

Statistical Computing with R: Masters in Data Sciences 503 (S25) Fourth Batch, SMS, TU, 2025

Shital Bhandary

Associate Professor

Statistics/Bio-statistics, Demography and Public Health Informatics

Patan Academy of Health Sciences, Lalitpur, Nepal

Faculty, Masters in Medical Research, NHRC/Kathmandu University

Faculty, FAIMER Fellowship in Health Professions Education, India/USA

Supervised Learning with classification models

- Decision Tree

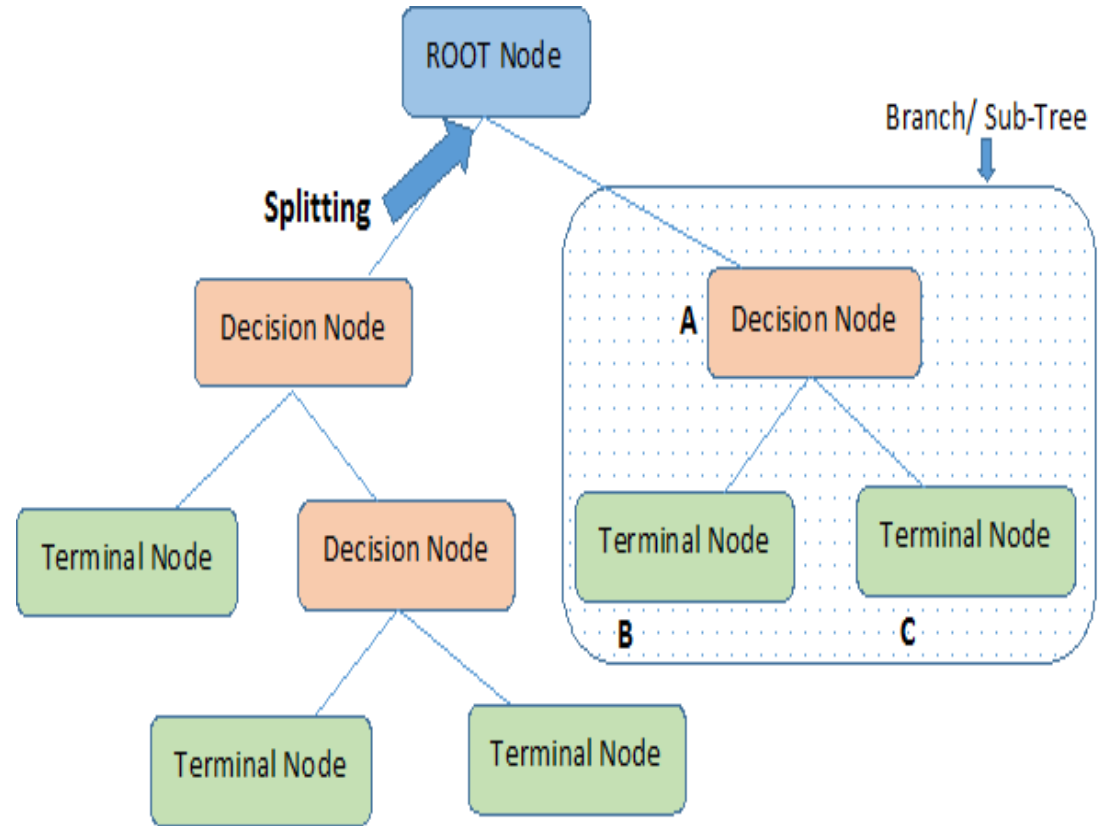
- Model Accuracy
- Model Prediction
- Model Validation

- Ensemble learning

- Bagging
- Random Forest: Improved Decision Tree Bagging
- Boosting
- Random Forest Boosting

Decision Tree: More at ““An Introduction to Statistical Learning with Applications in R” book!

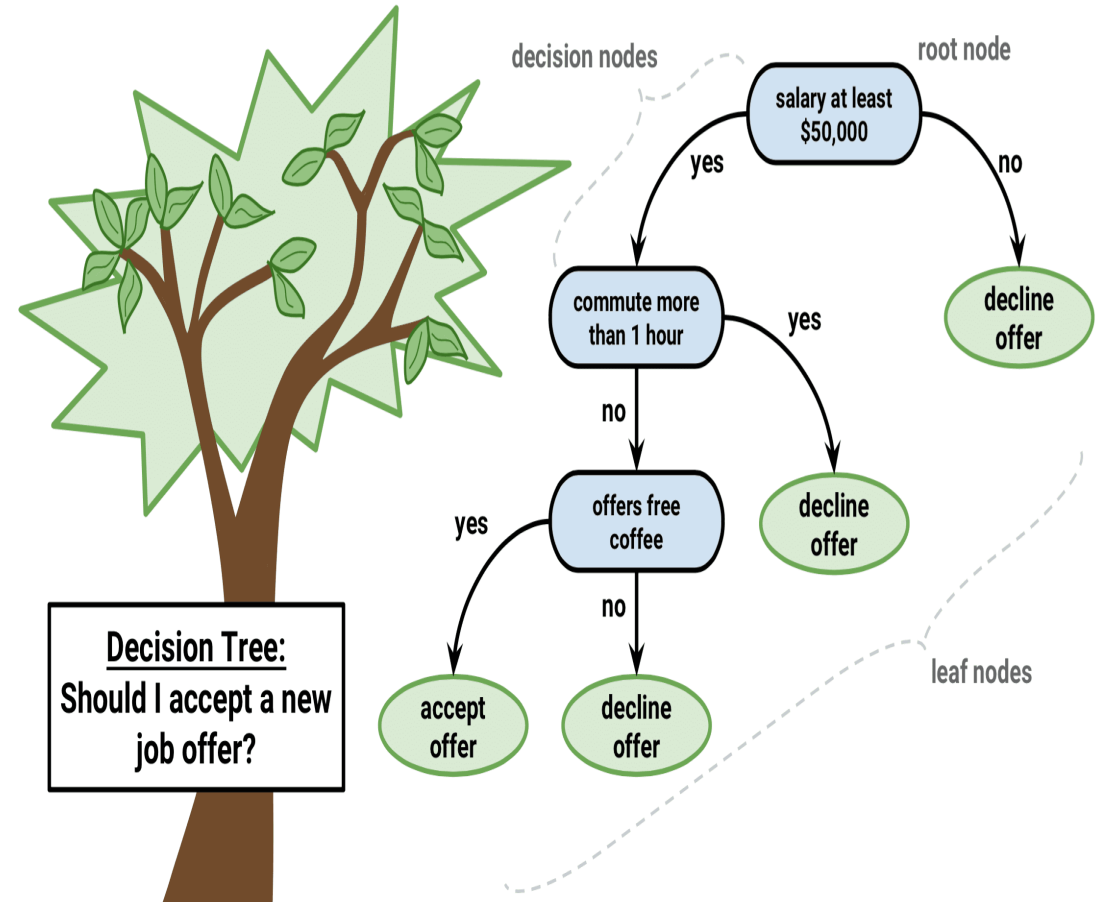
- Decision tree is a type of supervised learning algorithm that can be used in both regression and classification problems.
- It works for both categorical and continuous input and output variables.
- **Today we will focus on the classification tree**



Note:- A is parent node of B and C.

Classification tree:

- Classification Error Rate
- Gini Index
- Cross-Entropy
- **Confusion matrix accuracy and diagnostic measures**



Let's fit a decision tree classifier in the Cardiotocographic (CTG) data:

#Make sure the data is in the working library of R

- library(readr)
- Cardiotocographic <- read_csv("Cardiotocographic.csv")

#Save the long name as data

- data <- Cardiotocographic
- str(data)

#The structure correctly shows LB, AC and FM as numerical but incorrectly show NSP as numeric too!

- spec_tbl_df [2,126 x 22] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
- \$ LB : num [1:2126] 120 132 133 134 132 134 134 122 122 122 ...
- \$ AC : num [1:2126] 0 0.00638 0.00332 0.00256 0.00651 ...
- \$ FM : num [1:2126] 0 0 0 0 0 0 0 0 0 0 ...
- \$ NSP : num [1:2126] 2 1 1 1 1 3 3 3 3 ...

Change the NSP as factor variable as this is the dependent variable:

#NSP = Dependent variable (response)

#Three categories of NSP

- NSP = 1 = Normal CTG (No hypoxia or acidosis)
- NSP = 2 = Suspicious CTG (Low probability of hypoxia/acidosis)
- NSP = 3 = Pathological CTG (High probability of hypoxia/acidosis)

#NSP categories were formed using
“**Updated 2015 FIGO Intrapartum Fetal Monitoring Guidelines**” (figo.org)
(Using [SisPorto](#) algorithm)

#Changing NSP as a new factor variable i.e. NSPF

```
data$NSPF <- factor(data$NSP)
```

#Checking structure of NSPF

```
str(data$NSPF)
```

- Factor w/ 3 levels "1","2","3": 2 1 1 1 1 3 3 3 3 3 ...










#Key Independent variables (inputs)

- LB = Fetal Heart Rate
- AC = # of accelerations / minute
- FM = # of fetal movements / minute

Let's divide the data into train and test sets:

#Data partition

- `set.seed(1234)`
- `ind <- sample(2, nrow(data),
 replace=T, prob = c(0.8, 0.2))`
- `train <- data[ind==1,]`
- `test <- data[ind==2,]`

Environment	History	Connections	Tutorial
<div><div></div><div> Import Dataset ▾</div><div></div></div>			
<div> Global Environment ▾</div>			
Data			
<div> Cardiotocographic</div>	2126 obs. of 22 variables		
<div> data</div>	2126 obs. of 23 variables		
<div> test</div>	408 obs. of 23 variables		
<div> train</div>	1718 obs. of 23 variables		

Let's fit a decision tree classifier in the train data:

#Party package

- `install.packages("party")`
- `library(party)`

#Fit the decision tree model with only the first three variable to learn it today

- `tree <- ctree(NSPF ~ LB+AC+FM, data=train)`

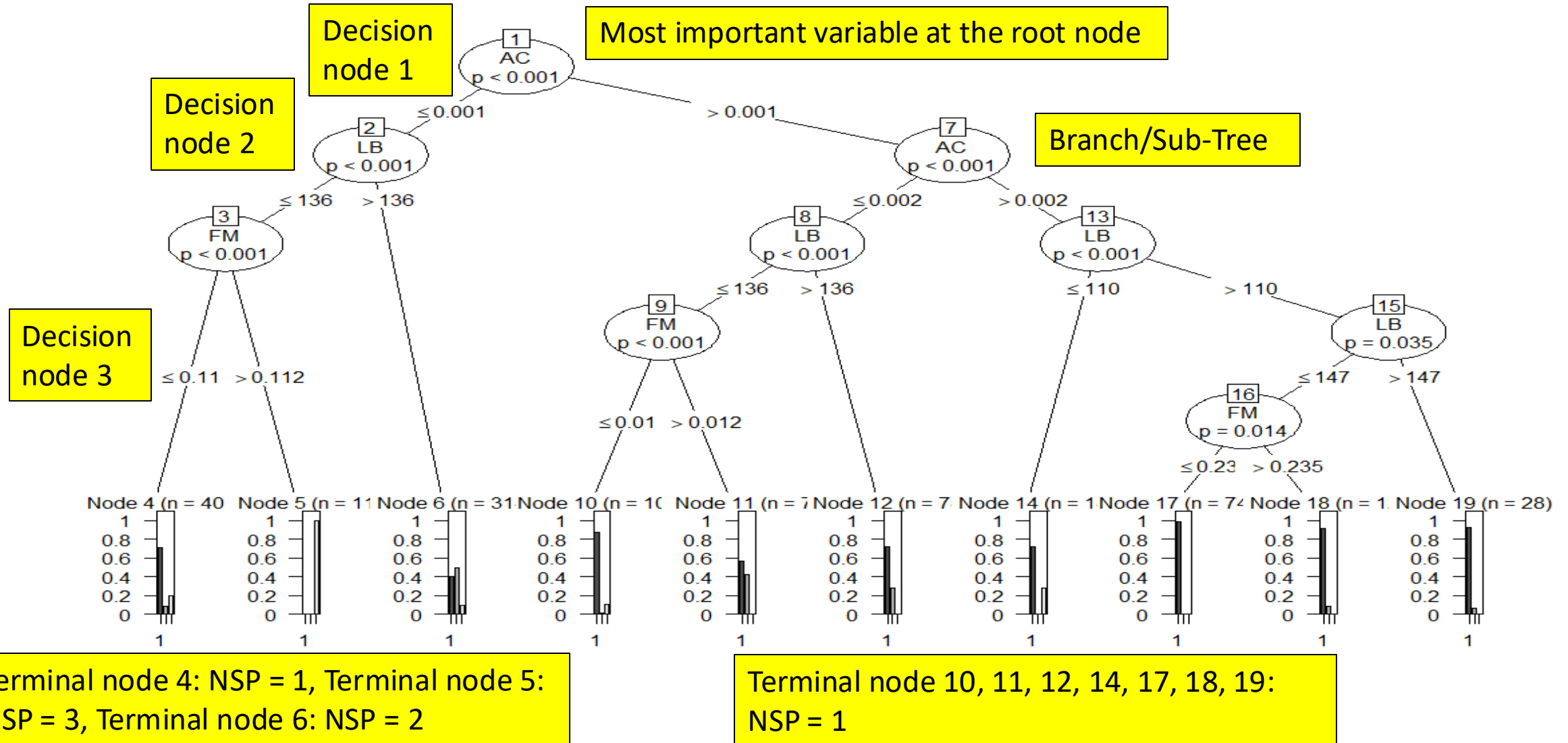
#Check the model

- `tree`

Conditional inference tree with 10 terminal nodes

- Response: NSPF
- Inputs: LB, AC, FM
- Number of observations: 1718
 - 1) $AC \leq 0.000834028$; criterion = 1, statistic = 263.403
 - 2) $LB \leq 136$; criterion = 1, statistic = 131.511
 - 3) $FM \leq 0.111898$; criterion = 1, statistic = 35.729

Let's plot the decision tree: `plot(tree)`



For predicting 3 variables we have a “tree” with 19 nodes! So, it is a “overfitting”!

#Pruning the tree with 99% confidence interval and split at 500 samples

- ```
tree1 <- ctree(NSPF ~ LB+AC+FM,
data=train, controls =
ctree_control(mincriterion=0.99,
minsplit=500))
```
- tree1

**#Now we have only 9 nodes in the model!**

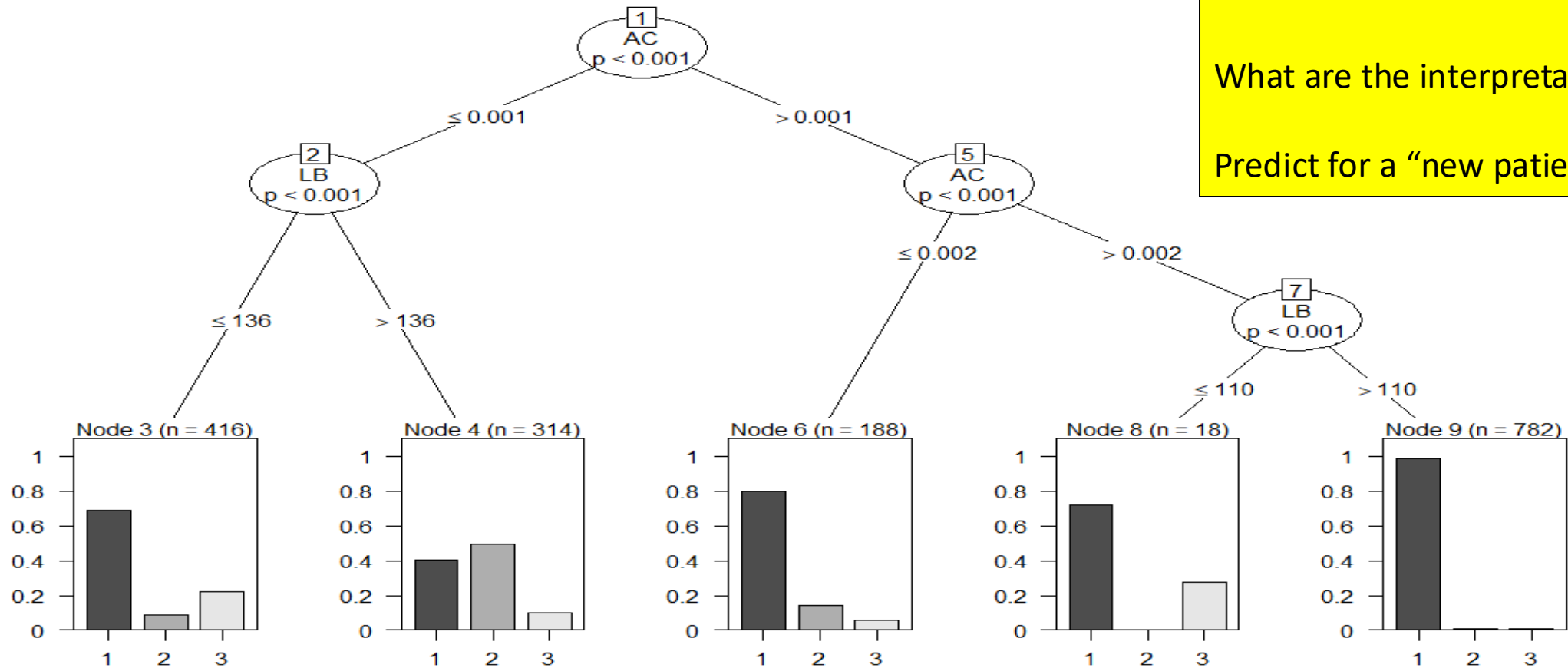
**Lets plot it and see how it helps us!**

**Conditional inference tree with 5 terminal nodes (total 9 nodes)**

- Response: NSPF
- Inputs: LB, AC, FM
- Number of observations: 1718
  - 1)  $AC \leq 0.000834028$ ; criterion = 1, statistic = 263.403
  - 2)  $LB \leq 136$ ; criterion = 1, statistic = 131.511
  - 3)\* weights = 416
  - ...
  - 7)  $LB > 110$
  - 9)\* weights = 782

This is why we need to use ensemble learning i.e. “Random Forest” to avoid overfitting for Decision Tree Model!

# Controlled Decision Tree: `plot(tree1)`



There is no FM variable now!

What are the interpretations?

Predict for a “new patient”!

# Let's predict now:

# Predict the categorical probabilities for each case in train data

- `predict(tree, type="prob")`

#Predict the category for each case in test data

- `predict(tree, test)`

**#Predicted probability: case 1**

- `[[1]]`
- **0.71111111 (prob. for NSP=1)**
- 0.09135802 (prob. for NSP=2)
- 0.19753086 (prob. for NSP = 3)

**#Predicted category for cases:**

```
[1] 1 1 1 1 1 3 1 1 1 1 1 2 1 1 2 1 1 1 1
1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1
[40] 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 2
2 2 2 2 2 1 2 2 2 2 1 1 1 2 2 1 1 2 2 2 1
```

Let's get the confusion matrix, accuracy and the misclassification error (mce) for "train" data

### #Confusion matrix

- `(tab <- table(predict(tree),  
train$NSPF))`

### #Accuracy

- `accuracy <-  
sum(diag(tab))/sum(tab)`

### #Misclassification error

- `mce <- 1 - accuracy`

### #Confusion matrix

|   | 1           | 2          | 3         |
|---|-------------|------------|-----------|
| 1 | <b>1222</b> | 70         | 101       |
| 2 | 126         | <b>156</b> | 32        |
| 3 | 0           | 0          | <b>11</b> |

### #Accuracy

`[1] 0.8084983`

### #Misclassification error

`[1] 0.1915017`

NSP classification  
was done by 3  
expert clinicians!

We need to optimize  
the model so that we  
can do prediction  
with confidence with  
the model rather  
than clinicians!

# Let's get the confusion matrix, accuracy and the misclassification error (mce) for "test" data

## #Prediction for test data

```
pred.test <- predict(tree,
newdata=test)
```

## #Confusion matrix

- (tab1 <- table(pred.test,  
test\$NSPF))
- #Accuracy
- accuracy1 <-  
sum(diag(tab1))/sum(tab1)
- #Misclassification error
- mce1 <- 1 - accuracy1

## #Confusion matrix

| pred.test | 1          | 2         | 3        |
|-----------|------------|-----------|----------|
| 1         | <b>274</b> | 20        | 27       |
| 2         | 33         | <b>48</b> | 4        |
| 3         | 0          | 1         | <b>1</b> |

## #Accuracy

```
[1] 0.7916667
```

**(Good, not excellent!)**

## #Misclassification error

```
[1] 0.2083333
```

We must optimize the model so that we can do prediction with confidence with the model rather than clinicians!

# Limitations of Decision Tree model:

<https://www.datacamp.com/community/tutorials/decision-trees-R>

- The decision trees discussed above suffer from *high variance*, meaning if you split the training data into 2 parts at random, and fit a decision tree to both halves, the results that you get could be quite different.
- In contrast, a procedure with *low variance* will yield similar results **if applied repeatedly** to distinct dataset.
- **We must use ensemble learning!**
  - Bagging, Random Forests and Boosting

- **Bagging**, or *bootstrap aggregation*, is a technique used to reduce the variance of your predictions by **combining the result of multiple classifiers modeled on different sub-samples** of the same dataset.
- Here is the equation for bagging:

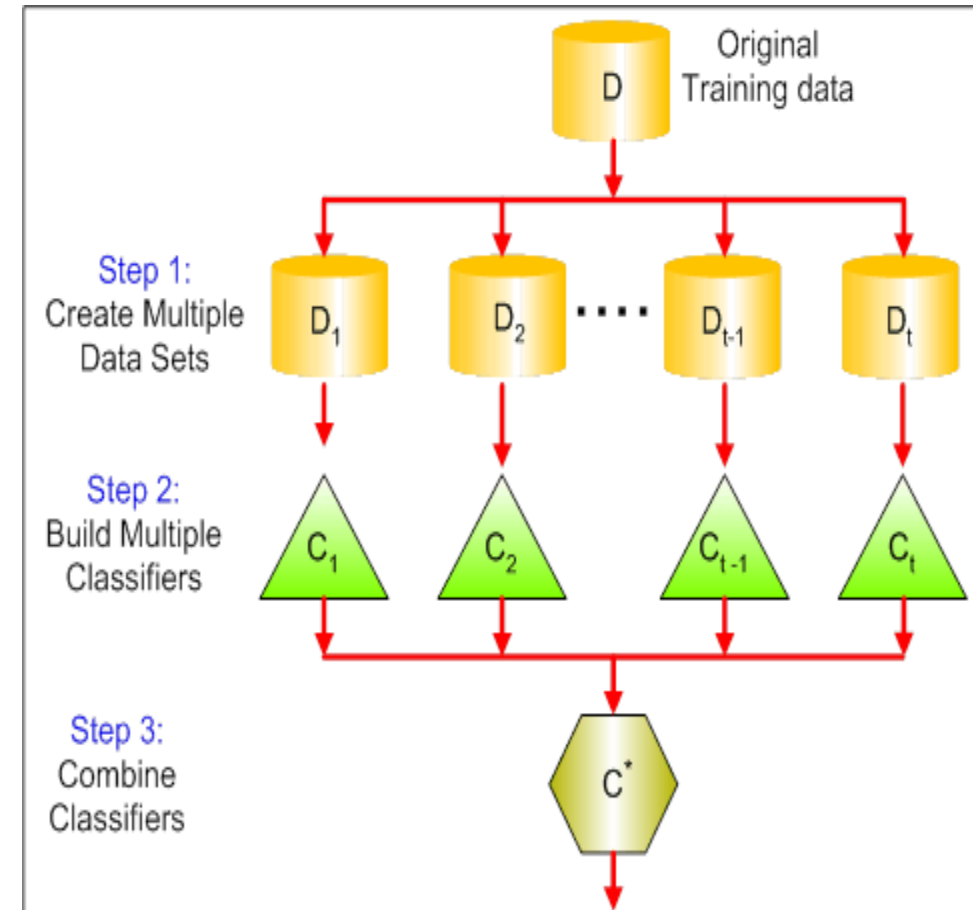
$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}_b(x)$$

Where, we generate B number of bootstrapped (resampled) training datasets and choose one with “vote”.

# Bagging:

<https://www.datacamp.com/community/tutorials/decision-trees-R>

- **Step 1:** Here you replace the original data with new data. The new data usually have a fraction of the original data's columns and rows, which then can be used as hyper-parameters in the bagging model.
- **Step 2:** You build classifiers on each dataset. Generally, you can use the same classifier for making models and predictions.
- **Step 3:** Lastly, you use an average value to combine the predictions of all the classifiers, depending on the problem. Generally, these combined values are more robust than a single model.





# Bagging in R with “ipred” package for CTG data (NSP = dependent variable):

#Bagging with ipred package

#install.packages(“ipred”) if required

- library(ipred)
- MBTree <- bagging(NSPF~LB+AC+FM, **data = train**, coob=T)
- print(MBTree)

#Prediction

- MBPredict1 <- predict(MBTree, test)
- MBPredict1

#Confusion matrix and accuracy

- confusionMatrix(MBPredict1, test\$NSPF)

- Bagging classification trees with 25 bootstrap replications
- Call: bagging.data.frame(formula = NSPF ~ LB + AC + FM, data = train, coob = T)
- Out-of-bag estimate of misclassification error: **0.1717**

## • Confusion Matrix and Statistics

|            | Reference |    |    |
|------------|-----------|----|----|
| Prediction | 1         | 2  | 3  |
| 1          | 273       | 26 | 17 |
| 2          | 25        | 37 | 2  |
| 3          | 9         | 6  | 13 |

Accuracy : 0.7917, 95% CI : (0.749, 0.8301) **(Same!)**

$$\text{MCE} = 1 - 0.7917 = 0.2083 = 20.83\%$$

# Random Forest: Improved Bagging model

<https://www.datacamp.com/community/tutorials/decision-trees-R>

- Random Forest provides an **improvement over bagged trees** by a small tweak that *decorrelates* the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, **a random sample of  $m$  predictors ( $mtry$ )** is chosen as split candidates from the full set of  $p$  predictors.
- **The split is allowed to use only one of those  $m$  predictors**
- This is the **main difference between random forests and bagging**; because as in bagging, the choice of predictor  **$m = p$  i.e. all the predictors are used.**

# Random Forest Requisites:

<https://www.datacamp.com/community/tutorials/decision-trees-R>

- First assume that the number of cases in the training set is  $K$ .
- Then, take a random sample of these  $K$  cases, and then use this sample as the training set for growing the tree.
- If there are  $p$  input variables, specify a number  $m < p$  such that at each node, we can select  $m$  random variables out of the  $p$ .
- The best split (votes) on these  $m$  is used to split the node.
- Each tree is subsequently grown to the **largest extent** possible and **no pruning** is needed.
- Finally, aggregate the predictions of the target trees to predict new data.
- **Random Forest package in R used 500 trees as “default”!**

# Advantages of Random Forests model:

<https://www.datacamp.com/community/tutorials/decision-trees-R>

- Random Forests is very effective at estimating missing data and maintaining accuracy when a large proportions of the data is missing.
- It can also balance errors in datasets where the **classes are imbalanced**.
- **Most importantly, it can handle massive datasets with large dimensionality.**
- However, one disadvantage of using Random Forests is that you **might easily “overfit” noisy datasets**, especially in the case of doing regression.

# Let's fit the Random Forest model now:

```
#Random Forests package
#install.packages("randomForest")
• library(randomForest)

#Setting seed for replication
set.seed(222)

#Fitting random forest model
• rfm <- randomForest(NSPF~ LB+AC+FM,
 data=train)

#Printing random forest model
• print(rfm)
```

**Type of random forest: classification**

Number of trees: 500

No. of variables tried at each split: 1

**OOB estimate of error rate: 17.35%**

Confusion matrix:

|    | 1           | 2          | 3         | class.error       |
|----|-------------|------------|-----------|-------------------|
| 1  | <b>1295</b> | 52         | 1         | 0.03931751        |
| 2  | 119         | <b>104</b> | 3         | <b>0.53982301</b> |
| 3. | 92          | 31         | <b>21</b> | <b>0.85416667</b> |

**Accuracy = (1295+104+21)/(1718)= 82.83% (Good!)**

**MCE = 1 – 0.8283 = 0.1717 = 17.17%**

# Prediction using “caret” package:

## #Let us use caret to predict

- library(caret)
- p1 <- predict(rfm, train)
- head(p1)

#Let us see the original values  
head(train\$NSPF)

```
> head(p1)
```

```
Cases: 1 2 3 4 5 6
```

```
Prediction: 1 1 1 1 1 1
```

- NSP Levels: 1 2 3

```
> head(train$NSPF)
```

```
• Cases: 1 2 3 4 5 6
```

```
• [1] 2 1 1 1 1 3
```

- NSP Levels: 1 2 3

For the first six cases, prediction using train data is “not perfect”!

# Let's check the accuracy and misclassification error with Confusion Matrix for "train" data:

- #Confusion matrix
- confusionMatrix(p1, train\$NSPF)

- Accuracy : **0.8661**  
95% CI : (0.8491, 0.8819)

Confusion Matrix and Statistics

|            | Reference |     |    |
|------------|-----------|-----|----|
| Prediction | 1         | 2   | 3  |
| 1          | 1304      | 82  | 79 |
| 2          | 44        | 144 | 25 |
| 3          | 0         | 0   | 40 |

Class: 1 Class: 2 Class: 3

|             |               |                |                |
|-------------|---------------|----------------|----------------|
| Sensitivity | <b>0.9674</b> | 0.63717        | 0.27778        |
| Specificity | 0.5649        | <b>0.95375</b> | <b>1.00000</b> |

- $MCE = 1 - 0.8661 = 0.1339$
- Fitted model in the train data is nearly 87% correct!
- Accuracy of fitted model in the test data = ??% correct??

# Let's check the accuracy and misclassification error with Confusion Matrix for "test" data:

#Prediction for test data

- `p2 <- predict(rfm, test)`

#Confusion matrix

- `confusionMatrix(p2, test$NSP)`

## Confusion Matrix and Statistics

| Prediction | Reference |    |    |
|------------|-----------|----|----|
|            | 1         | 2  | 3  |
| 1          | 293       | 31 | 27 |
| 2          | 14        | 38 | 2  |
| 3          | 0         | 0  | 3  |

- Accuracy : **0.8186 (Better!)**

- 95% CI : (0.7777, 0.8548)

Class: 1 Class: 2 Class: 3

- Sensitivity 0.9544 0.55072 0.09375

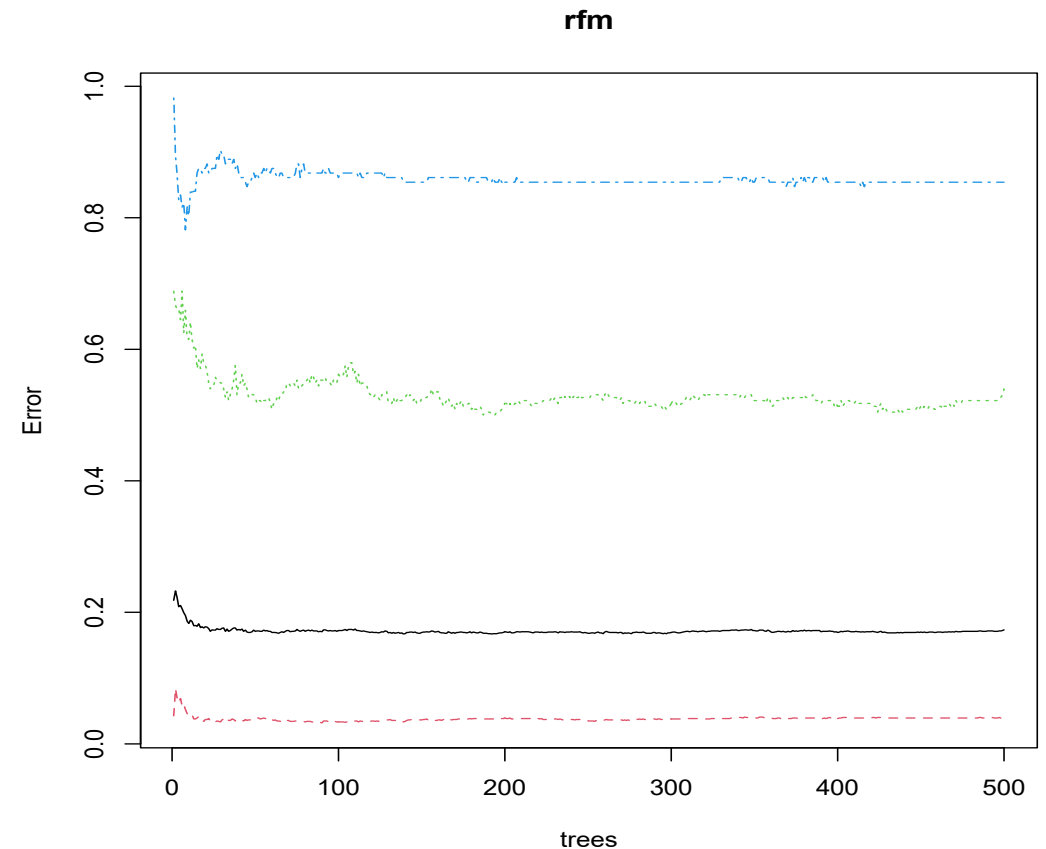
- Specificity 0.4257 0.95280 1.00000

- **MCE = 1 – 0.8186 = 0.1814 = 18.14% OOB error?**



# Let us evaluate the error and see if we can improve the model with it: `plot(rfm)`

- Here error reduced substantially till 200 trees
- Error did not change between 200-300 trees
- This means the default 500 trees are not required to get the lowest error



# Let us use the original data for the subsequent steps:

- NSP as factor instead of creating new variable NSPF
- Then divide this data into two partitions with 80% and 20% data
  - train = 80% random data
  - test = 20% random data

# Can we improve the random forest model?

## Yes. We can tune the random forest model!

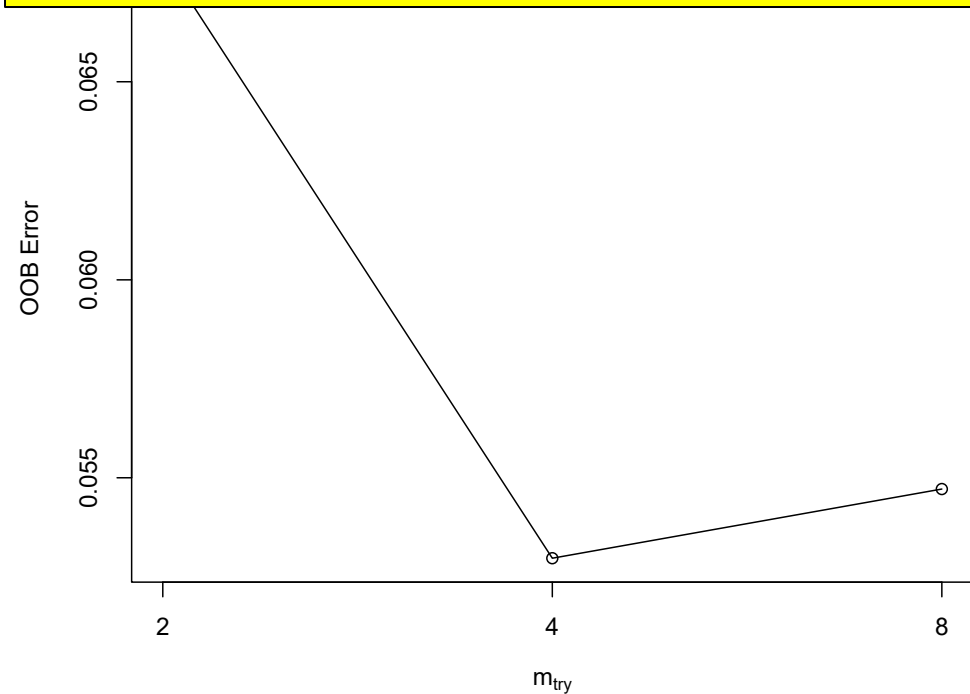
**# We can use “tuneRF” function of the randomForest package but we need to define train as data frame**

- `train <- as.data.frame(train)`

**# We can use the “tuneRF” now**

- `t <- tuneRF(train[,-22], train[,22],`
- `stepFactor = 0.5,`
- `plot = TRUE,`
- `ntreeTry = 300,`
- `trace = TRUE,`
- `improve = 0.05)`

Here `mtry` split parameter i.e. number of variables used in each split have lowest OOB error at 4, with 300 trees instead of default 500 trees!



# Lets fit the improved (fineTuned) random forest model and check the OOB error:

## # Improved “rfm” model:

- `rfm1 <- randomForest(NSP~ LB+AC+FM, data=train,`
- `ntree = 300,`
- `mtry = 4,`
- `importance = TRUE,`
- `proximity = TRUE)`
- `print(rfm1)`

## Call:

```
randomForest(formula = NSP ~ LB + AC + FM, data = train, ntree = 300, mtry = 4, importance = TRUE, proximity = TRUE)
```

Type of random forest: classification

Number of trees: 300

No. of variables tried at each split: 3

OOB estimate of error rate: **16.3%**

**(improved a bit but not by much!)**

- Accuracy=?? MSE = ???

- **(Better than RandomForest or Not?)**

# Boosting: Better than Random Forest?

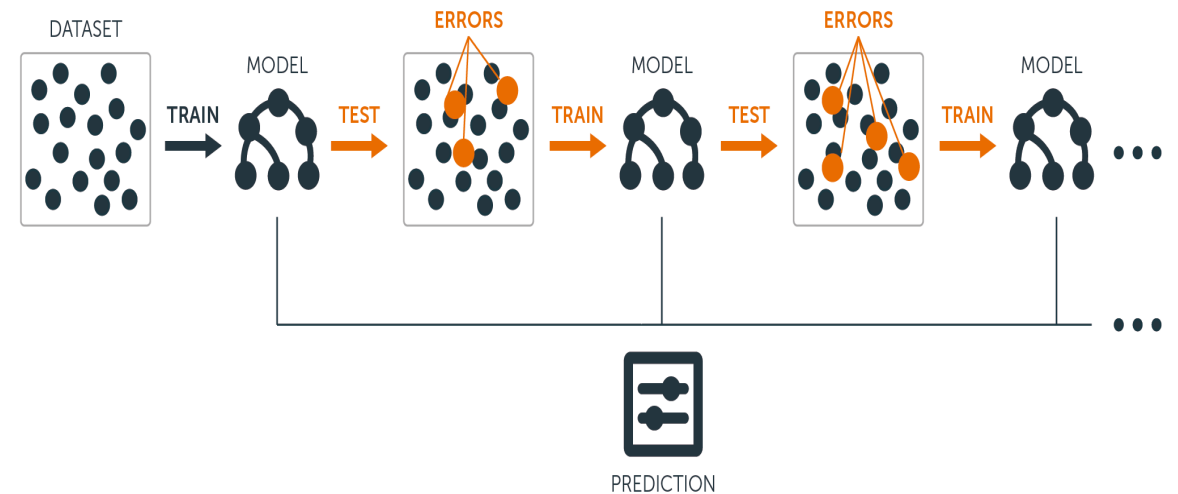
<https://www.datacamp.com/community/tutorials/decision-trees-R>

- **Boosting** is another approach to improve the predictions resulting from a decision tree.
- Like bagging and random forests, it is a general approach that can be applied to many statistical learning methods for regression or classification.
- Recall that bagging involves creating multiple copies of the original training dataset using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrapped dataset, **independent of the other trees.**

# How Boosting works?

<https://www.datacamp.com/community/tutorials/decision-trees-R>

- Boosting works in a similar way, except that the trees are grown *sequentially*: each tree is grown using information from previously grown trees.
- Boosting **does not involve bootstrap sampling**; instead, each tree is fitted on a modified version of the original dataset.



# Boosting with Generalized Boosting Regression Model (GBM): <https://rpubs.com/drwatkins/286356>

## #Generalized Boosting Regression Model

```
library(caret)
mod.gbm <-
train(NSP~.,data=train,method="gbm",verbose=
F)
print(mod.gbm)
```

## #Prediction and confusion matrix

```
pred.mod.gbm <-
predict(mod.gbm,test,data=test)
confusionMatrix(pred.mod.gbm, test$NSP)
```

## #Variable importance

```
summary(mod.gbm)
```

## # Confusion Matrix and Statistics

|            | Reference |    |    |
|------------|-----------|----|----|
| Prediction | 1         | 2  | 3  |
| 1          | 288       | 31 | 23 |
| 2          | 15        | 36 | 2  |
| 3          | 4         | 2  | 7  |

Accuracy : **0.8113 (Lower than Random Forest!)**

95% CI : (0.7699, 0.8481)

Error:  $1 - 0.8113 = 0.1887 = \mathbf{18.87\%}$

**Boosting with GBM did not improve the model!**

**Because: Our problem is classification not the regression, so this is GIGO!**

Boosting of random forest model using “xgboostTree” model with caret package in R: **Takes time to fit!**

### **#Boosting with extreme gradient boosting Tree model in caret**

```
mod.xgb <-
train(NSP~.,data=train,method="xgb
Tree",verbose=FALSE)
print(mod.xgb)
summary(mod.xgb)
```

### **#Prediction and confusion matrix**

```
pred.mod.xgb <-
predict(mod.xgb,test,data=test)
confusionMatrix(pred.mod.xgb,
test$NSP)
```

### **Confusion Matrix and Statistics**

**Reference**

| <b>Prediction</b> | <b>1</b>   | <b>2</b>  | <b>3</b>  |
|-------------------|------------|-----------|-----------|
| <b>1</b>          | <b>287</b> | <b>29</b> | <b>23</b> |
| <b>2</b>          | <b>14</b>  | <b>38</b> | <b>2</b>  |
| <b>3</b>          | <b>6</b>   | <b>2</b>  | <b>7</b>  |

Accuracy : 0.8137

**(Lower than Random Forest Model)**

OOB Error =  $1 - 0.8137 = 0.1853 =$   
**18.53%**



# The “best” model?

- Improved random forest model?
- Why?
- What is the lesson learnt?

# Self-Learning

- **Try to fit the decision tree, improved decision tree, random forest and boosting models for the same data using all the variables in the model**
- Which model gives the best result?
- Why?

# Question/queries?

- Next class
- Unsupervised learning
  - Dimensionality Reduction
- Factor analysis
  - Principal Component Analysis
  - Exploratory Factor Analysis
  - Confirmative Factor Analysis

# Thank you!

@shitalbhandary