1-1-2019

Improving random forests by feature dependence analysis

Silu Zhang

IMPROVING RANDOM FORESTS BY FEATURE DEPENDENCE ANALYSIS

A Dissertation
Presented in Partial Fulfillment of Requirements
for the Degree of Doctor of Philosophy
in the Computer and Information Science
The University of Mississippi

by

Silu Zhang

August 2019

ABSTRACT

Random forests (RFs) have been widely used for supervised learning tasks because of their high prediction accuracy, good model interpretability and fast training process. However, they are not able to learn from local structures as convolutional neural networks (CNNs) do, when there exists high dependency among features. They also cannot utilize features that are jointly dependent on the label but marginally independent of it. In this dissertation, we present two approaches to address these two problems respectively by dependence analysis. First, a local feature sampling (LFS) approach is proposed to learn and use the locality information of features to group dependent/correlated features to train each tree. For image data, the local information of features (pixels) is defined by the 2-D grid of the image. For non-image data, we provided multiple ways of estimating this local structure. Our experiments shows that RF with LFS has reduced correlation and improved accuracy on multiple UCI datasets. To address the latter issue of random forest mentioned, we propose a way to categorize features as marginally dependent features and jointly dependent features, the latter is defined by *minimum dependence sets* (MDS's) or by *stronger dependence sets* (SDS's). Algorithms to identify MDS's and SDS's are provided. We then present a *feature dependence mapping* (FDM) approach to map the jointly dependent features to another feature space where they are marginally dependent. We show that by using FDM, decision tree and RF have improved prediction performance on artificial datasets and a protein expression dataset.

ACKNOWLEDGEMENTS

This dissertation could not be completed without the guidance of my advisor, the help of all faculties and friends, and the support of my family.

First of all, I would like to sincerely thank my advisor, Dr. Yixin Chen for bringing me into the field of machine learning research and giving me sufficient training to build critical thinking skills. It was him who helped me find a profession that I am passionate about and can make a living from. At the time I joined his group, I had a purely biology background and knew little about computation or machine learning, but he was very good at setting appropriate goals for me at different stages of my PhD study. His guidance and encouragement helped me grow and my confidence was built along the way. In the past 4 years, he never "pushed" me for anything. However, his trust has always been a pressure for me, a pressure of not disappointing him.

Next, I want to express my gratitude to all faculties that helped me thrive in the past four years. I am extremely grateful to Dr. Dawn E. Wilkins for always giving me critical suggestions. Thanks for suggesting me to start doing research at an early stage of my PhD, reminding me that as a computer scientist, I should make more contributions on the computation side than on the application side, and more. Thanks Dr. Xin Dang for enhancing my mathematical and statistical skills. It is also her work of Gini correlation that motivated me to study the joint dependence between feature and label. Thanks Dr. H. Conrad Cunningham for building my programming skills as well as improving my writing skill by correcting my grammars on each of my assignment/report/thesis. And thanks Dr. Naeemul Hassan for training me on analyzing text data. I could not finish my PhD so smoothly without the help from any of you.

I would also like to take this opportunity to thank Dr. Walter (Wenchuang) Hu, who

was my PhD advisor at University of Texas at Dallas. He taught me essential skills like time management and multitasking to learn and work in an efficient way. More importantly, he let me know the importance of inner drive in conducting research. Even though I was not able to finish my PhD degree in his group and made up my mind to switch my major to computer science, he was very supportive of my decision. I do miss the time I spent in Dr. Hu's Lab and value the training I received there.

I also greatly value my internship experiences in the past two summers. Thanks Bryan Glazer and Lijing Xu, my mentors at Fedex, for training me on using Spark and analyzing big data. Thanks Dr. Xiang Chen and Wenan Chen for providing me the precious opportunity to work at St. Jude children's research hospital and hands-on experience with single cell RNA-seq data. The statistical concepts I learned from my internship project were very helpful in advancing my PhD research. I also learned from Wenan that problems encountered in research are nothing to be afraid of. Instead, they provide opportunities of making contributions by solving them.

Last but not least, I want to thank my husband, Ning Sun, for being so supportive of my PhD study these years, even if I chose a university that is 100 miles from where he was working 4 years ago. Ning, thank you for putting so many miles on your new car and taking as many responsibilities as you could in taking care of our boy with me regardless. Moving to Mississippi for our marriage has been the best decision I have ever made in my life!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOTATIONS

x	A scalar (ineger or real)
$\boldsymbol{x}$	A vector
X	A random variable (1-dimensional)
$\mathbf{X}$	A random vector (multi-dimensional)
$\mathbb{R}$	The set of real numbers
$\mathbb{R}^p$	The set of real valued vectors in the p-dimensional space
f_X	The characteristic function of X
$f_{X,Y}$	The joint characteristic function of X and Y
F_X	The cumulative distribution function (CDF) of X
$F_{X,Y}$	The joint CDF of X and Y
$\mathbb{E}_X f(X)$	Expectation of $f(X)$ with respect to X
$\|\|\cdot\|\|$	A norm
$\|\cdot\|$	The Euclidean norm, or L^2 norm
$X \perp\!\!\!\perp Y$	X and Y are independent
$X \not\perp\!\!\!\perp Y$	X and Y are dependent
$\mathcal{S}$	A set

$\overline{\mathcal{S}}$	The complement of $\mathcal{S}$
$\mathcal{S}_1 \cup \mathcal{S}_2$	The union of $\mathcal{S}_1$ and $\mathcal{S}_2$
$\bigcup_{i=1}^{s} \mathcal{S}_i$	The union of all $\mathcal{S}_i$, $i = 1, ...s$, i.e., $\mathcal{S}_1 \cup ... \cup \mathcal{S}_s$
$\mathcal{S}_1 \setminus \mathcal{S}_2$	Set subtraction, i.e., the set containing the elements of $\mathcal{S}_1$ that are not in $\mathcal{S}_2$
$\mathcal{S}_m$	The set of marginally dependent features
$\mathcal{S}_I$	The set of independent features
$\mathbf{X}_\mathcal{S}$	Data $\mathbf{X}$ represented using features in $\mathcal{S}$
$\{1, -1\}$	A set containing 1 and -1
$\Gamma(n)$	The complete gamma function.
$\mathcal{N}(\mu, \sigma^2)$	A normal distribution with mean μ and variance σ^2
p_{H_1}	The p-value of a statistical test with the alternative hypothesis being H_1
p_m	The p-value of a marginal dependence test
p_{joint}	The p-value of a joint dependence test

CHAPTER 1

INTRODUCTION

Among all supervised learning models, random forest is one of the most popular algorithms and has been applied to almost every field. It has high prediction performance, good model interpretability, and a fast training process. In this chapter, we will explain what a random forest is (Section 1.1), the existing types of ensemble learners (Section 1.2), why it has the mentioned advantages (Section 1.3, 1.4 and 1.5), its weakness and how to improve it by overcoming the weakness (Section 1.7).

1.1 Random Forests

Random Forests evolve from ensemble of decision trees. Bagging (Breiman, 1996) and boosting (Freund et al., 1996; Freund and Schapire, 1997) are the first two main approaches of using tree ensembles. Bagging, short for bootstrap aggregating, is to use bootstrap samples (sampling with replacement) of the training data to train each tree. Boosting assigns higher weights to misclassified examples to boost performance. Bagging can also be combined with boosting by using the normalized sample weights as the sampling distribution (Quinlan, 1993). (Dietterich, 2000) proposed to use random split from the K best splits. (Amit and Geman, 1997) used a random selection of the features to decide the best split on image classification and feature selection tasks. (Breiman, 2001) generalized this idea and showed the better performance of this approach – using bootstrap and best split from a random subset of features – than other variations and it becomes the most popular tree ensemble technique, known as random forest.

Random forest has been applied to a wide range of classification tasks, e.g, image classification and annotation (Bosch et al., 2007; Fu et al., 2012; Du et al., 2015; Huynh

et al., 2015), cancer prediction (Statnikov et al., 2008; Nguyen et al., 2013; Okun and Priisalu, 2007; Wu et al., 2003), speach recognition (Su et al., 2007; Xue and Zhao, 2008), remote sensing (Belgiu and Drăguţ, 2016), etc. In addition to serving as a classifier, random forest has also been extensively used as a feature selection method, because it evaluates feature importance during the training process. For example, as a gene selection method on the microarray data (Díaz-Uriarte and De Andres, 2006; Nguyen et al., 2013). Random forest has shown competitive performance as a feature selector compared with other popular feature selection methods, such as 1-norm SVM, SVM-RFE and mutual information (Zhang et al., 2019).

1.2 Ensemble Learners

Random forest is a type of *ensemble learners*. An *ensemble* consists of a number of learners known as *base learners*. The prediction of the ensemble is based on the predictions of all of its base learners. The combination of multiple weak learners usually results in a much stronger learner. Random forest is just an ensemble of Decision Trees. Other examples of ensemble learners are the nearest-neighbour ensemble (Zhou and Yu, 2005) and neural network ensemble (Krogh and Vedelsby, 1995). However, they are not as commonly used as random forest. The nearest-neighbour ensemble is not as competitive as random forests because it lacks randomness in constructing the base learners. The performance of an ensemble relies on not only the performance but also the diversity of its base learners (Krogh and Vedelsby, 1995). Similarly, it is also difficult to introduce diversity to neural networks. Krogh and Vedelsby achieved this by using cross-validation to construct the ensemble, i.e., different subsets of the training set were used to train the base neural networks of the ensemble. However, networks are not guaranteed to converge when the training set changes. Besides, training each neutral network is already very time consuming, which limits the number of base learners in the ensemble. A large network ensemble usually requires external computation resource such as a cluster.

1.3 Model Interpretability

A key reason for the success of random forest in multiple fields is its interpretability. When given the options of a highly accurate but black-box model (e.g. deep networks) and a relatively less accurate but more interpretable model (e.g. random forests), people tend to favor the latter, especially in medical domains. The interpretability of a random forest comes from its rule-based decision tree base learners. Here we give a brief review on decision tree classifiers. A more comprehensive explanation of decision trees can be found in (Breiman, 2017).

A decision tree model is a binary tree with each node (called a decision node) associated with a decision rule. Usually, the decision rule is about determining whether to go to the left or right child of the node based on the value of a feature. Each node partitions the data into two parts, therefore it is also called a *split*. For classification problems, at each node, any feature can be used to partition the data, but the feature achieving the maximum impurity decrease is called the *best split* and is used as the feature at that node. The training data is used to grow the tree until a stopping criterion is met. Each leaf node is associated with the label of the majority of training examples falling into that node. A test example starts from the root node and follows the path defined by the rules and finally reaches a leaf node. The label of the leaf node is predicted as the label of the test example. This ensures the interpretability of the decision tree classifier since one can always retrieve the information regarding how the decision is made to classify a test example by tracing the path.

1.4 Bias-Variance Decomposition

To better understand why an ensemble performs better than its base learners, it is helpful to decompose the generalization error of the ensemble into bias and variance terms, as shown in Krogh and Vedelsby (1995). Consider the task of learning a function f such that $f(\boldsymbol{x}) = y$ for any example $(\boldsymbol{x}, y)$. The distribution of input $\boldsymbol{x}$ is $p(\boldsymbol{x})$. The following

results can be generalized to several output variables and applied to any ensemble method.

The output of the ensemble $\overline{V}(\boldsymbol{x})$ is a weighted average over the outputs of all its base learners, i.e.,

$$\overline{V}(\boldsymbol{x}) = \sum_k \omega_k V_k(\boldsymbol{x}), \tag{1.1}$$

where $V_k(\boldsymbol{x})$ and ω_k are the output and the weight of the k^{th} learner, respectively. Note that the sum of the weights of all learners should be one. The *ambiguity* of a single learner on input $\boldsymbol{x}$ is defined as $a_k(\boldsymbol{x}) = (V_k(\boldsymbol{x}) - \overline{V}(\boldsymbol{x}))^2$, which describes the degree of disagreement between a single learner and the ensemble. The *ensemble ambiguity* on input $\boldsymbol{x}$ is the weighted average ambiguity over all its base learners, i.e.,

$$\overline{a}(\boldsymbol{x}) = \sum_k \omega_k a_k(\boldsymbol{x}) = \sum_k \omega_k (V_k(\boldsymbol{x}) - \overline{V}(\boldsymbol{x}))^2, \tag{1.2}$$

which can also be seen as the weighted variance around the weighted mean. Expand (1.2) to get

$$\overline{a}(\boldsymbol{x}) = \sum_k \omega_k V_k(\boldsymbol{x})^2 - 2\overline{V}(\boldsymbol{x}) \sum_k \omega_k V_k(\boldsymbol{x}) + \overline{V}(\boldsymbol{x})^2 \sum_k \omega_k. \tag{1.3}$$

Using (1.1) and $\sum_k \omega_k = 1$, (1.3) becomes

$$\overline{a}(\boldsymbol{x}) = \sum_k \omega_k V_k(\boldsymbol{x})^2 - \overline{V}(\boldsymbol{x})^2. \tag{1.4}$$

We can rewrite (1.4) as

$$\begin{aligned} \overline{a}(\boldsymbol{x}) &= \sum_k \omega_k V_k(\boldsymbol{x})^2 + y^2 - 2y\overline{V}(\boldsymbol{x}) - y^2 + 2y\overline{V}(\boldsymbol{x}) - \overline{V}(\boldsymbol{x})^2 \\ &= \sum_k \omega_k V_k(\boldsymbol{x})^2 + \sum_k \omega_k y^2 - 2y \sum_k \omega_k V_k(\boldsymbol{x}) - (y - \overline{V}(\boldsymbol{x}))^2 \\ &= \sum_k \omega_k (y - V_k(\boldsymbol{x}))^2 - (y - \overline{V}(\boldsymbol{x}))^2 \end{aligned} \tag{1.5}$$

Defining $\epsilon_k(\boldsymbol{x}) = (y - V_k(\boldsymbol{x}))^2$, which is the quadratic error of the k^{th} learner, $\overline{\epsilon}(\boldsymbol{x}) =$

$\sum_k \omega_k \epsilon_k(\boldsymbol{x})$, which is the weighted average error of base learners, and $e(\boldsymbol{x}) = (y - \overline{V}(\boldsymbol{x}))^2$ as the error of the ensemble, we can rewrite (1.5) as

$$e(\boldsymbol{x}) = \overline{\epsilon}(\boldsymbol{x}) - \overline{a}(\boldsymbol{x}). \tag{1.6}$$

(1.6) suggests that ensemble error can be decomposed as base learner error and ensemble ambiguity. The $\overline{\epsilon}(\boldsymbol{x})$ term can also be viewed as the bias of the model and $\overline{a}(\boldsymbol{x})$ reflects the variance. If we average over the input distribution, (6) becomes

$$\begin{aligned} \int p(\boldsymbol{x}) e(\boldsymbol{x}) d\boldsymbol{x} &= \int p(\boldsymbol{x}) \overline{\epsilon}(\boldsymbol{x}) d\boldsymbol{x} - \int p(\boldsymbol{x}) \overline{a}(\boldsymbol{x}) d\boldsymbol{x} \\ E &= \overline{E} - \overline{A}, \end{aligned} \tag{1.7}$$

where $E = p(\boldsymbol{x}) e(\boldsymbol{x}) d\boldsymbol{x}$, which is the generalization error of the ensemble, $\overline{E} = \int p(\boldsymbol{x}) \overline{\epsilon}(\boldsymbol{x}) d\boldsymbol{x} = \sum_k \omega_k \int p(\boldsymbol{x}) \epsilon_k(\boldsymbol{x})$, which is the weighted average of generalization error of base learners, and $\overline{A} = \int p(\boldsymbol{x}) \overline{a}(\boldsymbol{x}) d\boldsymbol{x}$, which is the expected ensemble ambiguity over input distribution. Since $\overline{A}$ is non-negative, (1.7) suggests that the performance of the ensemble is always better than the average performance of its base learners. To achieve better performance, we want to improve the performance of the base learners as well as increase the ensemble ambiguity, i.e., the more base learners disagree on each other, the better.

For neural network ensembles, where the number of base learners is small (usually less than 10), it is beneficial to optimize the weights. However, for random forests, which usually consist of hundreds or thousands of trees, uniform weights are used due to the computation concerns of finding the optimal weights.

1.5 Strength-Correlation Decomposition

Breiman derived an upper bound for the generalization error of random forests in terms of strength and correlation, where strength measures the performance of individual trees and correlation measures the dependence between them (Breiman, 2001). This is shown

in the following theorem (Breiman, 2001).

Theorem 1. An upper bound for the generalization error of random forests is given by

$$PE^* \leqslant \bar{\rho}(1 - s^2)/s^2,$$

where PE^* is the generalization error, s is strength and ρ is correlation.

Since the motivation of this work is mostly based on Theorem 1, here we show the proof of Theorem 1 with sufficient derivations. The original proof is in (Breiman, 2001).

Proof. To start the proof, we have the following definitions for random forests.

Definition 2. A random forest consists of a collection of decision tree classifiers $\{h(\mathbf{X}, \Theta_k), k = 1, ...\}$, where $\mathbf{X}$ is the input vector, and the $\{\Theta_k\}$ are independent identically distributed random vectors that used to generate trees.

Definition 3. The margin function for a random forest is

$$mr(\mathbf{X}, Y) = P_\Theta(h(\mathbf{X}, \Theta) = Y) - \max_{j \neq Y} P_\Theta(h(\mathbf{X}, \Theta) = j),$$

where $\mathbf{X}, Y$ are random vectors drawn from the input space and the subscript Θ indicates that the probability is over the Θ space.

We also need the following theorem proved in (Breiman, 2001).

Theorem 4. As the number of trees increases, for almost surely all sequences $\Theta_1, ..., PE^*$ coverages to

$$PE^* = P_{\mathbf{X},Y}(mr(\mathbf{X}, Y) < 0),$$

here the subscripts $\mathbf{X}, Y$ suggests that the probability is over the $\mathbf{X}, Y$ space.

Now define the *strength* of the set of classifiers $\{h(\mathbf{X}, \Theta)\}$ as

$$s = \mathbb{E}_{\mathbf{X},Y} mr(\mathbf{X}, Y), \tag{1.8}$$

and assume $s \geqslant 0$, from Chebychev's inequality we have

$$P_{\mathbf{X},Y}(|mr(\mathbf{X},Y) - s| \geqslant s) \leqslant \text{Var}(mr)/s^2,$$

where $Var(mr)$ is the simplified notation for $Var(mr(\mathbf{X},Y))$. Expanding the left hand side,

$$P_{\mathbf{X},Y}(mr(\mathbf{X},Y) \geqslant 2s) + P_{\mathbf{X},Y}(mr(\mathbf{X},Y) \leqslant 0) \leqslant \text{Var}(mr)/s^2.$$

Now it is obvious that

$$P_{\mathbf{X},Y}(mr(\mathbf{X},Y) < 0) \leqslant \text{Var}(mr)/s^2$$

From Theorem 4 we then have

$$PE^* \leqslant \text{Var}(mr)/s^2. \tag{1.9}$$

Var(mr) can be further decomposed in the following way. Let

$$\hat{j}(\mathbf{X},Y) = \arg\max_{j \neq Y} P_\Theta(h(\mathbf{X},\Theta) = j),$$

which is the class other than the true label that has highest votes. Then we can rewrite the margin as

$$mr(\mathbf{X},Y) = P_\Theta(h(\mathbf{X},\Theta) = Y) - P_\Theta(h(\mathbf{X},\Theta) = \hat{j}(\mathbf{X},Y)) \tag{1.10}$$

$$= \mathbb{E}_\Theta[I(h(\mathbf{X},\Theta) = Y) - I(h(\mathbf{X},\Theta) = \hat{j}(\mathbf{X},Y)], \tag{1.11}$$

where $I(\cdot)$ is the indicator function.

Definition 5. Define the raw margin function as

$$rmg(\Theta,\mathbf{X},Y) = I(h(\mathbf{X},\Theta) = Y) - I(h(\mathbf{X},\Theta) = \hat{j}(\mathbf{X},Y).$$

Then margin is the expectation of raw margin with respect to Θ, i.e.,

$$mr(\mathbf{X}, Y) = \mathbb{E}_{\Theta}[rmg(\Theta, \mathbf{X}, Y)]. \tag{1.12}$$

For any function f, the following holds

$$\mathbb{E}^2_{\Theta}[f(\Theta)] = \mathbb{E}_{\Theta,\Theta'}[f(\Theta)f(\Theta')], \tag{1.13}$$

if Θ and Θ' are independent with the same distribution. We use (1.13) multiple times for the following derivations. From (1.12) we have

$$\begin{aligned} mr(\mathbf{X}, Y)^2 &= \mathbb{E}^2_{\Theta}[rmg(\Theta, \mathbf{X}, Y)] \\ &= \mathbb{E}_{\Theta,\Theta'}[rmg(\Theta, \mathbf{X}, Y)rmg(\Theta', \mathbf{X}, Y)]. \end{aligned} \tag{1.14}$$

Now we can decompose $\mathrm{Var}(mr)$:

$$\mathrm{Var}(mr) = \mathbb{E}_{\mathbf{X},Y}[mr(\mathbf{X}, Y)^2] - \mathbb{E}^2_{\mathbf{X},Y}[mr(\mathbf{X}, Y)] \tag{1.15}$$

$$= \mathbb{E}_{\mathbf{X},Y}[\mathbb{E}^2_{\Theta} rmg(\Theta, \mathbf{X}, Y)] - [\mathbb{E}_{\mathbf{X},Y}\mathbb{E}_{\Theta}[rmg(\Theta, \mathbf{X}, Y)]]^2 \tag{1.16}$$

Using (1.13), the first term of (1.16) can be further derived as

$$\begin{aligned} \mathbb{E}_{\mathbf{X},Y}[\mathbb{E}^2_{\Theta}[rmg(\Theta, \mathbf{X}, Y)]] &= \mathbb{E}_{\mathbf{X},Y}\mathbb{E}_{\Theta,\Theta'}[rmg(\Theta, \mathbf{X}, Y)rmg(\Theta', \mathbf{X}, Y)] \\ &= \mathbb{E}_{\Theta,\Theta'}\mathbb{E}_{\mathbf{X},Y}[rmg(\Theta, \mathbf{X}, Y)rmg(\Theta', \mathbf{X}, Y)], \end{aligned} \tag{1.17}$$

and the second term of (1.16) is

$$\begin{aligned} [\mathbb{E}_{\mathbf{X},Y}\mathbb{E}_{\Theta}[rmg(\Theta, \mathbf{X}, Y)]]^2 &= \mathbb{E}^2_{\Theta}[\mathbb{E}_{\mathbf{X},Y}[rmg(\Theta, \mathbf{X}, Y)]] \\ &= \mathbb{E}_{\Theta,\Theta'}[\mathbb{E}_{\mathbf{X},Y}[rmg(\Theta, \mathbf{X}, Y)]\mathbb{E}_{\mathbf{X},Y}[rmg(\Theta', \mathbf{X}, Y)]]. \end{aligned} \tag{1.18}$$

Substitute (1.17) and (1.18) into (1.16),

$$\begin{aligned}
\mathrm{Var}(mr) &= \mathbb{E}_{\Theta,\Theta'}[\mathbb{E}_{\mathbf{X},Y}[rmg(\Theta,\mathbf{X},Y)rmg(\Theta',\mathbf{X},Y)] \\
&\quad - \mathbb{E}_{\mathbf{X},Y}[rmg(\Theta,\mathbf{X},Y)]\mathbb{E}_{\mathbf{X},Y}[rmg(\Theta',\mathbf{X},Y)]] \\
&= \mathbb{E}_{\Theta,\Theta'}[\mathrm{Cov}_{\mathbf{X},Y}(rmg(\Theta,\mathbf{X},Y), rmg(\Theta',\mathbf{X},Y))] \\
&= \mathbb{E}_{\Theta,\Theta'}[\rho_{\mathbf{X},Y}(rmg(\Theta,\mathbf{X},Y), rmg(\Theta',\mathbf{X},Y))\cdot \\
&\quad sd_{\mathbf{X},Y}(rmg(\Theta,\mathbf{X},Y))sd_{\mathbf{X},Y}(rmg(\Theta',\mathbf{X},Y))]. \qquad (1.19)
\end{aligned}$$

To simplify the notations, let $\rho(\Theta,\Theta') = \rho_{\mathbf{X},Y}(rmg(\Theta,\mathbf{X},Y), rmg(\Theta',\mathbf{X},Y))$, and $sd(\Theta) = sd_{\mathbf{X},Y}(rmg(\Theta,\mathbf{X},Y))$, then (1.19) becomes

$$\mathrm{Var}(mr) = \mathbb{E}_{\Theta,\Theta'}[\rho(\Theta,\Theta')sd(\Theta)sd(\Theta')]. \qquad (1.20)$$

Let

$$\bar{\rho} = \mathbb{E}_{\Theta,\Theta'}[\rho(\Theta,\Theta')sd(\Theta)sd(\Theta')]/\mathbb{E}_{\Theta,\Theta'}[sd(\Theta)sd(\Theta')],$$

which is the mean value of the correlation, then (1.19) becomes

$$\begin{aligned}
\mathrm{Var}(mr) &= \bar{\rho}\mathbb{E}_{\Theta,\Theta'}[sd(\Theta)sd(\Theta')] \\
&= \bar{\rho}\mathbb{E}^2_{\Theta}[sd(\Theta)] \qquad (1.21) \\
&\leqslant \bar{\rho}\mathbb{E}_{\Theta}[sd(\Theta)^2] \\
&= \bar{\rho}\mathbb{E}_{\Theta}[\mathrm{Var}(\Theta)], \qquad (1.22)
\end{aligned}$$

where $\mathrm{Var}(\Theta)$ is the simplified notation for $\mathrm{Var}_{\mathbf{X},Y}(rmg(\Theta,\mathbf{X},Y))$. Expanding $\mathbb{E}_\Theta[\mathrm{Var}(\Theta)]$,

$$\begin{aligned}
\mathbb{E}_\Theta[\mathrm{Var}(\Theta)] &= \mathbb{E}_\Theta[\mathrm{Var}_{\mathbf{X},Y}(rmg(\Theta,\mathbf{X},Y))] \\
&= \mathbb{E}_\Theta\{\mathbb{E}_{\mathbf{X},Y}[rmg(\Theta,\mathbf{X},Y)^2] - \mathbb{E}^2_{\mathbf{X},Y}[rmg(\Theta,\mathbf{X},Y)]\} \\
&= \mathbb{E}_\Theta\mathbb{E}_{\mathbf{X},Y}[rmg(\Theta,\mathbf{X},Y)^2] - \mathbb{E}_\Theta[\mathbb{E}^2_{\mathbf{X},Y}[rmg(\Theta,\mathbf{X},Y)]] \\
&\leqslant \mathbb{E}_\Theta\mathbb{E}_{\mathbf{X},Y}[rmg(\Theta,\mathbf{X},Y)^2] - [\mathbb{E}_\Theta\mathbb{E}_{\mathbf{X},Y}[rmg(\Theta,\mathbf{X},Y)]]^2 \\
&= \mathbb{E}_\Theta\mathbb{E}_{\mathbf{X},Y}[rmg(\Theta,\mathbf{X},Y)^2] - [\mathbb{E}_{\mathbf{X},Y}\mathbb{E}_\Theta[rmg(\Theta,\mathbf{X},Y)]]^2 \qquad (1.23)
\end{aligned}$$

Notice that $\mathbb{E}_\Theta[rmg(\Theta,\mathbf{X},Y)]$ is $mr(\mathbf{X},Y)$ (1.12) and $\mathbb{E}_{\mathbf{X},Y}[mr(\mathbf{X},Y)]$ is s (1.8), then (1.23) becomes

$$\begin{aligned}
\mathbb{E}_\Theta[\mathrm{Var}(\Theta)] &\leqslant \mathbb{E}_\Theta\mathbb{E}_{\mathbf{X},Y}[rmg(\Theta,\mathbf{X},Y)^2] - s^2 \\
&\leqslant 1 - s^2 \qquad (1.24)
\end{aligned}$$

Putting (1.9), (1.22) and (1.24) together completes the proof for Theorem 1.

□

1.6 Out-of-Bag Estimates for Strength and Correlation

As described in (Breiman, 2001), strength and correlation can be estimated by out-of-bag estimates. To estimate strength s, let

$$Q(\mathbf{X},j) = \sum_k I(h(\mathbf{X},\Theta_k) = j; (y,\mathbf{X}) \notin T_{k,B}) / \sum_k I((y,\mathbf{X}) \notin T_{k,B}),$$

where $T_{k,B}$ is the bootstrap training set for the k'th tree. Therefore $Q(\mathbf{X},j)$ is the proportion of out-of-bag votes cast at $\mathbf{X}$ for class j, i.e., for trees that do not have $(y,\mathbf{X})$ in the bootstrap training set, how many (in proportion) vote for class j. $Q(\mathbf{X},j)$ can be used as an estimate

for $P_\Theta(h(\mathbf{X},\Theta)=j)$. From (1.10) we can estimate $mr(\mathbf{X},Y)$ as

$$Q(\mathbf{X},Y)-\max_{j\neq Y}Q(\mathbf{X},j). \tag{1.25}$$

And since strength is the expectation of $mr(\mathbf{X},Y)$ (1.8), the out-of-bag estimate for strength can be obtained by taking the average of (1.25) on the whole sample.

The estimate for correlation can be derived as follows. (1.21) can be rewritten as

$$\bar{\rho}=\mathrm{Var}(mr)/\mathbb{E}^2_\Theta[sd(\Theta)]. \tag{1.26}$$

From (1.8),(1.10) and (1.15), we have

$$\mathrm{Var}(mr)=\mathbb{E}_{\mathbf{X},Y}[(P_\Theta(h(\mathbf{X},\Theta)=Y)-P_\Theta(h(\mathbf{X},\Theta)=\hat{j}(\mathbf{X},Y))^2]-s^2 \tag{1.27}$$

Using the average of $(Q(\mathbf{X},Y)-\max_{j\neq Y}Q(\mathbf{X},j))^2$ as the estimate of the first term and using estimate of s gives the estimate of $\mathrm{Var}(mr)$. $\mathrm{Var}(\Theta)$ can be derived as

$$\begin{aligned}
\mathrm{Var}(\Theta)&=\mathrm{Var}_{\mathbf{X},Y}(rmg(\Theta,\mathbf{X},Y))\\
&=\mathrm{Var}_{\mathbf{X},Y}(I(h(\mathbf{X},\Theta)=Y)-I(h(\mathbf{X},\Theta)=\hat{j}(\mathbf{X},Y))\\
&=\mathbb{E}_{\mathbf{X},Y}[(I(h(\mathbf{X},\Theta)=Y)-I(h(\mathbf{X},\Theta)=\hat{j}(\mathbf{X},Y))^2]\\
&\quad-\mathbb{E}^2_{\mathbf{X},Y}[I(h(\mathbf{X},\Theta)=Y)-I(h(\mathbf{X},\Theta)=\hat{j}(\mathbf{X},Y))]\\
&=\mathbb{E}_{\mathbf{X},Y}[I(h(\mathbf{X},\Theta)=Y)^2]-2\mathbb{E}_{\mathbf{X},Y}[I(h(\mathbf{X},\Theta)=Y)I(h(\mathbf{X},\Theta)=\hat{j}(\mathbf{X},Y))]\\
&\quad+\mathbb{E}_{\mathbf{X},Y}[I(h(\mathbf{X},\Theta)=\hat{j}(\mathbf{X},Y))^2]-\{\mathbb{E}_{\mathbf{X},Y}[I(h(\mathbf{X},\Theta)=Y)]\\
&\quad-\mathbb{E}_{\mathbf{X},Y}[I(h(\mathbf{X},\Theta)=\hat{j}(\mathbf{X},Y))]\}^2\\
&=\mathbb{E}_{\mathbf{X},Y}[I(h(\mathbf{X},\Theta)=Y)]+\mathbb{E}_{\mathbf{X},Y}[I(h(\mathbf{X},\Theta)=\hat{j}(\mathbf{X},Y))]\\
&\quad-\{\mathbb{E}_{\mathbf{X},Y}[I(h(\mathbf{X},\Theta)=Y)]-\mathbb{E}_{\mathbf{X},Y}[I(h(\mathbf{X},\Theta)=\hat{j}(\mathbf{X},Y))]\}^2\\
&=p_1+p_2-(p_1-p_2)^2,
\end{aligned}$$

where $p_1 = \mathbb{E}_{\mathbf{X},Y}[I(h(\mathbf{X},\Theta) = Y)]$, $p_2 = \mathbb{E}_{\mathbf{X},Y}[I(h(\mathbf{X},\Theta) = \hat{j}(\mathbf{X},Y))]$. Then

$$sd(\Theta) = (p_1 + p_2 - (p_1 - p_2)^2)^{1/2}. \tag{1.28}$$

For the kth tree, $sd(\Theta_k)$ can be estimated by using out-of-bag samples to estimate p_1 and p_2. Then taking the average of all trees gives the estimate of $sd(\Theta)$. Since $p_1 + p_2 \leqslant 1$, $p1 \geqslant 0$ and $p_2 \geqslant 0$, it is easy to show that $0 \leqslant sd(\Theta) \leqslant 1$.

The strength and correlation estimates are useful to study the behavior of random forests during experiments. We implemented the strength and correlation estimates in our random forest classifier and the results are shown in Chapter 4.

1.7 Weaknesses of Random Forests

There are two major weaknesses of random forests. First, it cannot capture the structural information of the features. For example, image data are usually represented by pixels and the pixels have a 2-D spatial structure, which can be captured by a Convolutional Neural Network (CNN) but not by a random forest. The arrangement of the features does not affect the performance of a random forest. And this could be one explanation of why CNNs perform better than random forests on image data. The importance of using feature locality is further explained in Chapter 3 and our solution to overcome this weakness of random forest is demonstrated in Chapter 4.

The other weakness of random forests comes from its base learner, the decision tree classifier. At each node, only one feature will be used to decide the best split. This nature of decision tree classifiers presents challenges in solving problems like XOR, where two features must be used at the same time to determine the class of an example. In the XOR problem, the input $\mathbf{X}$ has two features, X_1 and X_2. The label Y is 1 if both X_1 and X_2 are equal to 0 or both are equal to 1, and Y is 0 otherwise. The two classes cannot be separated by splitting on X_1 or X_2. In the XOR problem, X_1 and X_2 are marginally independent on the label but jointly dependent on the label. However, random forests can only take advantage of features

that are marginally dependent on the label. In Chapter 5 & 6, we further demonstrate the difference between marginal and joint dependence and present our solution to make use of joint dependent features.

1.8 Outline

Both of the two approaches to improve random forests proposed in this paper require dependence analysis. Therefore, we first introduce statistical tools for dependence measure in Chapter 2. We then explain the concept of feature locality that motivated our local feature sampling (LFS) approach in Chapter 3 and Chapter 4 demonstrates LFS in detail. Chapter 5 explains the difference between marginal and joint dependence and experimentally demonstrates that decision trees can take advantage of jointly dependent features. We then propose a feature dependence mapping technique to overcome this issue in Chapter 6. Chapter 7 concludes the dissertation.

CHAPTER 2

DEPENDENCE MEASURE

Dependence (also know as *association*) is a statistical relationship between two random variables. Analysis of dependence is crucial in machine learning, including both supervised and unsupervised learning. In general, supervised learning aims to study the dependence between features and the label, while unsupervised learning focus on the dependence between samples. Our approach to improve random forests is based on the analysis of dependence between features as well as the dependence between features and the label. We therefore precede our introduction to some statistical measures of dependence that are used in or related to our approach.

The measure of dependence is usually called *correlation*, which a number that indicates the degree to which a pair of variables are related. However, the presence of correlation is not sufficient to infer causal relationship. Here we focus our discussion on correlation. Among various correlations developed (Mari and Kotz, 2001), the most commonly used one is Pearson correlation, which measures the linear dependence between two random variables. In spite of its simple computation, Pearson correlation does not capture non-linear dependence and only applies to 1-dimensional random variables. To address this issue, two distance based correlations were developed: *distance correlation* (Székely et al., 2007; Székely and Rizzo, 2009) and *Gini correlation* (Dang et al., 2018). They both characterize non-linear dependence, with distance correlation measuring dependence between two numerical and arbitrary dimensional random variables, and Gini correlation measuring dependence between one numerical random variable with arbitrary dimension and a categorical random variable. We present the definitions and computations of the above-mentioned correlations in this Chapter.

2.1 Pearson Correlation

The Pearson correlation is also called "Pearson product-moment correlation coefficient" or "Pearson's correlation coefficient". The Pearson correlation between two random variables X and Y, denoted as $\rho_{X,Y}$ or $\mathrm{Cor}(X,Y)$, is defined as

$$\rho_{X,Y} = \mathrm{Cor}(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y},$$

where μ_X and μ_Y are expected values of X and Y and σ_X and σ_Y are standard deviations. Pearson correlation is symmetric and has a value between -1 and 1. The value 1 indicates direct (increasing) linear relationship and the value -1 indicates a decreasing (inverse) linear relationship. When the sign is not of interest, the squared value of $\rho_{X,Y}$, called Pearson R-squared, is a preferred measure of dependence. It is interpreted as the ratio of the explained variation to the total variation. A higher value suggests a stronger linear dependence but a zero value does not suggest independence. The computation complexity for Pearson correlation is $O(n)$, where n is the sample size. The simplicity in computation and the ubiquitous liner dependence makes it an effective dependence measure for a wide range of applications.

However, Pearson correlation suffers two main drawbacks. First, it is not always able to detect dependency between a pair of dependent random variables. Examples are $Y = X^2$ and $Y = \cos(X)$. In both cases, clearly there is a dependence between X and Y, but the Pearson correlation is zero. More generally, we have $\rho = 0$ if $Y = f(X)$ over the interval $(-a, a)$, $f(x)$ is a single-valued function, symmetrical about $x = 0$, and the points are sampled uniformly from the intervals. Second, Pearson correlation can only be applied two 1-dimensional random variables which also have to be numerical. It can not be directly applied to feature selection task when the response variable is categorical. In the case of sure independent screening (SIS) procedure (Fan and Lv, 2008), the class variable is treated as numerical one to apply Pearson correlation for screening out irrelevant features. In (Hall, 2000), a Correlation-based Feature Selection (CFS) was proposed to use Pearson correlation

to select useful features for classification problems where the class variable can be either numerical or categorical. In the case of class variable being numerical, Pearson correlation is directly applied. For the categorical case, a weighted Pearson's correlation is used. Let Y be the class variable that can take values $y_1, ..., y_K$, the weighted Pearson correlation is defined as

$$\rho_{X,Y} = \sum_{k=1}^{K} p_k \rho_{X,Y_k},$$

where p_k is the probability of Y taking value y_k, and Y_k is a binary vector which takes value 1 if Y has value y_k or 0 otherwise. Y_k is treated numerically to calculate ρ_{X,Y_k}.

2.2 Distance Correlation

The disadvantages of Pearson correlation motivated the development of distance correlation (dCor) based on pairwise distances between sample elements (Székely et al., 2007; Székely and Rizzo, 2009). Here we present some key results from their work.

Let $\mathbf{X} \in \mathbb{R}^p$ and $\mathbf{Y} \in \mathbb{R}^q$ be two numerical random vectors, where p and q are positive integers. Let $f_{\mathbf{X}}$ and $f_{\mathbf{Y}}$ be the characteristic functions of $\mathbf{X}$ and $\mathbf{Y}$ and $f_{\mathbf{X},\mathbf{Y}}$ be the joint characteristic function. Thus, $\mathbf{X}$ and $\mathbf{Y}$ are independent if and only if $f_{\mathbf{X},\mathbf{Y}} = f_{\mathbf{X}} f_{\mathbf{Y}}$. Therefore a natural way to measure dependence between $\mathbf{X}$ and $\mathbf{Y}$ is to have a suitable norm to define the distance between $f_{\mathbf{X},\mathbf{Y}}$ and $f_{\mathbf{X}} f_{\mathbf{Y}}$.

Definition 6. The *distance covariance* (dCov) between random vectors $\mathbf{X}$ and $\mathbf{Y}$ with finite first moments is defined by

$$\begin{aligned} \mathrm{dCov}^2(\mathbf{X}, \mathbf{Y}) &= ||f_{\mathbf{X},\mathbf{Y}}(\boldsymbol{t}, \boldsymbol{s}) - f_{\mathbf{X}}(\boldsymbol{t}) f_{\mathbf{Y}}(\boldsymbol{s})||^2 \\ &= \frac{1}{c_p c_q} \int_{\mathbb{R}^{p+q}} \frac{|f_{\mathbf{X},\mathbf{Y}}(\boldsymbol{t}, \boldsymbol{s}) - f_{\mathbf{X}}(\boldsymbol{t}) f_{\mathbf{Y}}(\boldsymbol{s})|^2}{|\boldsymbol{t}|^{1+p} |\boldsymbol{s}|^{1+q}} d\boldsymbol{t} d\boldsymbol{s}, \end{aligned}$$

where

$$c_d = \frac{\pi^{(1+d)/2}}{\Gamma((1+d)/2)}.$$

Definition 7. The distance correlation (dCor) between random vectors $\mathbf{X}$ and $\mathbf{Y}$ with finite first moments is defined by

$$\mathrm{dCor}^2(\mathbf{X},\mathbf{Y}) = \begin{cases} \dfrac{\mathrm{dCov}^2(\mathbf{X},\mathbf{Y})}{\sqrt{\mathrm{dVar}^2(\mathbf{X})\mathrm{dVar}^2(\mathbf{Y})}}, & \mathrm{dVar}^2(\mathbf{X})\mathrm{dVar}^2(\mathbf{Y}) > 0; \\ 0, & \mathrm{dVar}^2(\mathbf{X})\mathrm{dVar}^2(\mathbf{Y}) = 0, \end{cases}$$

where dVar is the *distance variance* defined by $\mathrm{dVar}(\mathbf{X}) = \mathrm{dCor}(\mathbf{X},\mathbf{X})$.

Distance correlation is different from classical correlations in two fundamental ways:

1) $\mathrm{dCor}(\mathbf{X},\mathbf{Y})$ is defined for $\mathbf{X}$ and $\mathbf{Y}$ in arbitrary dimension.

2) $\mathrm{dCor}(\mathbf{X},\mathbf{Y}) = 0$ characterized independence of $\mathbf{X}$ and $\mathbf{Y}$.

It satisfies $0 \le \mathrm{dCor} \le 1$, and $\mathrm{dCor} = 0$ if and only if $\mathbf{X}$ and $\mathbf{Y}$ are independent. For the sake of implementations, we need sample versions of distance statistics as estimations.

Definition 8. The sample version of distance covariance dCov_n and distance variance dVar_n of n i.i.d. samples $(\boldsymbol{x}_i, \boldsymbol{y}_i), i = 1, ..., n$, drawn from their joint distribution, are defined by

$$\mathrm{dCov}_n^2(\mathbf{X},\mathbf{Y}) = \frac{1}{n^2}\sum_{k,l=1}^{n} A_{kl}B_{kl},$$

$$\mathrm{dVar}_n^2(\mathbf{X}) = \mathrm{dCov}_n^2(\mathbf{X},\mathbf{X}) = \frac{1}{n^2}\sum_{k,l=1}^{n} A_{kl}^2,$$

where A_{kl} and B_{kl} are calculated from the Euclidean distance matrices $a_{kl} = |\boldsymbol{x}_k - \boldsymbol{x}_l|_p$ and $b_{kl} = |\boldsymbol{y}_k - \boldsymbol{y}_l|_q$, i.e.,

$$A_{kl} = a_{kl} - \bar{a}_{k\cdot} - \bar{a}_{\cdot l} + \bar{a}_{\cdot\cdot}, \quad B_{kl} = b_{kl} - \bar{b}_{k\cdot} - \bar{b}_{\cdot l} + \bar{b}_{\cdot\cdot}, \quad k, l = 1, ..., n,$$

where

$$\bar{a}_{k\cdot} = \frac{1}{n}\sum_{l=1}^{n} a_{kl}, \quad \bar{a}_{\cdot l} = \frac{1}{n}\sum_{k=1}^{n} a_{kl}, \quad \bar{a}_{\cdot\cdot} = \frac{1}{n^2}\sum_{k,l=1}^{n} a_{kl},$$

and similarly for $\bar{b}_{k\cdot}$,$\bar{b}_{\cdot l}$ and $\bar{b}_{\cdot\cdot}$.

Definition 9. The sample distance correlation (dCor) between random vectors $\mathbf{X}$ and $\mathbf{Y}$ can then be defined by

$$\mathrm{dCor}_n^2(\mathbf{X},\mathbf{Y}) = \begin{cases} \dfrac{\mathrm{dCov}_n^2(\mathbf{X},\mathbf{Y})}{\sqrt{\mathrm{dVar}_n^2(\mathbf{X})\mathrm{dVar_n}^2(\mathbf{Y})}}, & \mathrm{dVar}_n^2(\mathbf{X})\mathrm{dVar}_n^2(\mathbf{Y}) > 0; \\ 0, & \mathrm{dVar}_n^2(\mathbf{X})\mathrm{dVar}_n^2(\mathbf{Y}) = 0. \end{cases}$$

dCov_n and dCor_n have the following properties:

1) $\mathrm{dCov}_n(\mathbf{X},\mathbf{Y}) \geqslant 0$;

2) $\mathrm{dCov}_n(\mathbf{X},\mathbf{Y}) = 0$ if and only if every sample observation is identical;

3) $0 \leqslant \mathrm{dCor}_n(\mathbf{X},\mathbf{Y}) \leqslant 1$.

As the sample size n goes to infinity, dCov_n converges to dCov and dCor_n converges to dCor. The sample version of distance correlation dCor_n provides a good estimation of dependence between two random variables of arbitrary dimension and is used in our work to measure dependence between two sets of features, with each set considered as one multi-dimensional random variable.

From Definition 8 and 9, it is clear that the computation complexity of $\mathrm{dCov}_n(\mathbf{X},\mathbf{Y})$ or $\mathrm{dCor}_n(\mathbf{X},\mathbf{Y})$ is $O(n^2)$. When n is large, using a random subset of samples for dependence estimations is desired for time efficiency.

Distance correlation has been proven to be a better feature selection method than Pearson correlation in (Li et al., 2012), where a distance correlation based sure independence screening (DC-SIS) was proposed. By using dCor as the dependent measure, DC-SIS avoids screening out some non-linearly dependent features which could be otherwise filtered out if Pearson correlation is applied. Same as SIS, the class variable in DC-SIS is treated numerically. Distance correlation has also been applied to measure the dependence between time

series random vectors (Székely and Rizzo, 2013; Zhou, 2012), where the random vectors are multi-dimensional. As pointed out in (Székely and Rizzo, 2009), dCor can also be used to measure feature-label dependence when the response is multivariate.

2.3 Gini Correlation

Gini correlation was developed by (Dang et al., 2018) to measure dependence between a numerical random variable and a categorical variable.

Definition 10. Let $\mathbf{X} \in \mathbb{R}^d$ be a numerical random vector with its CDF being F, Y be a categorical random variable which can take K values $L_1, ..., L_K$, and $\mathbf{X}_k$ be a random variable conditional on $Y = L_k$ with its CDF being F_k, the Gini covariance (gCov) between $\mathbf{X}$ and Y is defined as the weighted energy distance between $\mathbf{X}_k$ and $\mathbf{X}$, i.e.,

$$\mathrm{gCov}(\mathbf{X}, Y) = \sum_{k=1}^{K} p_k T(\mathbf{X}_k, \mathbf{X}),$$

where $p_k = P(Y = L_k)$, $T(\mathbf{X}_k, \mathbf{X}) = 2\mathbb{E}|\mathbf{X}_k - \mathbf{X}| - \mathbb{E}|\mathbf{X}_k - \mathbf{X}'_k| - \mathbb{E}|\mathbf{X} - \mathbf{X}'|$, $\mathbf{X}$ and $\mathbf{X}'$ are independent random variables from F, $\mathbf{X}_k$ and $\mathbf{X}'_k$ are independent random variables from F_k.

Applying the Proposition 2 of (Székely and Rizzo, 2013), we have

$$T(\mathbf{X}_k, \mathbf{X}) = \frac{1}{c_d} \int_{\mathbb{R}^d} \frac{|f_k(\boldsymbol{t}) - f(\boldsymbol{t})|^2}{|\boldsymbol{t}|^{1+d}} d\boldsymbol{t},$$

where f_k and f are the characteristic functions of $\mathbf{X}_k$ and $\mathbf{X}$, respectively, and c_d is the same constant used in Definition 6. This suggests that $\mathrm{gCov}(\mathbf{X}, Y) \geqslant 0$ with equality to zero if and only if $\mathbf{X}$ and $\mathbf{X}_k$ are identically distributed for all $k = 1, ..., K$, or equivalently, $\mathbf{X}$ and Y are independent.

Gini covariance can also be represented by Gini mean difference (GMD). The Gini

mean differences are defined as

$$\Delta = \mathbb{E}|\mathbf{X} - \mathbf{X}'|,$$

$$\Delta_k = \mathbb{E}|\mathbf{X}_k - \mathbf{X}'_k|,$$

$$\Delta_{kl} = \mathbb{E}|\mathbf{X}_k - \mathbf{X}_l|.$$

Then gCov can be rewritten as

$$\begin{aligned}
\mathrm{gCov}(\mathbf{X}, Y) &= \sum_{k=1}^{K} p_k T(\mathbf{X}_k, \mathbf{X}) \\
&= \sum_{k=1}^{K} p_k [2\mathbb{E}|\mathbf{X}_k - \mathbf{X}| - \mathbb{E}|\mathbf{X}_k - \mathbf{X}'_k| - \mathbb{E}|\mathbf{X} - \mathbf{X}'|] \\
&= 2\sum_{k=1}^{K}\sum_{l=1}^{K} p_k p_l \Delta_{kl} - \sum_{k=1}^{K} p_k \Delta_k - \Delta \\
&= 2\Delta - \sum_{k=1}^{K} p_k \Delta_k - \Delta \\
&= \Delta - \sum_{k=1}^{K} p_k \Delta_k
\end{aligned}$$

The *Gini correlation* is then defined as

Definition 11.

$$\mathrm{gCor}(\mathbf{X}, Y) = \frac{\mathrm{gCov}(\mathbf{X}, Y)}{\Delta} = \frac{\Delta - \sum_{k=1}^{K} p_k \Delta_k}{\Delta}.$$

Since $\sum_{k=1}^{K} p_k \Delta_k$ is the weighted average of GMD within each group, and $\Delta - \sum_{k=1}^{K} p_k \Delta_k$ is the GMD between groups, then gCor can be interpreted as the ratio of the between-group Gini variation and the total Gini variation. $\mathrm{gCor}(\mathbf{X}, Y)$ has the following properties:

1) $0 \leqslant \mathrm{gCor}(\mathbf{X}, Y) \leqslant 1$;

2) $\mathrm{gCor}(\mathbf{X}, Y) = 0$ if and only if $\mathbf{X}$ and Y are independent;

3) $\text{gCor}(\mathbf{X}, Y) = 1$ if and only if F_k is a single point mass distribution.

When $d = 1$, it was also shown in (Dang et al., 2018) that

$$\Delta = 2 \int F(x)(1 - F(x))dx,$$

and gCor can be written as

$$\begin{aligned}\text{gCor}(X, Y) &= \frac{2 \int F(x)(1 - F(x))dx - 2 \sum_{k=1}^{K} p_k \int_{\mathbb{R}} F_k(x)(1 - F_k(x))dx}{2 \int F(x)(1 - F(x))dx} \\ &= \frac{\sum_{k=1}^{K} p_k \int_{\mathbb{R}} (F_k(x) - F(x))^2 dx}{\int F(x)(1 - F(x))dx}.\end{aligned}$$

This representation shows that gCor measures the distance between the marginal distribution $F(x)$ and the conditional distribution $F_k(x)$.

Like what we have shown for dCov and dCor, the sample versions of gCov and gCor are needed for implementation purposes.

Definition 12. Let $(\boldsymbol{x}_i, y_i), i = 1, ..., n$, be n i.i.d. samples drawn from the joint distribution of $\mathbf{X}$ and Y, and $\mathcal{I}_k$ be the index set of sample points with $y_i = L_k$. Then p_k is estimated by the sample proportion of category L_k, i.e., $\hat{p}_k = \frac{n_k}{n}$, where $n_k = |\mathcal{I}_k| > 2$. The sample estimators of Gini distance covariance Gini correlation are defined by

$$\text{gCov}_n(\mathbf{X}, Y) = \hat{\Delta} - \sum_{k=1}^{K} \hat{p}_k \hat{\Delta}_k,$$

$$\text{gCor}_n(\mathbf{X}, Y) = \frac{\hat{\Delta} - \sum_{k=1}^{K} \hat{p}_k \hat{\Delta}_k}{\hat{\Delta}},$$

where $\hat{\Delta}_k = \binom{n_k}{2}^{-1} \sum_{i<j \in \mathcal{I}_k} |\boldsymbol{x}_i - \boldsymbol{x}_j|_d$, $\hat{\Delta} = \binom{n}{2}^{-1} \sum_{1=i<j=n} |\boldsymbol{x}_i - \boldsymbol{x}_j|_d$.

Similiar to distance statistics, $\text{gCov}_n(\mathbf{X}, Y)$ and $\text{gCor}_n(\mathbf{X}, Y)$ converges to $\text{gCov}(\mathbf{X}, Y)$ and $\text{gCor}(\mathbf{X}, Y)$ as $n \to \infty$. However, different from distance statistics, the sample estimations $\text{gCov}_n(\mathbf{X}, Y)$ and $\text{gCor}_n(\mathbf{X}, Y)$ are unbiased. In other words, negative values may be

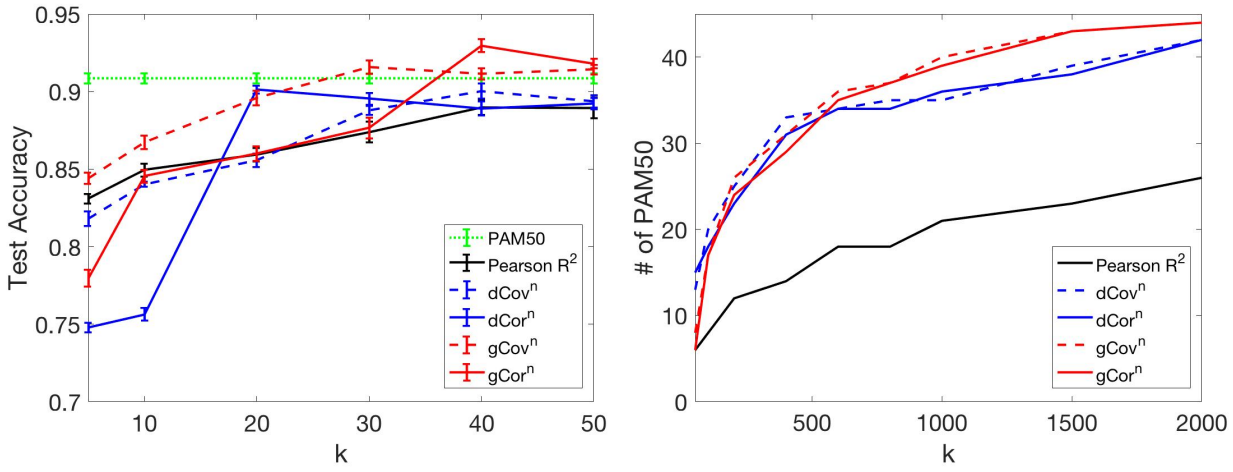
observed if $\mathbf{X}$ and Y are independent.

The complexity of a direct implementation of $\mathrm{gCov}_n(\mathbf{X}, Y)$ and $\mathrm{gCor}_n(\mathbf{X}, Y)$ according to Definition 12 is $O(n^2)$. It can be simplified to $O(n\log(n))$ when the dimension of $\mathbf{X}$ is 1, i.e., $d = 1$. This is because the Gini mean distance of a 1-dimensional random variable can be written as a linear combination of order statistics (Schezhtman and Yitzhaki, 1987). Assume the order statistics of $x_1, x_2, ..., x_n$ are $x_{(1)} \leqslant x_{(2)} \leqslant ... \leqslant x_{(n)}$, then

$$\hat{\Delta} = \binom{n}{2}^{-1} \sum_{1=i<j=n} |x_i - x_j| = \binom{n}{2}^{-1} \sum_{i=1}^{n} (2i - n - 1)x_{(i)}. \tag{2.1}$$

The calculation of (2.1) takes $O(n)$ and the sorting takes $O(n\log(n))$, thus the overall computation takes $O(n\log(n))$. This fast implementation is very useful when testing the dependence between each feature and the label, since the number of features can be very large. However, when testing the dependence between a set/subset of features (treated as a multidimensional random variable) and the label, only the $O(n^2)$ algorithm can be used. When n is large, using a random subset of samples for dependence estimations is recommended.

Since gGor measures the dependence between a numerical variable and a categorical variable, it is well suited to serve as a feature selector in a classification problem. One of our previous work (Zhang et al., 2019) is to use gCor or gCov to rank feature relevance and showed that gCor and gCov outperformed dCor, dCov and Pearson correlation. To apply Pearson correlation, we treated the class variable as numerical. To apply dCor or dCov, we used set difference to measure the sample distances of the class variable, i.e., the distance between to samples is 1 if they are of different classes, or 0 otherwise. Figure 2.1 shows the ranking performance of using Pearson R^2, dCov, dCor, gCov and gCor. PAM50 is the gold standard gene list for breast cancer subtype diagnosis. In both subfigures, k is the number of top features being selected. Figure 2.1a shows that gCov and gCor are able to select a smaller set of genes and the prediction is better than the gold standard. Figure 2.1b shows that Gini statistics are able to select more PAM50 genes than distance statistics as k



(a) (b)

Figure 2.1. Feature selection on the breast cancer dataset. (a) Test accuracy using the top k selected genes. (b) Number of PAM50 genes in the selected top k genes.

increases.

2.4 Permutation Tests

Like other statistical tests, a p-value needs to be calculated to measure the significance of dependence. The p-value is the probability of observing the test statistic or higher when the null hypothesis is true. A smaller p-value suggests the higher significance of the test. In our dependence test, if we use $\mathrm{gCor}_n(\mathbf{X}, Y)$ as the test statistic (similar if we use $\mathrm{dCor}_n(\mathbf{X}, Y)$), we will have the following null and alternative hypotheses:

$$H_0 \ : \ \mathrm{gCor}(\mathbf{X}, Y) = 0, (\mathbf{X} \text{ and Y are independent});$$
$$H_1 \ : \ \mathrm{gCor}(\mathbf{X}, Y) > 0, (\mathbf{X} \text{ and Y are dependent}).$$

A claim on dependence can be make by comparing the p-value with a pre-defined significance level α: if $p-\text{value} \leqslant \alpha$, we reject the null hypothesis and claim $\mathbf{X}$ and Y are dependent. For example, if we we observe $\mathrm{gCor}_n(\mathbf{X}, Y) = 0.01$ in our dependence test, a p-value $= 0.045$ is interpreted as: when $\mathbf{X}$ and Y are independent, $P(\mathrm{gCor}_n(\mathbf{X}, Y) \geqslant 0.01) = 0.045$. If the significance level α is set at 0.05, we can reject the null hypothesis and say $\mathbf{X}$ and Y are

dependent. The p-value is crucial for two reasons: 1) it can be used to determine whether or not two random variables are dependent; 2) the p-values of dependence tests can be used to sort the significance of dependence, e.g., in the application of feature selection.

A common approach to calculate p-value is to perform permutation tests. A permutation test is to calculated the test statistic ($\mathrm{gCor}_n(\mathbf{X}, Y)$ in the example above) after random permutating the sample observations of $\mathbf{X}$ or Y. After N permutation tests, the p-value can be estimated by the proportion of test statistics obtained in permutation tests that are greater than the observed test statistic in the dependence test, i.e.,

$$p-\text{value} = \frac{\sum_{i=1}^{N} I(\mathrm{gCor}_n(\mathbf{X}, Y)_{i^{th}perm} \geqslant \mathrm{gCor}_n(\mathbf{X}, Y))}{N}, \tag{2.2}$$

where $\mathrm{gCor}_n(\mathbf{X}, Y)_{i^{th}perm}$ is the notation for $\mathrm{gCor}_n(\mathbf{X}, Y)$ obtained in the i^{th} permutation test and $I(\cdot)$ is the indicator function. The larger the value of N is, the more accurate the p-value can be obtained. Therefore in our implementation, we used a relative large N ($N = 5000$) and parallelized permutation tests.

CHAPTER 3

FEATURE LOCALITY

3.1 Motivation

The idea of feature locality comes from the observation that convolutional neural networks (CNNs) are good at analyzing image data due to their ability to learn abstract representations from local features (pixels). In spite of their excellent performance, CNNs are black-box models that lack interpretability. This motivated us to borrow the idea of exploiting local information from CNNs to improve random forests, which are much more interpretable (discussed in Section 1.3).

Is local information of features useful to a random forest? The answer is yes, if the following hypothesis is true: features within a shorter distance are more dependent than features across a longer distance. For image data, the distance between features can be defined by the Euclidean distance between pixels on the 2D grid. For non-image data, the distance between features needs to be defined. We show some experimental results to validate our hypothesis below, but let's assume the hypothesis is true for now. We can therefore increase the variance of the tree outputs by constructing the forest in this way: we use a subset of features (instead of all features) as the input for each tree and let local features be in the same tree. As a consequence, features input to different trees are less correlated. Since a tree classifier is just a function mapping from input to output, by making the inputs less correlated, we are also forcing the outputs from different trees to be less correlated. Mathematically, let $\mathcal{A}$ and $\mathcal{B}$ be two subsets of features, and $\mathbf{X}_{\mathcal{A}}$, $\mathbf{X}_{\mathcal{B}}$ be the data represented using feature set $\mathcal{A}$ and $\mathcal{B}$, respectively. We then train a tree T_A using feature set $\mathcal{A}$ and another tree T_B using feature set $\mathcal{B}$ and let $\Theta_{\mathcal{A}}$ and $\Theta_{\mathcal{B}}$ be the parameters to

generate T_A and T_B, respectively. Then the class conditional correlation between these two trees is

$$
\begin{aligned}
&\rho(\Theta_{\mathcal{A}}, \Theta_{\mathcal{B}}|Y) \\
&= \rho_{\mathbf{X}}(rmg(\Theta_{\mathcal{A}}, \mathbf{X}, Y), rmg(\Theta_{\mathcal{B}}, \mathbf{X}, Y)) \\
&\propto \mathrm{Cov}_{\mathbf{X}}(rmg(\Theta_{\mathcal{A}}, \mathbf{X}, Y), rmg(\Theta_{\mathcal{B}}, \mathbf{X}, Y)) \\
&= \mathrm{Cov}_{\mathbf{X}}(rmg(\Theta_{\mathcal{A}}, \mathbf{X}_{\mathcal{A}}, Y), rmg(\Theta_{\mathcal{B}}, \mathbf{X}_{\mathcal{B}}, Y)) \\
&= \mathbb{E}_{\mathbf{X}}[rmg(\Theta_{\mathcal{A}}, \mathbf{X}_{\mathcal{A}}, Y) rmg(\Theta_{\mathcal{B}}, \mathbf{X}_{\mathcal{B}}, Y)] \\
&\quad - \mathbb{E}_{\mathbf{X}_{\mathcal{A}}}[rmg(\Theta_{\mathcal{A}}, \mathbf{X}_{\mathcal{A}}, Y)] \mathbb{E}_{\mathbf{X}_{\mathcal{B}}}[rmg(\Theta_{\mathcal{B}}, \mathbf{X}_{\mathcal{B}}, Y)] \\
&= \int rmg(\Theta_{\mathcal{A}}, \boldsymbol{x}_{\mathcal{A}}, Y) rmg(\Theta_{\mathcal{B}}, \boldsymbol{x}_{\mathcal{B}}, Y) p(\boldsymbol{x}_{\mathcal{A}}, \boldsymbol{x}_{\mathcal{B}}|Y) d\boldsymbol{x} \\
&\quad - \int rmg(\Theta_{\mathcal{A}}, \boldsymbol{x}_{\mathcal{A}}, Y) p(\boldsymbol{x}_{\mathcal{A}}|Y) d\boldsymbol{x}_{\mathcal{A}} \int rmg(\Theta_{\mathcal{B}}, \boldsymbol{x}_{\mathcal{B}}, Y) p(\boldsymbol{x}_{\mathcal{B}}|Y) d\boldsymbol{x}_{\mathcal{B}} \\
&= \int rmg(\Theta_{\mathcal{A}}, \boldsymbol{x}_{\mathcal{A}}, Y) rmg(\Theta_{\mathcal{B}}, \boldsymbol{x}_{\mathcal{B}}, Y) p(\boldsymbol{x}_{\mathcal{A}}, \boldsymbol{x}_{\mathcal{B}}|Y) d\boldsymbol{x} \\
&\quad - \int rmg(\Theta_{\mathcal{A}}, \boldsymbol{x}_{\mathcal{A}}, Y) rmg(\Theta_{\mathcal{B}}, \boldsymbol{x}_{\mathcal{B}}, Y) p(\boldsymbol{x}_{\mathcal{A}}|Y) p(\boldsymbol{x}_{\mathcal{B}}|Y) d\boldsymbol{x} \\
&= \int rmg(\Theta_{\mathcal{A}}, \boldsymbol{x}_{\mathcal{A}}, Y) rmg(\Theta_{\mathcal{B}}, \boldsymbol{x}_{\mathcal{B}}, Y) [p(\boldsymbol{x}_{\mathcal{A}}, \boldsymbol{x}_{\mathcal{B}}|Y) - p(\boldsymbol{x}_{\mathcal{A}}|Y) p(\boldsymbol{x}_{\mathcal{B}}|Y)] d\boldsymbol{x} \\
&\leqslant \{ \int [rmg(\Theta_{\mathcal{A}}, \boldsymbol{x}_{\mathcal{A}}, Y) rmg(\Theta_{\mathcal{B}}, \boldsymbol{x}_{\mathcal{B}}, Y)]^2 d\boldsymbol{x} \int [p(\boldsymbol{x}_{\mathcal{A}}, \boldsymbol{x}_{\mathcal{B}}|Y) - p(\boldsymbol{x}_{\mathcal{A}}|Y) p(\boldsymbol{x}_{\mathcal{B}}|Y)]^2 d\boldsymbol{x} \}^{1/2}.
\end{aligned}
\tag{3.1}
$$

Notice that the second integral in (3.1) is a dependence measure of $\mathbf{X}_{\mathcal{A}}$ and $\mathbf{X}_{\mathcal{B}}$. By making $\mathbf{X}_{\mathcal{A}}$ and $\mathbf{X}_{\mathcal{B}}$ less dependent, we can reduce the correlation between trees. In Chapter 1 we showed that the performance of random forest is controlled by the trade-off between strength and correlation. When the strength of individual trees is high enough, reducing the correlation is likely to improve the forest's performance.

3.2 Feature Dependence in Image Data

To validate our hypothesis empirically, we need a way to measure the dependence between features. Specifically, we are more interested in measuring dependence between two

groups of features, and the sizes of the two groups are not necessarily to be the same. The two groups of features can be viewed as two random vectors, possibly of different dimensions. An observation of the random vector is a sample represented by the features of that group. Therefore, measuring the dependence between the two groups of features is equivalent to measuring the dependence between the two random vectors. As introduced in Section 2.2, distance correlation (dCor) is a good dependence measure for random vectors of arbitrary and not necessarily equal dimensions. It is zero if and only if the two random vectors are independent.

We tested our hypothesis on one of the most popular image data sets: MNIST (LeCun et al., 2010). In our experiment, we randomly selected two m by m windows from the image as the two feature sets and tested the distance correlation of the two sets. 500 random samples (images) were used to calculate distance correlation. For implementation simplicity, the two windows under dependence test were of the same size, but they were not required to be. The window distance is measured by the Euclidean distance between the two left corner pixels of the two windows on the image and rounded to an integer. Given two pixels p_1 and p_2 with their coordinates on the image being (x_1, y_1) and (x_2, y_2), the distance between p_1 and p_2 is defined as

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}. \tag{3.2}$$

A schematic diagram of our experiment is shown in Figure 3.1. The conditional distance correlation $\mathrm{dCor}(\mathbf{X}_1, \mathbf{X}_2|Y)$ was first calculated and the unconditional distance correlation is calculated by taking the expectation of the $\mathrm{dCor}(\mathbf{X}_1, \mathbf{X}_2|Y)$, i.e., $\mathrm{dCor}(\mathbf{X}_1, \mathbf{X}_2) = \mathbb{E}_Y[\mathrm{dCor}(\mathbf{X}_1, \mathbf{X}_2|Y)]$. Here $\mathbf{X}_1$ and $\mathbf{X}_2$ denotes the two sets of features defined by the two windows and Y denotes the class label. The results are shown in Figure 3.2. Three different values of window size m were used: 10, 15 and 20. The range of the window distance is defined by the window size since the images are of a fixed size of 28 by 28. The distance was rounded to an integer for plotting and each point in the figure is an average of 100 experiments. The figure shows a clear trend of decreasing dependence when the distance between

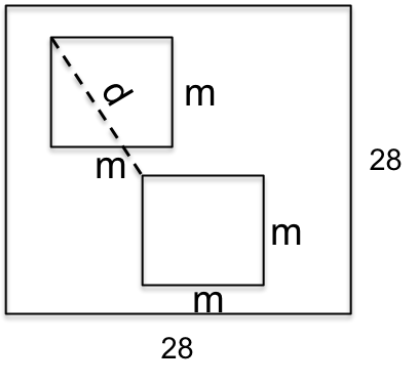


Figure 3.1. A schematic diagram of testing dependence between two windows of size $m \times m$ on MNIST data. The distance between two widows d is measured by the distance between the upper left corners of the windows.

the two windows is increased, which validates our hypothesis empirically. For window size $m = 10$ with $d > 15$, the slightly increase in dCor is because both windows include some margins of the image (black background regions).

3.3 Local Features of Non-Image Data

For non-image data, feature locality is not explicitly given as in the case of images, therefore it has to be defined or learned from the data. To learn the neighborhood information of the features, the pair-wise distance needs to be defined. The goal of defining the distance is to capture feature dependence such that dependent features should have a closer distance to each other. Therefore it is appealing to use a dependence measure to represent the distance. As discussed in Section 2.2, dCor is a good candidate for this purpose, but the computation is too expensive for pair-wise feature dependence measures, which is $(O(n^2d^2))$, where d is the number of features). To reduce computation complexity, we use Pearson correlation as a substitute. Other distance metrics that have been used in data visualization and dimensionality reduction techniques are also worth testing, since they are good approximations for sample dependence. Here we propose four ways of defining the pair-wise distance between features. Each feature can be represented by its values in the samples, i.e., a vector of length n, where n is the number of samples. The following distance

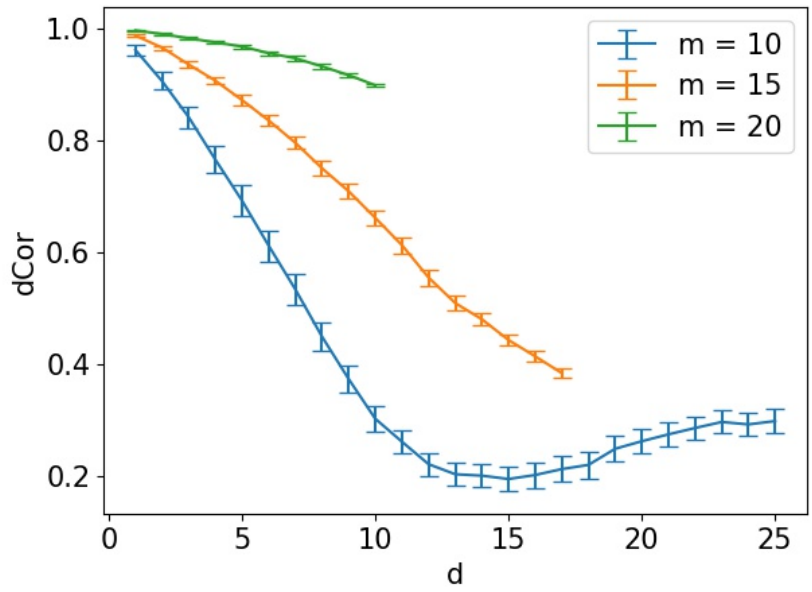


Figure 3.2. Feature dependence tested using distance correlation statistic on MNIST dataset.

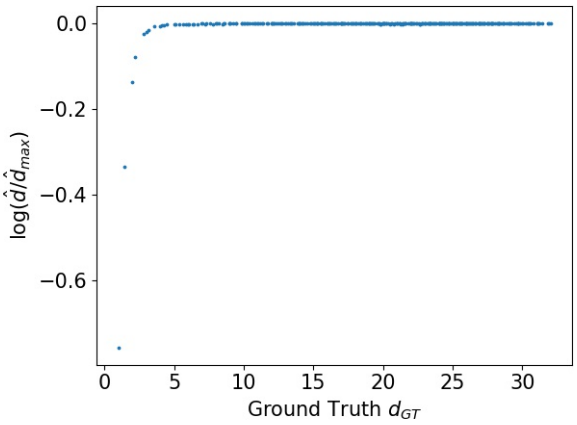
metrics are used in our study to calculate pair-wise distance of features: 1) $1 - \rho^2$, where ρ is the Pearson correlation; 2) Euclidean distance; 3) PCA Euclidean and 4) graph distance. The first one is a direct approach to capture feature dependence. Euclidean distance is the most commonly used distance metric in data analysis. PCA Euclidean is defined as the Euclidean distance between features after dimensionality reduction using PCA (Principle Component Analysis). This is the default distance used in t-SNE implementation in Python scikit-learn. The graph distance is defined as the shortest path between two features in the k-neighbor graph. In the k-neighbor graph, each node is a feature and it has undirected edges to its k-nearest neighbors defined by Euclidean distance and the weight of the edges are the Euclidean distances. This distance is used in the Isomap algorithm for dimensionality reduction (Tenenbaum et al., 2000) and is good for the Swiss Roll dataset. However, the choice of k is tricky and the graph can also be defined as a unweighted graph, which makes it difficult to find the optimal and stable distance representation for different datasets. We show the evaluation of the above mentioned distance metrics in Chapter 4. Once the pair-wise distance of features is defined, a random set of "local" features of size m can be

selected by randomly choosing a feature and its $(m-1)$ nearest neighbors.

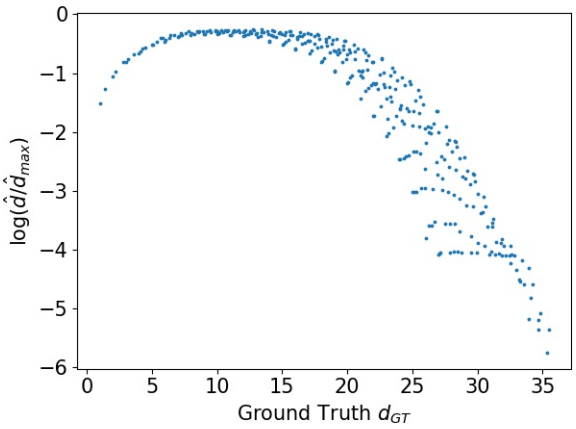
To provide some insight on how well these distances are defined, we compute these distances, denoted as $\hat{d}$, using the MNIST data and compare those with the ground truth, d_{GT}, which can be obtained by (3.2). For each distance type we defined, a scatter plot is generated and shown in Figure 3.3. The x-axis is the ground truth, the y-axis is the logarithmic of the normalized distance learned from the data. Since a particular value of d_{GT} may correspond to multiple values of $\hat{d}$, the median of which is used to generate the plot. A perfect distance measure should have a monotonic increasing trend as d_{GT} increases. As shown in Figure 3.3, none of the distance defined maintains monotonic increasing trend in the whole x-range. All of them serve as a good distance measure only within a certain range. Pearson distance is functional only when d_{GT} is less than 5. $\log(\hat{d}/\hat{d}_{max})$ is close to 0 when $d_{GT} > 5$. This is because a pair of pixels with a distance longer than 5 in between on the images has a Pearson correlation close to 0, resulting $\hat{d}$ to be near 1. Euclidean and PCA Euclidean are functional when $d_{GT} < 7$. Graph distances have a longer functional range, up to $d_{GT} = 12$. The behavior is not sensitive to the different choices of k. From the above results, Graph distances seem like better distance measures when the ground truth is not available.

3.4 Summary

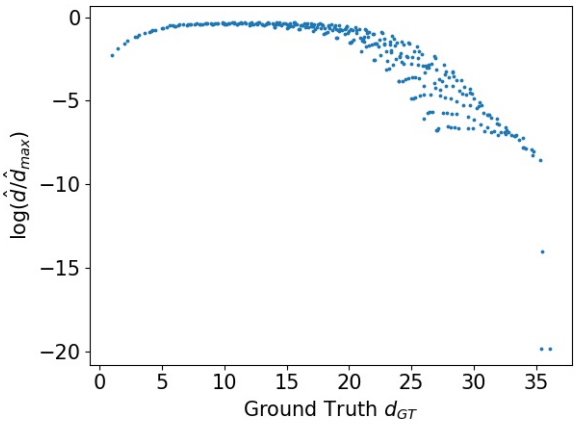
In this Chapter, we presented the concept of feature locality. For image data, the locations of features are given by their coordinates on the image. We validated that pixels within a shorter distance have higher dependency. For non-image data, we aim to find a organization of the features such that dependent features are clustered together. To achieve this goal, since the direct measure of dependence between every pair of features by distance correlation is too computationally expensive, we presented alternative ways of defining distance that captures dependence. The proposed distance measures were tested on the MNIST dataset and compared with the ground truth (distance defined by the pixel coordinates). The



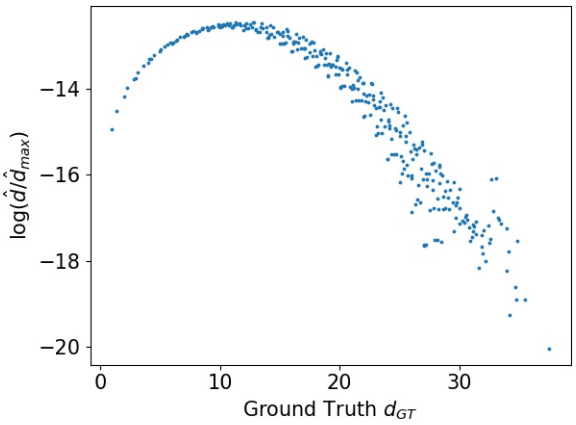
(a) Pearson $(1 - \rho^2)$



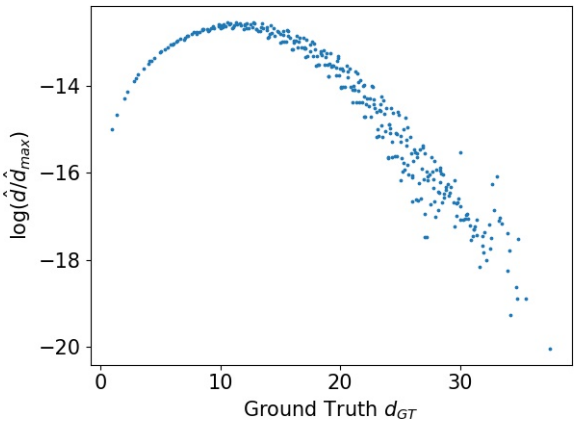
(b) Euclidean



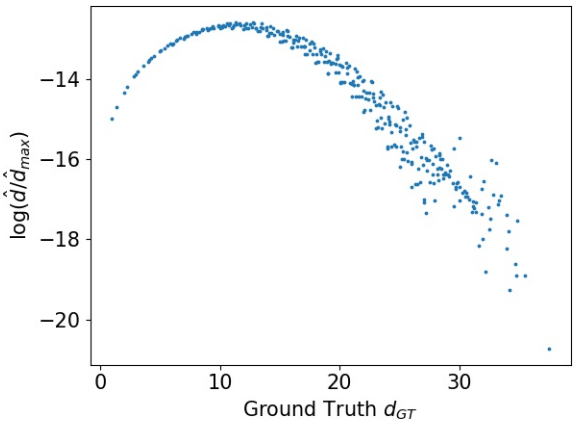
(c) PCA Euclidean



(d) Graph $(k = 5)$



(e) Graph $(k = 8)$



(f) Graph $(k = 10)$

Figure 3.3. The learned distances compared with the ground truth on the MNIST dataset.

graph distances show more promising results than others. In the next chapter, we show how to utilize feature locality to improve random forests.

CHAPTER 4

RANDOM FOREST WITH LOCAL FEATURE SAMPLING

4.1 Local Feature Sampling

The most popular way of building a random forest is by training each tree with bootstrap samples using all features. It has also been proposed to use a random subset of features to grow each tree, know as "the random subspace" (Barandiaran, 1998). However, the features in the subset are chosen by a pure random sampling without considering any locality information of the features. Here we propose a *local feature sampling* (LFS) approach, which is also using a feature subspace but our contribution is to include the locality information during the sampling process, i.e., a random patch or neighbourhood of features are used to construct each tree. In (Louppe, 2014), the term "random patch" is also used but what it actually means is random subspace. We name our version of random forest as RF-LFS (random forest with local feature sampling).

For image data, a random patch is defined by an $m \times m$ square region of pixels (total m^2 features) located at a random position in the image. The sampling process can be achieved by randomly picking a pixel from the image as the upper left corner of the patch and including all features in the $m \times m$ square. Notice that the valid region for picking the upper left corner pixel is not the whole image but a cropped image of size $(M-m)\times(M-m)$ if the whole image is of size $M \times M$. An example of how RF-LFS works on predicting a digit '8' is shown in Figure 4.1. The forest consists of 6 trees. Each tree has access to a random local region of the image during training and testing. This introduces varieties in the tree outputs since different trees have different access to the data. For example, tree 2 predicts '9' because it looks at the top part of digit '8' while tree 6 predicts '6' since it looks at the

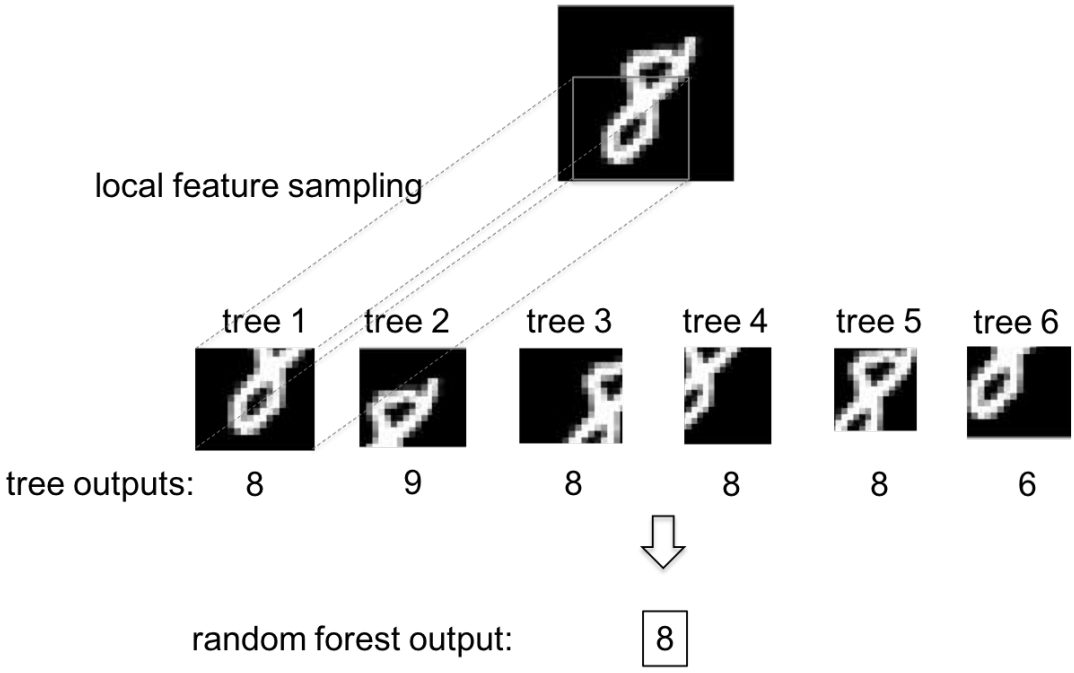


Figure 4.1. An example of how RF-LFS works on predicting a digit '8'.

bottom part. However, the majority votes for '8' which is correct.

For non-image data, a random neighbourhood of size m is generated by randomly selecting one feature from the whole feature set and then its $(m-1)$ nearest neighbours with the distance defined using one of the distance metrics described in Section 3.3. Therefore, a pair-wise distance metric of features needs to be calculated before the construction of trees.

4.2 Sampling Window Size

We call the size of a random patch or neighbourhood as the sampling window size. When the sampling window size is equal to the total number of features, then LFS is equivalent to not using LFS. Therefore, the default setting of current random forest can be viewed as a special case of our RF-LFS. The window size among trees can be kept the same or using a random size for each tree. In the former case, the window size is a hyper-parameter and in the latter case, the range of the window size is a hyper-parameter. We compare these two cases in Section 4.4.1.

4.3 Out-of-Bag Estimates of Error, Strength and Correlation

We implemented out-of-bag estimates of error, strength and correlation in the RF-LFS classifier, as described in Section 1.5. The estimates of strength and correlation help us in understanding how LFS affects the model and the out-of-bag error can be used as the validation performance to optimize hyper-parameters. Notice that the out-of-bag estimates require bootstrap. Since bootstrap is usually recommended to train random forests, we use bootstrap for all experiments.

4.4 Experimental Results

4.4.1 The MNIST Dataset

We first tested RF-LFS on the most popular benchmark image dataset, the MNIST dataset. We compare RF-LFS with 1) fixed and 2) random window size against the other three non-LFS based RFs as baselines: 3) RF using all features (this is the Python scikit-learn implementation); 4) RF using random subspaces with fixed size and 5) RF using random subspaces with random sizes. We use the following legends for the five methods under comparison: 1) LFS-fixed; 2) LFS-random; 3) all; 4) subspace-fixed and 5) subspace-random.

Figure 4.2 shows the results of test accuracy, RF out-of-bag accuracy (performance of RF on out-of-bag examples), tree out-of-bag accuracy (average performance of trees on out-of-bag examples), strength (out-of-bag estimates), correlation (out-of-bag estimates) and upper bound of generalization error (Theorem 1). The number of features sampled to grow each tree is a hyperparameter for subspace-fixed and LFS-fixed, but not for the other three methods. Therefore, we show the effect of this hyperparameter on the performance by using different values from 100 to 700. Each data point in the curve is the average of 30 runs. For method all, subspace-random, and LFS-random, a straight line is plotted. Among all five methods under comparison, LFS-random has the best performance (test accuracy), regardless of the hyperparameter used for other methods.

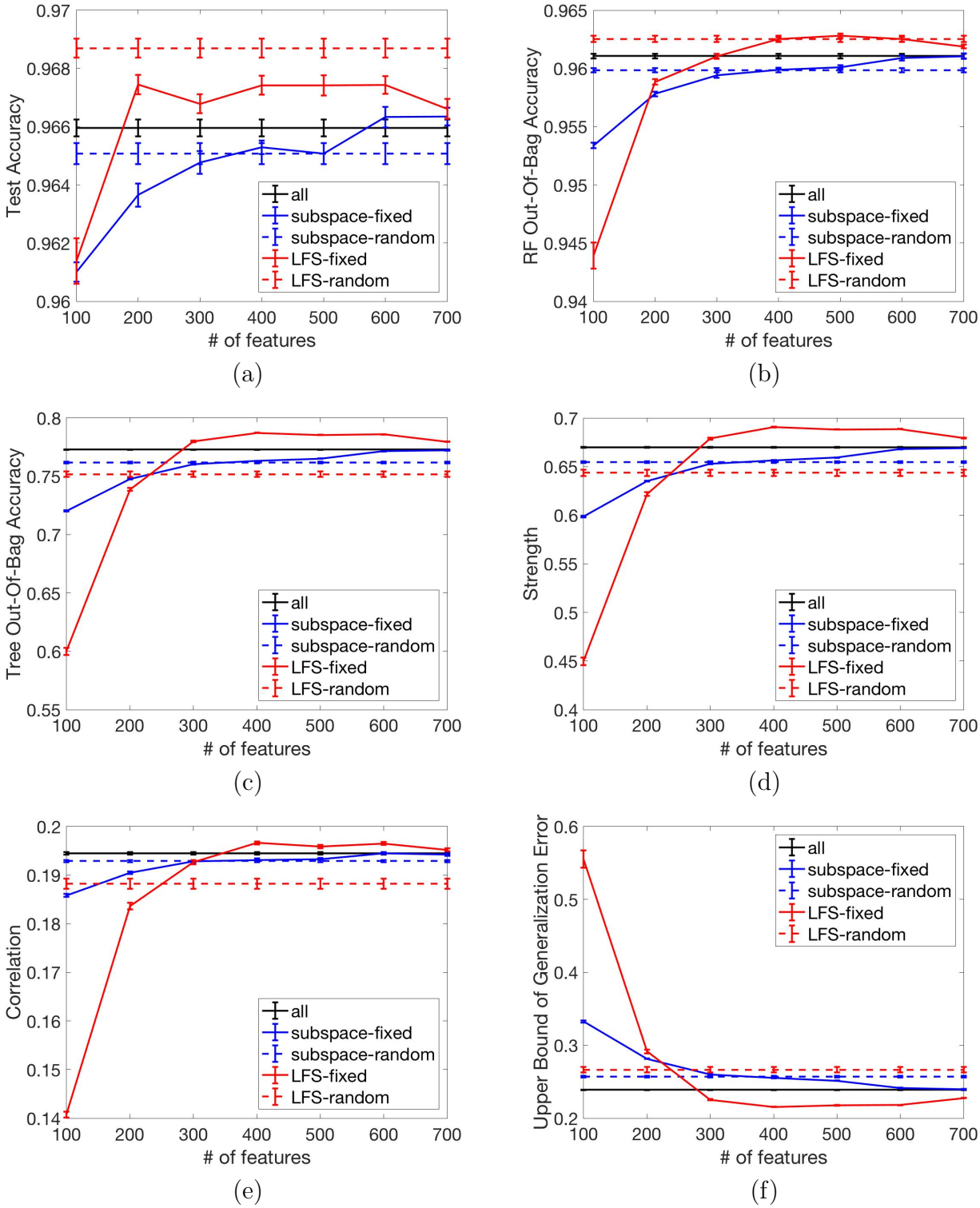


Figure 4.2. The MNIST dataset. Effect of number of features sampled to grow each tree.

Table 4.1. Performance on the MNSIST Dataset Using Optimal Number of Features for Sampling.

Method	Validation (%)	Test (%)	Strength	Correlation
all	96.11	96.60	0.670	0.195
subspace-fixed	96.09	96.63	0.668	0.195
subspace-random	95.98	96.51	0.655	0.193
LFS-fixed	**96.25**	96.74	**0.691**	0.197
LFS-random	**96.25**	**96.87**	0.644	**0.188**

We use RF out-of-bag accuracy as validation accuracy to choose the optimal number of features used for sampling and compare test accuracy. The results are shown in Table 4.1. The performance gain of LFS-random is due to the reduced correlation with sacrifice in strength. The decrease in strength is because of the use of local features, as expected. However, since now the correlation is more dominant in controlling the overall forest performance, reducing correlation among trees is beneficial. The fact subspace-fixed and LFS-random outperforms subspace-random shows that the performance gain comes from the local features sampling, rather than a simply usage of random subsets of features.

From Figure 4.2(f), we can see that the upper bound is loose, therefore does not correspond with test accuracy. This suggests that a lower upper bound on generalization error does not guarantee a higher test accuracy. Therefore it is more reasonable to use RF out-of-bag accuracy to optimize hyperparamters. Figure 4.2(d)&(c) also show the trade-off between strength and correlation, i.e., one cannot increase strength as well as reduce correlation at the same time. Since LFS-random outperforms LSF-fixed and subspace-random is comparable with subspace-fixed, we omitted methods with fixed sampling size for later experiments and no optimization on sampling size is further needed.

Notice that using LFS is not the only way to control the trade-off between strength and correlation, other hyperparameters like bootstrap and number of features used at each split (the *max_features* parameter of sklearn.ensemble.RandomForestClassifier) are also key hyperparameters that achieve this goal. The results shown in Figure 4.2 were obtained by setting bootstrap as true and number of features at each split as $log_2(n\_features)$. Our

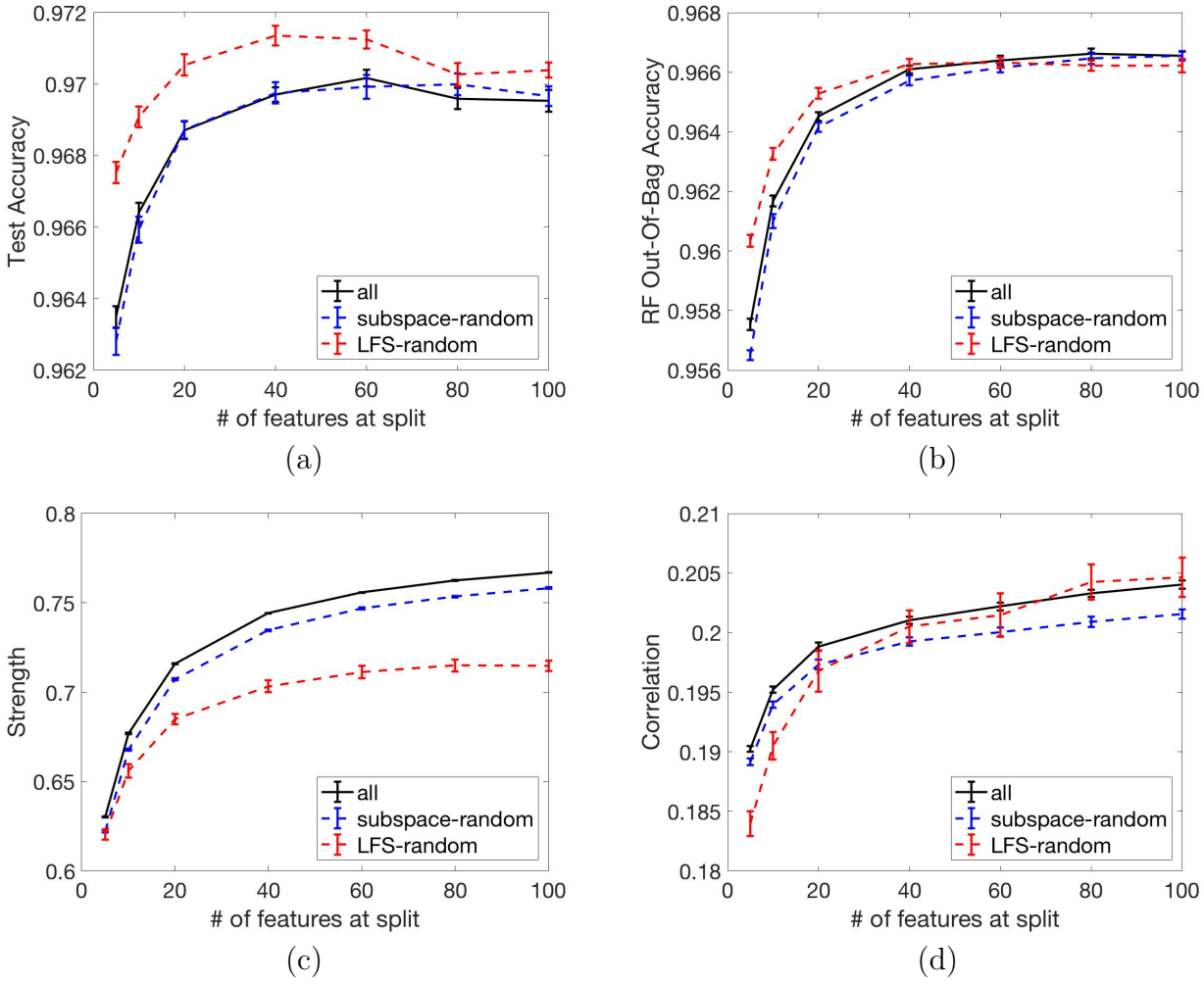


Figure 4.3. The MNIST dataset. Effect of number of features at each tree node to decide best split.

LFS is a third hyperparameter that should be tested along with these two to see if a third one is helpful. We keep bootstrap on for all tests and we compare LFS-random against all and subspace-random by varying the value for number of samples at split. The effect of the number of features at each split on the performance is shown in Figure 4.3. The performance gain due to LFS is small but significant. The validation and test accuracies, strength and correlation using the optimal number of features at each split are shown in Table 4.2.

We randomly selected 100 images from the test set that RF-all predicted wrong but RF-LFS-random predicted correctly, as shown in Figure 4.4. These are examples showing local information is more critical and generalizable than global information. For example,

Table 4.2. Performance on the MNIST Dataset Using Optimal Number of Features at Each Split.

Method	Validation (%)	Test (%)	Strength	Correlation
all	**96.66**	96.96	**0.763**	0.203
subspace-random	**96.66**	96.97	0.758	0.202
LFS-random	96.63	**97.14**	0.703	**0.201**

the image on the second row, second column is a "3", but it is so thick that RF-all predicted it as an "8". The image on the 5th row, 5th column is a "9", but RF-all interpreted it as a "0" because there is a big circle. However, by using local features, LFS was able to predict these images correctly.

We are also interested in what regions (groups of features) are essential to make the prediction correct for each class. Therefore, for each class, we selected the 10 trees in the forest that have highest accuracy for this particular class, and visualize the patches they used. The visualization result is shown in Figure 4.5. The white pixels are features selected and the intensity shows the frequency of a pixel being selected. It shows that trees have different focuses for different classes.

4.4.2 The ISOLET Dataset

We next test LFS on a non-image dataset—the ISOLET dataset, available at UCI Machine Learning Repository (Dheeru and Karra Taniskidou, 2017). This dataset contains recordings of human speaking letters from "a" to "z". The number of classes is 26, the number of examples is 7797 and the number of features is 617. We randomly hold out 50% of the data as test set and the other 50% as training set.

Since this is a non-image dataset, we need to choose a distance metric to define feature neighborhood. As mentioned in Section 3.3, we will test Pearson, Euclidean, PCA Euclidean and graph distance. Specifically, for the graph distance, the value of k needs to be determined to construct the k-neighbor graph. Here we tested the value of k to be 5, 8 and 10. The legend for methods under comparison are:

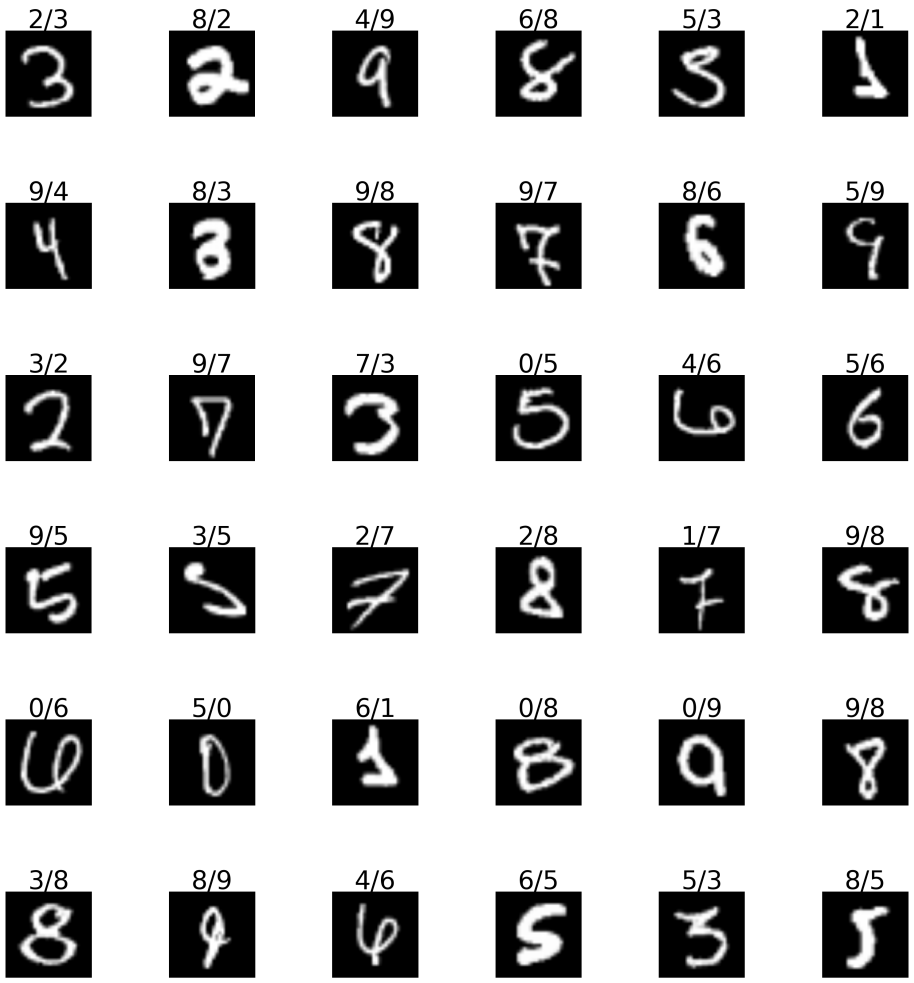


Figure 4.4. Some example images that RF-all predicted wrong but RF-LFS-random predicts correctly. The prediction result is shown in the format of "all/LFS" on the top of each image.

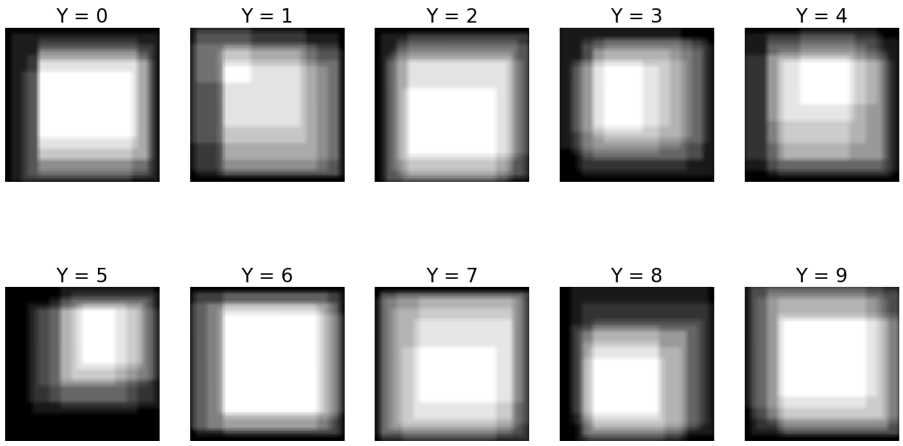
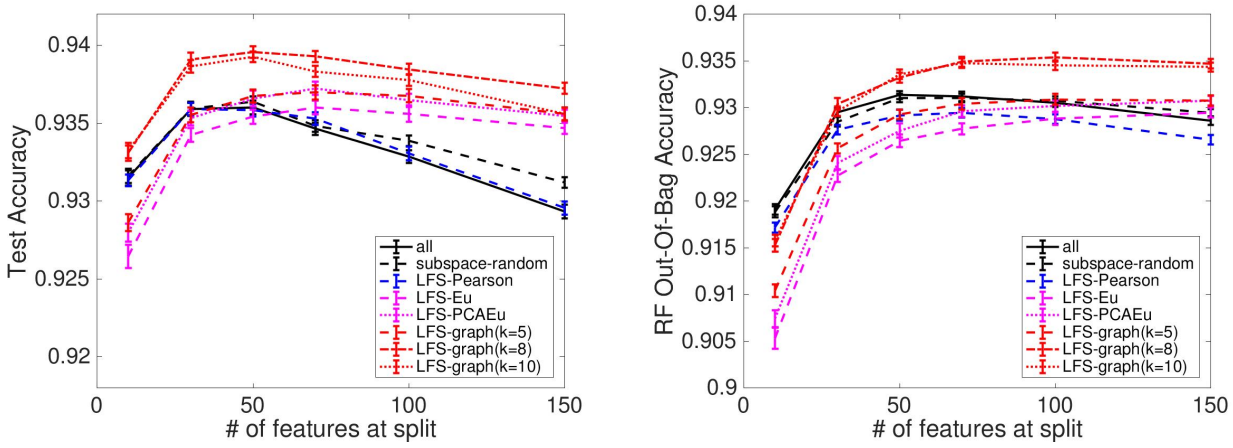


Figure 4.5. Random patches used by top-10 trees with highest accuracy for each class.

- all: RF using all features;
- subspace-random: RF using a random subset (of random size) of features without considering feature locality;
- LFS-Pearson: RF with LFS with random window size using $(1 - \rho^2)$ as the distance to define feature neighborhood;
- LFS-Eu: RF with LFS with random window size using Euclidean distance to define feature neighborhood;
- LFS-PCAEu: RF with LFS with random window size using Euclidean distance of features after dimensionality reduction using PCA. The number of components used for this dataset is 20;
- LFS-graph ($k = n$): RF with LFS with random window size using shortest path on the k-neighbor graph as the distance, where $k = n$.

The performance of methods under evaluation are shown in Figure 4.6. Each data point is an average of 100 runs. It is obvious that LFS-graph ($k = 8$) has the best performance in



(a) (b)

Figure 4.6. The ISOLET dataset. Effect of number of features at each tree node to decide best split.

Table 4.3. Performance on ISOLET Dataset Using Optimal Number of Features at Each Split.

Method	Validation (%)	Test (%)	Strength	Correlation
all	93.14	93.60	**0.570**	0.210
subspace-random	93.10	93.64	0.565	0.208
LFS-Pearson	92.92	93.58	0.534	0.188
LFS-Eu	92.88	93.56	0.486	0.175
LFS-PCAEu	93.08	93.55	0.501	0.181
LFS-graph ($k = 5$)	93.09	93.68	0.490	0.171
LFS-graph ($k = 8$)	**93.49**	**93.93**	0.487	**0.165**
LFS-graph ($k = 10$)	93.47	93.83	0.501	0.170

all x-range. The performances using optimal number of features at each split are shown in Table 4.3. As the table shows, all LFS methods based on graph distance outperform the sklearn implementation and subspace-random. It is also clear that the performance gain of these graph distance based LFS methods come from the reduced correlation. Among all distance metrics being tested, graph distance with $k = 8$ perform best and has the lowest correlation on this dataset. However, the optimal k can be data dependent. We chose to use graph distance with k set to 8 by default, as our distance metric for LFS to test on other non-image data.

4.4.3 More Datasets

From previous experimental results, we see LFS that can improve random forest. The sampling of features is done by randomly selecting patches of random size for image data or neighborhoods of random size defined by graph distance ($k = 8$) for non-image data. We from now on use a single notation LFS for our approach and simplify subspace-random as subspace. We tested our LFS on more datasets summarized in Table 4.4. HAR is the "Human Activity Recognition Using Smartphones Data Set" available on UCI (Anguita et al., 2013). It is a classification task of identifying six human activities (walking, walking_upstairs, waling_downstairs, sitting, standing, laying) using signals recorded by smartphones. The features are time and frequency domain variables calculated from sliding windows with 50% overlap, thus we would expect high dependency between these features. The UJIndoorLoc dataset is a classification of indoor locations (floors in buildings) using signal strength at Wireless Access Points (WAPs) (Torres-Sospedra et al., 2014). There is clearly a 3-D structure in features, but the information is not provided. Therefore we have to learn the locality information of the WAPs. Because this dataset is sufficient large and relatively easy, we use only 10% as training and 90% as testing.

The number of trees used for all datasets and methods is 100. The result for MNIST is the average of 30 runs (because of the large size of the dataset) and 100 runs for the other three datasets. We kept the training and test set unchanged for MNIST in order to compare with CNN. We shuffled other datasets to ensure similar training and testing distributions and used a larger hold-out to test method generalizability. The classification performances are shown in Table 4.5. We see consistent improvement using LFS across these datasets except the UJIndoorLoc dataset, where subspace performs best. This may due to the less dependency among features, i.e., the WAPs are sparsely scattered in the buildings, even though there is a 3D structure. In such scenario, using local features scarifies too much in strength, therefore is not beneficial for overall performance. For the other three datasets, we observed that subspace has about the same or even lower performance than using all

Table 4.4. Data Sets Summary

Data Set	Train Size	Test Size	Features	Classes	Shuffled
MNIST	60000	10000	784	10	No
ISOLET	3898	3899	617	26	Yes
HAR	5149	5150	561	6	Yes
UJIndoorLoc	2104	18944	520	13	Yes

Table 4.5. Performance on Multiple Datasets Using Optimal Number of Features at Each Split.

	Method	Datasets			
		MNIST	ISOLET	HAR	UJIndoorLoc
Test (%)	all	96.99	93.56	97.03	97.44
	subspace	96.98	93.61	97.10	**97.51**
	LFS	**97.10**	**93.87**	**97.22**	97.48
Validation (%)	all	**96.66**	93.09	97.22	**97.40**
	subspace	96.65	93.06	97.29	97.36
	LFS	96.61	**93.58**	**97.43**	96.90
Strength	all	**0.763**	**0.570**	**0.763**	0.672
	subspace	**0.763**	0.565	0.759	**0.681**
	LFS	0.709	0.502	0.735	0.645
Correlation	all	0.203	0.211	0.155	0.161
	subspace	0.203	0.208	0.154	0.164
	LFS	**0.201**	**0.172**	**0.137**	**0.159**

features, which suggests the importance of using local information. LFS reduced strength for all datasets because each tree only has access to local features. The correlation was reduced by LFS for all datasets compared to subspace and all, suggesting the effectiveness of using local information.

4.4.4 Discussions

We have some observations during experiments:

1) The improvement of using LFS is about less than 1%. Therefore, the performance gain is only observable on large datasets (about more than 1000 for the test set).

2) The dataset needs to have a reasonable number of features (hundreds) so that using LFS is beneficial. This is because when the number of features is small, there is less

redundancy and less dependency between features. Thus, using LFS is unlikely to reduce correlation but likely to reduce strength, therefore will not improve overall performance. On the other hand, when the number of features is too large, the computation of the distance matrix is too costly to be practical. In such scenarios, feature selection is desired before applying LFS. When the data dimension is large, the training data should also be sufficient large so that the strength is not too low to apply LFS. As a consequence, LFS works well on large data, in terms of both the sample size and dimension.

3) When the training set is sufficient large, the advantage of using LFS is more obvious when the hold-out percentage for the test set is large ($\geqslant 50\%$). This suggests that LFS can learn more abstract information from data and generalize better. This is understandable since the data dimension is reduced for each tree and local information can be learned efficiently.

4) LFS is more useful for multi-class than binary class classification tasks. This is because data with more classes tends to have more local structures that LFS takes advantage of. For example, if we only use digit "1" and "7" of the MNIST data and make it a binary classification problem, we would expect only the upper left pixels of the image to be useful for this task, i.e., fewer local structures are presented.

5) The minimum window size used for sampling is a hyper-parameter. When the features are highly dependent, a smaller window size can be used without losing too much strength while reducing correlation. However, if less dependency exists, a larger window size must be used to preserve enough strength and LFS will degrade to the sklearn implementation. In our experiments, the minimum window size was set to 100 for the MNIST dataset, 200 for the ISOLET and HAR datasets, and 300 for the UJIndoorLoc dataset.

6) The validation accuracy of LFS is not always higher than the baselines. This is because we used out-of-bag accuracy as validation. Since we shuffled the data before splitting training and testing sets, the data distributions of the two should be very similar. Note that each out-of-bag example is predicted by about 1/3 (the probability of being out-

of-bag sample during sampling with replacement) of the trees in the forest, the discrepancy between validation and test accuracy is due to the number of trees used as the ensemble. When LFS is applied, each tree is using a local structure, therefore we would expect to benefit more from having more trees than in the case LFS is not applied. A recommendation when applying LFS is to use sufficient number of trees (100 in our experiments).

The above mentioned observations can be used as a guideline regarding when to use LFS. We next present some potential usage of LFS.

LFS may have potential application on time-series data. Here time-series refers to data with each feature a record at a particular time point, not the response variable being time-series. The features of this type of data has a well structure given: they are organized in 1-D and features within a short time frame are highly dependent. We implemented LFS-randomTF that randomly sample features within a time frame to train each tree. However, we did not observe improvement in performance when applying LFS-randomTF on time-series datasets available on UCI. This is mainly because random forests or tree classifiers do not perform well on time-series data since the dimension is usually much higher than the sample size. In this case, the strength is the dominate term in determining the generalization error and one should aim to improve strength rather than reduce correlation in order to achieve a better prediction performance. However, we would expect to benefit from using LFS on time-series data when a sufficient large training set is given.

LFS can also be apply to other ensemble learners, e.g., kNN ensembles. Unlike decision trees, the training process of a kNN classifier has no randomness. Therefore, we can only introduce randomness in the process of constructing the ensemble. Bootstrap is not very effective for kNN ensemble, because the performance of kNN is insensitive to a small perturbation in the training samples. (Zhou and Yu, 2005) proposed to use random subspace on kNN ensemble and observed improved performance. We tested LFS on a kNN ensemble and compare it with the random subspace approach, using the same datasets. Because kNN is too slow when the training size is large, for the MNIST dataset, we randomly selected

Table 4.6. Performance of kNN ensemble on Multiple Datasets.

	Method	Datasets			
		MNIST	ISOLET	HAR	UJIndoorLoc
Test (%)	subspace	93.78	89.13	96.40	**92.03**
	LFS	**94.17**	**90.20**	**96.63**	91.45
Validation (%)	subspace	92.71	87.69	**96.17**	**90.97**
	LFS	**93.09**	**88.44**	95.95	89.77
Strength	subspace	**0.804**	**0.684**	**0.869**	**0.743**
	LFS	0.736	0.594	0.777	0.664
Correlation	subspace	0.308	0.414	0.147	0.402
	LFS	**0.280**	**0.317**	**0.144**	**0.350**

5000 training samples for training, with the test set unchanged. Other datasets are the same with previous experiments. We also set the number of base learners to 30 and the number of runs to 10 for the same reason. The results are shown in Table 4.6 and they are mostly consistent with those in Table 4.5.

LFS encourages us to analysis features as groups. As in the example shown in Figure 4.1, trees that have access to the middle part of the image are able to predict digit "8" correctly. In the contrast, trees that have only access to the top region or bottom region fails to generate the correct output. This suggests that some regions are more critical in prediction than others for a particular class. Unlike traditional random forest that only provides feature importance "individual wise", LFS provides some insights on class conditional feature importance "group wise", as shown in Figure 4.5. LFS can be used to cluster local feature sets into meaningful groups, like "center", "bottom", "top" in the case of MNIST data, and for non-image data as well, to help interpreting prediction results.

Learning the local information of features is a new direction of representation learning. Usually a new representation means change in data dimension and/or changes in the values of the features. However, in our LFS, we change neither of these. It is the organization of features that we are interested in. We aimed to learn a better organization of the features and enabled RF to benefit from it.

In spite of the significance on the performance gain in random forest by LFS, it is still not comparable to that of CNN, which is above 99% (LeCun et al., 1998). In the next two chapters, we propose another approach to improve random forest. However, due to the difference in model capability between random forest and CNN, we do not expect random forest to outperform CNN when training data are sufficient.

4.5 Related Work

Extensive efforts have been made to introduce randomness in random forest for lower correlation. The most significant one is bootstrap aggregating (Breiman, 1996), also called bagging, which uses bootstrap samples of the training data to grow each tree. A bootstrap sample is a random set drawn from the training set with replacement. In bagging, the diversity in trees comes from the difference in their bootstrap samples. Bagging can also be combined with boosting, an algorithm that maintains a set of weights for training examples and assigns higher weights to misclassified examples and lower weight to correctly classified examples. Then the weights from boosting can be used as the sampling distribution for bagging to generate bootstrap samples (Quinlan, 1993).

An alternative approach to increase diversity is to introduce randomness inside of each tree at the process of decision making. (Dietterich, 2000) proposed to use random split selection where the split at each node is selected at random from the K best splits. In this approach, every feature must be evaluated to determine the best K splits. To reduce complexity, (Amit and Geman, 1997) used a random selection of the features to decide the best split on image classification and feature selection tasks. (Breiman, 2001) generalized this idea and tested it on a variety of data sets.

The method that is most similar to ours is the "random subspace" approach (Barandiaran, 1998), where each tree only has access to a subset of features. Unlike ours, this subset of features is drawn randomly and uniformly from all features, without considering the dependence among these features. Our LFS approach aims to put dependent features in

the same tree and less dependent features into different trees.

4.6 Summary

In this chapter, we proposed a local feature sampling (LFS) approach to improve random forests. LFS uses a random subset of features that are close to/dependent on each other, to train each tree. Since dependent features are within one tree, feature sets for different trees are less dependent, thus reducing correlation among trees. Different distance measure to define feature locality were evaluated and the graph distance showed best performance, agreeing with the results in Chapter 3. On multiple UCI datasets where features are structured and highly correlated, our LFS approach outperformed random forest using all features or random subsets of feature to train each tree, suggesting the importance of learning local information of features. This chapter completes our work on reducing correlation to improve random forest. In the next chapters, we show our approaches to improve the strength of random forest.

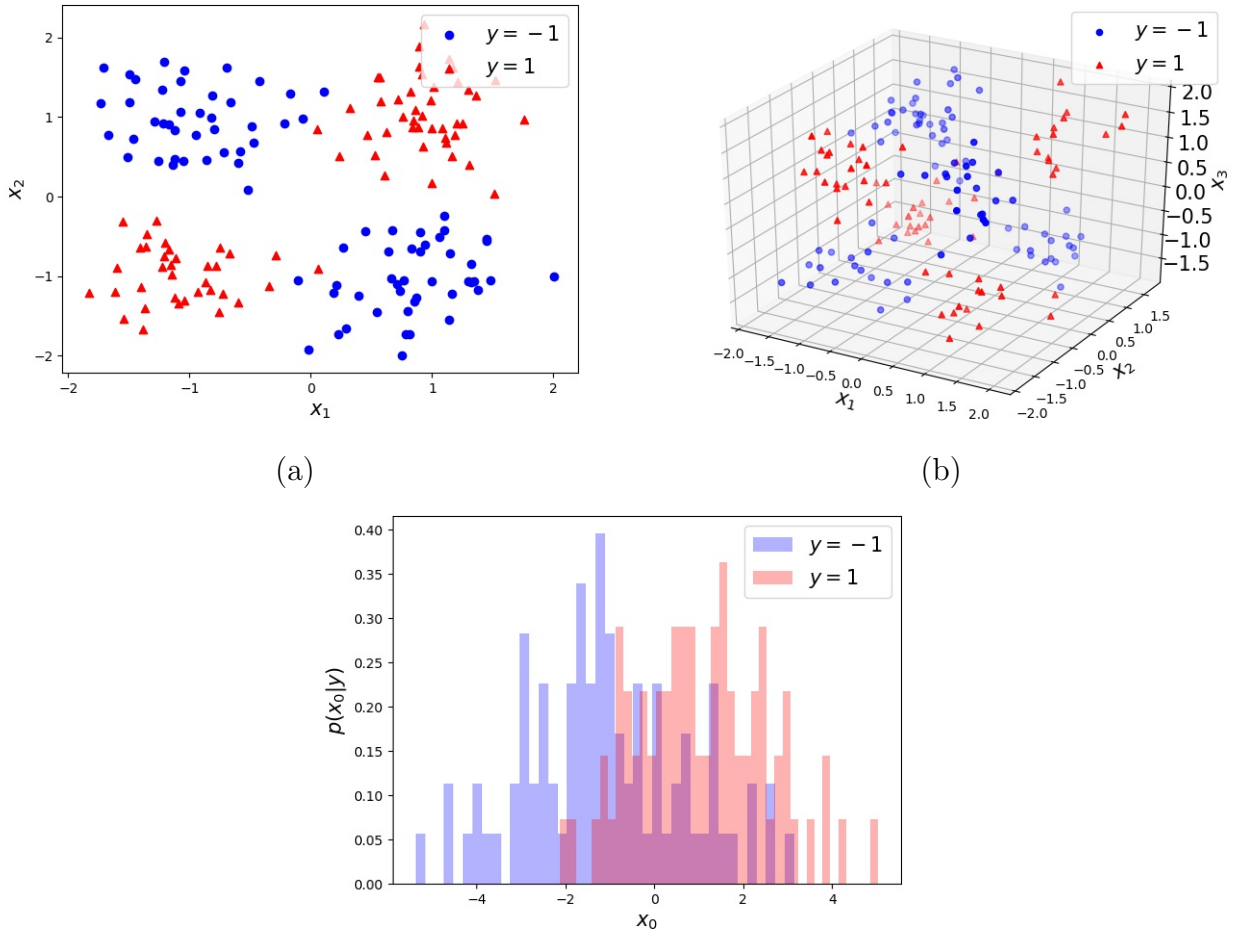
CHAPTER 5

FEATURE-LABEL DEPENDENCE

5.1 Marginal Dependence vs. Joint Dependence

In Section 1.7 we mentioned that one weakness of random forest or decision tree is that it can use only one feature to decide the best split at each decision node. This way of partitioning the data is not efficient when the marginal dependency is weak and the joint dependency is strong. A feature X_i is *marginally dependent* on Y if $X_i \not\perp\!\!\!\perp Y$. A set of features $\{X_1, ...X_k\}$ is *jointly dependent on* Y if $\mathbf{X} \not\perp\!\!\!\perp Y$, where $\mathbf{X} = [X_1, ..., X_k]$. The XOR problem is a good example. Since the original XOR problem contains only 4 points, we generalize it to a dataset containing arbitrary number of data points and use it as a demonstration example.

A generalized XOR dataset $(\mathbf{X}, Y)$, $\mathbf{X} \in \mathbb{R}^2$, is generated in this way: for each i.i.d sample $(\boldsymbol{x}_i, y_i)$, where $\boldsymbol{x}_i = [x_{i1}, x_{i2}]$, x_{i1} is drawn randomly from $\mathcal{N}(\mu_{i1}, \sigma^2)$, where μ_{i1} is randomly chosen from $\{-1, 1\}$ and σ^2 is a predefined value for variance. x_{i2} is generated the same way and independent of x_{i1}, i.e., x_{i2} is drawn randomly from $\mathcal{N}(\mu_{i2}, \sigma^2)$, where μ_{i2} is also randomly chosen from $\{-1, 1\}$. The label y_i is determined by both μ_{i1} and μ_{i2}, i.e., $y_i = \mu_{i1} * \mu_{i2}$, therefore $\mathbf{X} \not\perp\!\!\!\perp Y$. A dataset containing 200 points is visualized in Figure 5.1(a). The XOR dataset can also be extended to an arbitrary dimensional space, $\mathbf{X} \in \mathbb{R}^d$. For each i.i.d sample $(\boldsymbol{x}_i, y_i)$, where $\boldsymbol{x}_i = [x_{i1}, ..., x_{id}]$, x_{ik} is drawn randomly from $\mathcal{N}(\mu_{ik}, \sigma^2)$, and μ_{ik} is randomly chosen from $\{-1, 1\}$. The label y_i is determined by $y_i = \prod_{k=1}^{d} \mu_k$. A 3D XOR dataset is shown in Figure 5.1(b). To test the behavior of decision tree based classifiers, we can also add a marginally dependent feature X_0. For each sample i, x_{i0} is randomly drawn from $\mathcal{N}(y_i, \sigma'^2)$. The class conditional distributions of X_0 is shown in Figure 5.1(c).



(a) (b)

(c)

Figure 5.1. The generalized XOR datasets in (a) 2D and (b) 3D. (c) Class conditional distribution of a marginally dependent feature. In all datasets, blue denotes class -1 and red denotes class 1. $\sigma = 0.4$ for (a) and (b). $\sigma' = 1.5$ for (c).

As we can see from the figures, when $\sigma' > \sigma$, the data points are easier to separate in the dimensions defined by the joint dependent features than in the dimension defined by the marginal dependent feature. To demonstrate the effect of marginal dependent features and joint dependent features in a classification task, we generated 4 artificial datasets:

(i) A 2D XOR dataset, $\sigma = 0.4$

(ii) A 3D XOR dataset, $\sigma = 0.4$

(iii) A dataset with two features (X_1, X_2) generated the same way as in (i) and a marginally

Table 5.1. Test Accuracy (%) on Artificial Datasets.

Classifier	Datasets			
	i	ii	iii	iv
Decision Tree	91.78	65.60	83.50	68.60
Random Forest	97.65	77.60	92.48	75.38
kNN	**98.17**	**97.30**	**97.67**	**95.28**

Table 5.2. Feature Importance on Artificial Datasets.

Classifier	Feature	Datasets			
		i	ii	iii	iv
Decision Tree	X_0	NA	NA	0.377	0.476
	X_1	0.480	0.347	0.319	0.170
	X_2	0.520	0.333	0.304	0.181
	X_3	NA	0.320	NA	0.172
Random Forest	X_0	NA	NA	0.397	0.479
	X_1	0.495	0.334	0.304	0.173
	X_2	0.505	0.332	0.300	0.174
	X_3	NA	0.334	NA	0.174

dependent feature (X_0) with $\sigma' = 1.5$

(iv) A dataset with three features (X_1, X_2, X_3) generated the same way as in (ii) and a marginally dependent feature (X_0) with $\sigma' = 1.5$

We tested the performance of three classifiers on these datasets: decision tree, random forest (with 100 trees) and kNN ($k = 3$). Unlike tree based classifiers, kNN is able to use all features at the same time. It is well suited for the datasets we designed for this experiment. We used the Python scikit-learn implementations for all classifiers under comparison. In each run of experiment, four new datasets are generated with 20% hold-out for testing. 100 runs were performed and the average test accuracy is summarized in Table 5.1. For the 2D XOR dataset (i), even the two features are marginally independent of Y, decision tree and random forest can still achieve high accuracy ($> 91\%$). This is because there are only two features and the tree classifiers are forced to use one of them at each split. When the training size is sufficient, a decision can still partition the data in a way that the two classes are well

separated. However, as we increase the dimension by one, to generate the 3D XOR dataset (ii), it becomes much harder for the tree based learners, while a kNN can still maintain a high accuracy. As we introduce a marginally dependent feature, in the dimension defined by which the two classes are not well separated, the accuracies of both decision tree and random forest dropped significantly (comparing i with iii). In all datasets tested, kNN is able to achieve a high accuracy ($> 95\%$), due to its ability of using both jointly dependent and marginally dependent features.

To observe the effect of introducing a (bad) marginally dependent feature (X_0) on the tree based classifiers, we also list the feature importance generated by the classifiers in Table 5.2. The feature importance is computed as "the (normalized) total reduction of the criterion brought by that feature", according to the scikit-learn documentation. Briefly, it represents how useful it is to the classifier. As shown in the table, when X_0 is absent, both decision tree and random forest treat other jointly dependent features equally important. When X_0 is present, the classifiers tend to use X_0 more than others (this is more obvious in the case of dataset iv), even if X_0 is not a good feature for this classification task. In other words, a decision tree can be "fooled" by features like X_0. As the number of this type of features increases, the jointly dependent features will have little use for the tree based classifiers.

5.2 Minimum Dependence Sets

Motivated by the above examples, we can partition the whole set of features $\mathcal{S}$ into two sets:

$$\begin{aligned} \mathcal{S}_m &= \{X \in \mathcal{S} : X \not\perp\!\!\!\perp Y\}, \\ \overline{\mathcal{S}}_m &= \{X \in \mathcal{S} : X \perp\!\!\!\perp Y\}, \end{aligned}$$

where X denotes a 1-dimensional random variable (feature) and Y is the output label. $X \perp\!\!\!\perp Y$ ($X \not\perp\!\!\!\perp Y$) denotes X and Y being independent (dependent). We call $\mathcal{S}_m$ the

marginal dependence set as it is the set of features whose marginal distribution is dependent on Y. Therefore only features in $\mathcal{S}_m$ are useful for random forests/decision trees. However, "marginal independence" does not imply "joint independence", i.e., it is possible that $\overline{\mathcal{S}}_m \not\perp\!\!\!\perp Y$. Thus, we are more interested in partitioning the features $\mathcal{S}$ into two sets: dependent features $\mathcal{S}_D$ and independent features $\mathcal{S}_I$. $\mathcal{S}_I$ is given by

$$\mathcal{S}_I = \{X \in \mathcal{S} : \forall S \subseteq \mathcal{S}, S \perp\!\!\!\perp Y \text{ implies } S \cup \{X\} \perp\!\!\!\perp Y\}.$$

Unlike $\overline{\mathcal{S}}_m$, $\mathcal{S}_I$ is independent of Y by definition and is therefore of no use for any classifier. To identify $\mathcal{S}_D$, we need the following definition for *minimum dependence set*.

Definition 13. Given an input feature set $\mathcal{S}$ and an output label Y, a feature subset $S \subseteq \mathcal{S}$ is called a *minimum dependence set* (MDS) if $S \not\perp\!\!\!\perp Y$ and $S' \perp\!\!\!\perp Y$, $\forall S' \subset S$.

Then we have the following theorem:

Theorem 14. Given an input feature set $\mathcal{S}$ and an output label Y, $\mathcal{S}$ can be divided into two disjoint sets, i.e., $\mathcal{S}_D$ and $\mathcal{S}_I$, where $\mathcal{S}_D = \overline{\mathcal{S}}_I = \bigcup_i S_i$ is the union of all MDS's in $\mathcal{S}$.

Proof. From the definition of $\mathcal{S}_I$, we have its complement

$$\overline{\mathcal{S}}_I = \{X \in \mathcal{S} : \exists S \subseteq \mathcal{S}, S \perp\!\!\!\perp Y \text{ and } S \cup \{X\} \not\perp\!\!\!\perp Y\}.$$

It suffices to show that $\bigcup_i S_i = \overline{\mathcal{S}}_I$.

For any $X \in \bigcup_i S_i$, there exist i^* such that $X \in S_{i^*}$. As S_{i^*} is an MDS, we have $S_{i^*} \setminus \{X\} \perp\!\!\!\perp Y$ and $S_{i^*} \not\perp\!\!\!\perp Y$, i.e., $X \in \overline{\mathcal{S}}_I$. Therefore, $\bigcup_i S_i \subseteq \overline{\mathcal{S}}_I$.

For any $X \in \overline{\mathcal{S}}_I$, there exists $S \subseteq \mathcal{S}$ such that $S \perp\!\!\!\perp Y$ and $S \cup \{X\} \not\perp\!\!\!\perp Y$. Let $S^* = S \cup \{X\}$. For each element X' in S^* ($X' \neq X$), we remove X' if $(S^* \setminus \{X'\}) \not\perp\!\!\!\perp Y$. The resulting S^* is an MDS. Hence $X \in \bigcup_i S_i$. Therefore, $\overline{\mathcal{S}}_I \subseteq \bigcup_i S_i$. This completes the proof. □

However, since the number of MDS's grows quadratically with the size of $\mathcal{S}$, it is computationally unfeasible to identify $\mathcal{S}_D$. A computationally more friendly partition of $\mathcal{S}$ is $\mathcal{S}_C$ and $\overline{\mathcal{S}}_C$, where $\mathcal{S}_C$ is called the *minimum dependence cover* (MDC) defined as follows:

Definition 15. Given an input feature set $\mathcal{S}$ and an output label Y, a feature subset $\mathcal{S}_C$ is called a minimum dependence cover if $\mathcal{S}_C = \bigcup_{i=1}^{s} S_i$ where S_i, $i = 1, \cdots, s$, are mutually disjoint MDS's, and $\overline{\mathcal{S}}_C \perp\!\!\!\perp Y$.

We show several algorithms to identify an MDC in the next section. Once an MDC is identified, and if it is a superset of $\mathcal{S}_m$, we are interested in how to improve random forest by utilizing all features in this MDC, which will be discussed in Chapter 6. The relationship between the set concepts covered in this section is $\mathcal{S}_m \subseteq \mathcal{S}_C \subseteq \mathcal{S}_D \subseteq \mathcal{S}$.

5.3 Identifying an Minimum Dependence Cover

Given an input feature set $\mathcal{S}$ and an output label Y, finding the an MDC $\mathcal{S}_C$ can be achieved by first identifying the marginal dependent set $\mathcal{S}_m$ and then the MDC of $\overline{\mathcal{S}}_m$, i.e., $[\overline{\mathcal{S}}_m]_C$, taking the union of the two gives $\mathcal{S}_C$, i.e., $\mathcal{S}_C = \mathcal{S}_m \cup [\overline{\mathcal{S}}_m]_C$. Identifying $\mathcal{S}_m$ is easy and can be done in linear time, i.e., $O(d)$, where d is size of $\mathcal{S}$. An algorithm of finding $\mathcal{S}_m$ is given in Algorithm 1. For each dependence test, a p-value is calculated and compared with a predefined significance level α. The p-value of a dependence test with the alternative hypothesis being $X \not\perp\!\!\!\perp Y$ is denoted as $p_{X \not\perp\!\!\!\perp Y}$. If $p_{X \not\perp\!\!\!\perp Y} \leqslant \alpha$, we say X is dependent on Y, otherwise not dependent. We next propose several algorithms to identify an MDC $\mathcal{S}_C$ given an marginally independent features set $\mathcal{S}$ and output label Y.

We first present a top-down approach assuming that a feature set is dependent if it contains at least one MDS. Then we have Algorithm 2 to identify one MDS from the feature set. Each feature is hold-out and the remaining ones are tested for dependence with Y. If the dependence disappears, then the hold-out feature belongs to an MDS and is kept for further dependence tests, otherwise removed from the feature set and not participating dependence tests with other features. This process is repeated until every feature has been hold-out

once. To identify an MDC, a less efficient approach is shown in Algorithm 3. It simply find one MDS at a time, removing the identified MDS from the feature set, and repeat the process on the remaining feature set. The time complexity for this algorithm is $O(d^2)$ since the for loop in FindMDS($\mathcal{S}$, Y,α) needs $O(d)$ tests and the while loop iterates $O(d)$ times. Usually the size of $\mathcal{S}$ is large, therefore it is desired to have a faster algorithm. We then provide a $O(d\log(d))$ algorithm to find MDC, as shown in Algorithm 4. This algorithm uses a divide-and-conquer approach, which divides the original problem into two subproblems with half of the size. There will be $O(\log(d))$ number of combinations of subproblems and each combination requires $O(d)$ dependence tests, therefore a total of $O(d\log(d))$ operations. However, Algorithm 4 does not guarantee to return an MDC. Instead, it can be a subset of MDC since FINDMDS (line 15) can identify at most one MDS instead of all. As a consequence, the algorithm may be repeatedly applied to the remaining features, i.e., $\mathcal{S}\setminus\mathcal{S}_C$, if $(\mathcal{S}\setminus\mathcal{S}_C) \not\perp\!\!\!\perp Y$. Empirically this is an more efficient algorithm than Algorithm 3.

The problem with the top-down approach is that as more independent features being included in the feature set, the dependence of the whole set with Y becomes weaker. As a consequence, it is possible that the dependence between a big feature set with Y is too weak to be tested as significant in a dependence test even if the feature set contains an MDS. In another words, the assumption made in the top-down approach – a feature set is dependent if it contains at least one MDS – is not always correct. We then provide a bottom-up approach, which first identifies disjoint MDS's of size = 2, then size = 3, and so on. An algorithm to identify MDS's of size = 2 is shown in Algorithm 5. In this algorithm, a pair of features are evaluated in a dependence test with Y, they are added to MDC and excluded for future test if dependent, or kept for pairing with other features otherwise. For greater sizes, we can just increase the nesting levels of the for loop. The time complexity of this bottom-up approach is $O(d^2)$, which is not efficient when the feature set is large and a lot of independent features exist.

Algorithm 1 Find the marginal dependent set $\mathcal{S}_m$

function FINDSM($\mathcal{S}, Y, \alpha$)
    initialize $\mathcal{S}_m$ as an empty set
    for X in $\mathcal{S}$ **do**
        if $p_{X \not\perp\!\!\!\perp Y} \leqslant \alpha$ **then**
            Add X to $\mathcal{S}_m$
        end if
    end for
    return $\mathcal{S}_m$
end function

Algorithm 2 Find a minimum dependent set (MDS)

function FINDMDS($\mathcal{S}, Y, \alpha$)
    $\mathcal{S}_{MDS} = \mathcal{S}$
    for X in $\mathcal{S}_{MDS}$ **do**
        $\mathcal{S}' = \mathcal{S}_{MDS} \setminus \{X\}$
        if $p_{\mathcal{S}' \not\perp\!\!\!\perp Y} \leqslant \alpha$ **then**
            $\mathcal{S}_{MDS} = \mathcal{S}'$
        end if
    end for
    return $\mathcal{S}_{MDS}$
end function

Algorithm 3 Find an MDC ($O(d^2)$, a top-down approach)

function FINDMDC($\mathcal{S}, Y, \alpha$)
    assume $\mathcal{S}$ is marginally independent of Y
    initialize $\mathcal{S}_C$ as an empty set
    while $p_{\mathcal{S} \not\perp\!\!\!\perp Y} \leqslant \alpha$ **do**
        $\mathcal{S}_{MDS}$ =FINDMDS($\mathcal{S}$, Y,α)
        $\mathcal{S}_C = \mathcal{S}_C \cup \mathcal{S}_{MDS}$
        $\mathcal{S} = \mathcal{S} \setminus \mathcal{S}_{MDS}$
    end while
    return $\mathcal{S}_C$
end function

Algorithm 4 Find an MDC ($O(d\log(d))$, a top-down approach)

function FINDMDC2($\mathcal{S}, Y, \alpha$)
    assume $\mathcal{S}$ is marginally independent of Y
    initialize $\mathcal{S}_C$ as an empty set
    if $p_{\mathcal{S}\not\perp\!\!\!\perp Y} > \alpha$ **then**
        return $\mathcal{S}_C$
    end if
    random partition $\mathcal{S}$ into two disjoint sets: $\mathcal{S}_1$ and $\mathcal{S}_2$. The size difference between the two is at most 1.
    $\mathcal{S}_C1$ = FINDMDC2($\mathcal{S}_1$,Y)
    $\mathcal{S}_C2$ = FINDMDC2($\mathcal{S}_2$,Y)
    $\mathcal{S}_C = \mathcal{S}_C \cup \mathcal{S}_C1 \cup \mathcal{S}_C2$
    $\mathcal{S} = \mathcal{S} \setminus \mathcal{S}_C$
    if $p_{\mathcal{S}\not\perp\!\!\!\perp Y} > \alpha$ **then**
        return $\mathcal{S}_C$
    end if
    $\mathcal{S}_{MDS}$ =FINDMDS($\mathcal{S}$, Y)
    $\mathcal{S}_C = \mathcal{S}_C \cup \mathcal{S}_{MDS}$
    return $\mathcal{S}_C$
end function

Algorithm 5 Find an MDC ($O(d^2)$, a bottom-up approach)

function FINDMDC($\mathcal{S} = \{X_i, i = 1, ..., n\}, Y, \alpha$)
    assume $\mathcal{S}$ is marginally independent of Y
    initialize $\mathcal{S}_C$ as an empty set
    for $i = 1$ to $n - 1$ **do**
        for $j = i + 1$ to n **do**
            if $p_{\{X_i,X_j\}\not\perp\!\!\!\perp Y} \leqslant \alpha$ **then**
                $\mathcal{S}_C = \mathcal{S}_C \cup \{X_i, X_j\}$
                break
            end if
        end for
    end for
    return $\mathcal{S}_C$
end function

5.4 Stronger Dependence Sets

As tree based learners can only take advantage of features in $\mathcal{S}_m$, wasting those in $\mathcal{S}_C \setminus \mathcal{S}_m$, we are interested in transforming features in $\mathcal{S}_C \setminus \mathcal{S}_m$ into marginally dependent features (we show how in Chapter 6). However, in practice, the size of $\mathcal{S}_C \setminus \mathcal{S}_m$ might be too small to make a difference. If we set the significance level α to be very small, the size of $\mathcal{S}_C \setminus \mathcal{S}_m$ is increased but we may not able to identify any significant joint dependence. Motivated by this, we generalize the concept of minimum dependence set of size greater than 1 to a *stronger dependence set*.

Definition 16. Given an input feature set $\mathcal{S}$ and an output label Y, a feature subset $S \subseteq \mathcal{S}$ is called a stronger dependence set (SDS), if for $\forall S' \subset S$, the dependency between S' and Y is less than the dependency between S and Y, i.e., $p_{S' \not\perp\!\!\!\perp Y} > p_{S \not\perp\!\!\!\perp Y}$.

It is easy to show that MDS of size greater than 1 is a special case of SDS.

Proof. If S is an MDS with a significant level α, by definition, $p_{S \not\perp\!\!\!\perp Y} < \alpha$ and $p_{S' \not\perp\!\!\!\perp Y} > \alpha$ for $\forall S' \subset S$. Therefore, we have $p_{S' \not\perp\!\!\!\perp Y} > p_{S \not\perp\!\!\!\perp Y}$, $\forall S' \subset S$, thus S is also a SDS. □

To improve a decision tree classifier, we aim to first partition the feature set $\mathcal{S}$ into $\mathcal{S}_m$ and $\overline{\mathcal{S}}_m$ by setting a significant level α, then find all the disjoint SDS's in $\overline{\mathcal{S}}_m$ (finding all SDS's is too computational expensive), called a *minimum stronger dependence cover*.

Definition 17. Given an input feature set $\mathcal{S}$ and an output label Y, a feature subset is called a minimum stronger dependence cover (SDC), denoted as $\mathcal{S}_{SC}$ if $\mathcal{S}_{SC} = \bigcup_{i=1}^{s} S_i$ where S_i, $i = 1, \cdots, s$, are mutually disjoint SDS's, and $\overline{\mathcal{S}_{SC} \cup \mathcal{S}_m} \perp\!\!\!\perp Y$.

Since a MDS (of size > 1) is a special case of SDS, a SDC is a bigger set than MDC$\setminus\mathcal{S}_m$.

5.5 Identifying an SDC

The $O(d \log(d))$ algorithm to identify an MDC uses a divide-and-conquer (top down) approach. This relies on a good property of an MDC, which is that we can (most of the

time) tell if there exists an MDS in $\mathcal{S}$ by testing the dependence between $\mathcal{S}$ and Y. However, we are not able to tell if there exists an SDS in $\mathcal{S}$ by testing the dependence between $\mathcal{S}$ and Y. Since an SDS can be of any size greater than 1, we use a bottom up approach, similar to Algorithm 5, to first identify disjoint SDS's of size = 2, then size = 3, and so on. When looking for SDS's of size = 2, we only need to compare the joint dependency between a pair of features (X_i, X_j) and Y, $p_{(X_i,X_j)\not\perp\!\!\!\perp Y}$, and their marginal dependencies, $p_{X_i\not\perp\!\!\!\perp Y}$ and $p_{X_j\not\perp\!\!\!\perp Y}$. If $p_{(X_i,X_j)\not\perp\!\!\!\perp Y} < \min(p_{X_i\not\perp\!\!\!\perp Y}, p_{X_j\not\perp\!\!\!\perp Y})$, then $\{X_i, X_j\}$ is an SDS. Notice that we are only looking for disjoint SDS's, therefore it is not necessary to examine the dependency for every feature of features. Once an SDS is identified, the pair is excluded from future evaluations. The identification of SDS's of larger sizes are more computationally expensive. However, there will be fewer features left after SDS's of smaller sizes are discovered. In addition, to find an SDS of size m, we still only need to compare the joint dependency of the m features with the marginial dependencies, ignoring joint dependencies of size in $[3, m-1]$. This is because any joint dependencies of size in $[3, m-1]$ should be weaker than the strongest marginal dependency, otherwise these features would be discovered as an SDS in a previous step. The iteration stops when there is not enough features left. In practice, we found it efficient to identify only SDS's of size = 2 and 3. Few features left after all disjoint SDS's of size = 2 are discovered. The sudo code to find all disjoint SDS's of size = 2 is shown in Algorithm 6. Similar to Algoritm 5, the time complexity is $O(d^2)$ and the nesting levels of the for loop can be increased for greater sizes.

Unlike Algorithm 1-5, Algorithm 6 does not require a significance level α. Only p-values are compared. Therefore it is possible that an SDS has a higher p-value than the α used to select marginal dependent features $\mathcal{S}_m$. In practice, we set α to be a very small value to separate out very strong marginal dependent features and allow for discovery of stronger joint dependence. Even if for some features $p_{joint} > \alpha$, they may still be helpful to improve classification performance. However, it worth filtering out features with extreme large p_{joint}, e.g. $p_{joint} = 1$. These are usually features with constant values, for example, the boarder

Algorithm 6 Find an SDC ($O(d^2)$)

function FINDSDC($\mathcal{S} = \{X_i, i = 1, ..., n\}, Y$)
    assume $\mathcal{S}$ is marginally independent of Y
    initialize $\mathcal{S}_{SC}$ as an empty set
    for $i = 1$ to $n - 1$ **do**
        for $j = i + 1$ to n **do**
            if $p_{(X_i,X_j)\not\perp\!\!\!\perp Y} < \min(p_{X_i\not\perp\!\!\!\perp Y}, p_{X_j\not\perp\!\!\!\perp Y})$ **then**
                $\mathcal{S}_{SC} = \mathcal{S}_{SC} \cup \{(X_i, X_j\}$
                break
            end if
        end for
    end for
    return $\mathcal{S}_{SC}$
end function

pixels in the MNIST data.

5.6 Dependence Test

The p-values used in Algorithm 1-6 are obtained from performing dependence tests using Gini correlation as the test statistic. As discussed in Section 2.3, gCor is a perfect feature-label dependence measure. It can measure the marginal dependence as well as the joint dependence, and the two can also be compared via p-values. The direct compare of the test statistic is not fair since the dimensions are different, therefore permutation tests must be performed to obtain the p-values. However, performing permutation tests can be computationally expensive even with parallelization. From (2.2), a p-value of 0.001 can only be obtained with at least 1000 permutation tests. The higher precision we want on the p-value, the more permutation tests needed to be performed. In our experiment, it seems to be sufficient to set the number of permutation tests to be 5000. However, it is a waste to run 5000 times if the p-value is ~ 0.1, because a similar value can be obtained with only 100-200 runs. Thus, in our implementation, we start with a small number of tests, e.g., 100, and if the resulting p-value is zero, we will double the number of tests. Repeat this process until a non-zero p-value is obtained or the maximum number of test (5000) is reached.

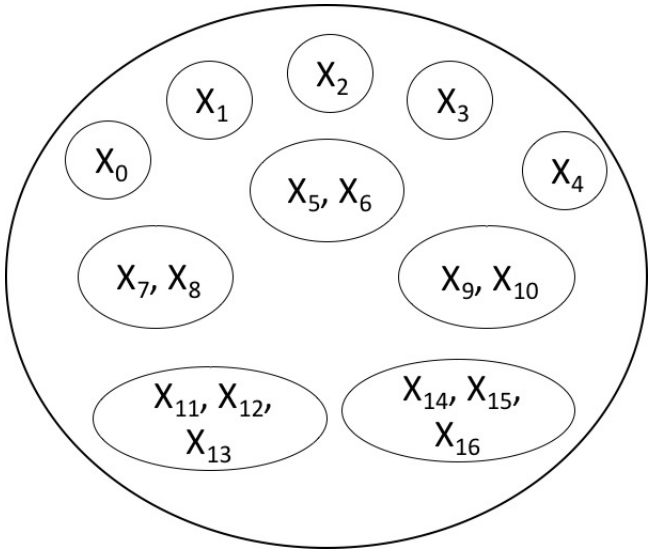


Figure 5.2. A schematic diagram of 17 features consisting of 10 MDS's. The bold circle indicates the whole feature set. The thin circles indicates MDS's.

5.7 Algorithm Evaluations

To verify the correctness of the proposed algorithms, we tested them on an artificial dataset with 17 features. The features consist of 10 MDS's: 5 MDS's are of size 1, 3 MDS's are of size 2 and 2 MDS's are of 3. A schematic diagram of the 10 MDS's is shown in Figure 5.2. Marginal dependent features (MDS's of size 1) are generated as described in Section 5.1. The MDS's of size 2 are generated the same way as the 2D XOR example, and the MDS's of size 3 are generated the same way as the 3D XOR example. The number of data points is 200. We first used Algorithm 1 to find the marginal dependent features, and compared the following proposed algorithms on the remaining ones:

1. The top-down $O(n \log(n))$ algorithm to identify an MDC (Top-down MDC).

2. The bottom-up $O(n^2)$ algorithm to identify an MDC (Bottom-up MDC).

3. The bottom-up $O(n^2)$ algorithm to identify an SDC (Bottom-up SDC).

The results are summarized in Table 5.3. Algorithm 1 was used to identify marginally dependent features and labeled them as -1, with the significant level α set at 0.05. The p-value of a marginal dependence test is denoted as p_m. Other algorithms worked on the marginally

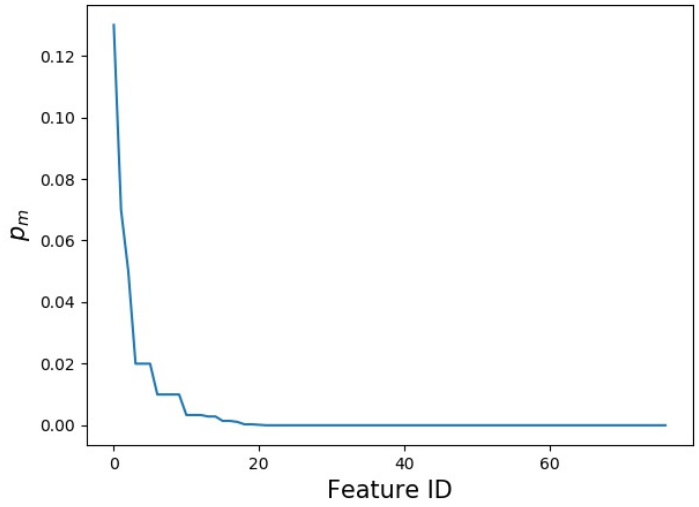


Figure 5.3. The marginal p-values (p_m) of the mice protein expression dataset in descending order.

independent features and assigned each feature a non-negative integer label. 0 means the feature is not dependent ($\in \mathcal{S}_I$), other values indicate the feature belongs to a MDS (or SDS) and features in the same MDS (or SDS) have the same label. The p-values of the joint dependence of features in the same MDS, SDS or $\mathcal{S}_I$ are listed in the columns denoted as p_{joint}. Comparing Table 5.3 with the ground truth shown in Figure 5.2, we can see that Algorithm 1 effectively identified all marginally dependent features. Among the other three approaches tested, only bottom-up MDC achieved an exact match with the ground truth. Top-down MDC was not able to detect any MDS among $\{X_{11}, ..., X_{15}\}$, which is because the joint dependence of $\{X_{11}, ..., X_{15}\}$ is too weak with $p_{joint} = 0.07 > \alpha$, so the algorithm stopped. It also detected $\{X_5, X_6, X_7, X_8\}$ as one MDS instead of two. This is because when any of these four features is removed, the remaining three have weak dependency with a p-value larger than 0.05, even though there is an MDS in these three features. Bottom-up SDC, on the other hand, identified $\{X_{13}, X_{15}\}$ and $\{X_{14}, X_{16}\}$ as two SDS's because the joint p-values are smaller than the marginal p-values, i.e., $0.2 < \min(0.26, 0.43)$ and $0.26 < \min(0.31, 0.27)$. Since the algorithm only looks for disjoint SDS's, $\{X_{11}, X_{12}, X_{13}\}$ and $\{X_{14}, X_{15}, X_{16}\}$ were not able to be identified.

Table 5.3. Testing Proposed Algorithms on an Artificial Dataset (10 MDS's).

Feature	p_m	Top-down MDC		Bottom-up MDC		Bottom-up SDC	
		Label	p_{joint}	Label	p_{joint}	Label	p_{joint}
X_0	0	-1	NA	-1	NA	-1	NA
X_1	0	-1	NA	-1	NA	-1	NA
X_2	0	-1	NA	-1	NA	-1	NA
X_3	0	-1	NA	-1	NA	-1	NA
X_4	0	-1	NA	-1	NA	-1	NA
X_5	0.94	1	0	1	0	1	0
X_6	0.22	1		1		1	
X_7	0.15	1		2	0	2	0
X_8	1	1		2		2	
X_9	0.92	2	0	3	0	3	0
X_{10}	0.91	2		3		3	
X_{11}	0.44	0	0.07	4	0.0033	0	0.73
X_{12}	0.74	0		4		0	
X_{13}	0.26	0		4		4	0.2
X_{14}	0.31	0		5	0.0006	5	0.26
X_{15}	0.43	0		5		4	0.2
X_{16}	0.27	0		5		5	0.26

We then tested the proposed algorithms on a real dataset. We used the mice protein expression dataset available on UCI (Higuera et al., 2015). The dataset contains 77 features, 8 classes and 1080 instances. Each feature is a protein expression and each class is a treatment. We chose to use this dataset because we observed some features have weak marginal dependence and would like to discover any (stronger) joint dependence. Since this dataset contains missing values, we imputed the missing values by column means (each column is a feature). For all experiments on this datasets, we used a random 50% hold-out as the test set and all dependence tests were performed using the training set. The marginal dependence of the features are described in Figure 5.3, where p_m is presented in a descending order. As we see from the figure, the majority of the features are significantly marginally dependent on the class label, with 56 of those having a p_m equals 0. To discover joint dependence, we set the significant level $\alpha = 0$ and run the proposed algorithm on the remaining 21 features. The identified MDS/SDS's are listed in Table 5.4. All three approaches were able to detect some

joint dependence. This is reasonable for this dataset since some proteins react with each other, e.g., enzyme-inhibitor, antibody-protein, etc. (Jones and Thornton, 1996) Bottom-up SDC is able to identify more joint dependent sets than the MDC approaches. This is because a SDS does not require $p_{joint} = 0$ but a MDS does. The top-down MDC identified the least number of MDS's due to the dependence drop at the upper levels. The above results suggest that if the computation time is not an issue, the bottom-up MDC is a better approach than the top-down MDC. An MDC contains fewer but more significant joint dependence sets than an SDC.

5.8 Summary

In this chapter, we first used some artificial datasets to show the difference between marginally dependent and jointly features and how a tree classifier fails to use joint dependencies and performs poor on datasets with strong joint dependence but weak marginal dependence. For real datasets, we aim to first identify these joint dependencies and then improve tree cased classifier by taking advantage of those. We therefore proposed the concepts of minimum dependence set (MDS) and stronger dependence set (SDS) to define joint dependent feature sets, followed by algorithms for implementations. We showed that MDS defines less dependent sets but the joint dependencies are more significant than those defined by SDS. In the next Chapter, we present an approach to utilize these joint dependence feature sets in decision tree based classifiers.

Table 5.4. Testing Proposed Algorithms on an Real Dataset (Mice Protein Expression).

	Top-down MDC	Bottom-up MDC	Bottom-up SDC
MDS/SDS's	$\{X_{44}, X_{72}\}, \{X_4, X_{58}\}, \{X_{49}, X_{59}\}, \{X_{16}, X_{51}\}, \{X_3, X_{68}\}, \{X_{29}, X_{66}\}, \{X_{14}, X_{28}\}$	$\{X_3, X_{26}\}, \{X_4, X_{28}\}, \{X_{14}, X_{27}\}, \{X_{16}, X_{49}\}, \{X_{29}, X_{66}\}, \{X_{44}, X_{58}\}, \{X_{51}, X_{52}\}, \{X_{68}, X_{69}\}$	$\{X_3, X_{14}\}, \{X_4, X_{26}\}, \{X_{16}, X_{27}\}, \{X_{23}, X_{28}\}, \{X_{29}, X_{31}\}, \{X_{44}, X_{49}\}, \{X_{51}, X_{52}\}, \{X_{58}, X_{59}\}, \{X_{66}, X_{68}\}, \{X_{69}, X_{72}\}$
# of MDS/SDS's	7	8	10
$\lvert\mathcal{S}_I\rvert$	7	5	1

CHAPTER 6

FEATURE DEPENDENCE MAPPING

In the previous Chapter, we show that decision tree based classifiers can only take advantage of marginal dependent features, wasting the joint dependence. This motivated us to map the joint dependent feature set to another space where it becomes marginally dependent. Mathematically, given a joint dependent but marginally independent feature set $\mathcal{S} = \{X_1, ...X_p\}$, let $\mathbf{X} = [X_1, ...X_p]$, Y is the class variable and $\mathbf{X} \not\perp\!\!\!\perp Y$, we are seeking for a function $f : \mathbb{R}^p \to \mathbb{R}$, such that $f(\mathbf{X}) \not\perp\!\!\!\perp Y$. We call this approach as *feature dependence mapping* (FDM) and f is the *mapping function*.

6.1 Mapping Functions

Any function that maps a multidimensional vector to a 1-dimensional one can be used as the mapping function for FDM. Here we present two approaches to FDM, 1) random projection and 2) distance to a random sample.

6.1.1 Random Projection

Random projection (RP) is a simple dimensionality reduction technique that projects the original p-dimensional data to a lower k-dimensional space using a random projection matrix $\mathbf{R}_{p\times k}$. Assuming the original data is $\mathbf{X}_{n\times p}$, the projected data $\mathbf{X}'_{n\times k}$ can be obtained by matrix multiplication, i.e.,

$$\mathbf{X}'_{n\times k} = \mathbf{X}_{n\times p}\mathbf{R}_{p\times k}.$$

The motivation of using random projection comes of Johnson-Lindenstrauss lemma (Johnson and Lindenstrauss, 1984), which states that if data points lie in a vector space of sufficiently

high dimension, they are possible to be projected into a lower dimensional space where the distances between them are approximately preserved. Random projection is computationally efficient: the calculation of $\mathbf{X}_{n\times p}\mathbf{R}_{p\times k}$ takes $O(npk)$. If the data matrix $\mathbf{X}$ is sparse, with about c non-zero values per row, the time complexity is then $O(nck)$ (Bingham and Mannila, 2001).

For our purpose of FDM, we need the projection matrix to be a p-dimensional vector of size $p \times 1$, denoted as $\boldsymbol{r}$, such that the projected data $X' = \mathbf{X}\boldsymbol{r}$ is one-dimensional. In our implementation, a random vector r can be generated using random numbers uniformly sampled from [-1, 1]. Usually a projection vector is a unit vector, i.e., $|r| = 1$. Since the length of r does not affect the dependence of the projected variable, it is not necessary to normalize $\boldsymbol{r}$ for FDM. We denote a feature mapping function using random projection as f_{RP} defined by

$$f_{RP}(\mathbf{X}; \boldsymbol{r}) = \mathbf{X}\boldsymbol{r}.$$

6.1.2 Distance to a Random Landmark

The distance operation $|\cdot|$ is a function that maps $\mathbb{R}^p$ to $\mathbb{R}$. The motivation of using distance as the mapping function comes from the superior performance of the kNN classifier on the artificial datasets presented in Section 5.1. The way how kNN works is to find the test example's k nearest neighbors and predict the label using the majority of the neighbors. And the information used for find the neighbors is the Euclidean distance between data points. In the k-neighbor search, all features are used to calculate the distance and this explains why kNN is able to use the joint dependence.

We hereby present the *distance to a random landmark* (DRL) approach, that maps the high dimensional data $\mathbf{X}_{n\times p}$, presented using its n samples $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$ as

$$\mathbf{X}_{n\times p} = \begin{bmatrix} \boldsymbol{x}_1^T \\ ... \\ \boldsymbol{x}_n^T \end{bmatrix},$$

to a one dimensional vector $\boldsymbol{d} = [d_1, ..., d_n]^T$, where d_i is the distance between data point i and a random landmark $\boldsymbol{x}_l$ selected from the training data, i.e.,

$$d_i = |\boldsymbol{x}_i - \boldsymbol{x}_l|.$$

We denote a feature mapping function using DRL as f_{DRL} defined by

$$f_{DRL}(\mathbf{X}; \boldsymbol{x}_l) = D(\mathbf{X}, \boldsymbol{x}_l) := \begin{bmatrix} |\boldsymbol{x}_1 - \boldsymbol{x}_l| \\ ... \\ |\boldsymbol{x}_n - \boldsymbol{x}_l| \end{bmatrix}.$$

6.2 Mapping Selection

Both mapping functions discussed above are able to map a multi-dimensional vector $\mathbf{X}$ to a 1-dimensional variable X', but they do not guarantee that $X' \not\perp\!\!\!\perp Y$. Notice that either function depends on a random parameter θ, which is a random vector $\boldsymbol{r}$ in the case of f_{RD} and a random landmark $\boldsymbol{x}_l$ in the case of f_{DRL}, thus it is desired to select k optimal θ's from m randomly generated ones with the objective function defined by dependence. The mapping selection procedure is described as follows. Given the training data $(\mathbf{X}_{n\times p}, Y)$ and a choice of mapping function $f \in \{f_{RD}, f_{DRL}\}$, m random parameters are generated, i.e., $\theta_1, ..., \theta_m$, the optimal θ^* can be obtained by

$$\theta^* = \underset{\theta\in\{\theta_1,...,\theta_m\}}{\arg\min}\ p_{f(\mathbf{X};\theta) \not\perp\!\!\!\perp Y}. \tag{6.1}$$

The computation of p-value needs a number of permutation tests. To reduce computation cost, the sample estimator of Gini correlation gCor_n can be used to estimate dependence, then (6.1) becomes

$$\theta^* = \underset{\theta\in\{\theta_1,...,\theta_m\}}{\arg\max}\ \mathrm{gCor}_n(f(\mathbf{X};\theta), Y). \tag{6.2}$$

The objective functions used in (6.1) and (6.2) can also be used to select the best k θ's, $\theta_1^*, ..., \theta_k^*$. In this case, the mapping function f is consist of k functions defined by $\theta_1^*, ..., \theta_k^*$, denoted as $f(\mathbf{X}; \boldsymbol{\theta}^*)$, where $\boldsymbol{\theta}^* = [\theta_1^*, ..., \theta_k^*]$. Explicitly,

$$f(\mathbf{X}; \boldsymbol{\theta}^*) = [f(\mathbf{X}; \theta_1^*), ..., f(\mathbf{X}; \theta_k^*)] = [X_1', ..., X_k'],$$

where $X_1', ...X_k'$ are mapped features that are marginally dependent on Y. When $k < p$, FDM can also serve as a dimensionality reduction approach. Here we only test the effect of dependence mapping, therefore we keep the dimension the same by setting $k = p$. The number of random parameters generated m should also be predefined. In our implementation, we set $m = 10$. The mapping selection is only performed in the training phase to obtain $\boldsymbol{\theta}^*$. In the testing phase, the learned $\boldsymbol{\theta}^*$ is directly applied.

6.3 Feature Dependence Mapping: An Algorithmic View

Feature dependence mapping relies on the identification of marginally independent (or weakly dependent) but jointly (more) dependent feature sets, MDS or SDS's. Therefore, the first step of FDM is to separate out marginally dependent features $\mathcal{S}_m$ and from the remaining feature set $\overline{\mathcal{S}_m}$, discover an MDC, $\overline{\mathcal{S}_m}_C$, or an SDC, $\overline{\mathcal{S}_m}_{SC}$. We use the notation $\mathbf{X}_{\mathcal{S}_m}$ to denote the data represented using only marginal dependent features. This step can be done by the algorithms proposed in Chapter 5. Next, for each MDS (SDS) in $\overline{\mathcal{S}_m}_C$ ($\overline{\mathcal{S}_m}_{SC}$), denoted as $\mathbf{X}_{S_i}$, find its optimal feature dependence mapping $f(\mathbf{X}; \boldsymbol{\theta}_i^*)$ by selecting $\boldsymbol{\theta}_i^*$ from m ($m = 10$ in our implementation) randomly generated θ's, $\theta_{i1}, ..., \theta_{im}$, according to (6.2). Then map $\mathbf{X}_{S_i}$ to $\mathbf{X}'_{S_i}$ by

$$\mathbf{X}'_{S_i} = f(\mathbf{X}_{S_i}; \boldsymbol{\theta}_i^*).$$

Assuming there are s disjoint MDS's (SDS's) in MDC (SDC), let $\boldsymbol{\Theta}^*$ be the set of optimal parameters, i.e., $\boldsymbol{\Theta}^* = \{\boldsymbol{\theta}_1^*, ..., \boldsymbol{\theta}_s^*\}$, $\mathcal{F}(\mathbf{X}; \boldsymbol{\Theta}^*)$ be the overall mapping function applied on the

whole data $\mathbf{X}$ (represented using all features), the new data generated after FDM is then

$$\begin{aligned}\mathbf{X}' &= \mathcal{F}(\mathbf{X}; \mathbf{\Theta}^*) \\ &= [\mathbf{X}_{\mathcal{S}_m}, f(\mathbf{X}_{S_1}; \boldsymbol{\theta}_1^*), ..., f(\mathbf{X}_{S_s}; \boldsymbol{\theta}_s^*)] \\ &= [\mathbf{X}_{\mathcal{S}_m}, \mathbf{X}'_{S_1}, ..., \mathbf{X}'_{S_s}].\end{aligned}$$

The transformed data $\mathbf{X}'$ has the same dimension as the original data $\mathbf{X}$ by setting $k = p$ as described in Section 6.2.

When applying FDM on decision trees, the training data $\mathbf{X}_{train}$ is first used to learn the optimal mapping, $\mathcal{F}(\mathbf{X}; \mathbf{\Theta}^*)$, and then both $\mathbf{X}_{train}$ and $\mathbf{X}_{test}$ are transformed to $\mathbf{X}'_{train}$ and $\mathbf{X}'_{test}$ by

$$\begin{aligned}\mathbf{X}'_{train} &= \mathcal{F}(\mathbf{X}_{train}; \mathbf{\Theta}^*) \\ \mathbf{X}'_{test} &= \mathcal{F}(\mathbf{X}_{test}; \mathbf{\Theta}^*).\end{aligned}$$

The decision tree will then be trained using $\mathbf{X}'_{train}$ and tested on $\mathbf{X}'_{test}$.

6.4 Random Forests with Feature Dependence Mapping

We present two approaches to apply FDM to random forests. The straight forward one is to first transform the training and test data and feed those to a random forest, the same way we apply it on a decision tree. In this case, FDM is a data prepossessing step before training a random forest. The other approach is to use FDM at the tree level, i.e., each tree has its own FDM transformed data to train on. Because of the randomness in the FDM procedure, which is the random sampling of θ's for selection, each tree can have the data transformed to a different feature space. This is likely to introduce more variations (less correlation) among trees, which is desired for an ensemble learner. We show the performance of these two approaches in 6.5. In our experiments, we name the former approach as FDM_RF and the latter one as RF_FDM. The schematic diagrams of these two approaches are shown

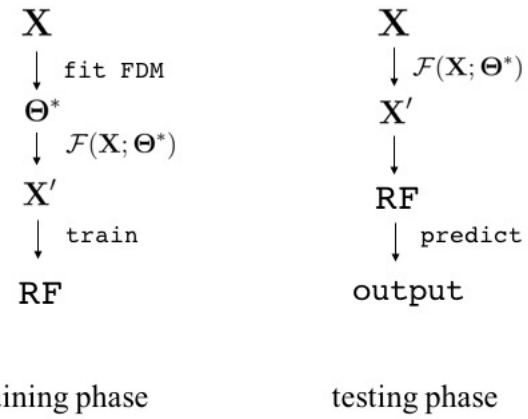


Figure 6.1. A schematic diagram of the FDM_RF approach.

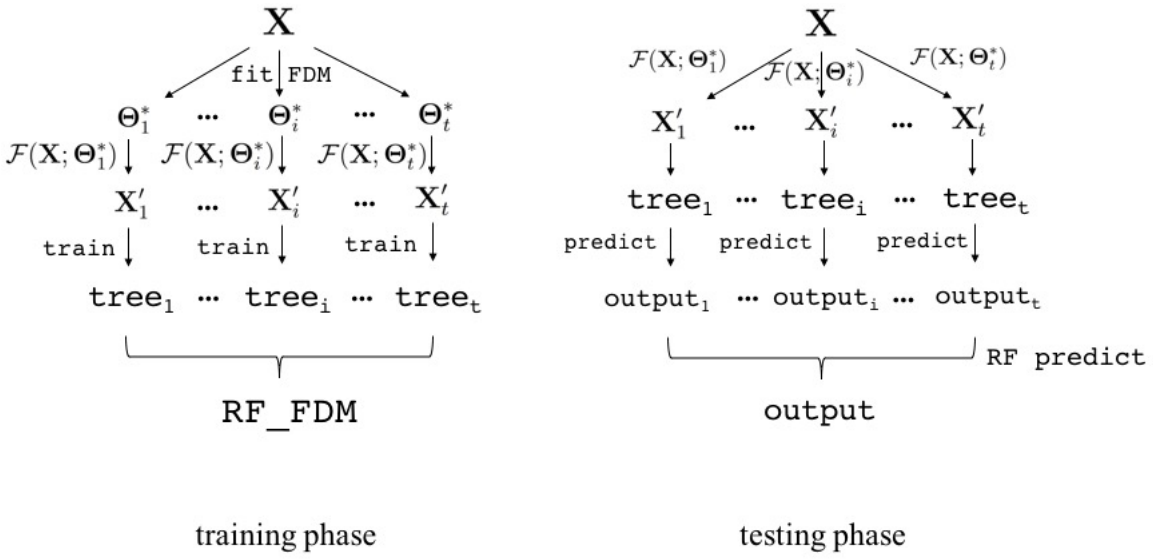


Figure 6.2. A schematic diagram of the RF_FDM approach.

in Figure 6.1 and 6.2.

6.5 Experimental Results

6.5.1 Artificial Datasets

We first tested FDM on the artificial datasets used in Chapter 5. The datasets are summarized in terms of MDS's in Table 6.1. An MDS of size = 1 is a marginal dependent feature. An MDS of size = 2 is a 2D XOR dataset and an MDS of size = 3 is a 3D XOR dataset. The results are averages of 100 runs. For each run, a new dataset is generated and

Table 6.1. Descriptions of the Artificial Datasets.

Dataset	# of MDS's		
	size = 1	size = 2	size = 3
i	0	1	0
ii	0	0	1
iii	1	1	0
iv	1	0	1
v	5	3	2

Table 6.2. Classification Accuracy (%) of Decision Trees Using FDM on Artificial Datasets.

FDM Function	Dataset				
	i	ii	iii	iv	v
f_I	91.78	65.50	83.50	68.60	79.25
f_{RP}	**92.97**	**86.02**	**91.67**	**85.20**	**91.45**
f_{DRL}	90.42	85.00	90.75	84.42	91.25

the three mapping functions were tested on the same dataset.

The results of using FDM on a decision tree classifier are shown in Table 6.2. Both mapping functions were tested: random projection f_{RP} and distance to random landmark f_{DRL}. The baseline to compare against is the decision tree without using FDM, or equivalently, FDM using the identity function f_I, i.e., $f_I(\mathbf{X}) = \mathbf{X}$. In our experiments, FDM used the ground truth feature information (the composition of features in terms of MDS's) to learn the mappings. The number of features evaluated at each split was set to "log2". The table shows that FDM has significantly improved decision tree performance on all datasets. As the dataset getting harder (comparing i and iii), the improvement brought by FDM is more significant. Comparing the two mapping functions under testing, RP performs better than DRL on all datasets.

The results in Table 6.2 were obtained by fixing the *max_features* parameter of decision tree at "log2". *max_features* is the number of features evaluated at each split node. If the total number of features is d, then by setting *max_features*='log2', we allow the classifier to consider $\log_2(d)$ features to select the best split. The decision tree is likely to perform better with more features evaluated at each split. To see how the performance gain from FDM

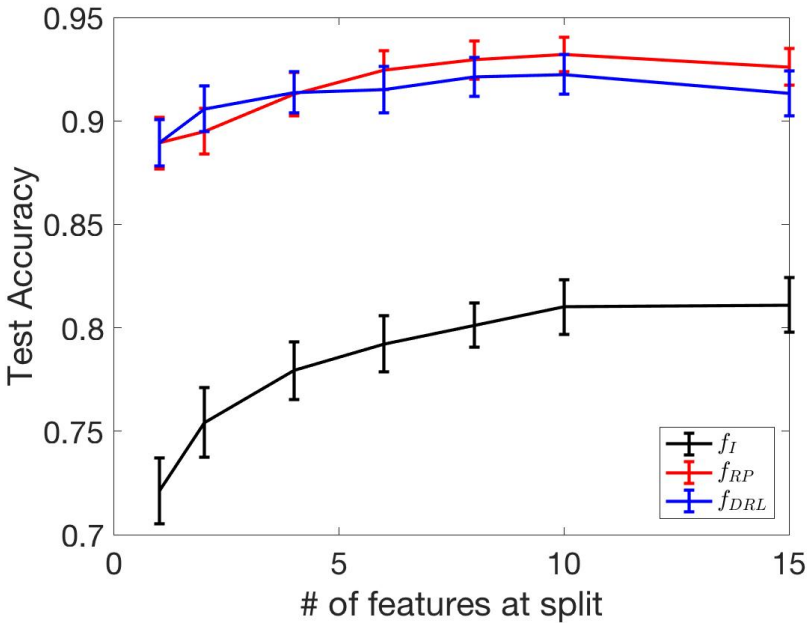


Figure 6.3. The effect of number of features used at each split on decision tree performance using the 10 MDS's dataset.

would be affected by varying this parameter, we ran experiments using different values of *max_features* on dataset v, which contains 17 features. As shown in Fig 6.3, FDM effectively improved decision tree regardless of number of features used at each split. DRL has more performance gain in the lower range of *max_features* (< 4) and RP is better in the higher range (> 4).

We then applied FDM on the random forest and observed the performance by recording test accuracy, out-of-bag accuracy, strength and correlation. In addition to the mapping functions, the two approaches of applying FDM on random forests – FDM_RF and RF_FDM – described in Section 6.4 were also tested. The *max_features* is set to "log2". Each experiment was repeated 100 times. The results are shown in Table 6.3. For test accuracy, our-of-bag accuracy and strength, the highest value of the five methods under comparison is shown in bold. For correlation, the lowest is shown in bold. We have the following observations from Table 6.3:

- The f_{RP} RF_FDM approach has the best performance on all datasets except dataset

i, similar to our observation from Table 6.2.

- All FDM approaches significantly improved strength and reduced correlation, compared with f_I. The increase in strength comes from the dependence mapping from joint dependence to marginal dependence. The decreased correlation comes from the availability of more marginal dependant features for the classifier to use.

- The RF_FDM approach performs better than FDM_RF for both mapping functions (f_{RP} and f_{DRL}). This is because by using a different mapping for each tree in the forest, we significantly reduced correlation while maintaining strength.

- f_{RP} is a better mapping function than f_{DRL} on all datasets, in both FDM approaches. It provides a better strength and a lower correlation.

Similar to LFS, FDM can be viewed as a hyper-parameter of RF, especially in the RF_FDM approach, where FDM is implemented inside of RF classifier. While LFS aims to reduce correlation, FDM aims to improve strength by transforming joint dependent features into marginal dependent features, and at the same time happens to reduce correlation because it provides more useful features thus introducing more randomness. Therefore, we tested the effect of *max_features*, which is another hyper-parameter that controls strength of RF, on the performance. The results are shown in Figure 6.4. The base line is shown in a black solid line. Red color represents the RP function and blue represents DRL. Dashed lines denote FDM_RF approaches and dotted lines denote RF_FDM approaches. As we can tell from the figure, methods with FDM significantly outperforms f_I (not using FDM) in the entire range of *max_features*, by increasing strength and reducing correlation. The strengths of all FDM based methods are similar. In terms of correlation, f_{RP}(RF_FDM) has the lowest values and is less affected by *max_features*. Our results on the artificial datasets suggest that FDM can boost RF performance as a new hyper-parameter when there exists strong joint feature dependencies.

Table 6.3. Testing Random Forests with FDM on Artificial Datasets.

	FDM Function	FDM Approach	Dataset i	ii	iii	iv	v
Test Accuracy (%)	f_I	NA	**98.02**	78.00	91.95	76.15	91.70
	f_{RP}	FDM_RF	93.97	90.62	95.60	90.07	99.65
		RF_FDM	96.92	**96.60**	**96.32**	**94.70**	**99.92**
	f_{DRL}	FDM_RF	92.95	88.90	94.08	89.15	99.27
		RF_FDM	97.12	96.02	96.70	93.05	99.88
Out-of-Bag Accuracy (%)	f_I	NA	**97.58**	75.97	92.3	75.84	90.85
	f_{RP}	FDM_RF	94.01	90.57	96.12	90.33	99.49
		RF_FDM	96.90	**96.07**	**97.25**	**94.62**	**99.84**
	f_{DRL}	FDM_RF	93.61	90.10	95.26	89.65	99.29
		RF_FDM	97.19	95.94	97.17	93.56	99.78
Strength	f_I	NA	0.789	0.250	0.627	0.348	0.539
	f_{RP}	FDM_RF	0.843	**0.685**	**0.837**	0.672	0.814
		RF_FDM	**0.857**	0.680	0.836	**0.675**	**0.816**
	f_{DRL}	FDM_RF	0.813	0.661	0.803	0.656	0.802
		RF_FDM	0.819	0.650	0.795	0.654	0.803
Correlation	f_I	NA	0.124	**0.134**	0.189	0.277	0.174
	f_{RP}	FDM_RF	0.140	0.217	0.114	0.215	0.069
		RF_FDM	**0.090**	0.146	**0.095**	**0.163**	**0.059**
	f_{DRL}	FDM_RF	0.158	0.227	0.136	0.225	0.076
		RF_FDM	0.113	0.140	0.113	0.173	0.066

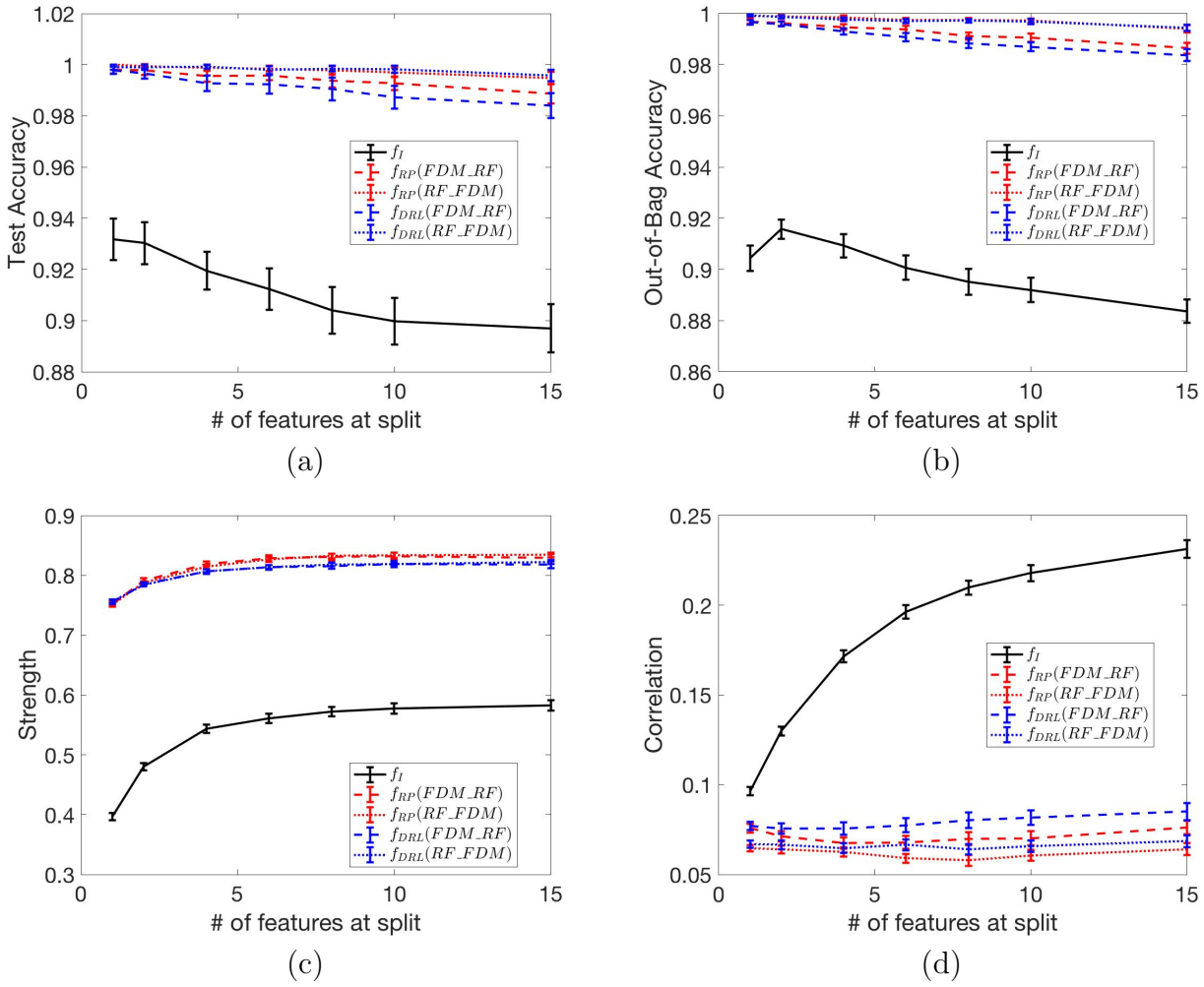


Figure 6.4. The effect of number of features used at each split on random forest performance using the 10 MDS's dataset.

6.5.2 The Mice Protein Expression Dataset

We next present our results of applying FDM on the mice protein expression dataset. Since the ground truth of joint feature dependence is not available, we used the approaches proposed in Chapter 5 to identify these joint dependence. Considering the feature set for this dataset is small, we only evaluate bottom-up MDC and bottom-up SDC, excluding top-down MDC because it identifies fewer MDS's than the bottom-up approach. Regarding FDM, both mapping functions were tested. Therefore, we have four methods under testing:

- MDS_RP: The mapping function f_{RP} is applied on each MDS;
- MDS_DRL: The mapping function f_{DRL} is applied on each MDS;
- SDS_RP: The mapping function f_{RP} is applied on each SDS;
- SDS_DRL: The mapping function f_{DRL} is applied on each SDS.

The four approaches were used to map features into a new space, and the marginal dependence of the features in the new feature space was compared with that in the original space, as shown in Figure 6.5. Due to the randomness in FDM, all FDM methods were performed 10 times. Features are sorted by p_m (p-value of a marginal dependence test) in ascending order and the median of p-values from 10 runs was used to generate the plot. As presented in Section 5.7, there are 56 features with $p_m = 0$ in the original feature space. By setting $\alpha = 0$, we let FDM approaches works on the remaining 21 features, keeping those 56 features unchanged. Thus, after FDM, there are at least 56 features with $p_m = 0$. The effectiveness of FDM approaches can be observed on the remaining features. We use black color to denote the original features. FDM approaches using MDS (SDS) to define joint dependence sets are shown in dashed (dotted) lines. FDMs using RP (DRL) as the mapping function are shown in red (blue) color. From Figure 6.5, it is clear that MDS_RP, SDS_RP and SDS_DRL significantly lowered p_m. MDS_DRL is the only one that did not work well. Comparing MDS with SDS, SDS shows better performance since it is more general than

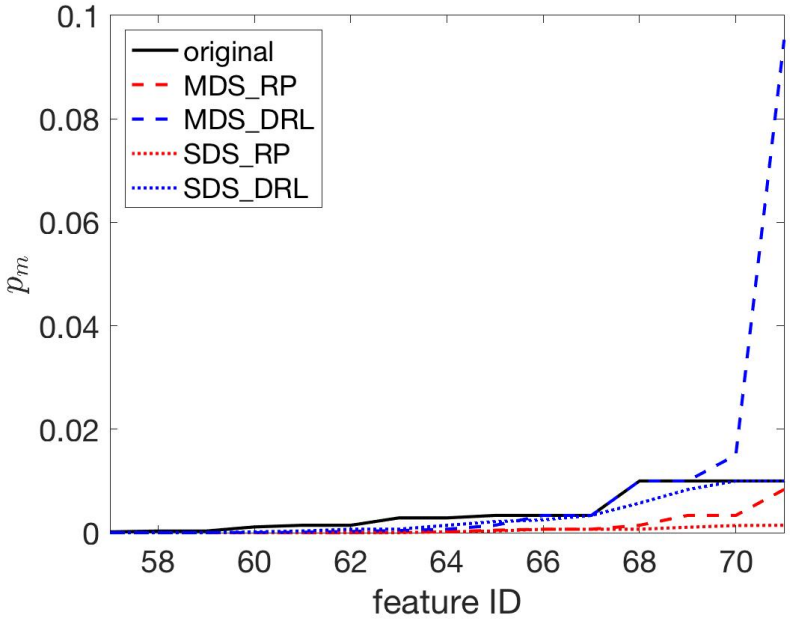


Figure 6.5. The marginal p-values (p_m) of the mice protein expression dataset in ascending order.

MDS therefore covers more joint dependence. With same joint dependence set being used, RP significantly outperforms DRL, which agrees with our results on the artificial datasets.

We next present the performance of a decision tree on this dataset using or not using FDM. Because of the huge variations in the test performance on this dataset, we repeat each experiment 300 times. The variations came from the randomness in the training process of decision tree and the randomness in the FDM process (each run used a different FDM mapping). The *max_features* parameter was set to "log2". The results are shown in Figure 6.6. The MDS_RP approach has the highest accuracy, exceeding using original features by about 0.7 % in accuracy. RP approaches outperform DRL approaches, which agrees with the results in Figure 6.5 and results obtained from the artificial datasets. Interestingly, MDS based methods have slightly higher test accuracy than SDS based methods, even though SDS based FDM provides more marginal dependent features (as shown in Figure 6.5). This may suggest that MDS's are more significant jointly dependent sets than SDS's, and their mappings are therefore more useful/meaningful marginal dependent features.

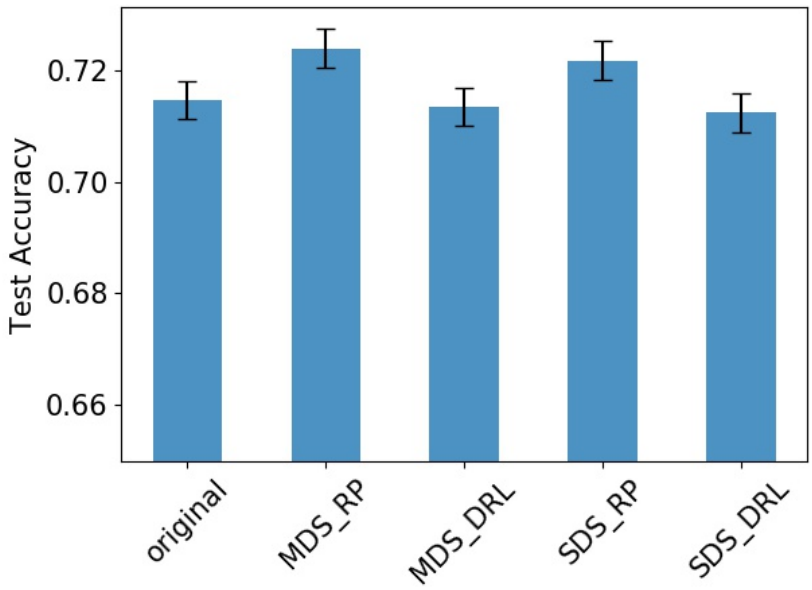


Figure 6.6. The performance of decision tree on the mice protein expression dataset with *max_features* = 'log2'.

Similar to previous experiments, we test the effect of *max_features* on the tree performance and the results are shown in Figure 6.7. All methods except SDS_DRL have some performance gain in all x-range, and more significant in the higher range. This suggests that by using FDM, we can further boost the tree performance with optimized *max_features*.

When applying FDM on random forest for a real dataset, many variants of FDM approach are available, since we have choices of MDS and SDS regarding how to define joint dependence sets; choices of RP and DRL regarding which type of mapping function to use; and choices of FDM_RF and RF_FDM regarding when to apply FDM. To reduce the number of methods under comparison, we exclude FDM_RF approaches in this experiment because RF_FDM has much better results than FDM_RF on the artificial datasets. Note that RF_FDM is more computationally expensive, but acceptable for the mice protein expression dataset.

We measured test accuracy, out-of-bag accuracy, strength and correlation of random forest with *max_features* set at different values. The results are shown in Figure 6.8. It is clear that RP based approaches perform better than DRL based approaches and original.

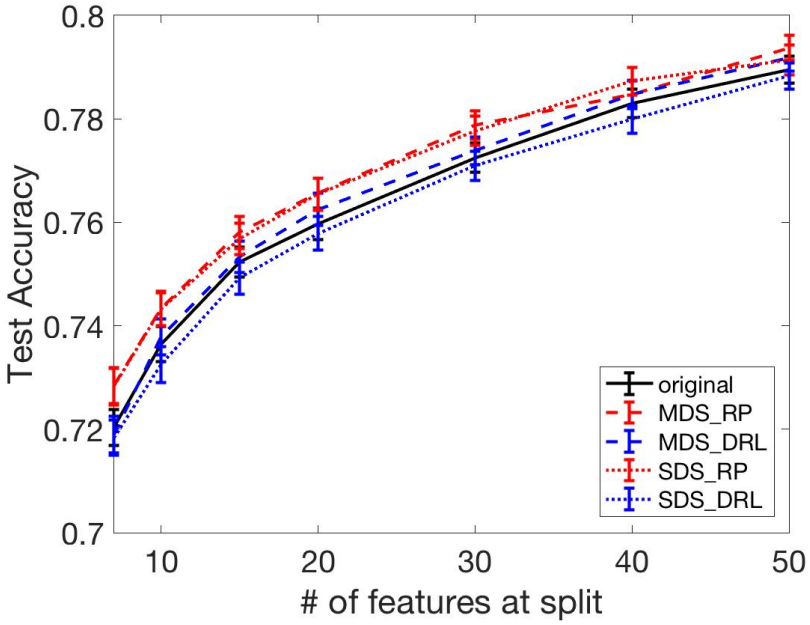


Figure 6.7. The effect of number of features used at each split on decision tree performance using the mice protein expression dataset.

MDS approaches and SDS approaches have similar out-of-bag accuracy but MDS based ones have higher test accuracy. The discrepancy between test accuracy and out-of-bag accuracy came from the number of trees used in the ensemble. For out-of-bag predictions, only about 1/3 of trees are available. Figure 6.8(c) shows that MDS_RP based approaches have higher strength than SDS based approaches (more clear for RP approaches), because MDS's have more significant jointly dependencies. Figure 6.8(d) shows that SDS based approaches have lower correlation than SDS based approaches, this is because more joint dependent sets are defined by SDS. When the number of trees is about 33 for out-of-bag prediction, the performance of MDS based approach is similar to SDS based approach (or even lower in the case of DRL), given the same mapping function. When the number of trees becomes 100 for test prediction, SDS based approaches start to show superiority, suggesting that correlation is the dominant term in determining the overall performance.

The optimal *max_features* can be obtained by maximizing the out-of-bag accuracy (validation). The performances of all methods using optimal *max_features* are summarized

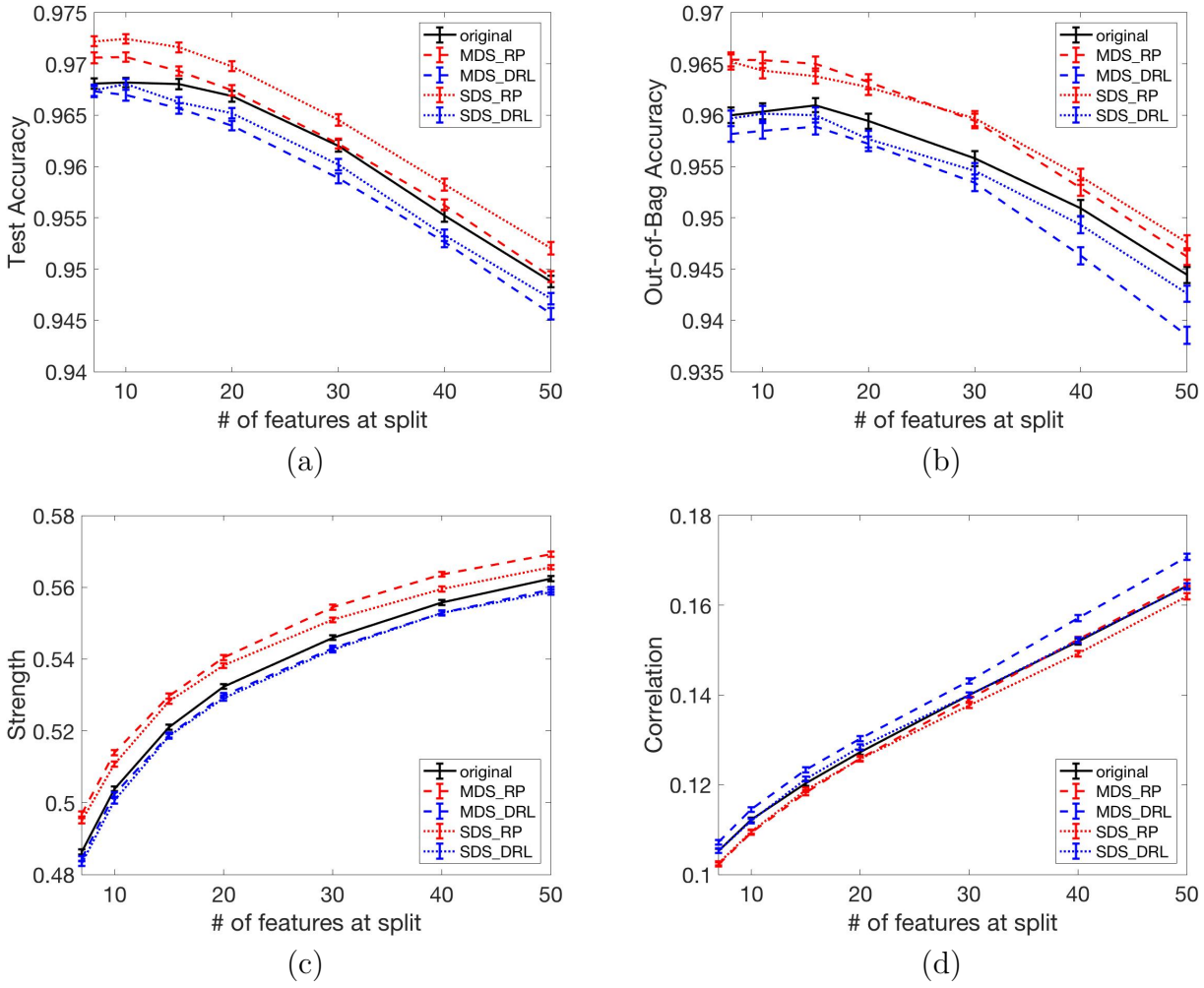


Figure 6.8. The effect of number of features used at each split on random forest performance using the mice protein expression dataset.

in Table 6.4. SDS_RP outperformed original in terms of test accuracy. When looking at the strength and correlation results, we observe that the performance gain came from the reduced correlation, rather than increased strength. This is an overall effect of using FDM and the optimization of *max_features*: as we increased strength by FDM, we were allowed to reduce correlation by using a smaller value of *max_features*. This is another good example showing the trade-off between strength and correlation.

Table 6.4. Performance of RF Using Optimal Number of Features at Each Split on the Mice Protein Dataset.

Method	Test (%)	Validation (%)	Strength	Correlation	Optimal *max_features*
original	96.80	96.10	**0.521**	0.120	15
MDS_RP	97.06	**96.54**	0.497	**0.102**	7
MDS_DRL	96.56	95.89	0.519	0.123	15
SDS_RP	**97.22**	96.52	0.495	**0.102**	7
SDS_DRL	96.74	95.97	0.483	0.105	7

6.5.3 Summary

In this section, we have presented our results of applying FDM on artificial datasets and a real dataset, the mice protein dataset. A summary of proposed FDM approaches are shown in Table 6.5. The joint dependence sets can be defined by minimum dependence sets (MDS's) or stronger dependence sets (SDS's), when the ground truth is not available. Our results show that even if the SDS approach is able to identify more joint dependence sets and provide more marginal dependent features in the mapped feature space, the MDS approach has better classification performance for decision trees because the joint dependence sets it defines are more significant. Once the joint dependence sets are defined, a mapping function with randomly generated parameters maps the joint dependent features into another feature space, and parameters that result in the most marginally dependent features are selected. Two mapping functions were evaluated: random projection (RP) and distance to random landmark (DRL). Both results from artificial datasets and the real datasets show that RP is a better mapping function than DRL. When applying FDM to random forests, two options are available: FDM applied at the forest level (the FDM_RF approach) or FDM applied at the tree level (the RF_FDM approach). The FDM_RF approach uses FDM as a data prepossessing step and is computationally efficient. The RF_FDM approach implements FDM inside of random forest, with each tree having its own FDM mapped features. This approach is more computationally expensive but provides more varieties among trees. Our results show that RF_FDM has significant better performance than FDM_RF.

Table 6.5. A Summary of Proposed Feature Dependence Mapping (FDM) Approaches

Joint Dependent Set	Mapping Function	RF Approach
Minimum Dependence Set (MDS)	Random Projection (RP)	FDM_RF
Stronger Dependence Set (SDS)	Distance to Random Landmark (DRL)	RF_FDM

To conclude, we have the following claims from our experiment:

- Either MDS and SDS has its own advantages. MDS is preferred if improving strength is more important and SDS is preferred when reducing correlation is more desired;
- f_{RP} is a better mapping function than f_{DRL};
- When integration with random forest, applying FDM at the tree level to generate a different feature mapping for each tree is more beneficial, provided that computation is not an issue.

6.6 Discussions

We have presented a feature dependence mapping (FDM) approach to utilize joint dependent features for decision trees and random forests. It shows significant improvement on artificial datasets, where the joint dependencies are strong and marginal dependencies are weak. However, for real data, most features have strong marginal dependence with the label, which limits the room for improvement by FDM. In the case of the mice protein expression dataset, the joint dependencies exist, but marginal dependencies are still dominant (52 out of 77 features are marginally dependent with $p_m = 0$). As a consequence, the performance gain brought by FDM is much less than what we observed on the artificial datasets.

In our implementation of FDM, the best k mappings are selected from the m randomly generated ones by optimizing the objective function defined in (6.2). This objective, however, only takes the feature-label dependence into account, ignoring feature-feature dependence. For better classification performance, we expect the features to be dependent on the label,

but as less dependent as possible on each other. This motivated us to use a mortified objective to consider both feature-label dependence and feature-feature dependence. For random projection, we can select the optimal k projection vectors $\boldsymbol{r}_1^*, ..., \boldsymbol{r}_k^*$ according to

$$\boldsymbol{r}_{\boldsymbol{i}}^* = \begin{cases} \underset{\boldsymbol{r} \in \{\boldsymbol{r}_1, ... \boldsymbol{r}_m\}}{\arg\max} \ \mathrm{gCor}_n(f_{RP}(\mathbf{X}; \boldsymbol{r}), Y), & \text{for } i = 1; \\ \underset{\boldsymbol{r} \in \{\boldsymbol{r}_1, ... \boldsymbol{r}_m\}}{\arg\max} \ \mathrm{gCor}_n(f_{RP}(\mathbf{X}; \boldsymbol{r}), Y) \prod_{j=1}^{i-1} |\sin \alpha_{\boldsymbol{r}, \boldsymbol{r}_j}|, & \text{for } i > 1, \end{cases} \tag{6.3}$$

where $\alpha_{\boldsymbol{r}, \boldsymbol{r}_j}$ denotes the angle between two projection vectors $\boldsymbol{r}$ and $\boldsymbol{r}_j$. $|\sin \alpha_{\boldsymbol{r}, \boldsymbol{r}_j}|$ is 0 if $\boldsymbol{r}$ and $\boldsymbol{r}_j$ are parallel and is 1 orthogonal. By using (6.3), we aim to find projection vectors that makes the mapped feature dependent on Y as well as being orthogonal to already selected vectors, thus encouraging independence between features.In the case of f_{DRL}, this can be achieved by selecting land marks that are far away from each other. Mathematically, the optimal k landmarks $\boldsymbol{x}_1^*, ..., \boldsymbol{x}_k^*$ can be obtained by

$$\boldsymbol{x}_{\boldsymbol{i}}^* = \begin{cases} \underset{\boldsymbol{x} \in \{\boldsymbol{x}_1, ... \boldsymbol{x}_m\}}{\arg\max} \ \mathrm{gCor}_n(f_{DRL}(\mathbf{X}; \boldsymbol{x}), Y), & \text{for } i = 1; \\ \underset{\boldsymbol{x} \in \{\boldsymbol{x}_1, ... \boldsymbol{x}_m\}}{\arg\max} \ \mathrm{gCor}_n(f_{RP}(\mathbf{X}; \boldsymbol{x}), Y) \sum_{j=1}^{i-1} |\boldsymbol{x} - \boldsymbol{x}_j|, & \text{for } i > 1. \end{cases} \tag{6.4}$$

We tested using (6.3) and (6.4) as the objective to select optimal mappings, but did not observe a better performance. The reason might be the number of selected mappings k is small (2 or 3 defined by the size of MDS or SDS), resulting in less dependency between selected mappings even with (6.2) as the objective.

The LFS approach proposed in Chapter 4 aims to reduce correlation of random forest, and FDM proposed in this Chapter aims to improve strength. Therefore, it is appealing to apply both at the same time. The integration of the two is straight forward on the implementation side, by using FDM as preprocessing step and let RF_LFS work on the transformed data. However, since either LFS and FDM has its own limitations, we failed to find a good application where both LFS and FDM can be effectively applied.

FDM can also be applied for regression problems with minor change in implementation. Since in a regression task, the response variable is numerical, any dependence test performed between feature and label should use dCor instead of gCor as the test statistic. All other implementations can be kept the same.

In spited of the limitation of FDM on improving classification performance, the concept of MDS and SDS in finding joint dependent feature sets has potential impact on feature representation learning. It provides us another angle to interpret the features. It can be used to explore the unknown interactions between genes, proteins, molecules, etc.

6.7 Related Work

Our FDM using random projection was inspired by (Breiman, 2001), in which an approach of using linear combinations of features to split the data was proposed. In this approach, L features are randomly selected from all features and the linear combination of these L features are used as a new feature. The coefficients of the L features are uniform random numbers on [-1, 1]. F such linear combinations are evaluated to decide the best split. However, this approach is different from ours in two aspects. First, the linear combination is applied on a random selected L features, which may not have a (stronger) joint dependence. Second, they did not perform any selection on the randomly generated linear projections. These two reasons explain why they fail to observe significant improvement on the performance.

The Gini correlation (gCor) proposed in (Dang et al., 2018) provides us a dependence measure between a set of features and the label, which made our implementation of FDM possible. It is also this work that encouraged us to deeply think about the joint dependence between features and the label and how this joint dependence can affect a classifier's performance. The distance correlation (dCor) presented in (Székely et al., 2007; Székely and Rizzo, 2009) can be similarly applied when the response variable is numeric, for instance, in the case of a regression task.

6.8 Summary

In this chapter, we proposed a feature dependence mapping (FDM) approach to map joint dependent features to another feature space where they are marginally dependent, therefore can be utilized by tree classifiers. Many variants of FDM are available, due to the choices of using MDS or SDS to define joint dependent feature sets, the choices of mapping functions – random projection (RP) or distance to a random landmark (DRL) – and the choices of applying FDM at the forest level or tree level, when integrated with random forest. Our results showed that both MDS and SDS are effective in identify joint dependencies, RP is a better mapping function than DRL, and applying FDM at the tree level is more beneficial than using FDM as data preprocessing step before trainning random forest, because by generating a different mapping for each tree, more varieties can be introduced. The improvements on artificial datasets are much more significant than that on the mice protein expression dataset, because marginal dependencies are dominant for this real dataset and make the room for improvement limited.

CHAPTER 7

CONCLUSION

Random forests haven been widely applied in many classification tasks because of its high prediction accuracy, good model interpretability and fast training process. As the computation power increases dramatically recent years, people tend to favor more accurate model like deep neural networks, even if they are black-box models. However, in some domains, especially medical fields, the interpretability of a model is extremely important. This motivated us to improve the prediction performance of random forest while maintaining its interpretability.

In Chapter 1, we explained that the performance of random forest is affected by the strength and correlation. In later chapters, we illustrated our two attempts to improve random forests. The first one is to reduce correlation by taking advantage of local information of features. We proposed a local feature sampling (LFS) approach to let each tree in the forest to focus on a "local region" of the data, thus reducing correlation among trees. For image data, where the pixels are used as features, the locations of the features are given by their coordinates on the image. For non-image data, we proposed ways to define the distance between features to learning the local information from the data. Our experimental results have shown significant improvement of using LFS on datasets where features are highly correlated. The second attempt is to increase strength by improving tree performance, which is achieved by utilizing jointly dependent features. To identify this joint dependence, we proposed the concepts of minimum dependent set (MDS) and stronger dependence set (SDS) and algorithms of finding MDS's and SDS's. We then illustrated that these jointly dependent features can be transformed to marginally dependent ones by a feature dependence mapping (FDM) approach. Two mapping functions were compared: random projection (RP) and distance to

a random landmark (DRL) and RP showed better performance. Our experiments show that FDM is effective when the data has significant joint dependence in features. The source code to repeat this work is available at https://github.com/zhangsilu17/Improving_random_forest.

Both LFS and FDM have some limitations. LFS is effective when features are structured and highly correlated. FDM is effective when joint dependencies exist and feature-label dependence is not dominated by marginal dependencies. In spite of their limitations, these two approaches have provided us new aspects of representation learning: the structure of features learned by LFS and the interaction between features learned by FDM. We call for a better understanding of the features and their behaviors in affecting classification performance while seeking for a higher prediction accuracy.

BIBLIOGRAPHY

BIBLIOGRAPHY

Amit, Y., and D. Geman (1997), Shape quantization and recognition with randomized trees, *Neural computation*, 9(7), 1545–1588.

Anguita, D., A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz (2013), A public domain dataset for human activity recognition using smartphones., in *Esann*.

Barandiaran, I. (1998), The random subspace method for constructing decision forests, *IEEE transactions on pattern analysis and machine intelligence*, 20(8).

Belgiu, M., and L. Drăguţ (2016), Random forest in remote sensing: A review of applications and future directions, *ISPRS Journal of Photogrammetry and Remote Sensing*, 114, 24–31.

Bingham, E., and H. Mannila (2001), Random projection in dimensionality reduction: applications to image and text data, in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 245–250, ACM.

Bosch, A., A. Zisserman, and X. Munoz (2007), Image classification using random forests and ferns, in *2007 IEEE 11th international conference on computer vision*, pp. 1–8, Ieee.

Breiman, L. (1996), Bagging predictors, *Machine learning*, 24(2), 123–140.

Breiman, L. (2001), Random forests, *Machine learning*, 45(1), 5–32.

Breiman, L. (2017), *Classification and regression trees*, Routledge.

Dang, X., D. Nguyen, Y. Chen, and J. Zhang (2018), A new gini correlation between quantitative and qualitative variables, *arXiv preprint arXiv:1809.09793*.

Dheeru, D., and E. Karra Taniskidou (2017), UCI machine learning repository.

Díaz-Uriarte, R., and S. A. De Andres (2006), Gene selection and classification of microarray data using random forest, *BMC bioinformatics*, 7(1), 3.

Dietterich, T. G. (2000), An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization, *Machine learning*, 40(2), 139–157.

Du, P., A. Samat, B. Waske, S. Liu, and Z. Li (2015), Random forest and rotation forest for fully polarized sar image classification using polarimetric and spatial features, *ISPRS Journal of Photogrammetry and Remote Sensing*, 105, 38–53.

Fan, J., and J. Lv (2008), Sure independence screening for ultrahigh dimensional feature space, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(5), 849–911.

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

Freund, Y., R. E. Schapire, et al. (1996), Experiments with a new boosting algorithm, in *icml*, vol. 96, pp. 148–156, Citeseer.

Fu, H., Q. Zhang, and G. Qiu (2012), Random forest for image annotation, in *European Conference on Computer Vision*, pp. 86–99, Springer.

Hall, M. A. (2000), Correlation-based feature selection of discrete and numeric class machine learning.

Higuera, C., K. J. Gardiner, and K. J. Cios (2015), Self-organizing feature maps identify proteins critical to learning in a mouse model of down syndrome, *PloS one*, 10(6), e0129,126.

Huynh, T., Y. Gao, J. Kang, L. Wang, P. Zhang, J. Lian, and D. Shen (2015), Estimating ct image from mri data using structured random forest and auto-context model, *IEEE transactions on medical imaging*, 35(1), 174–183.

Johnson, W. B., and J. Lindenstrauss (1984), Extensions of lipschitz mappings into a hilbert space, *Contemporary mathematics*, 26(189-206), 1.

Jones, S., and J. M. Thornton (1996), Principles of protein-protein interactions, *Proceedings of the National Academy of Sciences*, 93(1), 13–20.

Krogh, A., and J. Vedelsby (1995), Neural network ensembles, cross validation, and active learning, in *Advances in neural information processing systems*, pp. 231–238.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998), Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, 86(11), 2278–2324.

LeCun, Y., C. Cortes, and C. Burges (2010), Mnist handwritten digit database, *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 18.

Li, R., W. Zhong, and L. Zhu (2012), Feature screening via distance correlation learning, *Journal of the American Statistical Association*, 107(499), 1129–1139.

Louppe, G. (2014), Understanding random forests: From theory to practice, *arXiv preprint arXiv:1407.7502*.

Mari, D. D., and S. Kotz (2001), *Correlation and dependence*, World Scientific.

Nguyen, C., Y. Wang, and H. N. Nguyen (2013), Random forest classifier combined with feature selection for breast cancer diagnosis and prognostic, *Journal of Biomedical Science and Engineering*, 6(05), 551.

Okun, O., and H. Priisalu (2007), Random forest for gene expression based cancer classification: overlooked issues, in *Iberian Conference on Pattern Recognition and Image Analysis*, pp. 483–490, Springer.

Quinlan, J. R. (1993), C4. 5: Programs for empirical learning.

Schezhtman, E., and S. Yitzhaki (1987), A measure of association based on gin's mean difference, *Communications in statistics-Theory and Methods*, 16(1), 207–231.

Statnikov, A., L. Wang, and C. F. Aliferis (2008), A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification, *BMC bioinformatics*, 9(1), 319.

Su, Y., F. Jelinek, and S. Khudanpur (2007), Large-scale random forest language models for speech recognition, in *Eighth Annual Conference of the International Speech Communication Association*.

Székely, G. J., and M. L. Rizzo (2009), Brownian distance covariance, *The annals of applied statistics*, pp. 1236–1265.

Székely, G. J., and M. L. Rizzo (2013), The distance correlation t-test of independence in high dimension, *Journal of Multivariate Analysis*, 117, 193–213.

Székely, G. J., M. L. Rizzo, N. K. Bakirov, et al. (2007), Measuring and testing dependence by correlation of distances, *The annals of statistics*, 35(6), 2769–2794.

Tenenbaum, J. B., V. De Silva, and J. C. Langford (2000), A global geometric framework for nonlinear dimensionality reduction, *science*, 290(5500), 2319–2323.

Torres-Sospedra, J., R. Montoliu, A. Martínez-Usó, J. P. Avariento, T. J. Arnau, M. Benedito-Bordonau, and J. Huerta (2014), Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems, in *2014 international conference on indoor positioning and indoor navigation (IPIN)*, pp. 261–270, IEEE.

Wu, B., T. Abbott, D. Fishman, W. McMurray, G. Mor, K. Stone, D. Ward, K. Williams, and H. Zhao (2003), Comparison of statistical methods for classification of ovarian cancer using mass spectrometry data, *Bioinformatics*, 19(13), 1636–1643.

Xue, J., and Y. Zhao (2008), Random forests of phonetic decision trees for acoustic modeling in conversational speech recognition, *IEEE transactions on audio, speech, and language processing*, 16(3), 519–528.

Zhang, S., X. Dang, D. Nguyen, D. Wilkins, and Y. Chen (2019), Estimating Feature-Label Dependence Using Gini Distance Statistics, *arXiv e-prints*, arXiv:1906.02171.

Zhang, S., J. Wang, K. Xu, M. M. York, Y.-y. Mo, Y. Chen, and Y. Zhou (2019), A comparative study of multiclass feature selection on rnaseq and microarray data, *International Journal of Computational Biology and Drug Design*, 12(2), 128–142.

Zhou, Z. (2012), Measuring nonlinear dependence in time-series, a distance correlation approach, *Journal of Time Series Analysis*, 33(3), 438–457.

Zhou, Z.-H., and Y. Yu (2005), Ensembling local learners through multimodal perturbation, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(4), 725–735.

VITA

Education

B.S. Bioengineering, Zhejiang Univerisity, China, 2011.

M.S. Chemical Engineering, North Carolina State University, 2011.

M.S. Computer Science, University of Mississippi, 2017

Employment

Research Assistant, North Carolina State University, 2011–2013.

Research Assistant, University of Texas at Dallas, 2013–2015.

IT Intern, Fedex Service, 2017

Teaching Assistant, University of Mississippi, 2015–2018.

Intern, St. Jude Children's Research Hospital, 2018

Graduate Instructor, University of Mississippi, 2019

Publications

Khan, M. R., Hayes, G. J., **Zhang, S.**, Dickey, M. D., & Lazzi, G. (2012). A pressure responsive fluidic microstrip open stub resonator using a liquid metal alloy. IEEE Microwave and Wireless Components Letters, 22(11), 577-579.

Zhang, S., Zang, P., Liang, Y., & Hu, W. (2014, August). Determination of protein titration curves using Si Nanograting FETs. In Nanotechnology (IEEE-NANO), 2014 IEEE 14th International Conference on (pp. 934-938). IEEE.

Zang, P., **Zhang, S.**, Liang, Y., & Hu, W. (2014, August). Noise suppression with additional reference electrode for time-dependent protein sensing tests with Si nanograting FETs. In Nanotechnology (IEEE-NANO), 2014 IEEE 14th International Conference on (pp. 930-933). IEEE.

Liang, Y., **Zhang, S.**, & Hu, W. (2015, September). Detection of base pair charges during DNA extension with Si nanowire FETs towards DNA sequencing. In Nanotechnology Materials and Devices Conference (NMDC), 2015 IEEE (pp. 1-2). IEEE.

Zhang, S., Chen, Y., & Wilkins, D. (2017, May). A Probabilistic Approach to Multiple-Instance Learning. In International Symposium on Bioinformatics Research and Applications (pp. 331-336). Springer, Cham.

Zhang, S., Mo, Y. Y., Ghoshal, T., Wilkins, D., Chen, Y., & Zhou, Y. (2017, November). Novel gene selection method for breast cancer intrinsic subtypes from two large cohort study. In Bioinformatics and Biomedicine (BIBM), 2017 IEEE International Conference on (pp. 2198-2203). IEEE.

Zhang, S., Wang, J., Ghoshal, T., Wilkins, D., Mo, Y. Y., Chen, Y., & Zhou, Y. (2018). lncRNA Gene Signatures for Prediction of Breast Cancer Intrinsic Subtypes and Prognosis. Genes, 9(2), 65.

Ghoshal, T., **Zhang, S.**, Dang, X., Wilkins, D., & Chen, Y. (2019, April). Improving Performance of Convolutional Neural Networks via Feature Embedding. In Proceedings of the 2019 ACM Southeast Conference (pp. 31-38). ACM.

Zhang, S., Wang, J., Xu, K., York, M. M., Mo, Y. Y., Chen, Y., & Zhou, Y. (2019). A comparative study of multiclass feature selection on RNAseq and microarray data. International Journal of Computational Biology and Drug Design, 12(2), 128-142.

Zhang, S., X. Dang, D. Nguyen, D. Wilkins, and Y. Chen (2019), Estimating Feature-Label Dependence Using Gini Distance Statistics, arXiv e-prints, arXiv:1906.02171.