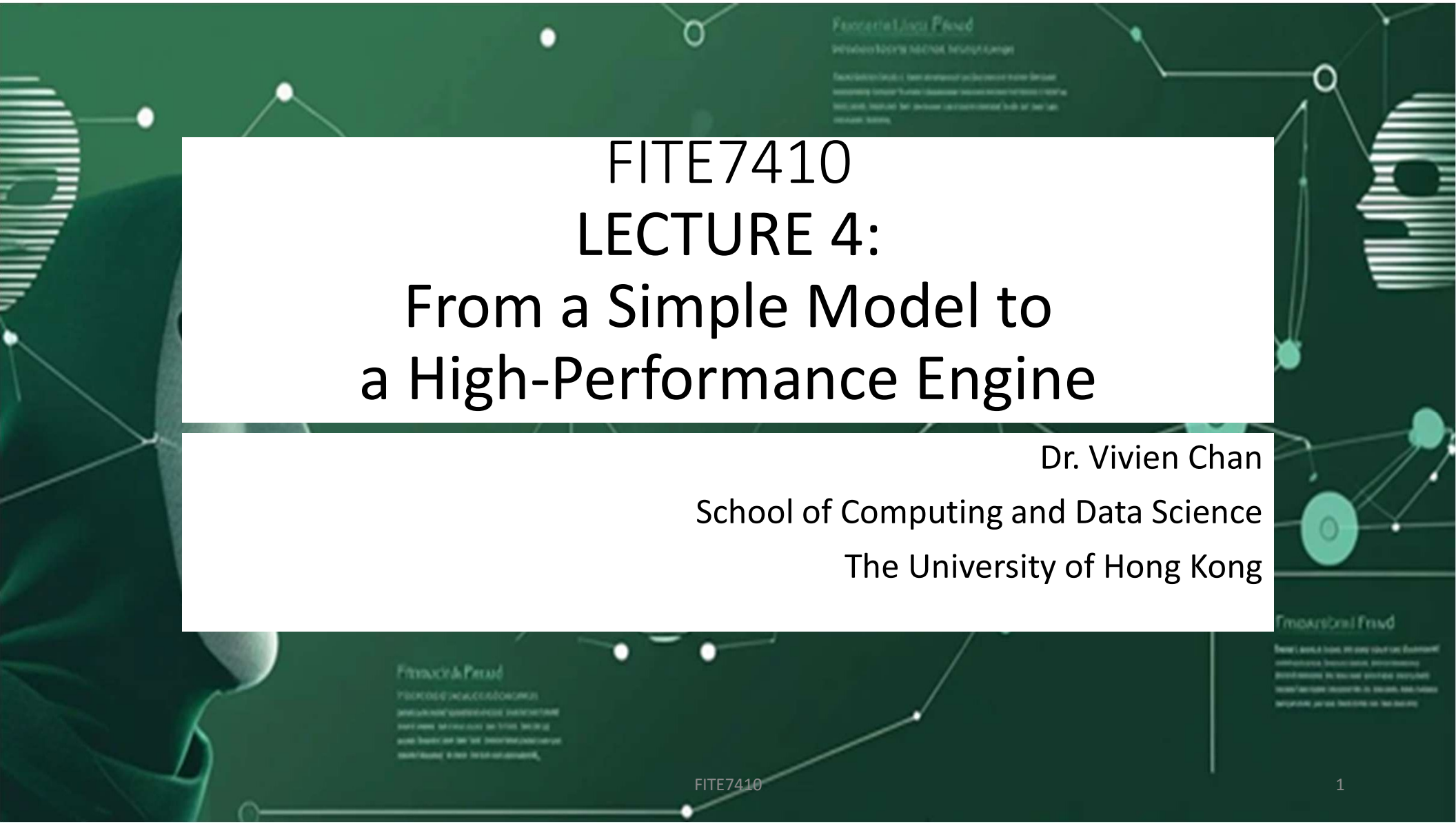


FITE7410
LECTURE 4:
From a Simple Model to a High-Performance Engine

Dr. Vivien Chan

School of Computing and Data Science

The University of Hong Kong

Agenda

Lecture goals:

- Deconstruct a Decision Tree:
  - Understand the building blocks of rule-based models, including concepts of *impurity*, *splitting*, and *pruning*.
- Build a High-Performance Engine:
  - Explain how Ensemble Learning, specifically Random Forest, combines many weak models to create a single, powerful and robust predictor.
- Handle the "Black Box" Problem:
  - Learn and apply techniques like Variable Importance to interpret the results of complex models, turning their predictions into actionable business intelligence.
- Critically Evaluate a Model's True Value:
  - Move beyond accuracy to assess a model's effectiveness in a real-world fraud detection scenario.

Decision Tree

Decision Trees - Basics

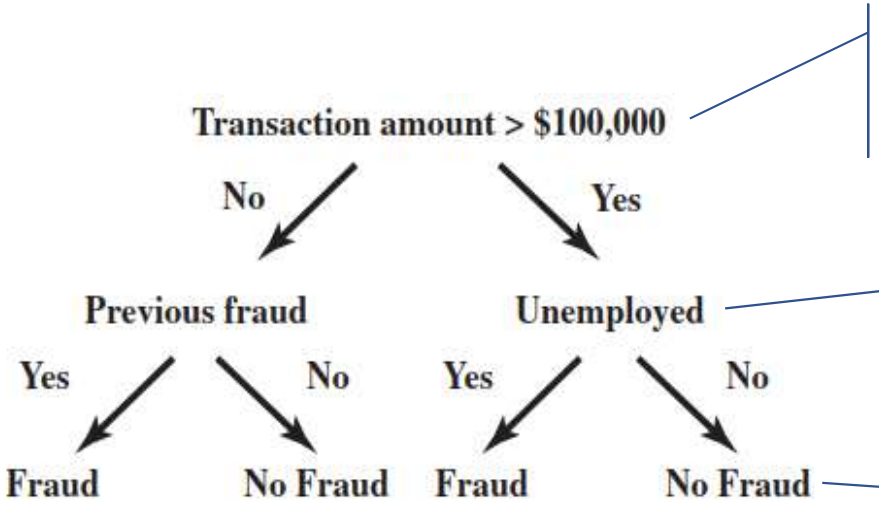


Figure 4.10 Example Decision Tree

Root

Starts with ALL attributes in the dataset

Decision Nodes

Leaf/Terminal Nodes

Classification or Regression result

Classification Decision Trees vs **Regression** Decision Trees

Decision Trees - Basics

- Splitting decision
  - Which attribute to split at what value?
  - e.g. Transaction amount is > $100, 000 or not, Previous fraud is yes or no, unemployed is yes or no
- Stopping decision
  - When to stop adding nodes to the tree?
- Assignment decision
  - What class (e.g., fraud or no fraud) to assign to a leave node?

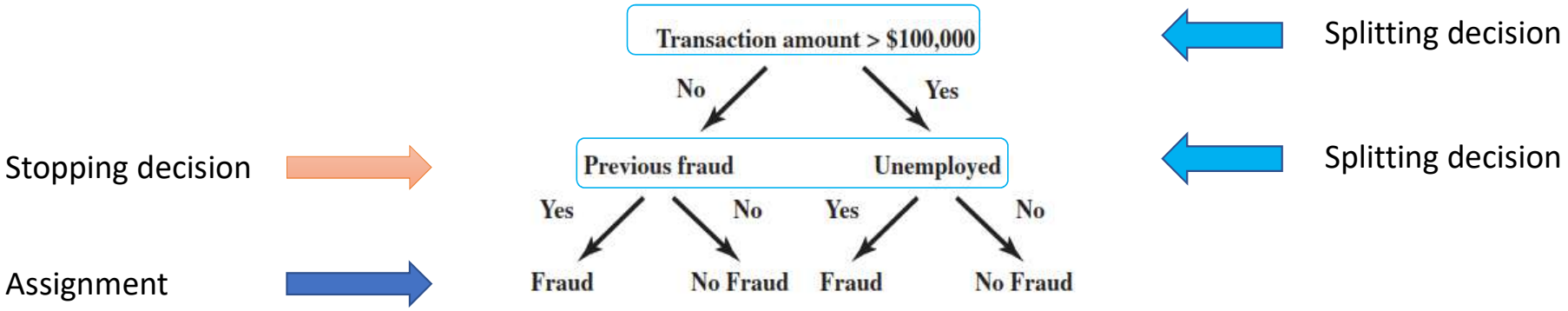


Figure 4.10 Example Decision Tree

Source: Fraud Analytics using Descriptive, Predictive, and Social Network Techniques, 1st ed, John Wiley & Sons Inc.

Decision Tree

- Decision Tree aims at minimizing “impurity” in the data
- What is impurity?
  - The **node impurity** is a measure of the homogeneity of the labels at the **node**.
  - Two **impurity** measures for classification (Gini **impurity** and entropy) and one **impurity** measure for regression (variance).

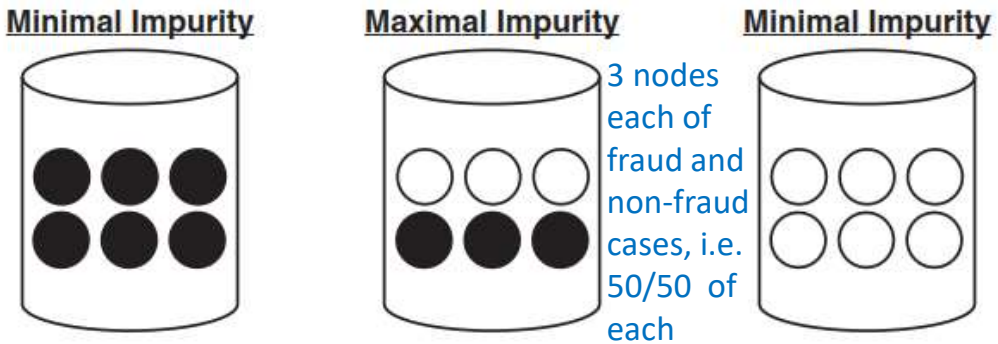


Figure 4.11 Example Data Sets for Calculating Impurity

Source: Fraud Analytics using Descriptive, Predictive, and Social Network Techniques, 1st ed, John Wiley & Sons Inc.

Decision Tree – Splitting decision

1/ Categorical variables

Entropy: $E(S) = -p_G \log_2(p_G) - p_B \log_2(p_B)$ (C4.5/See5)

Gini: $Gini(S) = 2p_G p_B$ (CART)

Chi-squared analysis (CHAID)

Where

p_G = proportions of "good" or non-fraud observations

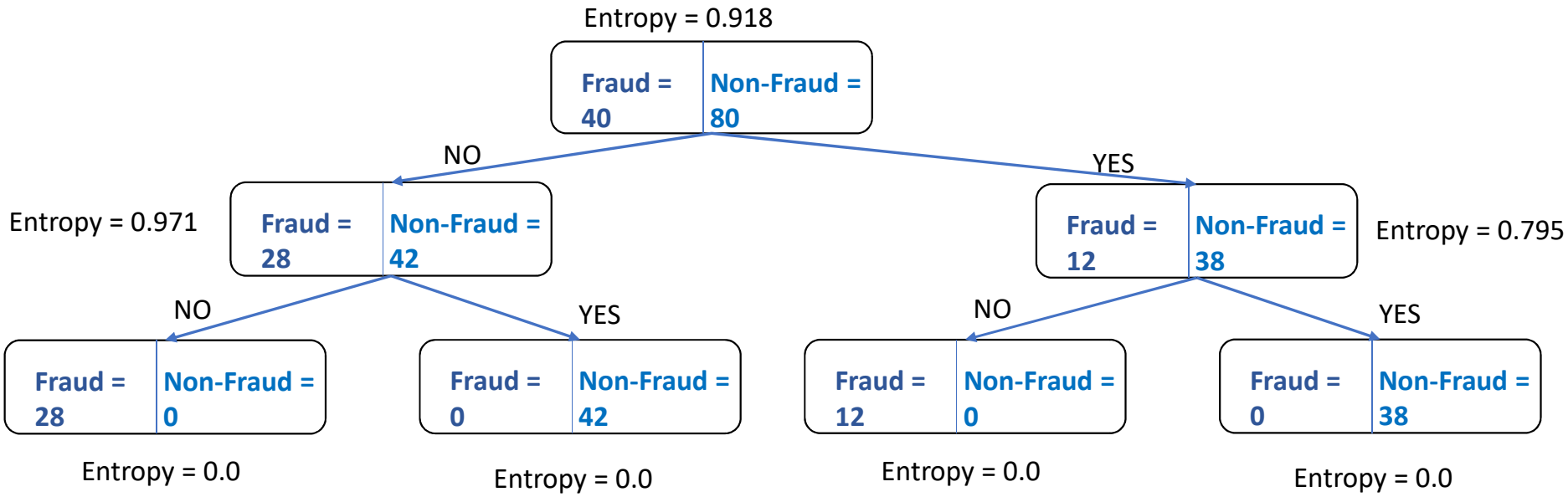
p_B = proportions of "bad" or fraud observations

2/ Continuous variables

- Use mean square error (MSE) or variance to predict the fraud percentage(FP)
- Also called Regression Tree

Source: Fraud Analytics using Descriptive, Predictive, and Social Network Techniques, 1st ed, John Wiley & Sons Inc.

Decision Tree - example



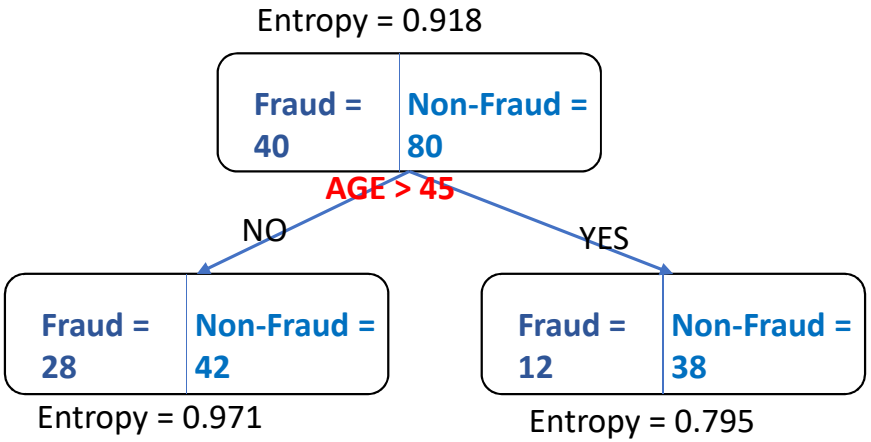
Information Gain

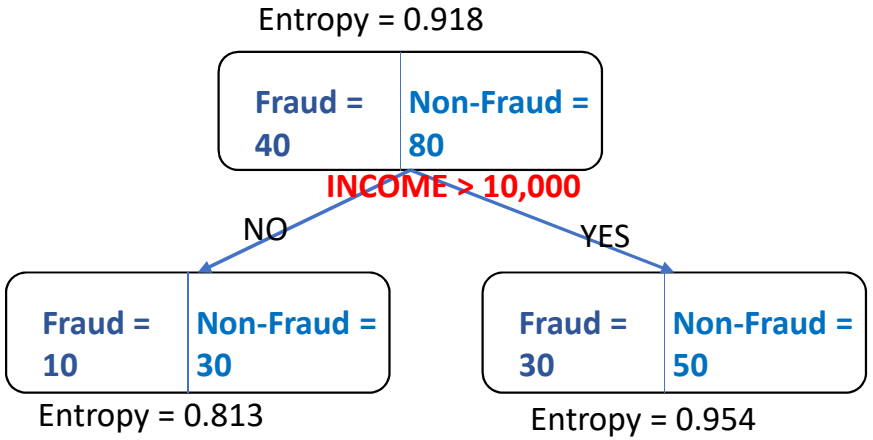
$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N} I(D_{left}) - \frac{N_{right}}{N} I(D_{right})$$

f: feature split on
D_p: dataset of the parent node
D_{left}: dataset of the left child node
D_{right}: dataset of the right child node
I: impurity criterion (Gini Index or Entropy)
N: total number of samples
N_{left}: number of samples at left child node
N_{right}: number of samples at right child node

Source: Fraud Analytics using Descriptive, Predictive, and Social Network Techniques, 1st ed, John Wiley & Sons Inc.

Decision Tree - example





$IG(D_p, AGE)$
= 0.918 – 70/120 x 0.971 – 50/120 x 0.795
= 0.02

$IG(D_p, INCOME)$
= 0.918 – 40/120 x 0.813 – 80/120 x 0.954
= 0.01

- Compare the IG for splitting decision using the attribute “AGE” and “INCOME”
- IG(AGE) > IG(INCOME)
  => use AGE as the splitting attribute

Source: Fraud Analytics using Descriptive, Predictive, and Social Network Techniques, 1st ed, John Wiley & Sons Inc.

Regression Tree - Example

- Regression tree splits - favour homogeneity within node and heterogeneity between nodes
- E.g. favour low MSE in a leave node

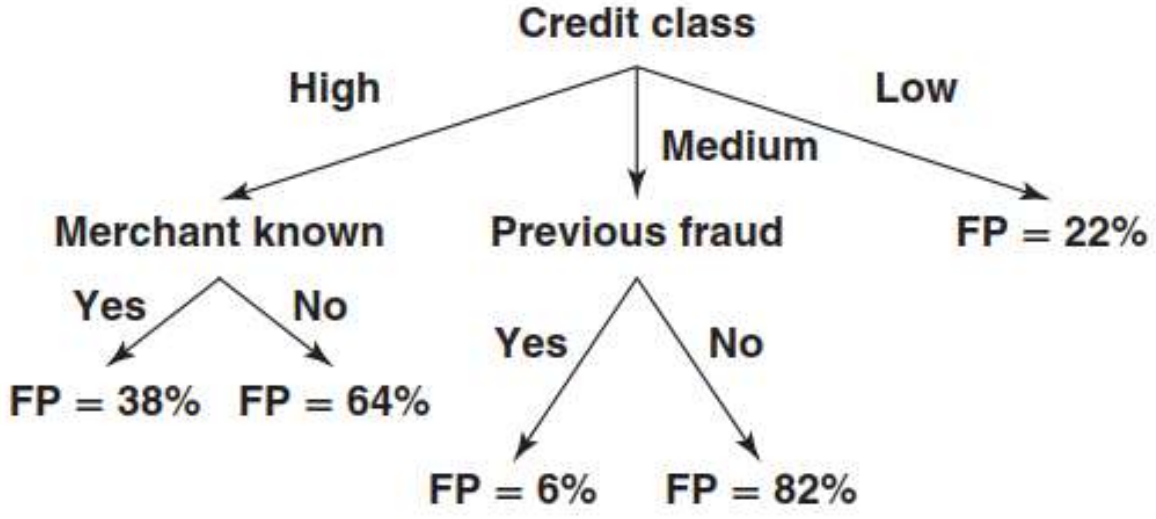


Figure 4.16 Example Regression Tree for Predicting the Fraud Percentage

Source: Fraud Analytics using Descriptive, Predictive, and Social Network Techniques, 1st ed, John Wiley & Sons Inc.

Classification vs Regression tree

Classification Tree

used when dependent variable is categorical.

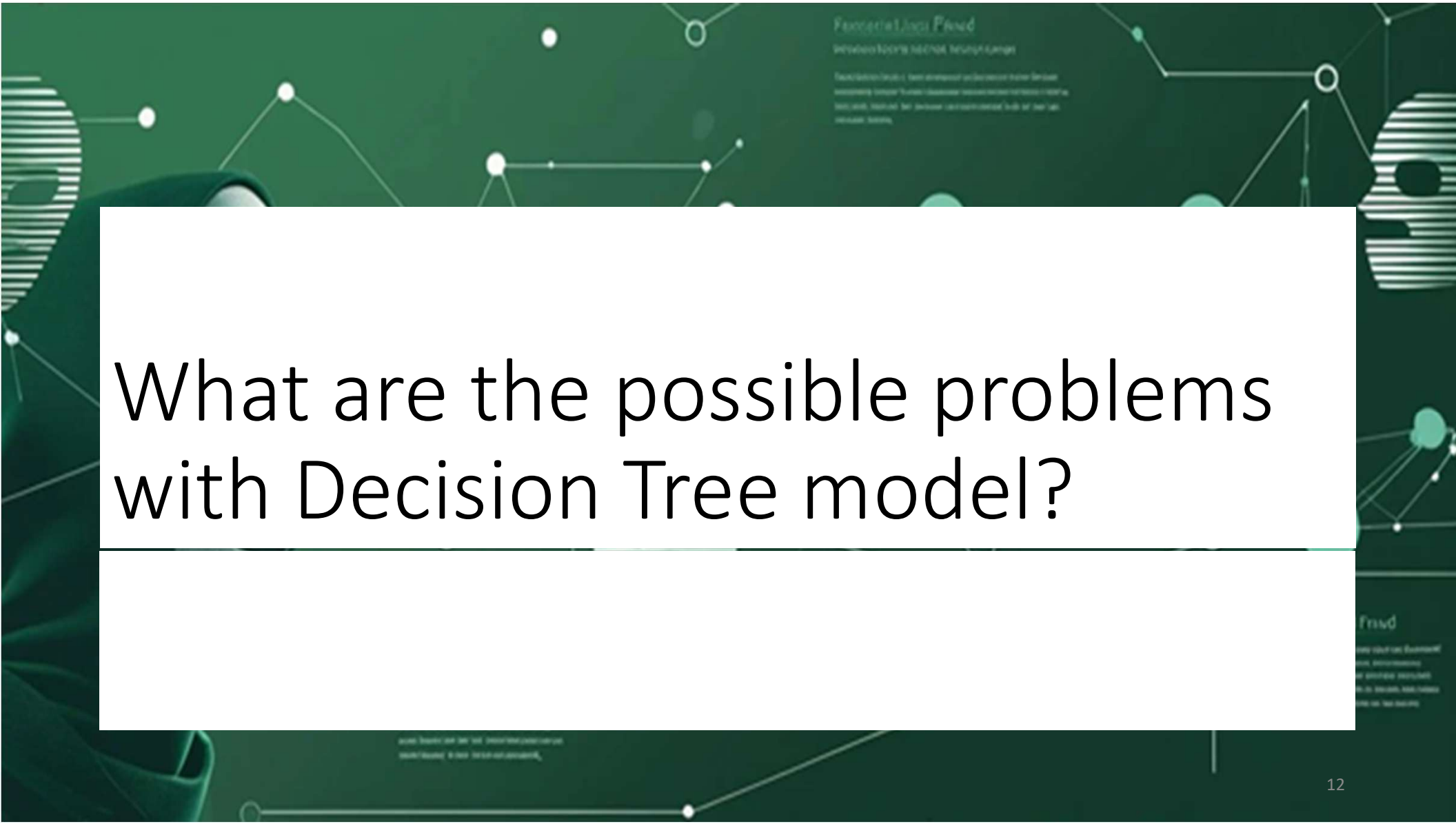
the value (class) obtained by terminal node in the training data is the mode of observations falling in that region

Regression Tree

used when dependent variable is continuous.

the value obtained by terminal nodes in the training data is the mean response of observation falling in that region

- Both the trees divide the predictor space (independent variables) into distinct and non-overlapping regions
- Both the trees follow a top-down greedy approach known as recursive binary splitting
- This splitting process is continued until a user defined stopping criteria is reached
- The splitting process results in fully grown trees until the stopping criteria is reached. But, the fully grown tree is likely to overfit data, leading to poor accuracy on unseen data.



What are the possible problems with Decision Tree model?

Challenges of Decision Tree

- Problem: Overfitting
  - Overfitting - refers to the condition when the model completely fits the training data but fails to generalize the testing unseen data
  - Problem of overfitting - it continues to grow the tree, i.e. the tree has become too complex and fails to correctly model the noise free pattern or trend in the data

- Solutions:
  - A) Pruning
    - Instead of creating a FULLY grown tree, remove the sections of the tree that are not useful to the classification result
    - **Pre-pruning** : Pre-defined criteria to stop growing the tree at early stage, e.g. depth of tree, minimal number of records after the split, etc.
    - **Post-pruning** : FULLY grown a tree and use a validation dataset to decide the size of the tree
  - B) Ensemble methods

To be discussed later

Decision Tree – Post-pruning

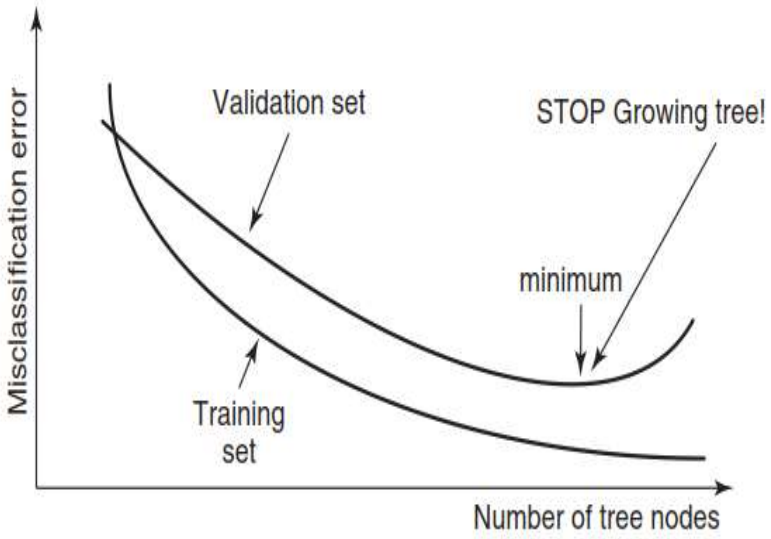


Figure 4.14 Using a Validation Set to Stop Growing a Decision Tree

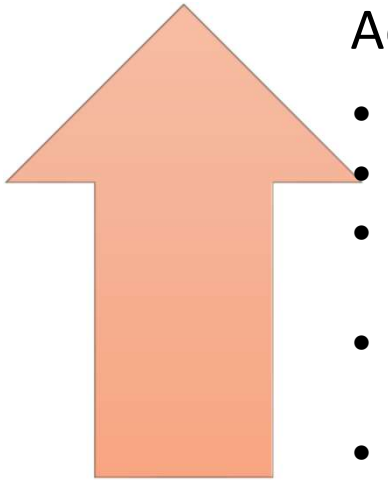
- Steps for stopping decision:
  - Split the data into Training set and Validation set
  - Usually, 70% of sample data will be used as Training set and 30% of the Training set as Validation set
  - Stop growing the tree when the misclassification error for Validation set reaches its minimal value

Source: Fraud Analytics using Descriptive, Predictive, and Social Network Techniques, 1st ed, John Wiley & Sons Inc.

Which dataset should we use to evaluate the performance of the Decision Tree?

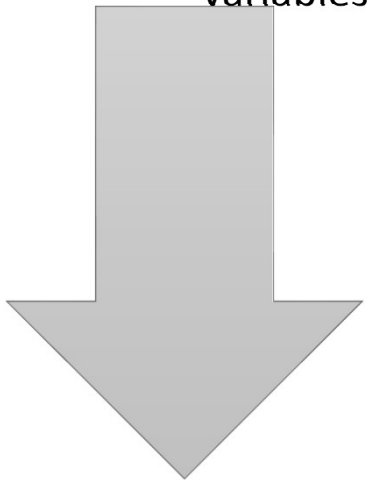
a. Training Dataset
b. Validation Dataset
c. Testing Dataset

Decision Tree



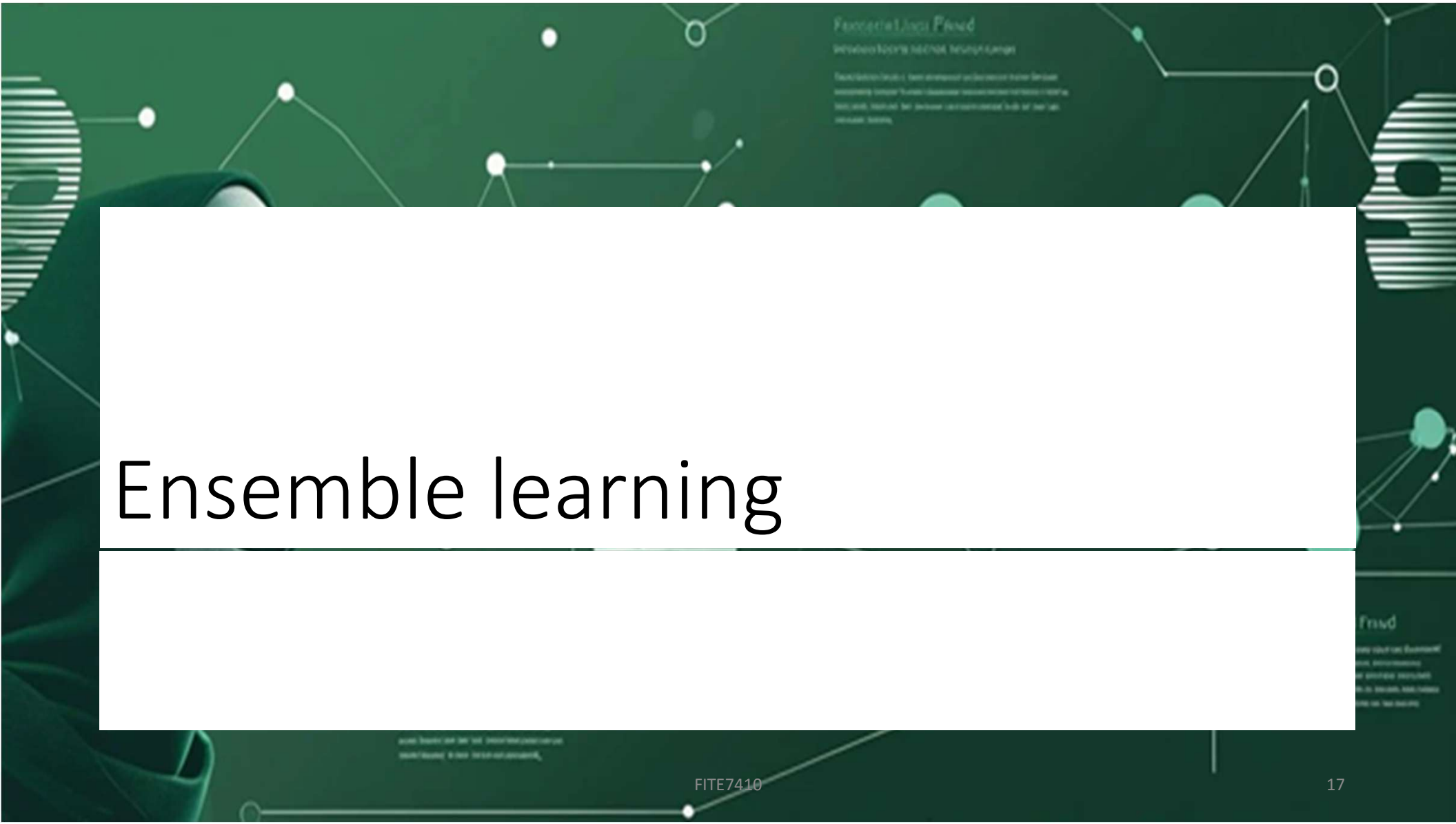
Advantages

- Easy to understand and interpret
- Useful in data exploration
- Less data clearing required – robust to outliers in inputs and no problem with missing values
- Data type not a constraint – can handle both continuous and categorical data for both input and target data
- Automatically detects interactions, accommodates nonlinearity and selects input variables



Disadvantages

- Prone to overfitting
- Splitting turns continuous input variables into discrete variables
- Unstable fitted tree – small change in the data result in a very different series of splits



Ensemble learning

Ensemble Learning

- Ensemble learning is a machine learning model that involve a group of prediction models (or “base models” or “weak learners”).

- Reasons of using ensemble learning:
  - 1/ Performance: An ensemble can make better predictions and achieve better performance than any single contributing model.
  - 2/ Robustness: An ensemble reduces the spread or dispersion of the predictions and model performance.

Ensemble Learning

- Homogenous ensemble learning
  - Combining SAME base models (“weak learners”) for training
- Heterogenous ensemble learning
  - Combining DIFFERENT base models for training
  - NOTE: the choice of these different base models should be coherent with how you aggregation these base models

Ensemble - Combining base models

Bagging

- Used for homogeneous base models
- Learns the base models independently from each other and in parallel
- Combines the base models by some kind of deterministic averaging process
- Objective: reduce variance

Example: Random Forest

Boosting

- Used for homogeneous base models
- Learns the base models sequentially in a very adaptive way, i.e. base models depends on previous ones
- Combines the base models following a deterministic strategy
- Objective: reduce bias

Example: AdaBoost, GradientBoost

Stacking

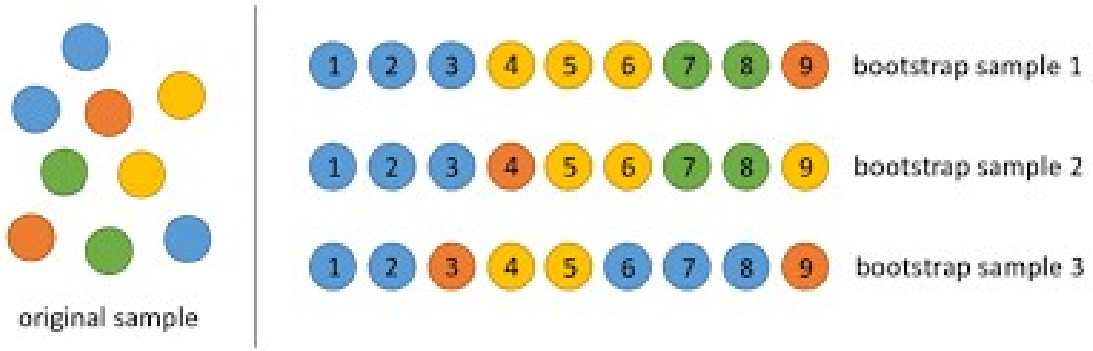
- Used for heterogenous base models
- Learns the base models in parallel
- Combines the base models by training a meta-model to output a prediction based on the different base models predictions
- Objective : improve performance results

Bagging - Bootstrap sampling

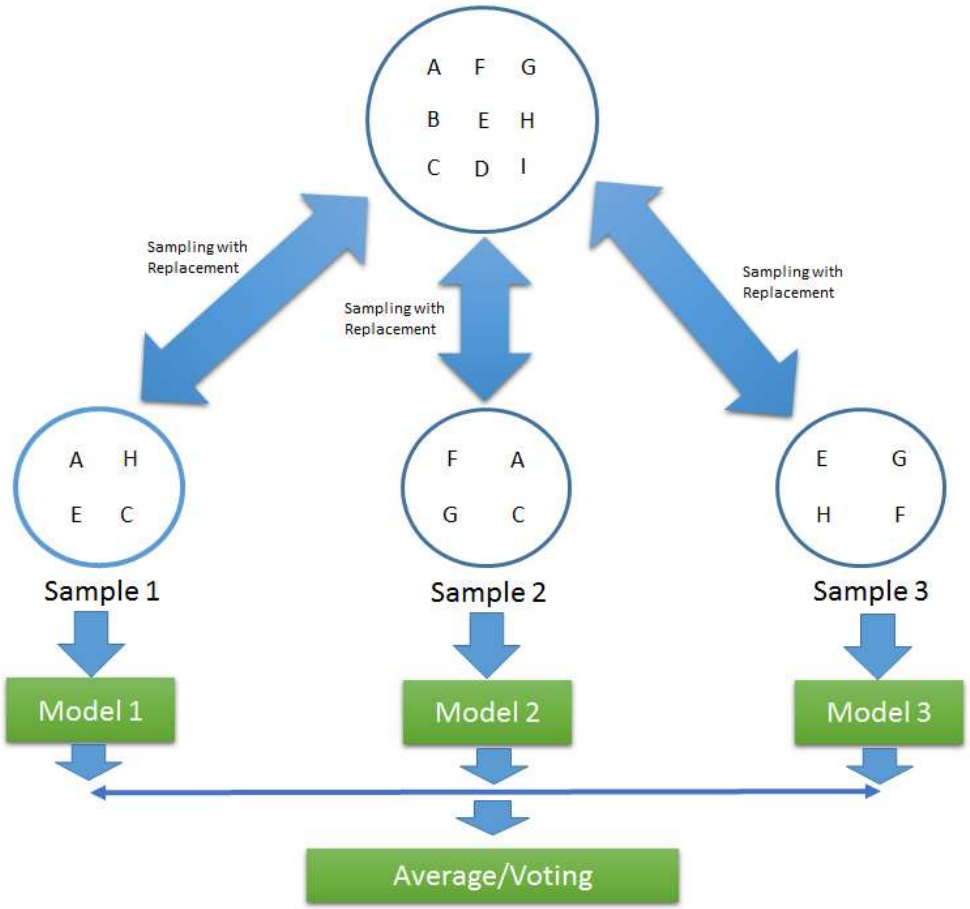
- Bootstrap sample is a statistic technique:
  - smaller sample of same size are repeated drawn, with replacement, from the larger original sample.
- Rationale behind:
  - The law of large numbers :
    - If you repeatedly sample again and again, it should approximate the large population.
  - Why replacement?
    - If the sample is a good representation of the large population, there will be more than one observation similar to another observation in the population

Bagging - Procedures of Bootstrapping

1. Choose a number of bootstrap samples to perform
2. Choose a sample size
3. For each bootstrap sample
    1. Draw a sample with replacement with the chosen size
    2. Calculate the statistic on the sample
4. Calculate the mean of the calculated sample statistics.

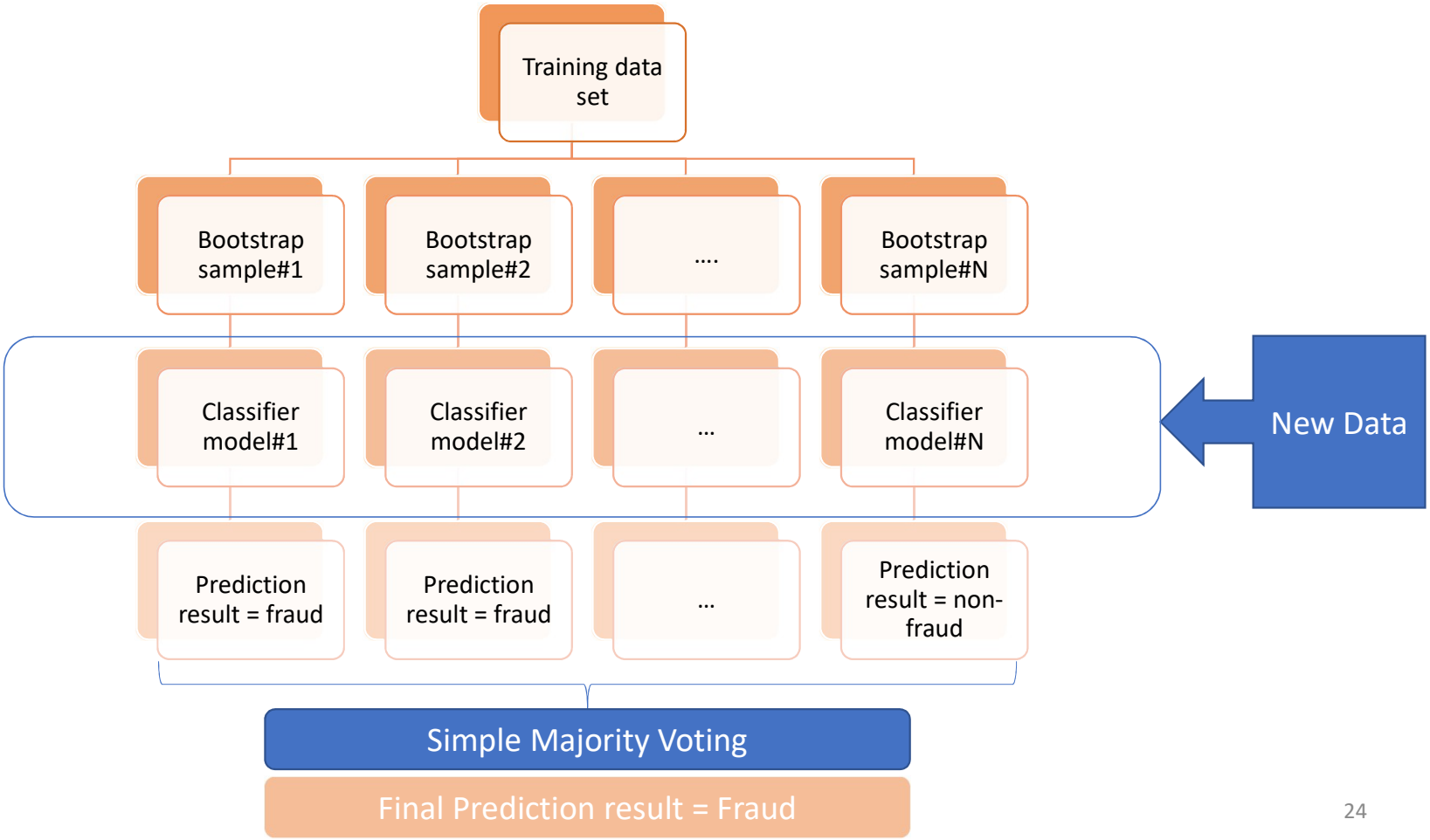


Bagging



- Bagging (Bootstrap aggregating) starts by taking N number of bootstraps from the underlying sample.
- The idea is to build a classifier for every bootstrap.
  - For classification : classified by letting all N classifiers vote and use a majority voting scheme
  - For regression : prediction by calculating the average of the outcome of the N prediction models

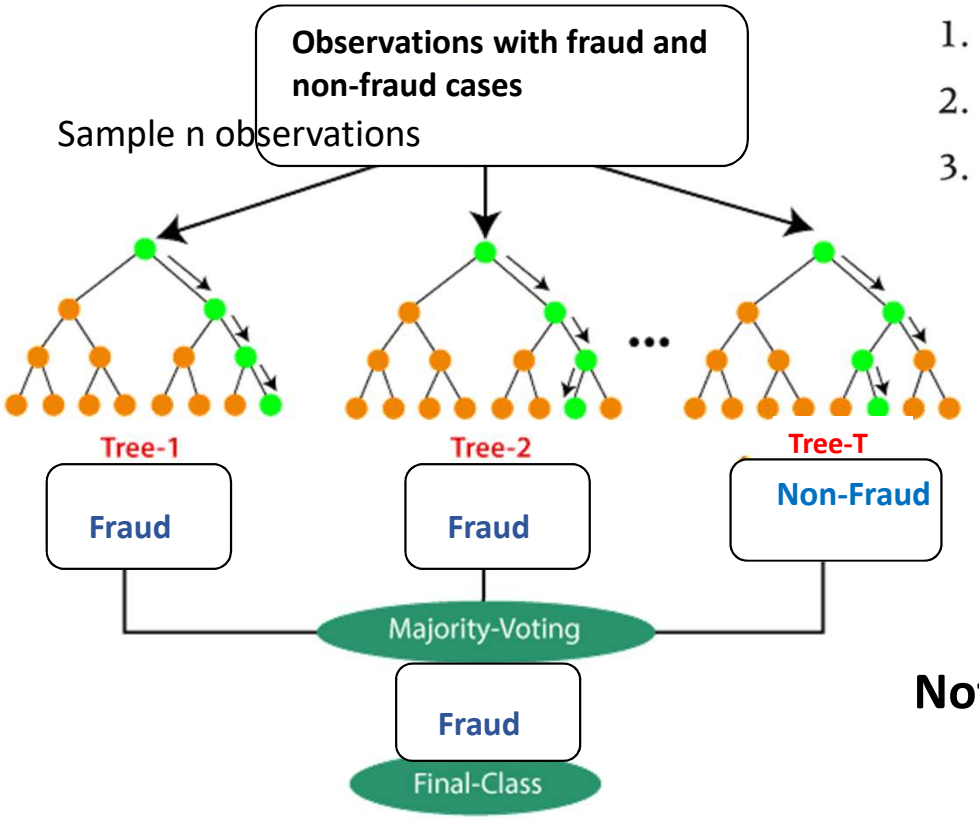
Bagging - example



Random Forest

Random Forest

- Random Forest is a forest of Decision Trees



1. Given a data set with n observations and N inputs.
2. m = constant chosen on beforehand.
3. For $t = 1, \ldots, T$
   a. Take a bootstrap sample with n observations.
   b. Build a decision tree whereby for each node of the tree, randomly choose m variables on which to base the splitting decision.
   c. Split on the best of this subset.
   d. Fully grow each tree without pruning.

Note: common choices of m is 1, 2 or $\log_2(N)+1$

Source: Fraud Analytics using Descriptive, Predictive, and Social Network Techniques, 1st ed, John Wiley & Sons Inc.

Random Forest – Step1

- Step 1: Create a bootstrap sample from the original dataset
- Example: number of observations = 5, number of features/variables = 4 (i.e. n=5, N=4)

Original Dataset

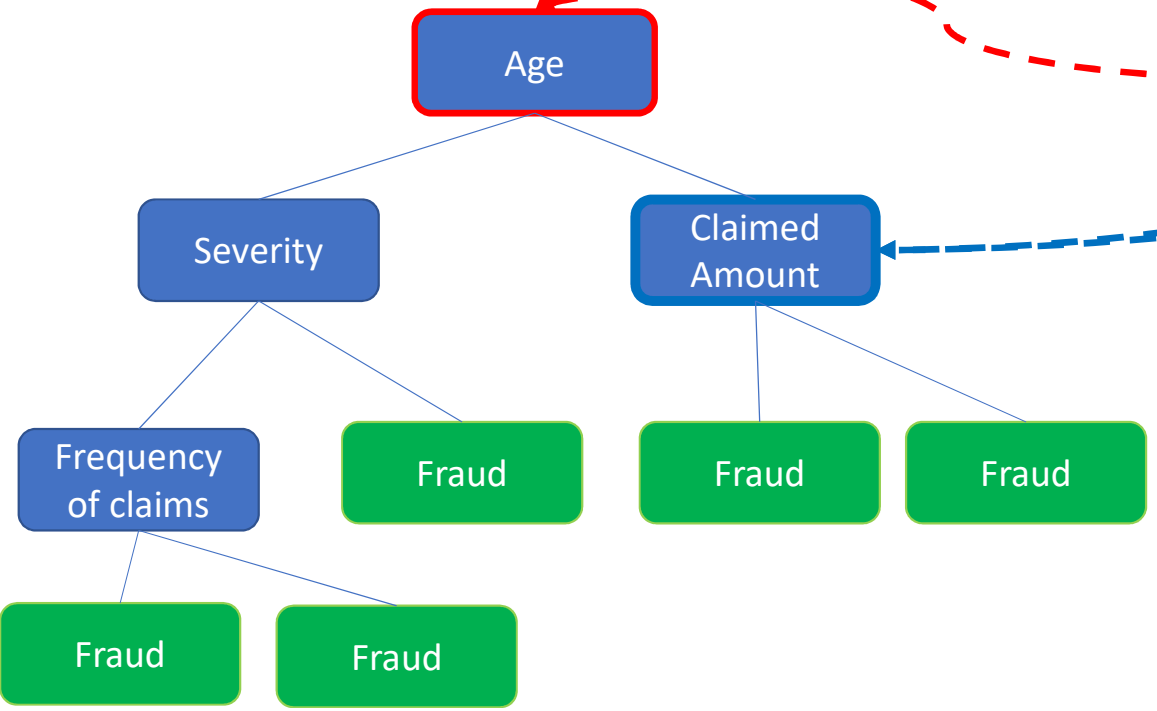
ID	Age	Severity	Claimed Amount	Frequency of claims	Fraud
1	35	High	$10,000	10	YES
2	45	Medium	$1,500	1	NO
3	28	Low	$1,000	2	NO
4	39	High	$10,000	8	YES
5	50	Low	$1,000	1	NO

Bootstrap sample

ID	Age	Severity	Claimed Amount	Frequency of claims	Fraud
3	28	Low	$1,000	2	NO
1	35	High	$10,000	10	YES
4	39	High	$10,000	8	YES
4	39	High	$10,000	8	YES
2	45	Medium	$1,500	1	NO

Random Forest – Step2

- Step 2: Create a decision tree with the bootstrap sample, BUT with a random subset of features/variables at each node
- Example: At each node, choose only 2 features at each node (m=2)

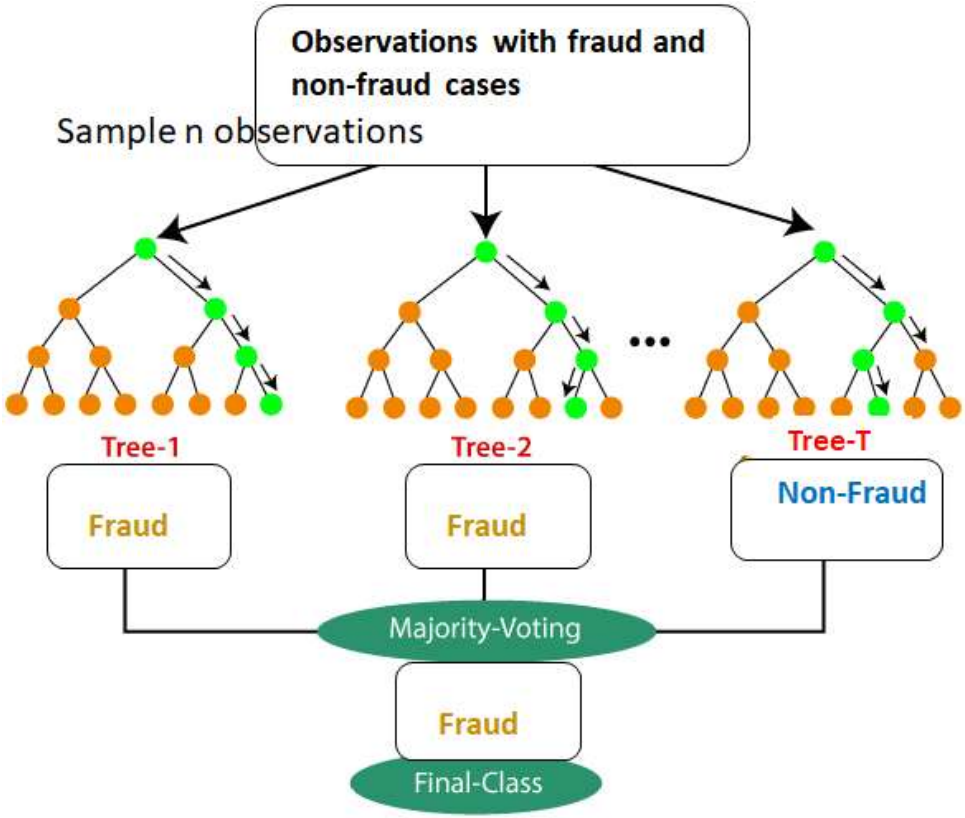


Bootstrap sample

ID	Age	Severity	Claimed Amount	Frequency of claims	Fraud
3	28	Low	$1,000	2	NO
1	35	High	$10,000	10	YES
4	39	High	$10,000	8	YES
4	39	High	$10,000	8	YES
2	45	Medium	$1,500	1	NO

Random Forest – Step3

- Step 3: Repeats step 1-2 for T times



- Random forest can be used for both classification tree or regression tree
- Achieve dissimilarities among the decision trees by:
  - Adopting a bootstrap procedure to select training samples for each tree
  - Selecting a random subset of attributes at each node
  - Training different base models of decision trees
- Result of random forest is a model with better performance compared to a single decision tree model

Out-of-Bag (OOB) Sample

Original Dataset

ID	Age	Severity	Claimed Amount	Frequency of claims	Fraud
1	35	High	$10,000	10	YES
2	45	Medium	$1,500	1	NO
3	28	Low	$1,000	2	NO
4	39	High	$10,000	8	YES
5	50	Low	$1,000	1	NO

Bootstrap sample

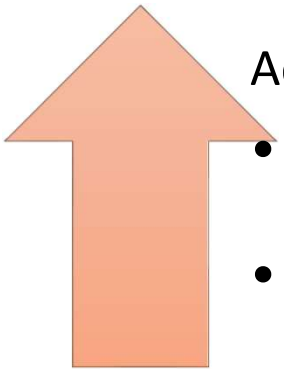
ID	Age	Severity	Claimed Amount	Frequency of claims	Fraud
3	28	Low	$1,000	2	NO
1	35	High	$10,000	10	YES
4	39	High	$10,000	8	YES
4	39	High	$10,000	8	YES
2	45	Medium	$1,500	1	NO

- **Out-of-Bag (OOB)** samples are those observations not being selected for use in the bootstrap sample
- Each of the OOB sample rows is passed through every decision tree that did not contain the OOB sample row in its bootstrap training data and a majority prediction is noted for each row
- **OOB score** = the number of correctly predicted rows from OOB sample

Can we use OOB score to evaluate the performance of the Random Forest model?

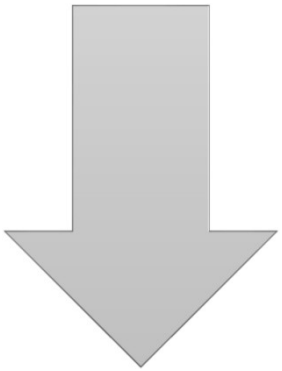
When is OOB score be useful?

Random Forest



Advantages

- Random Forest can achieve excellent predictive performance and suitable for the requirements of fraud detection
- Capable of dealing with data sets having only a few observations, but with lots of variables

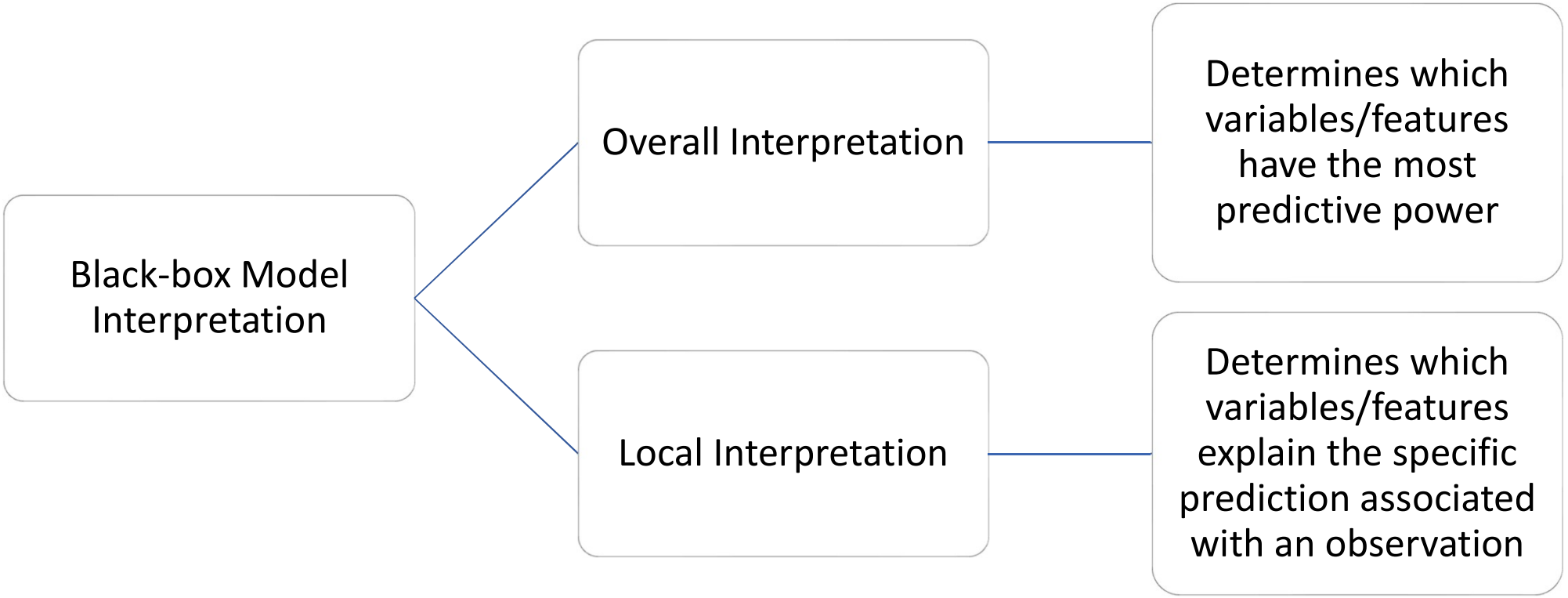


Disadvantages

- It is a black-box model.
  - **Solution**: Variable importance can be used to understand the internal workings of Random Forest (or any ensemble model)

How to interpret Random Forest

How to interpret a “black-box” model?



Overall Interpretation
- Variable/Feature Importance

Importance	Variable
10%	Var_5
8%	Var_1
6%	Var_2
3%	Var_8
3%	Var_6
2%	Var_22
1%	Var_13

Pic source: https://miro.medium.com/max/324/1*k82O3uoBloN7ymkJ12pMQQ.png

- Variables with HIGH importance can be used for further analysis, while variables with LOW importance can be discarded
- Variables with TOO much importance compared to others might be bugs in the data or model
- Can be an indication of whether a “new” variable is relevant or useful to your model, e.g. compare the variable importance for the model with and without the new variables
- Compare variable importance for different models

Local Interpretation

			3 variables with most contribution					
			1st variable		2nd variable		3rd variable	
ID	Probability of default	Class predicted	Name	Impact	Name	Impact	Name	Impact
12098321	**95%**	**1**	Var_1	+34%	Var_3	+19%	Var_9	+8%
12098322	**88%**	**1**	Var_1	+25%	Var_8	+14%	Var_3	+13%
12098323	**35%**	**0**	Var_7	-27%	Var_5	-23%	Var_1	-12%

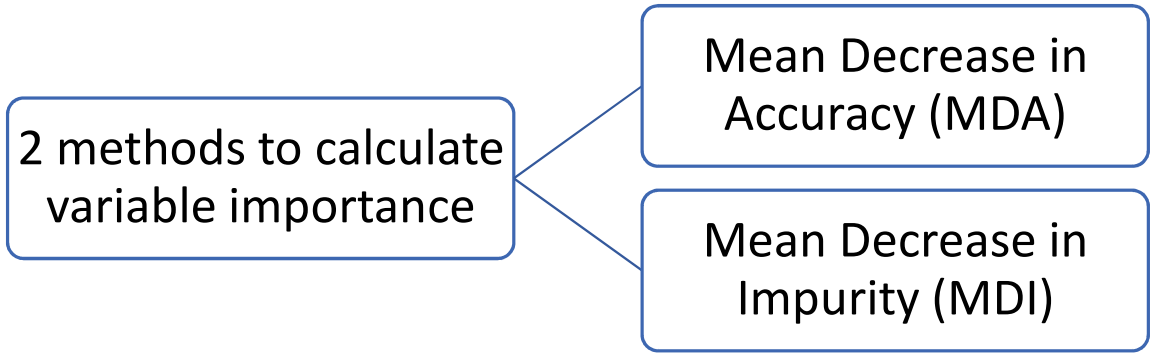
- To understand individual observations, i.e. reasons of the prediction, or which variables explains/predicts more for this specific observation

	Frequency of being in the top 3 variables			
	1st	2nd	3rd	***Total***
Var_1	43	15	12	***70***
Var_3	11	24	30	***65***
Var_8	10	8	12	***30***
Var_9	2	4	45	***51***

Pic source: https://miro.medium.com/max/1400/1*joBp6qhBfXCE8c_9q2F71Q.png

- To understand the most frequent reasons of the predictions. This might be particularly useful when you have an imbalanced dataset.

How to calculate “Variable Importance”?



- Related package in R
  - install.packages('caret’) *#package name*
  - library(caret) *#library name*
  - varImp(object, …) *# function to list variable importance*
  - varImpPlot *# function to plot variable importance for Random Forest model*

Variable Importance not only applicable to Random Forrest, also applicable to other ensemble model

Variable Importance - MDI

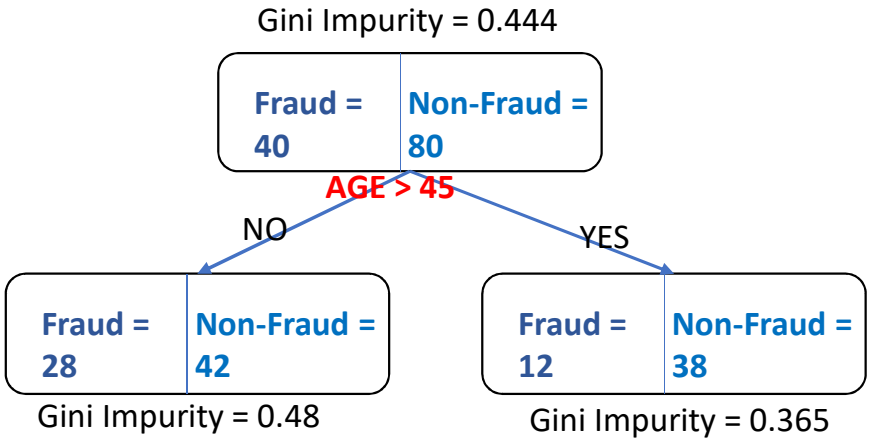
- Mean Decrease Impurity (MDI)
  - Measures the decrease in Gini impurity
  - What is Gini impurity for binary classification?
    - **Gini Impurity** is a measurement of the likelihood of an incorrect classification of a new instance of a random variable, if that new instance were randomly classified according to the distribution of class labels from the data set.

$$I_G = 1 - \sum_{j=1}^{c} p_j^2$$

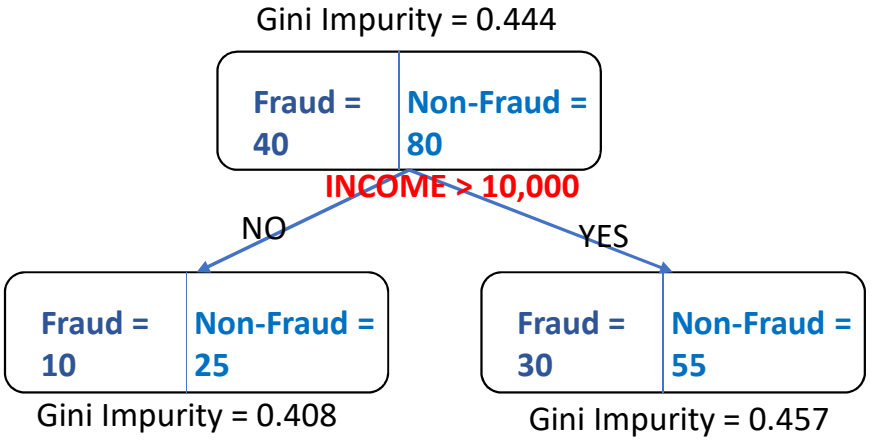
where c = number of class labels, p = proportions of samples belonging to class c in the node

Thus, for binary classification, $I_G = 1 - (p_1^2 + p_2^2)$

Gini Impurity - example



Gini impurity of "AGE" split
= weighted average of leaf nodes
= 70/120 x 0.48 + 50/120 x 0.365
= 0.432



Gini impurity of "INCOME" split
= weighted average of leaf nodes
= 35/120 x 0.408 + 85/120 x 0.457
= 0.443

- Compare the Gini Impurity for splitting decision using the attribute "AGE" and "INCOME"
- Gini Impurity(**AGE**) < Gini Impurity(**INCOME**)

  => use **AGE** as the splitting attribute

Variable Importance - MDI

- Mean Decrease Impurity (MDI)
  - Measures how effective the feature is at reducing impurity, or the total decrease in node impurity averaged over all trees of the ensemble
  - Counts the times a feature is used to split a node, weighted by the number of samples it splits
  - At each split in each tree, the improvement in the split-criterion is the importance measure attributed to the splitting variable, and is accumulated over all the trees in the forest separately for each variable

- Problems with MDI
  - Computationally efficient BUT biased towards continuous and high cardinality variables

Variable Importance - MDA

- Mean Decrease in Accuracy (MDA)
  - Measures the decrease in accuracy (Permutation Importance)
  - Steps of calculating MDA:
    - Use the out-of-bag (OOB) samples to construct variable importance, to measure the prediction strength of each variable
    - When the b^{th} tree is grown, the OOB samples are passed down the tree, and the prediction accuracy is recorded
    - Values for the j^{th} variable are randomly permuted in the OOB samples, and the accuracy is again computed
    - The decrease in accuracy as a result of this permutation is averaged over all trees, and is used as measure of the importance of variable j in the random forest.

Variable Importance

1. Permute the values of the variable under consideration (e.g., X_j) on the validation or test set.
2. For each tree, calculate the difference between the error on the original, unpermuted data and the error on the data with X_j permuted as follows:

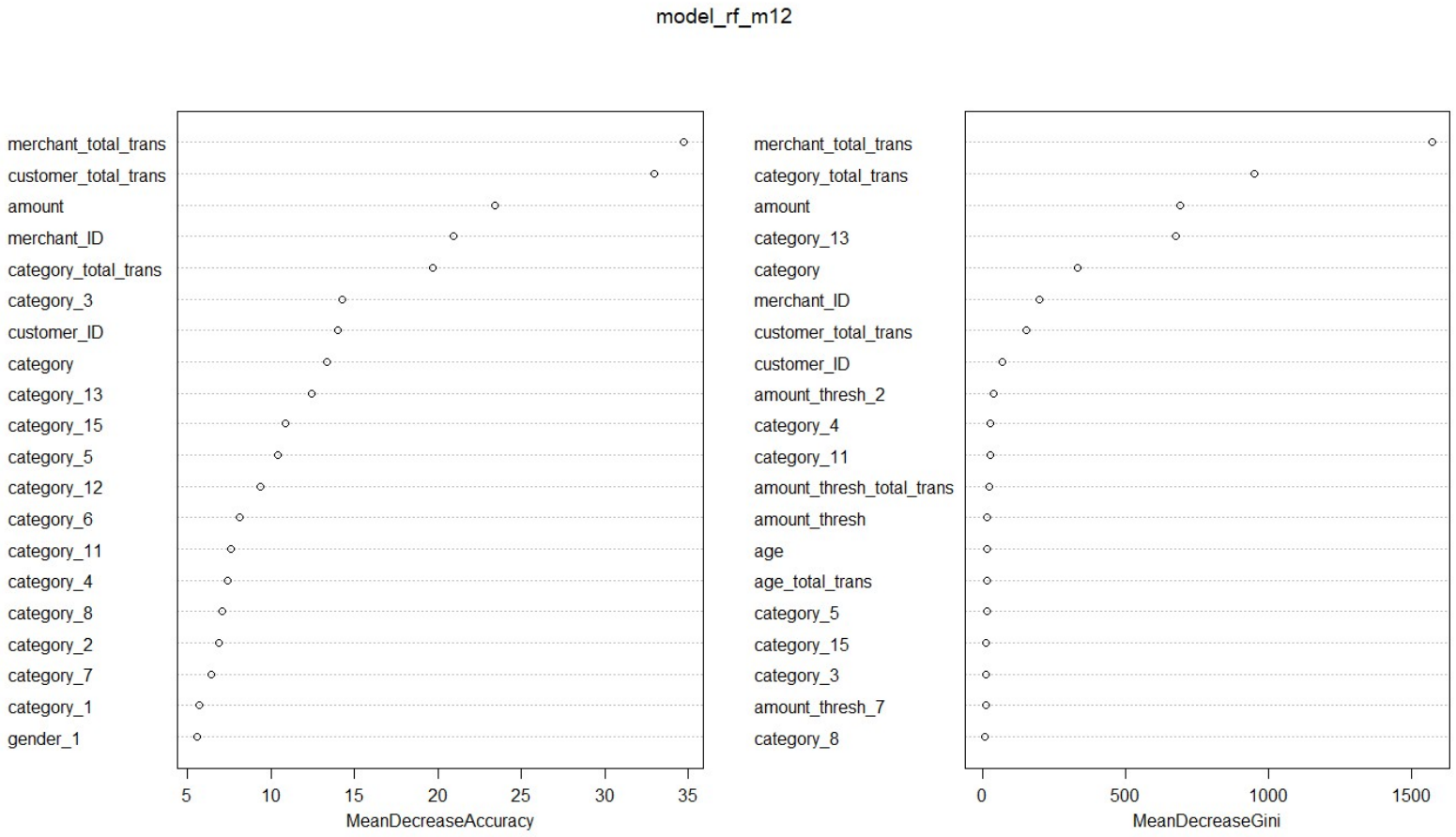
$$\mathrm{VI}(X_j) = -\frac{1}{ntree}\sum_t (error_t(D) - error_t(\tilde{D}_j))$$

whereby *ntree* represents the number of trees in the ensemble, D the original data, and $\tilde{D}_j$ the data with variable X_j permuted. In a regression setting, the error can be the mean squared error (MSE), whereas in a classification setting, the error can be the misclassification rate.

3. Order all variables according to their VI value. The variable with the highest VI value is the most important.

Source: Fraud Analytics using Descriptive, Predictive, and Social Network Techniques, 1st ed, John Wiley & Sons Inc.

Variable Importance – example in R



Using R to build Fraud Detection models

Before model building

Preparing data for model building

1. Perform data pre-processing (including EDA, Impute missing values, feature normalization, feature selection, feature creature, etc., if needed)
2. Splitting the dataset into training and testing datasets.
3. Imbalance Data Set handling
4. Prepare the class label
   - Use the function "**as.factor**" to convert the class label to factor

Example of EDA using R (re-cap of Lecture 1)

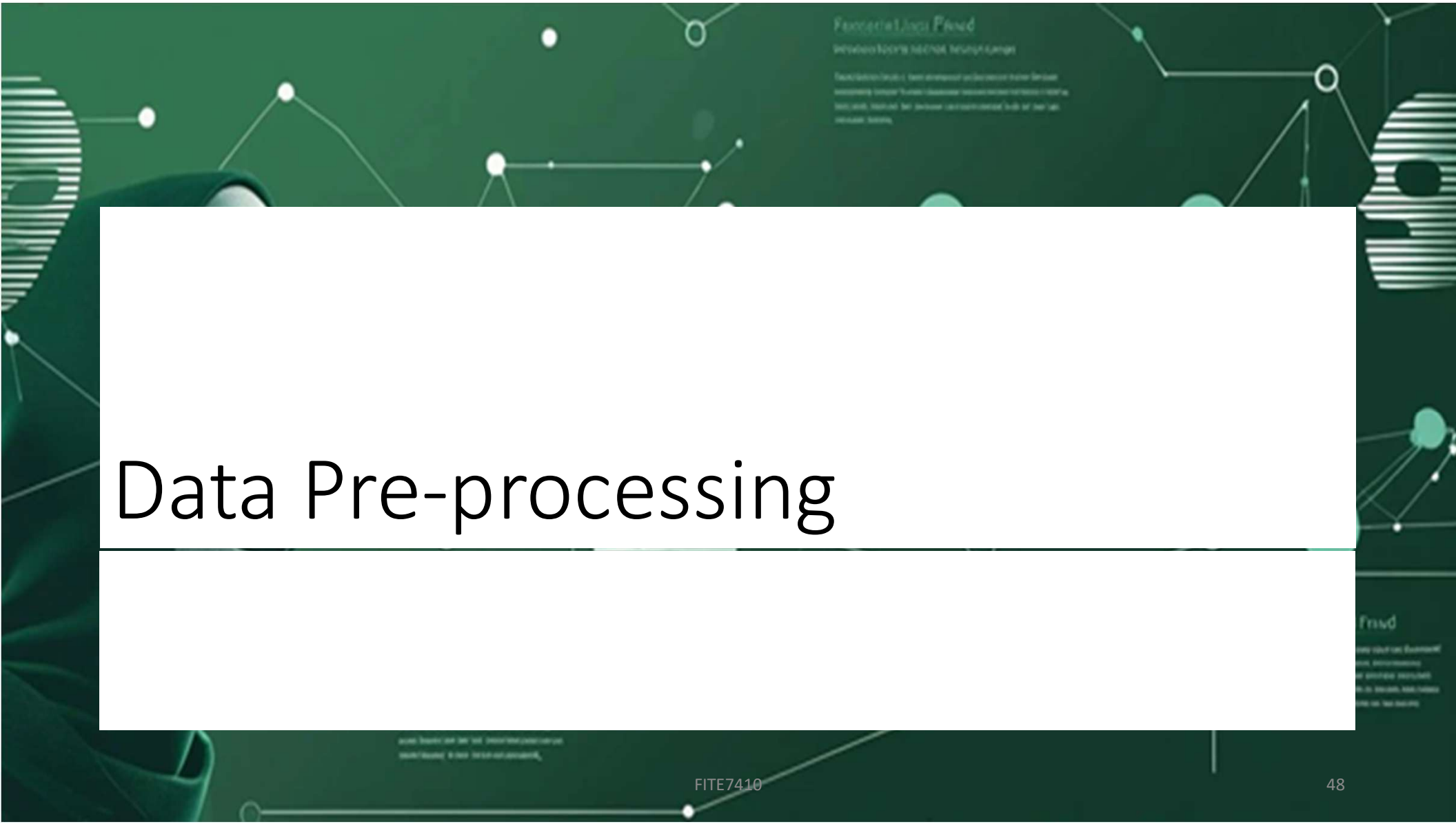
- Dataset : https://www.kaggle.com/mlg-ulb/creditcardfraud
- The Credit Card Fraud Detection Dataset comprises transactions that European credit card holders made in September 2013. The dataset shows transactions that occurred in two days.
- The dataset has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection.

Useful libraries in R

- library(tidyverse) # for data manipulation
- library(caret) # for models and confusionMatrix function

- library(ROSE) #for over-/under-sampling function
- library(precrec) # For both ROC and Precision-Recall curve and AUC

- library(rpart) #for building the decision tree model (Recursive Partitioning)
- library(rpart.plot) # for creating a visually appealing plot of the tree
- library(randomForest) # For the Random Forest functions



Data Pre-processing

Additional feature engineering functions in R

- scale
- Subset

The “scale” function

Use “help” function to view the R documentation of the function “scale”

```
help(scale)
```

scale {base} R Documentation

Scaling and Centering of Matrix-like Objects

Description

`scale` is generic function whose default method centers and/or scales the columns of a numeric matrix.

Usage

```
scale(x, center = TRUE, scale = TRUE)
```

Arguments

`x`	a numeric matrix(like object).
`center`	either a logical value or numeric-alike vector of length equal to the number of columns of `x`, where ‘numeric-alike’ means that `as.numeric(.)` will be applied successfully if `is.numeric(.)` is not true.
`scale`	either a logical value or a numeric-alike vector of length equal to the number of columns of `x`.

```r
# Load csv dataset
data <- read.csv('../input/creditcardfraud/creditcard.csv')

#Show structure of the dataset
print("Strucutre of dataset")
str(data)

#Show the Class label results
print("Original dataset")
table(data$Class)
```

```
[1] "Original dataset"

     0      1
284315    492
```

```
[1] "Strucutre of dataset"
'data.frame':   284807 obs. of  31 variables:
 $ Time  : num  0 0 1 1 2 2 4 7 7 9 ...
 $ V1    : num  -1.36 1.192 -1.358 -0.966 -1.158 ...
 $ V2    : num  -0.0728 0.2662 -1.3402 -0.1852 0.8777 ...
 $ V3    : num  2.536 0.166 1.773 1.793 1.549 ...
 $ V4    : num  1.378 0.448 0.38 -0.863 0.403 ...
 $ V5    : num  -0.3383 0.06 -0.5032 -0.0103 -0.4072 ...
 $ V6    : num  0.4624 -0.0824 1.8005 1.2472 0.0959 ...
 $ V7    : num  0.2396 -0.0788 0.7915 0.2376 0.5929 ...
 $ V8    : num  0.0987 0.0851 0.2477 0.3774 -0.2705 ...
 $ V9    : num  0.364 -0.255 -1.515 -1.387 0.818 ...
 $ V10   : num  0.0908 -0.167 0.2076 -0.055 0.7531 ...
 $ V11   : num  -0.552 1.613 0.625 -0.226 -0.823 ...
 $ V12   : num  -0.6178 1.0652 0.0661 0.1782 0.5382 ...
 $ V13   : num  -0.991 0.489 0.717 0.508 1.346 ...
 $ V14   : num  -0.311 -0.144 -0.166 -0.288 -1.12 ...
 $ V15   : num  1.468 0.636 2.346 -0.631 0.175 ...
 $ V16   : num  -0.47 0.464 -2.89 -1.06 -0.451 ...
 $ V17   : num  0.208 -0.115 1.11 -0.684 -0.237 ...
 $ V18   : num  0.0258 -0.1834 -0.1214 1.9658 -0.0382 ...
 $ V19   : num  0.404 -0.146 -2.262 -1.233 0.803 ...
 $ V20   : num  0.2514 -0.0691 0.525 -0.208 0.4085 ...
 $ V21   : num  -0.01831 -0.22578 0.248 -0.1083 -0.00943 ...
 $ V22   : num  0.27784 -0.63867 0.77168 0.00527 0.79828 ...
 $ V23   : num  -0.11 0.101 0.909 -0.19 -0.137 ...
 $ V24   : num  0.0669 -0.3398 -0.6893 -1.1756 0.1413 ...
 $ V25   : num  0.129 0.167 -0.328 0.647 -0.206 ...
 $ V26   : num  -0.189 0.126 -0.139 -0.222 0.502 ...
 $ V27   : num  0.13356 -0.00898 -0.05535 0.06272 0.21942 ...
 $ V28   : num  -0.0211 0.0147 -0.0598 0.0615 0.2152 ...
 $ Amount: num  149.62 2.69 378.66 123.5 69.99 ...
 $ Class : int  0 0 0 0 0 0 0 0 0 0 ...
```

Examples of using R – feature normalization

Scale only 1 attribute

- Center = subtracting the column means
- Scale = dividing by standard deviations (if centering is done)

Example 1

```
# Scale only 1 attribute, with centering and NO scaling
data[,c(30)] <- as.numeric(scale(data[,c(30)], center = TRUE, scale = FALSE))
```

Function “as.numeric” is used here to convert the data type back to numeric type

Before normalization

```
     Amount
Min.   :    0.00
1st Qu.:    5.60
Median :   22.00
Mean   :   88.35
3rd Qu.:   77.17
Max.   :25691.16
```

Scale = FALSE

```
    Amount.V1
Min.   : -88.350
1st Qu.: -82.750
Median : -66.350
Mean   :   0.000
3rd Qu.: -11.185
Max.   :25602.810
```

Scale = TRUE

```
    Amount.V1
Min.   : -0.35323
1st Qu.: -0.33084
Median : -0.26527
Mean   :  0.00000
3rd Qu.: -0.04472
Max.   :102.36206
```

Example 2

```
# Scale only 1 attribute, with centering and scaling
data[,c(30)] <- as.numeric(scale(data[,c(30)], center = TRUE, scale = TRUE))
```

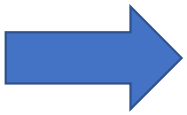
Examples of using R – feature normalization

- Scale multiple attributes

```
# Numerical data normalization for the whole dataset
for (i in 1:(ncol(data)-1))
{
    data[,i] <- as.numeric(scale(data[,i],center = TRUE, scale = TRUE))
}
summary(data)
```

"ncol" = Count the number of columns
1:n = generate a sequences of numbers from 1 to n

```
       V1                  V2
Min.   :-56.40751   Min.   :-72.71573
1st Qu.: -0.92037   1st Qu.: -0.59855
Median :  0.01811   Median :  0.06549
Mean   :  0.00000   Mean   :  0.00000
3rd Qu.:  1.31564   3rd Qu.:  0.80372
Max.   :  2.45493   Max.   : 22.05773
```



```
      V1.V1               V2.V1
Min.   :-28.798504   Min.   :-44.03521
1st Qu.: -0.469891   1st Qu.: -0.36247
Median :  0.009245   Median :  0.03966
Mean   :  0.000000   Mean   :  0.00000
3rd Qu.:  0.671693   3rd Qu.:  0.48672
Max.   :  1.253349   Max.   : 13.35773
```

Examples in R – variable selection
=> use the ‘subset’ function

```r
# Remove one column
cc_data <- subset(data, select=-c(Time))
str(cc_data)
```

```
'data.frame':   284807 obs. of  30 variables:
 $ V1    : num  -1.36 1.192 -1.358 -0.966 -1.158 ...
 $ V2    : num  -0.0728 0.2662 -1.3402 -0.1852 0.8777 ...
 $ V3    : num  2.536 0.166 1.773 1.793 1.549 ...
 $ V4    : num  1.378 0.448 0.38 -0.863 0.403 ...
 $ V5    : num  -0.3383 0.06 -0.5032 -0.0103 -0.4072 ...
 $ V6    : num  0.4624 -0.0824 1.8005 1.2472 0.0959 ...
 $ V7    : num  0.2396 -0.0788 0.7915 0.2376 0.5929 ...
 $ V8    : num  0.0987 0.0851 0.2477 0.3774 -0.2705 ...
 $ V9    : num  0.364 -0.255 -1.515 -1.387 0.818 ...
 $ V10   : num  0.0908 -0.167 0.2076 -0.055 0.7531 ...
 $ V11   : num  -0.552 1.613 0.625 -0.226 -0.823 ...
 $ V12   : num  -0.6178 1.0652 0.0661 0.1782 0.5382 ...
 $ V13   : num  -0.991 0.489 0.717 0.508 1.346 ...
 $ V14   : num  -0.311 -0.144 -0.166 -0.288 -1.12 ...
 $ V15   : num  1.468 0.636 2.346 -0.631 0.175 ...
 $ V16   : num  -0.47 0.464 -2.89 -1.06 -0.451 ...
 $ V17   : num  0.208 -0.115 1.11 -0.684 -0.237 ...
 $ V18   : num  0.0258 -0.1834 -0.1214 1.9658 -0.0382 ...
 $ V19   : num  0.404 -0.146 -2.262 -1.233 0.803 ...
 $ V20   : num  0.2514 -0.0691 0.525 -0.208 0.4085 ...
 $ V21   : num  -0.01831 -0.22578 0.248 -0.1083 -0.00943 ...
 $ V22   : num  0.27784 -0.63867 0.77168 0.00527 0.79828 ...
 $ V23   : num  -0.11 0.101 0.909 -0.19 -0.137 ...
 $ V24   : num  0.0669 -0.3398 -0.6893 -1.1756 0.1413 ...
 $ V25   : num  0.129 0.167 -0.328 0.647 -0.206 ...
 $ V26   : num  -0.189 0.126 -0.139 -0.222 0.502 ...
 $ V27   : num  0.13356 -0.00898 -0.05535 0.06272 0.21942 ...
 $ V28   : num  -0.0211 0.0147 -0.0598 0.0615 0.2152 ...
 $ Amount: num  149.62 2.69 378.66 123.5 69.99 ...
 $ Class : int  0 0 0 0 0 0 0 0 0 0 ...
```

```r
# Example : select a subset of attributes
new_data <- subset(data, select=c('V2','V4','Class'))
str(new_data)
```

```
'data.frame':   284807 obs. of  3 variables:
 $ V2    : num [1:284807, 1] -0.0441 0.1612 -0.8116 -0.1122 0.5315 ...
 $ V4    : num [1:284807, 1] 0.973 0.317 0.268 -0.61 0.285 ...
 $ Class: int  0 0 0 0 0 0 0 0 0 0 ...
```

For model building, we’ll remove the attribute “Time” as it has a very low correlation with “Class”

(Re-cap from Lecture 3): Splitting the dataset into TRAINING and TESTING datasets

```
# convert class to factor for data modelling
cc_data$Class <- as.factor(cc_data$Class)
```

```
# Set the seed for reproducibility
set.seed(123)

# Split the data into a TRAINING set and TESTING set
train_index <- createDataPartition(cc_data$Class, times = 1, p = 0.8, list =F)
cc_train <- cc_data[train_index,]
cc_test <- cc_data[-train_index,]
```

Set the "seed" so that the results would be reproducible

Split by 80%

```
[1] "Training dataset"

     0      1
227452    394
[1] "Testing dataset"

    0     1
56863    98
```

(Re-cap from Lecture 3) Create a balanced dataset for TRAINING

```
# ROSE
set.seed(9560)
train_rose <- ROSE(Class ~., data=cc_train)$data
```

```
[1] "ROSE sampling result"

     0      1
114081 113765
```

Model training – Random Forest, Decision Tree & Logistic Regression

Model#1: Training the Random Forest data model

Should importance of predictors be assessed?

```
model_rf <- randomForest(Class ~., data=train_rose, importance=TRUE)
model_rf
```

```
Call:
 randomForest(formula = Class ~ ., data = train_rose, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 5

        OOB estimate of  error rate: 0.19%
Confusion matrix:
       0      1  class.error
0 113742    339 0.0029715728
1     84 113681 0.0007383642
```

1.Number of trees used in the forest (ntree) and
2.Number of random variables used in each tree (mtry)

OOB = Out of Bag error = Misclassification rate

What are the Precision, Recall & F-score of this “TRAINING Data”?

Checking performance of the trained Random Forest data model

```
# evaluate performance of the trained model
rf_pred <- predict(model_rf, newdata = cc_test)
# show confusion matrix
confusionMatrix(data=rf_pred, reference=cc_test$Class, positive='1')
```

```
          Reference
Prediction     0     1
         0 56800    14
         1    63    84

               Accuracy : 0.9986
                 95% CI : (0.9983, 0.9989)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : 0.01641

                  Kappa : 0.6851

 Mcnemar's Test P-Value : 4.498e-08

            Sensitivity : 0.857143
            Specificity : 0.998892
         Pos Pred Value : 0.571429
         Neg Pred Value : 0.999754
             Prevalence : 0.001720
         Detection Rate : 0.001475
   Detection Prevalence : 0.002581
      Balanced Accuracy : 0.928017

       'Positive' Class : 1
```

Confusion Matrix

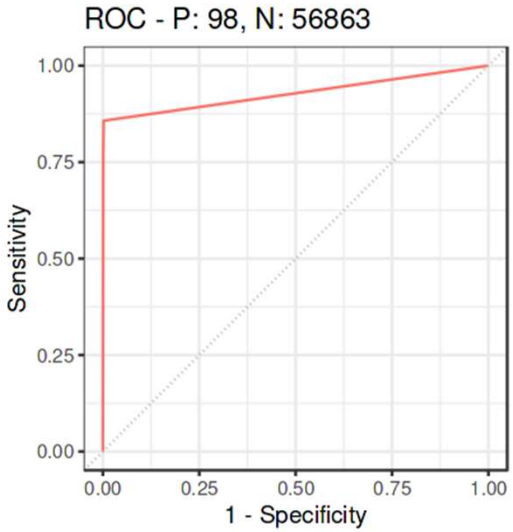
What does it mean when the performance results of training data is MUCH better than testing data?

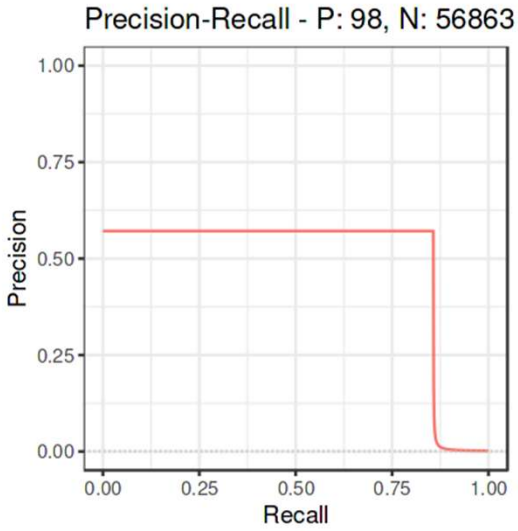
Checking performance of the trained Random Forest data model

```r
# Load the necessary package first
library(precrec)

# Now, your call to evalmod will work
rf_pred <- as.numeric(rf_pred)
rf_prc <- evalmod(scores = rf_pred, labels = cc_test$Class, mode = "rocprc")

# You can then plot the results
autoplot(rf_prc)
rf_prc
```

How to interpret the PRC valule?

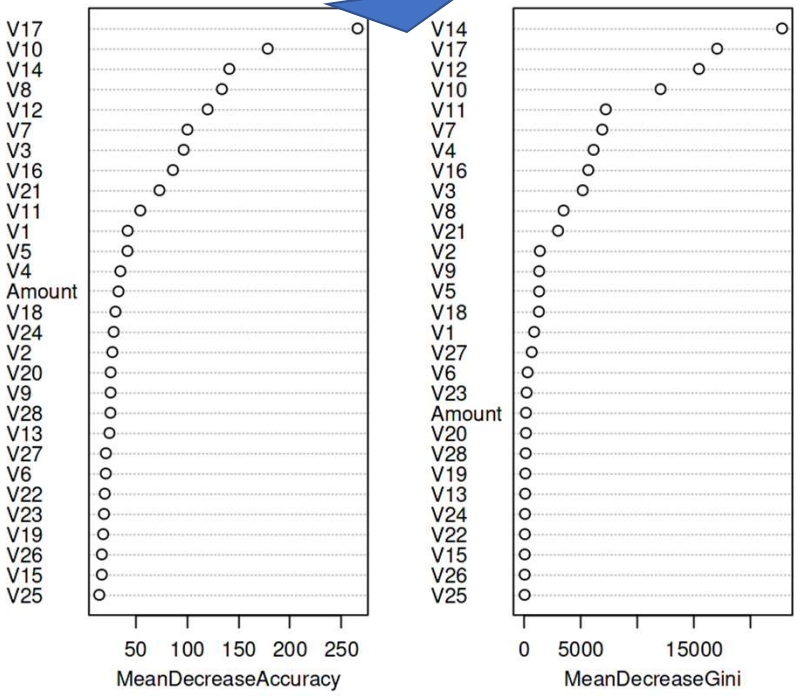
```
=== AUCs ===

  Model name Dataset ID Curve type       AUC
1         m1          1        ROC 0.9280175
2         m1          1        PRC 0.4910894


=== Input data ===

  Model name Dataset ID # of negatives # of positives
1         m1          1          56863              98
```





Review Variable Importance

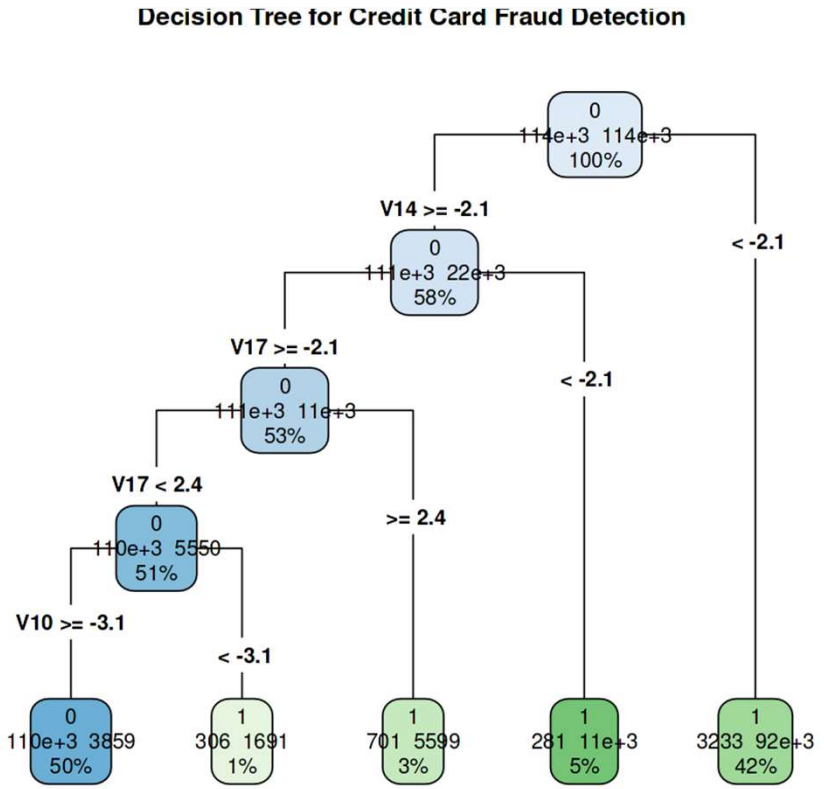
```
# Show the table for variable importance
importance(model_rf)

# Plot the variable importance graph
varImpPlot(model_rf)
```



	0	1	MeanDecreaseAccuracy	MeanDecreaseGini
V1	41.75381	35.1788348	42.17725	914.09891
V2	27.22381	5.9936211	27.32190	1416.08586
V3	96.20931	19.2630843	96.59587	5193.36789
V4	34.43602	28.8151942	35.13708	6150.42049
V5	41.84641	18.4325573	42.03529	1352.08094
V6	19.89637	10.6786058	20.93589	337.69504
V7	99.81098	34.5377909	100.29364	6916.77455
V8	133.03181	41.8661031	133.78654	3507.93871
V9	25.59387	0.4545569	25.62603	1357.87102
V10	177.85382	10.3493496	178.31056	12050.23481
V11	54.37764	-0.4715353	54.46402	7225.85687
V12	119.29456	32.1878985	119.92390	15439.79357
V13	22.59791	12.9683500	24.50356	124.99722
V14	139.66340	53.8425262	140.95547	22758.77212
V15	23.39970	3.8390665	17.08111	86.44051
V16	85.96450	19.7412643	86.20302	5685.70222
V17	263.37407	65.3453929	265.60650	17038.81596
V18	30.27789	4.3171494	30.33099	1332.58905
V19	17.84958	5.6026429	18.39922	141.08788
V20	25.15046	5.2417642	25.67064	192.70061
V21	72.31390	38.5532505	73.06818	3021.87116
V22	19.95383	-0.6867796	19.93675	99.60990
V23	18.84523	11.2720343	19.22489	247.79869
V24	17.95945	23.6328999	28.59616	109.41427
V25	14.70397	3.7930676	14.60244	75.51223
V26	12.28061	13.2504342	17.09303	80.88610
V27	20.83217	13.5521673	21.02153	701.44828
V28	21.01856	17.6616448	25.45854	169.20803
Amount	33.03964	-3.6710119	33.29033	193.20240

Model#2: Training the Decision Tree data model

```
# --- Build the Decision Tree Model ---
# We'll build a classification tree to predict 'Class' based on all other variables.
# The 'method = "class"' tells rpart we are doing a classification task.
tree_model <- rpart(Class ~ ., data = train_rose, method = "class")
```




```
# --- Visualize the Decision Tree ---
# 'type=4' and 'extra=101' create a plot that is easy to read.
# 'fallen.leaves=TRUE' arranges the terminal nodes at the bottom for clarity.
rpart.plot(tree_model,
           type = 4,
           extra = 101,
           fallen.leaves = TRUE,
           main = "Decision Tree for Credit Card Fraud Detection")
```

Checking performance of the trained Decision Tree data model

```
# Use the trained model to make predictions on the unseen test data.
tree_pred <- predict(tree_model, cc_test, type = "class")

# Show confusion matrix
confusionMatrix(data=tree_pred, reference=cc_test$Class, positive='1')
```

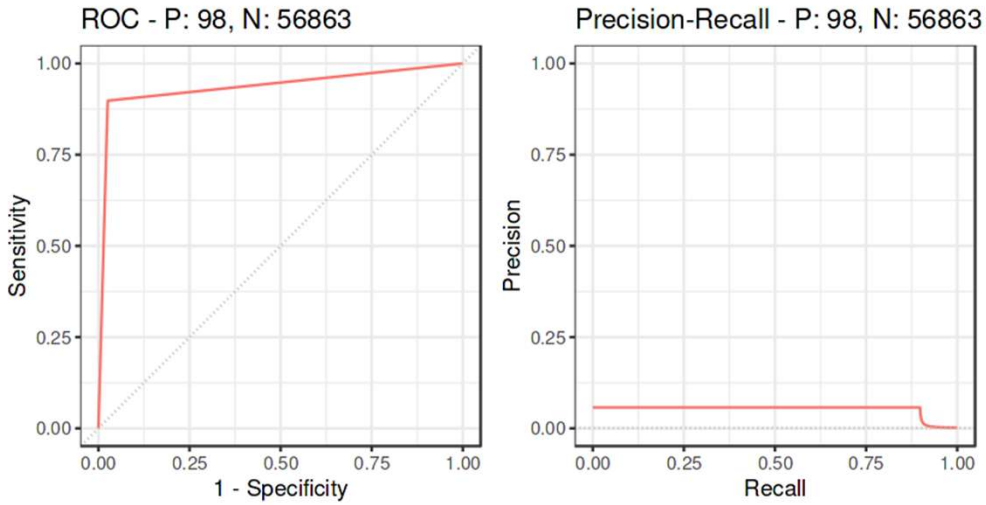
What are the key performance metrics?

```
          Reference
Prediction     0     1
         0 55415    10
         1  1448    88

               Accuracy : 0.9744
                 95% CI : (0.9731, 0.9757)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1048

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.897959
            Specificity : 0.974535
         Pos Pred Value : 0.057292
         Neg Pred Value : 0.999820
             Prevalence : 0.001720
         Detection Rate : 0.001545
   Detection Prevalence : 0.026966
      Balanced Accuracy : 0.936247

       'Positive' Class : 1
```

Checking performance of the trained Decision Tree data model

```
tree_pred <- as.numeric(tree_pred)
tree_prc <- evalmod(scores = tree_pred, labels = cc_test$Class, mode = "rocprc")

# You can then plot the results
autoplot(tree_prc)
tree_prc
```

```
 === AUCs ===

  Model name Dataset ID Curve type        AUC
1         m1          1        ROC 0.93624724
2         m1          1        PRC 0.05204753


 === Input data ===

  Model name Dataset ID # of negatives # of positives
1         m1          1          56863              98
```



Model#3: Training the Logistic Regression data model

```
# Using Logistic Regression as the modelling method
fit.lm <- glm(Class ~ ., data = train_rose, family = binomial(link="logit"))

# Show the LR result
summary(fit.lm)
```

```
Call:
glm(formula = Class ~ ., family = binomial(link = "logit"), data = train_rose)

Coefficients:
             Estimate Std. Error  z value Pr(>|z|)
(Intercept) -2.300684   0.010627 -216.500  < 2e-16 ***
V1          -0.032868   0.002383  -13.795  < 2e-16 ***
V2           0.069260   0.003352   20.663  < 2e-16 ***
V3          -0.081127   0.002505  -32.392  < 2e-16 ***
V4           0.468996   0.004231  110.851  < 2e-16 ***
V5           0.038154   0.003042   12.541  < 2e-16 ***
V6          -0.191502   0.005262  -36.396  < 2e-16 ***
V7          -0.040977   0.002555  -16.041  < 2e-16 ***
V8          -0.054661   0.002454  -22.277  < 2e-16 ***
V9          -0.132829   0.005318  -24.976  < 2e-16 ***
V10         -0.160337   0.003571  -44.896  < 2e-16 ***
V11          0.299599   0.005040   59.442  < 2e-16 ***
V12         -0.235614   0.003693  -63.804  < 2e-16 ***
V13         -0.167385   0.006649  -25.174  < 2e-16 ***
V14         -0.380932   0.003858  -98.726  < 2e-16 ***
V15         -0.027754   0.007160   -3.876 0.000106 ***
V16         -0.106495   0.004218  -25.245  < 2e-16 ***
V17         -0.039588   0.002505  -15.800  < 2e-16 ***
V18          0.010802   0.005214    2.072 0.038272 *
V19         -0.075016   0.006704  -11.189  < 2e-16 ***
V20         -0.040268   0.007581   -5.312 1.08e-07 ***
V21          0.026259   0.004162    6.310 2.80e-10 ***
V22          0.098960   0.007882   12.556  < 2e-16 ***
V23         -0.087500   0.006274  -13.947  < 2e-16 ***
V24         -0.104059   0.012230   -8.508  < 2e-16 ***
V25          0.002464   0.011541    0.213 0.830969
V26         -0.194458   0.014271  -13.626  < 2e-16 ***
V27          0.067022   0.011236    5.965 2.45e-09 ***
V28          0.203158   0.019474   10.432  < 2e-16 ***
Amount       0.213156   0.007184   29.672  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 315861  on 227845  degrees of freedom
Residual deviance: 100454  on 227816  degrees of freedom
AIC: 100514

Number of Fisher Scoring iterations: 8
```

Checking performance of the trained Logistic Regression data model

```r
# For the Confusion Matrix, we need the final 0/1 class predictions.
# This uses the fixed 0.5 threshold
lr_pred_class <- ifelse(predict(fit.lm, newdata = cc_test, type = 'response') > 0.5, 1, 0)
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 56160    11
         1   703    87

               Accuracy : 0.9875
                 95% CI : (0.9865, 0.9884)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1935

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.887755
            Specificity : 0.987637
         Pos Pred Value : 0.110127
         Neg Pred Value : 0.999804
             Prevalence : 0.001720
         Detection Rate : 0.001527
   Detection Prevalence : 0.013869
      Balanced Accuracy : 0.937696

       'Positive' Class : 1
```

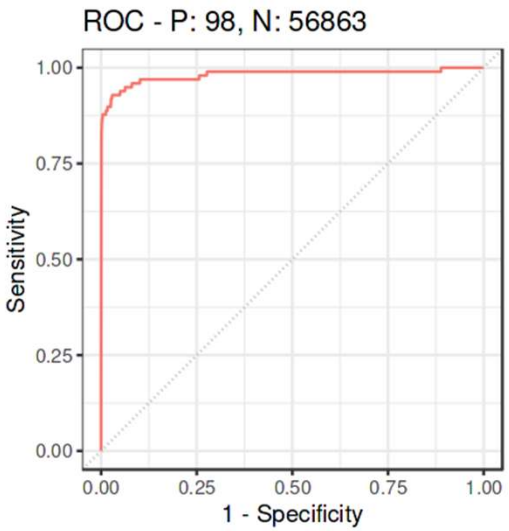
```r
# The confusionMatrix() function from the 'caret' package provides a detailed table
# including the matrix, accuracy, precision, recall, and F1-score.
# Note: Ensure both predicted classes and actual classes are factors.
# We set `positive = "1"` to get metrics for the "fraud" class.

conf_matrix <- confusionMatrix(
  data = as.factor(lr_pred_class),
  reference = as.factor(cc_test$Class),
  positive = "1"
)

# Print the full confusion matrix and associated metrics
print(conf_matrix)
```

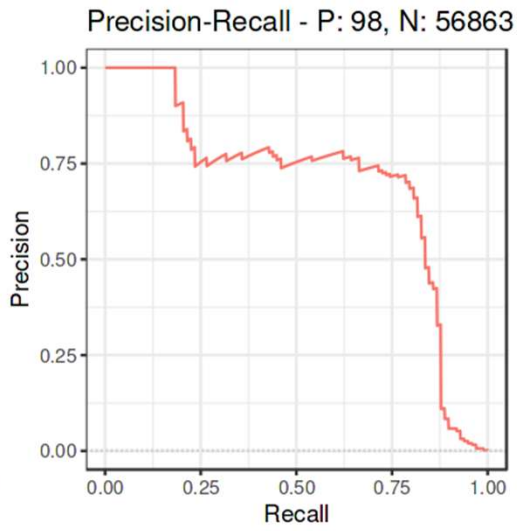
What are the key performance metrics?

Checking performance of the trained Logistic Regression data model

```
# Now, your call to evalmod will work
lr_pred_prob <- as.numeric(lr_pred_prob)
lr_prc <- evalmod(scores = lr_pred_prob, labels = cc_test$Class, mode = "rocprc")

# You can then plot the results
autoplot(lr_prc)
lr_prc
```

```
=== AUCs ===

  Model name Dataset ID Curve type       AUC
1         m1          1        ROC 0.9810971
2         m1          1        PRC 0.6985883


=== Input data ===

  Model name Dataset ID # of negatives # of positives
1         m1          1          56863              98
```





Model comparison

	Accuracy	**Precision**	**Recall**	**F1-score**	**ROC AUC**	**PRC AUC**
RF						
DT						
LR						

Key Takeaways

Key Takeaways (Lecture 3 & 4)

- Random Forest brings tangible value in high-volume, imbalanced fraud settings
- Must always interpret results and guard against false confidence
- Deployment is not the end—ongoing monitoring, business judgment, and adversarial defense are vital
- Transfer of skills: ensemble learning, metrics, model interpretation apply across fraud domains

The Analyst's Core Trade-Off:

As a fraud analyst, you will always navigate the tension between Model Performance and Model Interpretability. Your job is not just to build the most accurate model, but to build the most *effective* and *trustworthy* model that the business can confidently act upon.

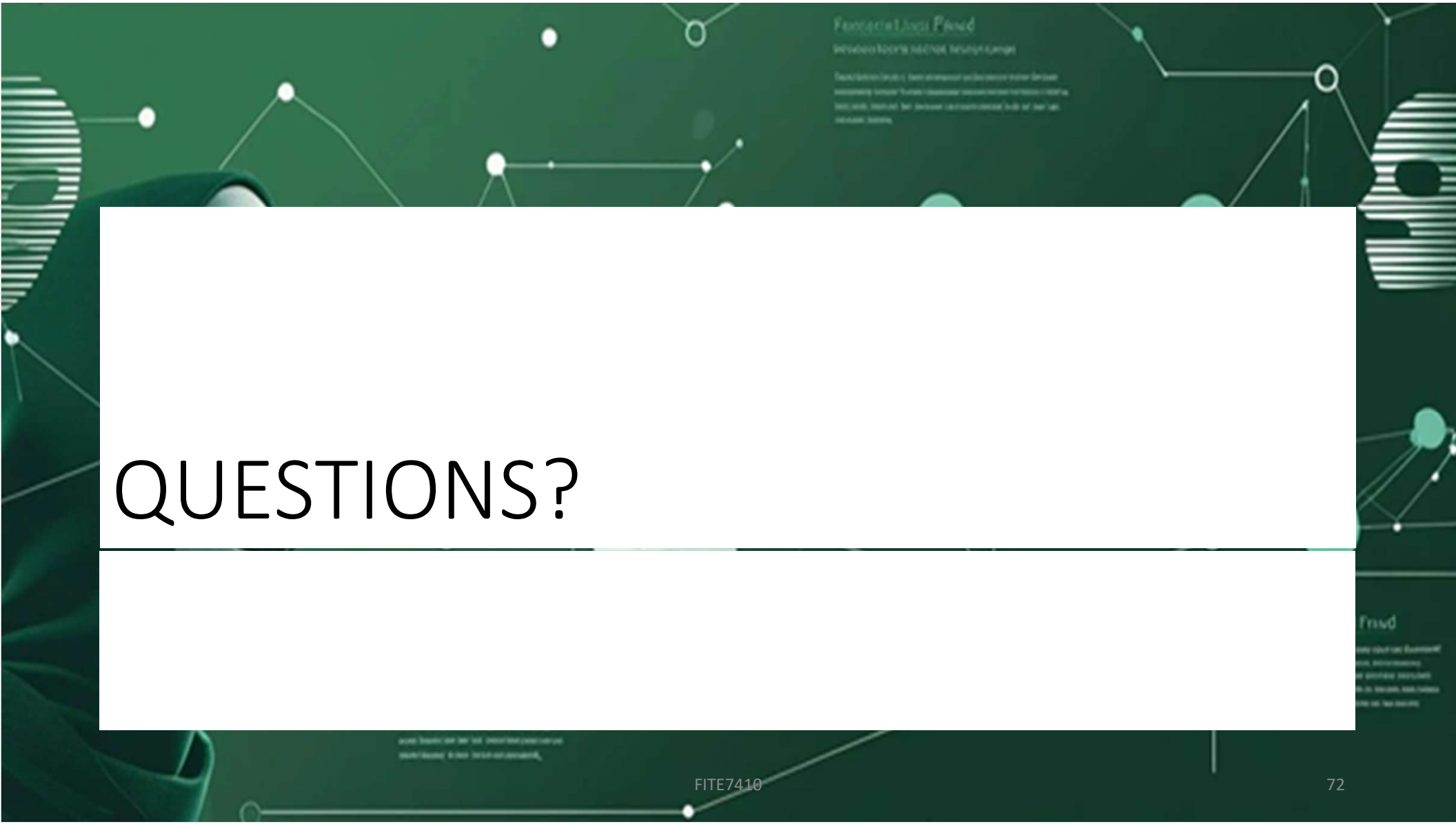
Some final thoughts – beyond the numbers

- Question 1: Is the Model Foolproof?
  - If fraudsters know that algorithms are looking at these specific ratios, what might they do next?
  - How could a model like this be deceived?

- Question 2: Deployment Decision
  - You are the head of internal audit. Based on this evidence, would you deploy this Random Forest model?
  - What operational challenges would you anticipate?

References

- Bart Baesens, Veronique Van Vlasselaer, Wouter Verbeke (2015). Fraud Analytics using Descriptive, Predictive, and Social Network Techniques, 1st ed, John Wiley & Sons Inc.
- Leonard W. Vona (2017). Fraud Data Analytics Methodology: The Fraud Scenario Approach to Uncovering Fraud in Core Business Systems, John Wiley & Sons, Inc.
- Perols, J. (2011). Financial statement fraud detection: An analysis of statistical and machine learning algorithms. *Auditing: A Journal of Practice & Theory, 30*(2), 19–50.



QUESTIONS?