IMAGE CLASSIFICATION UTILIZING TREES ENSEMBLE INDUCED REPRESENTATION TECHNIQUE

PRESENTED BY: FOONG JOAN TACK
SUPERVISED BY: Dr. ADELINA TANG LAI TOH

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE
MASTER of SCIENCE in COMPUTER SCIENCE
(BY RESEARCH)
at
SUNWAY UNIVERSITY
and
LANCASTER UNIVERSITY
OCTOBER 2014

Lancaster University

Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

I cannot show more gratitude to my thesis supervisor, Dr. Adelina Tang, for her trust on me to direct my own research freely. It means a lot to me as an inexperienced researcher. On the other hand, I owe a lot to the Python, scikit-learn and, in general, the open source community. They provide extensive toolkits, libraries, and, more importantly, selfless support to the newcomers. These supports has proved to be invaluable, without which my research would be exponentially more difficult.

Last but not least, I would like to thank the Ministry of Higher Education Malaysia for funding this research under the Skim Geran Penyelidikan Eksploratori (ERGS) project number ERGS/1/2012/STG07/02/1.

Abstract

Representing data with the appropriate features is the first step toward any successful machine learning task. Crafting representation is laborious manual work for domain experts who can understand the most important features. This thesis proposes to learn a representation directly from the data in order to mitigate the effects of data becoming too complicated and complex for domain experts to come out with useful features. To this end, random forests are used to learn the desired representation. The random forests has been used successfully in both classification and regression. However, in both cases, only a portion of information derived from the trees is used. In the case of classification, only the majority vote from the terminal nodes is considered whereas in regression, only the average of terminal nodes is used. The main idea of this research is to exploit the additional structure and information from the decision trees in the random forests to learn appropriate data representation for better image classification.

Chapter 1

INTRODUCTION

> Fundamental progress has to do with the reinterpretation of basic ideas.
> - *Alfred North Whitehead*

Basic ideas are the foundation on which novel methods, innovative approaches, and sexy applications rest. The more basic the idea, the more powerful it is. For the basic ideas are the building blocks that pervade our thinking. Representation is one such ubiquitous idea in the field of machine learning. It is the purpose of this thesis to exploit this idea to the fullest.

1.1 Subject Matter and Motivation

The subject matter of this thesis is the adaptive data representation induced by random forests. This thesis are going deep into the representation's properties, working mechanism, and intuitive interpretation. The discussion culminated in the explanation in chapter 6 of a curious fact: the representation of a dataset A can be learned from a possibly unrelated dataset B; yet produce comparable, sometimes even superior, results in a classification task.

Data representation is a fundamental question in machine learning. It has been revisited and attacked by various approaches over years. Kernel methods, one of the most successful learning techniques, can be viewed as constructing an implicit representation in the Reproducing Kernel Hilbert Space. The effort of learning explicit representation is rekindled by the recent success of deep learning. It is hard to interpret the representations produced by both methods, let alone understanding what exactly make them good representations. The only justification of using such representations is their performance. The black-box approach makes constructing representation a matter of trial-and-error. The main motivation of this thesis is to present a way to learn representation that can be understood easily in the sense

that there is a clear mental model that enables the reasoning of the representation. The thorough understanding of the principles behind the method enables us to adapt it in novel situations and hence make it more powerful.

1.2 Structure of The Paper

The first section of the literature review is devoted to the essence of kernel methods, which is the way it sidesteps the representation problem by creating another equally difficult problem - that is to find a suitable kernel function. The second part of the literature review focuses on deep learning. The basic idea of deep learning is simple and elegant but the technicalities are overwhelming. The last section of the literature review is about the development of the random forests from being a classifier to a representation learning technique.

Chapter 3 Methodology discusses the details of how the random forests can be used to learn an adaptive representation. A visualisation of the representation is shown in this chapter. After that, Chapter 4 Experiments introduces the datasets used in the experiments and give some visualisation to indicate the complexity of the datasets. Experiments are then set up to explore the properties of the representation.

Chapter 5 Results and Analysis compares the results of using the method in this thesis and that of using kernel methods and deep learning. Some observations about the properties of the representation are given here, and explanations of those observations are given as well. At the end of that chapter, a geometrical explanation of how the method works is given.

Chapter 6 introduces a new concepts called *vicinity* to explain the the working mechanism of the induced representation. A quick summary and future direction of the research are then given in the last chapter.

1.3 Contributions and Publications

The central underlying theme of this thesis is that intuitive understanding improves performance. The contributions of this thesis consist of:

1. Showing that the random forests induced representation can be learned from data different from the original data we are interested in and hence adding a new range of possibilities to improve the results.

2. Introducing the new concept of *vicinity* to explain how the induced representation works. A way to visualise the representation is also introduced.

During the course of this thesis, the following publication is presented at the First International Conference on Soft Computing and Data Mining (SCDM-2014).

A Qualitative Evaluation of Random Forest Feature Learning.[1] In Recent Advances on Soft Computing and Data Mining (2014) (pp. 359-368). Springer International Publishing.

The content of the paper is subsumed by the content in section 3.5 in this thesis.

Chapter 2

LITERATURE REVIEW

Sections 2.1 and 2.2, mention briefly about the importance and learning of representation. That will lead us naturally to deep learning. Section 2.3 discusses deep learning extensively as it is used to compare with the method in this thesis. So it is a good idea to keep in mind the complexity of the deep learning methods mentioned in these sections while comparing their results in chapter 5. It starts from the fundamentals of neural networks and quickly build up to the Restricted Boltzmann Machine, the autoencoder, and the Deep Belief Network.

After the discussion on deep learning, the kernel methods will be discussed in section 2.4. It is also used to compare with the method in this thesis. The kernel methods are shown to be equivalent to constructing implicit representations. After that, its limitation is discussed briefly. Then the Support Vector Machine is introduced because its kernelised version is the most commonly used kernel method and the linear version is used in the method proposed in this thesis. A brief introduction and geometrical interpretation is given without going deep into solving the optimisation problems.

In section 2.5, the history and the fundamentals of random forests is given. The famous applications of the random forests as a classifier and the recent development of using it to learn representations will be mentioned briefly. After the initial introduction, the fundamentals of the random forests will be discussed, starting from the basics of decision trees to the three randomisation schemes.

2.1 The Importance of Representation

The representation of an object can take many forms. It can be a high dimensional vector, an activation pattern in a neural network, or probabilistic distribution. Regardless, how to represent an object is one of the most fundamental questions in machine learning. A user might be interested in classifying images, processing

music, and extracting information from text, but a computer does not have direct access to the objects that interests us. A computer can only "see" the corresponding representation. Using a wrong representation is like judging a book by its cover. Imagine that the book cover is all one knows. In other words, a book is represented by its cover. How could one possibly infer how good the book is? Indeed, one might get some indication from its title, the design of the cover, the publisher, and many other minor features. It is hard, if not impossible, to judge a book by its cover, let alone the sophistication and complication involved in such a judgement. In terms of machine learning, a simple algorithm with appropriate representation beats complicated algorithms with inappropriate representations.

2.2 Learning Representation

Representation contributes enormously to the success of machine learning task because it is the input of machine learning algorithms and the only thing they see. Feature used to be hand engineered by domain experts to reflect their knowledge of the critical aspects about a particular problem. Back to our book judging example, it takes extensive training in literature to be able to judge a book like *Finnegans Wake* [1]. An average reader cannot make head or tail of the book, but a literary critic can parse the apparent Gibberish and form a mental representation in order to understand and furthermore judge the work.

Crafting an appropriate representation by hand is intrinsically difficult even in the hand of domain experts. Furthermore, there could be a new domain where experts are rare, if there is any; or the data can be so large and complicated that even an expert is not reliable. These are pressing and real problems in the era of big data.

Given that the traditional way of hand crafting representation does not scale and the importance of representation, the question of learning representation from the raw data becomes ever more relevant and indispensable. Representation learning has been a hot trend in the machine learning community. It is mainly due to the success of deep learning in traditional machine learning tasks [2] and real world application such as MAVIS (Microsoft Audio Video Indexing Service) [3]. Deep learning itself is an attempt to construct multiple layers of feature representations in such a way that higher level abstractions can be represented.

[1] A novel written by James Joyce, often referred as one of the most difficult fiction in the English literature.

2.3 Deep Learning and Representation Learning

Deep learning attempts to learn multiple layers of representation, where the higher level representation is the composition of its lower level counterparts [4]. Each layer corresponds to a different level of abstraction. Take an image of a man sitting on a chair, the lowest level of representation could be just the pixel intensities; various configurations of the pixels that represent lines and edges then constitutes the second layer; the third layer could then be the geometrical shapes composed by different lines and edges; the forth layer in turn could be the shapes of a man and a chair, which are the composition of basic geometrical shapes; the final layer could represent the spatial relationship between the man and the chair, that is the man is *sitting on* the chair. The first breakthrough of deep learning is the success of deep belief nets [5] in the MNIST [6] digit recognition problem. The state-of-the-art result was long held by the Support Vector Machine (SVM). A more recent breakthrough is achieved in the ImageNet dataset, which achieves 15.3% error rate, lower than the state-of-the-art 26.1% [2]. MAVIS(Microsoft Audio Video Indexing Service) speech system, released in 2012, is based on deep learning [3] as well. Traditional deep learning has been focusing on various type of neural network such as deep belief net [5], autoencoder [7, 8], Restricted Boltzmann Machine [9], and sparse coding [10, 11]. Since deep belief nets and stacked autoencoders are to be compared with the method in this thesis, the following sections will be devoted to explaining the basic principles of deep learning from bottom up.

2.3.1 Neural Networks

The neural network is a Turing-complete computational model [12]. It consists of a network of individual computing units, called nodes, which can have either discrete *activation states* such as 0 and 1 or continuous activation state such as any real number form 0 to 1. A node can send signal to those connected to it and thus affects their states to the extent allowed by the strength of it connections, called *weights*, to the others. Typically, a neural network consists of a few layers. The activation pattern of a layer, z_l, is given by the activation pattern of its lower layer, z_{l-1}, via the formula:

$$z_l = \alpha(W z_{l-1} + b) \tag{2.1}$$

W is the *weight matrix* whose (i, j) entry w_{ij} is the weight of the connection between node-j and node-i. α is the *activation function*, which often takes the form of a sigmoid function.

$$\alpha(x) = \frac{1}{1+e^x} \tag{2.2}$$

A neural network is usually trained by stochastic gradient descent and back-propagation is used to calculate the gradients [13]. The detail of the algorithms can be found in many good books [14, 15, 16] As a way of computing a function $h : X \rightarrow Y$, the neural network takes an input x as represented by the activation pattern of the input layer, and transforms it it the second layer as a new representation, and so on and so forth until it reaches the output layer as the final representation.

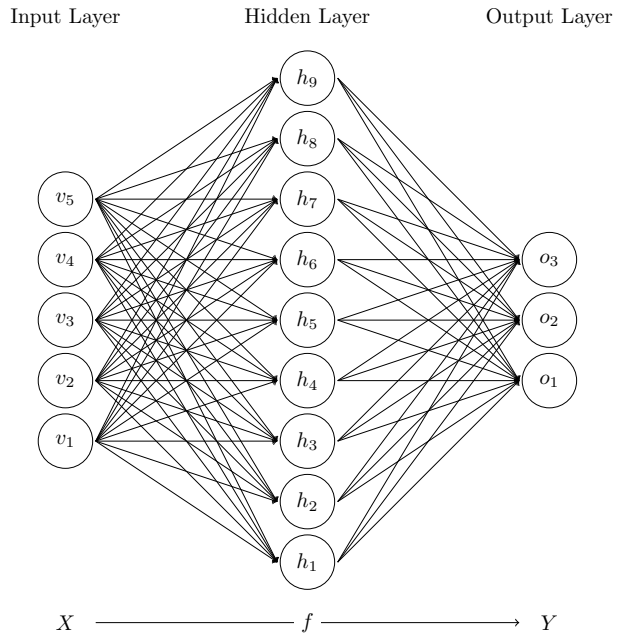


Fig. 2.1. Neural Network.

Although it is appealing to learn such multiple levels of representation [4, 17], it is notoriously difficult to train such deep architecture using back-propagation [18]. The first breakthrough come from Hinton's 2006 paper [5]. Hinton and his student found that instead of training the whole network at once, they could train the network layer by layer and then fine tune the weight through back-propagation. The most common types of network that serves as the fundamental unit of such deep architecture are Restricted Boltzmann Machine (RBM) and autoencoder. Sections 2.3.2 and 2.3.3 introduce autoencoder and RBM as well as the respective deep architectures that are composed by them.

2.3.2 Autoencoder and Stacked Autoassociator

An autoassociator is also known as an autoencoder and a Diabolo network [4, 19, 13]. Instead of learning the correct label of an input like a typical neural network, an autoassociator is trained to learn a new representation in such a way that the

original input can be reconstructed from the new representation. As shown in Fig. 2.2, the data is forced to flow through the hidden "bottle neck" layer, and is reconstructed at the output layer.

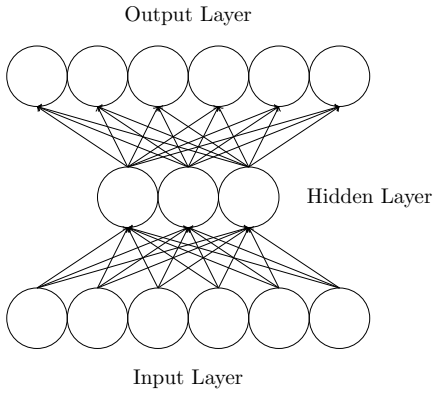


Fig. 2.2. Autoencoder.

The training of the autoencoder requires the measure of the difference between the input data and the reconstructed data, which is called *reconstruction error* (RE) here. To train the autoencoder is thus the same as minimising the reconstruction error. If the usual mean squared error is used and the hidden layer consists of linear units, then the k hidden units can be used to represent the k principal components of the data. The RE used here is the negative log-likelihood of the reconstruction, given the new representation of the hidden layer,$h(x)$.

$$RE = -log(p(x|h(x))) \tag{2.3}$$

As shown in Fig.2.3, after a single layer of autoassociator is trained, the output units can be removed and the hidden units can then be treated as the input data of another layer of autoassociator. The same process can be iterated as many times as required. Finally, a supervised layer can be added at the top of the whole network so that the whole network can be fine tuned according to the supervised criterion by back-propagation.

2.3.3 Restricted Boltzmann Machine and Deep Belief Network

Restricted Boltzmann Machine

Restricted Boltzmann Machine (RBM) is a kind of stochastic neural network in the sense that its activation function does not determine its node's states but only

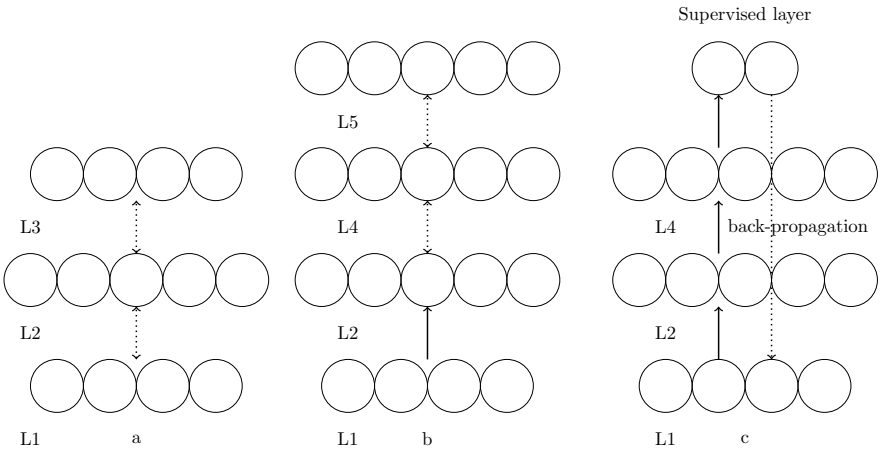


Fig. 2.3. Construction of stacked autoassociator.

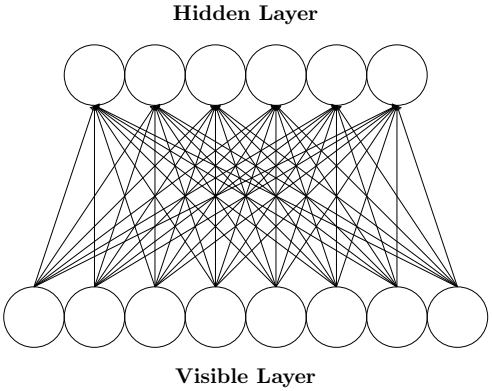


Fig. 2.4. Restricted Boltzmann Machine.

the probability of it being active. It first appeared in [20] as *harmonion*. As shown in Fig. 2.4, it consists of a layer of visible units and another layer of hidden units. Each visible unit is connected fully to all hidden units but there is no connection among the same type of units.

Training RBM as Maximising the Likelihood Function

The goal of the RBM is to find the probability distribution that best describes the data by maximising the *likelihood function*. The details on training an RBM can be found in these papers [21, 22, 23]. Given a dataset $D = \{x_1, x_2, ..., x_n\}$, and a family of probability distribution that is specified by a set of parameters $\Theta = \{\theta_1, \theta_2, ..., \theta_m\}$ the goal is to find the parameters which make the values in the dataset the most probable values of the distribution. Formally, the likelihood function of a sample x under the condition of certain parameter θ is defined as

$$L(\theta|x) = p(x|\theta) \tag{2.4}$$

$L(\theta|x)$ measures the probability of observing x given the parameter θ. Thus, the probability of observing the whole dataset, can be measured by multiplying all the likelihood functions for every sample in the dataset. The likelihood function of observing the whole dataset D is given by

$$\begin{aligned} L(\theta|D) &= \prod_x L(\theta|x) \\ &= \prod_x p(x|\theta) \end{aligned} \tag{2.5}$$

The goal of learning is to maximize the likelihood function $L(\theta|D)$, which is the same as minimising the negative log-likelihood function

$$-log(L(\theta|D)) = -\sum_x log(p(x|\theta)) \tag{2.6}$$

In the case of RBM where each sample x consists of hidden units h and visible units v, it is only required to maximize the likelihood of the activation pattern for the visible units. So given the joint distribution of the visible units and the hidden units, $p(x) = p(v,h)$, only marginalised probability $p(v) = \sum_h p(v,h)$ is interesting because instead of maximising $\prod_x p(x|\theta)$ as in equation 2.5, equation 2.7 is to be maximized.

$$\prod_v p(v|\theta) = \prod_v \sum_h p(v,h) \tag{2.7}$$

In general, it is not possible to minimize the function analytically, one must resort to approximation technique such as stochastic gradient descent (SGD). In its simplest form, SGD finds the minimum value of a function,$f : R^n \to R$, by first choose a random point, $x_0 \in R^n$, and then iteratively updates its value via the rule $x_t = x_{t-1} - \eta \nabla f(x_t)$. The idea is that the function decreases the most along the direction $\nabla f(x_t)$ at the point x_t and after sufficient number of iterations, the function will settle down at its minimum value. In the case of RBM, the possibility of

it being in a state is given by the equations

$$p(v,h) = \frac{1}{Z} exp(-E(v,h)) \tag{2.8a}$$

$$-E(v,h) = \sum_i \sum_j w_{ij} h_i v_j + \sum b_j v_j + \sum c_i h_i \tag{2.8b}$$

$$Z = \sum_{v,j} exp(-E(v,h)) \tag{2.8c}$$

v and h represent the states of the visible and hidden units respectively. b_i and c_j are the bias units corresponding to the visible unit v_i and hidden unit h_j. To train an RBM is the same as to maximize its likelihood function, given by substituting equation (2.8a) into equation 2.6

$$\begin{aligned} log(L(\Theta|v)) &= log(p(v|\Theta)) \\ &= log(\frac{1}{Z} \sum_h exp(-E(v,h))) \\ &= log(\sum_h exp(-E(v,h))) - log(\sum_{v,h} exp(-E(v,h))) \end{aligned} \tag{2.9}$$

To maximize the log-likelihood function using gradient descent, one must first calculate its gradient

$$\begin{aligned} \frac{\partial log(L(\theta|v))}{\partial \theta} &= \frac{\partial}{\partial \theta} log(\sum_h exp(-E(v,h))) - \frac{\partial}{\partial \theta} log(\sum_{v,h} exp(-E(v,h))) \\ &= -\frac{1}{\sum_h exp(-E(v,h))} \sum_h exp(-E(v,h)) \frac{\partial E(v,h)}{\partial \theta} \\ &\quad + \frac{1}{\sum_{v,h} exp(-E(v,h))} \sum_{v,h} exp(-E(v,h)) \frac{\partial E(v,h)}{\partial \theta} \\ &= -\sum_h p(h|v) \frac{\partial E(v,h)}{\partial \theta} + \sum_{v,h} p(v,h) \frac{\partial E(v,h)}{\partial \theta} \\ &= -E_{p(h|v)} \left[\frac{\partial E(v,h)}{\partial \theta} \right] + E_{p(v,h)} \left[\frac{\partial E(v,h)}{\partial \theta} \right] \end{aligned} \tag{2.10}$$

In the light of equation (2.8b), the $\frac{\partial E(v,h)}{\partial \theta}$ is easy to calculate. The parameters here are just the weights of the connections w_{ij}, the bias units for the visible units b_i

and the bias units for the hidden units c_j.

$$\frac{\partial E(v,h)}{\partial w_{ij}} = h_i v_i \tag{2.11}$$

$$\frac{\partial E(v,h)}{\partial b_j} = v_j \tag{2.12}$$

$$\frac{\partial E(v,h)}{\partial c_i} = h_i \tag{2.13}$$

Thus the gradient given by equation 2.10 can be calculated efficiently using Markov Chain Monte Carlo (MCMC) should $p(h|v)$ and $p(v,h)$ can be sampled efficiently. However, in the case of RBM $p(h|v)$ can be calculated analytically

$$\begin{aligned} p(h|v) &= \frac{p(v,h)}{p(v)} \\ &= \frac{-exp(\sum_{i,j} w_{ij} h_i v_j - \sum_j b_j v_j - \sum_i c_i h_i)}{\sum_h -exp(\sum_{i,j} w_{ij} h_i v_j - \sum_j b_j v_j - \sum_i c_i h_i)} \\ &= \frac{\prod_i exp(c_i h_i + \sum_j w_{ij} h_i v_j)}{\prod_i \sum_h exp(c_i h_i + \sum_j w_{ij} h_i v_j)} \\ &= \prod_i p(h_i|v) \end{aligned} \tag{2.14}$$

If the units in the RBM are binary, that is h_i, v_j can only be either 0 or 1, then $p(h_i|v)$ takes on a particular simple form

$$\begin{aligned} p(h_i = 1|v) &= \frac{exp(c_i + \sum_j W_{ij} x_j)}{1 + exp(c_i + \sum_j W_{ij} x_j)} \\ &= sigm((c_i + \sum_j W_{ij} x_j) \end{aligned} \tag{2.15}$$

Notice that equation 2.15 is just the activation function for the hidden unit h_i. Now what left to be done is to sample from $p(h,v)$ efficiently. And this can be done by using contrastive divergence [23], which is just an approximation of the normal Gibb sampling.

Deep Belief Network as stacked RBM

After the RBM has been trained, it can be used as a normal feed-forward network. Given an input at the visible layer v, the activation pattern $p(h|v)$ of the hidden layer can be calculated using equation 2.15. As shown in Fig.2.5, the hidden layer can then be used as the visible layer to train another level of RBM. The same

procedure can be repeated as many times as required. Note that the training of RBM is unsupervised. In the case of supervised learning such as classification, a supervised layer can be added to the top of the DBN as in Fig.2.5(c). Now the whole network can be trained as normal multilayer neural network using back-propagation. In fact, unsupervised learning is used to find a suitable set of initial weights for the network so that it will not be stuck in local minima.

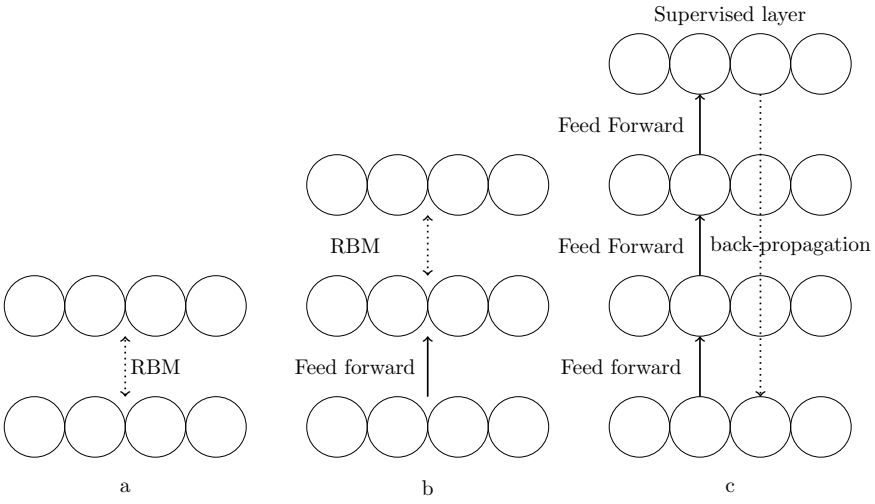


Fig. 2.5. Construction of Deep Belief Network.

2.4 Kernel Methods as Implicit Representation

Data representation determines the performance of machine learning methods [17], to such a large extent that much of the effort has been poured into the data preprocessing, data transformation, and, in general, molding the data into a suitable representation for effective machine learning. Kernel methods [24] sidestep the problem of finding representation. It relies on the machine learning algorithms that can be *kernelized*, i.e. the algorithm can be formulated solely in terms of the dot product of the data points whose explicit representation is not needed. For example, the blue points and red points in the Fig. 2.6 are not separable by straight line in the form of $f(x) = \sum a_i x_i$. However, the problem can often be solved by

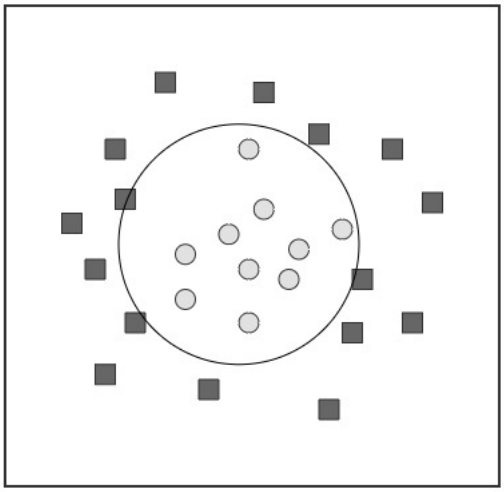


Fig. 2.6. Cannot be separated by a straight line in the form of $f(x) = \sum a_i x_i$

introducing some non-linearity via the kernel function k. Then, the curve in the form of $f(x) = \sum a_i k(x, x_i)$ can often do the job. By the Representer Theorem [24], $f(x)$ is equivalent to a straight line in the form of $f(x) =< w, \Phi(x) >$ where $< \Phi(x), \Phi(y) >= k(x, y)$. The result can be interpreted as Φ maps the points to a space where they are linearly separable. However, by using the kernel function, Φ needs not to be known explicitly. In other words, Φ induces a better representation for the data – in the sense that they are linearly separable – but no where in the learning process is it required. This is how kernel methods sidestep the representation problem.

Despite the success of kernel methods such as Support Vector Machines (SVM), their computational complexity, which typically scales cubically with respect to the number of training samples [25], prevent them from being applied to large datasets. On the other hand, it is the nature of kernel methods to need very large

numbers of instances in order to learn highly varying functions [26]. There are, nevertheless, fast methods [27, 28, 29] for huge datasets that rely on explicit representations. However, the performance issue is not the only problem, nor the most critical one for there are a lot of research [30, 31, 25, 32] geared towards speeding up the learning process. Although kernel methods allow us to map the data into a higher dimensional complicated feature space which gives us more leverage in solving the learning problem, the main problem here is that the mapping is *fixed*. It is not adaptive to the data. So either the existing kernels can do a good job or a new kernel has to be engineered. There are a lot of kernels tailored for specific purposes:

- For images, Pyramidal kernel [33] takes into account the local interaction between image patches.
- Tree kernel [34] measures the similarity between trees.
- Fisher kernel [35] measures the similarity of two objects based on a statistical model.
- Binet-Cauchy kernels [36] are used in analysing dynamic scenes.

The list can go on and on but, invariably, all of the kernels are devised with much ingenuity. Constructing a suitable kernel is almost as hard as constructing an explicit representation, if not harder. It is also noteworthy that there is a lot of research focussing on learning the kernels from the data. Some of them involve using semi-definite programming [37], matrix exponentiated gradient updates [38], combining different kernels [39], and, more exotically, defining a Reproducing Kernel Hilbert Space on the space of kernel itself [40, 41]. The lesson here is that kernel methods enjoy a lot of good properties and, at the same time, suffer from the issues of performance and flexibility. Yet, a lot of good work has been done in improving and refining the kernel methods. In this thesis, the author chose the approach of learning the explicit representation, not only because it avoids the problems and shortcomings of the kernel methods, but also it ties intimately with *deep learning*, a recent trend in the machine learning community that shows great promise.

The Support Vector Machine (SVM) is the most used kernel method and it will be compared to our method. So it is worth spending the following sections explaining the basic principles of the SVM.

2.4.1 Support Vector Machine

The Support Vector Machine (SVM) is a large margin linear binary classifier. It seeks to classify the data by using a hyperplane. In general, there is more than one

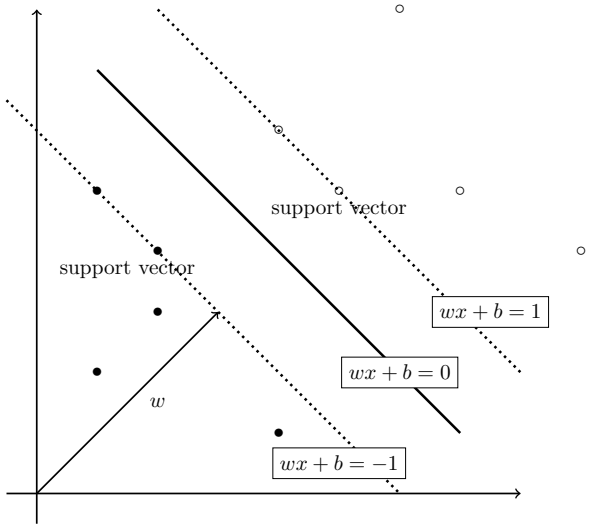


Fig. 2.7. Geometry of SVM.

hyperplane that can separate the data points. SVM chooses the one hyperplane that maximizes the *margin*. The margin is simply the distance of the closest point from the hyperplane.

Given a dataset $D = \{(x_i, y_i)_i^n\}$, with $x_i \in R^m$ and $y_i \in \{+1, -1\}$, the goal of a SVM is to find a hyperplane in the form of $wx + b = 0$ that separate the positive samples from the negative samples (see Fig.2.7). Since the distance between any point x to a hyperplane $wx + b = 0$ is given by $\frac{|wx+b|}{||w||}$, the margin is simply

$$min_x \frac{|wx+b|}{||w||}, \text{ x ranges from all the dataset} \tag{2.16}$$

Without loss of generality, the whole dataset can be scaled so that $min_x|wx+b| = 1$, and hence the margin is simply $1/||w||$. The classification problem, can be stated as finding w and b such that $wx_i + b \geq 0$ if $y_i = 1$, $wx_j + b < 0$ if $y_j = -1$, and at the same time with the largest margin. In a compact form, the problem is given by

$$\begin{aligned} &max_{w,b} \frac{1}{||w||} \\ &\text{s.t. } y_i(wx_i + b) \geq 1 \text{ for all } i \end{aligned} \tag{2.17}$$

Notice that to maximize $\frac{1}{||w||}$ is equivalent to minimize $\frac{1}{2}||w||^2$. The classifier that satisfies the above conditions is called *hard margin classifier* since the data is implicitly assumed to be linearly separable. To accommodate some noise and errors

that render the data not linearly separable, the condition is relaxed by introducing non-negative *slack variables*, ξ_i, such that $y_i(wx_i + b) \geq 1 - \xi_i$. To compensate for the introduction of these slack variables, a penalty term $\frac{C}{m}\sum_i^m \xi_i$ is added to the objective function to be minimised. C is the penalty parameter that controls the amount of error to be tolerated. Hence the general form of the problem is given by:

$$min_{w,b,\xi}\frac{1}{2}||w||^2 + \frac{C}{m}\sum_i^m \xi_i \text{ such that} \tag{2.18}$$

$$y_i(wx_i + b) \geq 1 - \xi_i \text{ and} \tag{2.19}$$

$$\xi_i > 0 \tag{2.20}$$

By using the technique of Lagrangian multiplier and some algebraic manipulation, the problem can be stated as:

$$min_\alpha \sum_{i,j} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) - \sum_1^m \alpha_i \tag{2.21}$$

$$\text{s.t. } \sum_1^m \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq \frac{C}{m}$$

The equations 2.21 is an instance of quadratic programming, which, in this case, can be solved efficiently with Sequential Minimal Optimization (SMO) algorithm [42].

2.5 Ensemble of Decision Trees

As observed in the paper [4], ensemble of trees such as boosted trees and random forests can be viewed as 3-levels deep architecture. What interests us is not the ensemble serves as a classifier, but the fact that the outputs from all the trees in the ensemble form a distributed representation [43, 44] of the training data. The exact form of the representation will be spelled out explicitly in the later part of this thesis, it suffices now to note that the representation provides a very rich description of the input data in the sense that the number of output patterns it can discriminate is exponential in the number of its parameters [45].

Despite all the good properties mentioned above, only 2 papers are dedicated to this effort [46, 47]. It is the intention of this article to further investigate the properties of this representation and its application in classification. Leo Breiman published his seminal book "Classification and Regression Trees (CART)" [48]

in 1984, in which he described the fundamental principles in using decision trees for both classification and regression and paved the way for future research. In the same year, JR Quinlan published one of the most popular tree constructing algorithms,the "C4.5" in his book "C4.5: Programs for machine learning" [49]

Ensemble methods are ways to combine various weak learners in order to get better results. One of the earliest ensemble methods is the "boosting" algorithm introduced by Schapire [50]. The idea of combining the strengths of many decision trees is not new. Amit and Geman introduced the use of random generated node tests in constructing many decision trees for handwritten digit recognition in their papers [51, 52] published in 1994 and 1997. The term "Random Decision Forests" was introduced by Ho in his paper [53], in which he used the random partition of the feature space to build the trees. In Ho's subsequent work [54], he also showed that his method yielded better results compared to both boosting and pruned C4.5-trained trees.

However, random forests[2] only began to gain serious attention after Breiman published his paper [55] in 2001. He laid down the theoretical framework for random forests and introduced a new way of constructing the decision trees by combining his earlier work in "bagging" [56] and Ho's method.

The random forests and its variant enjoy a lot of successes in the fields such as machine learning, computer vision and medical imaging [57, 58, 59, 60, 61]. One of the famous applications of the ensemble of decision trees is the detection of Higgs boson at CERN [62]. Outside of the academic setting, random forests is used in the Microsoft Kinect [63] for human pose detection. This thesis will explore feature learning using random forests.

2.6 Learning Representation Using Random Forests

It is not a new idea to use decision tree to learn a representation for the data. In the field of face recognition, decision tree has been used to learn Local Binary Pattern (LBP) for facial regions. Its performance on standard facial recognition dataset is better than that of those regular state-of-the-art feature descriptors [64].

Furthermore, the learning process of these kinds of tree-based representation is fast enough to be used to do facial alignment at 3000 Frame per Second (FPS) movie [65].

On the other hand, in the field of general object recognition, random forests is used to learn the codebook for the Bag-of-Words representation of images [46]. The Bag-of-Words model is the prototype of many images representation. Hence it is a good idea to give a brief introduction here to serve as a background knowl-

[2] random forests is the trademark of Leo Breiman

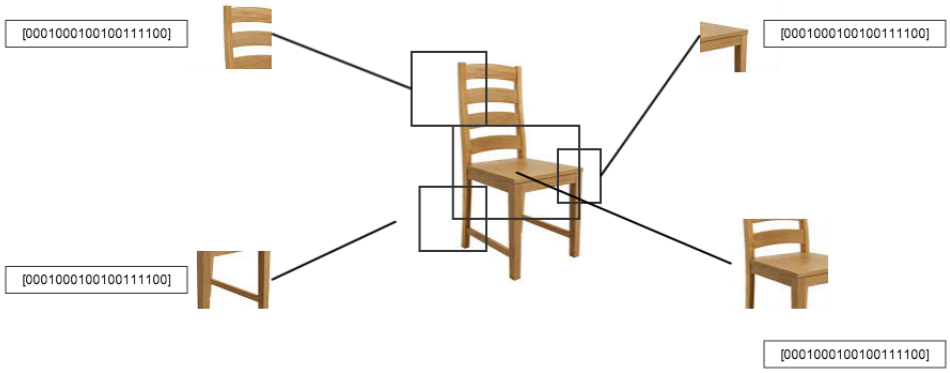


Fig. 2.8. Sampling patches and extracting local descriptors.

edge for better understanding of other models introduced here. Bag-of-Words model is originated in the field of text classification, where the text documents are modelled as a histogram of words occurred inside the documents. Following the same vein of idea, the images are represented as a histogram of "visual words" which is often called *codeword* in the literature. In general, there are 2 steps in generating the Bag-of-Words model representation:

1. Sampling patches and extracting local descriptors from images (see Fig.2.8).

2. Quantising the local descriptors using learned *codebook* (see Fig.2.9).

The *codebook* is essentially a mapping from the local descriptors to *codewords*, which furthermore ensures that similar local descriptors are mapped to one codeword. The learning of the codebook is essentially a clustering job. Instead of using traditional methods such as K-mean clustering, [46] uses Extremely Randomised Clustering trees.

Besides images data, similar tree-based method can be used in all kinds of data. In [47], the authors propose a similar method to induce tree-based representation. The authors then perform extensive empirical experiments on artificial dataset and other standard machine learning datasets such as UCI datasets [66]. At the end, they find out that the induced representation is generally better than the original.

All of the representations introduced above are binary in nature, the paper [67] introduces 2 ways to enrich the representation by considering the distance to the decision boundary.

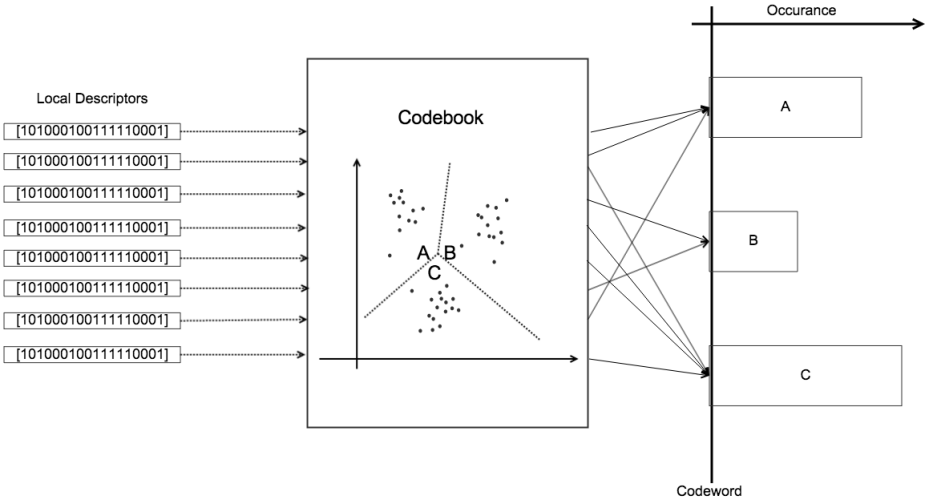


Fig. 2.9. Constructing the histogram of codeword.

2.7 Decision Trees

Decision trees can be regarded as the *partitions of the feature space*. The essence of the decision tree algorithm is to split the feature space into partitions that best organise the data and apply the same procedure to the partitions recursively, as shown in Algorithm 2.7-1. Fig. 2.10 illustrates the correspondence between the splitting of the tree and the partition of the feature space.

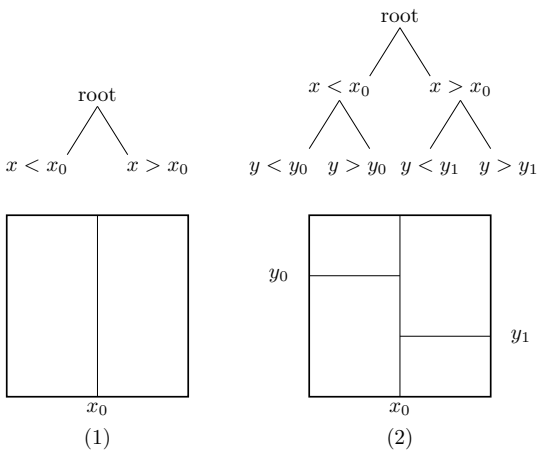


Fig. 2.10. Partition of the feature space.

Algorithm 2.7-1. Generic Tree Building

```
/*build_tree :: dataset -> tree*/
def build_tree(D):
  if stop_split?(D):
    return None
  else:
    s  = best_split(D)
  Ds = split(D,s)
  n  = node()
  attach([build_tree(D) for D in Ds],n)
  return n
```

The most important part in the algorithm 2.7-1 is the function *best_split*. All the samples in the dataset are split into 2 partitions according to whether one of their features is greater or less than a certain threshold. In Fig. 2.11, the samples have 2 features, i.e. x and y, and they are split according to whether their x-coordinate is greater or less than x_0, the threshold. In another word, a split is given by a

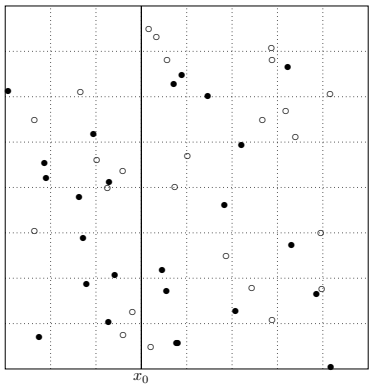


Fig. 2.11. Best split of the feature space.

feature and a threshold, as represented by the dotted lines in Fig. 2.11. To find the best split, choose one among the dotted lines according to some criteria. The best known criteria are the *gini impurity* found in the CART [48] algorithm, and the *information gain* found in the ID3 [68] and C4.5 [49] algorithms. As a split is determined by a feature and a threshold (a splitting point), all possible splits are given by

```
allSplit = [(f,p) for f in F for p in P]
```

where **F** is the set of all features and **P** is the set of all splitting point. As shown by algorithm 2.7-2, it is required to search through the set of all possible splits and find the one that maximizes the score determined by a given criterion.

Algorithm 2.7-2. Best Split

```
/*best_split :: Dataset -> bestSplit*/
def best_split(D):
  F = feature to be considered
  P = split point to be considered
  allSplit = [(f,p) for f in F for p in P]
  i = argmax[score(s,D) for s in allSplit]
  bestSplit = allSplit[i]
  return bestSplit
```

2.8 Random Forests

A random forests *RF* is simply a collection of decision trees $(T^1, T^2, ..., T^n)$ that are trained in a randomised way. Remember that training a decision tree is just like choosing the best dotted line in Fig. 2.11. The most obvious ways of introducing randomness into this process are

1. Random Subspace [54]: Random subset of all features is used in the training of decision trees.

2. Random Splitting [69]: Instead of choosing the split that minimize the entropy or gini index, random split is chosen.

3. Bagging [56]: Random subset of all training dataset is used in the training of decision trees.

A unified treatment of the three methods above will be given in chapter 3

2.9 Summary

Deep learning is trying to learn many levels of representation. The idea is that a higher level representation can capture meaningful features and thus yield better result in other machine learning tasks. However, its theory is hard to understand, and in practice it is very hard to be trained efficiently due to its many tuning parameters. For us, the main defect of deep learning is that there is no direct interpretation of the higher level representation. Higher level representation is just a vague statement without knowing exactly how the levels are built and the ways they interact.

Kernel Methods trys to sidestep the representation problem by using the kernel functions. However, there is no one kernel function to rule them all. To find a suitable kernel function is as hard as to find a suitable representation. Kernel methods do provide a new way to attack the representation problem, but there is neither a shortcut nor a universal method.

The focus of this theis is to use the random forests to learn a representation. Unlike kernel methods, it generates explicit representations, which makes it more composable with other methods. Unlike deep learning methods, one can interprete the generated representation and give an intuitive explanation on how the generated representation is adapted to different datasets. Such intuitive understanding is much valued because it forms the basis for the later rigorous understanding and provides the confidence as well as competence for the users to adapt the methods to their own requirements.

Chapter 3

METHODOLOGY

The first section establishes the notation and introduces the terminology used throughout this thesis. In the second and third section, random forests as representation transformer is explained in detail. The fourth section is about the visualisation of the new representation. The last section introduces a new classifier using the new representation.

3.1 Notation

In the field of machine learning, there are various types of data. The type of data could be images, sound, text, etc. The fact that the type of a particular data sample/point X is T is denoted by $X :: T$. In the case of supervised learning, each data point X is associated with a label Y. Depending on the situation, the type of the label could be an integer from 0 to 9 in the case of digit classification, or it could be negative and positive in the case of sentiment analysis. Similarly, $Y :: L$ denotes the fact that the type of Y is L.

A dataset is a sequence of data points and their corresponding labels, if there is any. Therefore a dataset, D, consists of data of type T and label of type L is denoted as:

$$D :: [T \times L]$$
$$D = \{(X^i, Y^i)\}_{i=1}^{n}$$

For instance, in the case of MNIST dataset introduced in section 5, T would be the type of images, L would be the type of digits from 0 to 9, X^i would be the image of some digits and Y^i would be the corresponding digit.

X^i is used to denote the actual i-th object of the dataset, such as the image in the MNIST dataset. Although conceptually it is a single object, it could be

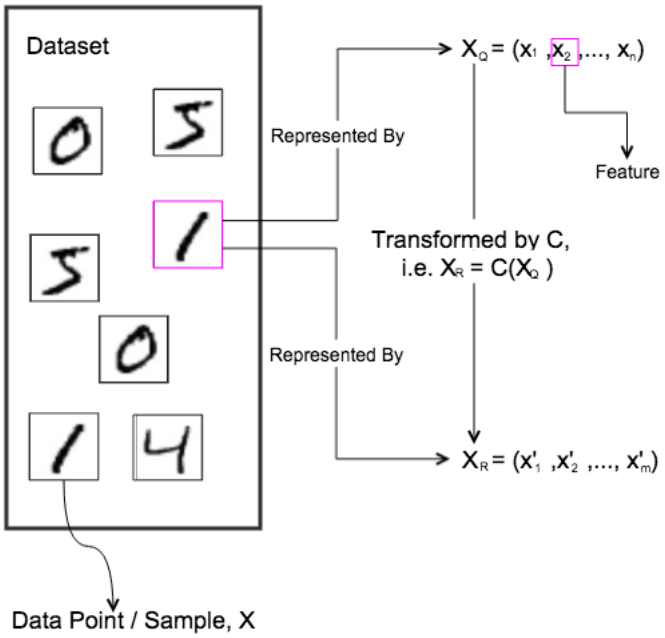


Fig. 3.1. Transformation between different representations.

represented in various ways such as pixels values, shape context, and many other features. X^i in the form of a concrete representation R is denoted by X_R^i (see Fig. 3.1). A representation R can therefore be thought as a function that takes an object of type T to its concrete representation in a feature space, F.

$$R :: T \to F$$
$$R(X) = X_R$$

The feature space F is often the product space of individual features represented as $f_1, f_2, ..., f_n$. Therefore the concrete representation is often in the form of vector.

$$\begin{aligned} X &:: T \\ R &:: T \to F \\ F &:: f_1 \times f_2 \times ... \times f_n \\ X_R &= R(X) \\ &= (x_1, x_2, ..., x_m) \text{ where } \forall i, x_i :: f_i \end{aligned}$$

In the case of MNIST dataset, the type of the data points is images. The feature used, f_i is the pixel values of the images, i.e. $f_i = \mathbb{Z}[0,255]$ where $\mathbb{Z}[0,255]$ represents the integers from 0 to 255. As the images have 20×20 pixels, the feature space is the product of 400 copies of $\mathbb{Z}[0,255]$ and thus the image is represented by a vector with 400 elements.

$$
\begin{aligned}
X :: T &= Images \\
f_i &= \mathbb{Z}[0,255] \\
F &:: f_1 \times f_2 \times ... \times f_n \\
R_{pixel} &:: Images \to F \\
X_{pixel} &= R_{pixel}(X) \\
&= (x_1, x_2, ..., x_4 00) \text{ where } \forall i, x_i :: \mathbb{Z}[0,255]
\end{aligned}
$$

Unless otherwise stated, data point X would hereafter refer to the concrete representation so the subscript R is suppressed without ambiguity.

A *classifier* is the concrete implementation of a classification algorithm like SVM, decision tree, and neural network. A classifier is often denoted by the name or the short form of the name of the algorithm it implements, for instance, SVM (Support Vector Machine), T (Decision Tree), NN (Neural Network). It can also be generically denoted as C. A classifier is determined by many parameters. Some of them are general, such as the training set used; some are specific to the algorithm used, such as the number of layers in the case of neural network. $C[p_1, p_2, ..., p_n]$ denotes classifier C instantiated with the parameters $p_1, p_2, ..., p_n$. For example, a Support Vector Machine trained with dataset D, soft margin parameter C is denoted by $SVM[D,C]$. Non-essential parameters are suppressed.

The most common usage of a classifier C is to assign a label $\hat{Y}$ to an unseen data sample X. This can be denoted as $\hat{Y} = C(X)$, which can be read as C predicts the label of X to be $\hat{Y}$. In this thesis, a classifier can also be used to transform a data sample X from a representation R to another representation Q, in which case, C is called *transformer* and the transformation is denoted as $X_Q = C(X_R)$.

The ambiguity of whether $C(X)$ denotes prediction or transformation should be easily resolved by the context since, in this thesis, the only classifier that is used to transform representation is the random forests [1].

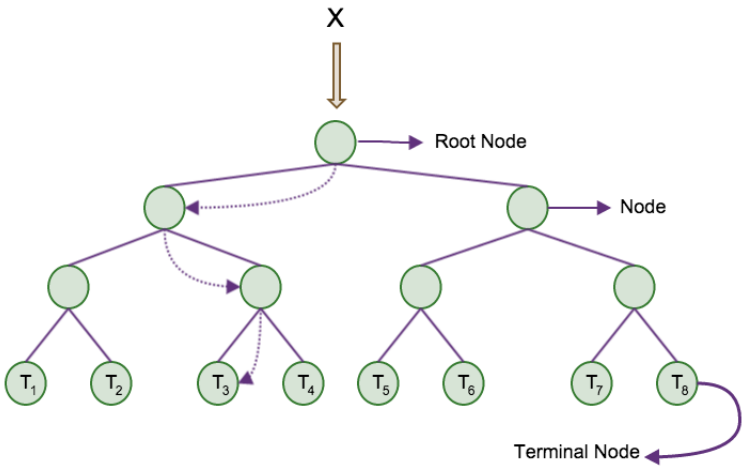


Fig. 3.2. X is transformed by the decision tree, T. $T(X) = (0,0,1,0,0,0,0,0)$

3.2 Decision Tree as Transformer

Figure 3.2 shows a typical application of a decision tree, T. Given a sample $X = (x_1, x_2, ..., x_n)$, each node, N, in the tree carries 2 pieces of information:

1. A particular feature of the sample, say, x_j.
2. A threshold, θ.

The sample X is fed into the decision tree through its root node. It will select and traverse to either left or right children of the root node depending on whether the feature x_j is greater or less than the threshold θ. It will continue traversing down the same way until it reaches the terminal node.

Each node splits the feature space into 2 sides. On one side, $x_j > \theta$; on the other, $x_j \leq \theta$. Thus decision tree a with m terminal nodes recursively splits the feature space into m regions. The interpretation of decision tree as recursive partitioning provides the foundation for giving a unified treatment of different randomisation scheme in the training of random forests in the next section.

Given a decision tree with m terminal nodes, $T = \{T_1, T_2, ..., T_m\}$, and a data point, X, a new representation of X, X_{new}, can be defined as:

$$X_{new} = T(X) = (T_1(X), T_2(X), ..., T_m(X)) \tag{3.1}$$

[1] A more verbose notation could be used to resolve the ambiguity. In the case of prediction, the notation $\hat{Y} = C.predict(X)$ could be used. In the case of transformation, the notation $X_Q = C.transform(X_R)$ could be used.

where

$$T_i(X) = \begin{cases} 1 \text{ if } X \text{ falls into } T_i \\ 0 \text{ otherwise} \end{cases} \tag{3.2}$$

For example in Fig.3.2, the data point X falls under the terminal node T_3. According to equation 3.2, $T_i(X) = 0$ for all $i \neq 3$ and $T_3(X) = 1$. Thus the new representation of X induced by the decision tree T is given by:

$$\begin{aligned} X_{new} &= T(X) \\ &= (T_1(X), T_2(X), ..., T_8(X)) \\ &= (0,0,1,0,0,0,0,0) \end{aligned}$$

3.3 Random Forests as Transformer

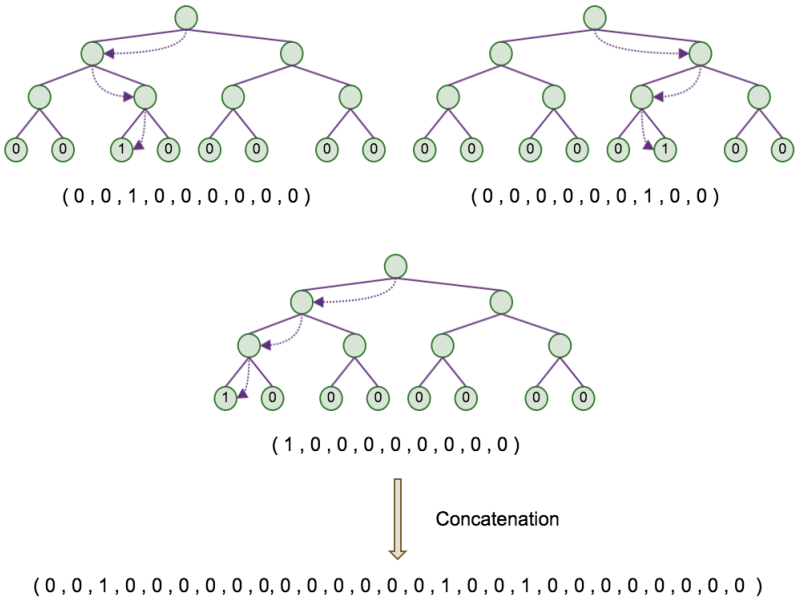


Fig. 3.3. The random forests consists of three decision trees transform a data point by concatenating the vectors transformed by individual decision tree

A random forests, RF, is just a collection of decision trees $\{T^1, T^2, ..., T^n\}$ that are trained independently and with some randomness introduced. The exact nature of the randomness will be explained in the next section. This section focuses on how the random forests can be used to induce new representation.

Knowing that a single decision tree can transform a data point into a binary vector as specified by equations 3.1 and 3.2, the most natural way for a random forests to transform a data point is simply concatenating all the vectors transformed by its constituent decision tree. Figure 3.3 shows the random forests consists of three decision trees transform the data point into a long binary vector.

Following the previous notation, the transformation can be written as:

$$\begin{aligned} RF(X) &= T^1(X) \oplus T^2(X) \oplus ... \oplus T^n(X) \\ &= (T_1^1(X), T_2^1(X), ..., T_m^1(X)) \\ &\oplus (T_1^2(X), T_2^2(X), ..., T_m^2(X)) \\ &\oplus ... \\ &\oplus (T_1^n(X), T_2^n(X), ..., T_m^n(X)) \oplus \\ &= (T_1^1(X), ..., T_m^1(X), T_1^2(X), ..., T_m^2(X), ..., T_m^n(X)) \end{aligned}$$

where $\oplus$ is defined as:

$$(x_1, x_2, ..., x_n) \oplus (y_1, y_2, ..., y_m) = (x_1, ..., x_n, y_1, ..., y_m)$$

3.4 The Training of Random Forests

The first part of this section is about the basic of decision tree as recursive partitioning of the feature space. Basic notations are established. It will lead naturally to the unified treatment of ways to introduce randomness into the training of random forests.

3.4.1 Decision Tree as Recursive Partition

The essence of section 2.7 is reiterated here, in a slightly different form, so that it can lead naturally to the training of random forests and to spare the reader from referring to previous sections repeatedly.

Given the data points of the form $X = (x_1, x_2, .., x_n)$, a partition is given by a particular feature x_i and its corresponding threshold value θ_i. The tuple (i, θ_i) is called the **splitting point**. A **splitting set** consists of a collection of splitting points. The features in the splitting set are called **candidate features**. The thresholds in the splitting set are called **candidate thresholds**.

A feature space $F = \prod F_i$ is said to be split by the splitting point $p = (i, \theta)$ into F_L and F_R, which is denoted by

$$S(F, p) = (F_L, F_R)$$

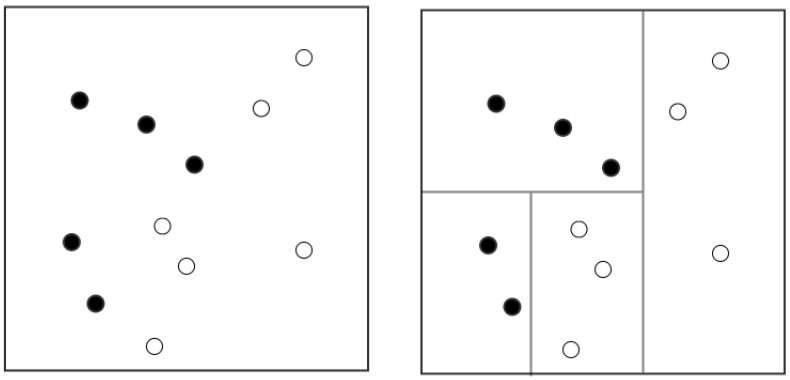


Fig. 3.4. Decision tree as recursive partition of the feature space

if only it can be written as

$$\begin{aligned} F &= F_L \cup F_R \text{ where} \\ F_L &= \{(f_1, f_2, ..., f_n) | f_i < \theta, f_{j \neq i} \in F_j\} \\ F_R &= \{(f_1, f_2, ..., f_n) | f_i \geq \theta, f_{j \neq i} \in F_j\} \end{aligned}$$

Fig. 3.4 shows the feature space is split recursively into regions in which all the data points are of the same label. The *Recursive Partition* of a feature space F is either a single region or it is a pair of left and right partitions. A region is a subspace of the feature space so they both have the same type.

$$\begin{aligned} data\ Partition = Region \mid \\ Pair\ (Left\ Partition)(Right\ Partition) \end{aligned}$$

A decision tree can be thought of as a function that takes a feature space into a partition.

$$Tree :: F \rightarrow Partition$$

$$Tree(F) = \begin{cases} F; \text{ if stopping criteria is met} \\ Pair(Left\ Tree(F_L))(Right\ Tree(F_R)); (F_L, F_R) = S(L, p_{max}) \end{cases}$$

It is now clear from the definition of $Tree$, there are 2 important factors, i.e.

1. Stopping Criteria
2. Best Splitting Point, p_{max}

Stopping Criteria

This determines when to stop the recursive process. It could one of the following:

1. A predetermined number of iterations has been reached.
2. All the data in a particular region is of the same label.
3. The number of data points in the regions is less than a fixed number.

Splitting Point

The best splitting point is the one that yields the maximum decrease of impurity. The *impurity* of a feature space measures the heterogeneity of data points in it. The impurity is at its minimum if all the data points in the feature space have the same label. *Entropy* and *gini impurity* are the two most common measures of impurity.

Let D be a dataset of the form $\{(X_i, Y_i)\}_{i=1}^{n}$, L the set of all unique label, and F the feature space, p_l the probability of finding data points of label l in the feature space, then the entropy, $E(F)$, and gini impurity, $G(F)$, can be defined as

$$E(F) = \sum_{l \in L} p_l log(p_l)$$
$$G(F) = 1 - \sum_{l \in L} p_i^2$$

After the feature space is splited into F_L and F_R, the impurity of the partition (F_L, F_R) can be defined as the average impurity of the individual subspace.

$$E(F_L, F_R) = \frac{\text{number of points in } F_L}{\text{number of points in } F} E(F_L) + \frac{\text{number of points in } F_R}{\text{number of points in } F} E(F_R)$$
$$G(F_L, F_R) = \frac{\text{number of points in } F_L}{\text{number of points in } F} G(F_L) + \frac{\text{number of points in } F_R}{\text{number of points in } F} G(F_R)$$

The decrease of impurity caused by a splitting point p, $\Delta(p)$, is given by

$$S(F, p) = (F_L, F_R)$$
$$\Delta(p) = P(F) - P(F_L, F_R); \text{ where } P \in \{E, G\}$$

Given a splitting set $\{(i, \theta_i), (j, \theta_j), ..., (k, \theta_k)\}$, the best split among them is the one that makes the maximum decrease in impurity. The canonical splitting set consists of all features and all the values of the features in the dataset.

To reiterate, the important factors influencing the training of the decision tree are:

1. Dataset
2. Stopping Criteria
3. Best Splitting Point
   (a) Splitting Set
   (b) Measure of Impurity

The next section shows how to manipulate some of these factors to train a collection of independent decision trees in a randomised way.

3.4.2 Randomisation Scheme

A random forests RF is simply a collection of decision trees $(T^1, T^2, ..., T^n)$ that are trained in a randomised way. Randomisation can be introduced into the factors dataset and splitting set.

Bagging

Bagging introduces randomness into the dataset. Given a dataset D of size N, a new dataset of size N' is formed by sampling from D uniformly and with replacement. Repeating the above procedure m times, yields m datasets of size N'. Each of the datasets can then be used in training classifier, decision tree in this case. Normally, one would choose $N' = N$, in which case approximately 63.2% of the new dataset is unique. In another word, only a subset of the original dataset is used in training (See Fig.3.5(c)). [56] gives a detailed treatment of this method.

There are 2 ways of introducing randomness into the splitting set. Let $F = \{f_1, f_2, ..., f_n\}$ be the set of candidate feature, and $\Theta(f_i)$ be the set of all candidate threshold for the feature f_i. Given a data point is of the form $X = (x_1, x_2, ..., x_n)$, F_0 is the canonical set of candidate features, which consists of the whole feature space. $\Theta_0(f_i)$ is the canonical set of candidate threshold for the feature f_i, which consists of all the value of f_i in the dataset. Randomness can be introduced by restricting the canonical candidate features and canonical candidate threshold randomly.

Random Subspace

Instead of considering the canonical candidate feature set F, a random subset of the whole feature space, $F*$, is chosen. The splitting set can be written as

$$((f, \Theta_0(f) | \text{ for } f \text{ in } F*))$$

Remember that choosing the best split is equivalent to choosing the best dotted line in Fig.2.11, then this method is equivalent to choosing the dotted line among the vertical lines or the horizontal lines (See Fig.3.5(a)). This method is called random subspace and is first introduced in [54].

Random Splitting

Usually the candidate thresholds of a particular feature f_i consists of all its values in the dataset. It is called the canonical threshold set for the feature f_i and is denoted by $\Theta_0(f_i)$. Random Splitting uses a different threshold set denoted by $\Theta_1(f_i)$. $\Theta(f_i)$ consists of a random value of the feature f_i in the dataset.

In this case, the splitting set can be written as

$$((f, \Theta_1(f) | \text{ for } f \text{ in } F))$$

F can be either the full feature space or just a random subspace.

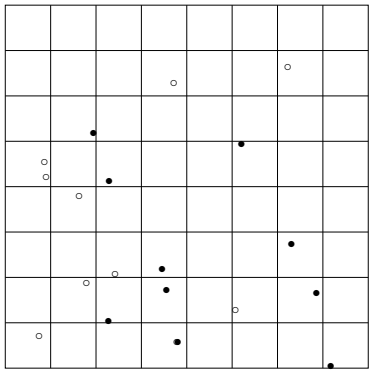
Again, this is equivalent to choosing the best line among the subsets of all dotted lines (See Fig.3.5(b)). This is the method used in [69].

3.5 Visualization of the feature

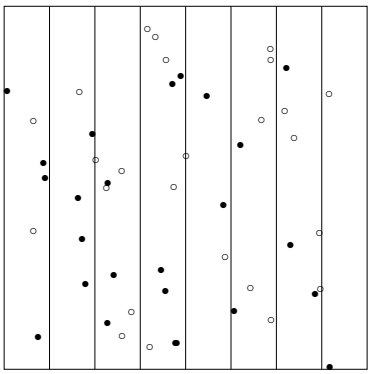
The result in this section shows the general properties of the learned representation using the MNIST dataset.

Fig. 3.6 shows the first 15 digits from the dataset. A random forests consists of 30 trees is trained using randomly generated data. To be precise, the data used here consists of 50000 vectors of dimension 784×1, drawn from random uniform distribution. There is no relationship at all with the MNIST dataset. However, it is possible to use this random forests transform the MNIST dataset into new representation. The right diagram in Fig. 3.7 shows the reconstruction of the first 15 digits using the new representation. For comparison, the left diagram in Fig. 3.7 shows the reconstruction of the same 15 digits using random forests trained using 50000 digits from the dataset. As shown by the comparison in Fig. 3.7, the reconstructed digits using the original dataset are more visually recognisable.

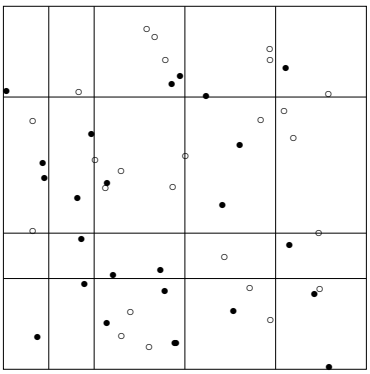
To show the individual contribution of the trees inside the random forests, Fig. 3.8 and 3.9 show the progressive reconstruction of the digit “5”. Fig. 3.8 shows



(a) Bagging.



(b) Random Subspace.



(c) Random Splitting.

Fig. 3.5. 3 ways of randomizing

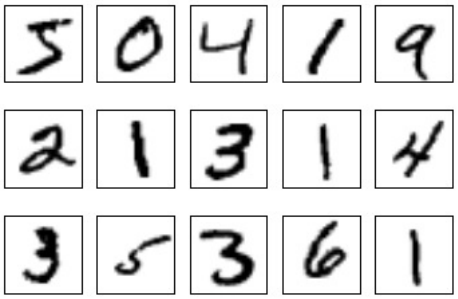


Fig. 3.6. First 15 digits from the MNIST dataset.

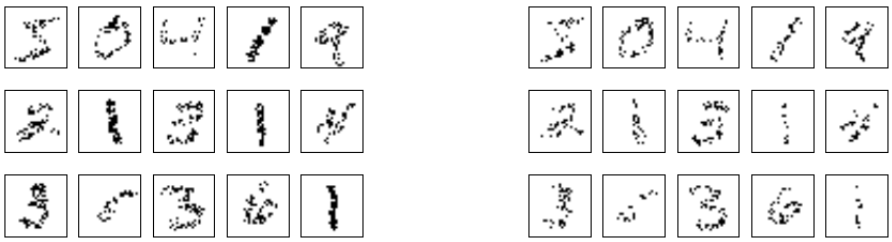


Fig. 3.7. The left diagram shows the reconstruction using the MNIST dataset, and the right reconstruction using random data.

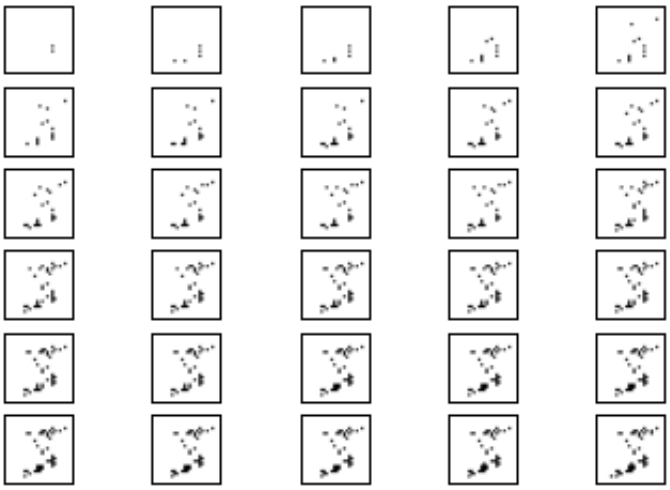


Fig. 3.8. Progressive reconstruction using random forests trained with random data.

the reconstruction using random forests trained with random data, while Fig. 3.9 shows the reconstruction using random forests trained with the original dataset. The diagram is to be read from left to right and from top down. The first image shows the reconstruction using only the first tree; the second image uses the first and second; so on and so forth until the final one uses all of the trees.

3.5.1 Reconstruction of Image

Fig. 3.10 demonstrate how the reconstruction of images from its representation works. The representation determines uniquely a terminal node in every decision tree. Each terminal node in turn determines a unique path to traverse from itself to the root. Each node in the path of traversal tells a piece of information, i.e. a feature x_i is greater or less than a threshold value θ_i. Collecting all the information from all the trees in the random forests, the original data point can be reconstructed

Fig. 3.9. Progressive reconstruction using random forests trained with original dataset.

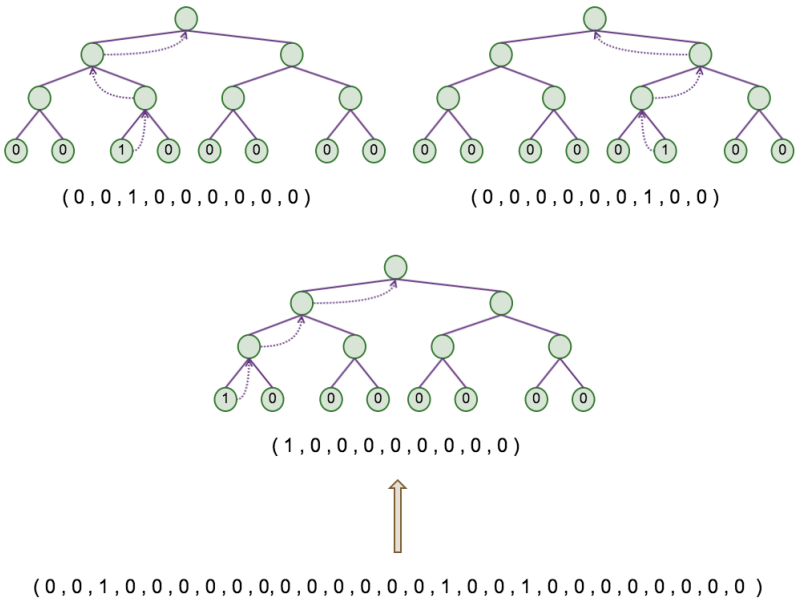


Fig. 3.10. The representation is reversed-engineered into images.

appropriately.

3.6 Naive Random Forests Classification

Instead of using another classifier on top of the newly learned representation as suggested earlier, a naive classifier that just looks at the similarity of the new representation is proposed. This method is considered naive because it ignores the intra-class variation of the sample in the sense that the sum of all the samples of the same class is used as a prototype of this class.

Although it is not as good as another state-of-the-art classifier such as SVM, it will be shown in the next section (Table 5.2) that it is consistently better than the single layer neural network. Henceforth, the method presented here shall be referred to as the **naive random forests classification (NRFC)**.

In the training phase of NRFC, a random forests $RF = (T^1, T^2, ..., T^k)$ is trained on the whole training set $\{(X_i, Y_i)\}$. All the data points $\{X_i\}$ can be transformed into $\{T(X_i)\}$ (see Fig.3.11a). For each distinct label Y in the dataset, all the samples that are labelled Y are selected, that is $L_Y = \{X_i | Y_i = Y\}$, and sum over their transformed representation, $C(Y) = \sum\{T(v) | v \in L_Y\}$. Then for each distinct label Y, there is a corresponding vector $C(Y)$ that is constructed as above (see Fig.3.11b).

Given any X in the testing set, the similarity between X and a certain class Y is defined as the number of indices such that both $T(X)_i$ and $C(Y)_i$ are not zero. Then the label of x is predicted as the Y that is most similar to X (see Fig.3.12).

3.7 Conclusion

This section introduces notation and terminology used throughout this thesis. The notations and terminologies are then used to give a precise explanation on how random forests can be used as a transformer of representation. Visualisation of the induced representation is provided for intuitive understanding. Finally, a naive classifier using the induced representation is given. Various experiments are performed in the next section. The experiments are aimed to explore the properties of the induced representation as well as compare its performance in classification task to other methods such as Deep Belief Network, Stacked Autoassociator, and Support Vector Machine.

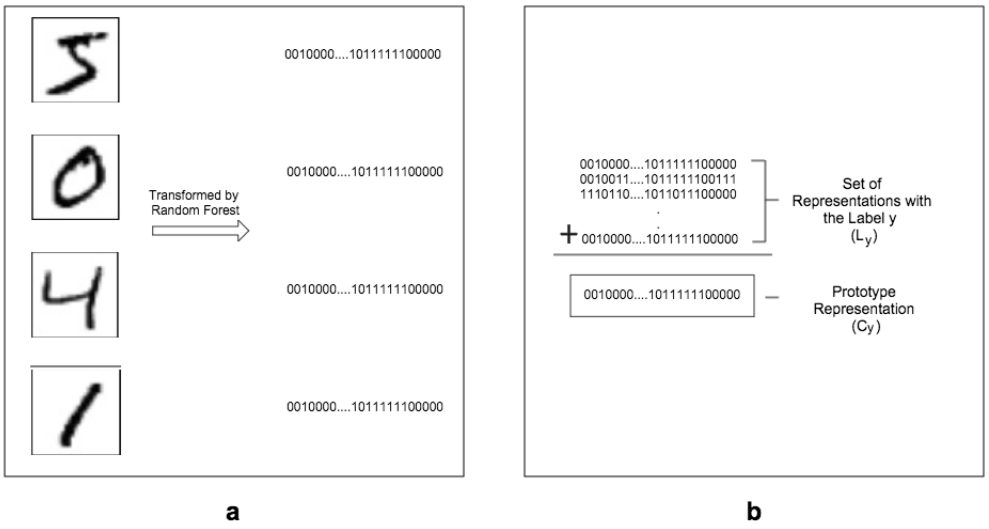


Fig. 3.11. Training of NRFC.

Testing Data, T(x)	Prototype Representations of Distinct Labels, C(y)	Similarity
0010000....1011111100000	0010000....1011111100000	10
	1110000....1010011100000	3
	0010011....1000111100000	6
	0010000....1011111100000	20
	1110000....1010011100000	9
	0010011....1000111100000	1

Fig. 3.12. How NRFC Classify Unseen Data.

Chapter 4

EXPERIMENTS

4.1 Overview of The Datasets

The eight datasets used here were presented in the paper [70]. They are constructed for the purpose of evaluating the performance of various deep learning architecture. Since the combination of random forests induced feature and linear SVM is proposed as one of the deep architectures here, it is only natural to compare its performance with that of the others.

The first four datasets are subsets of the Mixed National Institute of Standards and Technology dataset (MNIST)[71], and its various perturbed versions. The fifth dataset consists of randomly generated rectangles, for which the learning task is to differentiate between the rectangles whose length is greater than its width and those with length less than its width. The sixth dataset is essentially the same as the fifth. The only difference is that the interior of the rectangles in the sixth dataset is replaced by a random patch from a black and white image. The learning task for the last dataset is to classify its images into those that contain convex region and those that contain non-convex region.

The following sections present the basic information of the the datasets, some of their random samples, and, most interestingly, their t-SNE visualization[72]. t-SNE (t-Distributed Stochastic Neighbour Embedding), similar to PCA (Principal Component Analysis), is a dimensionality reduction technique developed specially with the aim of visualising high dimensional data in mind. Each point in the visualisation represents a sample in the dataset, the distance between the points represents their similarity, and the colour of the point represents its category. The difficulty of the learning task can be easily grasped by observing the distribution of the clusters.

Table 4.1. Overview of the datasets used.

Dataset	Abbreviation	Train/Test/Valid
MNIST Original	M	10000/2000/50000
MNIST Rotated	MR	10000/2000/50000
MNIST Rotated with Image Background	MBIR	10000/2000/50000
MNIST with Image Background	MBI	10000/2000/50000
MNIST with Random Background	MBR	10000/2000/50000
Rectangle	R	1000/200/50000
Rectangle with Image Background	RBI	10000/2000/50000
Convex	C	6000/2000/50000

4.1.1 MNIST Original

Basic Properties A subset of the original MNIST dataset[71]. Fig. 4.1 and Fig. 4.2 show some samples from the dataset and its corresponding t-SNE visualisation .

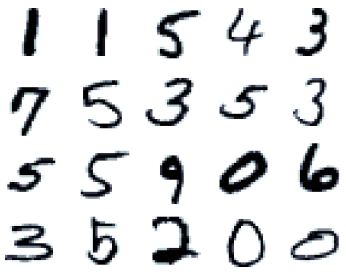


Fig. 4.1. Original MNIST.

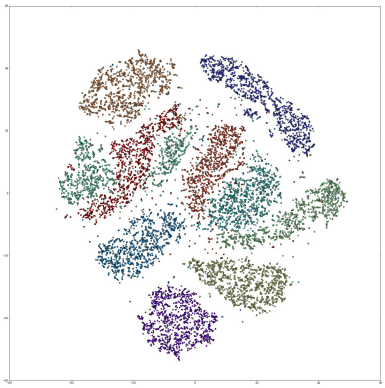


Fig. 4.2. t-SNE Visualization of Original MNIST.

4.1.2 Rotated MNIST

The digits are rotated by an angle uniformly distributed in the interval [0, 2π]. Fig. 4.3 and Fig. 4.4 show some samples from the dataset and its corresponding t-SNE visualization.

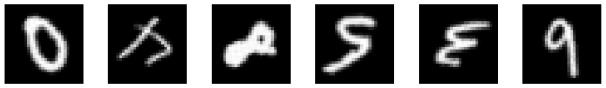


Fig. 4.3. Rotated MNIST.

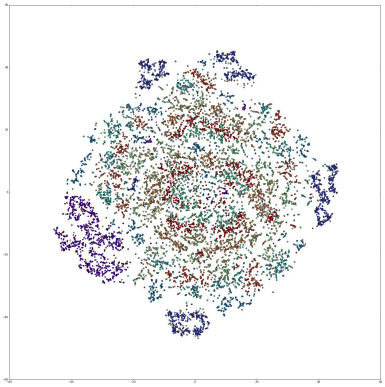


Fig. 4.4. t-SNE Visualization of Rotated MNIST.

4.1.3 Rotated MNIST + Background Images

In addition to rotating the digits, the background of the image is replaced by a random patch from a black and white image. Fig. 4.5 and Fig. 4.6 show some samples from the dataset and its corresponding t-SNE visualization.

Fig. 4.5. Rotated MNIST + Background.

4.1.4 MNIST + Background Images

Each pixel of the background is replaced by a random value uniformaly generated in the interval [0,255]. Fig. 4.7 and Fig. 4.8 show some samples from the dataset and its corresponding t-SNE visualisation.

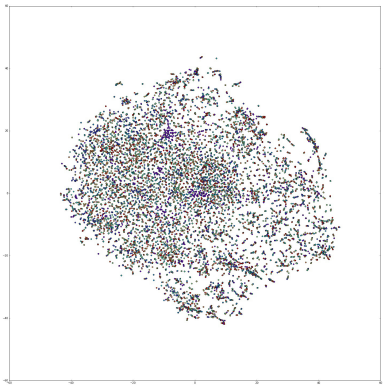


Fig. 4.6. t-SNE Visualisation of Rotated MNIST + Background.

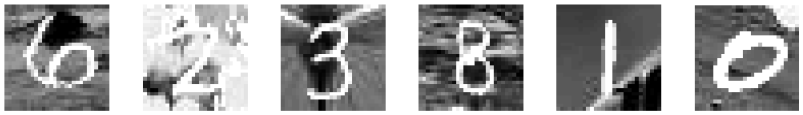


Fig. 4.7. MNIST + Background.

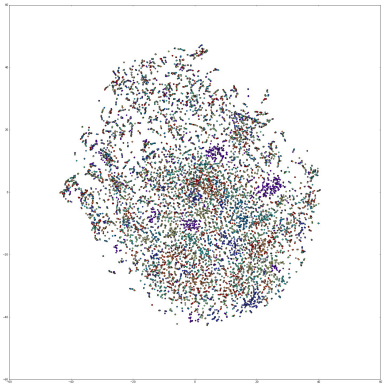


Fig. 4.8. t-SNE Visualisation of MNIST + Background.

4.1.5 MNIST + Random Background

The background of the image is replaced by a random patch from a black and white image. Fig. 4.9 and Fig. 4.10 show some samples from the dataset and its corresponding t-SNE visualisation.

4.1.6 Rectangle

Each rectangle is uniquely determined by its top left point, width and height. All of these are randomly chosen provided that the rectangle generated fits into the

Fig. 4.9. MNIST + Random Background.

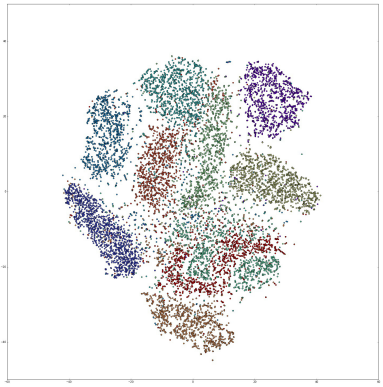


Fig. 4.10. t-SNE Visualisation of MNIST + Random Background.

size of 28x28 pixel. Fig. 4.11 and Fig. 4.12 show some samples from the dataset and its corresponding t-SNE visualisation.

Fig. 4.11. Rectangles dataset.

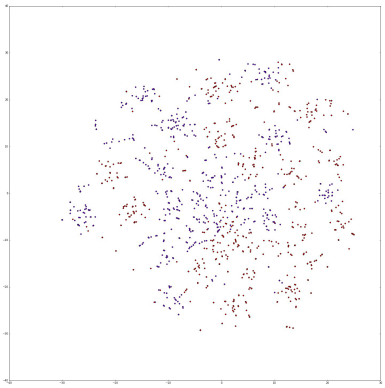


Fig. 4.12. t-SNE Visualisation of Rectangles dataset.

4.1.7 Rectangle + Images

The rectangles are generated the same way as the Rectangle dataset. The interior of the rectangle is then replaced by a random patch from a black and white image. Fig. 4.13 and Fig. 4.14 show some samples from the dataset and its corresponding t-SNE visualisation.

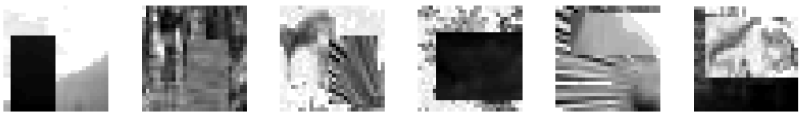


Fig. 4.13. Rectangles + Images Background.

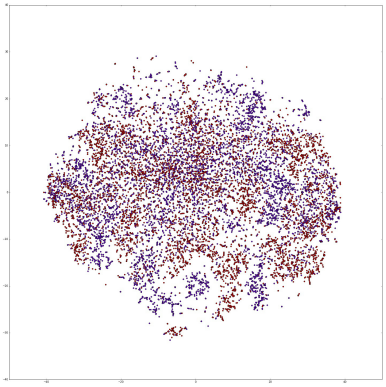


Fig. 4.14. t-SNE Visualisation of Rectangles + Images Background.

4.1.8 Convex vs. Non-convex

Convex images are constructed by intersecting randomly generated half plane. On the other hand, non-convex images are chosen among the union of convex images, which could be convex or non-convex. Fig. 4.15 and Fig. 4.16 show some samples from the dataset and its corresponding t-SNE visualisation.

Fig. 4.15. The first 3 samples are convex set(the white region) and the last 3 are not.

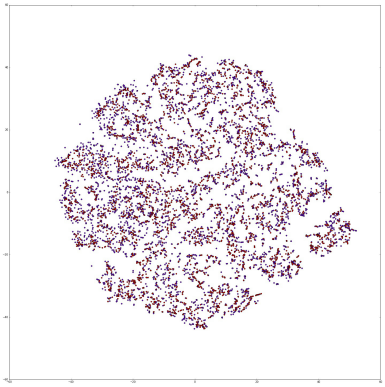


Fig. 4.16. t-SNE Visualisation of Convex dataset.

4.2 General Architecture of Experiments

A couple of experiments are performed to explore the properties of the representation induced by the random forests. Since there is no way to measure how good is a representation, its application in classification task is the focus here and classification error rate is used as a proxy of the representation's performance. To put the classification error rate into perspective, the result is going to be compared with those obtained by various deep learning methods and kernel methods.

All of the experiments contain 3 phases (see Fig.4.17 - 4.19):

1. Phase 1 is the training of the random forests. The trained random forests in this phase is used to transform the raw data into new representation. Most of the time will be spend on this phase, because this thesis focuses on the new representation generated by the learned random forests and there are a lot of parameters influencing the learning of the random forests, such as number of trees, depth of the tree, and splitting criterion. The training data used in this phase is called *representation training data* since it is used to train the random forests that generates new representation.

2. Phase 2 is the training of the classifier. The classifier is trained with data transformed by the learned random forest in phase 1. The data used here is called *classifier training data*. Classifier training data is not necessary to be the same as the representation training data in phase 1. In fact, the usage of different representation training data and classifier training data is a novelty of the method in this thesis. The representation training data here is being treated as one of the tuning parameters.

   Since the representation is the main focus here, only simple classifiers are going to be used. The first classifier simply calculating the similarity between the the testing data and the average of all the training data with the same label. The second classifier is the linear Support Vector Machine. No parameter tuning is done for the SVM so that the result can be mainly attributed to the representation.

3. Phase 3 is the actual application of the trained random forests and the trained classifier. The trained random forests first transforms the testing data before feeding it to the trained classifier.

4.3 Experiment Setup

5 factors of the random forests are to be considered, which are going to affect the classification errors. The 5 factors are

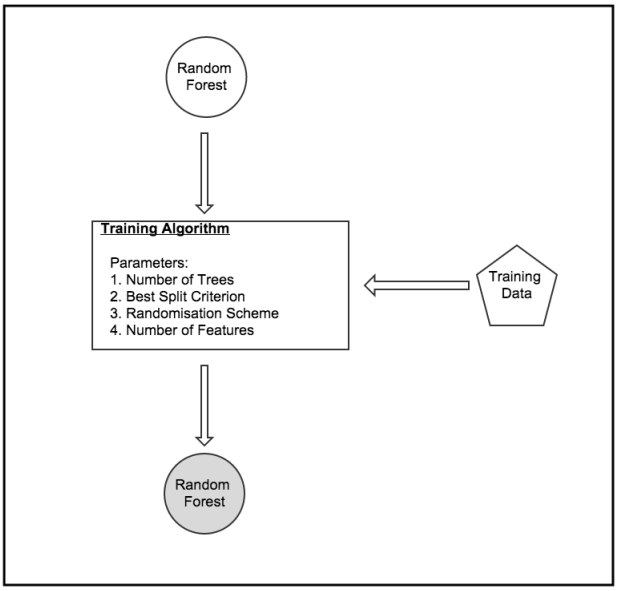


Fig. 4.17. Phase 1: Training Random Forests.

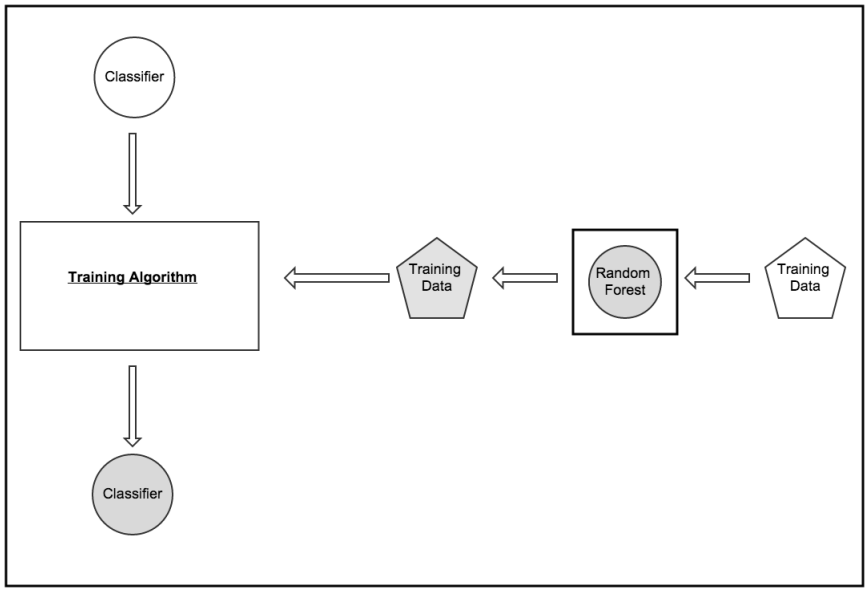


Fig. 4.18. Phase 2: Training Classifier.

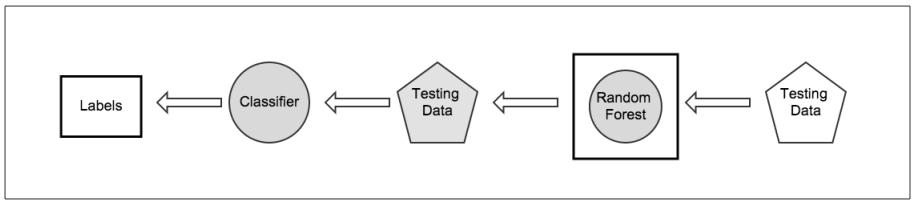


Fig. 4.19. Phase 3: Classifying in Action.

1. The number of trees inside the random forests. The numbers considered here are 100, 150, 200, 250, 300, 350, and 400.

2. The criterion used to consider the best split. Both of the most common criterion are used, namely entropy and gini index.

3. The number of features used in choosing the best split. The numbers considered here are the log2 and square root of the number of total features.

4. The training data used to train the random forests. For each of the 8 datasets used, 8 different representations are learned using 8 random forests trained on the 8 datasets individually.

5. The ways of randomisation allowed in the training of the random forests. The original random forests introduced by Breiman that does not involve random splitting are going to be compared with the extra random trees that does.

The data is split according to Table.4.1 to ensure that the experimental results are comparable to the result in [70]. And for every combination of the 5 factors above, the experiment is run from phase 1 to phase 3 for 20 times. The results are recorded, summarised, compared, and analysed in Chapter 6.

Chapter 5

RESULTS AND ANALYSIS

5.1 Results

Table 5.1 shows the algorithms used to compare with the method in this thesis, and their corresponding abbreviation. Table 5.2 shows the results using the proposed method in this thesis, compares to other methods such as support vector machine, neural network, deep belief network, and stacked autoassociator, whereas, Table 5.3 shows the ranking of each method for every dataset. The number inside the bracket is the ratio of the error produced by that particular method to the minimum error. For example, the best method will get the number 1; and the method that is twice as bad as the best method will get the number 2.

Fig.5.1 to 5.8 show the results of the experiments. Each of the figure shows the classification error for a particular dataset conditioned on the 5 factors mentioned above. A figure consists of 2 rows, which represent the different randomisation scheme used. The upper row uses the extra random trees while the lower row uses

Table 5.1. Algorithms Used for Comparison

Algorithm	Abbreviation
SVM with polynomial kernel	PSVM
SVM with Gaussian kernel	GSVM
Neural network with 1 hidden layer	Nnet
Stacked Autoassociator with 3 hidden layers	SAA-3
Deep Belief Network with 1 hidden layer	DBN-1
Deep Belief Network with 3 hidden layers	DBN-3
Random Forest encoding + linear SVM	LRFE
Random Forest encoding + Hamming distance	NRFE

Table 5.2. Error Rates using different algorithms on the datasets. The best error rate is in bold.

Algorithm	Datasets							
	M	**MBR**	**MBI**	**MR**	**MRBI**	**R**	**RI**	**C**
PSVM	03.69 ± 0.17 %	16.62 ± 0.33 %	24.01 ± 0.37 %	15.42 ± 0.32 %	56.41 ± 0.43 %	02.15 ± 0.13 %	24.05 ± 0.37 %	19.82 ± 0.35 %
GSVM	**03.03 ± 0.15** %	14.58 ± 0.31 %	22.61 ± 0.37 %	11.11 ± 0.28 %	55.18 ± 0.44 %	02.15 ± 0.13 %	24.04 ± 0.37 %	19.13 ± 0.34 %
Nnet	04.69 ± 0.19 %	20.04 ± 0.35 %	27.41 ± 0.39 %	18.11 ± 0.34 %	62.16 ± 0.43 %	07.16 ± 0.23 %	33.20 ± 0.41 %	32.25 ± 0.41 %
SAA-3	03.46 ± 0.16 %	11.28 ± 0.28 %	23.00 ± 0.37 %	**10.30 ± 0.27** %	51.93 ± 0.44 %	02.41 ± 0.13 %	24.05 ± 0.37 %	18.41 ± 0.34 %
DBN-1	03.94 ± 0.17 %	09.80 ± 0.26 %	16.15 ± 0.32 %	14.69 ± 0.31 %	52.21 ± 0.44 %	04.71 ± 0.19 %	23.69 ± 0.37 %	19.92 ± 0.35 %
DBN-3	03.11 ± 0.15 %	**06.73 ± 0.22** %	16.31 ± 0.32 %	10.30 ± 0.27 %	47.39 ± 0.44 %	02.60 ± 0.14 %	22.50 ± 0.37 %	**18.63 ± 0.34** %
***LRFE**	03.49 ± 0.16 %	11.50 ± 0.28 %	**11.77 ± 0.28** %	11.95 ± 0.28 %	**45.88 ± 0.43** %	**01.67 ± 0.11** %	**20.26 ± 0.36** %	19.70 ± 0.35 %
***NRFE**	04.67 ± 0.18 %	15.32 ± 0.31 %	14.92 ± 0.31 %	14.47 ± 0.31 %	53.42 ± 0.44 %	03.09 ± 0.15 %	27.49 ± 0.39 %	19.55 ± 0.35 %

Table 5.3. Ranking of the Error Rates.The number inside the bracket is the ratio of a particular error rate to the best error rate.

Algorithm	Datasets							
	M	**MBR**	**MBI**	**MR**	**MBIR**	**R**	**RBI**	**C**
PSVM	4(1.22)	7(2.47)	7(2.04)	6(1.50)	7(1.23)	2(1.29)	5(1.19)	6(1.08
GSVM	1(1.00)	5(2.17)	5(1.92)	2(1.08)	6(1.20)	2(1.29)	4(1.19)	3(1.04
Nnet	8(1.55)	8(2.98)	8(2.33)	7(1.76)	8(1.35)	8(4.29)	7(1.64)	8(1.75
SAA-3	3(1.14)	3(1.68)	6(1.95)	1(1.00)	3(1.13)	3(1.44)	5(1.19)	1(1.00
DBN-1	5(1.30)	2(1.46)	3(1.37)	5(1.43)	4(1.14)	7(2.82)	3(1.17)	7(1.08
DBN-3	2(1.03)	1(1.00)	4(1.38)	1(1.00)	2(1.03)	4(1.56)	2(1.11)	2(1.01
LRFE	6(1.15)	4(1.71)	1(1.00)	3(1.16)	1(1.00)	1(1.00)	1(1.00)	5(1.07
NRFE	7(1.54)	6(2.28)	2(1.27)	4(1.40)	5(1.16)	6(1.85)	6(1.36)	4(1.06

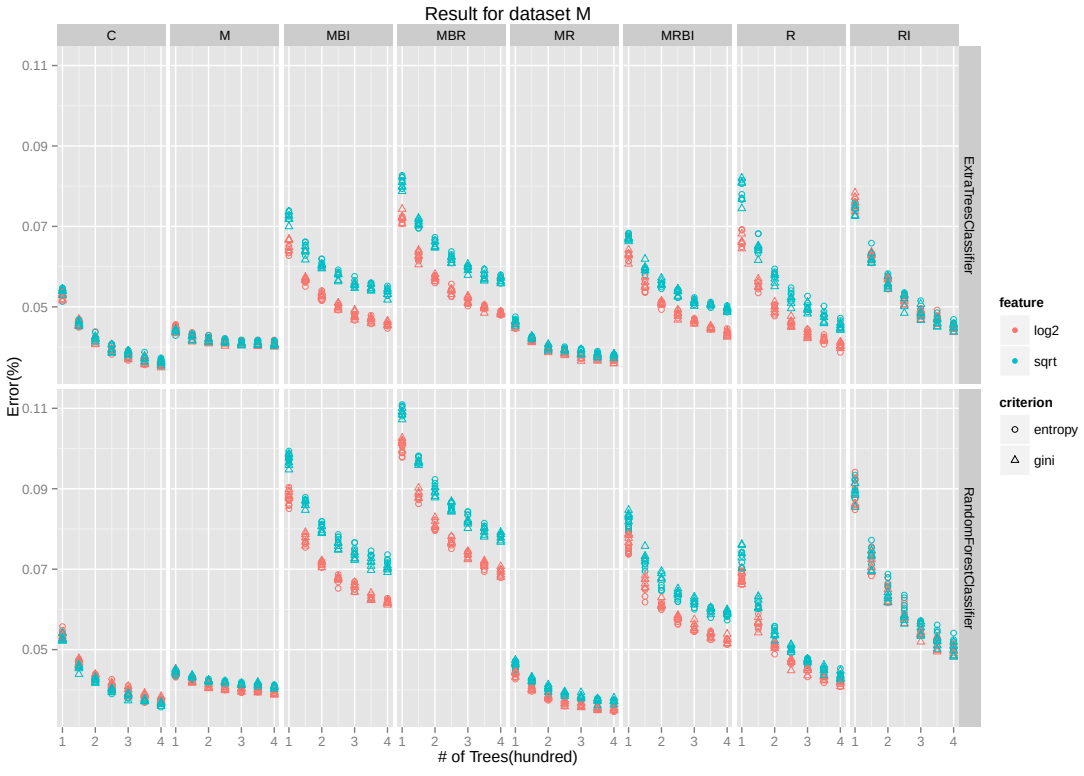


Fig. 5.1. Result for the MNIST original dataset.

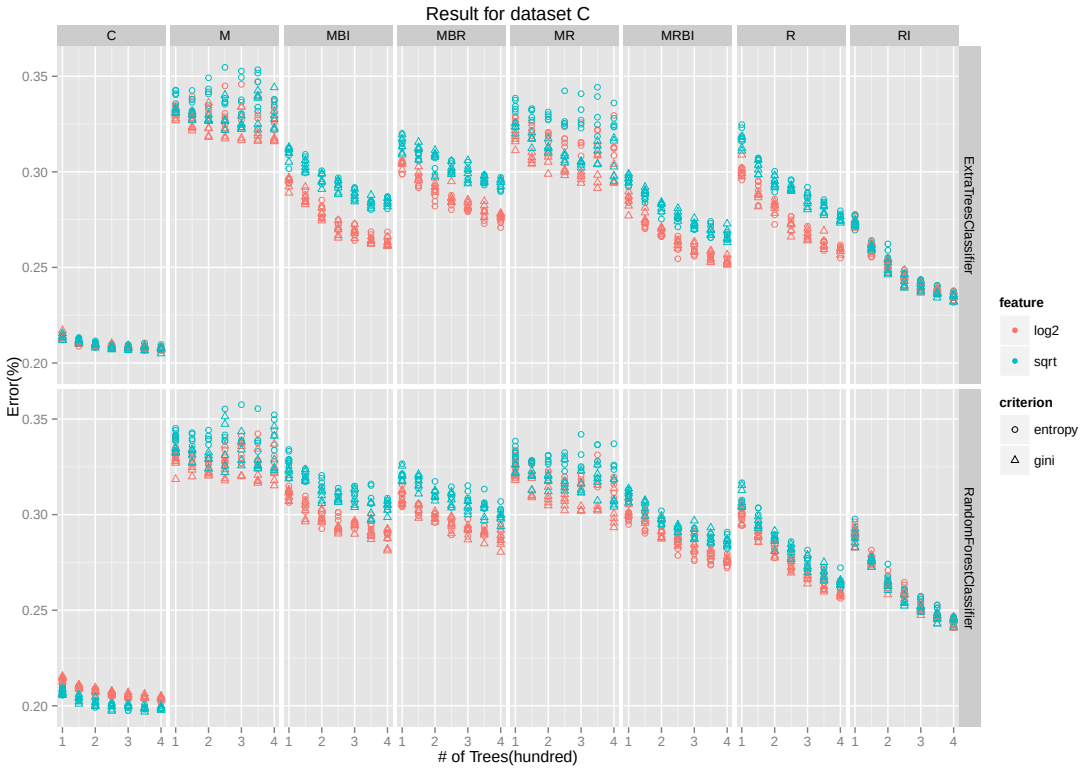


Fig. 5.2. Result for Convex dataset.

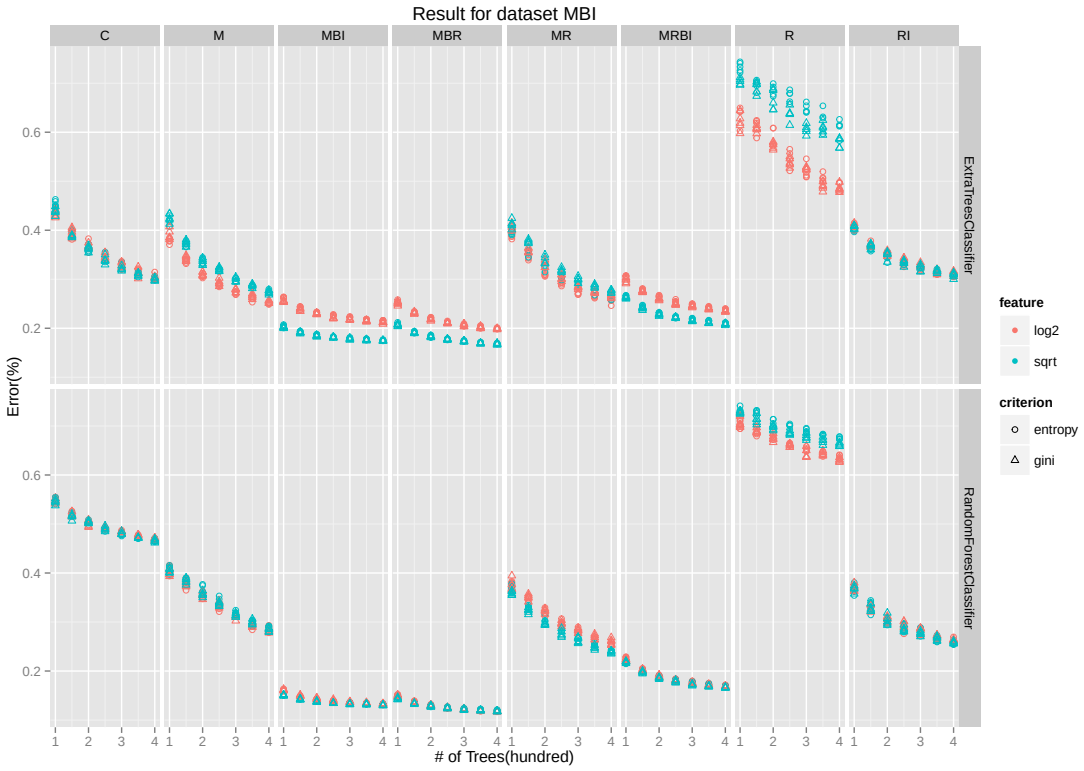


Fig. 5.3. Result for dataset MNIST with background image.

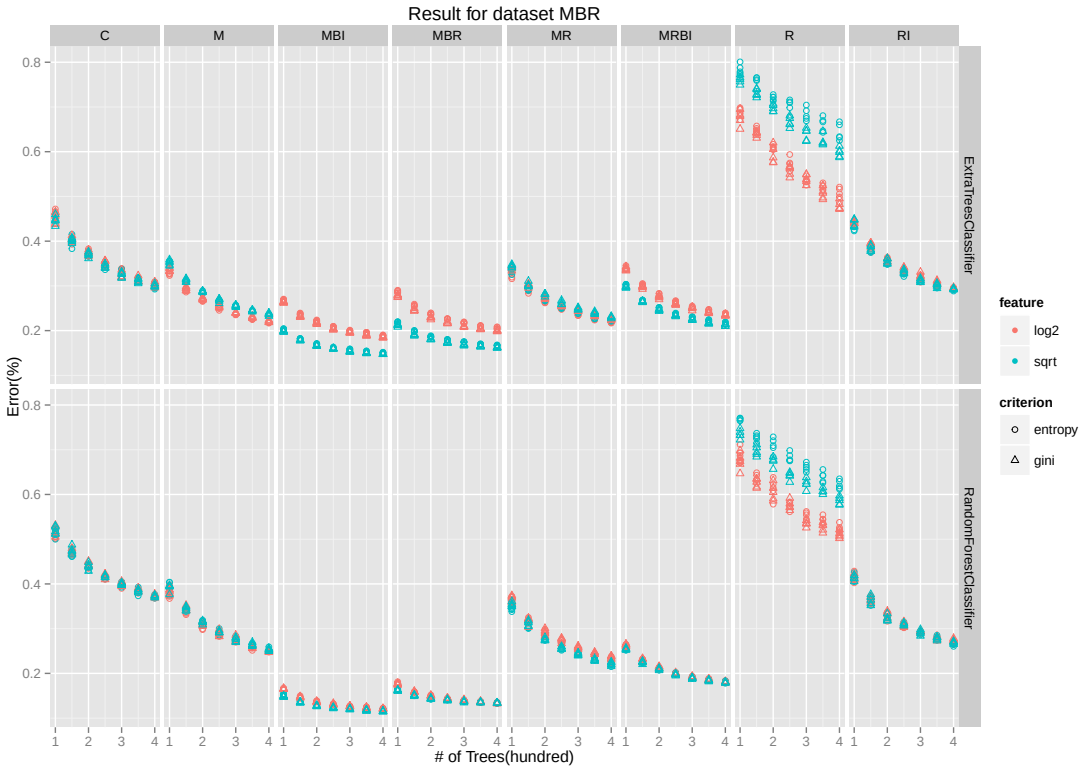


Fig. 5.4. Result for dataset MNIST with random background.

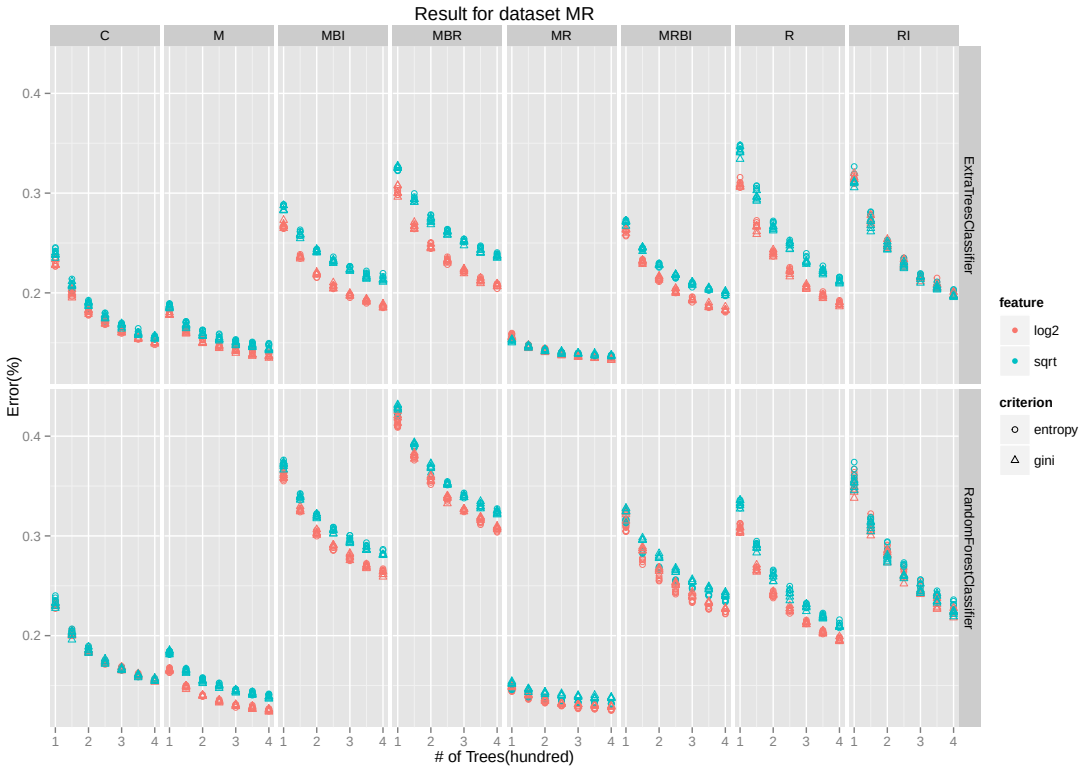


Fig. 5.5. Result for dataset MNIST with rotation.

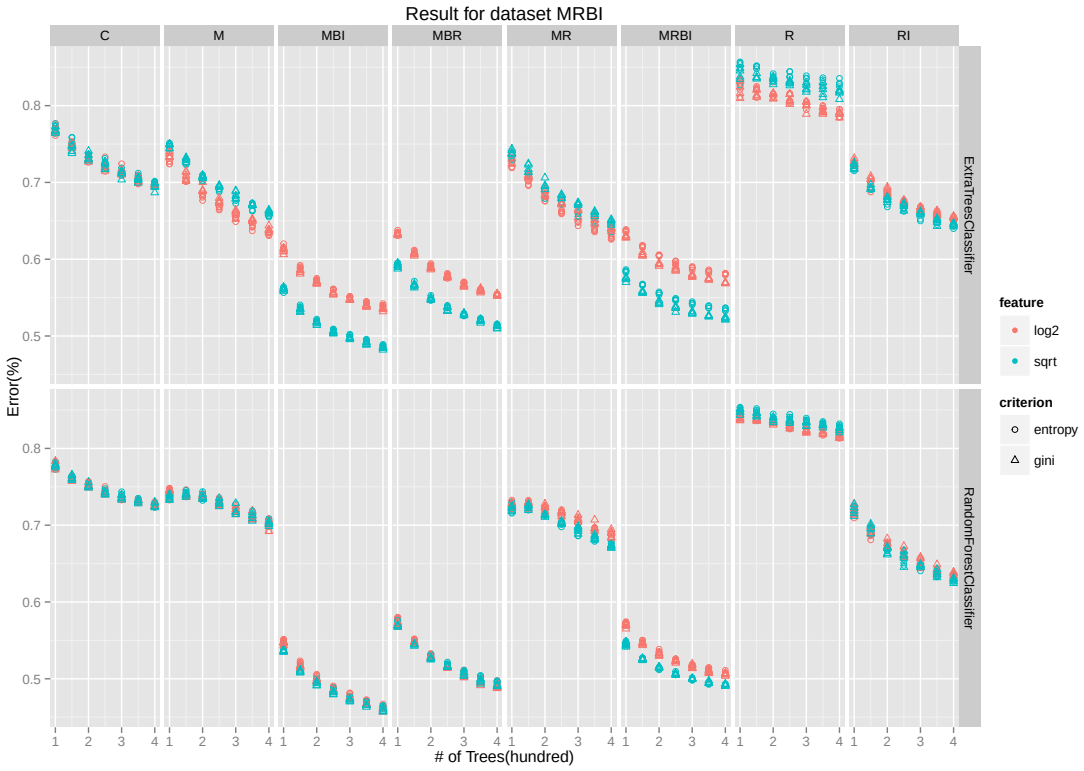


Fig. 5.6. Result for dataset with rotation and background images.

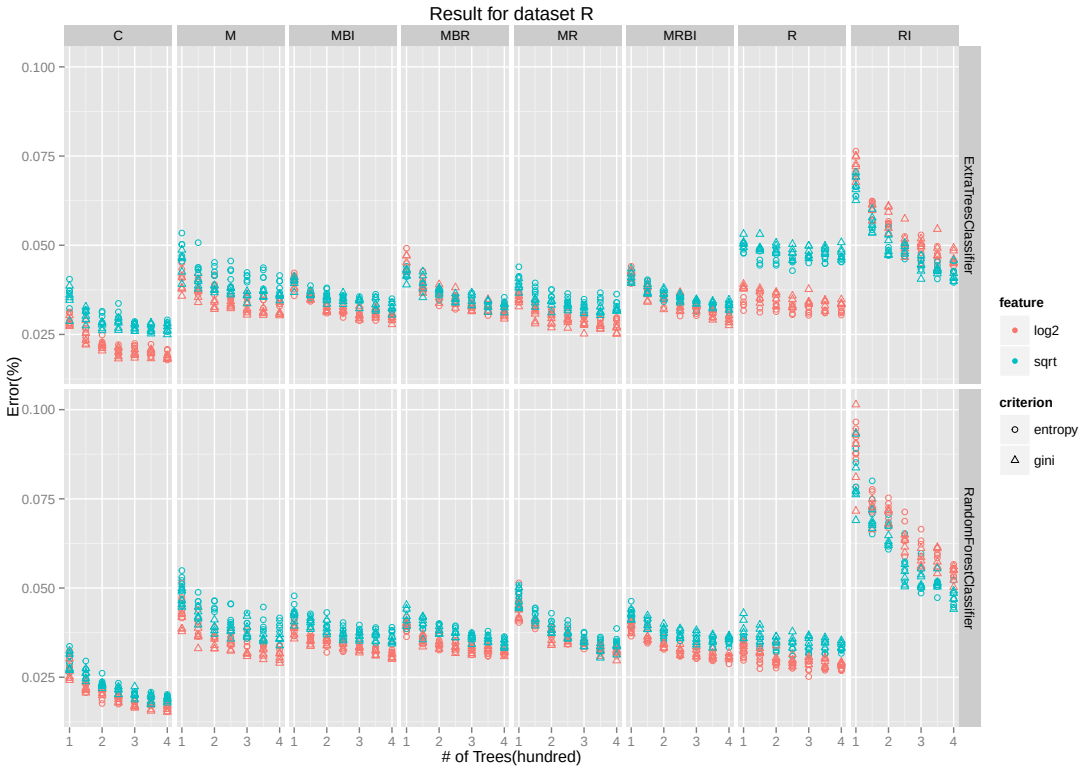


Fig. 5.7. Result for rectangle dataset.

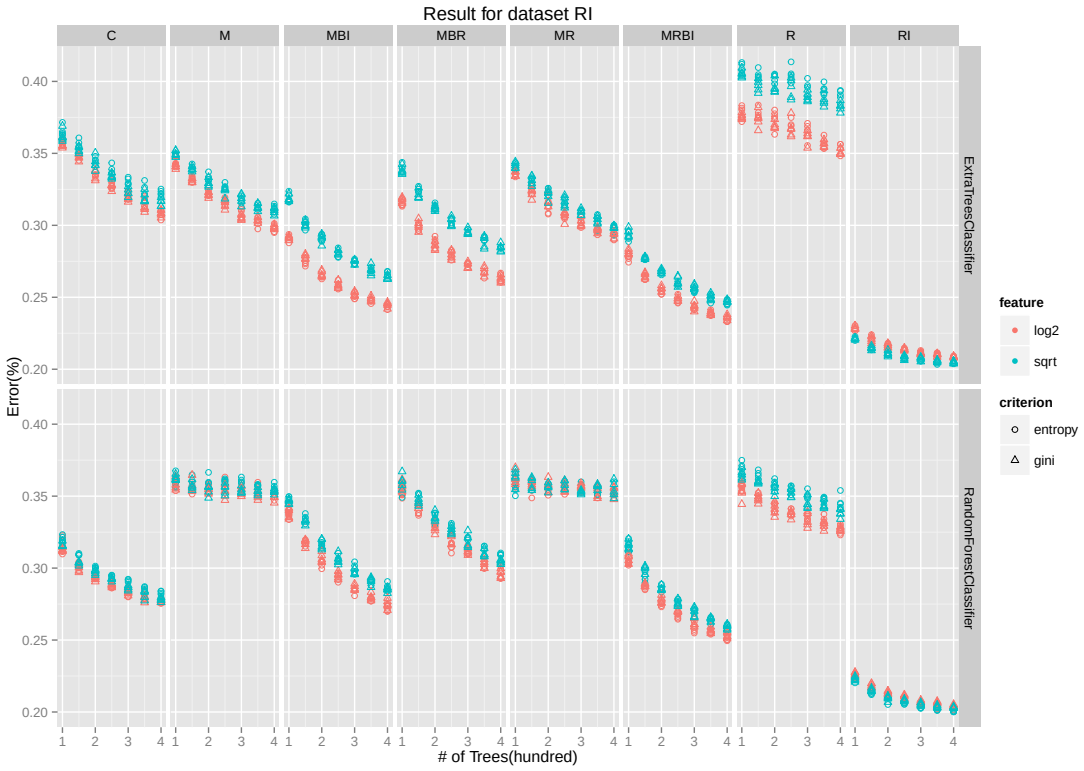


Fig. 5.8. Result for dataset rectangles with background image.

the original random forests. There are 8 columns in each row, which represent the 8 different datasets used to train the random forests. Each cell in a row shows the graph of classification error against the number of trees. Different colours and shapes shows the different number of features used and the different criterion used respectively.

5.2 Observations

5.2.1 General Trend

Observe the general trend of the classification errors going down as the number of trees increases. However, the rate of increasing is decreasing as depicted by the slopes of the graphs getting smaller (see Fig.5.1 to 5.8). **It empirically validates the claim that the representational power of the random forests increases asymptotically to a maximum value**.

5.2.2 The Effect of Training Data

In general, using the same data for the representation training data and classifier training data yields good enough result, if not the best. Dataset M is a notable exception, which achieves its best result not using itself as the representation training data but C and MR. Similar results are seen for MBR using MBI, MBRI using MBI, and R using C.

5.2.3 The Effect of Criterion Used and Number of Features Used

The graphs show no distinct separation between different shapes, and it indicates that different best split criteria contibures little to the classification error. On the other hand, the clear separation of colours suggests that the number of features used in choosing the best split does give a sizeable impact to the classification errors.

5.2.4 The Effect of Randomisation Scheme

Two kinds of random forests are used here, one is the original random forests proposed by Breiman, which does not make use of random splitting; and another is the extra random trees, which does make use of the random splitting. In general, the original random forests yields lower classification errors. A similar decreasing

trend is observed in both kinds of random forests but in most cases, the extra random trees starts with a higher classification error. This is an expected result, since the representation power of a single extra random tree should be lower because of the random splitting.

5.3 Analysis

This section proposes an analysis of the representation learned by the random forests. The properties of the representation mentioned in the analysis will be confirmed by empirical experiments. Given a dataset

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$$

where y_i is the label of the instance x_i. Furthermore, let $F = \{T_1, T_2, ..., T_m\}$ be a random forests with m trees. For a single tree T_k, one is interested in the the probability of any two instances x_i, x_j having the same label, that is $y_i = y_j$, given that the tree T_k assigns the same node to the two instances. The probability mentioned is to be called *representational power*, rp. The intuitive idea here is that a decision tree represents an instance as one of its node and the power of this representation can be measured by the probability of the instances having the same label given that they have the same representation.

Suppose the event of the two instances having the same label is denoted as d, and the event that the T_k assigns the same node to both of the instances is denoted as n_k, then the representational power can be written as:

$$rp = p(d|n_k) \tag{5.1}$$

The idea of the representational power of a tree can be generalize to representational power of a forests via the equation (5.2)

$$rp = p(d|n_1, n_2, ..., n_m) \tag{5.2}$$

With a bit of manipulation, equation (5.2) can be readily rewritten as equation (5.3)

$$\begin{aligned} p(d|n_1, n_2, ..., n_m) &= \frac{p(d, n_1, n_2, ..., n_m)}{p(d, n_1, n_2, ..., n_m)} \\ &= \frac{p(n_1)p(d|n_1)p(n_2|d, n_1)...p(n_m|d, n_1, n_2, ..., n_{m-1})}{p(n_1)p(n_2|n_1)...p(n_m|n_1, n_2, ..., n_{m-1})} \\ &= p(d|n_1)\frac{p(n_2|d, n_1)}{p(n_2|n_1)}...\frac{p(n_m|d, n_1, ..., n_{m-1})}{p(n_m|n_1, ..., n_{m-1})} \end{aligned} \tag{5.3}$$

Observe that there are two main parts in the equation (5.3). $p(d|n_1)$ is the representational power of the first tree. The second part is the most interesting, which consists of factors of the form

$$\frac{p(n_j|d,n_1,...,n_{j-1})}{p(n_j|n_1,...,n_{j-1})}$$

Notice the numerator, $p(n_j|d,n_1,...,n_{j-1})$, denotes the probability of two instances are assigned the same node by the tree T_j given that all the previous trees assign the same node to them and that they are, in fact, of the the same label. On the other hand, the denominator, $p(n_j|n_1,...,n_{j-1})$, denotes the similar probability, only in which case the two samples are not necessary of the same label.

There are **two main observations** can be drawn from the equation (5.3):

1. Knowing that the two instances are in fact of the same label should increase the probability of they being assigned the same node. In terms of the formula, $p(n_j|d,n_1,...,n_{j-1})$ should be greater than $p(n_j|n_1,...,n_{j-1})$. That is, all the factors in the form $\frac{p(n_j|d,n_1,...,n_{j-1})}{p(n_j|n_1,...,n_{j-1})}$ in equation (5.3) is greater than one. In other word, **the representational power of the forests increases with the number of trees inside it**.

2. The more the tree assigns the same node to the two instances the higher the probability that the two instances are of the same label. In terms of the equation,

$$\frac{p(n_j|d,n_1,...,n_{j-1})}{p(n_j|n_1,...,n_{j-1})} \to 1 \text{ as } j \to \infty$$

That is to say the information that they are of the same label become less important as the number of trees grows.

Combining the two observations, it can be expected that the representational power of the forests increases with the number of trees as each of the factors in the equation (5.3) is greater than 1. However it could also be expected that the rate of increase slows down as the number of trees increases for the factors that converge to 1. In other word, the representational power increases asymptotically to a maximum value.

Chapter 6

AN EMERGENT CONCEPT

In the previous section, it has been empirically established that the method proposed in this thesis works but no justification as to why it works is provided. The aim of this section is to provide such justification and, more specifically, to answer two questions. First of all, since linear SVM is used as the classifier, the random forests must somehow transform the original feature space into a new feature space, in which the data points are linearly separable. The mechanism of this transformation is known. Secondly, the method here, curiously enough, works even when two different datasets are used to train the random forest and the linear SVM. It seems that a good representation is independent of its classification. How this works is the main question here.

The plan here is to apply the method in this thesis to some really simple dataset, so simple that the underlying principles can be revealed easily. The first toy dataset consists of 2 circles, one inside another. The inner circle is labeled white while the outer circle is labeled black. Fig.6.1 shows the decision boundaries of applying 3 different methods to the dataset. The 3 methods are Random Forests á la Breiman, SVM with RBF kernel, and finally using linear SVM on transformed data. The points in darker region are more probable to be labeled white. Conversely, the points in lighter region are more probable to be labeled black.. The effect of using the forests-induced representation is similar to that of using a kernel. The only difference here is that the RBF kernel produces a smoother decision boundary. This is expected since, after all, the forests-induced representation is binary in nature. Fig.6.2 shows a similar result for a slightly more complicated dataset. Again, the kernelised SVM produces a smooth decision boundary. The decision boundary of the method in this paper is less smooth; however, it is still smoother than that of the Random Forests.

Both of the examples show that the forests-induced representation does a similar job as a kernel. Unlike the kernel method, it achieves a similar effect by directly constructing the high dimensional representation. On the other hand, one

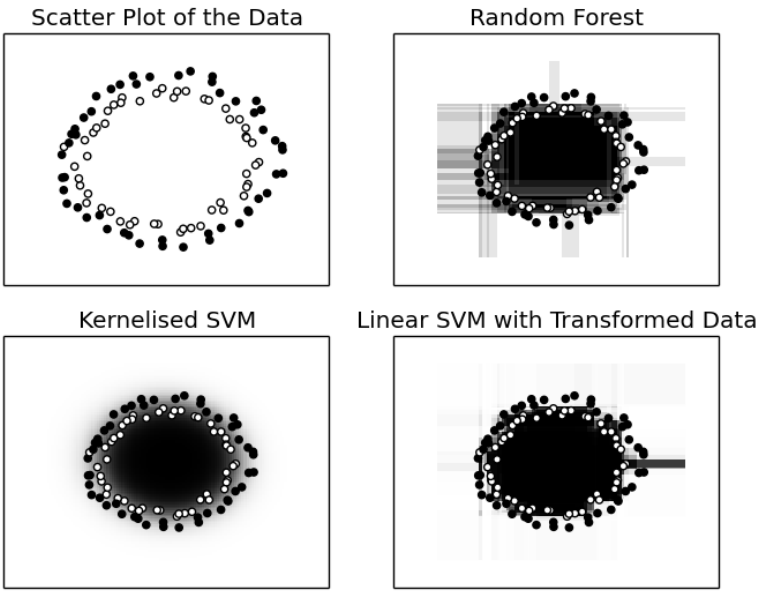


Fig. 6.1. Circle Dataset.

of the main selling points of the kernel method is that explicit representation is not necessary.

6.1 The Vicinity Concept

To look into the details of how this representation works, a new concept called **vicinity** is proposed here. Intuitively, the vicinity of a point is the average distance from itself to the other neighbourhood points. The forests transforms the original feature space into a new feature space, in which the data points are linearly separable. As shown by Fig.6.3, the new feature space could be stretched, compressed, and subjected to many other transformations. One way to characterise the transformation is to look at how it distorts the distance between 2 points. Observe that the points in the original feature space that are within 1 unit distance from a point p might not be the same points in the transformed feature space. This observation motivates the following definition of **vicinity**.

Definition Let $P = \{p_1, p_2, ..., p_n\}$ be a set of points that are 1 unit distance from a point p, and T is an arbitrary transformation, then the *vicinity* of p in the trans-

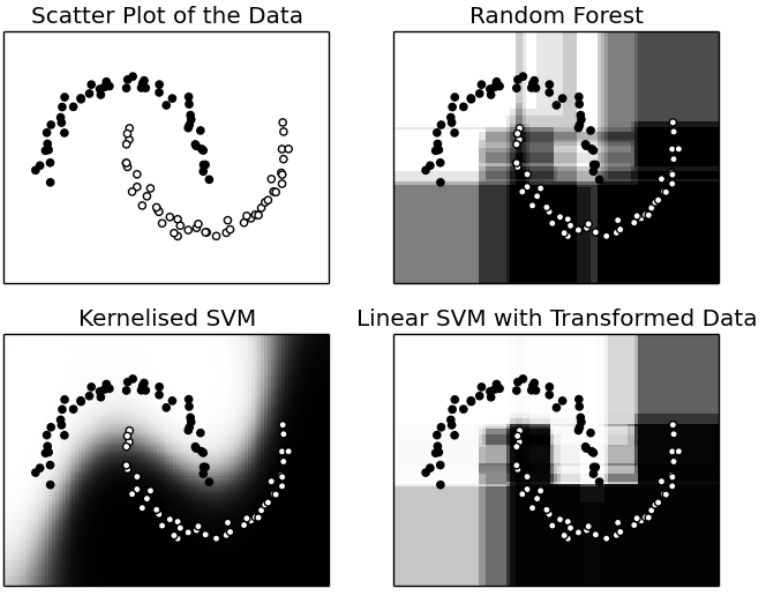


Fig. 6.2. Moon Dataset.

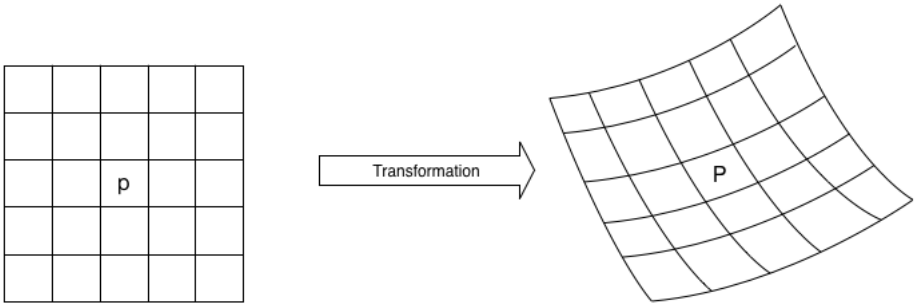


Fig. 6.3. Transforming of Feature Space.

formed space is given by the following formula:

$$vic(p) = \frac{\sum_{i=1}^{n} |p - p_i|}{n}$$

Notice that when T is the identity transformation, the vicinity of any point will be constantly 1. When the vicinity is greater than 1, the distances between points are stretched and the points are being pushed away from each other. Conversely when the vicinity is less than 1, the distances between points are compressed and the points are being pulled together. In the first case, the points are being *spread-*

out; and in the second case, the points are being *compressed*. In a highly varying region in the space, that is, the labels or values of the points vary a lot in a small area, the vicinity is expected to be greater than 1. This is because for the decision trees inside the forests to differentiate them, it is more than likely that they will be assigned to different terminal nodes. The more the trees assign different terminal nodes to the points, the further away they are in the new feature space, and hence, a high vicinity. By the same token, a highly uniform region will likely have a low vicinity.

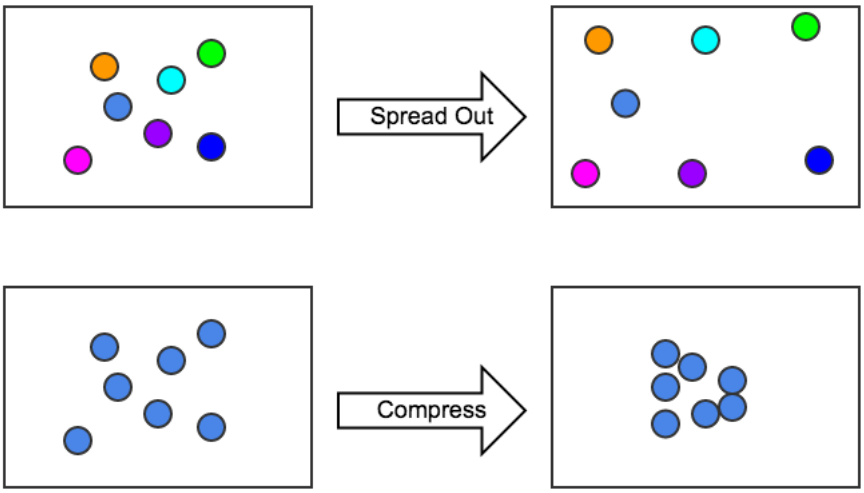


Fig. 6.4. A highly varying region tends to be blown up while a uniform region tends to be compressed.

The Fig.6.5 and 6.6 show the vicinity plot of the previous 2 toy datasets. The red region indicates high vicinity while the dark region indicates low vicinity. Observe that the high vicinity regions tend to be the boundaries between two classes of points. It is because the boundaries are the places where the labels of the points vary a lot.

It can be easily imagined that as the boundaries points being spread-out and the points of different classes being push away from each other to many different dimensions. As a result, the points are linearly separable in the new feature space. This also explain why random forests trained with random data generate decent representation. It can be considered that the job of the forests-induced representation is to spread out the highly varying regions and insert more space in between the points in service of better classification. In the case of using points uniformly distributed across the original feature space, the whole space is being spread out approximately to the same degree. In most cases, the points are still being pushed away far enough to be linearly separable.

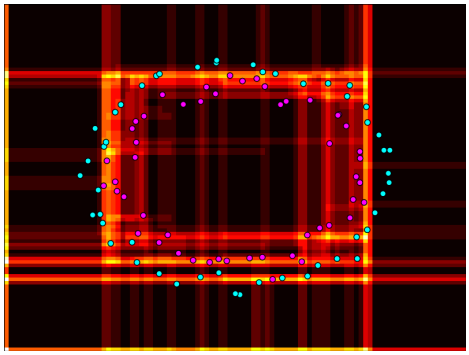


Fig. 6.5. Vicinity plot for the circle dataset.

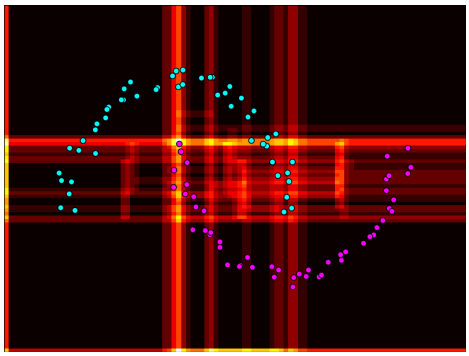


Fig. 6.6. Vicinity plot for the moon dataset.

To illustrate the points above, considers the dataset consisting of a cross. The line from left to right is labeled white and the line from right to left is labeled black. The decision boundaries in Fig.6.7 show that the method in this thesis works equally well compared to the random forests as classifier and the SVM. The vicinity plot (Fig.6.8) also shows that the expected high vicinity area is located at the boundary of the two classes of points, that is, at their intersection.

Note that the result above is obtained by training the random forests with the cross dataset, the same dataset used in classification. In order to illustrate the point that representation generation and classification are independent tasks, the random forests is trained with another dataset and then used to generate a new representation for the classification task. The other dataset used here consists of uniformly distributed points clustered at the intersection of the 2 lines. As shown by the similarity of Fig.6.7 and Fig.6.9, the fact that the random forests is trained with different dataset does not affect the classification result as long as the other dataset is carefully chosen.

If the dataset used to generate the new representation is poorly chosen, where the points are placed far from the highly varying region, then the quality of the

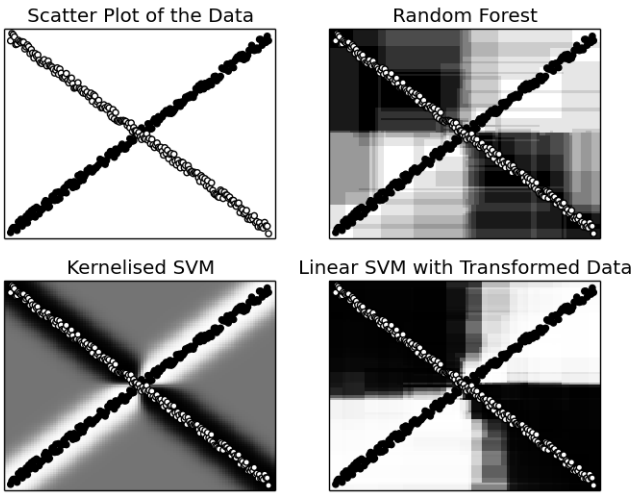


Fig. 6.7. Decision boundary for the Cross dataset.

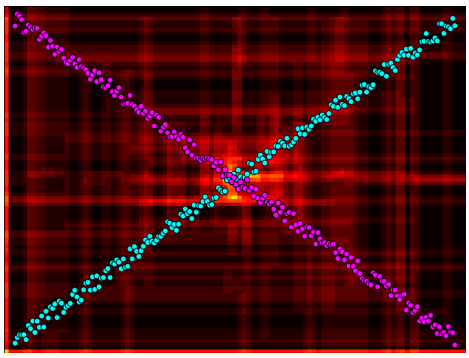


Fig. 6.8. Vicinity plot for the Cross dataset.

classification will deteriorate rapidly. For example, if the random forests is trained using the dataset with data points clustered at the corner of the feature space, far from the intersection point of the two lines, then the classification result is far worse than using other kinds of methods (see Fig.6.11). As shown by the vicinity plot in Fig.6.12, the large part of the upper left corner is compressed. Even an effective classifier such as SVM has great difficulty in differentiating the white points from the black points because they are so close together in the new feature space.

The learning of representation is independent from the performing of classification. A priori knowledge can thus be imposed into the representation. Suppose a certain region in the feature space is known a priori to be highly varying and

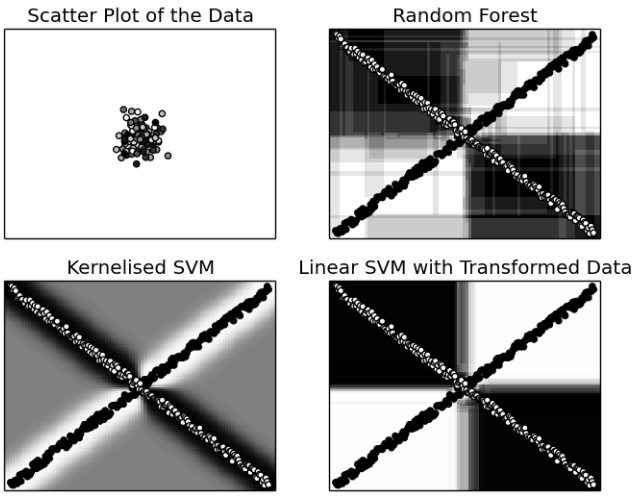


Fig. 6.9. Decision boundary for the cross dataset using random forests trained with points clustered at the intersection point.

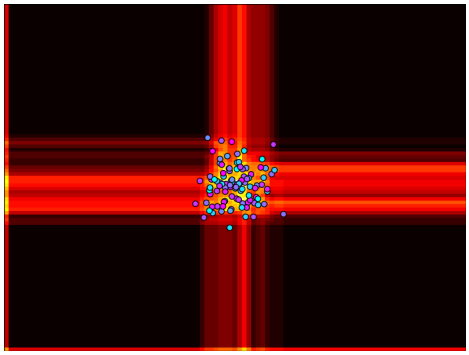


Fig. 6.10. Vicinity plot for the cross dataset using random forests trained with points clustered at the intersection point.

yet scarse in data, a random forests can be trained using random data clustered around that region. As a result, the new representation will provide a more refined structure around that area and thus provide the condition for better classification.

6.2 Conclusion

A new concept called **vicinity** is introduced in this chapter and it is used to explain the working principles behind the forests-induced representation. Intuitively, the

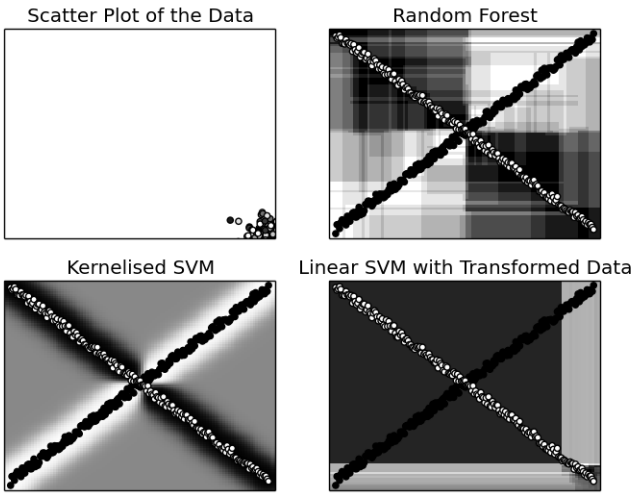


Fig. 6.11. Decision boundary for the cross dataset using random forests trained with points clustered at the tail of the line.

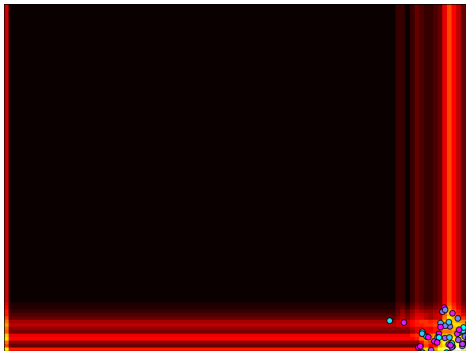


Fig. 6.12. Vicinity plot for the cross dataset using random forests trained with points clustered at the tail of the line.

forests-induced representation spreads out the highly varying region where different classes of data points are entangled closely together. As a result, the previously entangled data points are being pushed away from each other to other dimensions and hence they become linearly separable. By separating the concerns of learning representation and performing classification, a new way to improve the result of the classification is introduced. A priori knowledge can be imposed by carefully choosing the dataset used to generate the new representation. Further research along this direction will be discussed in the next chapter.

Chapter 7

CONCLUSION AND FUTURE DIRECTIONS

In this final chapter, the proposed method in this thesis is put into the perspectives of self-taught learning and manifold learning, two broader machine learning frameworks. Finally, three ways are pointed out to extend the existing work. One could deepen the theoretical analysis and contruct a refined mathematical model to explain the representation. One could also try to improve the efficiency and effectiveness of the algorithm. Last but not least, one could explore the possible applications of the proposed method.

7.1 Self-Taught Learning and Manifold Learning

In the setting of self-taught learning [73], there are two datasets D and D', one is supposed to learn a high level feature representation from D' and use it to represent the data in D. Thus, the method proposed in this thesis can be classified as a kind of self-taught learning. The authors of the paper [73] used sparse coding to learn the high level feature representation, whereas, in this thesis, the random forests is used. The same paper stressed that the main benefit of self-taught learning is the ease of getting large amount of unlabelled data D'. Take an example from the paper, should one want to classify the images of rhino and elephant, it is hard to collect a large number of labelled images of the two animals. On the contrary, there are virtually limitless number of unlabelled images, which are not limited to be either rhino or elephant, on the internet. The same benefit applies to the proposed method in this thesis as well. On the other hand, this thesis goes from one extreme of using totally random dataset D' to the other extreme of using strategically chosen dataset D'. Nonetheless, the proposed method in this thesis shares the same spirit of self-taught learning.

Manifold learning is another machine learning paradigm that is closely related to the method proposed here. The data in machine learning usually lives in a very high dimensional space. Even a relatively small image of size 20x20 lives in a four hundred dimensional space. However, a random point in that four hundred dimensional is very unlikely to be a meaningful image. Thus it is reasonable to assume that all the data live in a low dimensional subspace. Manifold learning is the attempt to find such subspace. The proposed method here seemingly goes to the opposite direction. The dimension of the induced representation is very much higher than that of the original representation. Nonetheless, the induced representation is very sparse, in the sense that most of its components are zero. Thus the method proposed in this thesis can be viewed as embedding the original representation in a low dimensional subspace of a high dimensional feature space.

7.2 Future Directions

Section 5.3 and the whole of chapter 6 are devoted to the theoretical explanation of the induced representation. One could collect the most essential parts to build a coherent mathematical model under the frameworks mentioned in the previous section. The main payoff of such an effort is a sound foundation for further development of the method.

There is also much room for improvement in algorithms. So far only binary trees are used in the construction of the random forests. The usage of a more complicated tree stucture is a direction worth pursuing. Besides, the proposed representation in this thesis is intrinsically binary. All the components are either one or zero depending on which terminal node the data eventually traverse to. In the course of traversal, the decision of choosing the nodes is solely based on whether a particular feature is greater or less than a certain threshold. Whether the feature is greater than the threshold by orders of mangnitude or just by a fraction is immaterial. Perhaps it is a good idea to encode such information into the representation.

Although the method proposed in this thesis is general enough to apply to many classification tasks, it is best used in the case where the labelled data is scarse and the unlabelled data is abundant. This is the same argument applied to self-taught learning. A more interesting case is when one knows a priori wherein the data is scarce. One could then train the random forests using another dataset that is dense in that particular region. The resulting representation would enable better discrimination of data points in that region.

Anomaly and fraud detection are possible use cases here. Usually, there are huge number of data points that represent the normal situation and relatively small amount of data points that are anomalistic. Suppose one want to detect if a nuclear

reactor is working normally. There will be a lot of data of the nuclear reactor when it is working normally but there will be very little data when it is faulty because, in such a situation, even a single faulty case is catastrophic.

At the end, another application that is not directly related to machine learning is presented. The random forests induced representation can serve as an interesting variation of the probabilistic data structure, Bloom Filter [74]. It was first conceived by Burton Howard Bloom in 1970 as a space efficient way to test if an element is in a set. At its core, a Bloom Filter is given by a bit array of size m (think of it as a binary vector of length m) and k hash functions. The hash functions map the set elements to one of the m array positions. The array is initialize to be all zero. An element is added by setting the values of the positions mapped by the hash functions to be one. To query if an element is in the set, compare the positions of the elements mapped by the hash functions to the bit array. If all the corresponding values in the positions are one, then it is possible that the element is in the set; otherwise, it is not in the set. Note the similarity between the bit array in a Bloom Filter and the random forests induced representation, as well as that between the function performed by the hash functions and that by the decision trees. Good hash functions are required to be independent from each other and are fast to compute. Decision trees fit the description perfectly well. Since Bloom Filter is an important data structure that is used in many big projects such as Google Big Table [75], it is worthwhile to explore its alternative implementation.

The three directions of extending the work in this thesis, namely theory, algorithm, and application are interwoven intimately. Advancement in either direction would certainly be beneficial for the other two. It is hoped that this thesis will be the starting point for future research in these directions.

Bibliography

[1] Tang, A., Foong, J.T.: A qualitative evaluation of random forest feature learning. In: Recent Advances on Soft Computing and Data Mining. Springer (2014) 359–368

[2] Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25. (2012) 1106–1114

[3] Seide, F., Li, G., Yu, D.: Conversational speech transcription using context-dependent deep neural networks. In: in Proc. Interspeech '11. (2011) 437–440

[4] Bengio, Y.: Learning deep architectures for ai. Foundations and trends® in Machine Learning **2**(1) (2009) 1–127

[5] Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural computation **18**(7) (2006) 1527–1554

[6] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE, 86(11):2278-2324. Volume 86., IEEE (1998) 2278–2324

[7] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. Technical report, DTIC Document (1985)

[8] Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786) (2006) 504–507

[9] Salakhutdinov, R., Mnih, A., Hinton, G.: Restricted boltzmann machines for collaborative filtering. In: Proceedings of the 24th international conference on Machine learning, ACM (2007) 791–798

[10] Olshausen, B.A., Field, D.J.: Sparse coding with an overcomplete basis set: A strategy employed by v1? Vision research **37**(23) (1997) 3311–3325

[11] Lee, H., Battle, A., Raina, R., Ng, A.: Efficient sparse coding algorithms. In: Advances in neural information processing systems. (2006) 801–808

[12] Siegelmann, H.T., Sontag, E.D.: Turing computability with neural nets. Applied Mathematics Letters **4** (1991) 77–80

[13] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. In: Neurocomputing: foundations of research, MIT Press (1988) 696–699

[14] Bishop, C.M.: Neural Networks for Pattern Recognition. 1 edn. Oxford University Press, USA (January 1996)

[15] Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). 1st ed. 2006. corr. 2nd printing 2011 edn. Springer (October 2007)

[16] Haykin, S.: Neural Networks: A Comprehensive Foundation. 2nd edn. Prentice Hall PTR, Upper Saddle River, NJ, USA (1998)

[17] Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. Pattern Analysis and Machine Intelligence, IEEE Transactions on **35**(8) (Aug 2013) 1798–1828

[18] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., Montréal, U.D., Québec, M.: Greedy layer-wise training of deep networks. In: In NIPS, MIT Press (2007)

[19] Bourlard, H., Kamp, Y.: Auto-association by multilayer perceptrons and singular value decomposition. Biological cybernetics **59**(4-5) (1988) 291–294

[20] Smolensky, P.: Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In Rumelhart, D.E., McClelland, J.L., PDP Research Group, C., eds.: Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. MIT Press, Cambridge, MA, USA (1986) 194–281

[21] Hinton, G.: A practical guide to training restricted boltzmann machines. Momentum **9**(1) (2010) 926

[22] Hinton, G.E.: Training products of experts by minimizing contrastive divergence. Neural computation **14**(8) (2002) 1771–1800

[23] Carreira-Perpinan, M.A., Hinton, G.E.: On contrastive divergence learning. In: Proceedings of the tenth international workshop on artificial intelligence and statistics, Society for Artificial Intelligence and Statistics NP (2005) 33–40

[24] Scholkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge, MA, USA (2001)

[25] Kashima, H., Tsuyoshi, I., SUGIYAMA, M.: Recent advances and trends in large-scale kernel methods. IEICE Transactions on Information and systems **92**(7) (2009) 1338–1353

[26] Bengio, Y., Lecun, Y. In: Scaling learning algorithms towards AI. MIT Press (2007)

[27] Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions (2008)

[28] Singer, Y., Srebro, N.: Pegasos: Primal estimated sub-gradient solver for svm. In: In ICML. (2007) 807–814

[29] Bottou, L.: Large-Scale Machine Learning with Stochastic Gradient Descent. In Lechevallier, Y., Saporta, G., eds.: Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010), Paris, France, Springer (August 2010) 177–187

[30] Bottou, L.: Large-scale kernel machines. The MIT Press (2007)

[31] Sun, P., Yao, X.: Sparse approximation through boosting for learning large scale kernel machines. Neural Networks, IEEE Transactions on **21**(6) (2010) 883–894

[32] Rahimi, A., Recht, B.: Random features for large-scale kernel machines. In: In Neural Infomration Processing Systems. (2007)

[33] Decoste, D., Schölkopf, B.: Training invariant support vector machines. Machine Learning **46**(1-3) (2002) 161–190

[34] Collins, M., Duffy, N.: Convolution kernels for natural language. In: Advances in Neural Information Processing Systems 14, MIT Press (2001) 625–632

[35] Jaakkola, T.S., Haussler, D.: Probabilistic kernel regression models. In: In Proceedings of the 1999 Conference on AI and Statistics, Morgan Kaufmann (1999)

[36] Vishwanathan, S., Smola, A.J., Vidal, R.: Binet-cauchy kernels on dynamical systems and its application to the analysis of dynamic scenes. International Journal of Computer Vision **73**(1) (2007) 95–119

[37] Lanckriet, G., Cristianini, N., Bartlett, P., Ghaoui, L.E.: Learning the kernel matrix with semi-definite programming. Journal of Machine Learning Research **5** (2002) 2004

[38] Tsuda, K., Rätsch, G., Warmuth, M.K.: Matrix exponentiated gradient updates for on-line learning and bregman projection. In: Journal of Machine Learning Research. (2005) 995–1018

[39] Cristianini, N., Kandola, J., Elisseeff, A., Shawe-Taylor, J.: On optimizing kernel alignment (2001)

[40] Ong, C.S., Williamson, R.C., Smola, A.J.: Hyperkernels. In: Advances in neural information processing systems. (2002) 478–485

[41] Ong, C.S., Williamson, R.C., Smola, A.J.: Learning the kernel with hyperkernels. In: Journal of Machine Learning Research. (2005) 1043–1071

[42] Platt, J.C.: Advances in kernel methods. In Schölkopf, B., Burges, C.J.C., Smola, A.J., eds.: title = Advances in Kernel Methods. MIT Press, Cambridge, MA, USA (1999) 185–208

[43] Hinton, G.E.: Learning distributed representations of concepts. In: Proceedings of the eighth annual conference of the cognitive science society, Amherst, MA (1986) 1–12

[44] Bengio, Y., Schwenk, H., Senécal, J.S., Morin, F., Gauvain, J.L.: Neural probabilistic language models. In: Innovations in Machine Learning. Springer (2006) 137–186

[45] Bengio, Y., Delalleau, O., Simard, C.: Decision trees do not generalize to new variations. Computational Intelligence **26**(4) (2010) 449–467

[46] Moosmann, F., Nowak, E., Jurie, F.: Randomized clustering forests for image classification. Pattern Analysis and Machine Intelligence, IEEE Transactions on **30**(9) (2008) 1632–1646

[47] Vens, C., Costa, F.: Random forest based feature induction. In: Data Mining (ICDM), 2011 IEEE 11th International Conference on. (2011) 744–753

[48] Breiman, L.: Classification and regression trees. The Wadsworth and Brooks-Cole statistics-probability series. Chapman & Hall (1984)

[49] Quinlan, J.R.: C4. 5: programs for machine learning. Volume 1. Morgan kaufmann (1993)

[50] Schapire, R.E.: The strength of weak learnability. Machine learning **5**(2) (1990) 197–227

[51] Amit, Y., Geman, D.: Randomized inquiries about shape: An application to handwritten digit recognition. Technical report, DTIC Document (1994)

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

[53] Ho, T.K.: Random decision forests. In: Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on. Volume 1., IEEE (1995) 278–282

[54] Ho, T.K.: The random subspace method for constructing decision forests. Pattern Analysis and Machine Intelligence, IEEE Transactions on **20**(8) (1998) 832–844

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

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

[57] Bosch, A., Zisserman, A., Muoz, X.: Image classification using random forests and ferns. In: Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on, IEEE (2007) 1–8

[58] Criminisi, A., Shotton, J., Robertson, D., Konukoglu, E.: Regression forests for efficient anatomy detection and localization in ct studies. In: Medical Computer Vision. Recognition Techniques and Applications in Medical Imaging. Springer (2011) 106–117

[59] Fanelli, G., Gall, J., Van Gool, L.: Real time head pose estimation with random regression forests. In: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE (2011) 617–624

[60] Geremia, E., Menze, B.H., Clatz, O., Konukoglu, E., Criminisi, A., Ayache, N.: Spatial decision forests for ms lesion segmentation in multi-channel mr images. In: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2010. Springer (2010) 111–118

[61] Leistner, C., Saffari, A., Santner, J., Bischof, H.: Semi-supervised random forests. In: Computer Vision, 2009 IEEE 12th International Conference on, IEEE (2009) 506–513

[62] Chatrchyan, S., Khachatryan, V., Sirunyan, A.M., Tumasyan, A., Adam, W., Aguilo, E., Bergauer, T., Dragicevic, M., Erö, J., Fabjan, C., et al.: Observation of a new boson at a mass of 125 gev with the cms experiment at the lhc. Physics Letters B **716**(1) (2012) 30–61

[63] Criminisi, A., Shotton, J.: Decision forests for computer vision and medical image analysis. Springer (2013)

[64] Maturana, D., Mery, D., Soto, A.: Face recognition with decision tree-based local binary patterns. In: Computer Vision–ACCV 2010. Springer (2011) 618–629

[65] Ren, S., Cao, X., Wei, Y., Sun, J.: Face alignment at 3000 fps via regressing local binary features. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Oral (2014)

[66] Bache, K., Lichman, M.: UCI machine learning repository (2013)

[67] Miroslav Kobetski (CVAP, KTH), J.S.: Discriminative tree-based feature mapping. In: Proceedings of the British Machine Vision Conference, BMVA Press (2013)

[68] Quinlan, J.R.: Induction of decision trees. Machine learning **1**(1) (1986) 81–106

[69] Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Machine learning **63**(1) (2006) 3–42

[70] Larochelle, H., Erhan, D., Courville, A., Bergstra, J., Bengio, Y.: An empirical evaluation of deep architectures on problems with many factors of variation. In: Proceedings of the 24th international conference on Machine learning, ACM (2007) 473–480

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

[72] Van der Maaten, L., Hinton, G.: Visualizing data using t-sne. Journal of Machine Learning Research **9**(11) (2008)

[73] Raina, R., Battle, A., Lee, H., Packer, B., Ng, A.Y.: Self-taught learning: transfer learning from unlabeled data. In: Proceedings of the 24th international conference on Machine learning, ACM (2007) 759–766

[74] Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7) (1970) 422–426

[75] Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS) **26**(2) (2008) 4