol{y}}_i^\top \hat{\boldsymbol{y}}_j\}_{i,j=1}^{p}$, LKA produces sparser solutions than Stacked Generalisation does. For the same type of base learners, keeping the same boostrap ratio while reducing the variance of the base learner helps our proposed method to outperform Stacked Generalisation. Therefore, we can expect that, when reducing the depth of a decision tree or increasing the number of nearest neighbours in k-NN algorithm, the bias increases while the variance decreases, and the predictions made by individual predictors gradually become similar to each other, then LKA should have fewer remaining predictors than Stacked Generalisation does. The empirical evidence shown in Fig. 4.7 and 4.9 supports our analysis. Whilst for neural networks, as they exhibit the behaviour of a low-bias and low-variance learner, LKA keeps a smaller number of predictors than Stacked Generalisation does. The observation presented in 4.11 matches our analysis as well.

The important aspect of our LKA-based method is that it improves the performance on top of bagging and it prunes predictors to reduce the computational cost and runtime for the inference after learning. Compared with Stacked Generalisation, our proposed LKA-based method performs similarly whilst keeping fewer predictors.

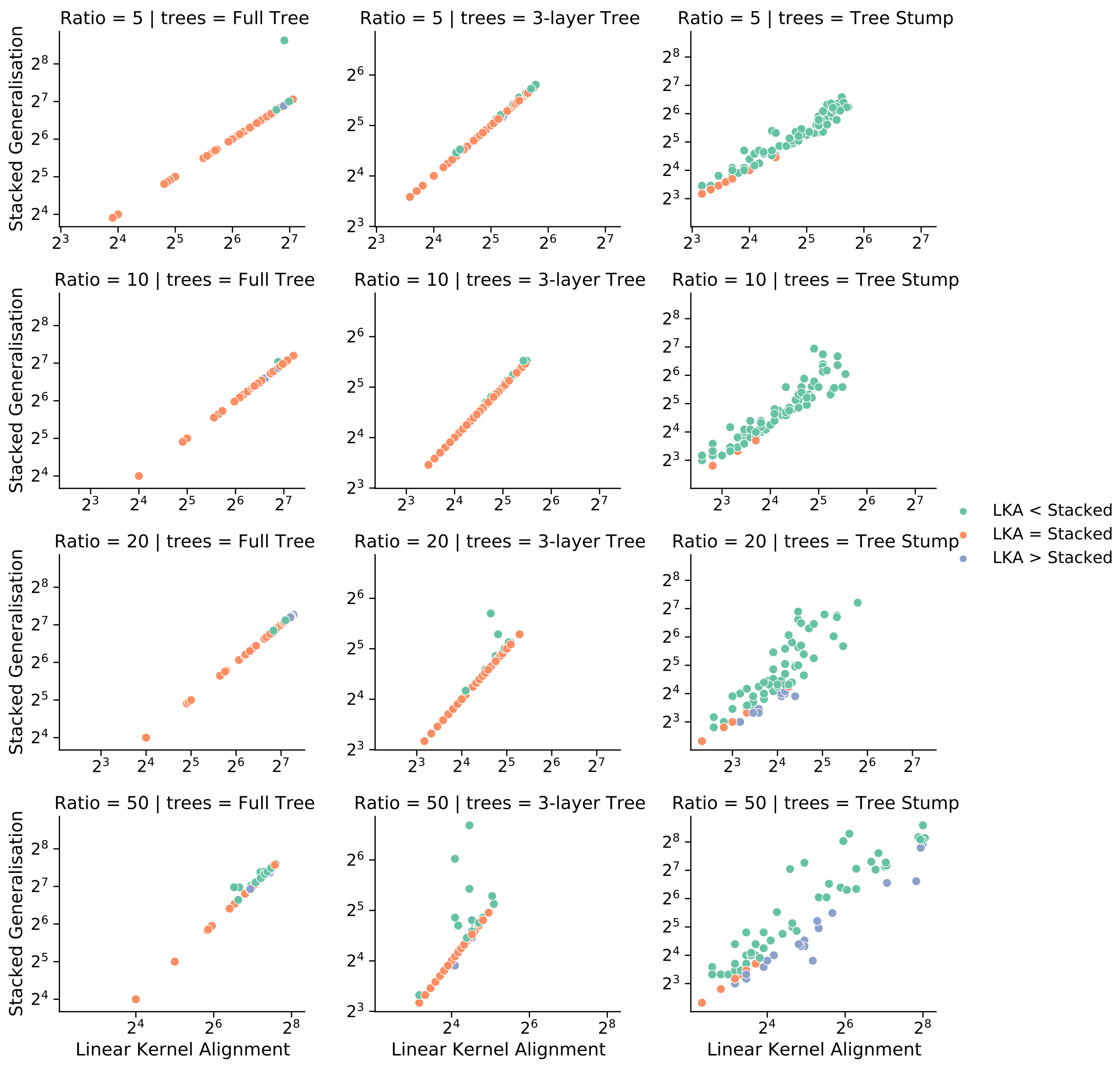


**Figure 4.7**: The Number of Remaining Predictors by Stacked Generalisation vs. LKA for Regression. Decision Trees are the base learners. By reducing the depth of a decision tree (moving from the leftmost column to the rightmost one), the bias increases and the variance decreases, and, based on our analysis, LKA produces sparser solution than Stacked Generalisation does.

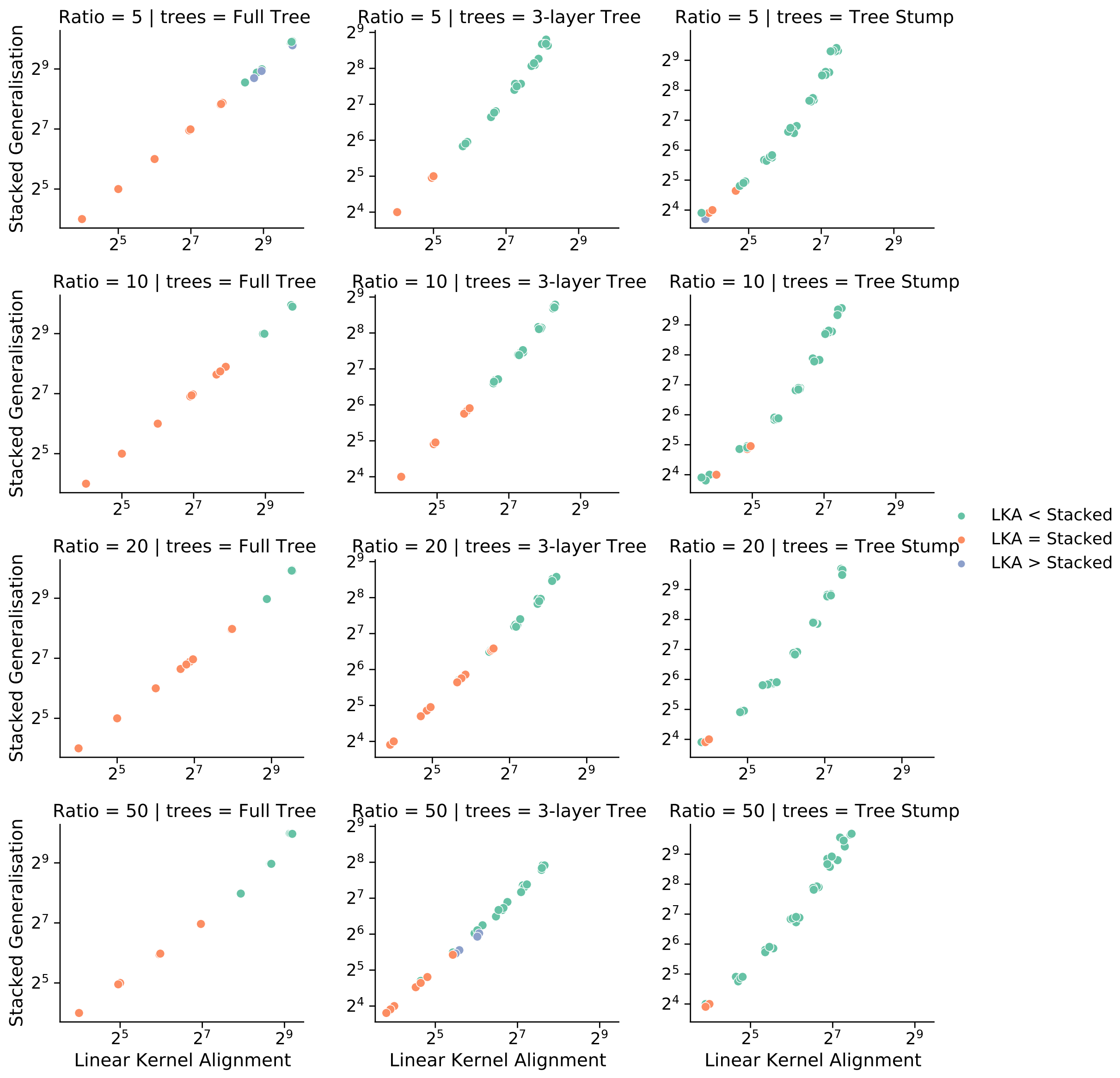


**Figure 4.8**: The Number of Remaining Predictors by Stacked Generalisation vs. LKA for Classification. Decision Trees are the base learners. The phenomenon we described on the regression task is more salient on the classification task. LKA prunes more predictors than Stacked Generalisation does when the base learner is high-bias and low-variance.

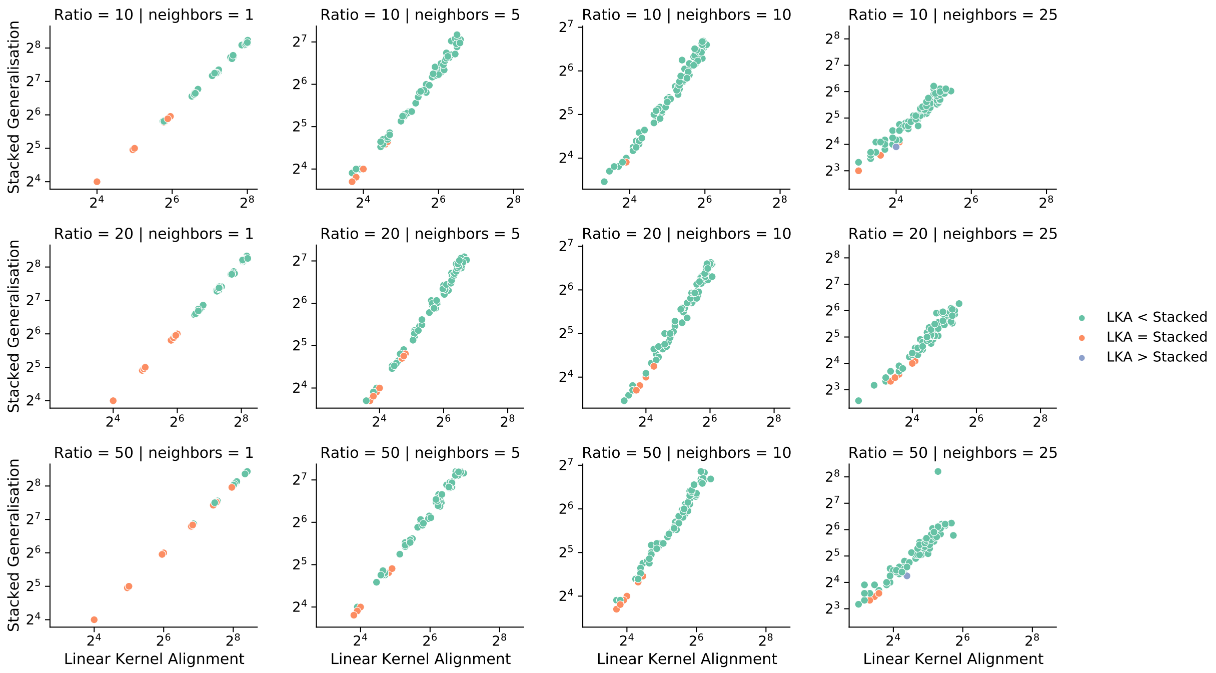


**Figure 4.9**: The Number of Remaining Predictors by Stacked Generalisation vs. LKA for Regression. K-nearest Neighbors are the base learners. With increasing number of neighbours in consideration, LKA produces sparser solution than Stacked Generalisation does, which results in a smaller set of predictors for inference after learning.

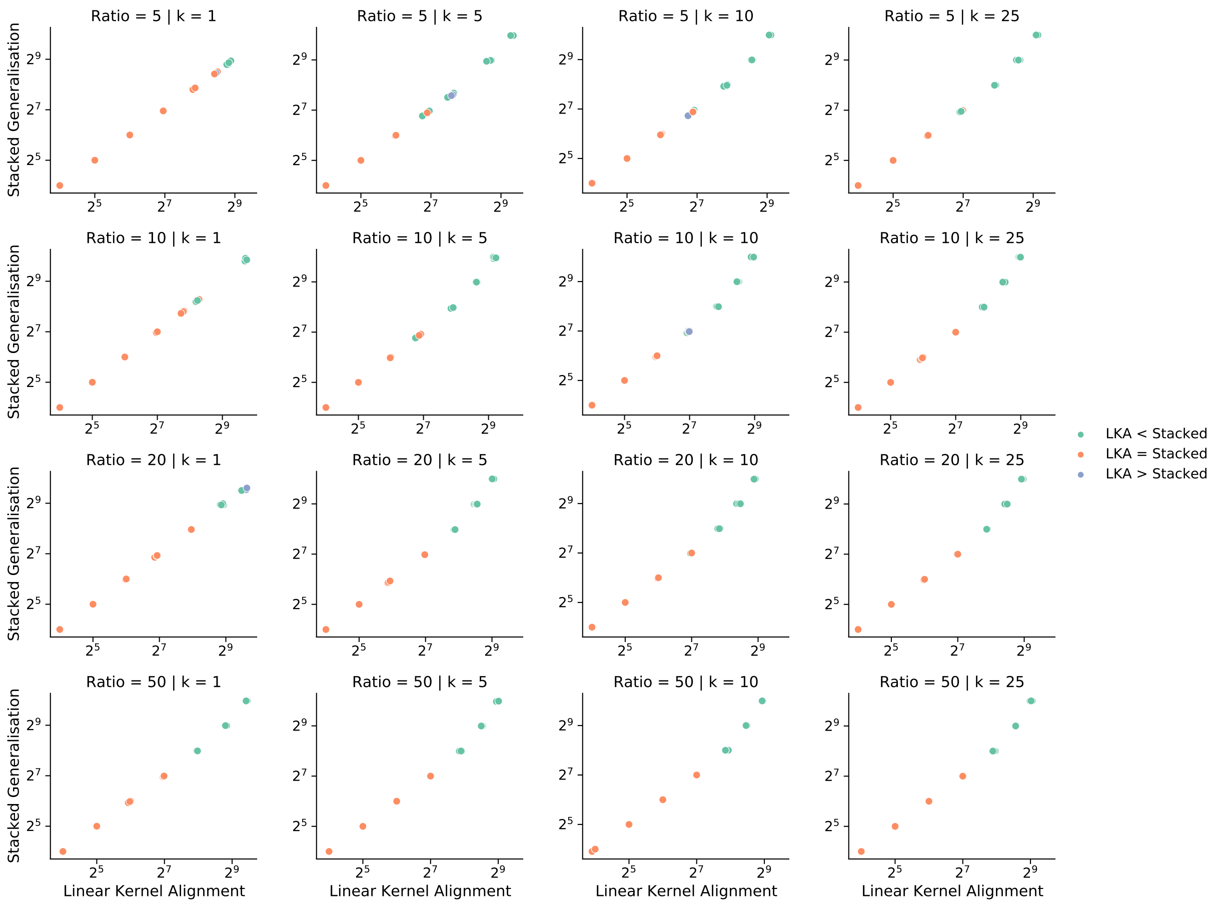


**Figure 4.10**: The Number of Remaining Predictors by Stacked Generalisation vs. LKA for Classification. K-nearest Neighbors are the base learners. The advantage of LKA over Stacked Generalisation in terms of keeping a small number of predictors isn't prominent, but LKA either keeps a smaller set of predictors than Stacked Generalisation does or they keep the same number of predictors after learning.

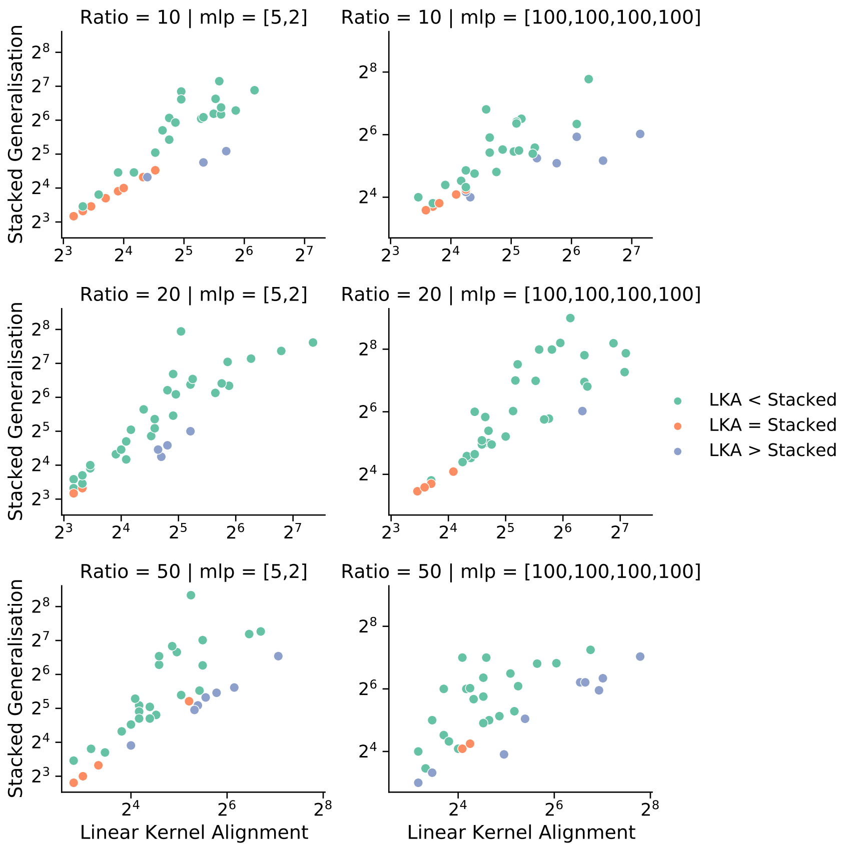


**Figure 4.11**: The Number of Remaining Predictors by Stacked Generalisation vs. LKA for Regression. Neural networks with two different sizes are base learners respectively. Overall, LKA prunes more predictors than Stacked Generalisation does, which accelerates the inference after learning.

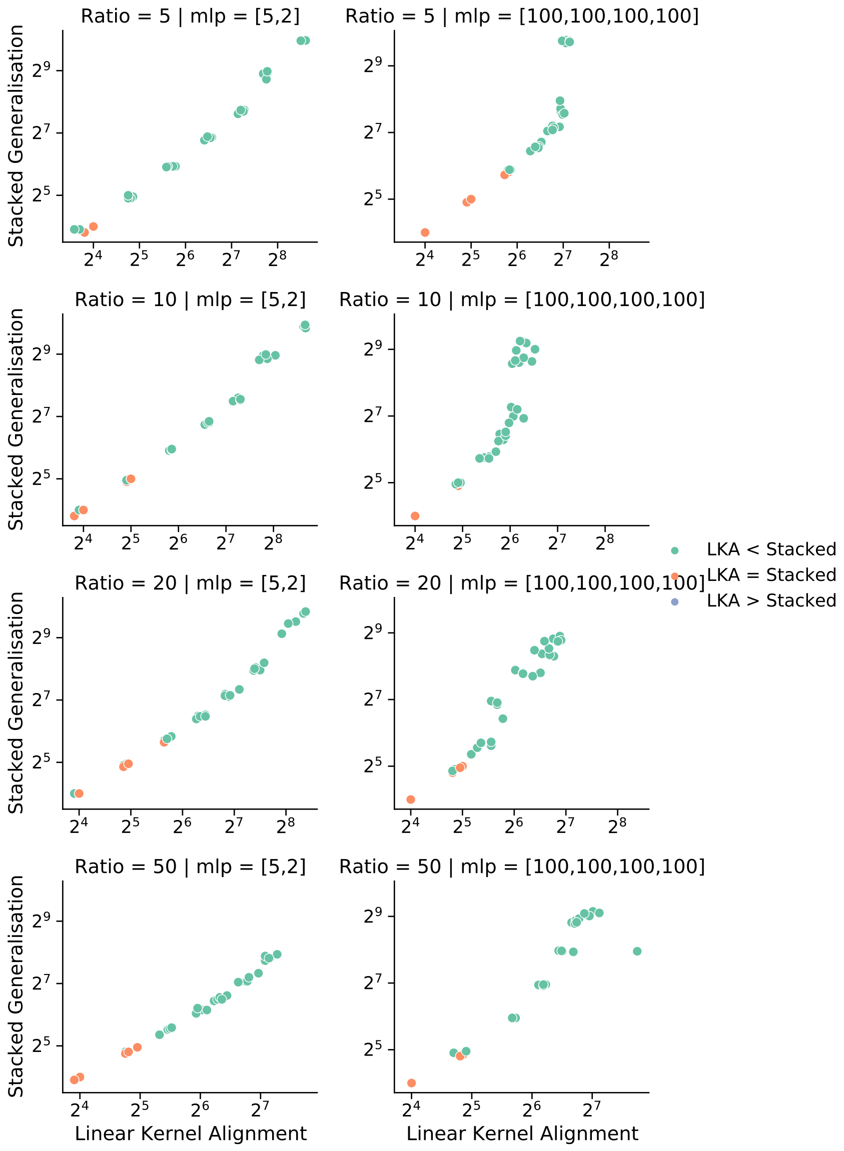


**Figure 4.12**: The Number of Remaining Predictors by Stacked Generalisation vs. LKA for Classification. Neural networks with two different sizes are base learners respectively. The observation is similar here as we have for the regression task. LKA keeps fewer predictors than Stacked Generalisation does.

**Pruning by Ranking:** As each predictor learns on a subset of the training data, one can rank all predictors by their predictive errors on the whole training set and accumulate them sequentially according to the sorted predictive errors. In Fig. 4.13, 4.14, and 4.15, we visualised the accumulation progress with 256 predictors in total as line plots, and errors and number of remaining predictors produced by Stacked Generalisation and LKA are plotted as dots.

One can simply stop the sequential accumulation by early stopping, and it helps bagging with k-NN algorithms to perform much better, and has a smaller positive effect on bagging with small neural networks. In the rest of the cases, accumulating all is optimal. In terms of ensemble pruning, both LKA and Stacked Generalisation prune a significant number of predictors, and boost the performance in most situations.

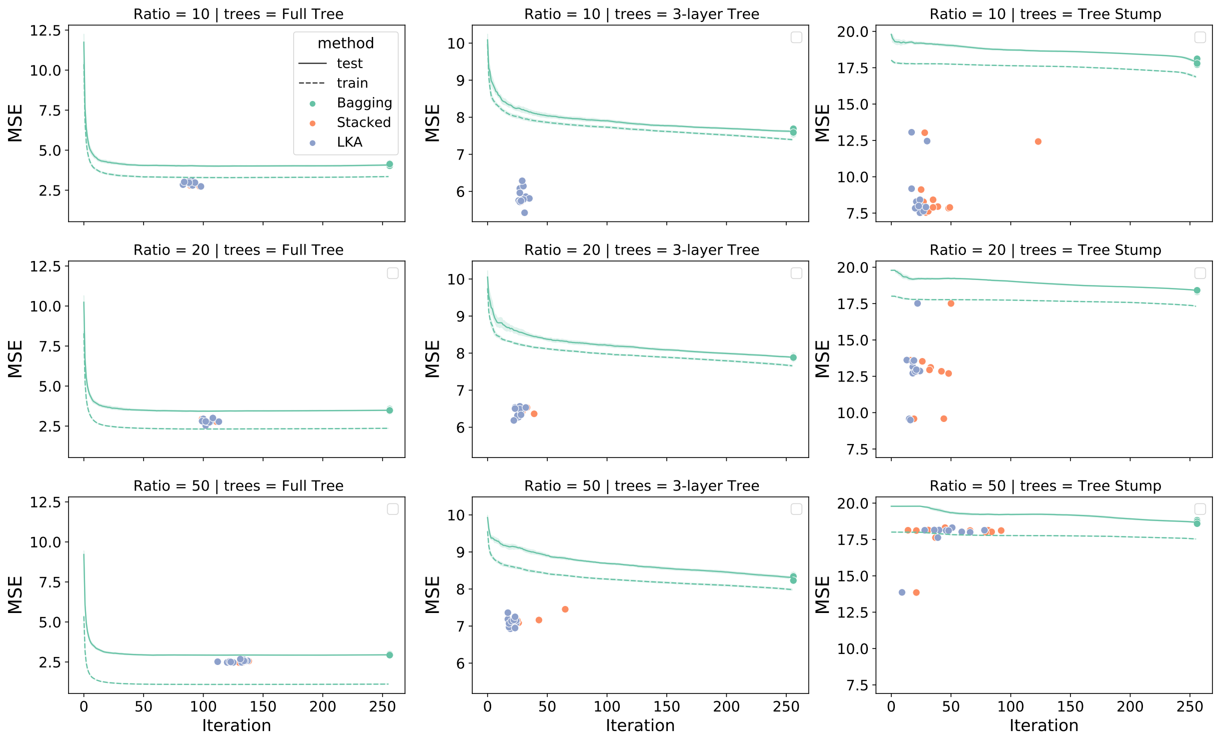


**Figure 4.13**: **Bagging with 256 decision trees.** Predictors are sorted based on their predictive errors on the entire training set, and accumulated sequentially into making final predictions on the test set. On top of the predictors, LKA and Stacked Generalisation are both applied respectively, and the error and the number of remaining predictors of each method is plotted as dots. We can see that both methods are able to achieve similar test errors with many fewer predictors.

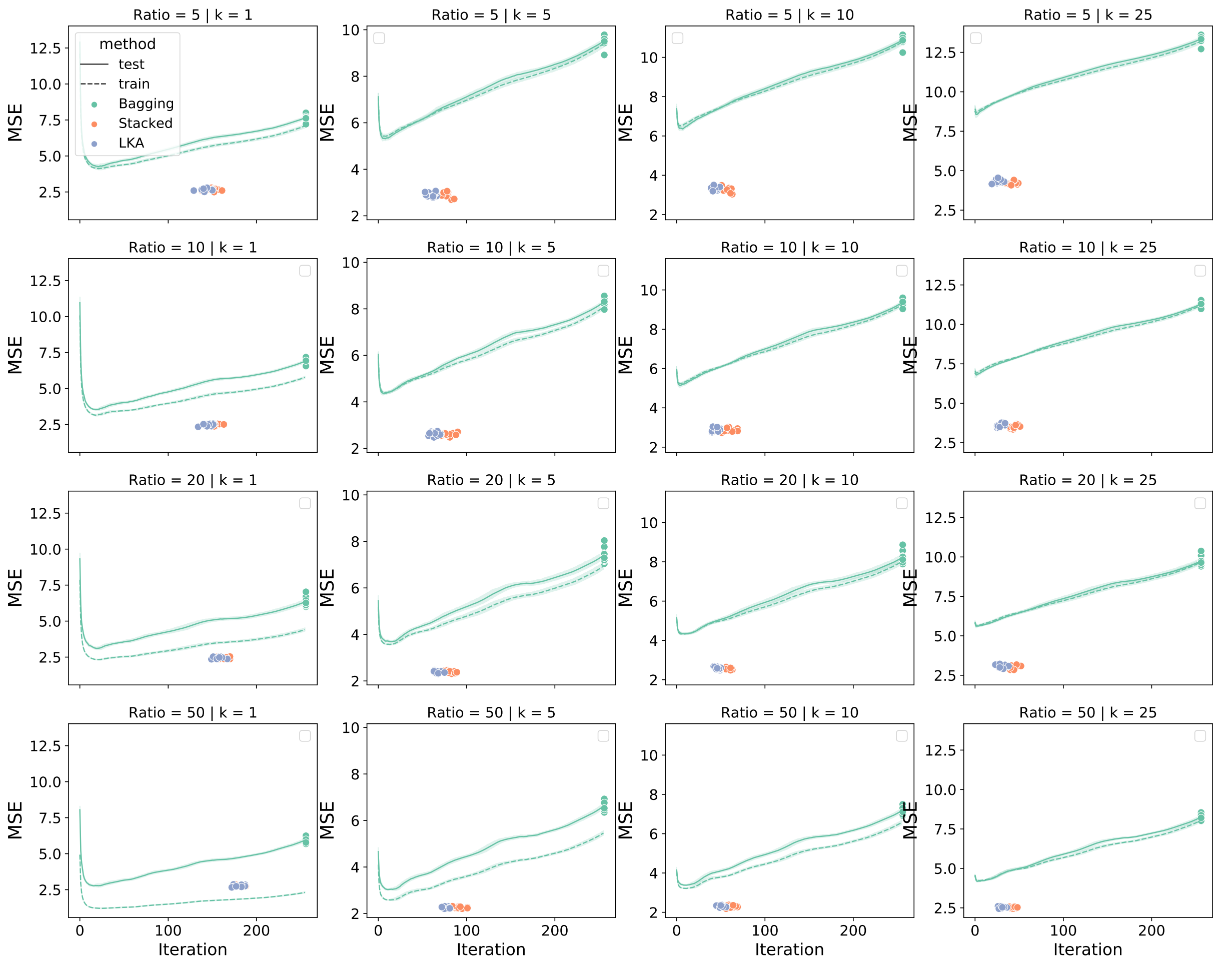


**Figure 4.14**: **Bagging with 256 k-nearest neighbors regressors.** Predictors are sorted and accumulated according to the ascending order of their predictive errors on the training set. In many cases, LKA keeps fewer predictors than Stacked Generalisation does, and outperforms bagging.
One can stop the accumulation when the training error stops decreasing. However, even with early stopping, expect for the only case in the bottom left, bagging performs worse than LKA and Stacked Generalisation.

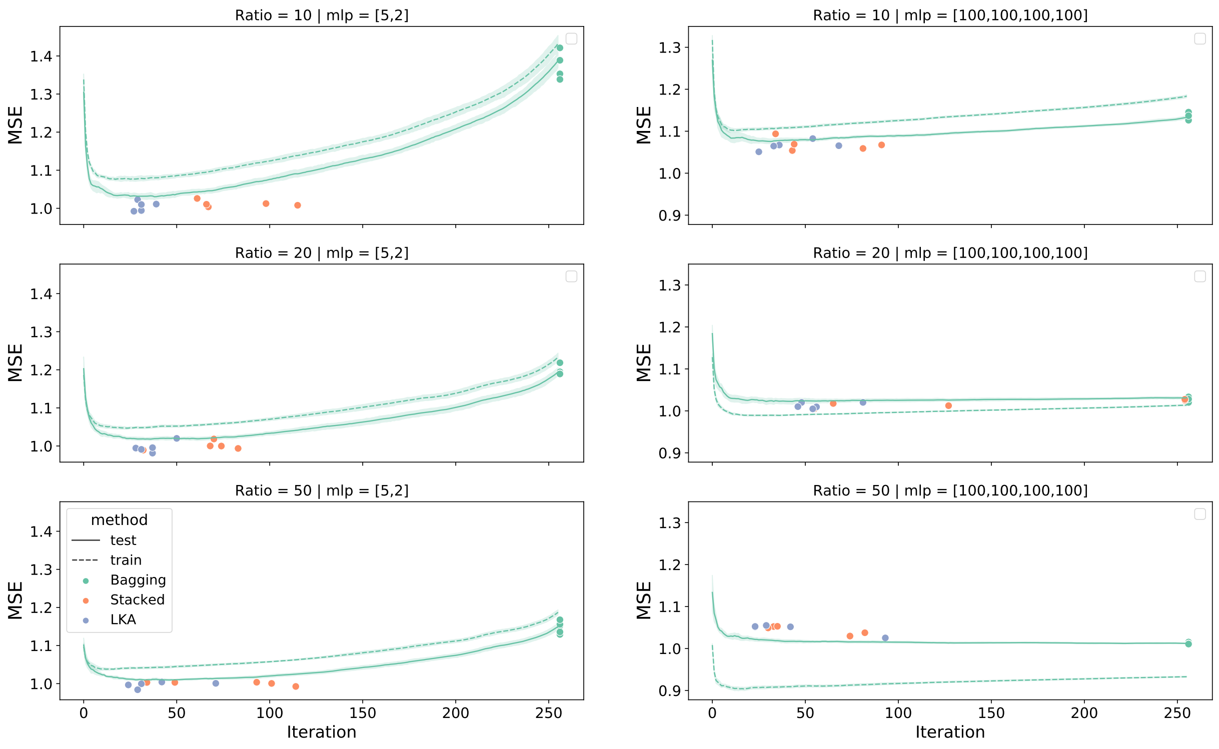


**Figure 4.15**: **Bagging with 256 neural networks.** Overall, LKA keeps fewer predictors than Stacked Generalisation does, and outperforms bagging expect for the case in the bottom right. The same early stopping helps bagging with small neural networks more than it does for bagging with large neural networks.

## 4.5 Boosting

Boosting algorithms advance bagging by iteratively fitting weak learners so that the weighted sum of the predictions from these weak learners gradually approximates the target. The choice of a proper weak learner is important in determining the success of boosting. Conventionally, for a binary classification task, a suitable weak learner only needs be able to perform better than chance level, and then by chaining weak learners, the training error reduces exponentially w.r.t. the number of learners applied in boosting. Boosting fails when the learner's complexity is very high, as it causes following learners to learn from spurious gradient signals or errors.

For boosting, it is not a trival task to select a weak learner for a given task. A useful rule-of-thumb is to pick a tree stump, which has only one layer with three parameters, or a decision tree with very few layers. Based on our discussion, our proposed method is able to improve over the best learner due to the linear regression step, and reduce the inference time by selecting only a small subset of the learnt predictors for making predictions.

In our experiments, we first train a gradient boosting algorithm on the given dataset, and then our proposed method is applied on top of the predictors produced by boosting. We experimented with $p \in \{2^1, 2^2, ..., 2^{10}\}$ and $\alpha \in \{0.1, 0.2, 0.5\}$.

### 4.5.1 Datasets

Here, we mainly experimented with the regression task, and the dataset is the same one as we generated for the bagging experiments.

### 4.5.2 Base Learners

Two boosting algorithms are used to provide predictors, which are AdaBoost [FS97, Dru97], and Gradient Boosting [Fri01]. In both algorithms, three decision trees with different depths — 1, 3, and 10 — are applied as base learners.

The procedure of experiments on boosting is the same as the one on bagging. A boosting algorithm with a base learner is trained on the dataset, and then our method and stacked generalisation are applied to prune the predictors using the training data.

### 4.5.3 Results

Fig. 4.16 and 4.17 present results with Gradient Boosting as the algorithm for learning predictors. Stacked Generalisation is only able to reach a similar error rate with boosting when the base learner is low-bias and high-variance, however, with that base learner, boosting doesn't perform as well as it does with a weak learner, which is high-bias and low-variance. In the situation where 3-layer Trees or Tree Stumps are used as the base learners, Stacked Generalisation performs worse than boosting. Meanwhile, our proposed LKA gives better performance than boosting itself when the bootstrap ratio is small, and keeps similar performance with boosting in other situations. In terms of the pruned predictors, the difference on the numbers of remaining predictors becomes significant when more initial predictors are used in boosting. Overall, LKA achieves faster inference speed with better performance than Stacked Generalisation on predictors learnt with Gradient Boosting.

Fig. 4.18 and 4.19 use AdaBoost algorithm. Both LKA and Stacked Generalisation reach lower test error than AdaBoost does, whilst LKA provides a strong ability in pruning predictors when a large number of predictors are learnt through AdaBoost.

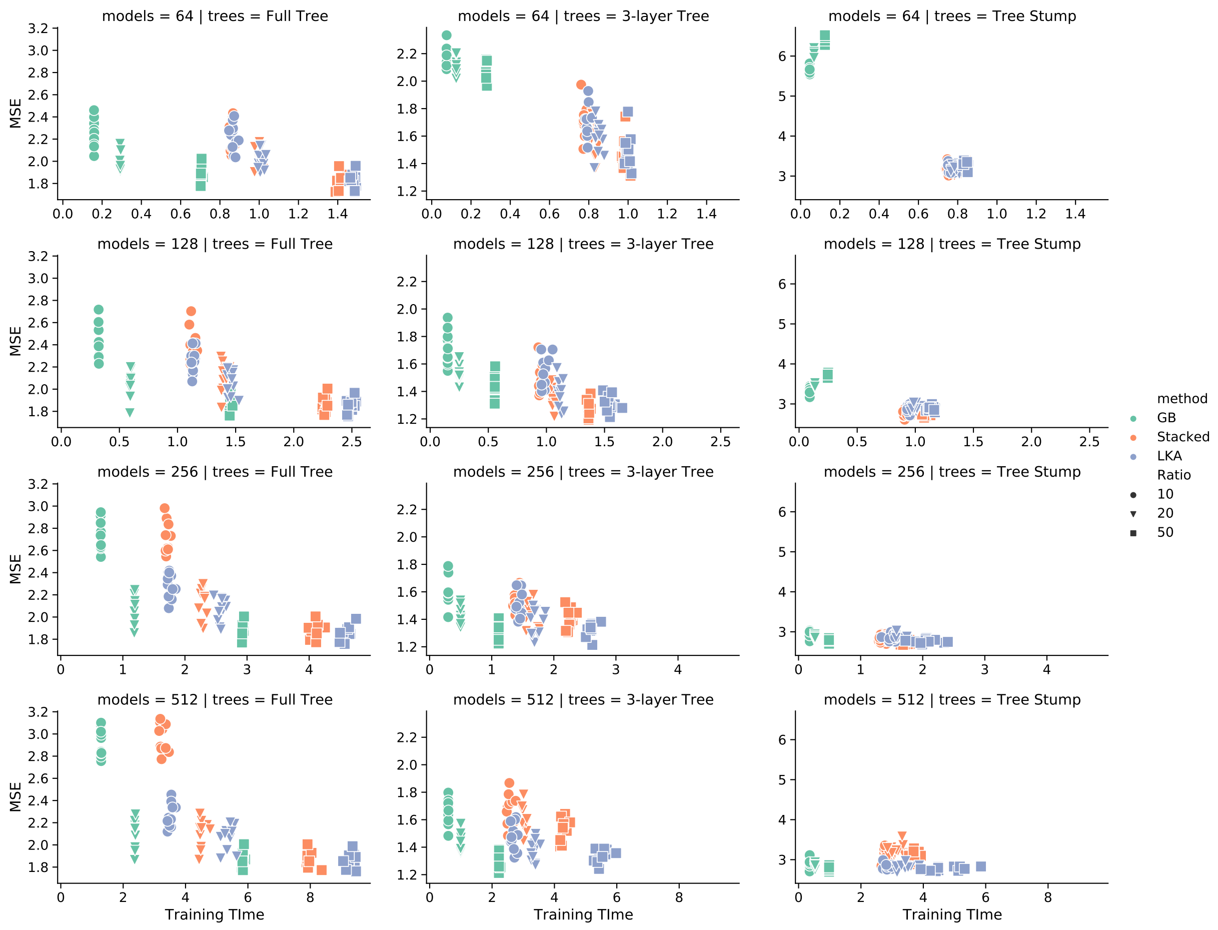


**Figure 4.16**: **Gradient Boosting with Decision Trees.** Stacked Generalisation gives worse performance than boosting itself when 3-layer trees or tree stumps are applied in learning with a bootstrap ratio 50%. Our proposed LKA is able to perform as well as boosting, and improves the performance of boosting when the bootstrap ratio is small.

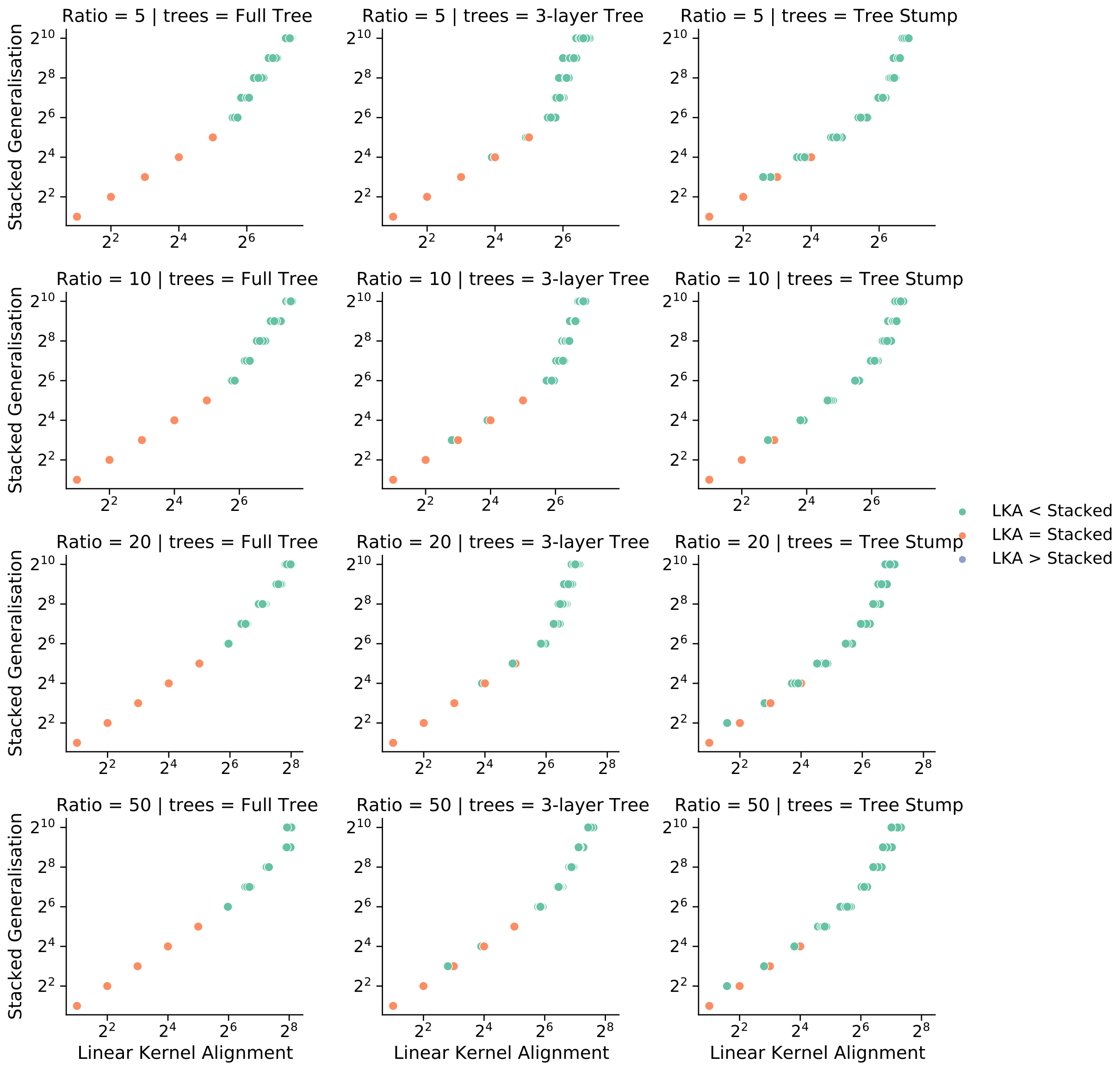


**Figure 4.17**: **Number of Remaining Predictors.** Gradient Boosting is the initial algorithm that provides predictors. The difference between LKA and Stacked Generalisation is salient when more than 64 initial predictors are used in boosting, and LKA prunes more predictors than Stacked Generalisation does.

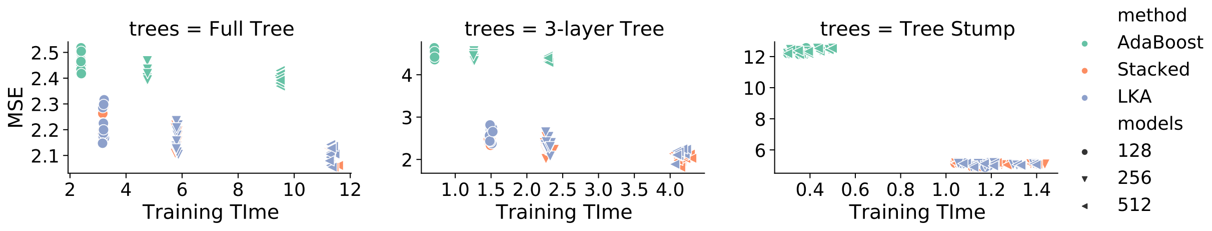


**Figure 4.18**: **AdaBoost with Decision Trees.** Both methods provide better performance than AdaBoost does on the test set. There is no significant difference on the runtime and the error rate between LKA and Stacked Generalisation.

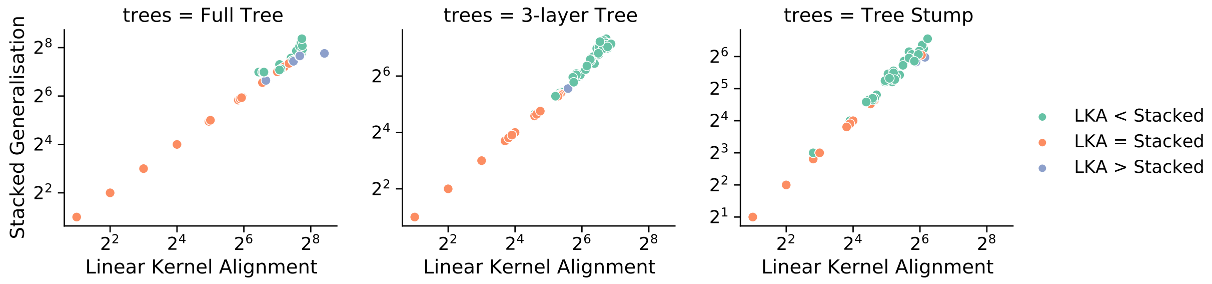


**Figure 4.19**: **Number of Remaining Predictors.** AdaBoost algorithm is the initial algorithm that provides predictors. A similar trend can be observed as the one we described for Gradient Boosting. When more initial predictors are applied, LKA demonstrates a stronger pruning functionality than Stacked Generalisation does.

### 4.5.4 Iterative Pruning in Boosting

Boosting iteratively fits a base learner to the residual, and at the end of each iteraction, a weight is searched and assigned to the fitted learner which becomes a predictor to achieve the maximum error reduction. Therefore, the training error decays as more predictors are learnt. A naïve approach on pruning the predictors is based on the decay of the regression error on the whole training set. Fig. 4.20 presents early stopping on Gradient Boosting, and Fig. 4.21 on AdaBoost.

In both cases, LKA prunes around half of the predictors whilst keeping the regression error at the same level as boosting or reducing it to a lower level. In comparison to LKA, Stacked Generalisation only does pruning on AdaBoost and keeps all predictors on Gradient Boosting.

## 4.6 Conclusion

We proposed a two-step ensemble pruning algorithm that is based on linear kernel alignment. Experiments are conducted on both bagging and boosting with various base learners. Pruning predictors with our method doesn't hurt the performance, and, in most cases, it improves on top of the initial performance provided by the selected ensemble learning algorithm. Compared with the one-step regression-based pruning method, ours keeps fewer predictors, which boosts the inference speed after learning.

## 4.7 Acknowledgements

Chapter 4, in part, is currently being prepared for submission for publication of the material. Tang, Shuai; de Sa, Virginia R.. "Ensemble Pruning with Linear Kernel Alignment". The dissertation author was the primary investigator and author of this material.

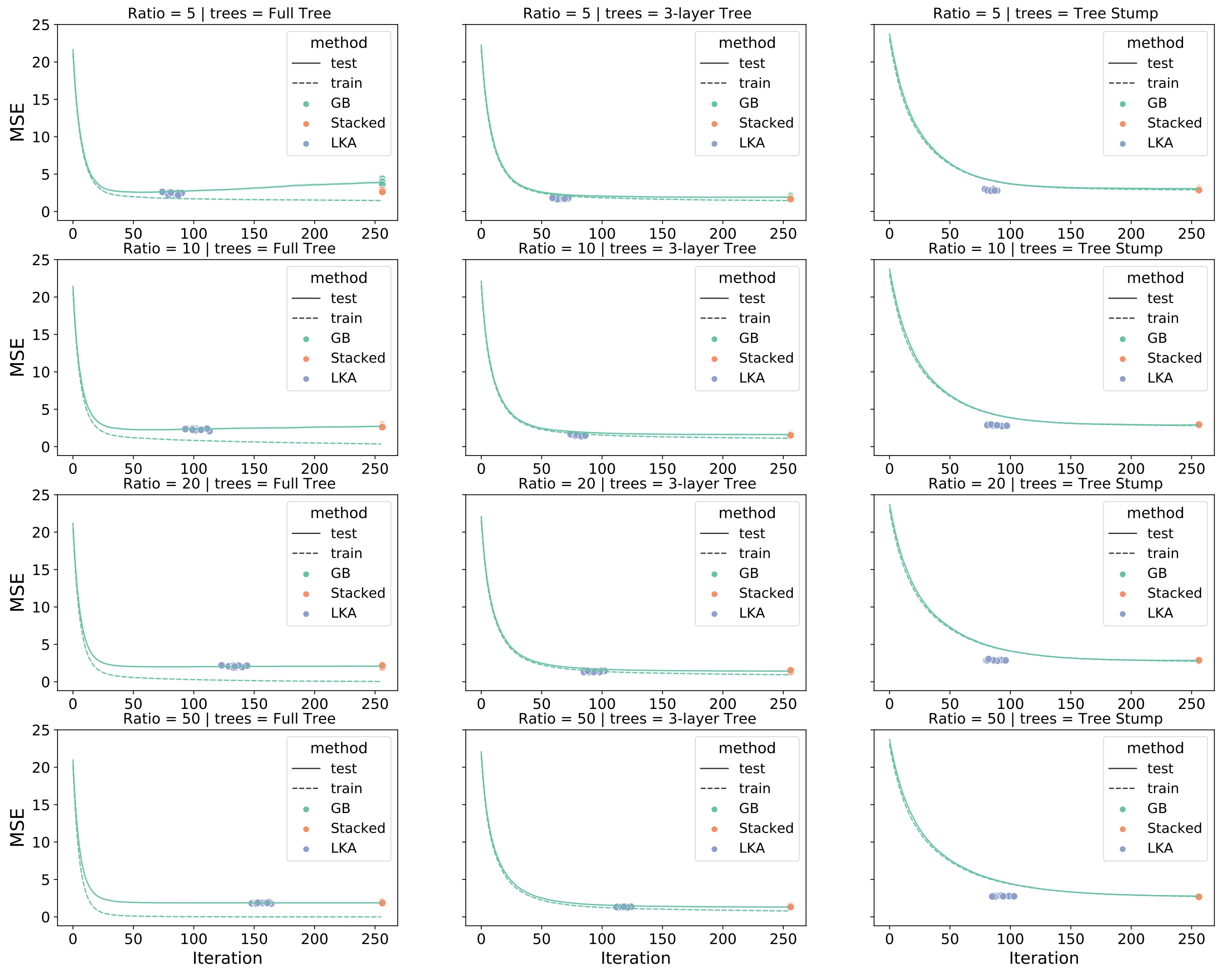


**Figure 4.20**: **Gradient Boosting with Early Stopping.** Since each base learner learns from a random subset of the training set, which has been shown to be effective in avoiding overfitting in practice, one can use the performance improvement on the whole training set as a indicator on when to stop accumulating predictors.
With bootstrapping in each iteration, the error rate keeps decreasing as predictors are being accumulated. Therefore, early stopping on the whole training set isn't informative enough. However, LKA is able to prune around half of the predictors whilst keeping the performance on the test set the same as the final performance of Gradient Boosting or providing better performance when high-bias low-variance learners are used. In comparison, Stacked Generalisation always keeps the entirety of the predictors for the inference time, which is unnecessary.

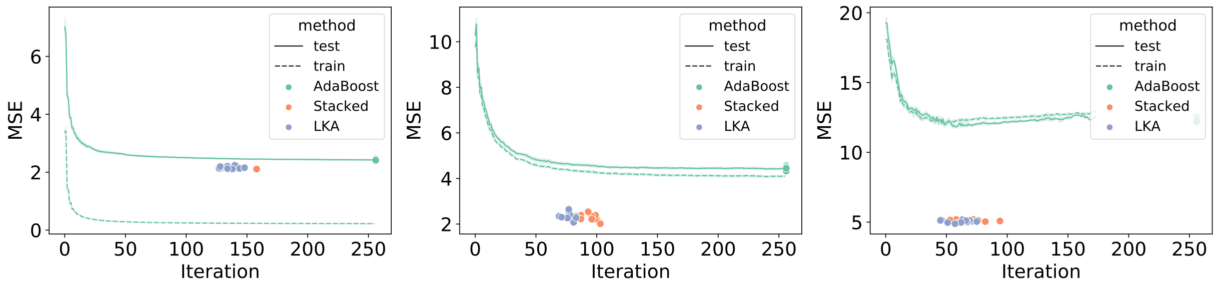


**Figure 4.21**: **AdaBoost with Early Stopping.** The same early stopping can be applied on the AdaBoost algorithm to determine when to stop accumulating predictors. Overall, we can see that with early stopping, AdaBoost doesn't perform as well as LKA or Stacked Generalisation on the test set. Both LKA and Stacked Generalisation prune more than half of the predictors, which leads to fast inference speed. The difference between the two algorithms is not significant enough in terms of either the performance or the number of remaining predictors.

# Chapter 5

# Summary

In the thesis, we build connections between kernel machines and domain generalisation with neural networks through the algorithm for learning kernels with only linear kernels.

Chapter 2 accelerates the computation of linear kernel alignment with sketching techniques that give decent theoretical guarantees, and demonstrates that the alignment score is not only a suitable similarity measure, but also a score that is predictive of transferability of a pretrained neural network on a downstream task. Future work includes using the alignment score to determine when the pretrained model needs to be finetuned as data samples come in.

Chapter 3 pushes the limit of transfer learning without finetuning by accumulating multiple layers, and the layer selection is done through learning kernels. Our proposed method performs as well as accumulating all layers, and better than using only the top layer. Further speedup comes from using the Nyström method for obtaining low-rank approximations at individual layers. With the same time budget, our method also outperforms directly finetuning the pretrained neural network.

Chapter 4 considers the algorithm for learning kernels for ensemble pruning. Our proposed method improves performance on top of bagging and boosting with many fewer remaining predictors, and prunes more predictors than other related work.

Through these three chapters of discussion, we empirically examine the practicality of linear kernel alignment and the algorithm of learning kernels in the field of domain generalisation, and with the help of the theory behind the algorithm for learning kernels, we are able to interpret and analyse experimental results with confidence.

# Bibliography

[Alt92] Naomi S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[AS64] Milton Abramowitz and Irene A Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55. US Government printing office, 1964.

[BC14] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances on Neural Information Processing Systems*, 2014.

[BFOS83] Leo Breiman, Jerome H. Friedman, Richard Olshen, and Charles J. Stone. Classification and regression trees. 1983.

[BG19] Alon Brutzkus and Amir Globerson. Why do larger models generalize better? A theoretical perspective via the XOR problem. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 822–830, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[BNS06] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

[Bre96] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[BY02] Peter Bühlmann and Bin Yu. Analyzing bagging. *The Annals of Statistics*, 30(4):927–961, 2002.

[CBIK$^+$18] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *ArXiv*, abs/1812.01718, 2018.

[CKEST06] Nello Cristianini, Jaz Kandola, Andre Elisseeff, and John Shawe-Taylor. On kernel target alignment. In *Innovations in Machine Learning*, pages 205–256. Springer, 2006.

[CLM$^+$15] Michael B Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 181–190, 2015.

[CM06] Corinna Cortes and Mehryar Mohri. On transductive regression. In *Advances in Neural Information Processing Systems*, 2006.

[CM11] Corinna Cortes and Mehryar Mohri. Domain adaptation in regression. In *Algorithmic Learning Theory*, 2011.

[CMR11] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Ensembles of kernel predictors. In *Annual Conference on Uncertainty in Artificial Intelligence*, 2011.

[CMR12] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *Journal for Machine Learning Research*, 13:795–828, 2012.

[CNL11] Adam Coates, Andrew Y. Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[CNW15] Michael B Cohen, Jelani Nelson, and David P Woodruff. Optimal approximate matrix product in terms of stable rank. In *International Colloquium on Automata, Languages and Programming*, 2015.

[CW13] Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *ACM Symposium on the Theory of Computing*, 2013.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

[DDS$^+$09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[DG03] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

[Dru97] Harris Drucker. Improving regressors using boosting techniques. In *International Conference on Machine Learning*, volume 97, pages 107–115. Citeseer, 1997.

[DYXY07] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for transfer learning. In *International Conference on Machine Learning*, 2007.

[Elm90] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[FH17] Nicholas Frosst and Geoffrey E. Hinton. Distilling a neural network into a soft decision tree. In *Workshop on Comprehensibility and Explanation in AI and ML*, volume abs/1711.09784, 2017.

[FM82] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[Fri91] Jerome H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pages 1–67, 1991.

[Fri01] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[FS97] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[GBSS05] Arthur Gretton, Olivier Bousquet, Alexander J. Smola, and Bernhard Schölkopf. Measuring statistical dependence with Hilbert-Schmidt norms. In *Algorithmic Learning Theory*, 2005.

[GHS+05] Arthur Gretton, Ralf Herbrich, Alexander J. Smola, Olivier Bousquet, and Bernhard Schölkopf. Kernel methods for measuring independence. *Journal of Machine Learning Research*, 6:2075–2129, 2005.

[GM16] Alex Gittens and Michale W. Mahoney. Revisiting the nyström method for improved large-scale machine learning. *The Journal of Machine Learning Research*, 17(1):3977–4041, 2016.

[GPSW17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, 2017.

[HRZZ09] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and Its Interface*, 2:349–360, 2009.

[HS97] Sepp Hochreiter and Juergen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[HSW89] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[HVD15] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition)*, pages 770–778, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[Jag19] Meena Jagadeesan. Understanding sparse jl for feature hashing. In *Advances in Neural Information Processing Systems*, 2019.

[Joa98] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[JVZ19] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference*, 2019.

[KMT12] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Sampling methods for the Nyström method. *Journal for Machine Learning Research*, 13:981–1006, 2012.

[KNLH19] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey E. Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, 2019.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, Toronto, 2009.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. 1998.

[MBS13] Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. Domain generalization via invariant feature representation. In *International Conference on Machine Learning*, 2013.

[MFSS17] Krikamol Muandet, Kenji Fukumizu, Bharath K. Sriperumbudur, and Bernhard Schölkopf. Kernel mean embedding of distributions: A review and beyonds. *ArXiv*, abs/1605.09522, 2017.

[MRB18] Ari S. Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Advances in Neural Information Processing Systems*, 2018.

[NHSA20] Cuong Nguyen, Tal Hassner, Matthias Seeger, and Cedric Archambeau. LEEP: A new measure to evaluate transferability of learned representations. In *International Conference on Machine Learning*, 2020.

[NMC05] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *International Conference on Machine Learning*, 2005.

[NN13] Jelani Nelson and Huy L Nguyên. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *2013 IEEE 54th annual symposium on foundations of computer science*, pages 117–126. IEEE, 2013.

[NWC$^+$11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[PGC$^+$17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[PGM$^+$19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[PP13] Ninh Pham and Rasmus Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *SIGKDD Conference on Knowledge Discovery and Data Mining*, 2013.

[PY10] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010.

[RDS$^+$15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252, 2015.

[RGYSD17] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems*, 2017.

[RR$^+$07] Ali Rahimi, Benjamin Recht, et al. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2007.

[SEZ$^+$14] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. 2nd international conference on learning representations, iclr 2014. In *International Conference on Learning Representations*, 2014.

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[TdS20] Shuai Tang and Virginia R de Sa. Deep transfer learning with ridge regression. *arXiv preprint arXiv:2006.06791*, 2020.

[TMD$^+$20] Shuai Tang, Wesley J Maddox, Charlie Dickens, Tom Diethe, and Andreas Damianou. Similarity of neural networks with gradients. *arXiv preprint arXiv:2003.11498*, 2020.

[TN20] Shuai Tang and Cuong Nguyen. Personal communication. 2020.

[TPB00] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[TPV09] Grigorios Tsoumakas, Ioannis Partalas, and Ioannis Vlahavas. An ensemble pruning primer. In *Applications of supervised and unsupervised ensemble methods*, pages 1–13. Springer, 2009.

[UT19] Madeleine Udell and Alex Townsend. Why are big data matrices approximately low rank? *SIAM Journal on Mathematics of Data Science*, 1(1):144–160, 2019.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[WBW$^+$10] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[WLI$^+$05] Jason Weston, Christina Leslie, Eugene Ie, Dengyong Zhou, Andre Elisseeff, and William Stafford Noble. Semi-supervised protein classification using cluster kernels. *Bioinformatics*, 21(15):3241–3247, 2005.

[WLK$^+$20] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *ArXiv*, abs/2006.04768, 2020.

[Wol92] David H Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[Woo14] David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10:1–157, 2014.

[XGD$^+$17] Saining Xie, Ross B. Girshick, P. Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2017.

[YCBL14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, 2014.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *The British Machine Vision Conference*, 2016.