

École polytechnique de Louvain

Information theory and machine learning: Information Bottleneck theory applied to classification trees and random forests

Author: Martin DE VROEY

Supervisor: **Jean-Charles DELVENNE**

Readers: **Marco SAERENS, Michel VERLEYSEN**

Academic year 2022–2023

Master [120] in Mathematical Engineering

Abstract

The Information Bottleneck (IB) method has been proposed as a theoretical framework to explain the effectiveness of deep learning. It has since become a widely-studied concept in the field of deep learning. However, research on the application of IB theory to other machine learning algorithms is almost nonexistent. In this work we explore the application of IB theory to decision trees and random forests for classification tasks. After establishing the theoretical background, we show evidence against some of the conclusions of earlier work on IB theory in the case of classification tasks. Further, we present a novel analysis of random forests through the IB method. Finally we apply the IB method to a nondeterministic classification task.

Acknowledgments

I would like to express my gratitude to my supervisor, Prof. Jean-Charle Delvenne, for his guidance, support, and encouragement throughout the process of researching and writing this thesis.

I also want to thank my friends and family for their continued support and encouragement and for listening to my rambling about a subject they did not always fully understand.

Finally, I would like to thank Elise for providing me with a perfect work environment for most of the redaction of my thesis, even though the cats were a little distracting.

Thank you.

Contents

1	Introduction	1
2	The Information Bottleneck framework	3
2.1	Mutual information	3
2.2	The information plane	4
2.3	The IB curve	5
2.4	The information theory of deep learning	7
3	Decision trees	9
3.1	How Decision Trees Work	11
3.2	Advantages and Disadvantages	14
3.3	Pruning	15
3.4	Summary	15
4	Random forests	17
4.1	How Random Forests Work	18
4.2	Advantages and Disadvantages	20
4.3	Summary	21
5	IB theory of decision trees and random forests	22
6	Empirical analysis	25
6.1	Replicated experiments	25
6.1.1	IB curves for deterministic classification tasks	26
6.1.2	Experiment 1: The IB curve contains suboptimal representations ?	27
6.1.3	Experiment 2: Better representations than IB optimal solution ?	29
6.1.4	Discussion	29
6.2	Experiments on Random Forests	30
6.2.1	The random forest representation in the information plane	31

6.2.2	Discussion	34
7	Nondeterministic classification	35
7.1	Nondeterministic classification tasks	35
7.2	Solving the IB problem	37
7.2.1	Implementation	38
7.3	Experiments	39
7.3.1	Creating training sets	39
7.3.2	Evaluating the models	40
7.3.3	Discussion	43
8	Conclusions and further work	44

List of Figures

2.1	<i>The IB curve, obtained by solving the IB problem for different values of β. Any representation of X falls within the feasible area.</i>	6
2.2	<i>The IB curve is piece-wise linear if $I(Y; X) = H(Y)$. [11]</i>	7
2.3	<i>DNN layers as successive representations of X in a Markov chain. [14]</i>	8
3.1	<i>Three different representations of a classification tree of kyphosis data. [2]</i>	9
3.2	<i>tree after one split</i>	10
3.3	<i>tree after two splits</i>	10
3.4	<i>tree after three splits</i>	11
3.5	<i>Node impurity measures for a classification problem with two classes. p is the proportion of class 1 observations in the node.</i>	12
4.1	<i>Diagram illustrating the use of a random forest. [1]</i>	18
6.1	<i>IB curves computed for the toy and MNIST dataset</i>	27
6.2	<i>Models trained on S_{train} (green stars) and on S_{full} (red dot) compared in the information plane using different parts of the datasets. I_Y and I_X are computed using S_{train} in (a) and (b), S_{full} in (c) and (d) and S_{test} in (e) and (f).</i>	28
6.3	<i>Pruned models (red dots) compared to non-pruned (green star) models in the information plane.</i>	29
6.4	<i>IB curves for the MNIST dataset (a) and forest representation T and final representation $\hat{Y}$ for a single tree.</i>	31
6.5	<i>Evolution of random forest representations T and predictions $\hat{Y}$ visualised in the information plane, as more trees are added to the forests. The color gradients illustrate the number of trees in a forest. Maximal depths for the trees of the forests are : (a) 2, (b) 5, (c) 10 and (d) no maximal depth.</i>	32
7.1	<i>Examles of ambiguous handwritten digits from the MNIST dataset.</i>	36

7.2	<i>Joint probability distribution $p(x, y = 1)$ (a) and conditional probability distribution $p(y = 1 x)$, where $x \in X$ are coordinates in a 2D grid and $y \in Y$ are binary target values.</i>	37
7.3	<i>IB curve obtained for the distribution $p(x, y)$ displayed in figure 7.2.</i>	39
7.4	<i>Two training sets. Purple and yellow pixels are elements of the training sets labeled 0 and 1 respectively. Blue pixels are elements that are not in the training sets. Some pixels may appear several times in the training sets since they were drawn with replacement. .</i>	40
7.5	<i>Deterministic map $p(\hat{y} x)$ generated by tr_2.</i>	41
7.6	<i>Nondeterministic IB curve and the predictions of four models compared in the information plane.</i>	42

Chapter 1

Introduction

In this thesis, we will explore the application of Tishby’s Information Bottleneck (IB) theory to decision trees and random forests. Decision trees and random forests are popular machine learning algorithms that are commonly used for both classification and regression. By using the insights of the Information Bottleneck theory, we aim to understand the trade-off between the accuracy and simplicity of these models, and to explore the applications of Information Bottleneck theory on machine learning methods other than deep neural networks.

The Information Bottleneck theory was first proposed by Tishby in 1999, and it has since become a widely-studied concept in the field of machine learning and in particular deep neural networks. This theory posits that there is a fundamental limit to the amount of information that can be extracted from a given dataset by a machine learning model, and that the goal of a machine learning algorithm is to find the optimal balance between the amount of useful information extracted from the input and the complexity of the internal representations.

In this thesis, we will apply the Information Bottleneck theory to the specific case of classification trees and random forests. We will begin by providing an overview of the Information Bottleneck theory and classification trees and random forests separately. Next, we will explore the ways in which the Information Bottleneck theory can be applied to these models.

After providing a theoretical foundation for our work, we will move on to the experimental phase of our study. We will conduct a series of experiments, first to replicate some of the experiments from Tytgat 2021 Masters thesis[3] and call into question some of its conclusions, then to explore the application of the Information Bottleneck theory to random forests. We will also introduce nondeterministic classification tasks and how Information Bottleneck theory applies to them.

For this last point we will need to develop a tool to solve the Information Bottleneck optimization problem.

Finally, we will summarize our findings and discuss their implications. We will also provide suggestions for future research in this area.

Overall, the goal of this thesis is to continue the exploration of the application of the Information Bottleneck theory to classification trees, started in the Tytgat 2021 Masters thesis[3] , and extend it to random forests as well as nondeterministic classification tasks. By examining the trade-off between accuracy and simplicity in these models, we aim to identify if the Information Bottleneck theory of machine learning has applications beyond deep neural networks.

Chapter 2

The Information Bottleneck framework

The Information Bottleneck method was introduced in 1999 by Naftali Tishby, Fernando C. Pereira, and William Bialek [16], and has since become a widely studied concept in the fields of machine learning and deep learning. It relies on information theory to find the optimal way to summarize (e.g. by clustering) a random variable X while preserving the maximum amount of information about the relevant observed variable Y . The term 'bottleneck' comes from the idea of squeezing the information that X provides about Y through a 'bottleneck'[16].

The Information Bottleneck method can also be understood as the problem of making a compressed representation T of X from which Y can be accurately predicted. There is a tradeoff between accuracy and compression, but any representation T that makes this tradeoff optimally, should satisfy

$$\min_{p(t|x)} I(X;T) - \beta I(T;Y)$$

where $I(X;T)$ and $I(T;Y)$ are the mutual information between X and T , and between T and Y respectively, and β is a Lagrange multiplier.

2.1 Mutual information

The main concept we need from information theory is mutual information. Mutual information is a measure of the mutual dependence between two random variables. It measures the amount of information that one random variable contains about another random variable.

More formally, let X and Y be two random variables with joint distribution $p(x, y)$. The mutual information between X and Y is defined as:

$$I(X; Y) = \sum_x \sum_y p(x, y) \log \left[\frac{p(x, y)}{p(x)p(y)} \right]$$

where $p(x)$ and $p(y)$ are the marginal distributions of X and Y , respectively.

Intuitively, mutual information measures the amount of overlap between the probability distributions of X and Y . If X and Y are independent, then their joint distribution is simply the product of their marginal distributions, and the mutual information is zero. On the other hand, if X and Y are highly dependent, then the mutual information is large.

Mutual information can also be expressed in terms of the Kullback-Leibler divergence:

$$I(X; Y) = D_{KL}(p(x, y) || p(x)p(y))$$

where the Kullback-Leibler divergence $D_{KL}(p || q)$ is a measure of distance between distributions p and q .

Alternatively, the mutual information can be expressed in terms of the entropy and conditional entropy of the variables X and Y :

$$I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

where $H(X)$ is the entropy of X , and $H(X|Y)$ is the conditional entropy of X given Y .

Though mutual information measures the link between two variables, it is more general than correlation. Consider $Z = \psi(X)$, where ψ is an invertible transformation. We have

$$I(Z; X) = I(X; X) = H(X)$$

showing that Z contains all the information of X , while measuring the correlation of X and Z might not reveal any particular link if ψ is non-linear.

2.2 The information plane

Consider two dependent random variables X and Y . Since they are dependent we have $I(X; Y) > 0$. Let T be a representation of X through a (possibly stochastic)

map. We can formulate this as a Markov chain:

$$Y \longrightarrow X \longrightarrow T$$

Any such representation T can be uniquely mapped to a point in the information plane defined by

$$(I(T; X), I(T; Y))$$

and respects the data processing inequality:

$$I(T; Y) \leq I(T; X)$$

Intuitively this means that T can't contain more information about Y than X contains, since all the information about Y that T has, comes from X .

2.3 The IB curve

Consider again a representation T such that

$$Y \longrightarrow X \longrightarrow T$$

We want to find such a representation T that is maximally informative about Y , while containing as little superfluous information as possible. Such a representation is similar to the concept of minimal sufficient statistic. In this context a minimal sufficient statistic would be a map or partition of X that captures all the information X contains about Y . Formally we can express this as follows:

$$T(X) = \arg \min_{S(X): I(S(X); Y) = I(X; Y)} I(S(X); X)$$

Where both S and T are maps or partitions of X .

Finding a minimal sufficient statistic is not possible in many cases. For this reason, Tishby et al.(1999)[16] proposed a relaxation of this optimisation problem, where T could be a stochastic map of X and T has to capture as much as possible of $I(Y; X)$ instead of all of it. This leads to a different optimisation problem:

$$\min_{p(t|x), p(y|t), p(t)} I(X; T) - \beta I(T; Y)$$

Where $t \in T$, $x \in X$, $p(t | x)$ is the stochastic map from X to T and β is a Lagrange multiplier. β determines the level of the tradeoff between compression of

X and prediction of Y , called the information bottleneck tradeoff. The solution to this problem is given by the following equations[14]:

$$\begin{cases} p(t | x) = \frac{p(t)}{Z(x; \beta)} \exp(-\beta D_{KL}[p(y | x) \| p(y | t)]) \\ p(t) = \sum_x p(t | x) p(x) \\ p(y | t) = \sum_x p(y | x) p(x | t) \end{cases}$$

where $Z(x; \beta)$ is the normalization function.

Solving these equations for different values of β and plotting the resulting optimal representations in the information plane yields the Information Bottleneck curve, as shown in figure 2.1[19]. This curve delimits the feasible region of the information plane for representations of X .

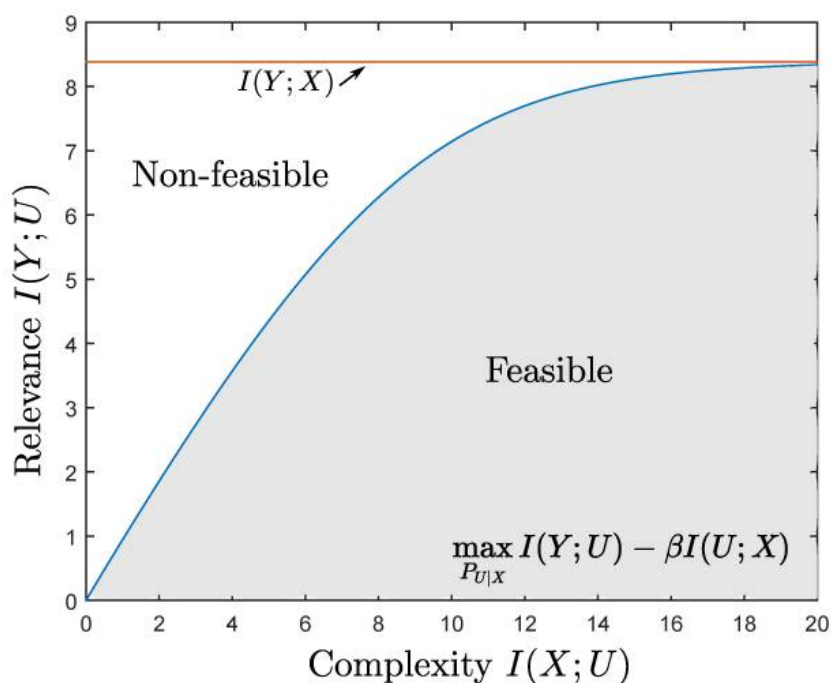


Figure 2.1: The IB curve, obtained by solving the IB problem for different values of β . Any representation of X falls within the feasible area.

If a deterministic map $f(X) = Y$ exists, then Y is fully determined by X , $I(Y; X) = H(Y)$ and the IB curve becomes piece-wise linear, as shown in figure 2.2.

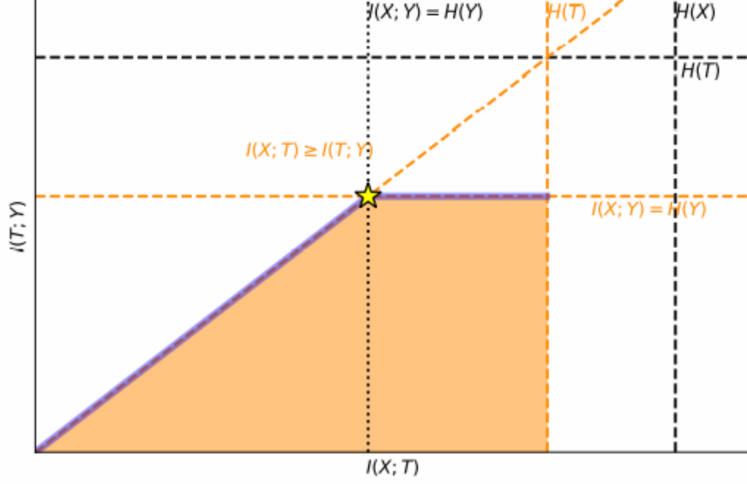


Figure 2.2: The IB curve is piece-wise linear if $I(Y; X) = H(Y)$. [11]

2.4 The information theory of deep learning

The Information Bottleneck has recently been proposed as a theoretical framework to explain the great success of Deep Neural Networks (DNNs) on numerous machine learning tasks [14]. The idea is to view the hidden layers of the network as representations T of the input X containing some amount of information about Y . In this view the successive layers form a Markov chain of representation ending in the prediction of the network Y , as shown on figure 2.3. We can thus write two chains of data processing inequalities for these representations:

$$\begin{aligned} I(X; Y) &\geq I(T_1; Y) \geq I(T_2; Y) \geq \dots \geq I(T_k; Y) \geq I(\hat{Y}; Y) \\ H(X) &\geq I(X; T_1) \geq I(X; T_2) \geq \dots \geq I(X; T_k) \geq I(X; \hat{Y}). \end{aligned}$$

The layer representations can be visualised in the information plane by computing $I(T; X)$ and $I(T; Y)$. Tishby et al. (1999) suggested that optimized DNNs layers should approach the Information Bottleneck curve in the information plane [16]. Pushing this idea further in Tishby et al. (2015) [17], they claimed that Deep Learning implicitly tries to solve the IB optimisation problem through stochastic gradient descent and back propagation.

Later Shwartz-Ziv and Tishby proposed that visualising these representations in the information plane reveals the training dynamics of a DNN. In particular that it can be divided into two phases: empirical error minimization (ERM) during

which $I(T, Y)$ increases, and representation compression during which $I(X, T)$ decreases. However, Saxe et al. countered this claim in their work[12], stating that the presence of this compression phenomenon in DNNs depends on the particular activation function being used. Noshad et al. further explored this controversy, and found more evidence in favor of the initial claim[9]. According to Goldfeld et al., the compression phase is caused by geometric factors rather than information-theoretic ones, a perspective that has been supported by Geiger (2020).[7][6]

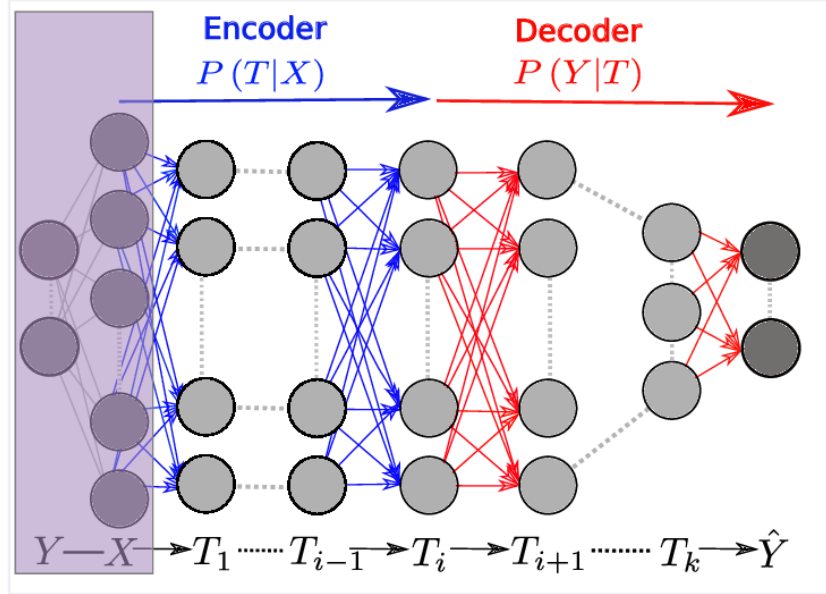


Figure 2.3: *DNN layers as successive representations of X in a Markov chain.*[14]

Chapter 3

Decision trees

Decision trees are a supervised learning method used in both regression and classification tasks. Decision trees are among the most popular machine learning algorithms given their intelligibility and simplicity[18]. In a classification tree, the goal is to create a model that predicts the class of a given data point by learning simple decision rules inferred from the data features. The decision tree is a supervised learning model that works by recursively partitioning the data into smaller and smaller subsets, based on a set of decision rules. The decision tree is a powerful tool for both classification and regression problems.

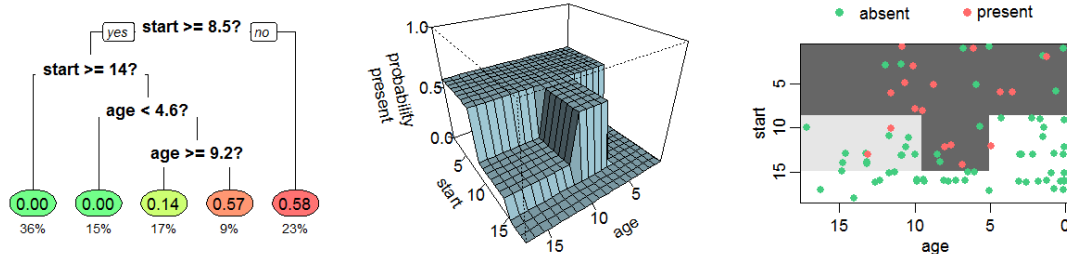


Figure 3.1: Three different representations of a classification tree of kyphosis data.[2]

Example

To illustrate how trees are used, consider a small dataset with two features $X1$ and $X2$ and a binary target variable y . We can represent this dataset as follows:

X_1	X_2	y
1	0	0
0	5	1
2	2	1
5	7	0
3	1	1

Table 3.1: Dataset to illustrate how classification trees are built.

Now, let's build a classification tree using this dataset. At each split, the algorithm chooses the feature and the splitting point that results in the best score for some splitting criterion (possible criteria are discussed further in this chapter).

In our example, we start with the dataset in table 3.1 and choose X_1 as the feature to split on. We split the dataset into two subsets: one where $X_1 > 1$ and one where $X_1 \leq 1$. The impurity of each subset can be calculated using a metric such as the Gini index or the entropy.

After the first split, the tree looks like this:

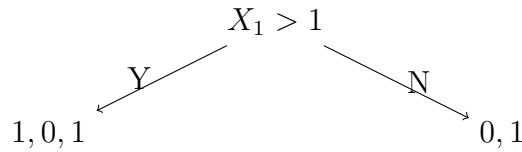


Figure 3.2: tree after one split

Now, let's say that we want to further split the left child (where $X_1 > 0$). We can do this by choosing a feature and a splitting point that results in the largest reduction in impurity. In this case, let's say that we choose X_2 as the feature and split the subset into two subsets: one where $X_2 > 2$ and one where $X_2 \leq 2$.

After the second split, the tree looks like this:

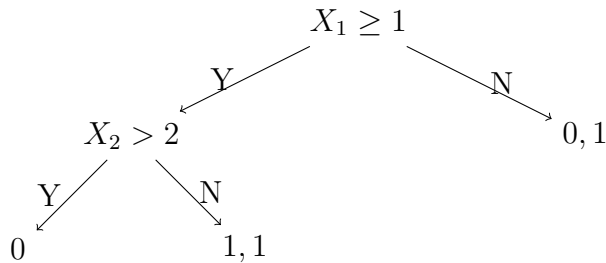


Figure 3.3: tree after two splits

After a final split, all leaves are pure and the resulting tree looks like this:

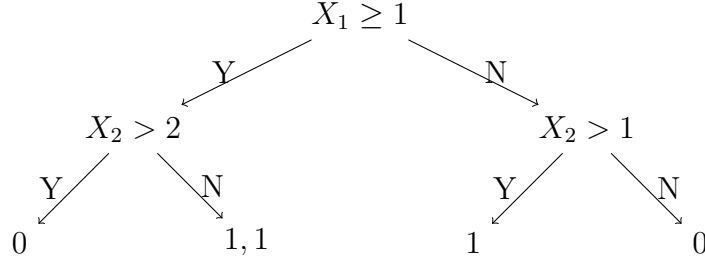


Figure 3.4: tree after three splits

3.1 How Decision Trees Work

A decision tree is a hierarchical tree structure with nodes representing decisions and leaves representing outcomes. The decision tree works by recursively partitioning the data into subsets, based on the values of the features. At each node, the feature that best splits the data into two subsets for some splitting criterion is chosen. This splitting process continues until all the leaf nodes are pure, i.e. all partitions only contain samples from a single class, or until some other stopping criterion is attained, e.g. the tree reaches a specified depth. Notable decision tree algorithms include: ID3 (Iterative Dichotomiser 3), C4.5 (successor of ID3), CART (Classification And Regression Tree) and Chi-square automatic interaction detection (CHAID). In this chapter we focus on CART and it is also the algorithm used in the experiments of this thesis, however ID3 and C4.5 follow similar approaches.

The CART algorithm works as follows:[13]

A decision tree partitions the feature space of a given set of training data, represented by $x_i \in R^n, i = 1, \dots, l$ and a label vector $y \in R^l$, into smaller regions such that the samples with the same labels or similar target values are grouped together. Let the data at node m be represented by Q_m with n_m samples. For each candidate split $\theta = (j, t_m)$ consisting of a feature j and threshold t_m , the data is partitioned into $Q_m^{\text{left}}(\theta)$ and $Q_m^{\text{right}}(\theta)$ subsets according to the following equations:

$$Q_m^{\text{left}}(\theta) = \{(x, y) \mid x_j \leq t_m\}$$

$$Q_m^{\text{right}}(\theta) = Q_m \setminus Q_m^{\text{left}}(\theta)$$

The quality of a candidate split is then evaluated using an impurity function or loss function $H()$, depending on whether the task is classification or regression, as follows:

$$G(Q_m, \theta) = \frac{n_m^{\text{left}}}{n_m} H(Q_m^{\text{left}}(\theta)) + \frac{n_m^{\text{right}}}{n_m} H(Q_m^{\text{right}}(\theta))$$

The parameters that minimize the impurity are selected according to the following equation:

$$\theta^* = \operatorname{argmin}_{\theta} G(Q_m, \theta)$$

The process is then recursed for the subsets $Q_m^{\text{left}}(\theta^*)$ and $Q_m^{\text{right}}(\theta^*)$ until the maximum allowable depth is reached or certain stopping criteria are met, such as $n_m < \min_{\text{samples}}$ or $n_m = 1$.

The decision tree is constructed using a greedy approach. At each node, the data is split according to some criterion.

Splitting criteria

There are several different splitting criteria that can be used in the CART algorithm, each with its own advantages and disadvantages.

The three criteria discussed in this section are illustrated in figure 3.5.

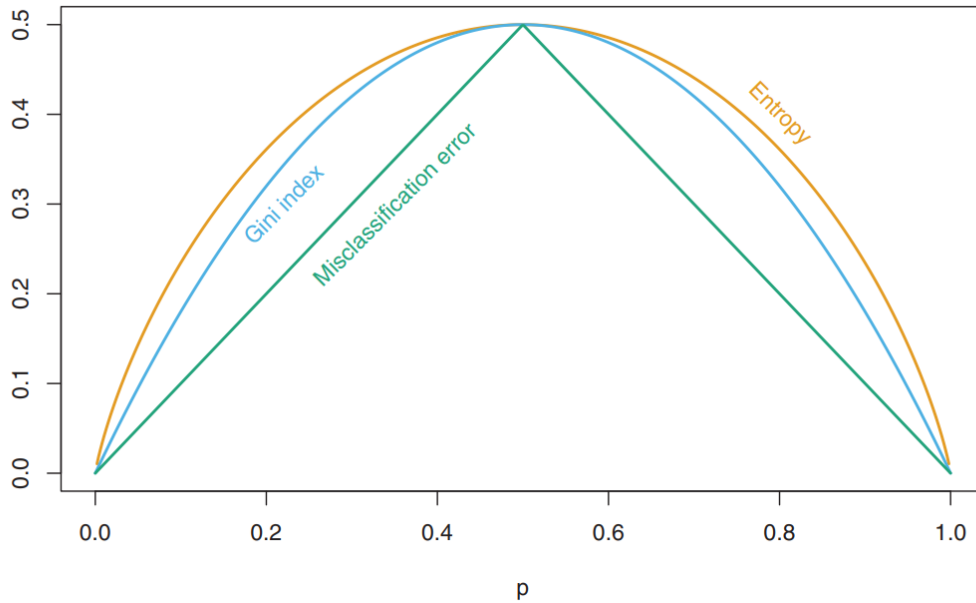


Figure 3.5: Node impurity measures for a classification problem with two classes. p is the proportion of class 1 observations in the node.

One commonly used splitting criterion is the Gini index, which measures the purity of the nodes resulting from a split. A pure node is one in which all the observations belong to a single class. The Gini index is calculated by summing the squared proportions of each class in the node, with a value of 0 indicating a pure node. The Gini index is preferred for its simplicity and ease of calculation, but it is sensitive to the presence of outliers in the data.

Another popular splitting criterion is the entropy measure, which is based on the concept of information theory. Entropy is calculated by summing the negative proportions of each class, multiplied by the logarithm of the proportions. The entropy measure is more robust to the presence of outliers, but it is more computationally complex to calculate than the Gini index.

A third splitting criterion is the classification error rate, which is simply the proportion of observations in the node that are misclassified. The classification error rate is easy to understand and calculate, but it tends to be biased towards splits that result in larger nodes, which can lead to overfitting of the model.

More formally:[13] In a classification task where the target takes on values $0, 1, \dots, K - 1$, the prediction for a given region is based on the proportion of each class within that region. For a node m , the proportion of class k observations in the node is calculated as follows:

$$p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k)$$

There are two common measures of impurity: the Gini impurity and the log loss or entropy. The Gini impurity is calculated as follows:

$$H(Q_m) = \sum_k p_{mk} (1 - p_{mk})$$

The log loss or entropy is calculated as follows:

$$H(Q_m) = - \sum_k p_{mk} \log(p_{mk})$$

And the classification error rate is calculated as follows:

$$H(Q_m) = 1 - \max_k p_{mk}$$

If node m is a terminal node, the prediction for that region is given by

$$\hat{y}_m = \operatorname{argmax}_k p_{mk}$$

3.2 Advantages and Disadvantages

Advantages:

Simplicity and clarity:

Decision trees are transparent models, which provide clear explanations for the decisions in the model. This is in contrast to black box models, such as artificial neural networks, which can be difficult to interpret. The transparency of decision trees makes them easy to understand and interpret, even for non-experts. In addition, decision trees closely mimic human decision making, making them intuitive and a useful tool for modeling human behavior and decisions.[5]

Versatility:

Decision trees requires little data preparation or preprocessing, since they are able to handle both numerical and categorical data and there is no need to normalize data or create dummy variables[5]. Additionally, they have built-in feature selection. Irrelevant features will not be used in splitting criteria and can easily be identified, conversely the most important attributes will be used near the root of the tree[10]. They can be trained on large datasets and classify large amounts of new data while keeping reasonable computation costs.

Limitations:

Decision trees can be sensitive to changes in the training data, meaning that small changes in the data can result in significant changes to the structure of the tree and the predictions it generates[5].

Building decision trees is a problem that is known to be NP-complete. This is true even for relatively simple concepts and under different definitions of optimality[8]. As a result, many decision tree learning algorithms are greedy and make the locally optimal choice at each step in the tree. These algorithms do not guarantee that the resulting decision tree will be the globally optimal one.

Decision trees can be prone to overfitting, which can lead to poor generalization performances. To avoid overfitting, various pruning mechanisms can be used to reduce the size of the tree.

3.3 Pruning

Pruning is a process used to simplify decision trees by removing or collapsing nodes that do not contribute significantly to the accuracy of the tree. There are two main types of pruning: pre-pruning and post-pruning.

Pre-pruning involves replacing a stop criterion in the induction algorithm with a criterion that limits the growth of the tree. This is done to prevent the tree from becoming too large and overfitting the training data. Pre-pruning can be more efficient than post-pruning because it does not require the induction of the entire training set. However, it can also suffer from the "horizon effect," where the tree is prematurely terminated by the stop criterion.

Post-pruning, also known as just "pruning," involves removing or collapsing nodes and subtrees to reduce the complexity of the tree. This can both reduce the size of the tree and improve the classification accuracy of unseen objects. However, it may also decrease the accuracy of the tree on the training set.

There are two main approaches to pruning: bottom-up and top-down. Bottom-up pruning starts at the bottom of the tree and works its way up, evaluating the relevance of each node. If a node is not relevant for classification, it is removed or collapsed into a leaf. This method is considered to be more thorough because it ensures that no relevant sub-trees are lost. Examples of bottom-up pruning methods include Reduced Error Pruning (REP), Minimum Cost Complexity Pruning (MCCP), and Minimum Error Pruning (MEP).

Top-down pruning, on the other hand, starts at the root of the tree and works its way down, evaluating the relevance of each node. If a node is not relevant for classification, it is removed or collapsed into a leaf. This method can result in the loss of entire sub-trees, even if they are relevant, because it prunes at inner nodes. An example of a top-down pruning method is Pessimistic Error Pruning (PEP), which has been shown to produce good results with unseen objects.

3.4 Summary

Decision trees are a powerful supervised learning method used in both regression and classification tasks. They mostly use greedy algorithms, that recursively divide the feature space into increasingly homogeneous subspaces. They are easy to interpret and explain and can handle both numerical and categorical data. However, they

can be prone to overfitting, which can lead to poor generalization of the data. Pruning the tree can help to improve the generalization of the model by removing the unnecessary branches.

Chapter 4

Random forests

The source for much of this chapter is chapter 15 of The Elements of Statistical Learning [4].

Random forests are a supervised learning method used in both regression and classification tasks. Random forests are an ensemble learning technique that combines the predictions of multiple decision trees. The main idea behind random forests is to construct a large number of decision trees, each of which is trained on a different subset of the data. The final prediction is made by averaging the predictions of the individual trees.

Random forests are considered to be one of the most powerful and accurate machine learning algorithms. They can handle both numerical and categorical data and are relatively robust to overfitting.

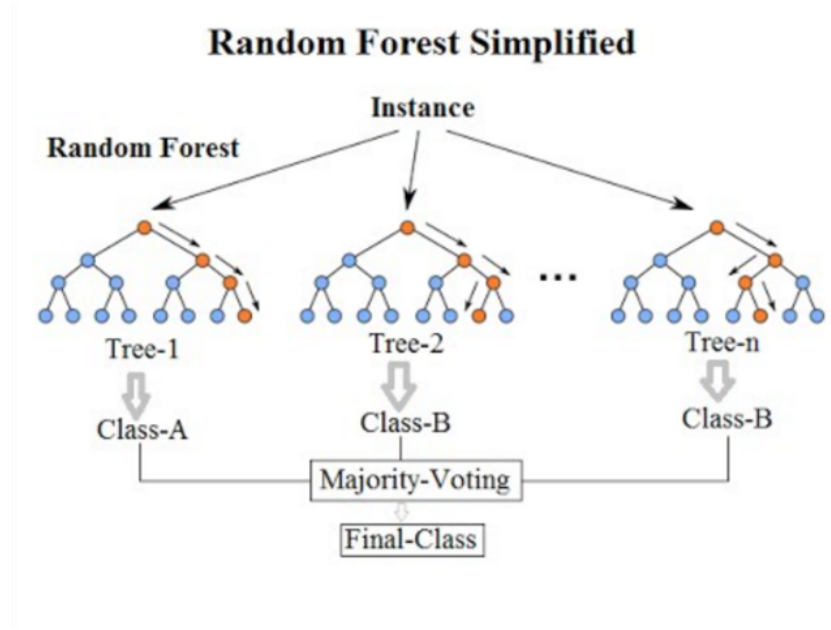


Figure 4.1: Diagram illustrating the use of a random forest.[1]

4.1 How Random Forests Work

Random forests work as follows:

Algorithm 1 Random Forest

- 1: **for** $m = 1$ to M **do**
 - 2: Draw a bootstrap sample Z of size N from the training data.
 - 3: Grow a random-forest tree T_m with Z .
 - 4: **end for**
 - 5: The random forest is the set of trees $\{T_m\}_{m=1}^M$.
 - 6: To make a prediction at a new point x :
 - 7: Let $C_m(x)$ be the class prediction of the m th random-forest tree.
 - 8: Then $C_{rf}(x) = \text{majority vote } \{C_m(x)\}_{m=1}^M$.
-

Some implementations, like the one we use for our experiments, don't use majority vote. Instead they have each tree assign a probability for each class, average those probabilities over all the trees and pick the class with the highest average probability.

Trees are high variance models, by averaging over many trees we can reduce the variance and improve the accuracy of the model.

Consider M independent identically distributed random variables, each with variance σ^2 . The variance of the average of those M random variables is $\frac{1}{M}\sigma^2$. Trees in a forest are not independent since their bootstrap dataset are built using the same underlying dataset. The variance of the average of M identically distributed, but not independent, random variables is

$$\phi\sigma^2 + \frac{1-\phi}{M}\sigma^2$$

Where ϕ is the pairwise correlation of the random variables. As M increases the second term decreases but the first stays constant. To improve the variance reduction of the bagging process we seek to reduce the pairwise correlation of the trees. For this reason, the random forest tree growing algorithm differs from the standard decision tree growing algorithm in that, when splitting a node, it selects the best splitting feature from a randomly chosen subset of features rather than considering all features. The size of this subset is typically the square root of the total number of features.[4]

Bagging or bootstrap aggregating

Bagging is an essential component of the random forest algorithm. To create a bagged model, we create multiple bootstrapped samples of the training data. A bootstrapped sample is created by randomly sampling observations from the original training set with replacement. This means that, in a single bootstrapped sample, some observations may be sampled more than once, while others may not be included at all.

To illustrate the process of bootstrapping, imagine we have a training dataset with 6 observations, each with three features and a label:

Observation	x_1	x_2	x_3	y
1	1.2	3.4	5.6	7.8
2	9.1	2.3	4.5	6.7
3	8.9	7.8	6.7	5.6
4	4.5	3.4	2.3	1.2
5	9.8	8.7	7.6	6.5
6	5.4	4.3	3.2	2.1

To create a bootstrapped sample, we randomly choose observations from the original dataset with replacement.

For example, we might choose the following observations for our bootstrapped sample:

Observation	x_1	x_2	x_3	y
5	9.8	8.7	7.6	6.5
2	9.1	2.3	4.5	6.7
5	9.8	8.7	7.6	6.5
4	4.5	3.4	2.3	1.2
3	8.9	7.8	6.7	5.6
2	9.1	2.3	4.5	6.7

Notice that we have included some observations more than once (e.g., 5) and excluded others (e.g., 1). This is what it means to sample with replacement. We can then train a decision tree model on this bootstrapped sample and use it to make predictions. We can repeat this process multiple times to create multiple bootstrapped samples and train multiple decision tree models. Finally, we can average the predictions of all the models or use a majority vote to make a final prediction.

The use of bagging gives random forests an important feature: out of bag (oob) samples. An observation from the training set is an unseen datapoint for trees grown on bootstrap samples in which the observation did not appear. The set of trees that were not trained on a given observation form a random forest predictor for said sample. We can thus compute a misclassification rate for oob samples and use this oob error instead of k-fold cross validation.

Bootstrapping is a statistical method that can be used for a variety of purposes beyond random forests and other ensemble machine learning models. One other common use of bootstrapping is tuning hyperparameters for model selection. Overall, bootstrapping is a versatile and widely used statistical method that can be applied in many different contexts.

4.2 Advantages and Disadvantages

Random forests keep some of the advantages of decision trees, especially their versatility: They require just as little data preparation or preprocessing, and also have built in feature selection.

Additionally they alleviate some of the downsides of trees: They are less sensitive to small changes in the training data and are relatively robust to overfitting.

However, random forests have some disadvantages. They are more computationally expensive, and the models are more difficult to interpret and explain than decision trees.

4.3 Summary

Random forests are a powerful supervised learning algorithm used in both regression and classification tasks. They use bagging to build many distinct trees and add randomness to the tree growing algorithm. They are versatile, relatively robust to overfitting and generally perform better than decision trees. However, they can be computationally expensive, and the results aren't as transparent as decision trees.

Chapter 5

IB theory of decision trees and random forests

Representations in the information plane

To evaluate a model in the IB framework we need to place the representations of input data that the model creates in the information plane and measure its distance to the IB curve, i.e. the curve that maximises mutual information between the representation and the target variable for a given level of complexity.

Tytgat (2021)[3] defines intermediate representations T for every level of a tree defined by the nodes reached at that level (which corresponds to a partition of the feature space). In our replication of Tytgat (2021)'s (section 6.1) experiments we only use the final representation $\hat{Y}$ defined by the classification performed by the tree, since no conclusions were drawn from the intermediate representations.

To clarify what we mean by 'final representation', consider a sample with features x , it is represented by $\hat{y}$ if it is classified by the tree in class $\hat{y}$. Two values need to be computed for $\hat{Y}$:

- $I_Y = I(\hat{Y}; Y) = H(\hat{Y}) - H(\hat{Y}|Y)$: The mutual information of the classification by the tree and the actual classification.
- $I_X = I(\hat{Y}; X) = H(\hat{Y}) - H(\hat{Y}|X) = H(\hat{Y})$ (since $\hat{Y}$ is fully determined by X and thus $H(\hat{Y}|X) = 0$) : The mutual information of the classification by the tree and the input features.

The entropy and conditional entropy of $\hat{Y}$ are computed as follows:

$$\begin{aligned}
H(\hat{Y}) &= - \sum_{\hat{y} \in \mathcal{Y}} p(\hat{y}) \log p(\hat{y}) \\
H(\hat{Y}|Y) &= - \sum_{y \in \mathcal{Y}} p(y) H(\hat{Y}|y) \\
&= \sum_{y \in \mathcal{Y}} p(y) \sum_{\hat{y} \in \mathcal{Y}} p(\hat{y}|y) \log p(\hat{y}|y) \\
p(\hat{y}) &= \frac{\text{number of samples classified as } \hat{y}}{\text{total number of samples}} \\
p(\hat{y}|y) &= \frac{\text{number of samples classified as } \hat{y}, \text{ actually in } y}{\text{total number of samples in } y}
\end{aligned}$$

And the entropy of the target variable Y is necessary to draw the IB curve and is computed as:

$$H(Y) = - \sum_{y \in \mathcal{Y}} p(y) \log p(y)$$

Where $\mathcal{Y}$ is the set of classes for a classification task. Computing $p(y)$, $p(\hat{y})$ and $p(\hat{y}|y)$ empirically can be done with unseen data from a test set or with training data. The effect of using seen or unseen data is explored in the experiments in section 6.1.

For our new experiments on random forest we use the same final representation $\hat{Y}$ as for trees and define a new intermediate representation T defined by the leaf by which an observation is classified in each tree of the forest. Let (x, y) be an observation, its forest representation t is defined by

$$t = \{L_1, L_2, \dots, L_M\}.$$

Where L_t identifies a leaf in tree t and M is the number of trees in the forest. We can observe how this representation T evolves in the IB plane as we add more trees to a random forest. This last point parallels how the evolution of representations in DNNs through learning epochs has been analysed in many previous works (e.g.[14]).

This representation T describes how the feature space is divided by each tree of the forest. The partitions and classifications performed by different trees of

the forest are not independent variables and treating a list of such partitions as a variable may cause difficulties, however information theory and the IB method in particular are well suited to analyse such a representation as measures of mutual information and entropy take the redundancy in T into account.

We compute the IB coordinates of T in a similar way as for $\hat{Y}$

- $I_Y = I(T; Y) = H(T) - H(T|Y)$: The mutual information of the forest representation and the output variable.
- $I_X = I(T; X) = H(T) - H(T|X) = H(T)$ (since T is fully determined by X and thus $H(T|X) = 0$) : The mutual information of the forest representation and the input features.

The entropy and conditional entropy of T are computed as follows:

$$H(T) = - \sum_{t \in \mathcal{T}} p(t) \log p(t)$$

$$\begin{aligned} H(T|Y) &= - \sum_{t \in \mathcal{T}} p(t) H(t|y) \\ &= \sum_{t \in \mathcal{T}} p(t) \sum_{t \in \mathcal{T}} p(t|y) \log p(t|y) \end{aligned}$$

$$\begin{aligned} p(t) &= \frac{\text{number of samples represented as } t}{\text{total number of samples}} \\ p(t|y) &= \frac{\text{number of samples represented as } t \text{ and labeled as } y}{\text{total number of samples labeled } y} \end{aligned}$$

Where $\mathcal{T}$ is the set of all random forest representations.

The random forest representation T and its associated mutual information values are used in section 6.2. We compute $p(t)$ and $p(t|y)$ empirically using unseen data from a test set.

Chapter 6

Empirical analysis

6.1 Replicated experiments

In this section we replicate and explore further two of the experiments from Tytgat 2021 Masters thesis[3]. Specifically, we replicate « EXPERIMENT II : The IB curve contains sub-optimal representations » and « EXPERIMENT III : Better representations than IB optimal solution » from section 5.4.2 of the previous thesis, with particular attention to the data sets used to assess IB optimally.

In the first experiment we train one model on the full dataset and another model on a partial dataset. Both models can perfectly classify data they were trained on and appear IB optimal when their information plane coordinates I_Y and I_X are computed on (part of) their training data. Since the IB curves computed using different parts of a dataset coincide when the output distribution of those parts is equal, it was observed in [3] that the model trained on the partial dataset falls on the IB curve of the full set and should thus be optimal, even though it can't classify the full set perfectly.

As we demonstrate in section 6.1.2 this apparent contradiction depends on the data set used to compute I_Y and I_X .

In the second experiment we use pruning to improve a model trained on a training set. The pruned model generalises better than a non-pruned model trained on the same set. In [3] it was shown that the pruned model is not IB optimal and since the non-pruned model is IB optimal according to the preceding experiment of that thesis, it was concluded that a better model can be evaluated as worse by the IB framework.

In our replication of this experiment, section 6.1.3, we show that the pruned model is indeed not IB optimal, but if I_Y and I_X are computed for both models using the test set, e.i. the set for which the pruned model performs better, the pruned model is closer to IB optimally than the non-pruned one.

Like in the original thesis, we perform both experiments on two different classification tasks:

- a toy dataset consisting of a 150 by 150 grid of black or white pixels. Each sample has two features: its x and y coordinates in the grid and a target value of 0 or 255 corresponding to the color of the pixel.
- the MNIST handwritten digits dataset : 70000 grids of 28 by 28 pixels representing digits from 0 to 9. The pixels are the features taking values between 0 and 255 and the target values are the represented digits.

For both tasks, we split the datasets in two, with 6/7 of the full sets (S_{full}) as training sets (S_{train}) and the remaining 1/6 as test sets (S_{test}). We use the `DecisionTreeClassifier` class from the Sci-kit learn library¹ for the trees with default parameters unless stated otherwise. Mutual information and entropy values are computed using functions from the NPEET package².

6.1.1 IB curves for deterministic classification tasks

The quality of a model in the IB framework is measured by its distance to the IB curve in the information plane. As we consider deterministic classification tasks the curve is piece-wise linear with an elbow at $(H(Y), H(Y))$, $I_Y = I_X$ from 0 to $H(Y)$ and $I_Y = H(Y)$ for $I_X \geq H(Y)$. Thus the only value needed to draw the curve is $H(Y)$. We can compute this value using S_{full} , S_{train} and S_{test} , however for well balanced datasets the three curves coincide as the entropy of Y only depends on its distribution. This is demonstrated on Figure 6.1.

¹<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

²<https://github.com/gregversteeg/NPEET>

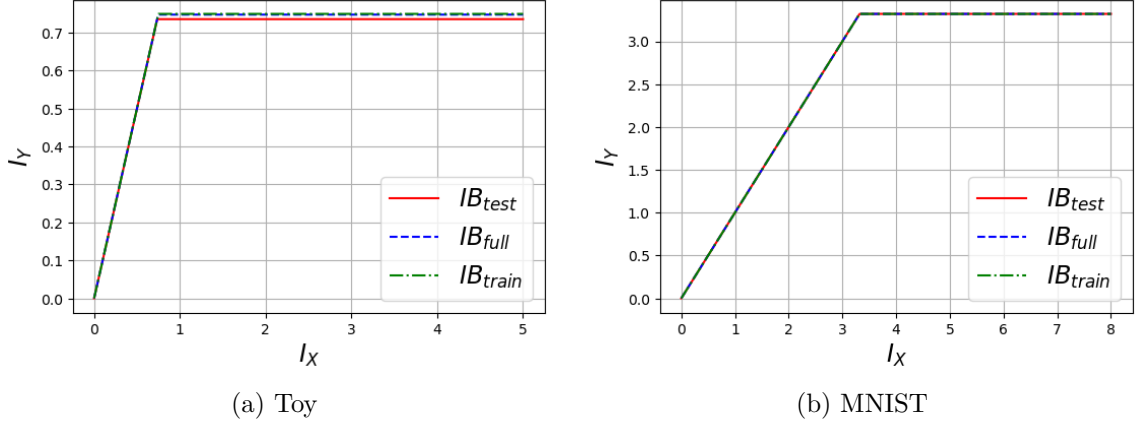


Figure 6.1: *IB curves computed for the toy and MNIST dataset*

6.1.2 Experiment 1: The IB curve contains suboptimal representations ?

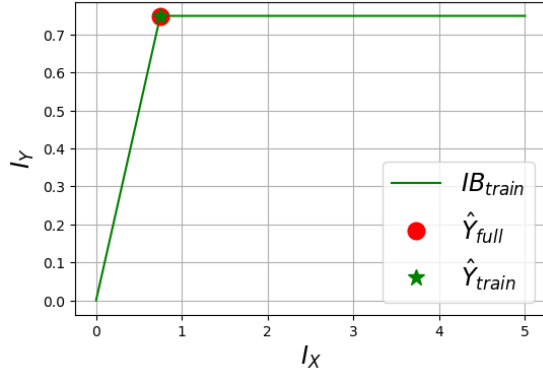
In this experiment we train two trees, the first (M_{full}) on the full data set S_{full} and the second (M_{train}) on the training set S_{train} . Both trees are trained to perfectly fit their respective training set.

Figure 6.2 shows how they perform in the IB framework depending on the data used to compute I_Y and I_X . M_{full} is represented with a red dot and M_{train} with a green star. When we compute I_Y and I_X using S_{train} as shown in (a) and (b), both models appear IB optimal. This is expected as they both achieve perfect classification of S_{train} .

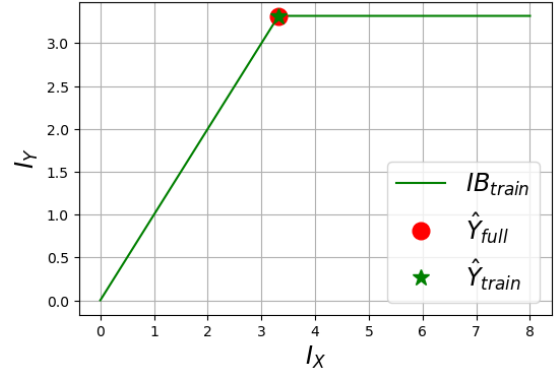
When computed with S_{full} as shown in (c) and (d), M_{full} remains IB optimal while M_{train} no longer is. This is consistent with their correct classification rates for the full set : M_{full} scores 100% for toy and MNIST and M_{train} 99,8% for toy and 98,19% for MNIST.

Finally, when we compute I_Y and I_X using S_{test} as shown in (e) and (f), M_{full} remains IB optimal and a perfect classifier and M_{train} scores even lower in the IB framework as well as for correct classification rate with 98,66% for toy and 87,35% for MNIST.

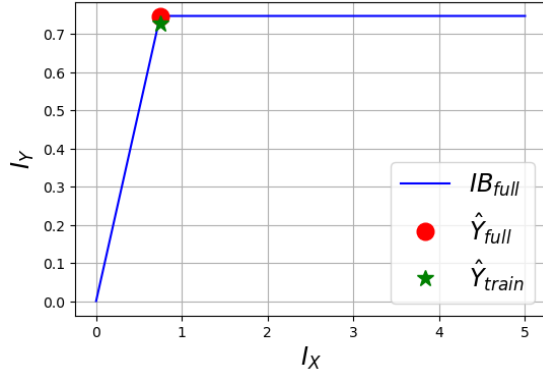
Both models are IB optimal when measured with their respective training sets.



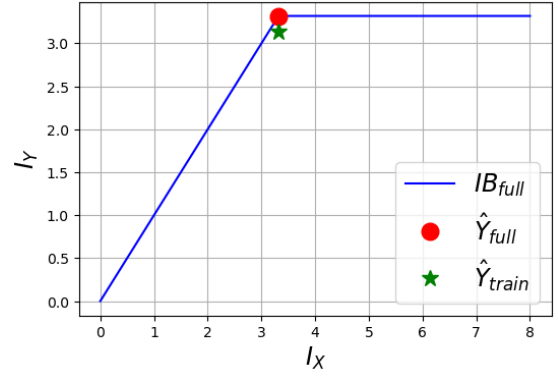
(a) Toy



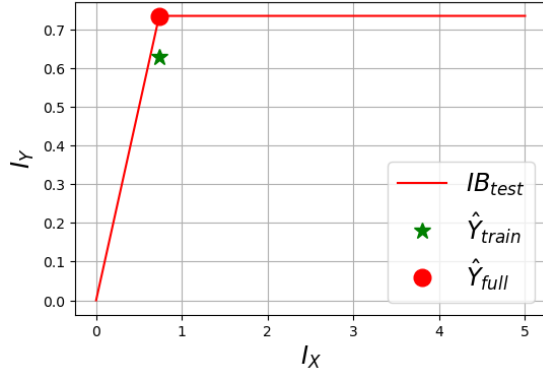
(b) MNIST



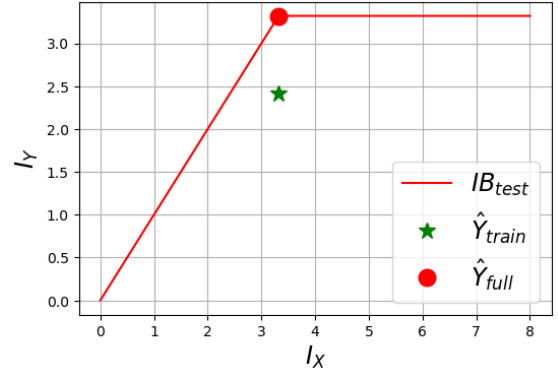
(c) Toy



(d) MNIST



(e) Toy



(f) MNIST

Figure 6.2: Models trained on S_{train} (green stars) and on S_{full} (red dot) compared in the information plane using different parts of the datasets. I_Y and I_X are computed using S_{train} in (a) and (b), S_{full} in (c) and (d) and S_{test} in (e) and (f).

6.1.3 Experiment 2: Better representations than IB optimal solution ?

In this experiment we train a third tree (M_{pruned}) on the training set S_{train} using complexity cost pruning. The α value is selected using a 5-folds cross validation procedure. We then compare this new model to M_{train} from the previous experiment. For the S_{test} data set M_{pruned} achieves a correct classification rate of 98,88% on the toy task and 87,92% on the MNIST task and M_{train} scores 98,66% for toy and 87,35% for MNIST.

On Figure 6.4 we compare the two models in the information plane with M_{pruned} as a red ball and M_{train} as a green star. I_Y and I_X are computed using S_{test} . We can see that M_{pruned} is slightly closer to the IB optimum than M_{train} .

M_{pruned} performs better according to the IB framework and the correct classification rate.

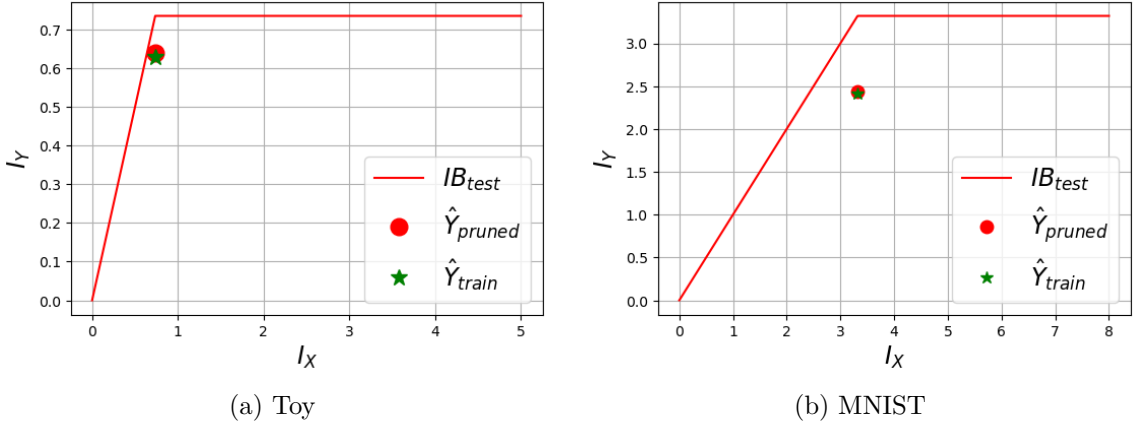


Figure 6.3: *Pruned models (red dots) compared to non-pruned (green star) models in the information plane.*

6.1.4 Discussion

Contrary to what was demonstrated in two of the experiments in Tytgat (2021)[3], we find no inconsistencies between the IB framework and the correct classification rate for deterministic classification tasks, provided I_Y and I_X and the correct classification rate are computed with the same data-set.

The preceding experiments and analysis do not depend on the type of algorithm used and could be performed with any supervised classification method. Evaluating a model in the IB framework as is done here essentially boils down to comparing

the mutual information of the classification performed by the model and the actual classification ($I(\hat{Y}; Y)$) to the entropy of the actual classification ($H(Y)$). Intuitively it seems logical that if $\hat{Y}$ is closer to Y in terms of fraction of misclassifications, their mutual information will be higher.

6.2 Experiments on Random Forests

One common way the IB method is used to study deep neural networks is by analyzing how the layer representations evolve in the information plane, over the stochastic gradient descent learning epochs. In the following experiment, we apply a similar analysis to random forests. As shown in previous experiments, there is limited interest in analyzing solely the final representation or classification of a model in the IB plane as it is independent of the type of model used and is always consistent with the accuracy of the model. Therefore, we analyze an intermediate representation T that is particular to random forests. This representation was introduced in section 5. We analyze how this representation evolves in the IB plane as we add more trees to the forest. We repeat this procedure for random forests with different maximum depths.

The experiments from this section are performed on the MNIST handwritten digits dataset (70000 grids of 28 by 28 pixels representing digits from 0 to 9. The pixels are the features taking values between 0 and 255 and the target values are the represented digits). 6/7 of the dataset are sampled as training set and the remaining 1/7 is used as test set.

The RandomForestClassifier class from the Sci-kit learn library³ is used for the random forests with default parameters⁴ unless stated otherwise. Mutual information and entropy values are computed using functions from the NPEET package⁵.

³<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

⁴We use `max_features='sqrt'`, which is the default parameter in the latest version of Sci-kit learn, some older versions have a different default value for `max_features`.

⁵<https://github.com/gregversteeg/NPEET>

6.2.1 The random forest representation in the information plane

Like for the previous experiments we draw the IB curves for the sets and once again we see that the curves coincide for the training set, the test set and the full set since $H(Y)$ is almost identical for the three sets. We also plot the prediction $\hat{Y}_{tr}$ and the forest representation T_{tr} (For a single tree T takes the value of a leaf identifier.) of a single decision tree with default parameters in the information plane.

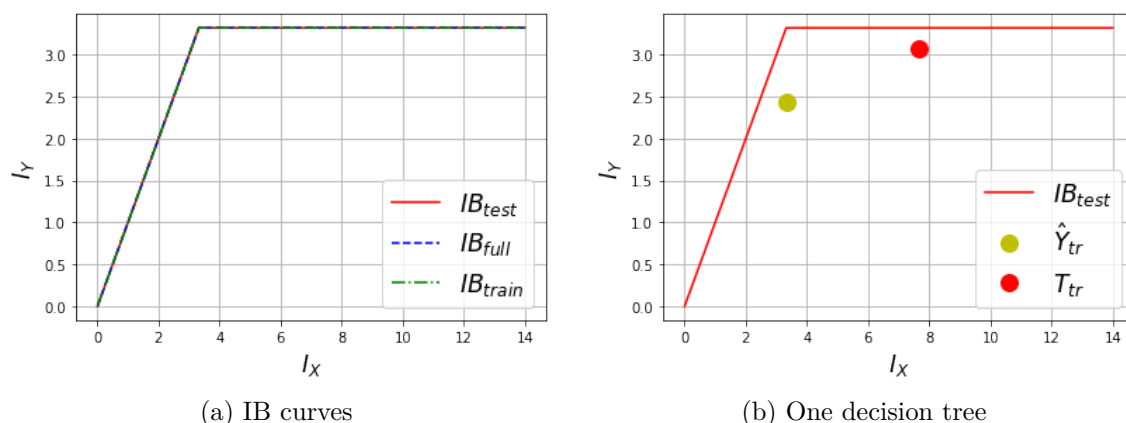


Figure 6.4: *IB curves for the MNIST dataset (a) and forest representation T and final representation $\hat{Y}$ for a single tree.*

We can see on figure 6.4(b) that a lot of information is lost when going from T_{tr} to Y_{tr} , most of that information is irrelevant to the classification task, since the difference is greater along the I_X axis, but some relevant information is lost as well.

We now perform the same experiment on four random forests: rf_2 with a maximum depth of 2, rf_5 with a maximum depth of 5, rf_{10} with a maximum depth of 10 and rf_{full} where the trees are allowed to grow to their full height. For each forest we plot the predictions $\hat{Y}$ and the forest representations T in the information plane as we add more trees to the forest. Forests rf_5 , rf_{10} and rf_{full} are allowed to grow from 1 to 100 trees and rf_2 from 1 to 200 trees. The results are shown in figure 6.5.

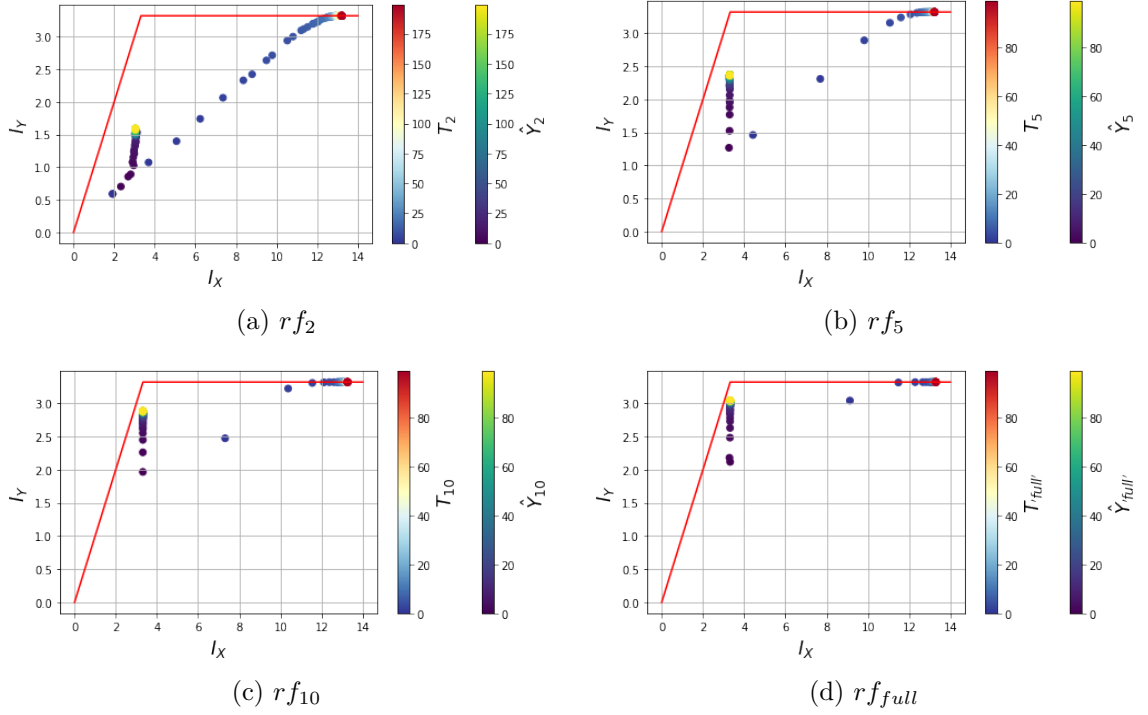


Figure 6.5: *Evolution of random forest representations T and predictions $\hat{Y}$ visualised in the information plane, as more trees are added to the forests. The color gradients illustrate the number of trees in a forest. Maximal depths for the trees of the forests are : (a) 2, (b) 5, (c) 10 and (d) no maximal depth.*

Like in the case with a single tree, a lot of information is lost between T and $\hat{Y}$, some of which is relevant to the classification.

Looking at the representations T , we see that in all four forests $I(T; Y)$ reaches its maximal value. It seems that with enough trees the random forests capture all the relevant information $I(Y; X)$ in the representation T , but loses some of that information when turning T into a prediction $\hat{Y}$. Looking at numerical values of $I(T; X)$ in table 6.1, we see that they approach $H(X)$, which means they capture almost all the information of X . The random forest representation T can thus be described as good in terms of relevance, but quite poor in terms of compression of X .

$I(T_2; X) =$	13.22011274133556
$I(T_5; X) =$	13.2404582873021
$I(T_{10}; X) =$	13.25973155908619
$I(T_{full}; X) =$	13.287712379549445
$H(X) =$	13.287712379549445

Table 6.1: $I(T; X)$ for complete random forests.

We now look at how many trees each forest needs to reach the maximum value of $I(T; Y)$, i.e. the number of trees for which $I(T; Y) = H(Y)$. Let us call that number of trees m_i for forest i , where $i = \{2, 5, 10, full\}$:⁶

- $m_2 = 82$
- $m_5 = 21$
- $m_{10} = 7$
- $m_{full} = 5$

We might expect the accuracy to stop improving when adding more trees than m_i . However this is not the case as shown in table 6.2.

Model	Accuracy with m_i trees	Max accuracy
rf_2	0.6232	0.6466
rf_5	0.8351	0.8538
rf_{10}	0.913	0.9455
rf_{full}	0.9223	0.9688

Table 6.2: Accuracy of the four models with m_i trees compared to the maximal accuracy attained when adding more trees.

In order to verify that the representations T contain all the relevant information about Y with m_i trees in the forests, we train models⁷ with representations T as features and Y as labels. If these models achieve perfect accuracy for their training set, then we know that a perfect map exists and that T does indeed contain all the relevant information. The results of this test are shown in table 6.3.

⁶The exact numbers given above vary a lot with the random aspects of the experiment: the randomness in the random forest algorithm and splitting the dataset into a training set and a test set. For example, in some runs $I(T_2; Y)$ took more than 100 trees to converge. However, the qualitative conclusions remain the same.

⁷We use trees to learn the maps between T and Y , but the type of model used is irrelevant.

Model	m_i trees	$m_i - 1$ trees
rf_2	1.0	0.9999
rf_5	1.0	0.9999
rf_{10}	1.0	0.9998
rf_{full}	1.0	0.9999

Table 6.3: Accuracy on training sets of decision trees trained on (T, Y) for T with just enough trees to reach maximal $I(T; Y)$ (m_i) and with one tree short of reaching maximal $I(T; Y)$ ($m_i - 1$).

These results confirm that the random forest representation T , for a forest with m_i trees, contains enough relevant information to achieve perfect classification. The random forest algorithm is unable to use all that information and can increase its accuracy by adding redundant information to T , i.e. by adding more trees.

6.2.2 Discussion

Our analysis of random forests through the Information Bottleneck theory revealed the following: With enough trees the random forest representation T captures all the relevant information $I(Y; X)$, but some of that information is lost when turning T into predictions $\hat{Y}$. T also captures a lot of irrelevant information and if we keep adding trees we get $I(T; X) = H(X)$. Thus, T is a highly relevant but poorly compressed representation of X .

We can confirm that, for a deterministic classification task, when T captures all the relevant information, a perfect map exists between T and Y .

When adding more trees after $I(T; Y)$ reaches its maximum, both $I(T; X)$ and the accuracy of the model keep increasing. This goes against the idea from IB theory according to which better compression leads to better accuracy, since the relevance I_Y of T stays constant, the compression I_X gets worse, and the accuracy gets better.

Chapter 7

Nondeterministic classification

Until now we have only considered cases where a deterministic map $f(X) = Y$ exists, Y is fully determined by X and $I(Y; X) = H(Y)$. In those cases the IB curve is piece-wise linear and can be drawn using only $H(Y)$. We now turn our attention to cases where the map between X and Y is stochastic.

7.1 Nondeterministic classification tasks

A deterministic classification task is one in which the class of an input sample can be accurately predicted with certainty, based on the input features. In other words, given the same input features, the correct class prediction will always be the same.

On the other hand, a nondeterministic classification task is one in which the class of an input sample cannot be accurately predicted with certainty, based on the input features. This could be due to a variety of factors, such as noise or uncertainty in the data, or the complexity of the problem itself. In a nondeterministic classification task, the classifier may produce different predictions for the same input sample on different runs, or may have a certain probability of predicting the correct class.

The MNIST handwritten digits dataset is a deterministic classification task: each of the 70000 grids of 28 by 28 pixels is labeled as a single digit from 0 to 9. However we could modify this dataset to make it nondeterministic: by adding several copies of grids of 28 by 28 pixels that don't represent a single digit clearly and labeling those copies differently. For example, consider the two digits in figure 7.1. Both are labeled 4 but might be read as 9. If we add copies of those images labeled 9 to the dataset, the classification task becomes nondeterministic. We could also create new ambiguous images and add several copies of those with different labels. By imagining the full space of grids of 28 by 28 pixels, we notice that most

points in that space can't be clearly labeled as a single digit.

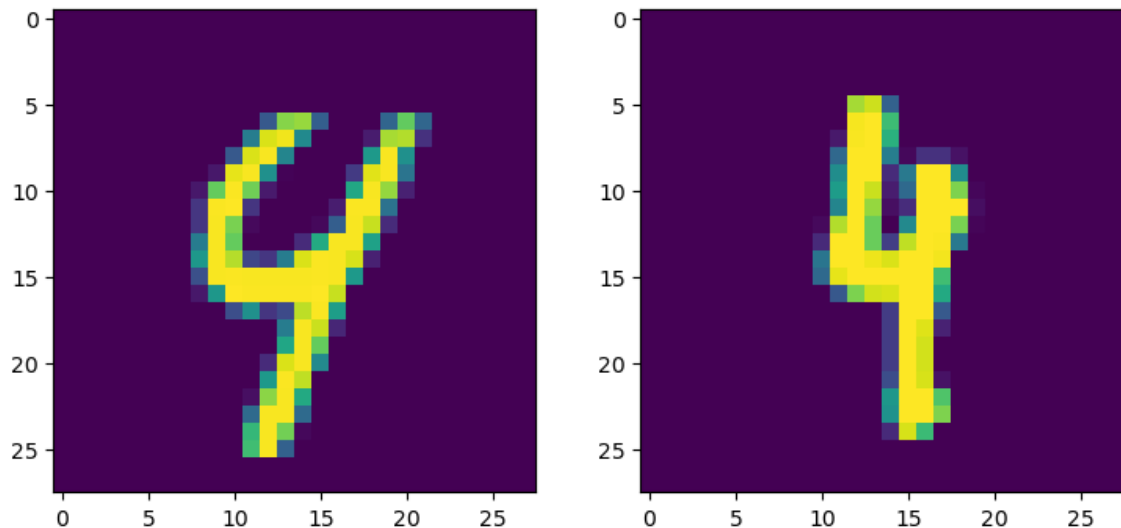


Figure 7.1: *Examples of ambiguous handwritten digits from the MNIST dataset.*

Many real world classification tasks can be made nondeterministic by adding ambiguous points of the feature-space to the dataset. However, to calculate the IB curve we need to know the full joint distribution $p(x, y)$. Therefore, we use a simple synthetic classification problem, similar to the toy dataset used in section 6.1. The feature space is a 2D, 150 by 150 grid and the labels are binary. Instead of giving each point a unique label, we define its joint probability distribution $p(x, y)$. $p(x, y)$ and $p(y | x)$ are shown on figure 7.2 and we chose a uniform distribution for $p(x)$.

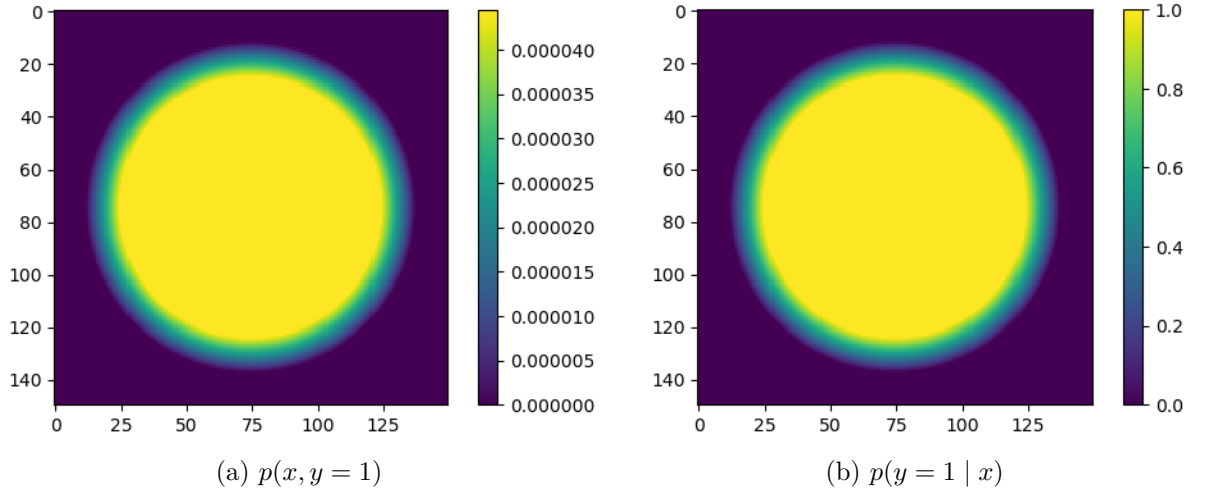


Figure 7.2: Joint probability distribution $p(x, y = 1)$ (a) and conditional probability distribution $p(y = 1 | x)$, where $x \in X$ are coordinates in a 2D grid and $y \in Y$ are binary target values.

In figure 7.2 the purple and yellow area represents points that can be labeled $y = 0$ and $y = 1$ respectively, with certainty. The green area represents points that can be labeled as either $y = 0$ or $y = 1$ with some finite probability.

7.2 Solving the IB problem

For the classification task defined above, we now want to find the best representation T of X defined by a stochastic map $p(t | x)$. Recall the optimisation problem introduced in chapter 2

$$\min_{p(t|x), p(y|t), p(t)} I(X; T) - \beta I(T; Y)$$

and the equations that solve the problem

$$\begin{cases} p(t | x) = \frac{p(t)}{Z(x; \beta)} \exp(-\beta D_{KL}[p(y | x) \| p(y | t)]) \\ p(t) = \sum_x p(t | x) p(x) \\ p(y | t) = \sum_x p(y | x) p(x | t) \end{cases}$$

where $Z(x; \beta)$ is a normalization factor.

We can use these equations to find a locally optimal $p(t | x)$ with the following algorithm:

Algorithm 2 The Information bottleneck (IB) method.[15]

Require: $p(x, y), \beta \geq 0$

```
1: Initialize:  $q^{(0)}(t | x)$ 
2:  $q^{(0)}(t) = \sum_x p(x)q^{(0)}(t | x)$ 
3:  $q^{(0)}(y | t) = \frac{1}{q^{(0)}(t)} \sum_x p(x, y)q^{(0)}(t | x)$ .
4:  $n = 0$ 
5: while not converged do
6:    $n = n + 1$ 
7:    $q^{(n)}(t | x) = \frac{q^{(n-1)}(t)}{Z(x, \beta)} \exp \left[ -\beta D_{\text{KL}} \left[ p(y | x) | q^{(n-1)}(y | t) \right] \right]$ 
8:    $q^{(n)}(t) = \sum_x p(x)q^{(n)}(t | x)$ 
9:    $q^{(n)}(y | t) = \frac{1}{q^{(n)}(t)} \sum_x q^{(n)}(t | x)p(x, y)$ 
10: end while
```

In the algorithm above distributions represented with a p are given and fixed and those represented with a q vary from iteration to iteration. Once the algorithm converges we can compute $I(T; X)$ and $I(T; Y)$.

7.2.1 Implementation

We found no usable implementation of algorithm 2 and thus had to make our own. This tool is presented as part of this Master's thesis and is available on Github¹. In our implementation we make the following implementation choices:

- we check convergence by computing the difference between $q^{(n)}(t | x)$ and $q^{(n-1)}(t | x)$ in the following way:

$$difference = \sum_{t,x} |q^{(n)}(t | x) - q^{(n-1)}(t | x)|$$

If this difference falls below a specified threshold ϵ the iterations stop.

- The normalisation factor $Z(x; \beta)$ ensures that $q^{(n)}(t | x)$ remains a correct probability distribution : $\sum_t q^{(n)}(t | x) = 1$. We impose this constraint in a different way :

$$a_x(t) = q^{(n-1)}(t) \exp \left[-\beta D_{\text{KL}} \left[p(y | x) | q^{(n-1)}(y | t) \right] \right]$$
$$q^{(n)}(t | x) = \frac{a_x(t)}{\sum_t a_x(t)}$$

¹https://github.com/MartinDeVroey/IB_solver

- Once we converge to a distribution $p(t | x)$ (and $p(t)$ and $p(y | t)$), we compute $I(T; X)$ and $I(T; Y)$ as follows:

$$I(T; X) = \sum_{x,t} p(t | x) p(x) \log \left[\frac{p(t | x)}{p(t)} \right]$$

$$I(T; Y) = \sum_{y,t} p(y | t) p(t) \log \left[\frac{p(y | t)}{p(y)} \right]$$

By plotting the values $I(T; X)$ and $I(T; Y)$ obtained for different values of β , in the information plane, we get the IB curve:

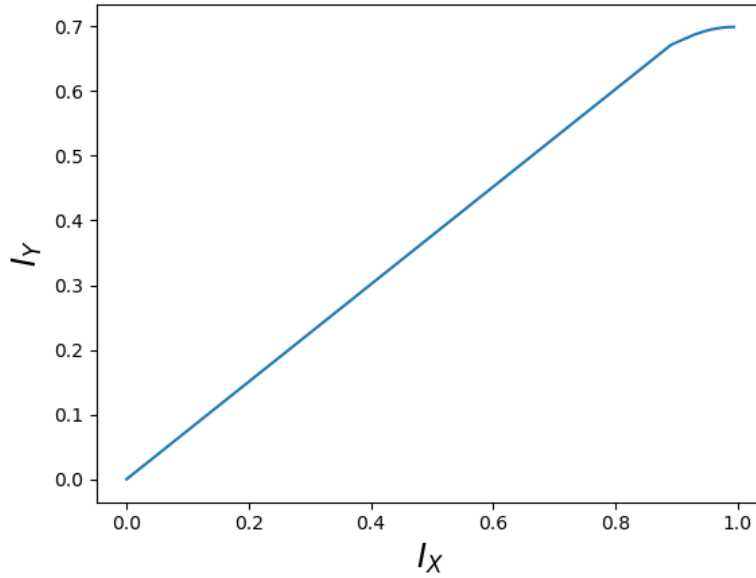


Figure 7.3: *IB curve obtained for the distribution $p(x, y)$ displayed in figure 7.2.*

7.3 Experiments

7.3.1 Creating training sets

To train models we need to draw samples from the distribution $p(y, x)$. Since we chose a uniform distribution for $p(x)$, we can simply draw elements from X at random with replacement. We then label the drawn elements 1 or 0 according to the conditional distribution $p(y | x)$. Using this process, we generate two sets of different sizes, Set 2 is 8 times as large as Set 1. We show the two training sets on figure 7.4.

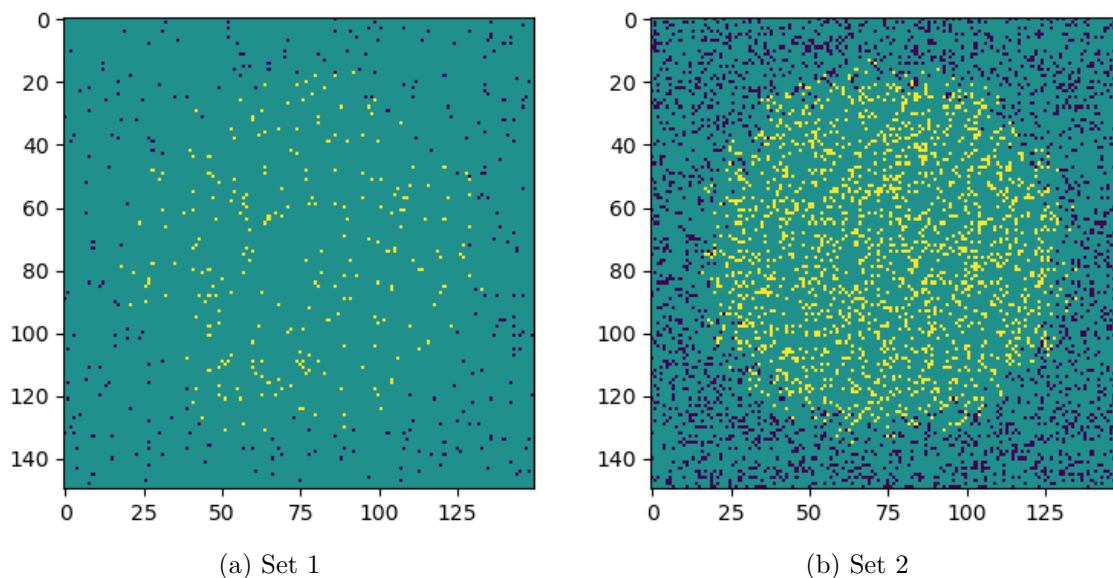


Figure 7.4: *Two training sets. Purple and yellow pixels are elements of the training sets labeled 0 and 1 respectively. Blue pixels are elements that are not in the training sets. Some pixels may appear several times in the training sets since they were drawn with replacement.*

7.3.2 Evaluating the models

Using the training sets, we train four models: one decision tree and one random forest on each set. We name the models trained on Set 1 tr_1 and rf_1 and those trained on Set 2 tr_2 and rf_2 . tr_1 and tr_2 are instances of the `DecisionTreeClassifier` class with default parameters and rf_1 and rf_2 are instances of the `RandomForestClassifier` class with default parameters, both classes come from the Sci-kit learn library².

We then have the models make a prediction for every point of the feature space X . This creates a map $p(\hat{y}|x)$ for each model. Those maps are deterministic, meaning that $p(\hat{y}|x)$ only contains ones and zeros. An example of such a map is given on figure 7.5.

²<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

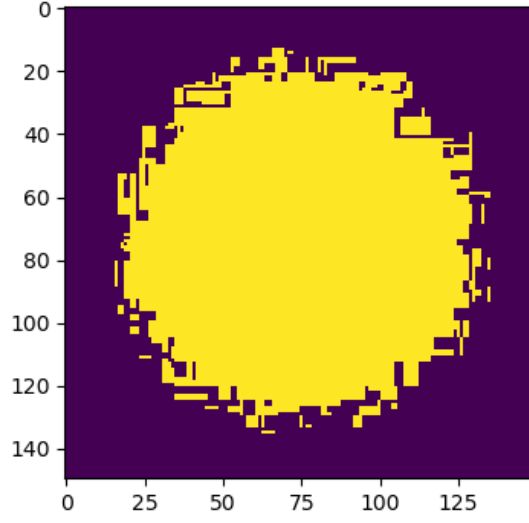


Figure 7.5: *Deterministic map $p(\hat{y}|x)$ generated by tr_2 .*

Using these maps $p(\hat{y} | x)$ and $p(x)$, $p(y)$ and $p(y, x)$ which are given, we compute the following distributions:

$$p(\hat{y}) = \sum_x p(\hat{y} | x)p(x)$$

$$p(y | \hat{y}) = \sum_x p(y | x)p(x | \hat{y}) = \sum_x p(y, x) \frac{p(\hat{y} | x)}{p(\hat{y})}$$

and then the values for $I(\hat{Y}; X)$ and $I(\hat{Y}; Y)$:

$$I(\hat{Y}; X) = \sum_{x, \hat{y}} p(\hat{y} | x)p(x) \log \left[\frac{p(\hat{y} | x)}{p(\hat{y})} \right]$$

$$I(\hat{Y}; Y) = \sum_{y, \hat{y}} p(y | \hat{y})p(\hat{y}) \log \left[\frac{p(y | \hat{y})}{p(y)} \right]$$

Finally, we can display the representations $\hat{Y}$ in the information plane:

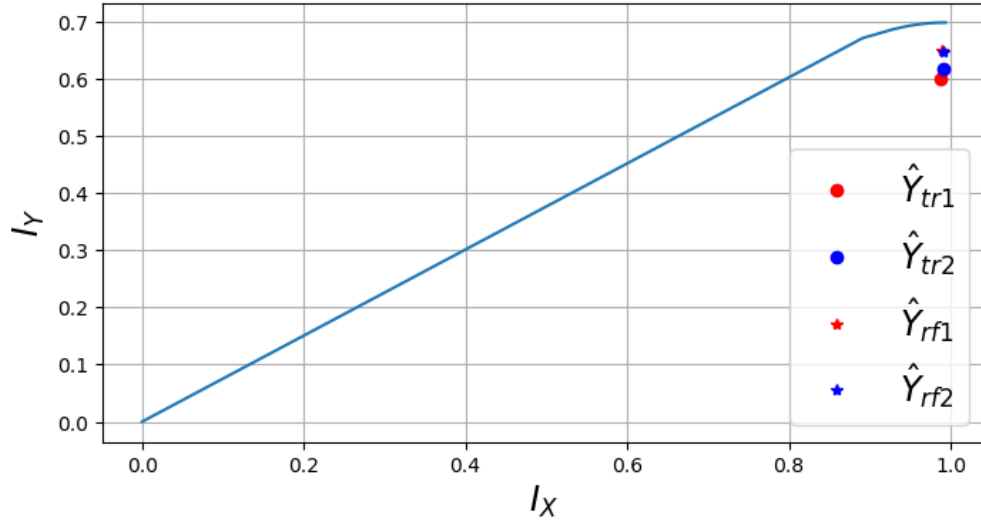


Figure 7.6: *Nondeterministic IB curve and the predictions of four models compared in the information plane.*

In figure 7.6 we see that, according to the IB method, the random forests perform better than the decision trees and tr_2 performs better than tr_1 . These results were expected since random forests generally perform better than trees and tr_2 was trained with 8 times more data than tr_1 . The more surprising result is that rf_1 is rated slightly higher than rf_2 even though rf_2 had more training data. To give the models an accuracy or correct classification rate we use only the deterministic part of the feature space, since we cannot determine if a sample that does not have a unique label has been correctly classified.

The accuracy scores are the following:

Model	Accuracy
tr_1	0.9887432795698925
tr_2	0.9982078853046595
rf_1	0.9994959677419355
rf_2	0.9999439964157706

According to this measure, rf_2 does perform better than rf_1 as is expected from a model with more training data. We can explain the discrepancy between IB optimality and the accuracy, by the fact that the former takes the nondeterministic part of the feature space into account, while the latter does not.

7.3.3 Discussion

The experiments of this chapter are inconclusive, however they do present an outline for possible further work on nondeterministic classification tasks in the IB framework. The Python tool developed for this chapter is publicly available and may also be used for further work³.

³https://github.com/MartinDeVroey/IB_solver

Chapter 8

Conclusions and further work

In this thesis, we explored the application of Tishby’s Information Bottleneck (IB) theory to classification trees and random forests. We started by introducing the IB framework. Then we dedicated a chapter to decision trees and one to random forests. We then described how the IB framework can be applied to decision trees and random forests and in particular how these models can be represented in the information plane. We proposed an intermediate representation of the input data, that is particular to random forests and showed how this representation can be placed in the information plane.

At that point we started our experiments. We first replicated experiments on trees from Tytgat (2021)[3] to show that some of its results and its conclusions depend on an inconsistent choice of data for the estimation of mutual information.

We then analysed random forest models by looking at how the random forest representation and the final classification evolve in the information plane as more trees are added to a forest. We found that with enough trees the random forest representation captures all the relevant information, but some of that information is lost when making a prediction. In IB terms, we can say that the random forest representation is a highly relevant but poorly compressed representation of the input.

We found that adding trees keeps improving the accuracy of the random forest even when the relevance of the random forest representation is maximal and its compression gets worse. This goes against the IB theory.

Finally, we introduced nondeterministic classification tasks, for which solving the IB problem and drawing the IB curve is nontrivial. We described how the problem can be solved numerically and presented our own implementation of the algorithm. We then trained models for a nondeterministic classification tasks and

compared the predictions of those models in the information plane alongside the IB curve generated by our IB solver.

Further work. It may be worthwhile to look at past work done on the IB method for similar inconsistencies as the ones we found in Tytgat (2021)[3], especially in papers making conflicting claims on certain aspects of the IB theory.

Based on our work with random forests, it would be interesting to look for methods to better exploit the relevant information captured by the forest representation. Additionally, since the random forest representation is a poor compression of the input, it may be interesting to modify the random forest algorithm to get better compression of the input and see if this results in better generalisation, as the IB theory suggests.

Finally, to further our work on nondeterministic classification, a first possible step is to check the results given by our implementation by solving the IB problem for a classification task where we can get an analytical solution e.g., when the input and the output are jointly Gaussian.

Bibliography

- [1] Random forest template - tibco spotfirer wiki page, 2017.
- [2] File: Cart tree kyphosis.png, 2018.
- [3] Tytgat Alexandre. An investigation of the information bottleneck (IB) method in supervised learning and how decision trees can be represented in the IB framework. Master’s thesis, Ecole polytechnique de Louvain, Université catholique de Louvain, 2021. Prom. : Delvenne, Jean-Charles.
- [4] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*. Springer, 2009.
- [5] James Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer, New York, 2015.
- [6] Bernhard C. Geiger. On information plane analyses of neural network classifiers – a review. *arXiv:2003.09671*, 2020.
- [7] Ziv Goldfeld et al. Estimating information flow in deep neural networks. In *Icml 2019*, pages 2299–2308, 2019.
- [8] Laurent Hyafil and RL Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.
- [9] Morteza Noshad et al. Scalable mutual information estimation using dependence graphs. *arXiv:1801.09125*, 2018.
- [10] Foster Provost. *Data Science for Business: What You Need to Know About Data Mining and Data-Analytic Thinking*. O’Reilly, Sebastopol, Calif., 1st edition, 2013.
- [11] Borja Rodriguez Gálvez. *The Information Bottleneck: Connections to Other Problems, Learning and Exploration of the IB Curve*. PhD thesis, 06 2019.

- [12] Andrew M. Saxe et al. On the information bottleneck theory of deep learning. *ICLR 2018 Conference Blind Submission*, 2019(12):124020, 2018.
- [13] scikit learn. Decision trees - scikit-learn. <https://scikit-learn.org/stable/modules/tree.html#tree-mathematical-formulation>, 2021. Accessed: 2022-12-18.
- [14] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv:1703.00810*, 2017.
- [15] DJ Strouse and David J. Schwab. The deterministic information bottleneck. *Neural Computation*, 29:1611–1630, 2015.
- [16] Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method. In *The 37th annual Allerton Conference on Communication, Control, and Computing*, pages 368–377, September 1999.
- [17] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. *CoRR*, abs/1503.02406, 2015.
- [18] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, and Zhi-Hua Zhou. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.
- [19] Abdellatif Zaidi, Inaki Estella Aguerri, and Shlomo Shamai. On the information bottleneck problems: Models, connections, applications and information theoretic views. *CoRR*, abs/2002.00008, 2020.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl