Boosting algorithms

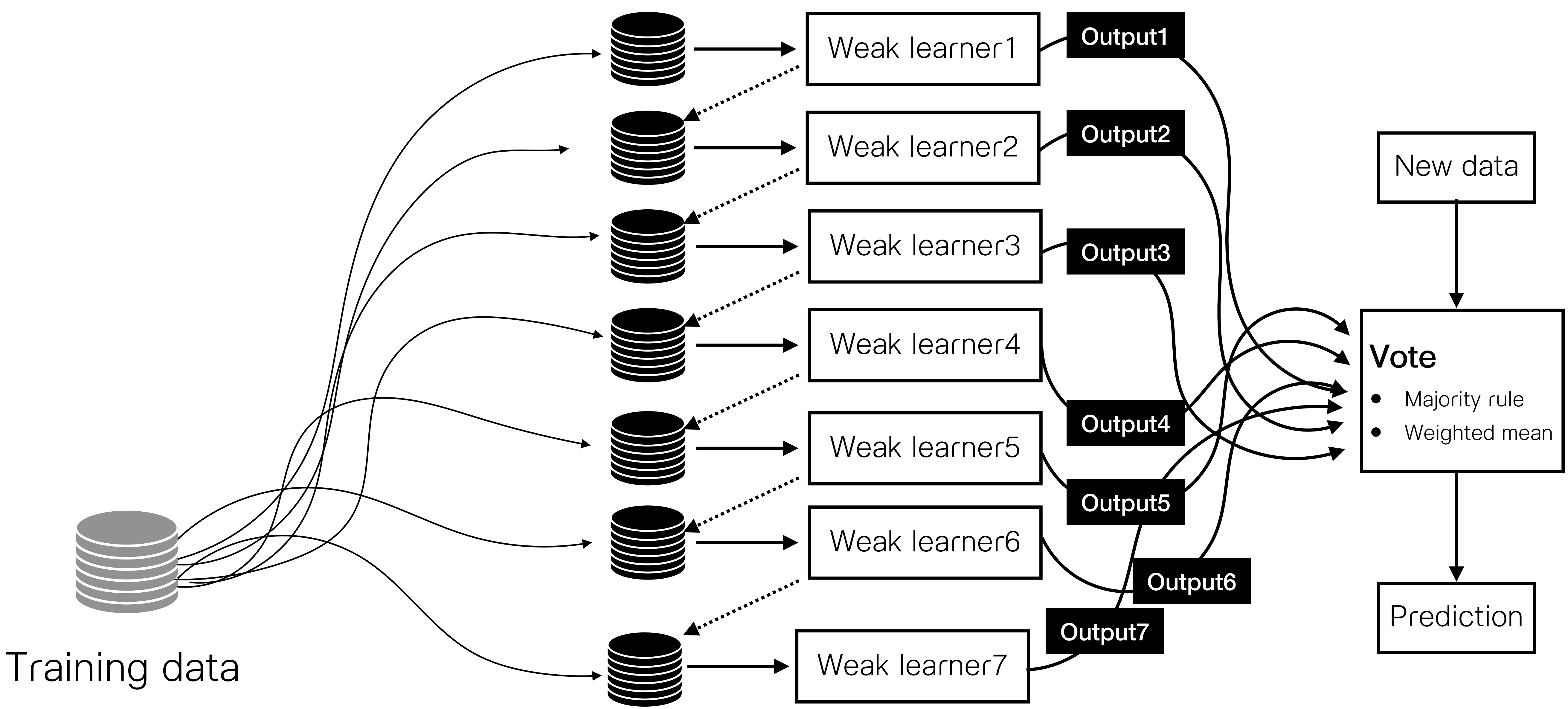
What's Boosting in ML?

How Boosting algorithms work?

Types of Boosting algorithm.

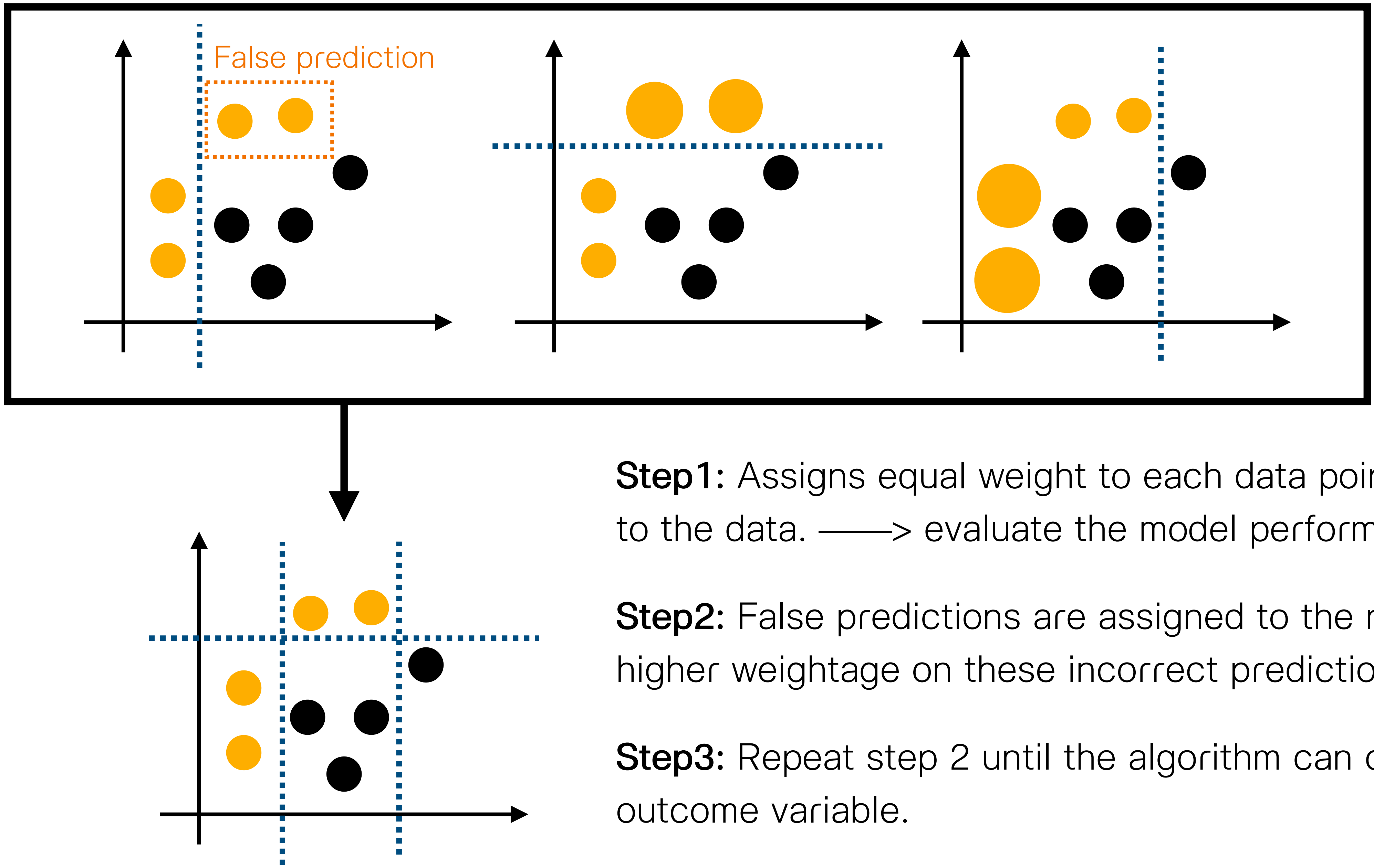
Boosting algorithms

- Boosting is an ensemble learning techniques that uses a set of ML algorithms to **combines weak learner** (or base learner) to **form a strong learner** in order to increase the accuracy of the model.
- During the training phase, the performance of the model is improved by assigning a higher weight to the previous incorrectly classified subjects.



How does Boosting algorithm work?

Basic concept behind the working of the boosting algorithm is to generate multiple weak learner and combine their predictions to form one strong rule



Step1: Assigns equal weight to each data point and fit the ML model to the data. ——> evaluate the model performance

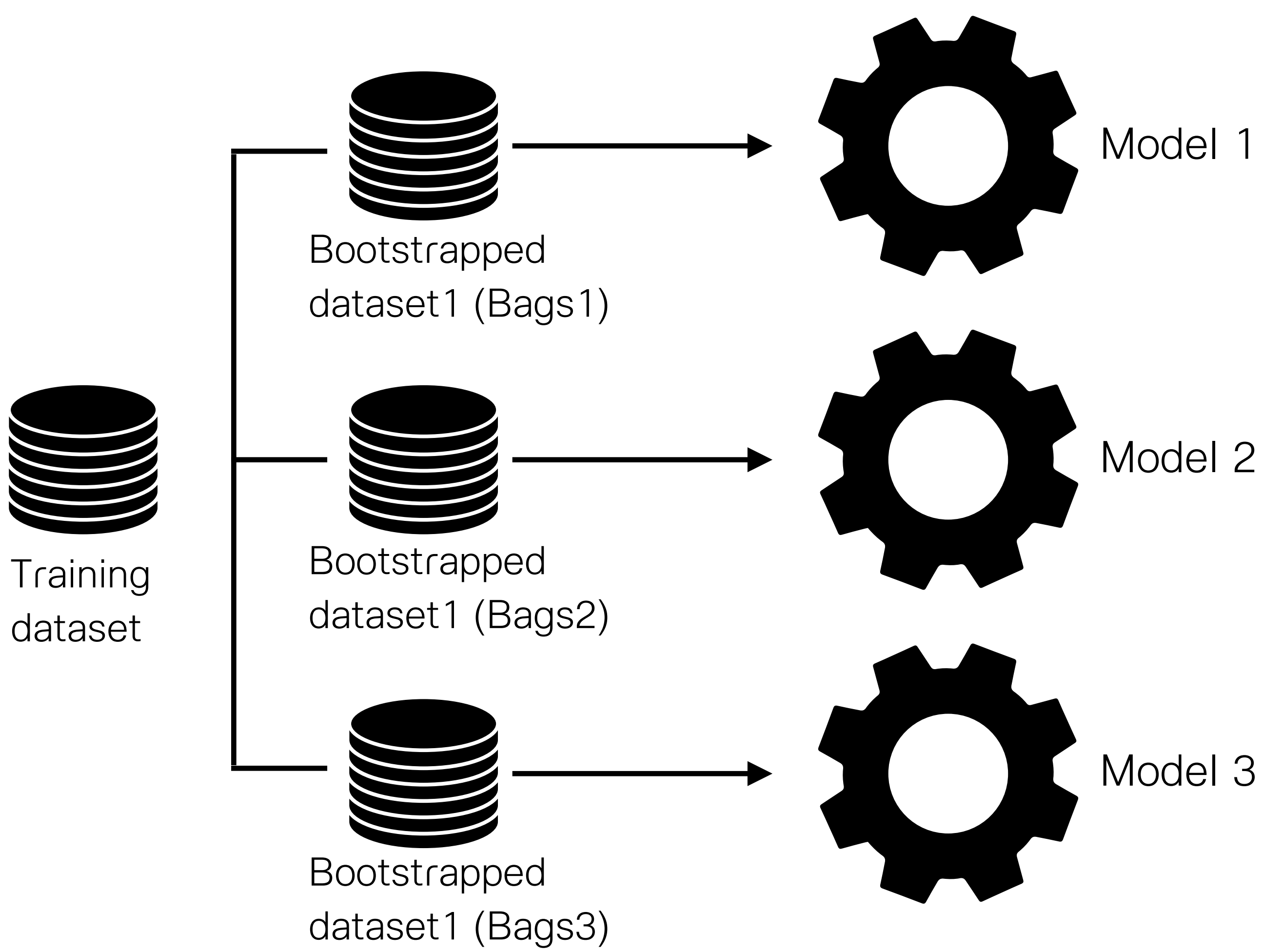
Step2: False predictions are assigned to the next dataset with higher weightage on these incorrect predictions.

Step3: Repeat step 2 until the algorithm can correctly predict the outcome variable.

Ensemble learning

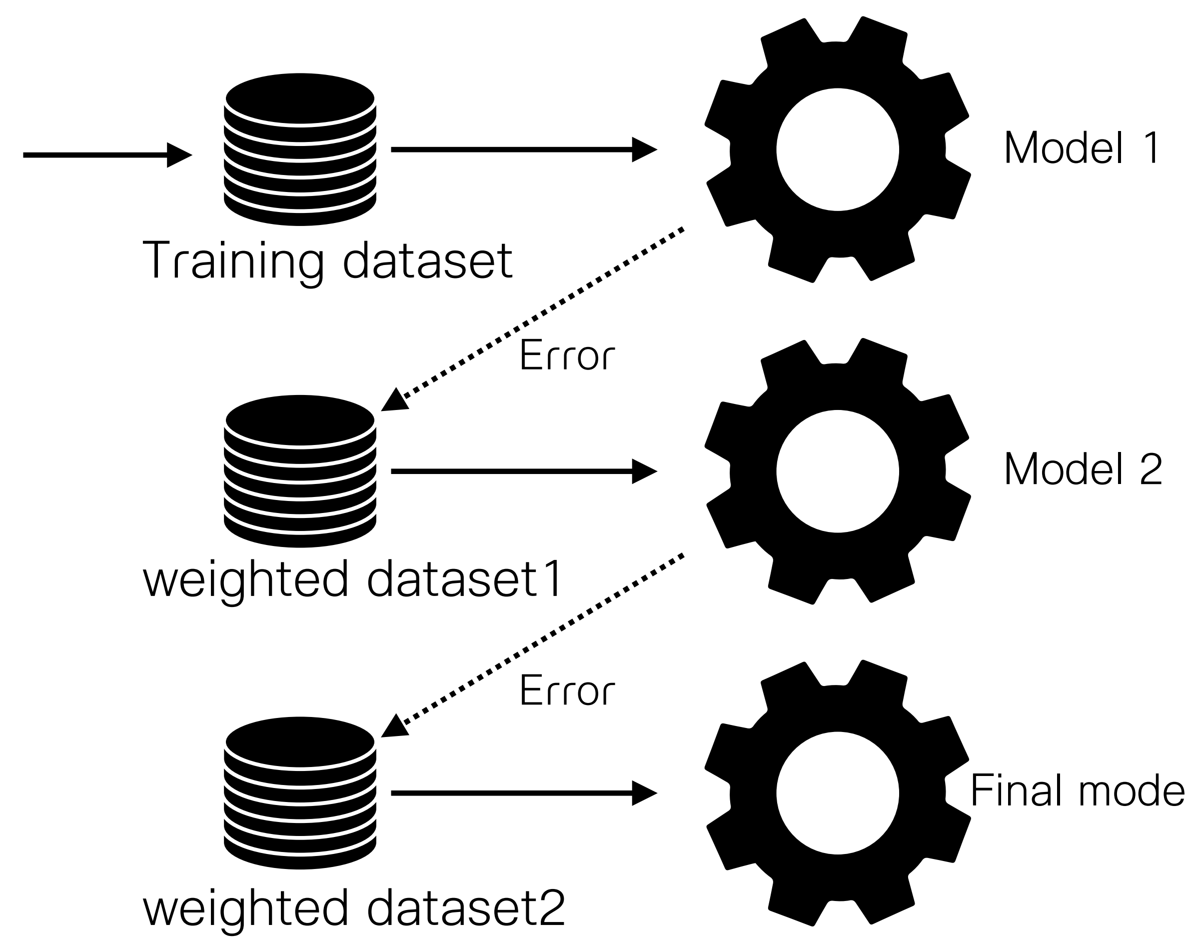
Bagging - Parallel

- Build multiple ML models using same algorithm with subset of training dataset randomly selected from the full training dataset



Boosting - Sequential

- Perform iterative process to reduce errors of previous models by selecting points which give wrong predictions and try to predict them with successive model.



Types of Boosting algorithms

1. AdaBoost (Adaptive Boosting)
2. Gradient Boosting
3. XGBoost

AdaBoost

AdaBoost (Adaptive boosting)

Step1: Each data point is weighted equally for the first decision stump.

Step2: Mis-classified data points are assigned higher weights

Step3: A new decision stump is drawn by considering the data points with higher weights as more significant.

No

Is all data points fall into the correct class?

Yes

Stop

Note: AdaBoost can also be used on a regressions problem, but it's most commonly seen in classification problems.

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.314&rep=rep1&type=pdf

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.4560&rep=rep1&type=pdf

Original AdaBoost algorithm

1. Given a training dataset (X, y) which contains data points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$
2. Construct distribution of W_t on $\{1, 2, 3, \ldots, m\}$, where $W_t(i)$ is the weight attributed to subject i on the iteration t
3. Initial weight are calculated by $W_1(i) = 1/m$, and the next weight are calculated by

$$W_{t+1}(i) = W_t(i) F_i / Z_t$$

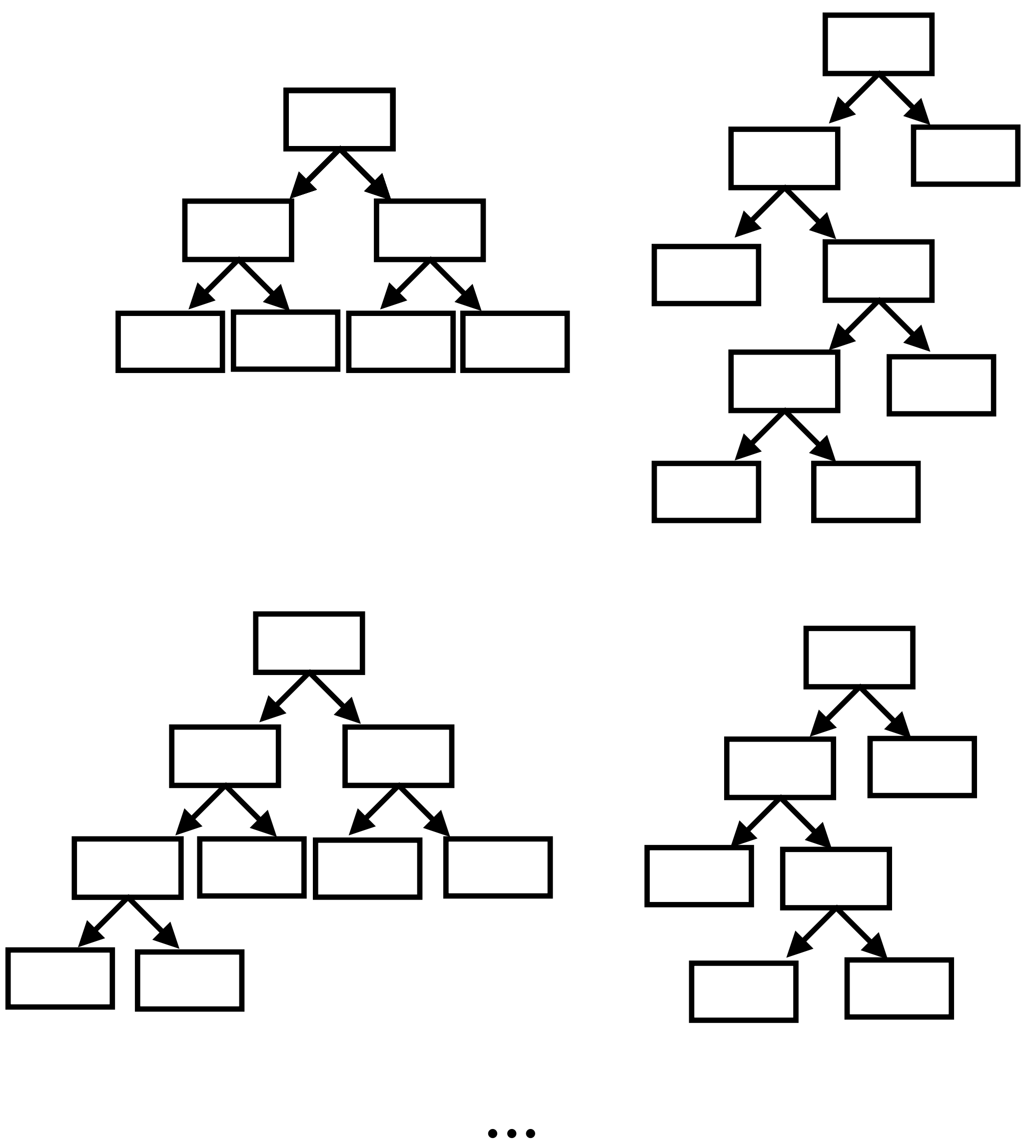
- Where $F_i = e^{-\alpha_t}$ if $M_t(i) \neq y_i$, and $F_i = e^{\alpha_t}$ if $M_t(i) = y_i$
- $\epsilon_t = \sum_{i=1}^{m} W_t(i) \times \delta_i$ where $\underbrace{\delta_i = 0 \text{ if } M_t(i) = y_i}_{\text{Correct classified}}$, and $\underbrace{\delta_i = 1 \text{ if } M_t(i) \neq y_i}_{\text{Mis-classified}}$
- $\alpha_t = \frac{1}{2} ln(\frac{1 - \epsilon_t}{\epsilon_y})$

4. Build up a classifier (weak learner) $M_t : X \rightarrow \{-1, 1\}$
5. The final classification considered using the weighted average of the classifier

$$sign(\sum_{t=1}^{T} \alpha_t M_t(x))$$

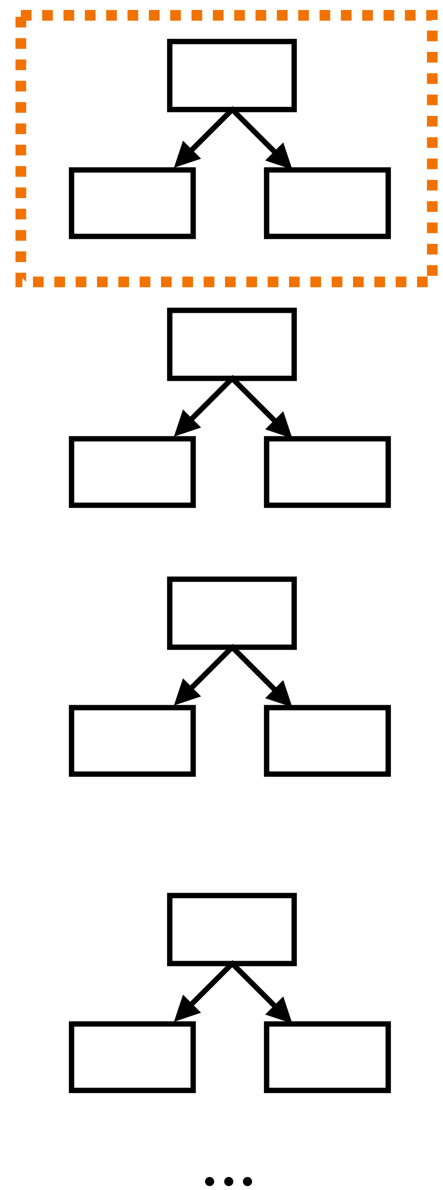
http://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf

Main concepts behind AdaBoost



Random forest

Called “stump”



AdaBoost

- Technically, stumps are weak learner, that is not great at making accurate classifications. **AdaBoost** combined a lot of weak learners to make classification.
- In a random forest, each decision tree is made independently of the others. In contrast, Forest of stumps made with **Adaboost, order is matter.**(the error that the first stump makes influence how the second stump is made, and …. so on …)
- Hence some stump will get more weight in the classification than others.

STA	LOC	SYS	AGE
Dead	No	36	27
Dead	No	48	59
Dead	Yes	44	77
Dead	No	62	54
Alive	Yes	112	87
Alive	Yes	108	69
Alive	No	140	63
Alive	Yes	138	30

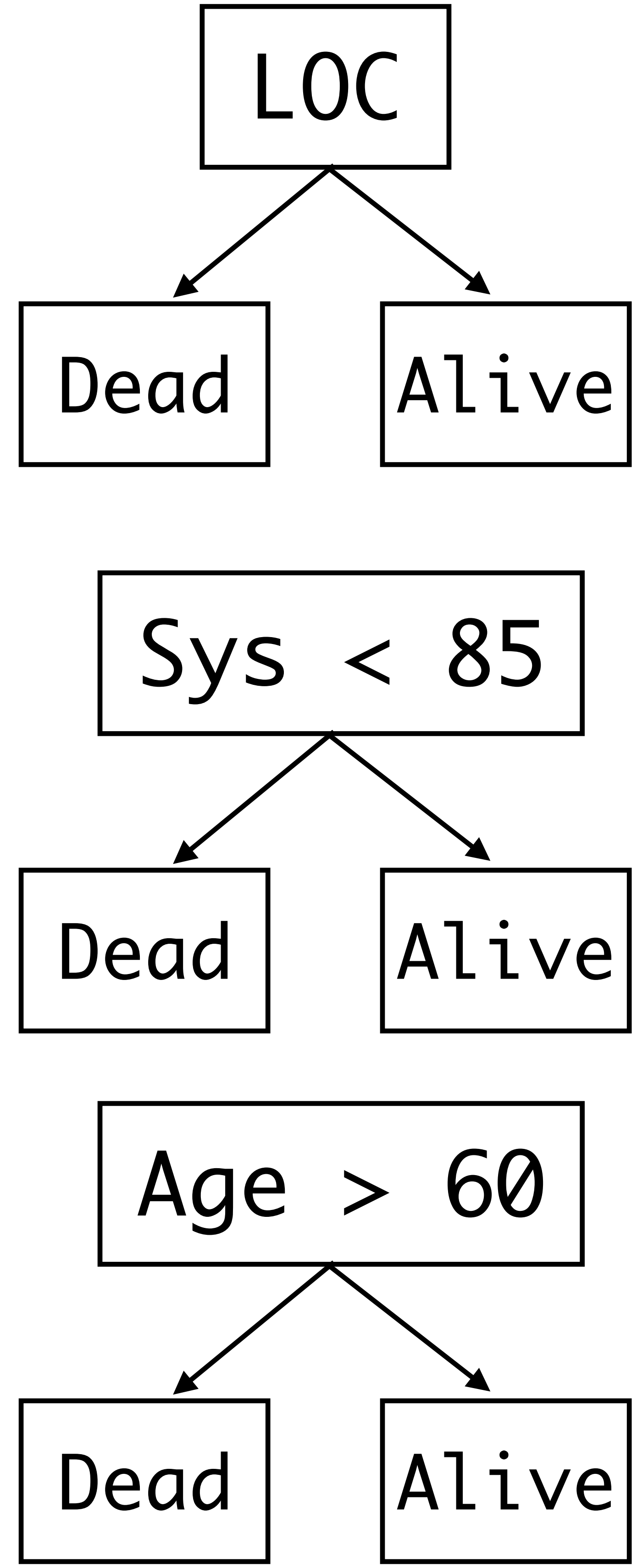
- STA - life status
- LOC - level of consciousness
- Sys - systolic blood pressure
- Age - patient's age

Step1: All subjects get the same weight: $W_1(i) = 1/m = 1/8$

STA	LOC	SYS	AGE	Subject weight
Dead	No	36	27	1/8
Dead	No	48	59	1/8
Dead	Yes	44	77	1/8
Dead	No	62	54	1/8
Alive	Yes	112	87	1/8
Alive	Yes	108	69	1/8
Alive	No	140	63	1/8
Alive	Yes	138	30	1/8

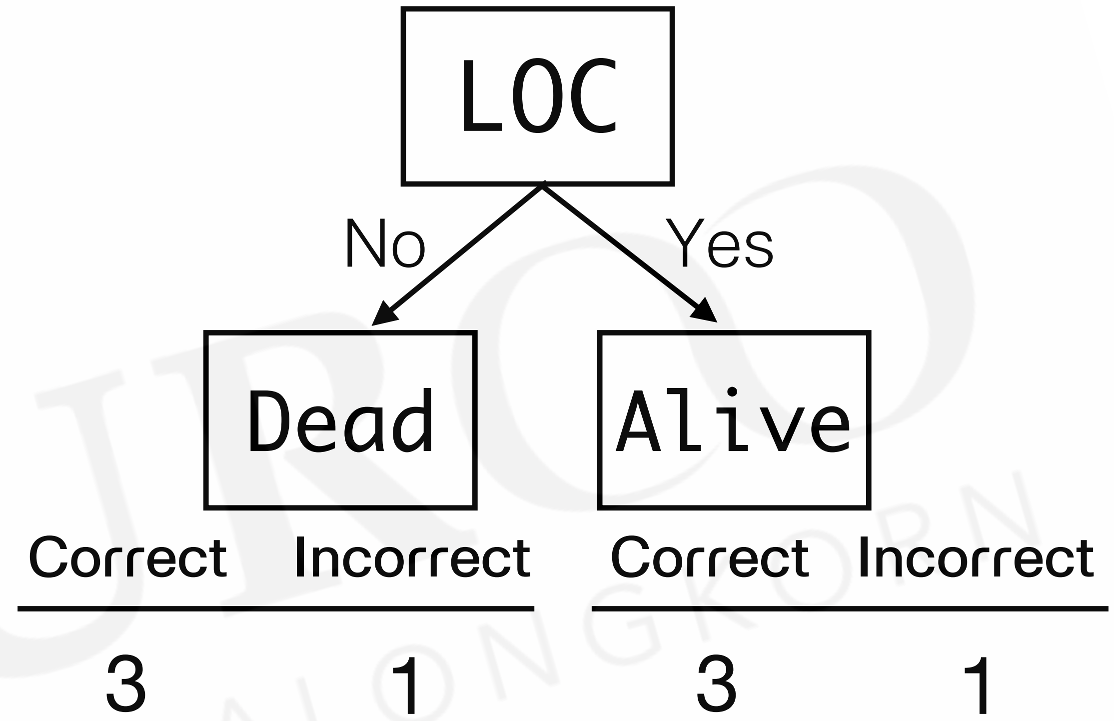
Step2: Build up the first stump

STA	LOC	SYS	AGE	Subject weight
Dead	No	36	27	1/8
Dead	No	48	59	1/8
Dead	Yes	44	77	1/8
Dead	No	62	54	1/8
Alive	Yes	112	87	1/8
Alive	Yes	108	69	1/8
Alive	No	140	63	1/8
Alive	Yes	138	30	1/8



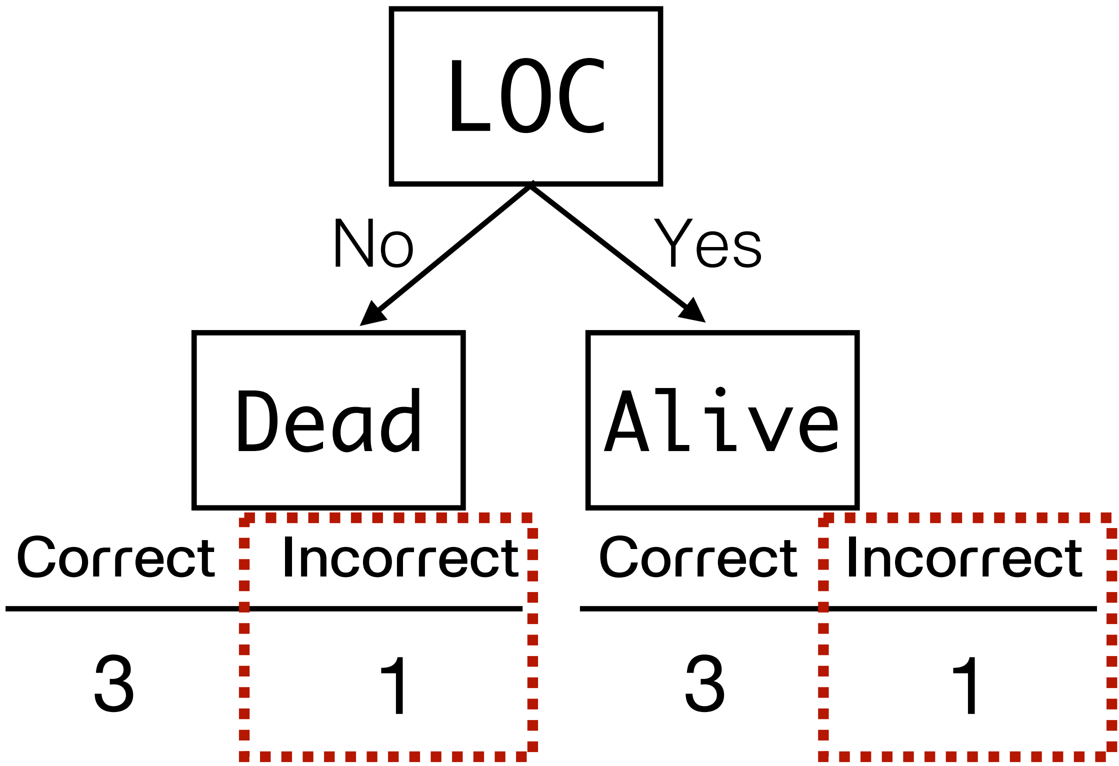
Step2: Build up the first stump

STA	LOC	SYS	AGE	Subject weight
Dead	No	36	27	1/8
Dead	No	48	59	1/8
Dead	Yes	44	77	1/8
Dead	No	62	54	1/8
Alive	Yes	112	87	1/8
Alive	Yes	108	69	1/8
Alive	No	140	63	1/8
Alive	Yes	138	30	1/8



Step3: Evaluate the prediction error of the stump

STA	LOC	SYS	AGE	Subject weight
Dead	No	36	27	1/8
Dead	No	48	59	1/8
Dead	Yes	44	77	1/8
Dead	No	62	54	1/8
Alive	Yes	112	87	1/8
Alive	Yes	108	69	1/8
Alive	No	140	63	1/8
Alive	Yes	138	30	1/8



$$\text{Total Error} = \epsilon_t = \sum_{i=1}^{m} W_t(i) \times \delta_i = (1/8)(6)(0)+(1/8)(2)(1) = 2/8$$

where $\delta_i = 0$ if $M_t(i) = y_i$, and $\delta_i = 1$ if $M_t(i) \neq y_i$

Step4: Calculate **the amount of say** (weight: α_t) of the stump

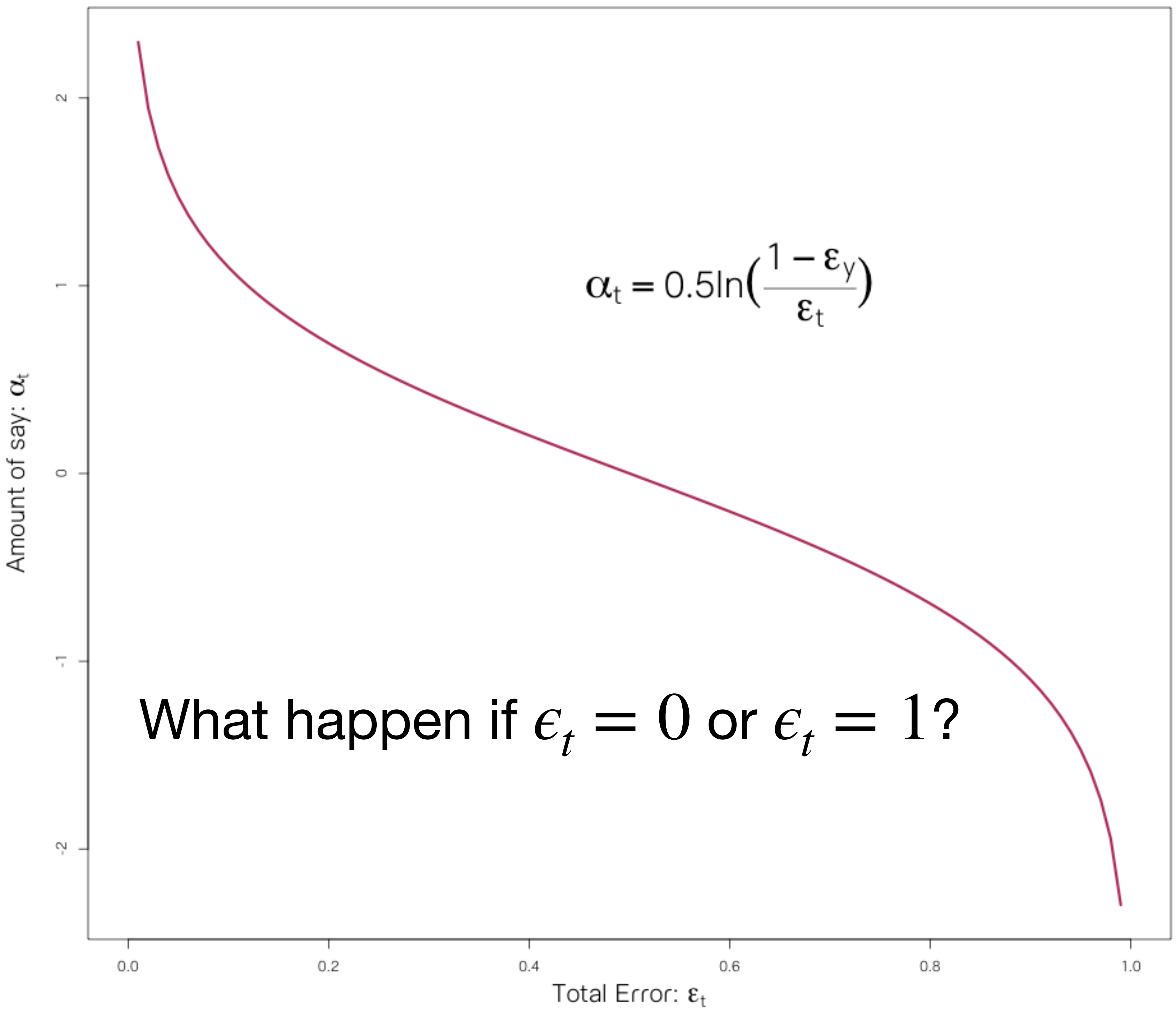
STA	LOC	SYS	AGE	Subject weight
Dead	No	36	27	1/8
Dead	No	48	59	1/8
Dead	Yes	44	77	1/8
Dead	No	62	54	1/8
Alive	Yes	112	87	1/8
Alive	Yes	108	69	1/8
Alive	No	140	63	1/8
Alive	Yes	138	30	1/8

$$\alpha_t = \frac{1}{2} ln(\frac{1 - \epsilon_t}{\epsilon_t})$$

Total Error = $\epsilon_t = \sum_{i=1}^{m} W_t(i) \times \delta_i$ = (1/8)(6)(0)+(1/8)(2)(1) = 2/8

where $\delta_i = 0$ if $M_t(i) = y_i$, and $\delta_i = 1$ if $M_t(i) \neq y_i$

The amount of say: $\alpha_t = \frac{1}{2} ln(\frac{1 - \epsilon_t}{\epsilon_t})$



What happen if $\epsilon_t = 0$ or $\epsilon_t = 1$?

Step4: Calculate **the amount of say** (weight: α_t) of the stump

STA	LOC	SYS	AGE	Subject weight
Dead	No	36	27	1/8
Dead	No	48	59	1/8
Dead	Yes	44	77	1/8
Dead	No	62	54	1/8
Alive	Yes	112	87	1/8
Alive	Yes	108	69	1/8
Alive	No	140	63	1/8
Alive	Yes	138	30	1/8

$$\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$$

$$= \frac{1}{2} ln(\frac{1-2/8}{2/8})$$

$$= 0.549$$

Total Error = $\epsilon_t = \sum_{i=1}^{m} W_t(i) \times \delta_i$ = (1/8)(6)(0)+(1/8)(2)(1) = 2/8

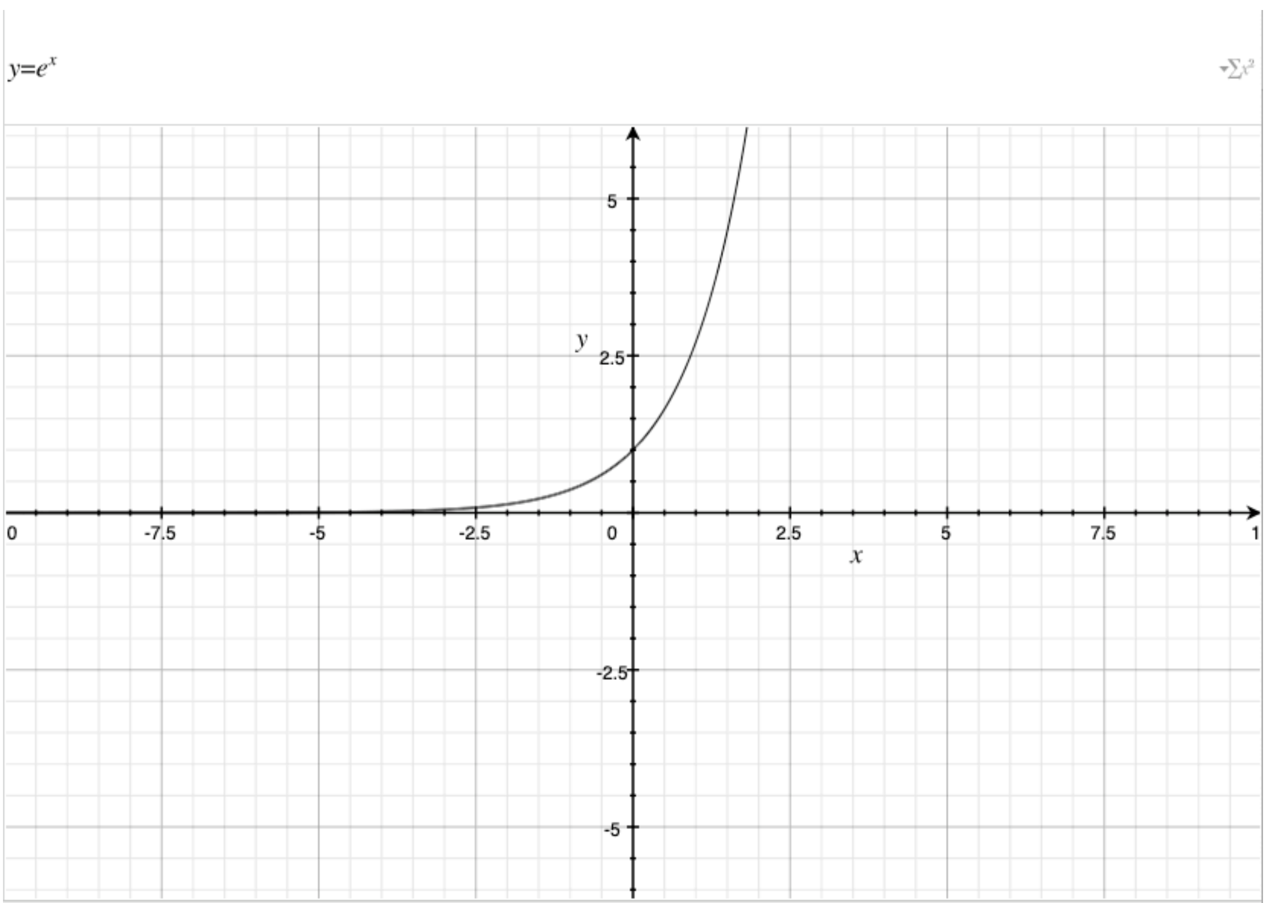
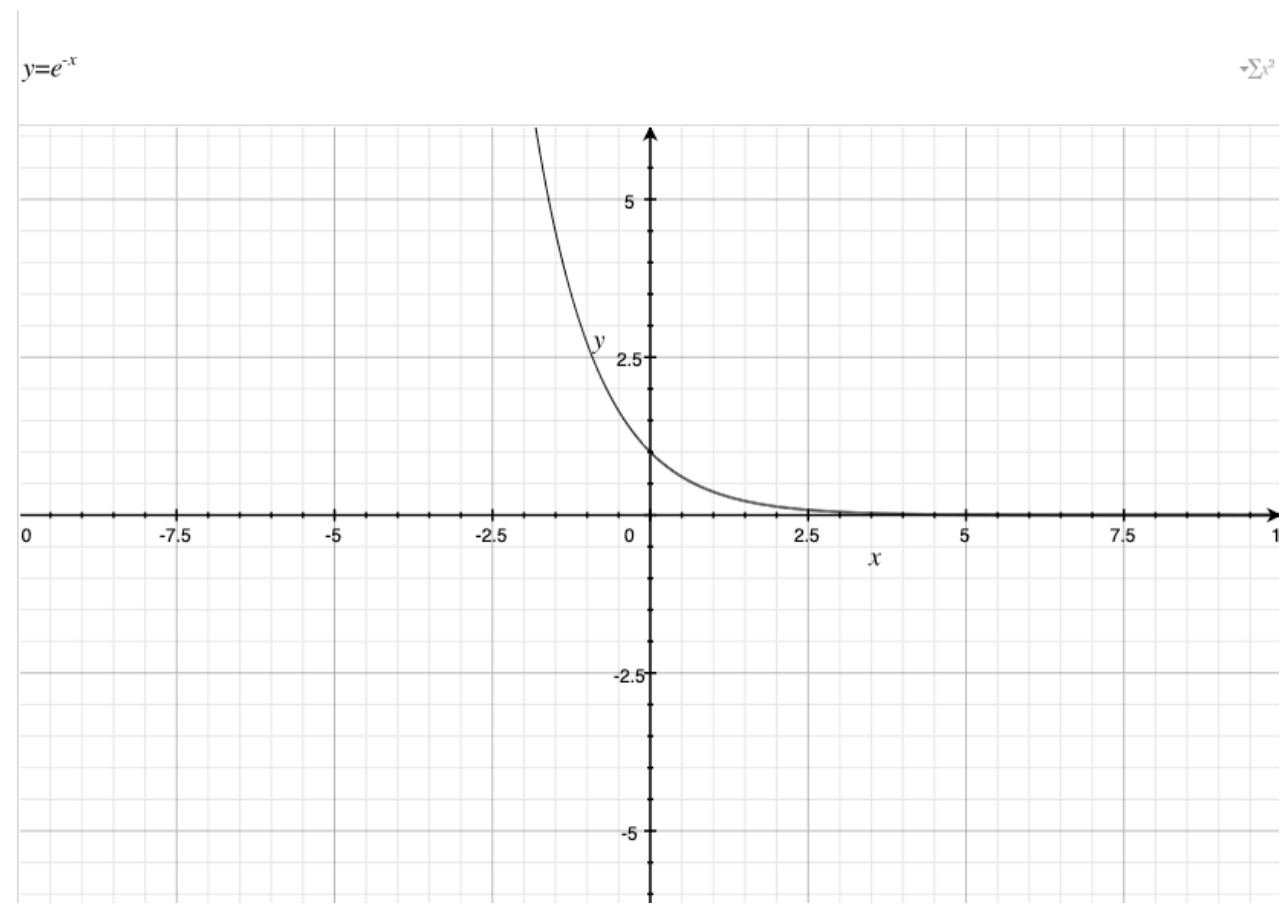
where $\delta_i = 0$ if $M_t(i) = y_i$, and $\delta_i = 1$ if $M_t(i) \neq y_i$

Get back to the step1: Reweight the subjects using previous stump.

STA	LOC	SYS	AGE	Subject weight	New weight*
Dead	No	36	27	1/8	0.072
Dead	No	48	59	1/8	0.072
Dead	Yes	44	77	1/8	0.216
Dead	No	62	54	1/8	0.072
Alive	Yes	112	87	1/8	0.072
Alive	Yes	108	69	1/8	0.072
Alive	No	140	63	1/8	0.216
Alive	Yes	138	30	1/8	0.072

New weight* = $W_{t+1}(i) = W_t(i)$ x e^{α_t} ; if $M_t(i) \neq y_i$

= $W_t(i)$ x $e^{-\alpha_t}$; if $M_t(i) = y_i$


y=e^x
y
x
5
2.5
-2.5
-5
-7.5
0
7.5
1

y=e^-x
y
x
5
2.5
-2.5
-5
-7.5
0
7.5
1

Get back to the step1: Reweight the subjects using previous stump.

STA	LOC	SYS	AGE	Subject weight	New weight	Normalized weight
Dead	No	36	27	1/8	0.072	0.083
Dead	No	48	59	1/8	0.072	0.083
Dead	Yes	44	77	1/8	0.216	0.250
Dead	No	62	54	1/8	0.072	0.083
Alive	Yes	112	87	1/8	0.072	0.083
Alive	Yes	108	69	1/8	0.072	0.083
Alive	No	140	63	1/8	0.216	0.250
Alive	Yes	138	30	1/8	0.072	0.083

New weight (final) = $W_{t+1}(i)$ = $W_t(i)$ x $\frac{e^{\alpha_t}}{Z_t}$; if $M_t(i) \neq y_i$

= $W_t(i)$ x $\frac{e^{-\alpha_t}}{Z_t}$; if $M_t(i) = y_i$

Get back to the step2: Build up the next stump

STA	LOC	SYS	AGE	Adjusted subject weight
Dead	No	36	27	0.083
Dead	No	48	59	0.083
Dead	Yes	44	77	0.250
Dead	No	62	54	0.083
Alive	Yes	112	87	0.083
Alive	Yes	108	69	0.083
Alive	No	140	63	0.250
Alive	Yes	138	30	0.083

Instead of using a weighted Gini or weighted Entropy index, we can make a new collection of data points that contains duplicate copies of the data points correspond to their weights.

STA	LOC	SYS	AGE	Adjusted subject weight
Dead	No	36	27	0.083
Dead	No	48	59	0.083
Dead	Yes	44	77	0.250
Dead	No	62	54	0.083
Alive	Yes	112	87	0.083
Alive	Yes	108	69	0.083
Alive	No	140	63	0.250
Alive	Yes	138	30	0.083

Weighted random sampling with replacement →

STA	LOC	SYS	AGE

STA	LOC	SYS	AGE	Adjusted subject weight
Dead	No	36	27	0.083
Dead	No	48	59	0.083
Dead	Yes	44	77	0.250
Dead	No	62	54	0.083
Alive	Yes	112	87	0.083
Alive	Yes	108	69	0.083
Alive	No	140	63	0.250
Alive	Yes	138	30	0.083

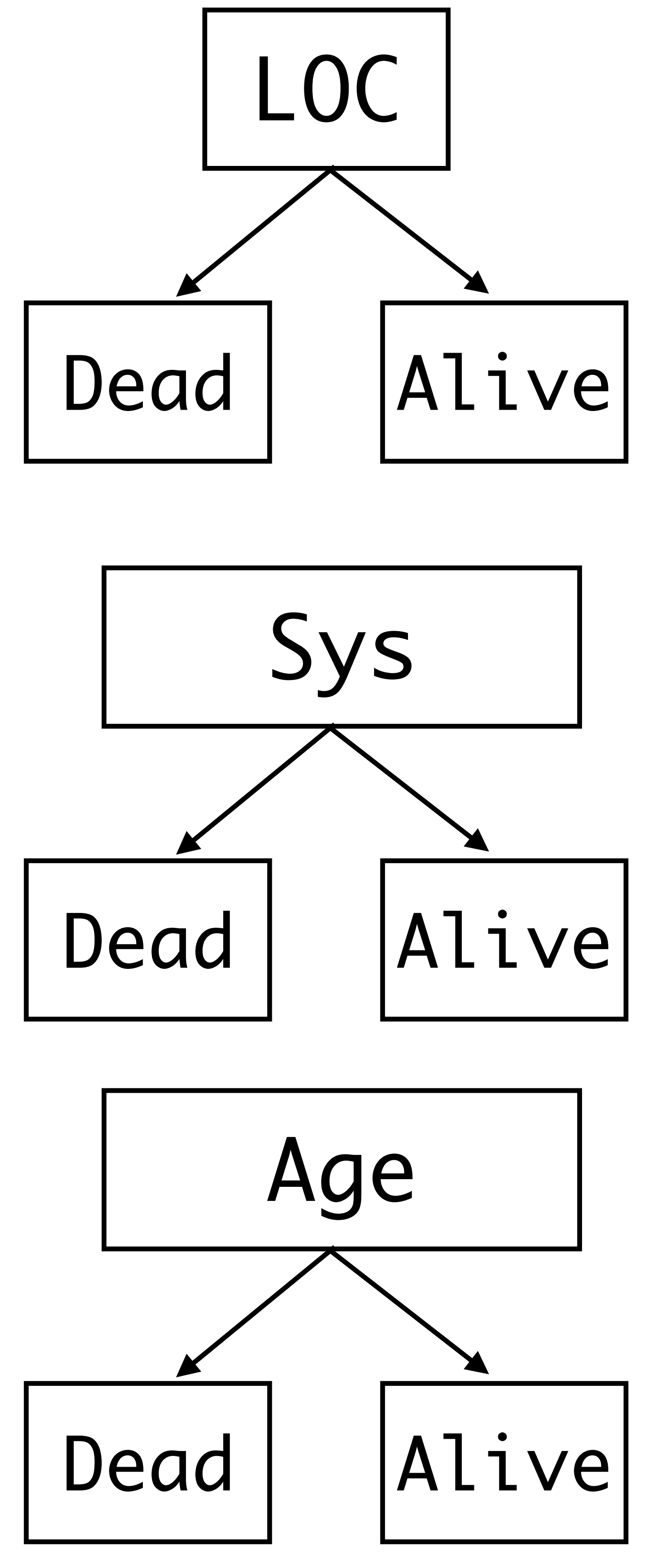
Weighted random sampling with replacement →

STA	LOC	SYS	AGE
Dead	Yes	44	77
Dead	Yes	44	77
Alive	No	140	63
Dead	No	36	27
Dead	No	48	59
Dead	Yes	44	77
Alive	No	140	63
Dead	No	36	27

Get back to the step2: Build up the next stump

Use the new collection of data points as a training data

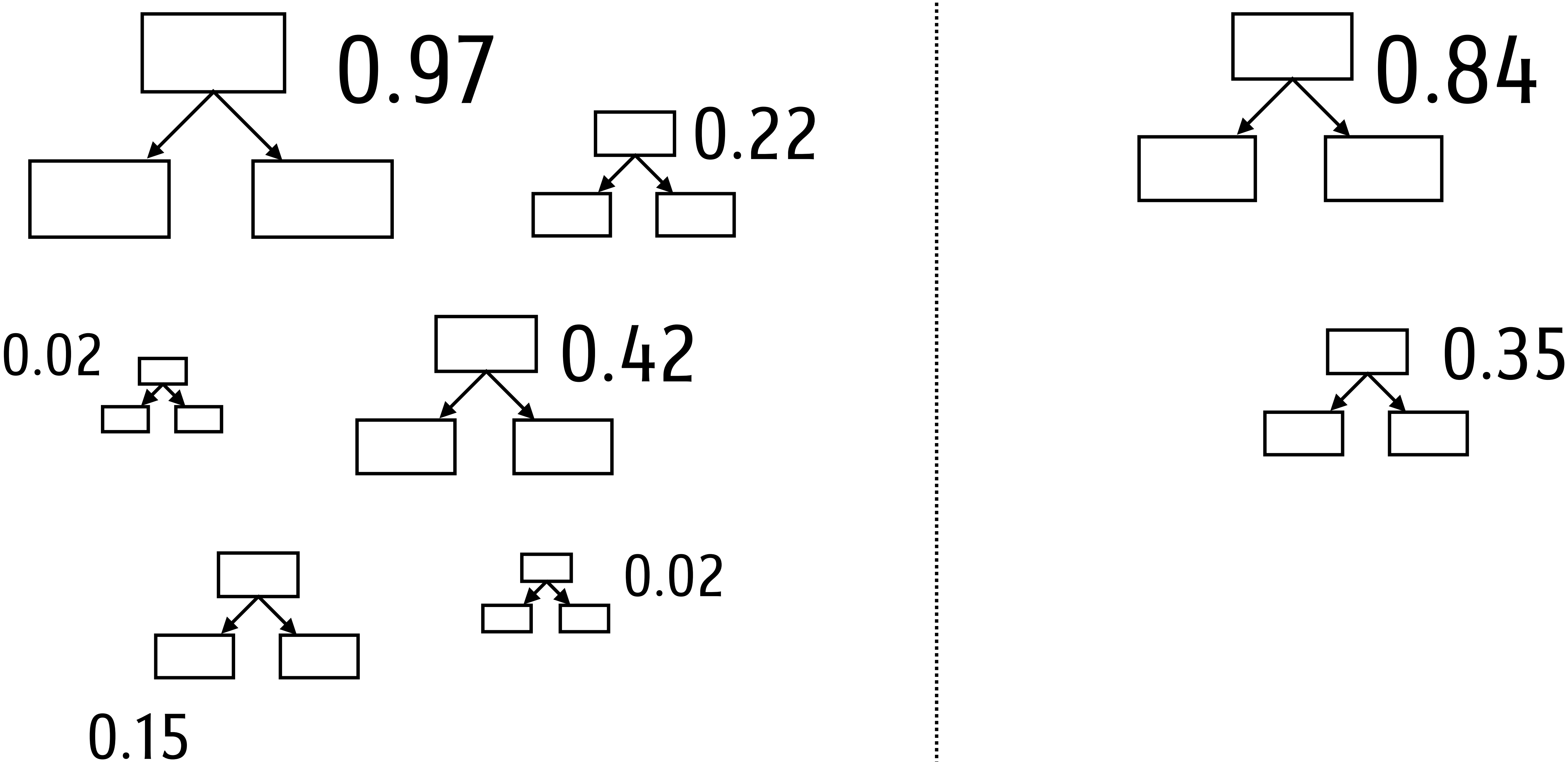
STA	LOC	SYS	AGE	Weight
Dead	Yes	44	77	1/8
Dead	Yes	44	77	1/8
Alive	No	140	63	1/8
Dead	No	36	27	1/8
Dead	No	48	59	1/8
Dead	Yes	44	77	1/8
Alive	No	140	63	1/8
Dead	No	36	27	1/8



repeat …

STA = Alive (-1)

STA = Dead (1)



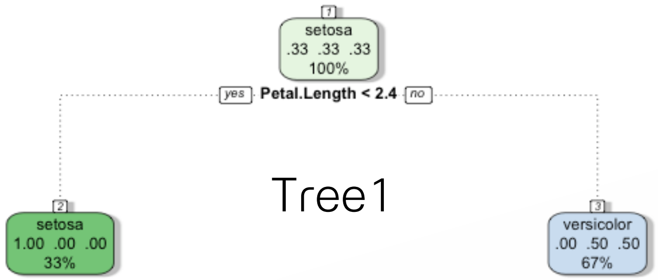
Total amount of say = -1.8

Total amount of say = 1.19

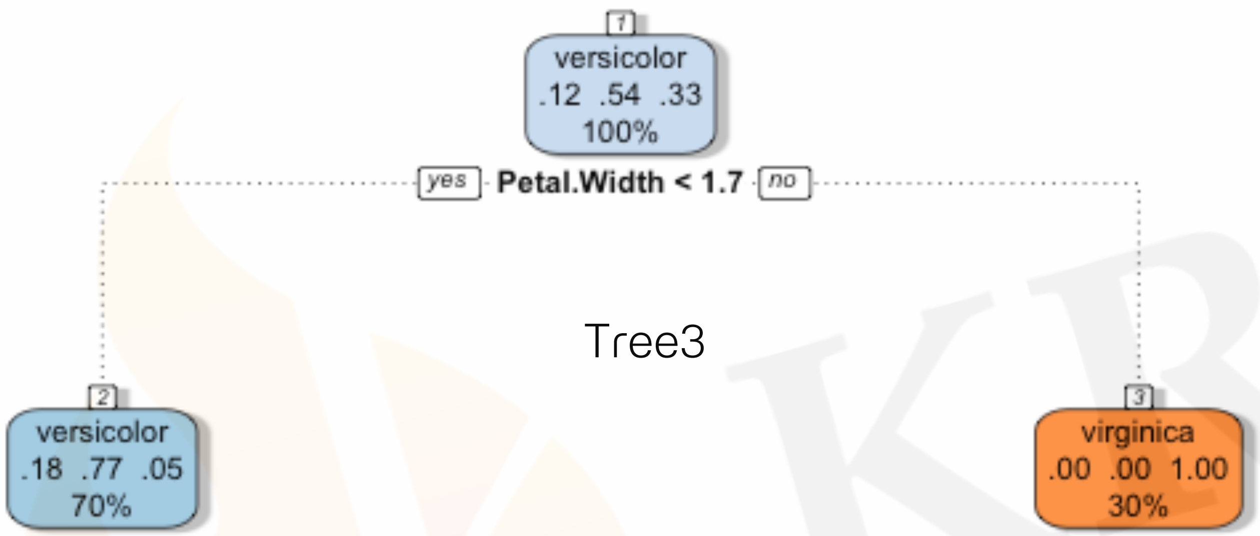
AdaBoosting using R

```
fit.ada
AdaBoost.M1

120 samples
  4 predictor
  3 classes: 'setosa', 'versicolor', 'virginica'

No pre-processing
Resampling: Bootstrapped (5 reps)
Summary of sample sizes: 24, 24, 24, 24, 24
Resampling results across tuning parameters:

  coeflearn  maxdepth  mfinal  logLoss    AUC        prAUC      Accuracy   Kappa     Mean_F1
  Breiman    1          50     0.4750813  0.9697754  0.3008532  0.9375000  0.906250  0.9373596
  Breiman    1         100     0.4767168  0.9803060  0.4699178  0.9375000  0.906250  0.9373596
  Breiman    1         150     0.4756446  0.9783854  0.4825737  0.9354167  0.903125  0.9352625
  Breiman    2          50     0.3659022  0.9818685  0.7150678  0.9354167  0.903125  0.9352625
  Breiman    2         100     0.3745246  0.9817708  0.7789088  0.9354167  0.903125  0.9352625
  Breiman    2         150     0.3806631  0.9822754  0.7898933  0.9354167  0.903125  0.9352625
```

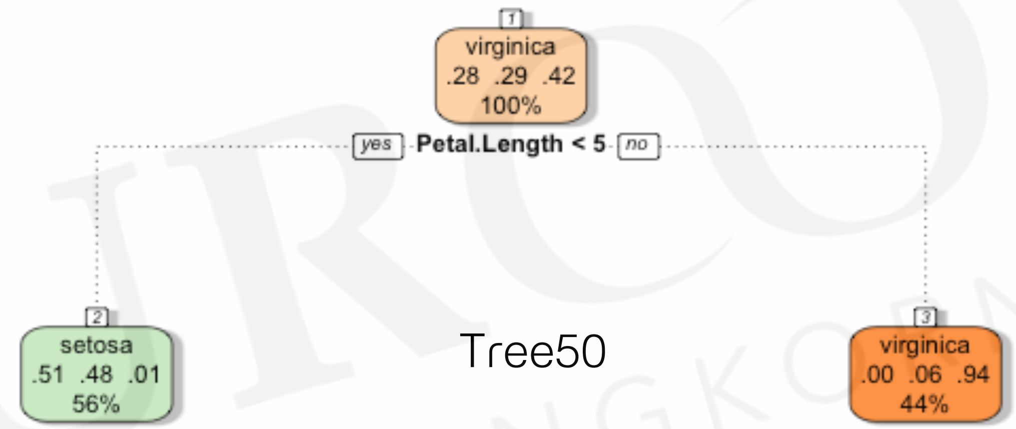
AdaBoosting using R



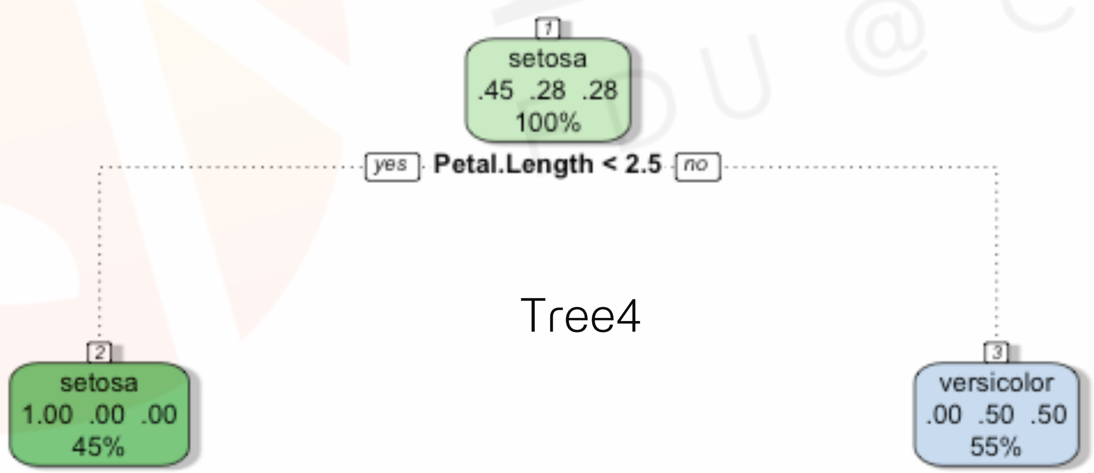
Tree1



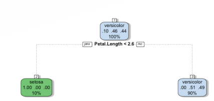
Tree3



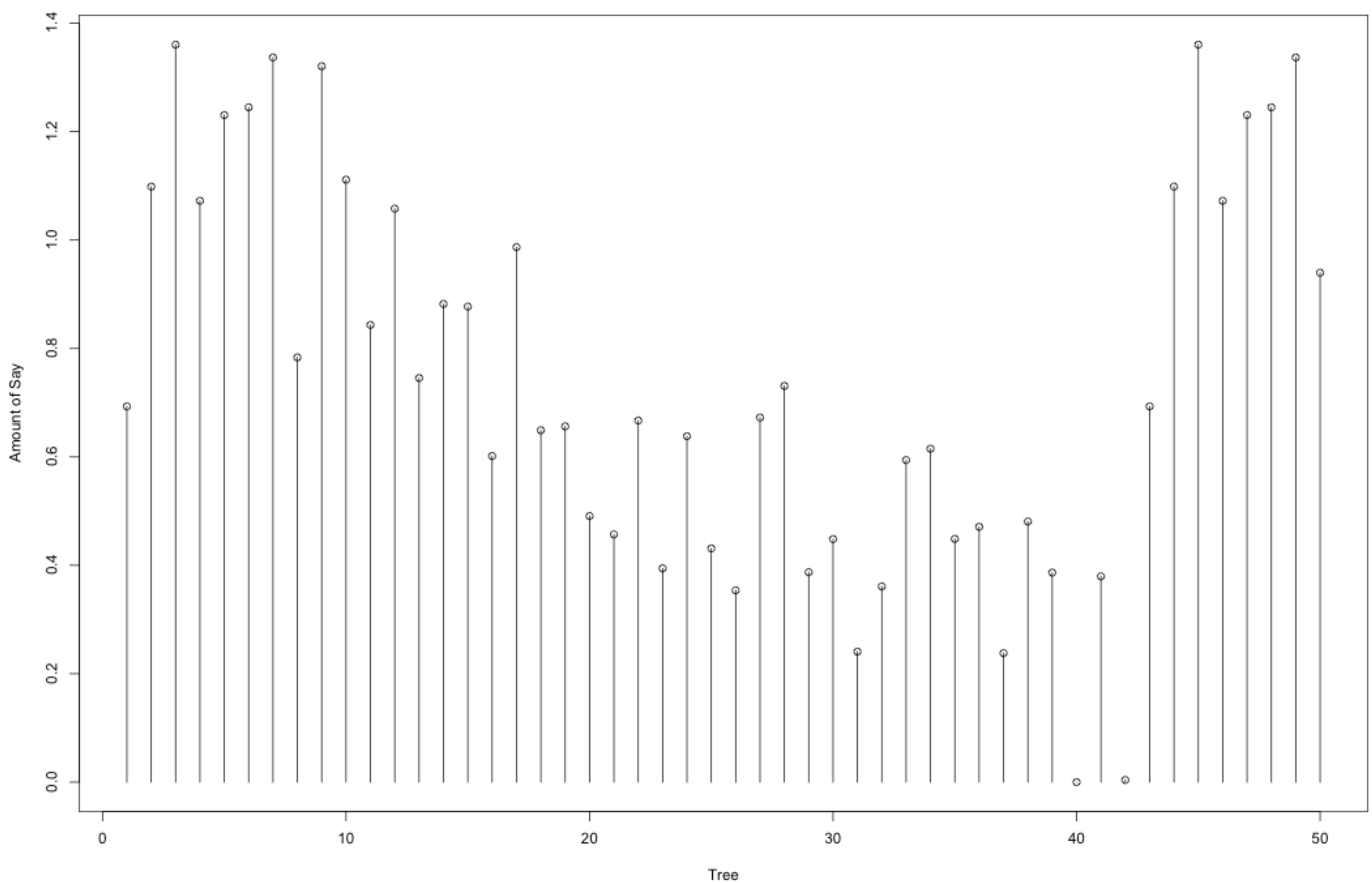
Tree50



Tree4



Tree40


Amount of Say
Tree
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
10
20
30
40
50

Gradient Boosting (BGM)

- Regression
- Classification

Gradient Boosting for Regression

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be differentiable Loss function.

Step 1: fit the null model (model with only constant term) $\Longrightarrow$ $F_0(\mathbf{X}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

Step2: for m in $1 : M$

1. Compute $\underbrace{e_{im}}_{\text{Pseudo residual}} = -\left[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ 2, \ldots,\ n$
2. Fit a regression tree to the e_{im} values and create terminal region R_{jm} for $j = 1, \ldots, J_m$
3. For $j = 1,\ 2, \ldots, J_m$ compute $\gamma_{jm} = \underset{x}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$
4. Update $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \underbrace{\nu}_{\text{Learning rate [0,1]}} \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

Output: $F_M(\mathbf{X})$

Let $(X, \mathbf{y}) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be differentiable Loss function.

Training dataset

SES	Level	Gender	Discipline
-1.0	Primary	Male	27
0.5	Secondary	Female	59
1.0	UnderGrad	Female	77
1.5	Primary	Male	54
1.5	UnderGrad	Male	87
1.0	Secondary	Female	69

$$\sum_{i=1}^{n} \frac{1}{2}(y_i - \hat{y}_i)^2$$

Loss Function

$$L(y_i, F(\mathbf{x_i}) = \hat{y}_i) = \frac{1}{2}(y_i - \hat{y}_i)^2$$

$$\text{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\frac{\partial L}{\partial \hat{y}_i} = -(y_i - \hat{y}_i)$$

where $F(\mathbf{x_i})$ is a function that give us the predicted values.

Step 1: fit the null model (model with only constant term) $\implies F_0(\mathbf{X}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

$L(y_i, \gamma) = \frac{1}{2}(y_i - \gamma)^2$ is loss function; where y_i and γ refer to observed and predicted value respectively.

$\sum_{i=1}^{n} L(y_i, \gamma)$ is the overall Loss function

$\underset{\gamma}{\operatorname{argmin}}$ means we need to find a predicted value that minimized the above sum.

$$\frac{\partial \sum_{i=1}^{n} L(y_i, \gamma)}{\partial \gamma} = -\sum_{i=1}^{n} (y_i - \gamma) = 0$$

$\implies \gamma = \frac{\sum_{i=1}^{n} y_i}{n}$ Hence, given the loss function $L(y_i, \gamma) = \frac{1}{2}(y_i - \gamma)^2$ the value of γ that minimizes the above sum is the average of the observed outcome. $\implies F_0(\mathbf{X}) = \gamma = \frac{\sum_{i=1}^{n} y_i}{n}$

Step 1: fit the null model (model with only constant term) $\Longrightarrow$ $F_0(\mathbf{X}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

SES	Level	Gender	Discipline
-1.0	Primary	Male	27
0.5	Secondary	Female	59
1.0	UnderGrad	Female	77
1.5	Primary	Male	54
1.5	UnderGrad	Male	87
1.0	Secondary	Female	69

Hence, $F_0(\mathbf{X}) = \dfrac{27 + 59 + \ldots + 69}{6} = 62.17$

That means the null model (or initial predicted value) is a tree with just one leaf.

62.17

- Gradient boost starts by making a single leaf, instead of a tree or stump.
- This leaf represent an prior or initial guess for the weight of all subjects

Step2: for m in $1:M$ ← We will produce M trees. (i.e. M=100)

1. Compute $e_{im} = -\left[\dfrac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1, 2, ..., n$

2. Fit a regression tree to the e_{im} values and create terminal region R_{jm} for $j = 1, ..., J_m$

3. For $j = 1, 2, ..., J_m$ compute $\gamma_{jm} = \underset{x}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

4. Update $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_j m)$

Step2: for m in $1 : M$

Let $m = 1$

1. Compute $e_{im} = -[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ 2, ..., \ n$

Since the loss function is $L(y_i, F(\mathbf{x_i}) = \hat{y}_i) = \frac{1}{2}(y_i - \hat{y}_i)^2$,

$$e_{i(m=1)} = -[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}]_{F(x)=F_0(x)}$$

$$= -[\frac{\partial L(y_i, F_0(\mathbf{x_i}))}{\partial F_0(\mathbf{x_i})}]$$; where $F_0(\mathbf{x}_i) = \mu = 62.17$

$$= (y_i - 62.17)$$

hence, e_{im} is a residual of i-th sample, and m-th trees

$$e_{1,1} = 27 - 62.17 = -35.17$$

$$\cdots \qquad \cdots$$

$$e_{6,1} = 69 - 62.17 = 6.83$$

Step2: for m in $1 : M$

SES	Level	Gender	Discipline	(Pseudo) Residual=actual-predicted ($e_{i,1}$)
-1.0	Primary	Male	27	-35.17
0.5	Secondary	Female	59	-3.17
1.0	UnderGrad	Female	77	14.83
1.5	Primary	Male	54	-8.17
1.5	UnderGrad	Male	87	24.83
1.0	Secondary	Female	69	6.83

Average value of Discipline score **62.17**

Step2: for m in $1 : M$

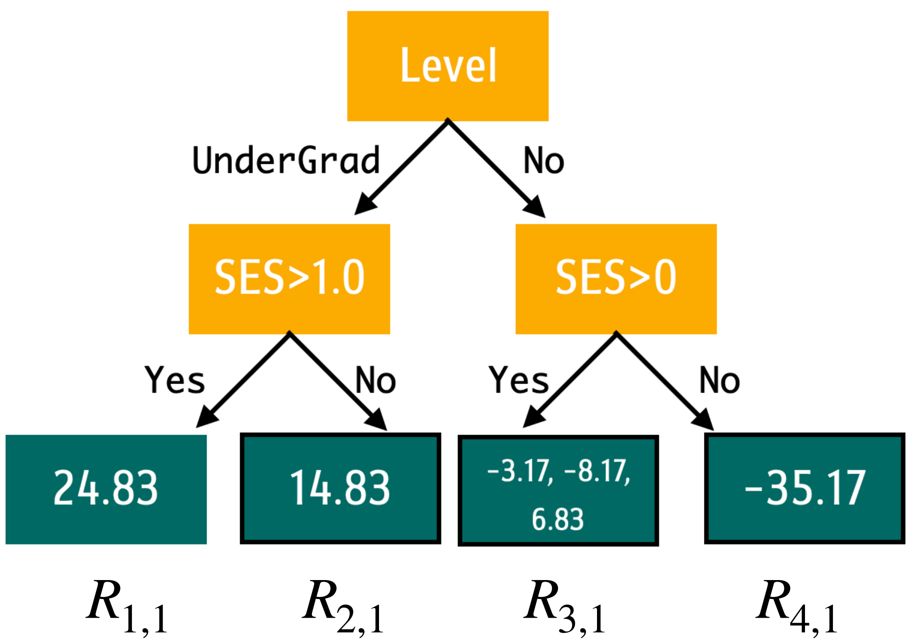
Let $m = 1$

1. Compute $e_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1, 2, \ldots, n$
2. Fit a regression tree to the e_{im} values and create terminal region R_{jm} for $j = 1, \ldots, J_m$

- We will build a regression tree to predict the residual instead of outcome.

SES	Level	Gender	Discipline	$e_{i,1}$
-1.0	Primary	Male	27	-35.17
0.5	Secondary	Female	59	-3.17
1.0	UnderGrad	Female	77	14.83
1.5	Primary	Male	54	-8.17
1.5	UnderGrad	Male	87	24.83
1.0	Secondary	Female	69	6.83

Predict



The leaves are the terminal regions R_{jm} for $j = 1,2,\ldots,J_m = 4$

Step2: for m in $1 : M$

Let $m = 1$

1. Compute $e_{im} = -[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ 2,\ldots,\ n$
2. Fit a regression tree to the e_{im} values and create terminal region R_{jm} for $j = 1,\ldots,J_m$
3. $\boxed{\text{For } j = 1,\ 2,\ldots,J_m}$ compute $\gamma_{jm} = \underset{x}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

For the each leaf in the new regression tree, we calculate the new estimate value (γ_{jm}) of the output value for each leaf

SES	Level	Gender	Discipline	New prediction
-1.0	Primary	Male	27	
0.5	Secondary	Female	59	
1.0	UnderGrad	Female	77	
1.5	Primary	Male	54	
1.5	UnderGrad	Male	87	62.17+24.83 = 87
1.0	Secondary	Female	69	

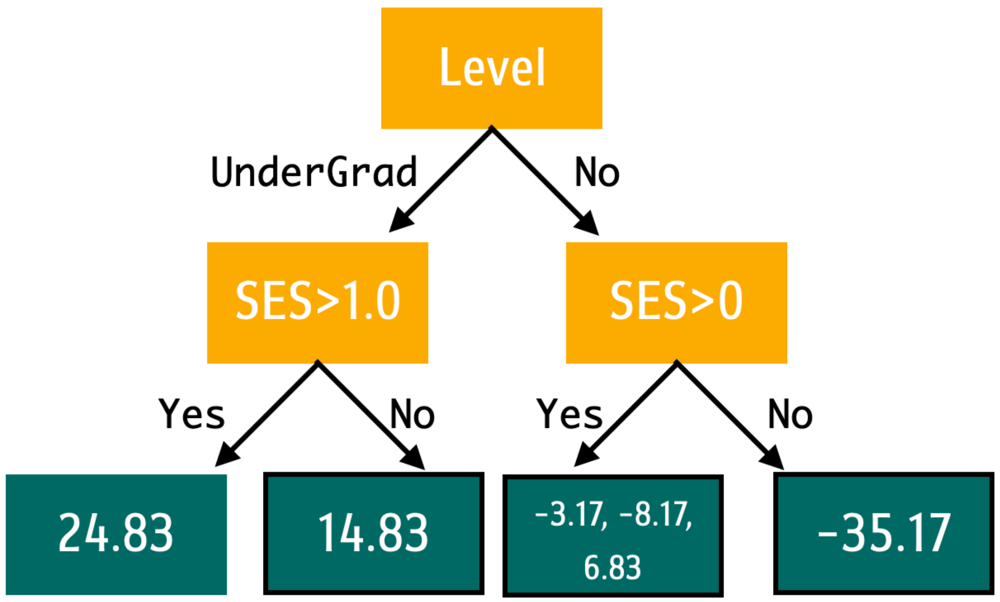
$$\gamma_{1,1} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$$

$$= \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} \frac{1}{2}(y_i - (F_{m-1}(\mathbf{x_i}) + \gamma))^2$$

$$= \underset{\gamma}{\operatorname{argmin}} \frac{1}{2}(87 - (62.17 + \gamma))^2$$

$$= \underset{\gamma}{\operatorname{argmin}} \frac{1}{2}(24.83 - \gamma))^2 \implies \frac{\partial \frac{1}{2}(24.83 - \gamma))^2}{\partial \gamma} = 0$$

$$\implies \gamma = 24.83$$



$R_{1,1}$ $R_{2,1}$ $R_{3,1}$ $R_{4,1}$

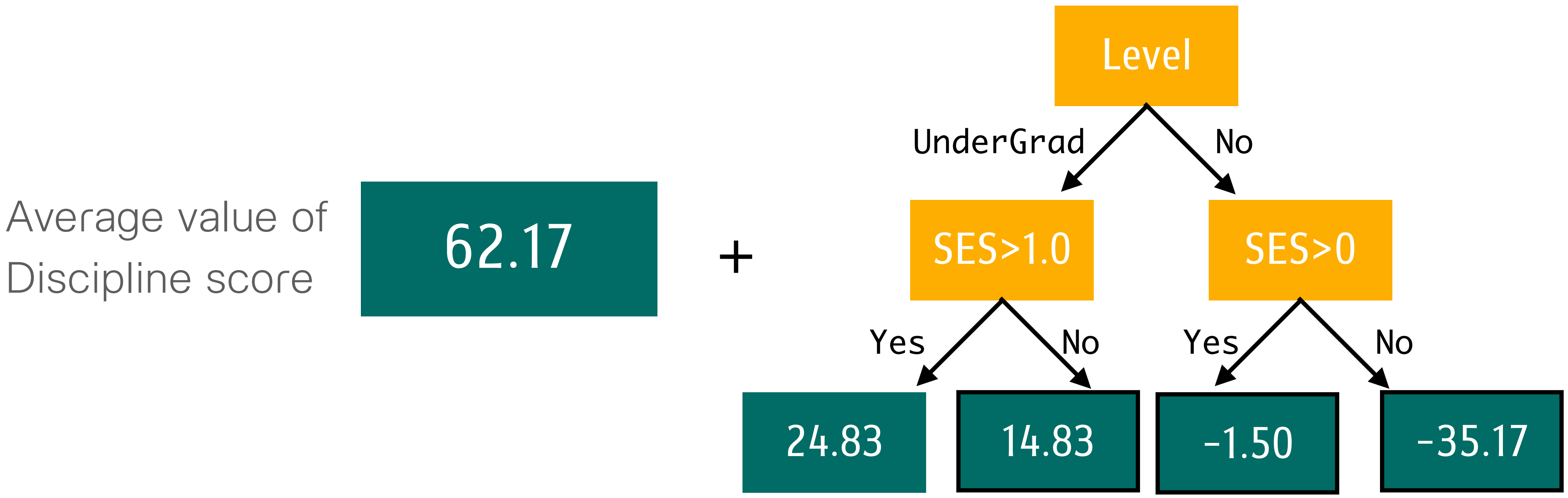
$$\gamma_{4,1} = 14.83 - 35.17$$

$$\gamma_{1,1} = 24.83$$

$$\gamma_{2,1} = 14.83$$

$$\gamma_{3,1} = \underset{\gamma}{\operatorname{argmin}}\{\frac{1}{2}(59 - (62.17 + \gamma))^2 + \frac{1}{2}(54 - (62.17 + \gamma))^2 + \frac{1}{2}(69 - (62.17 + \gamma))^2\}$$

$$= \frac{-3.17 - 8.17 + 6.83}{3} = -1.50$$



SES	Level	Gender	Discipline	New prediction
-1.0	Primary	Male	27	62.17-35.17 = 27
0.5	Secondary	Female	59	62.17-1.50 = 60.67
1.0	UnderGrad	Female	77	62.17+14.83 = 77
1.5	Primary	Male	54	62.17-1.50 = 60.67
1.5	UnderGrad	Male	87	62.17+24.83 = 87
1.0	Secondary	Female	69	62.17-1.50 = 60.67

Overfit

Step2: for m in $1 : M$

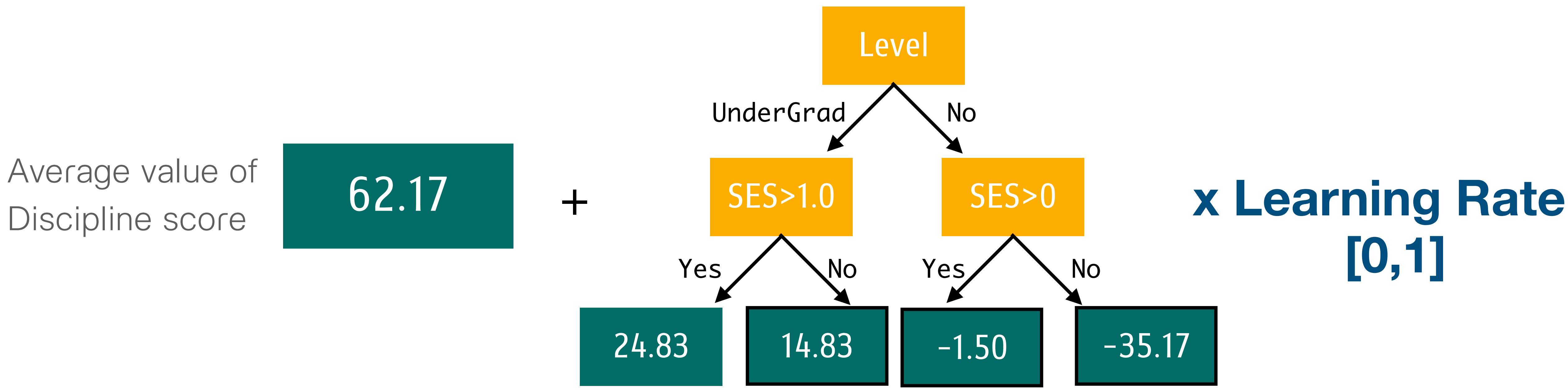
Let $m = 1$

1. Compute $e_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ 2, \ldots,\ n$
2. Fit a regression tree to the e_{im} values and create terminal region R_{jm} for $j = 1, \ldots, J_m$
3. For $j = 1,\ 2, \ldots, J_m$ compute $\gamma_{jm} = \underset{x}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$
4. Update $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_j m)$

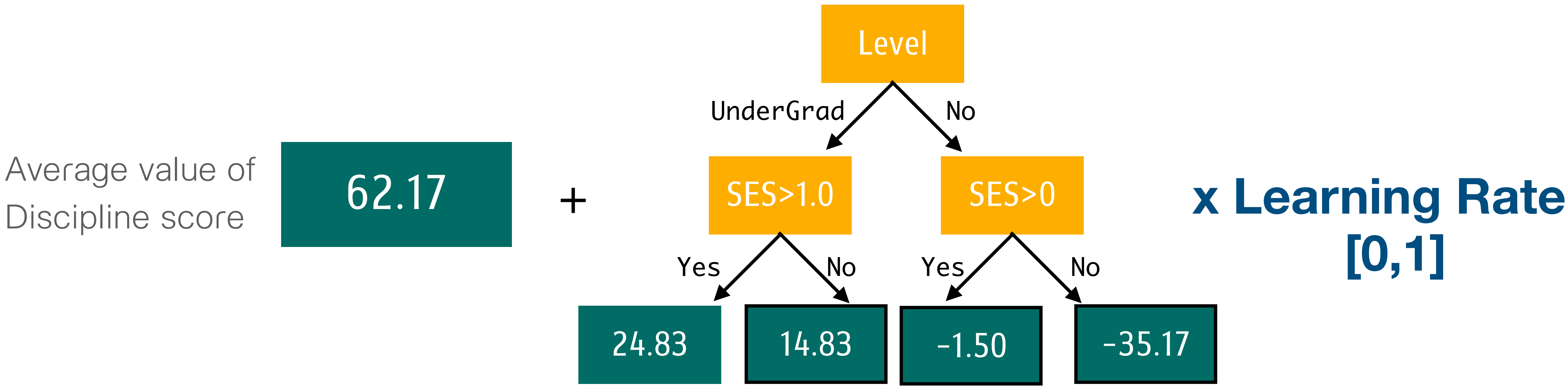
This is the new prediction

$$F_1(\mathbf{x}) = 62.17 + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_j m)$$

- $\nu \in (0,1)$ is the learning rate
- Learning rate is an effect of the regression tree on the final prediction



SES	Level	Gender	Discipline	New prediction
-1.0	Primary	Male	27	62.17-35.17(0.1) = 58.65
0.5	Secondary	Female	59	62.17-1.50(0.1) = 62.02
1.0	UnderGrad	Female	77	62.17+14.83(0.1) = 63.65
1.5	Primary	Male	54	62.17-1.50(0.1) = 62.02
1.5	UnderGrad	Male	87	62.17+24.83(0.1) = 64.65
1.0	Secondary	Female	69	62.17-1.50(0.1) = 62.02



- We will see that with the learning rate = 0.1, the prediction values are not as good as the prediction values with learning rate = 1.0. However there are little better than initial prediction = 62.17.

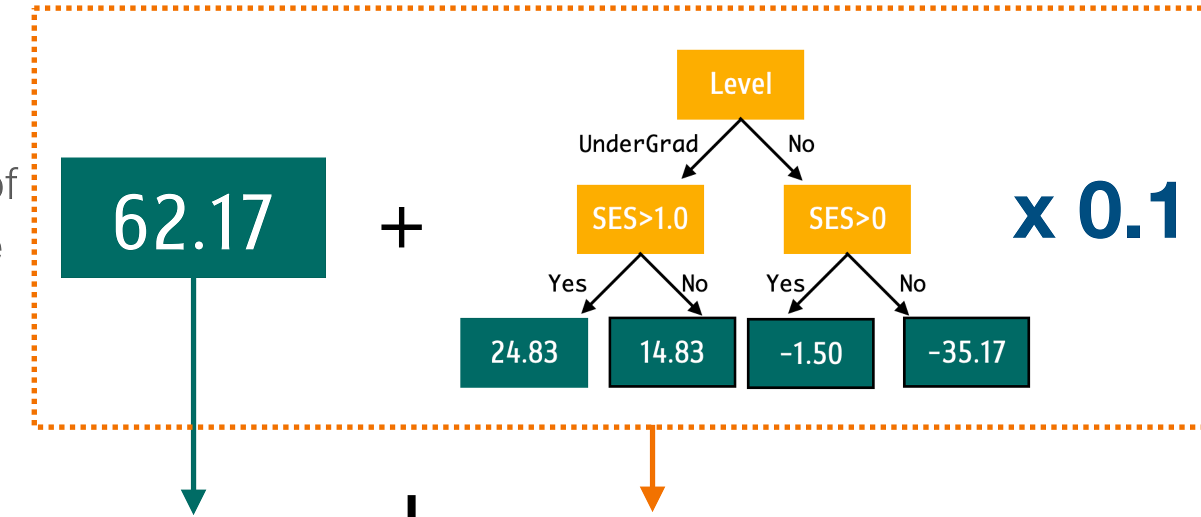
- Hence, by scaling the tree with a learning rate results in a small step in the right direction.

- **Empirical study shows that taking lots of small steps in the right direction results in better predictions with a testing dataset (lower variance).**

https://statweb.stanford.edu/~jhf/ftp/trebst.pdf

Repeat step2: Build another tree based on the error of previous tree.

Average value of Discipline score

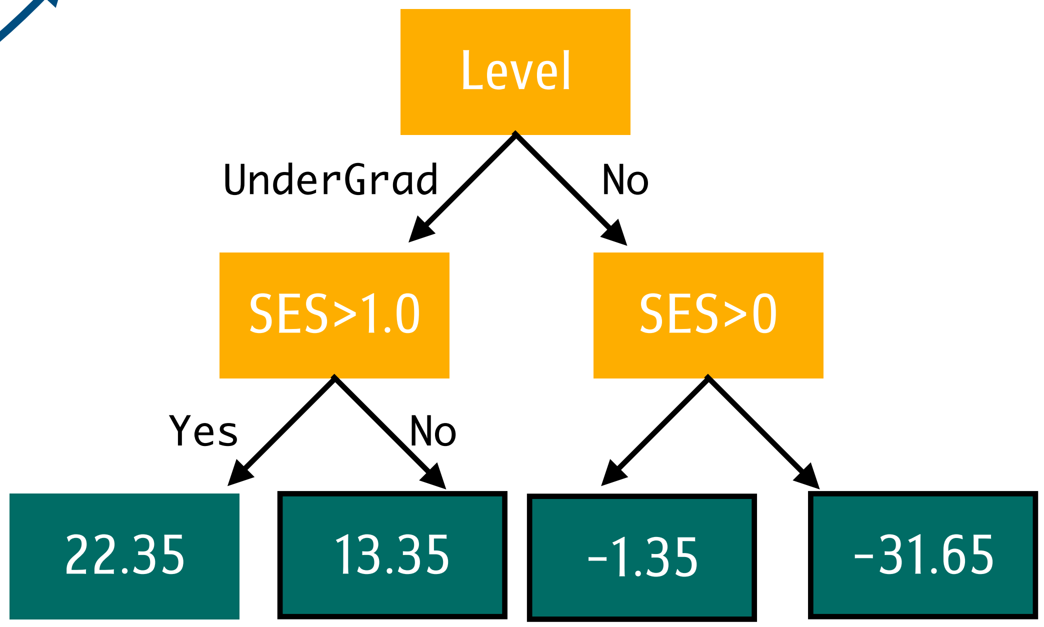


SES	Level	Gender	Discipline	Residual1	Residual2
-1.0	Primary	Male	27	-35.17	-31.65
0.5	Secondary	Female	59	-3.17	-3.02
1.0	UnderGrad	Female	77	14.83	13.35
1.5	Primary	Male	54	-8.17	-8.02
1.5	UnderGrad	Male	87	24.83	22.35
1.0	Secondary	Female	69	6.83	6.98

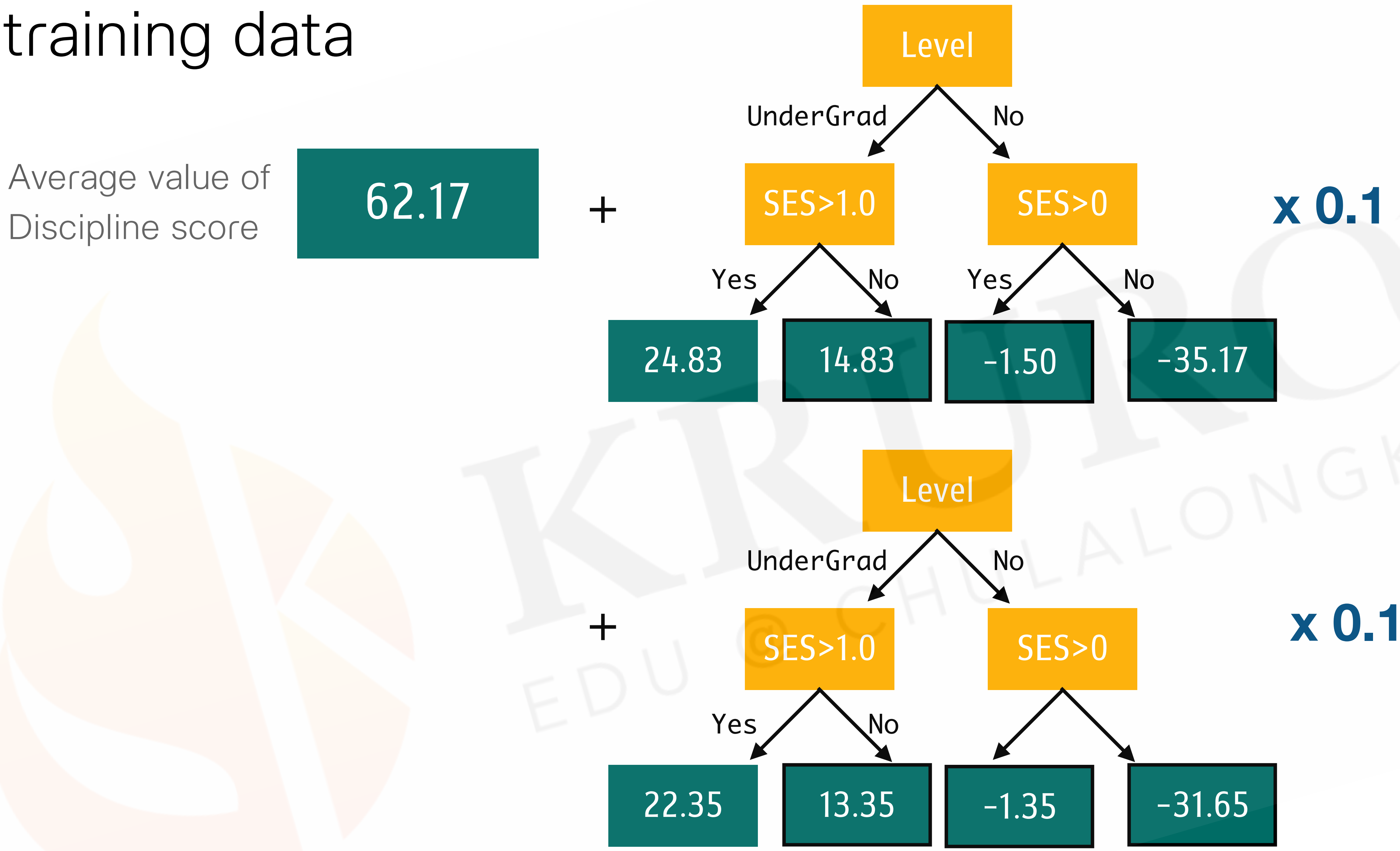
Repeat step2: Build another tree based on the error of previous tree.

SES	Level	Gender	Discipline	Residual2
-1.0	Primary	Male	27	-31.65
0.5	Secondary	Female	59	-3.02
1.0	UnderGrad	Female	77	13.35
1.5	Primary	Male	54	-8.02
1.5	UnderGrad	Male	87	22.35
1.0	Secondary	Female	69	6.98

Predict



Combine the original leaf with the new tree to make a new prediction of an individual discipline from the training data



= Prediction value

Combine the original leaf with the new tree to make a new prediction of an individual discipline from the training data

SES	Level	Gender	Discipline	Prediction1	Prediction2
-1.0	Primary	Male	27	58.65	62.17+0.1*(-35.17)+(0.1)*(-31.65) = 55.49
0.5	Secondary	Female	59	62.02	
1.0	UnderGrad	Female	77	63.65	
1.5	Primary	Male	54	62.02	
1.5	UnderGrad	Male	87	64.65	
1.0	Secondary	Female	69	62.02	

repeat …

GBM Hyperparameters

- n.trees: number of trees
- bag.fraction: proportion of observations to be sampled in each tree
- n.minobsinnode: minimum number of observations in the trees terminal nodes
- interaction.depth: maximum nodes per tree
- shrinkage: learning rate

https://cran.r-project.org/web/packages/gbm/gbm.pdf

Gradient Boosting for Regression

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be differentiable Loss function.

Step 1: fit the null model (model with only constant term) $\implies F_0(\mathbf{X}) = \underset{\gamma}{\text{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

Step2: for m in $1 : M$

1. Compute $\underset{\text{Pseudo residual}}{e_{im}} = -\left[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ 2, \ldots,\ n$
2. Fit a regression tree to the e_{im} values and create terminal region R_{jm} for $j = 1, \ldots, J_m$
3. For $j = 1,\ 2, \ldots, J_m$ compute $\gamma_{jm} = \underset{x}{\text{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$
4. Update $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

Learning rate [0,1]

Output: $F_M(\mathbf{X})$

Gradient Boosting for Classification

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be differentiable Loss function.

Step 1: fit the null model (model with only constant term) $\implies F_0(\mathbf{X}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

Step2: for m in $1 : M$

1. Compute $\underset{\text{Pseudo residual}}{e_{im}} = -[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ 2, \ldots,\ n$
2. Fit a regression tree to the e_{im} values and create terminal region R_{jm} for $j = 1, \ldots, J_m$
3. For $j = 1,\ 2, \ldots, J_m$ compute $\gamma_{jm} = \underset{x}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$
4. Update $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \underset{\text{Learning rate [0,1]}}{\nu} \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

Output: $F_M(\mathbf{X})$

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be differentiable Loss function.

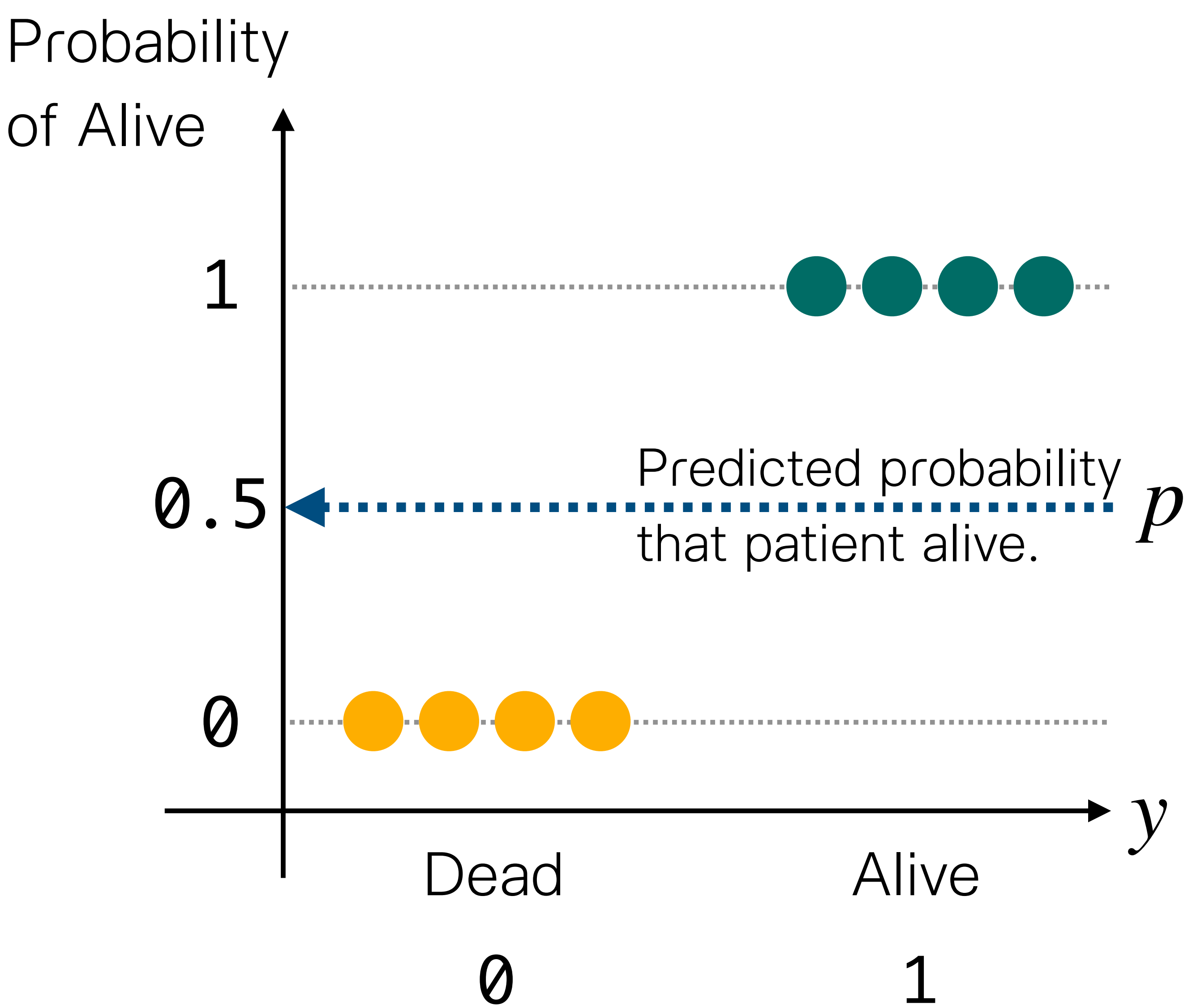
Training Dataset

LOC	SYS	AGE	STA
No	36	27	Dead
No	48	59	Dead
Yes	44	77	Dead
No	62	54	Dead
Yes	112	87	Alive
Yes	108	69	Alive
No	140	63	Alive
Yes	138	30	Alive

- STA - life status
- LOC - level of consciousness
- Sys - systolic blood pressure
- Age - patient’s age

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be differentiable Loss function.

How to define **the Loss function?**



Let y_i be patient status (STA), and by using probability theory, we have

$$y_i \sim Ber(p)$$

where p is the predicted probability, and y_i is the observed value of STA (0 = dead or 1 = alive)

So, the observation model is

$$p(y_i) = p^{y_i}(1-p)^{1-y_i} \quad ; y_i = 0,\ 1$$

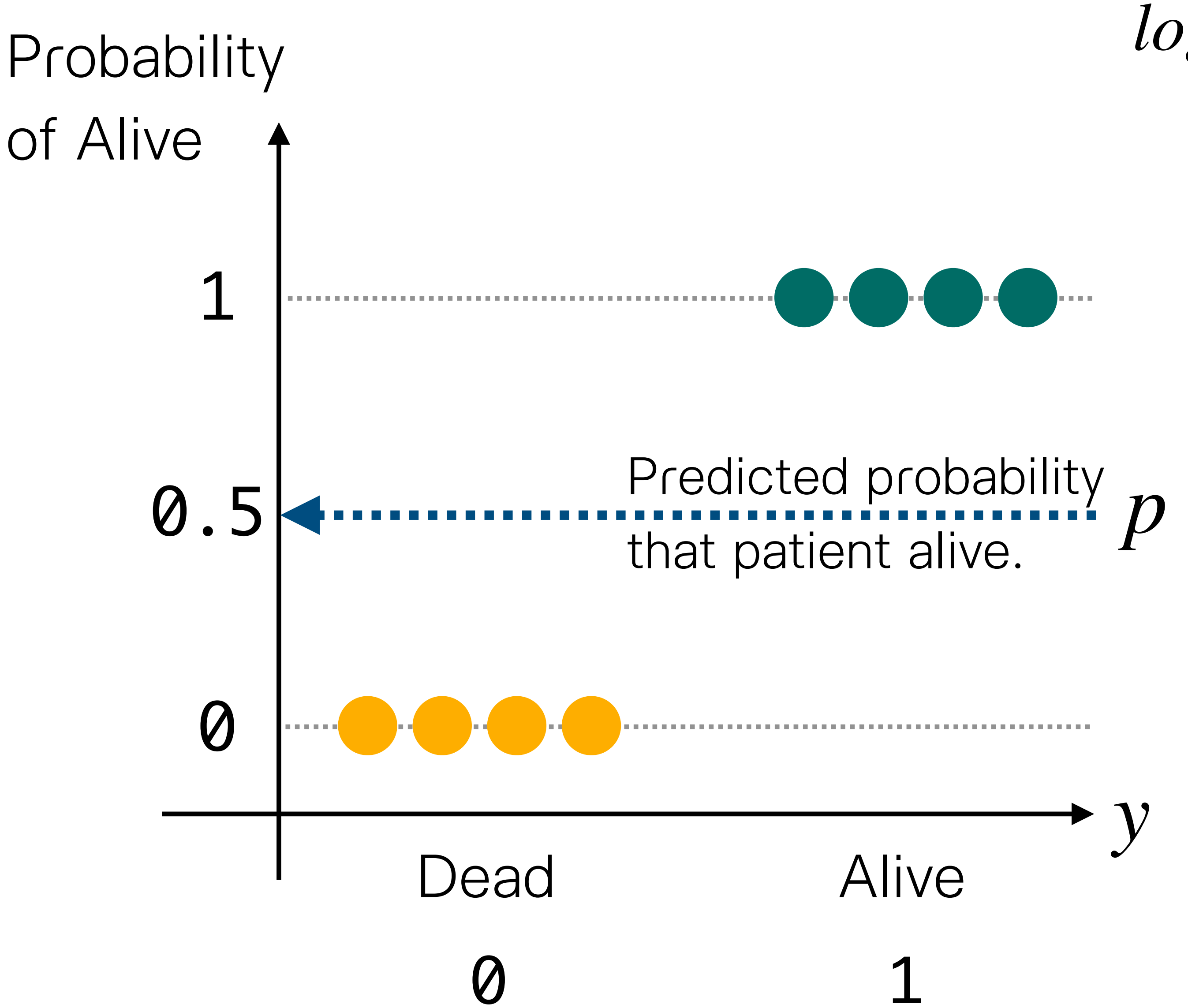
Hence **the likelihood function** is

$$p(\mathbf{y} \,|\, p) = \Pi_{i=1}^{n} p^{y_i}(1-p)^{1-y_i}$$

Consistency between y and p

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be diferentiable Loss function.

How to define **the Loss function?**



$$p(\mathbf{y}\,|\,p) = \Pi_{i=1}^{n} p^{y_i}(1-p)^{1-y_i}$$

$$log[p(\mathbf{y}\,|\,p)] = log[\Pi_{i=1}^{n} p^{y_i}(1-p)^{1-y_i}]$$

$$log(p^{y_i}(1-p)^{1-y_i})$$

$$log(p^{y_i}) + log((1-p)^{1-y_i})$$

$$y_i log(p) + (1-y_i) log((1-p))$$

- The Log-likelihood of the data given the prediction is

$$\sum_{i=1}^{n} [y_i log(p) + (1-y_i) log(1-p)]$$

per observation log-likelihood

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be diferentiable Loss function.

log-likelihood function

$$\sum_{i=1}^{n} \{y_i log(p) + (1 - y_i) log(1 - p)\}$$

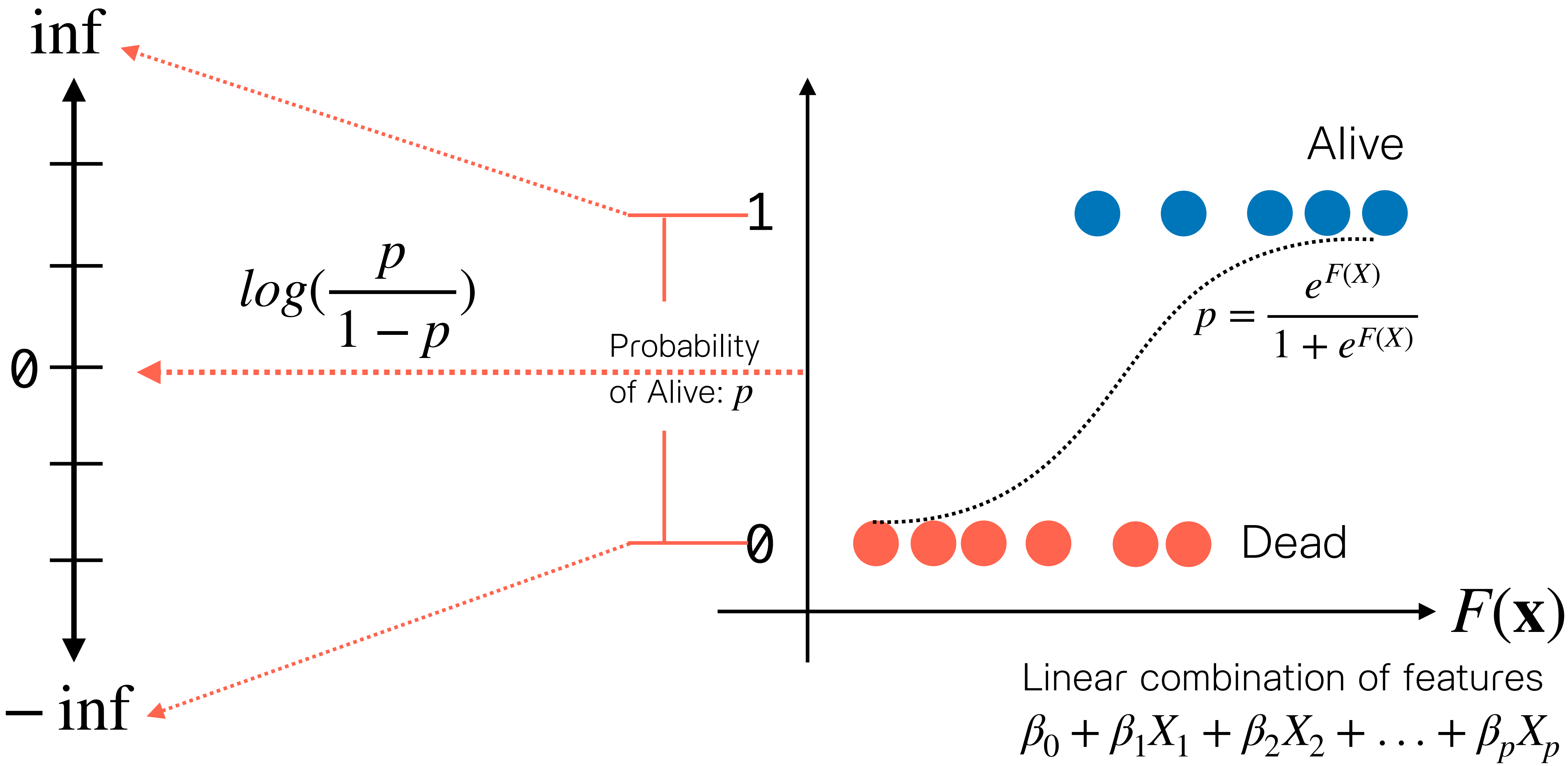
LOC	SYS	AGE	STA	Log-likelihood
No	36	27	Dead	log(0.5)=-0.6931
No	48	59	Dead	log(0.5)=-0.6931
Yes	44	77	Dead	log(0.5)=-0.6931
No	62	54	Dead	log(0.5)=-0.6931
Yes	112	87	Alive	log(0.5)=-0.6931
Yes	108	69	Alive	log(0.5)=-0.6931
No	140	63	Alive	log(0.5)=-0.6931
Yes	138	30	Alive	log(0.5)=-0.6931

$$(0) log(0.5) + (1 - 0) log(1 - 0.5)$$

$$(1) log(0.5) + (1 - 1) log(1 - 0.5)$$

- Since Likelihood is a **goodness of fit** of the model and **log() function is an one to one increasing function, then larger value of log-likelihood means better prediction**.
- That is why, when building logistic regression, the goal is to maximize the log-likelihood.

Review



Since $\frac{p}{1-p} = e^{F(X)} \implies log(\frac{p}{1-p}) = log(odds) = F(X)$

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be differentiable Loss function.

Loss function

- Hence, if we want to use the log-likelihood as a Loss function (where a smaller values represent better fitting model), then we have to multiply the log-likelihood by (-1) that is:

$$L(y, F(\mathbf{X})) = -\sum_{i=1}^{n} \{y_i log(p_i) + (1 - y_i) log(1 - p_i)\}$$

Sometimes called **logLoss** or Cross-entropy

Consider per observation of (-1) x log-likelihood

$$-[y_i log(p_i) + (1 - y_i) log(1 - p_i)] \implies -y_i log(p_i) - (1 - y_i) log(1 - p_i)$$

$$\implies -y_i (log(p_i) - log(1 - p_i)) - log(1 - p_i)$$

$$\implies -y_i [log(\frac{p_i}{1 - p_i})] - log(1 - p_i)$$

$$\implies -y_i [log(odds_i)] - log[1 - \frac{e^{log(odds_i)}}{1 + e^{log(odds_i)}}]$$

$$-y_i[log(odds)] - log[1 - \frac{e^{log(odds)}}{1+e^{log(odds)}}]$$

$$\implies -\, y_i[log(odds)] - log[\frac{1+e^{log(odds)} - e^{log(odds)}}{1+e^{log(odds)}}]$$

$$\implies -\, y_i[log(odds)] - log[\frac{1}{1+e^{log(odds)}}]$$

$$\implies -\, y_i[log(odds)] - log[\frac{1}{1+e^{log(odds)}}]$$

$$\implies -\, y_i[log(odds)] - [log(1) - log(1+e^{log(odds)})]$$

$L(y_i, F(\mathbf{x}_i)) = -\, y_i log(odds) + log(1+e^{log(odds)})$ <——- This is our **Loss function.**

(per observation)

Note: the prediction function $F_i(\mathbf{x_i}) = log(odds)$

Next step, we need to show that the Loss function is differentiable.

$$\frac{\partial}{\partial log(odds)} - y_i log(odds) + log(1 + e^{log(odds)})$$

$$= -y_i + \frac{e^{log(odds_i)}}{1 + e^{log(odds_i)}}$$

$$= -y_i + p_i$$

Loss function

$$L(y, F(\mathbf{X})) = \sum_{i=1}^{n} [-y_i log(odds_i) + log(1 + e^{log(odds_i)})]$$

$$\frac{\partial}{\partial log(odds)} L(y, F(\mathbf{X})) = \sum_{i=1}^{n} [-y_i + p_i]$$

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be diferentiable Loss function.

Step 1: fit the null model (model with only constant term) $\implies F_0(\mathbf{X}) = \underset{\gamma}{\text{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

Like when using a gradient boost for regression, we have to specify the initial prediction via

Since $L(y, \gamma) = \sum_{i=1}^{n} [-y_i\gamma + log(1 + e^{\gamma})]$, and $\frac{\partial}{\partial\gamma} L(y, \gamma) = -\sum_{i=1}^{n} y_i + np$

Let $\frac{\partial}{\partial\gamma} L(y, \gamma) = -\sum_{i=1}^{n} y_i + np = 0$

$$\implies p = \frac{\sum_{i=1}^{n} y_i}{n}$$

$$p = \frac{0 + 0 + 0 + 0 + 1 + 1 + 1 + 1}{8} = \frac{1}{2}$$

LOC	SYS	AGE	STA
No	36	27	Dead
No	48	59	Dead
Yes	44	77	Dead
No	62	54	Dead
Yes	112	87	Alive
Yes	108	69	Alive
No	140	63	Alive
Yes	138	30	Alive

From the initial predicted probability, we have $log(odds) = log(\frac{p}{1-p}) = 0 = F_0(\mathbf{x})$

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be diferentiable Loss function.

Step 1: fit the null model (model with only constant term) $\Longrightarrow F_0(\mathbf{X}) = \underset{\gamma}{\text{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

$$\Longrightarrow F_0(\mathbf{X}) = 0$$

Step2: for m in $1 : M$

1. Compute $e_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ 2, \ldots,\ n$
2. Fit a regression tree to the e_{im} values and create terminal region R_{jm} for $j = 1, \ldots, J_m$
3. For $j = 1,\ 2, \ldots, J_m$ compute $\gamma_{jm} = \underset{x}{\text{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$
4. Update $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_j m)$

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be diferentiable Loss function.

Step 1: fit the null model (model with only constant term) $\implies F_0(\mathbf{X}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

$$\implies F_0(\mathbf{X}) = 0$$

Step2: for $m = 1$

1. Compute $e_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1, 2, \ldots, n$

$$
\begin{aligned}
e_{i1} &= -\left[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}\right]_{F(x)=F_0(x)} \\
&= -\left[-y_i + \frac{e^0}{1 + e^0}\right] \\
&= y_i - 0.5
\end{aligned}
$$

LOC	SYS	AGE	STA	e_{i1}	
No	36	27	Dead	-0.5	$\leftarrow$ $0 - 0.5$
No	48	59	Dead	-0.5	
Yes	44	77	Dead	-0.5	
No	62	54	Dead	-0.5	
Yes	112	87	Alive	0.5	$\leftarrow$ $1 - 0.5$
Yes	108	69	Alive	0.5	
No	140	63	Alive	0.5	
Yes	138	30	Alive	0.5	

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be diferentiable Loss function.

Step 1: fit the null model (model with only constant term) $\implies F_0(\mathbf{X}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

$$\implies F_0(\mathbf{X}) = 0$$

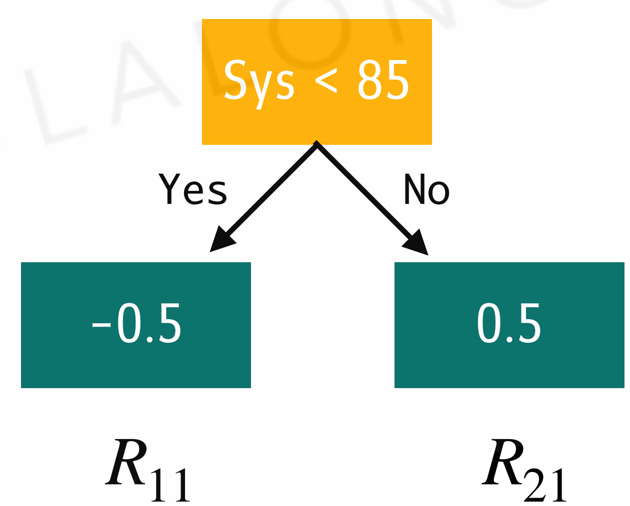
Step 2: for $m = 1$

1. Compute $e_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1, 2, \ldots, n$

2. Fit a regression tree to the e_{im} values and create terminal region R_{jm} for $j = 1, \ldots, J_m$

LOC	SYS	AGE	STA	e_{i1}
No	36	27	Dead	-0.5
No	48	59	Dead	-0.5
Yes	44	77	Dead	-0.5
No	62	54	Dead	-0.5
Yes	112	87	Alive	0.5
Yes	108	69	Alive	0.5
No	140	63	Alive	0.5
Yes	138	30	Alive	0.5

Predict



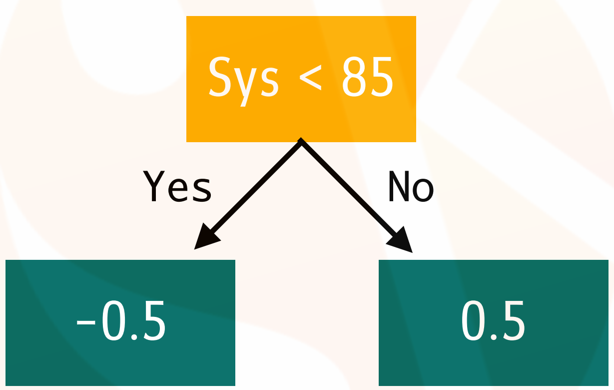
Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be diferentiable Loss function.

Step 1: fit the null model (model with only constant term) $\implies F_0(\mathbf{X}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

$$\implies F_0(\mathbf{X}) = 0$$

Step 2: for $m = 1$

1. Compute $e_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1, 2, \ldots, n$
2. Fit a regression tree to the e_{im} values and create terminal region R_{jm} for $j = 1, \ldots, J_m$
3. For $j = 1, 2, \ldots, J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$



R_{11} R_{21}

$$\gamma_{11} = \frac{-0.5}{0.25} = -2 \qquad \gamma_{21} = \frac{0.5}{0.25} = 2$$

$$\gamma = \frac{\sum_{i=1}^{n} e_{im}}{\sum_{i=1}^{n} p_i(1 - p_i)}, \text{ where } p = \frac{e^{log(odds)}}{1 + e^{log(odds)}}$$

Let $(X, y) = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ be a training dataset, and $L(y_i, F(\mathbf{x_i}))$ be diferentiable Loss function.

Step 1: fit the null model (model with only constant term) $\implies F_0(\mathbf{X}) = \underset{\gamma}{\text{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

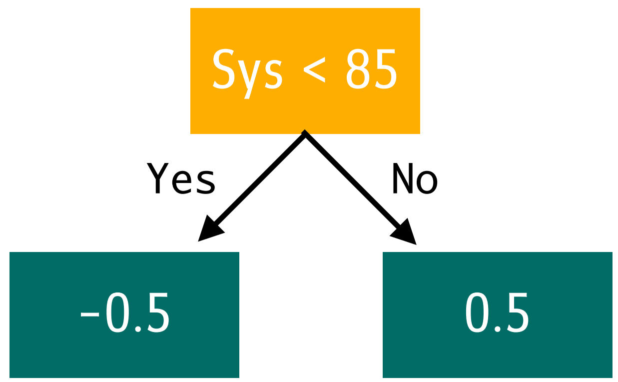
$\implies F_0(\mathbf{X}) = 0$

Step 2: for $m = 1$

1. Compute $e_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ 2, \ldots,\ n$
2. Fit a regression tree to the e_{im} values and create terminal region R_{jm} for $j = 1, \ldots, J_m$
3. For $j = 1,\ 2, \ldots, J_m$ compute $\gamma_{jm} = \underset{\gamma}{\text{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$
4. Update $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_j m)$

$$F_1(\mathbf{x}) = F_0(\mathbf{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_j m)$$

- $\nu \in (0,1)$ is the learning rate
- Learnign rate is an effect of the regression tree on the final prediction



R_{11} $\qquad$ R_{21}

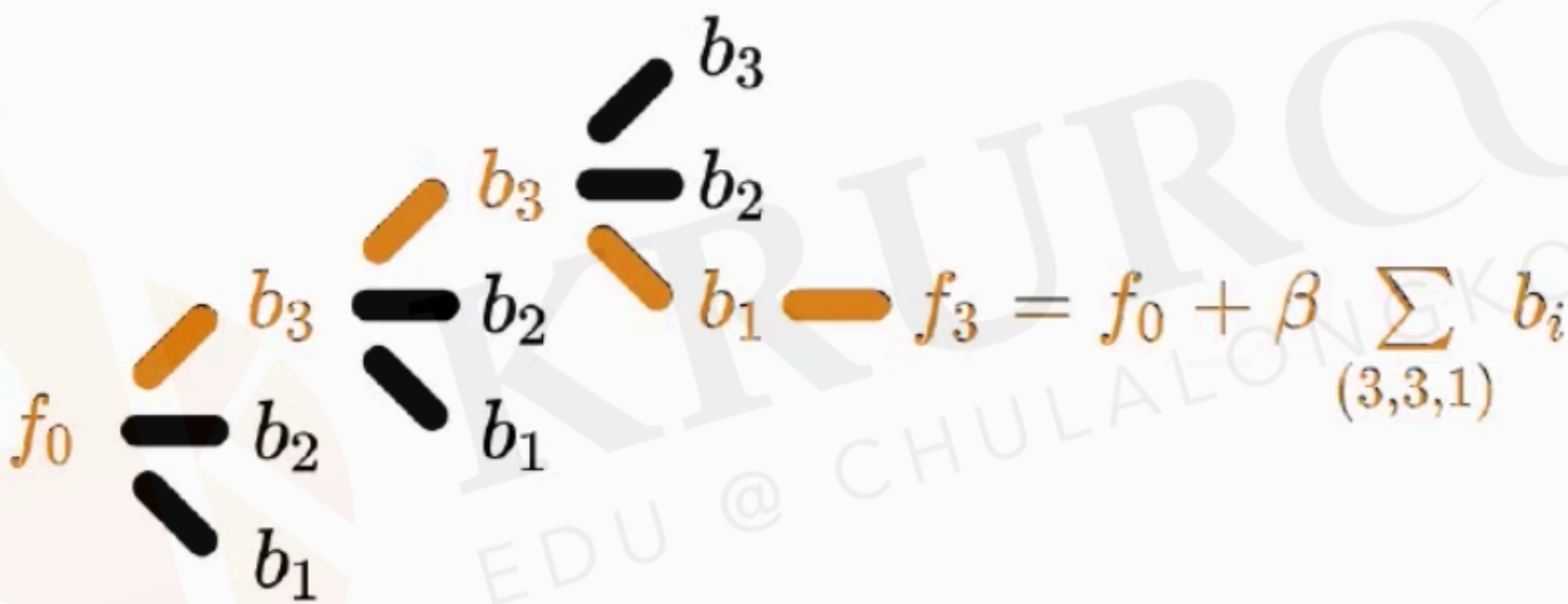
$$\gamma_{11} = \frac{-0.5}{0.25} = -2 \qquad \gamma_{21} = \frac{0.5}{0.25} = 2$$

$$F_1(\mathbf{x}) = 0 + (0.9)\sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_j m)$$

LOC	SYS	AGE	STA	New predicted log(odds)	prob(Alive)
No	36	27	Dead	-1.8	0.141851064900488
No	48	59	Dead	-1.8	0.141851064900488
Yes	44	77	Dead	-1.8	0.141851064900488
No	62	54	Dead	-1.8	0.141851064900488
Yes	112	87	Alive	1.8	0.858148935099512
Yes	108	69	Alive	1.8	0.858148935099512
No	140	63	Alive	1.8	0.858148935099512
Yes	138	30	Alive	1.8	0.858148935099512

repeat …

Component-wise Gradient boosting



https://www.youtube.com/watch?v=nOMSQJU51Tk

1. Initialize the function estimate $\hat{f}^{[0]}$ with offset values. Note that $\hat{f}^{[0]}$ is a vector of length n. In the following paragraphs, we will generally denote the vector of function estimates at iteration m by $\hat{f}^{[m]}$.

2. Specify a set of *base-learners*. Base-learners are simple regression estimators with a fixed set of input variables and a univariate response. The sets of input variables are allowed to differ among the base-learners. Usually, the input variables of the base-learners are small subsets of the set of predictor variables $x_1, \dots, x_p$. For example, in the simplest case, there is exactly one base-learner for each predictor variable, and the base-learners are just simple linear models using the predictor variables as input variables. Generally, the base-learners considered in this paper are either penalized or unpenalized least squares estimators using small subsets of the predictor variables as input variables (see Section 3.2.1 for details and examples). Each base-learner represents a modeling alternative for the statistical model. Denote the number of base-learners by P and set $m = 0$.

3. Increase m by 1, where m is the number of iterations.

4. a) Compute the negative gradient $-\frac{\partial \rho}{\partial f}$ of the loss function and evaluate it at $\hat{f}^{[m-1]}(\mathbf{x}_i^\top)$, $i = 1, \dots, n$ (i.e., at the estimate of the previous iteration). This yields the negative gradient vector

$$\mathbf{u}^{[m]} = \left(u_i^{[m]}\right)_{i=1,\dots,n} := \left(-\frac{\partial}{\partial f}\rho\left(y_i, \hat{f}^{[m-1]}(\mathbf{x}_i^\top)\right)\right)_{i=1,\dots,n}.$$

   b) Fit each of the P base-learners to the negative gradient vector, i.e., use each of the regression estimators specified in step 2 separately to fit the negative gradient. The resulting P regression fits yield P vectors of predicted values, where each vector is an estimate of the negative gradient vector $\mathbf{u}^{[m]}$.

   c) Select the base-learner that fits $\mathbf{u}^{[m]}$ best according to the residual sum of squares (RSS) criterion and set $\hat{\mathbf{u}}^{[m]}$ equal to the fitted values of the best-fitting base-learner.

   d) Update the current estimate by setting $\hat{f}^{[m]} = \hat{f}^{[m-1]} + \nu\,\hat{\mathbf{u}}^{[m]}$, where $0 < \nu \leq 1$ is a real-valued step length factor.

5. Iterate Steps 3 and 4 until the stopping iteration m_{stop} is reached (the choice of m_{stop} is discussed below).

https://cran.r-project.org/web/packages/mboost/vignettes/mboost_tutorial.pdf

Package ‘compboost’

October 28, 2018

Type Package

Title C++ Implementation of Component-Wise Boosting

Version 0.1.0

Maintainer Daniel Schalk <daniel.schalk@stat.uni-muenchen.de>

Description C++ implementation of component-wise boosting implementation of component-wise boosting written in C++ to obtain high runtime performance and full memory control. The main idea is to provide a modular class system which can be extended without editing the source code. Therefore, it is possible to use R functions as well as C++ functions for custom base-learners, losses, logging mechanisms or stopping criteria.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.1.0

Imports Rcpp (>= 0.11.2), methods, glue, R6, checkmate

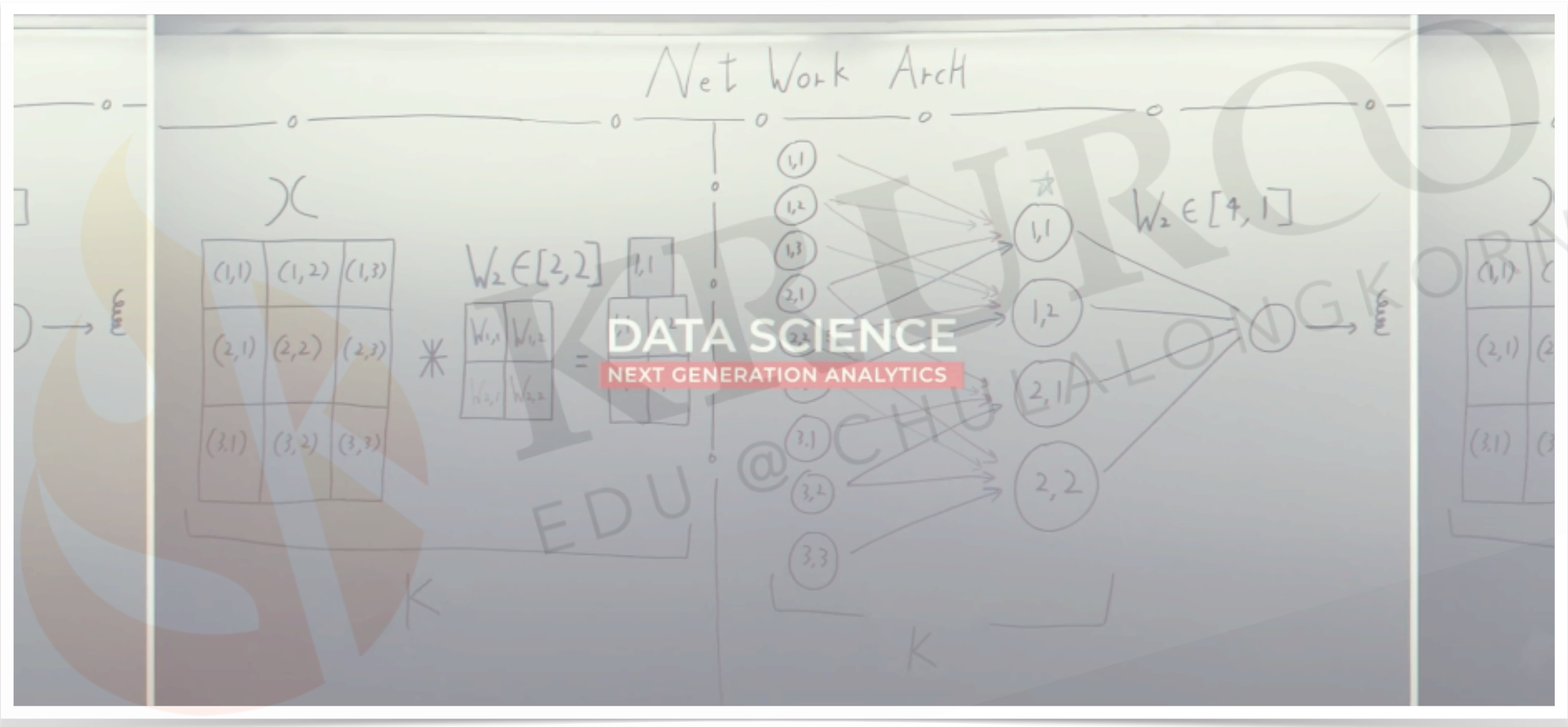
LinkingTo Rcpp, RcppArmadillo

Suggests RcppArmadillo (>= 0.9.100.5.0), ggplot2, testthat, rpart,

```
> boostLinear()
> boostSplines()
```

https://cran.r-project.org/web/packages/compboost/compboost.pdf

Automated Feature Selection via component-wise boosting algorithm



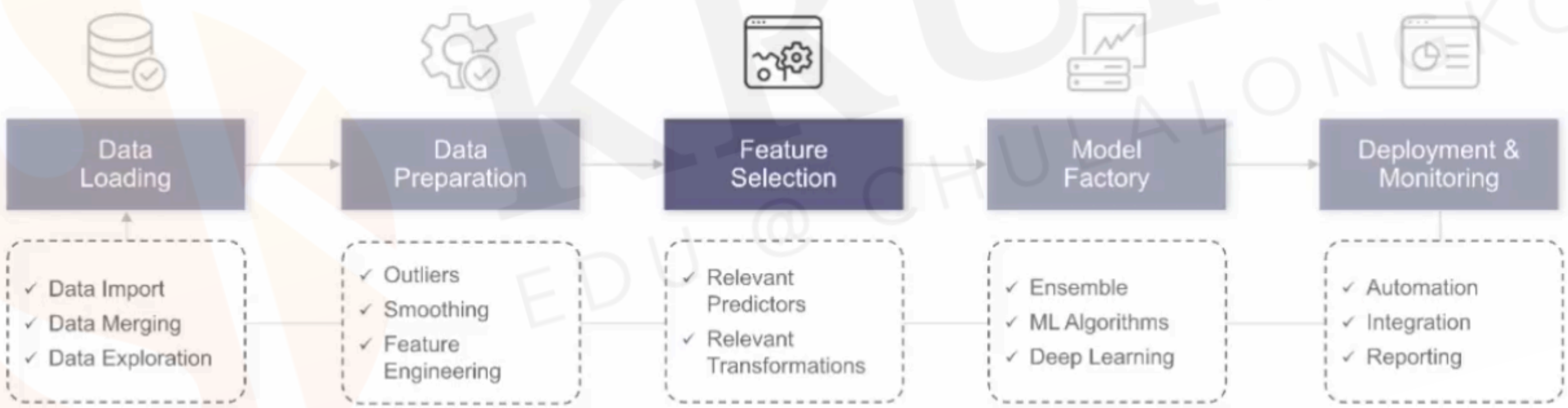
https://www.youtube.com/watch?v=ucSt28PPUPY&t=1264s

https://github.com/STATWORX/bounceR

Feature Selection

- Features contain information about the outcome variable.
- More features = more information?

  = better prediction performance?



(STATWORX, 2018)

Feature Selection

- Irrelevant features
- Redundant feature

Feature Selection methods

- **Filter methods** - using bivariate statistic
- **Wrapper methods**
  - iteratively searching for the best feature set
  - Model-based approach

https://www.youtube.com/watch?v=ucSt28PPUPY&t=1264s

https://topepo.github.io/caret/variable-importance.html#model-specific-metrics

Result: Stability Score Distribution for Feature Space S

foreach *n_rounds in N_ROUNDS* **do**

  initialize Stability Score Distribution Ω^j;

  foreach *n_mods in N_MODS* **do**

    draw s_i features from S^*;

    bootstrap q_i obs from Q obs;

    run componentwise boosting on set (q_i, s_i);

    penalize score in Ω^j for s_i^- for not surviving boosting run ;

    reward score in Ω^j for s_i^+ for surviving boosting run ;

    if *score for feature* $k <$ *threshold* **then**

      cut feature k from S^*

    return Stability Score Distribution $\Omega^j_{updated}$

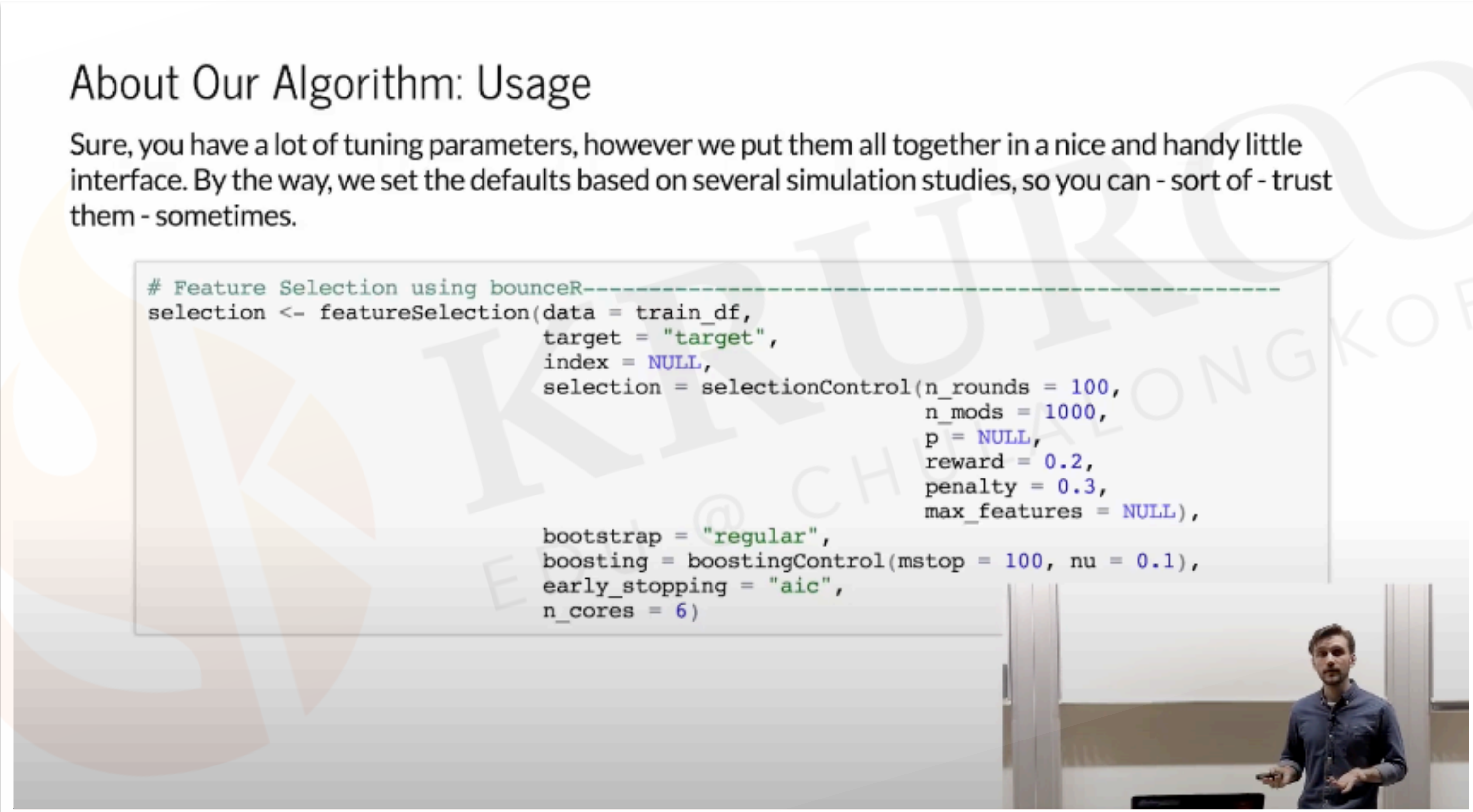
  end

  return average stability score distribution $\bar{\Omega}$

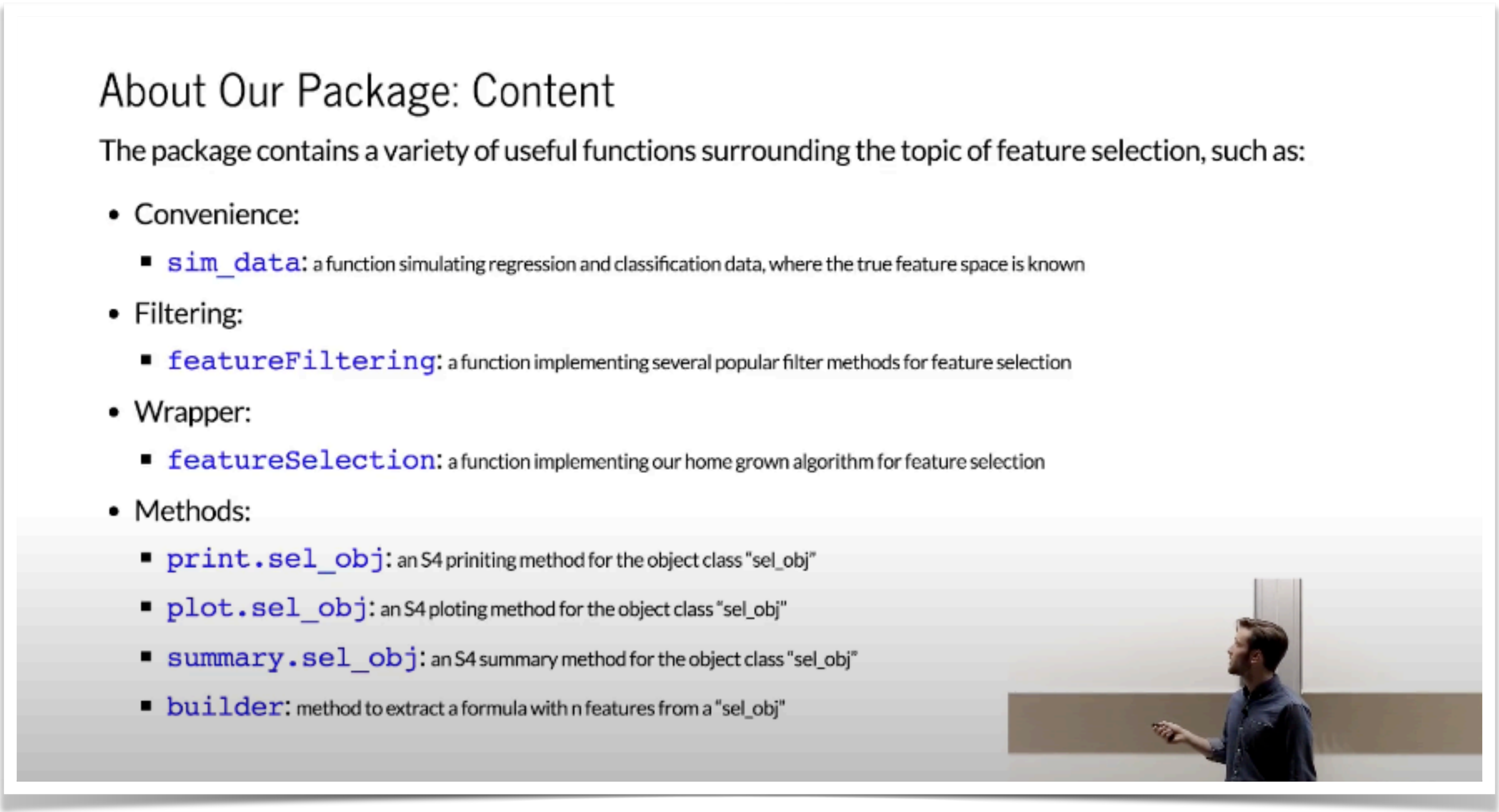
end

https://www.youtube.com/watch?v=ucSt28PPUPY&t=1264s

```r
# load devtools
install.packages(devtools)
library(devtools)

# download from our public repo
devtools::install_github("STATWORX/bounceR")

# source it
library(bounceR)
```



https://www.youtube.com/watch?v=ucSt28PPUPY&t=1264s

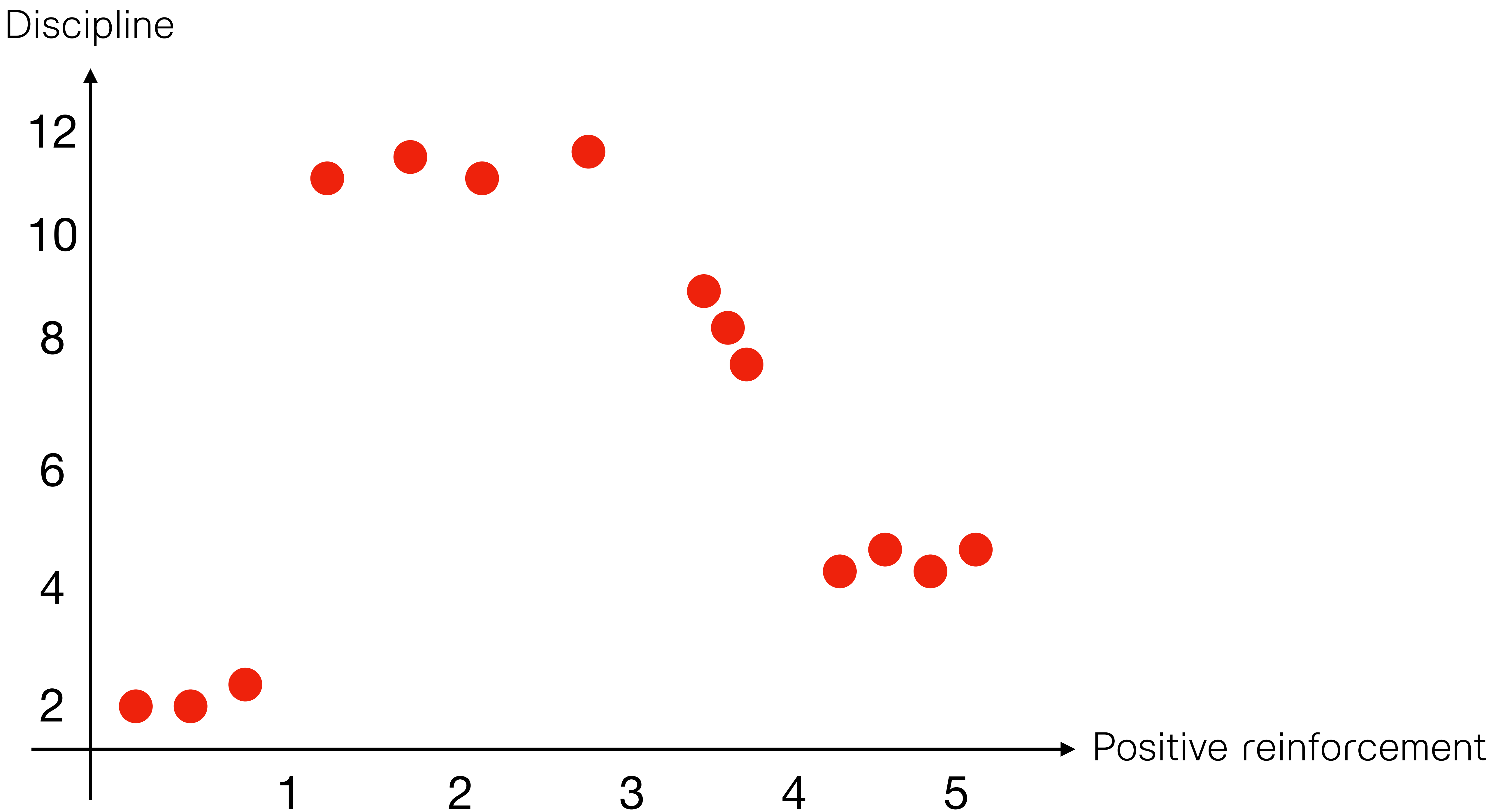


https://www.youtube.com/watch?v=ucSt28PPUPY&t=1264s

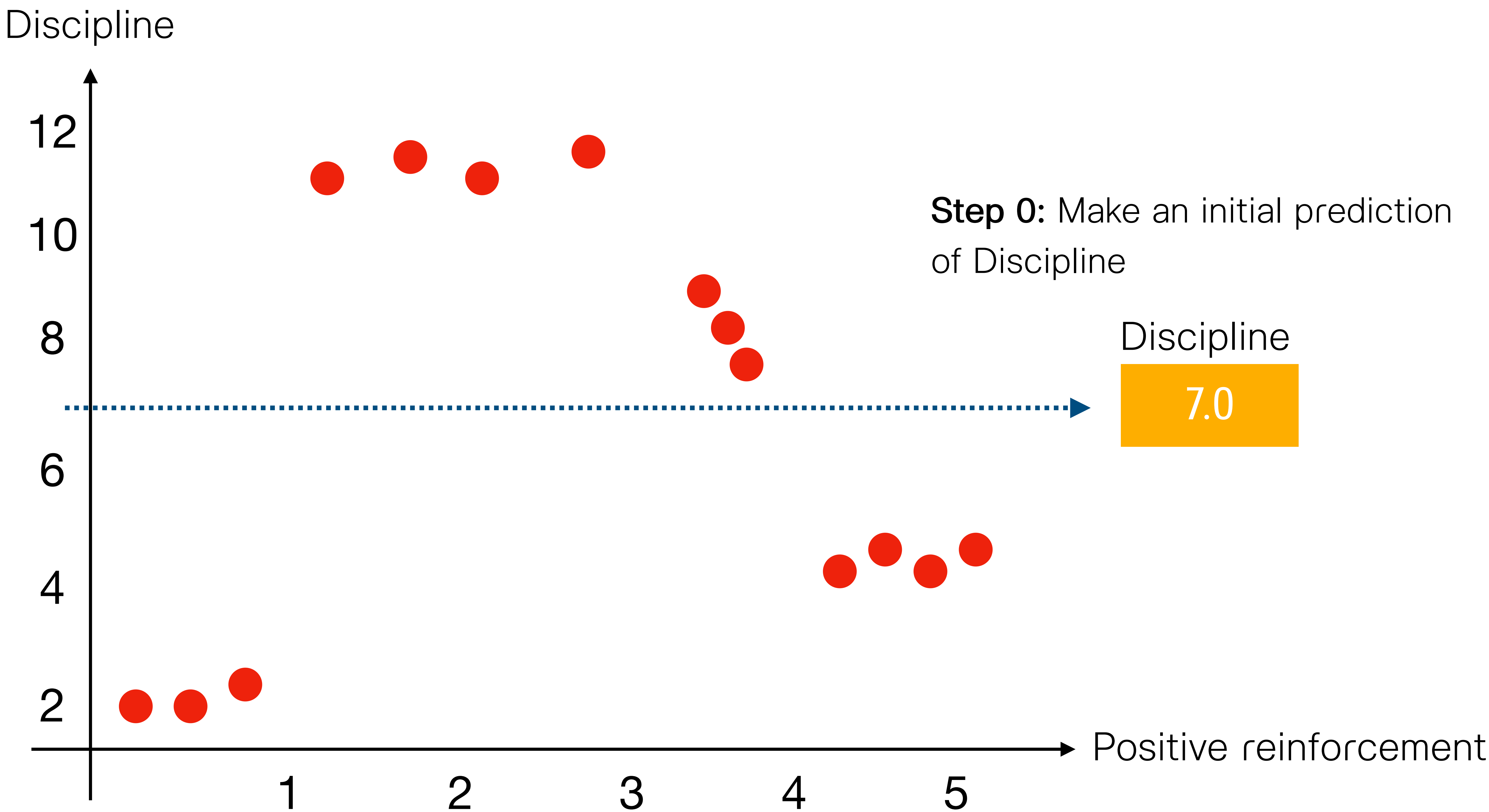
XGBoost Trees

- Regression
- Classification

XGBoost Trees for Regression

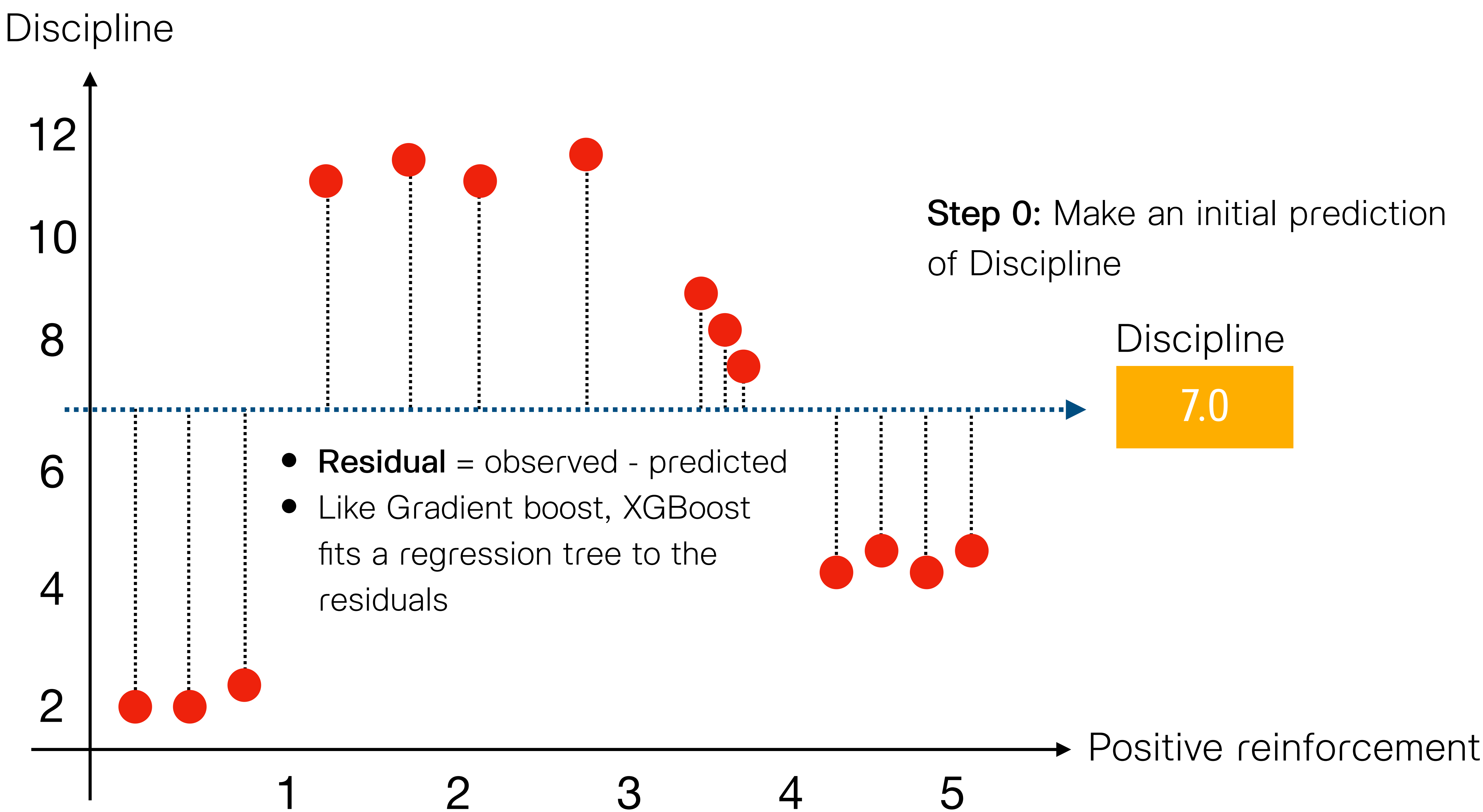


XGBoost Trees for Regression



Step 0: Make an initial prediction of Discipline

XGBoost Trees for Regression



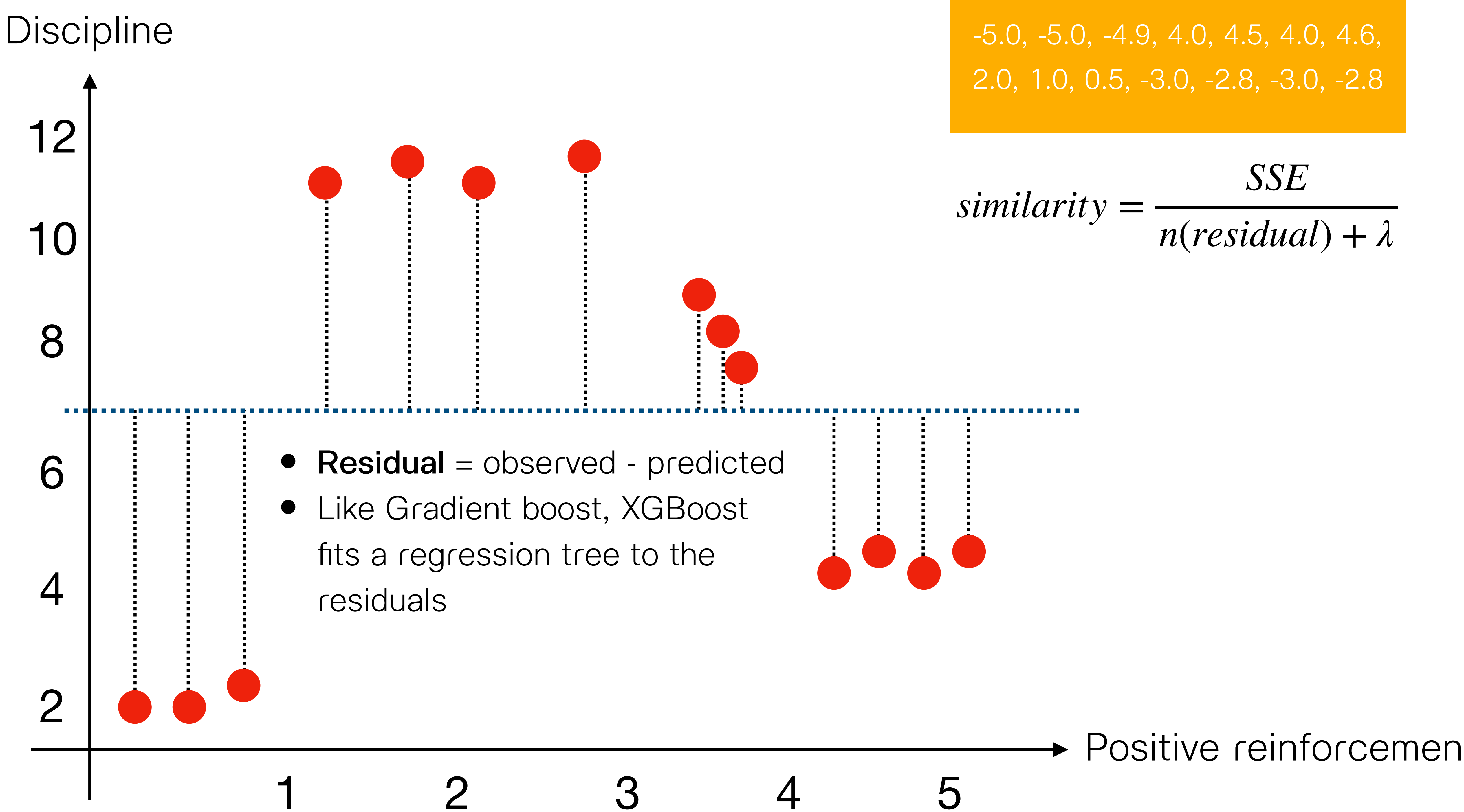
Step 0: Make an initial prediction of Discipline

- **Residual** = observed - predicted
- Like Gradient boost, XGBoost fits a regression tree to the residuals

XGBoost Trees for Regression

-5.0, -5.0, -4.9, 4.0, 4.5, 4.0, 4.6,
2.0, 1.0, 0.5, -3.0, -2.8, -3.0, -2.8

$$similarity = \frac{SSE}{n(residual) + \lambda}$$

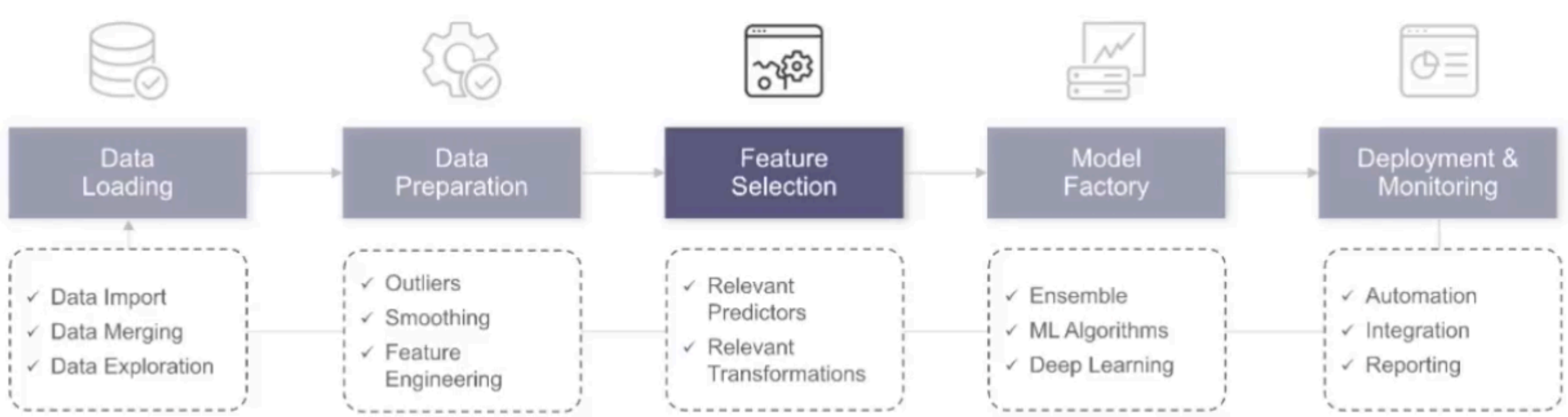


- **Residual** = observed - predicted
- Like Gradient boost, XGBoost fits a regression tree to the residuals

Support Vector Machine

- Introduction
- Support Vector Classification - Linear Kernels
- Polynomial Kernels
- Radius Basis Function Kernels

[redacted]



Data Loading
Data Preparation
Feature Selection
Model Factory
Deployment & Monitoring
✓ Data Import
✓ Data Merging
✓ Data Exploration
✓ Outliers
✓ Smoothing
✓ Feature Engineering
✓ Relevant Predictors
✓ Relevant Transformations
✓ Ensemble
✓ ML Algorithms
✓ Deep Learning
✓ Automation
✓ Integration
✓ Reporting

Feature Selection Problem

Why do we need feature selection?

With too many features, most models can’t be identified or are noisy

The rigth combination of features boosts performance, not algorithm.

Features are all information you have to predict your outcome.

ref: STATWORX

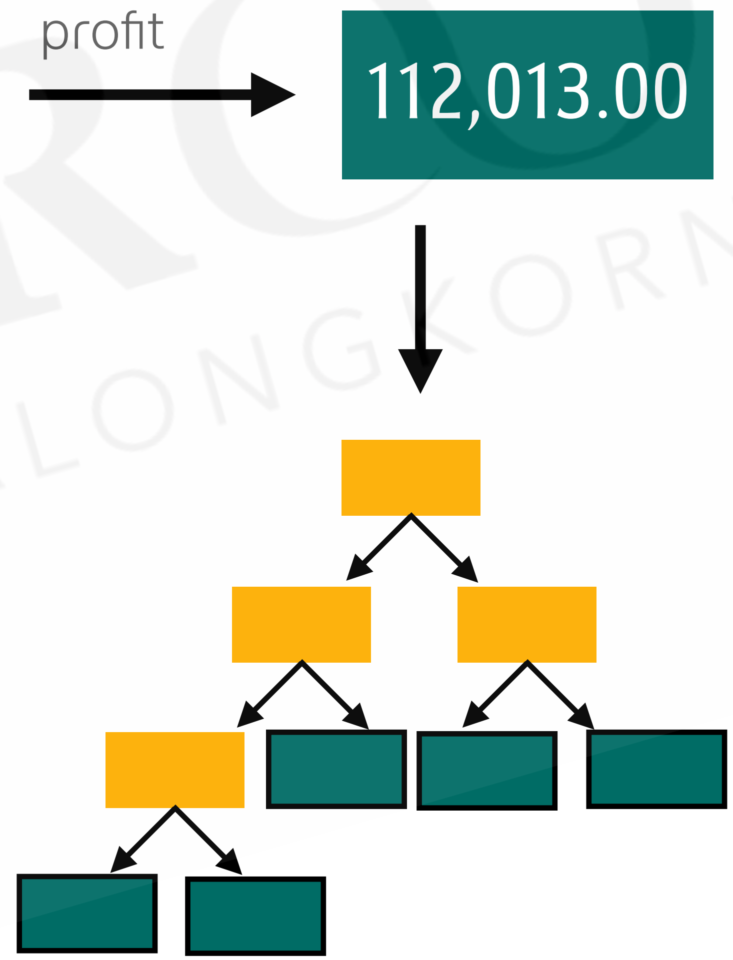
How can we do feature selection?

- Manualy selecting (testing) all feature set.
- Using correlation matrix to filter the relevant features.
- Loop over all feature set combinations and evaluate them based on performance gain

Gradient Boost for Regression

```
> head(dat)
  R.D.Spend Administration Marketing.Spend      State   Profit
1  165349.2      136897.80        471784.1   New York 192261.8
2  162597.7      151377.59        443898.5 California 191792.1
3  153441.5      101145.55        407934.5    Florida 191050.4
4  144372.4      118671.85        383199.6   New York 182902.0
5  142107.3       91391.77        366168.4    Florida 166187.9
6  131876.9       99814.71        362861.4   New York 156991.1
```

Average value of profit



- Then Gredient Boost builds a tree.
- Like AdaBoost , this tree is based on the error made by the previous tree.
- In contrast with AdaBoost, this tree is usually larger than a stump.

Step1: Build up the initial leaf

SES	Level	Gender	Discipline
-1.0	Primary	Male	27
0.5	Secondary	Female	59
1.0	UnderGrad	Female	77
1.5	Primary	Male	54
1.5	UnderGrad	Male	87
1.0	Secondary	Female	69

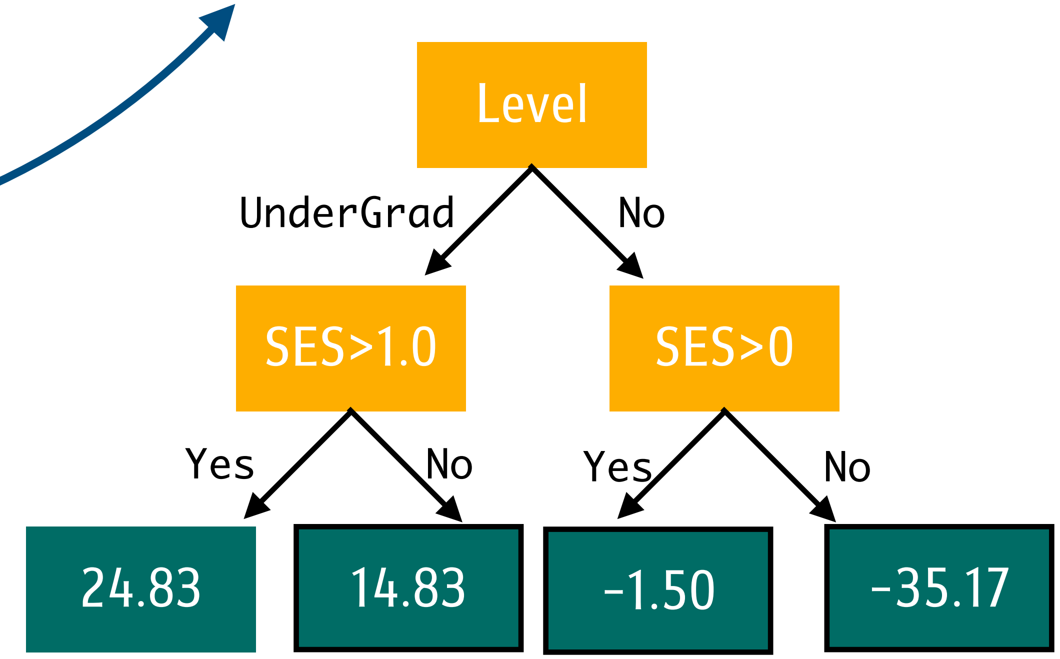
Average value of Discipline score

62.17

Step2: Build a tree based on the error from the first tree.

SES	Level	Gender	Discipline	(Pseudo)Residual = actual - predicted
-1.0	Primary	Male	27	-35.17
0.5	Secondary	Female	59	-3.17
1.0	UnderGrad	Female	77	14.83
1.5	Primary	Male	54	-8.17
1.5	UnderGrad	Male	87	24.83
1.0	Secondary	Female	69	6.83

Predict



Gradient Boosting using R

```
> install.packages("gbm")
> library(gbm)
```

Step2: Build a tree based on the error from the first tree.

SES	Level	Gender	Learning Discipline
-1.0	Primary	Male	27
0.5	Secondary	Female	59
1.0	UnderGrad	Female	77
1.5	Primary	Male	54
1.5	UnderGrad	Male	87
1.0	Secondary	Female	69

Average value of LD score

62.17

Objective: we will create a forest of stumps with AdaBoost to predict a patient status (1 = alive or 0 = dead)

1. Give an **equal weight** to each data point.

```
> head(dat,10)
```

	ID	STA	AGE	CAN	SYS	TYP	LOC	Weight1
1	8	0	27	0	142	1	0	0.005
2	12	0	59	0	112	1	0	0.005
3	14	0	77	0	100	0	0	0.005
4	28	0	54	0	142	1	0	0.005
5	32	0	87	0	110	1	0	0.005
6	38	0	69	0	110	1	0	0.005
7	40	0	63	0	104	0	0	0.005
8	41	0	30	0	144	1	0	0.005
9	42	0	35	0	108	1	0	0.005
10	50	0	70	1	138	0	0	0.005

Objective: we will create a forest of stumps with AdaBoost to predict a patient status (1 = alive or 0 = dead)

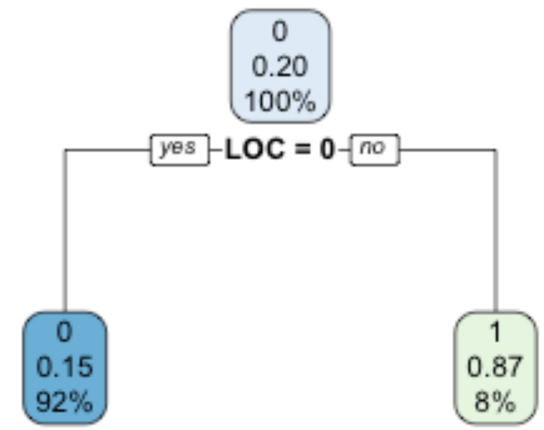
```
> head(dat,10)
```

	ID	STA	AGE	CAN	SYS	TYP	LOC	Weight1
1	8	0	27	0	142	1	0	0.005
2	12	0	59	0	112	1	0	0.005
3	14	0	77	0	100	0	0	0.005
4	28	0	54	0	142	1	0	0.005
5	32	0	87	0	110	1	0	0.005
6	38	0	69	0	110	1	0	0.005
7	40	0	63	0	104	0	0	0.005
8	41	0	30	0	144	1	0	0.005
9	42	0	35	0	108	1	0	0.005
10	50	0	70	1	138	0	0	0.005

1. Give an **equal weight** to each data point.

$$weight = \frac{1}{m} = \frac{1}{200}$$

2. Make the first stump



Objective: we will create a forest of stumps with AdaBoost to predict a patient status (1 = alive or 0 = dead)

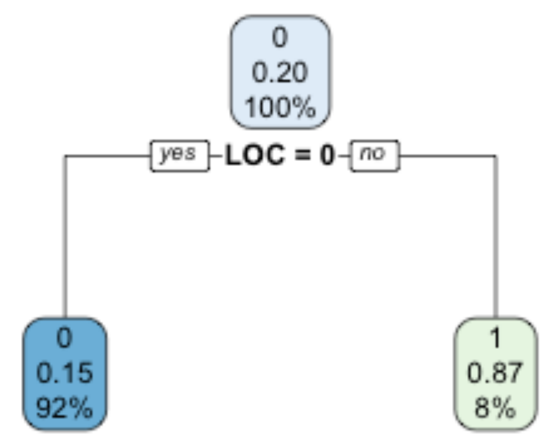
```
> head(dat,10)
```

	ID	STA	AGE	CAN	SYS	TYP	LOC	Weight1
1	8	0	27	0	142	1	0	0.005
2	12	0	59	0	112	1	0	0.005
3	14	0	77	0	100	0	0	0.005
4	28	0	54	0	142	1	0	0.005
5	32	0	87	0	110	1	0	0.005
6	38	0	69	0	110	1	0	0.005
7	40	0	63	0	104	0	0	0.005
8	41	0	30	0	144	1	0	0.005
9	42	0	35	0	108	1	0	0.005
10	50	0	70	1	138	0	0	0.005

1. Give an **equal weight** to each data point.

$$weight = \frac{1}{m} = \frac{1}{200}$$

2. Make the first stump



3. Evaluate classification error of LOC

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 158  27
         1   2  13
```

Objective: we will create a forest of stumps with AdaBoost to predict a patient status (1 = alive or 0 = dead)

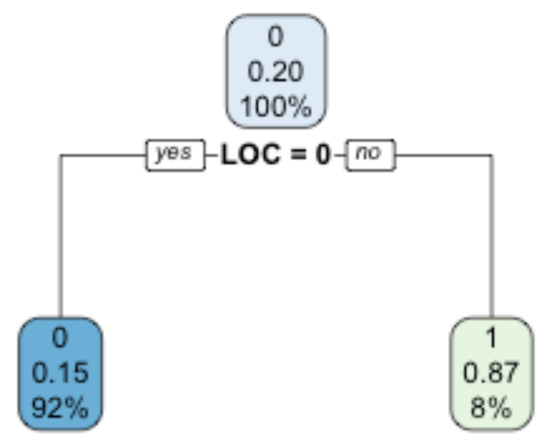
```
> head(dat,10)
```

	ID	STA	AGE	CAN	SYS	TYP	LOC	Weight1
1	8	0	27	0	142	1	0	0.005
2	12	0	59	0	112	1	0	0.005
3	14	0	77	0	100	0	0	0.005
4	28	0	54	0	142	1	0	0.005
5	32	0	87	0	110	1	0	0.005
6	38	0	69	0	110	1	0	0.005
7	40	0	63	0	104	0	0	0.005
8	41	0	30	0	144	1	0	0.005
9	42	0	35	0	108	1	0	0.005
10	50	0	70	1	138	0	0	0.005

1. Give an **equal weight** to each data point.

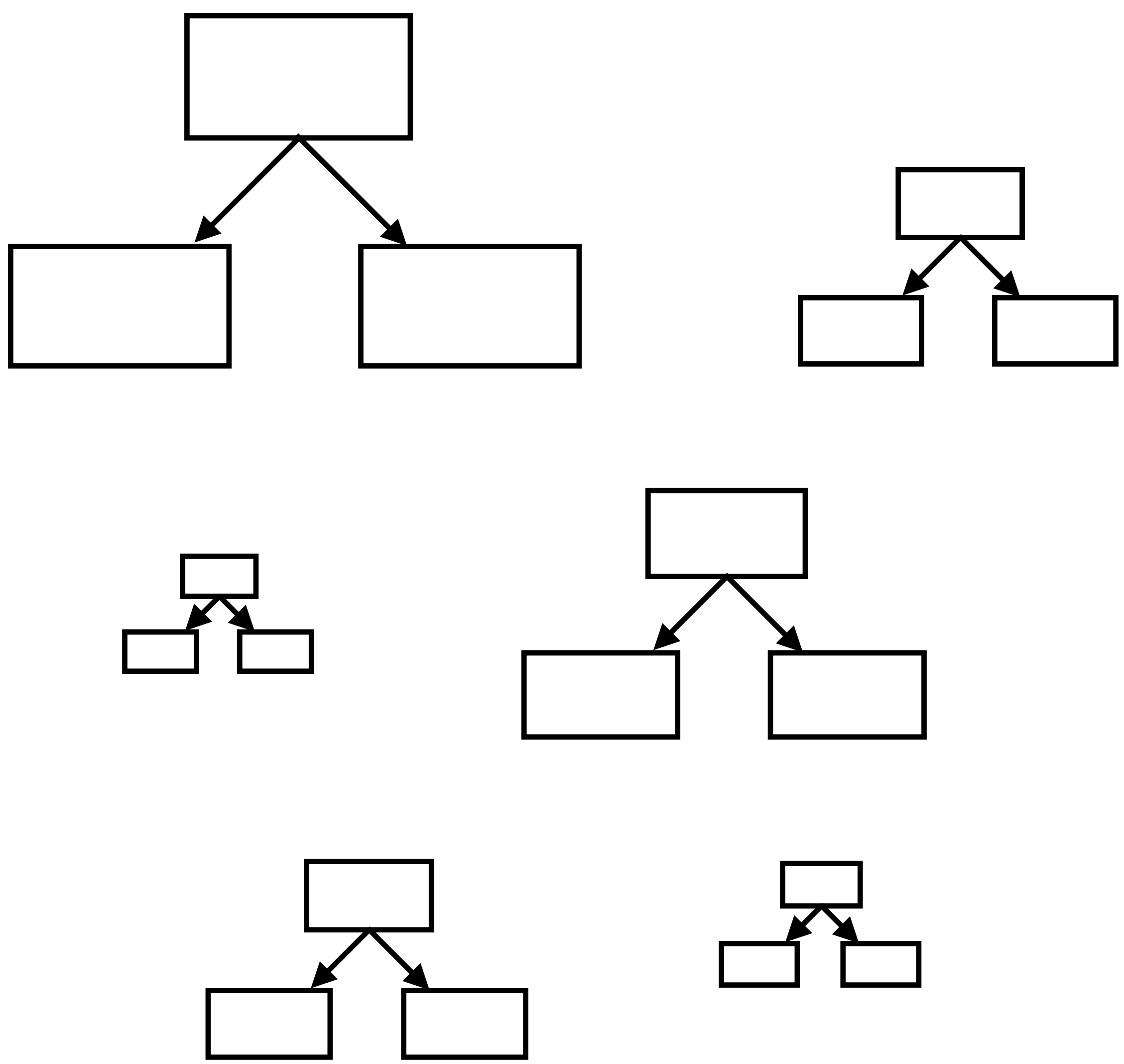
$$weight = \frac{1}{m} = \frac{1}{200}$$

2. Make the first stump



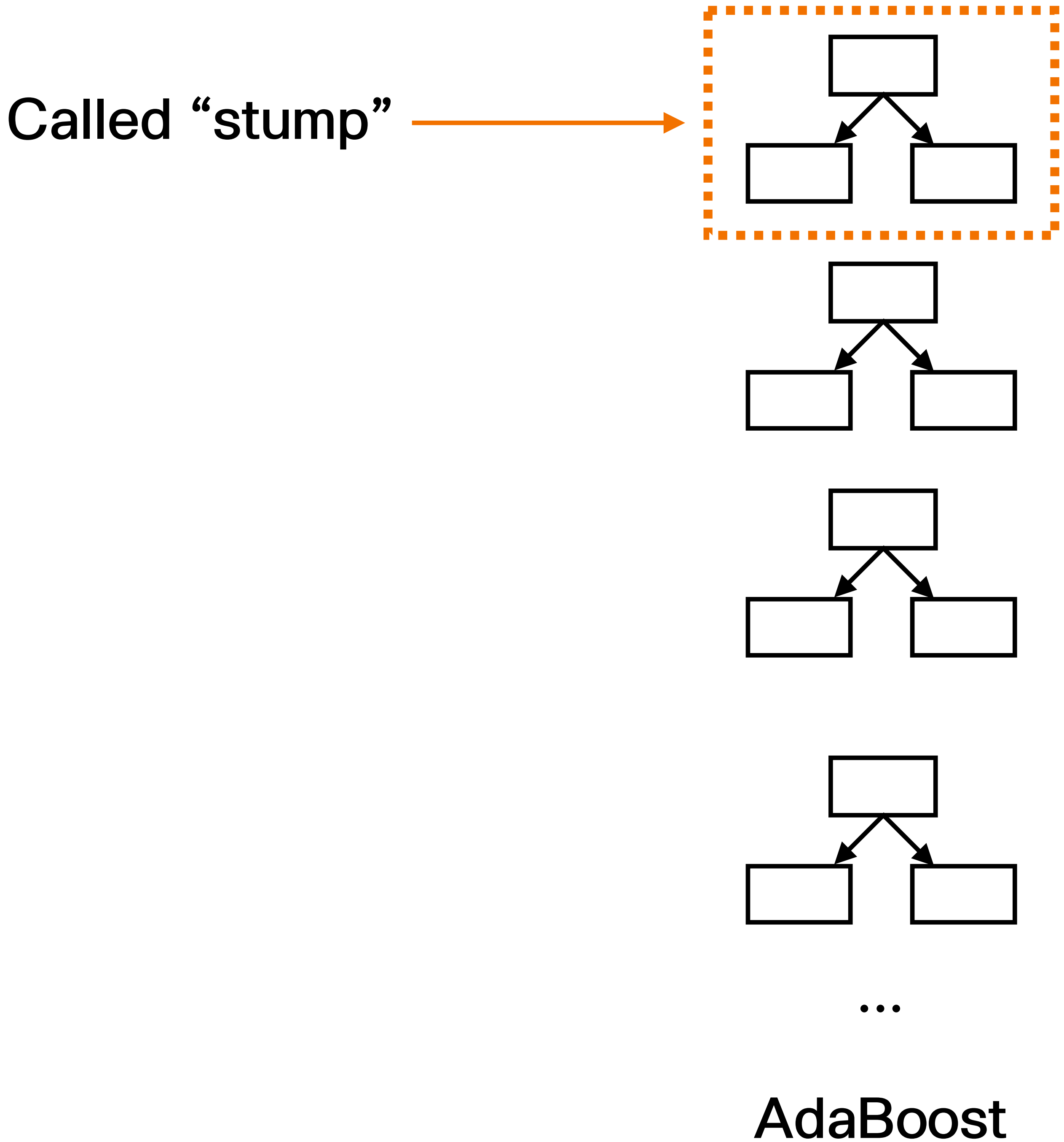
3. Evaluate classification error of LOC

Main concepts behind AdaBoost

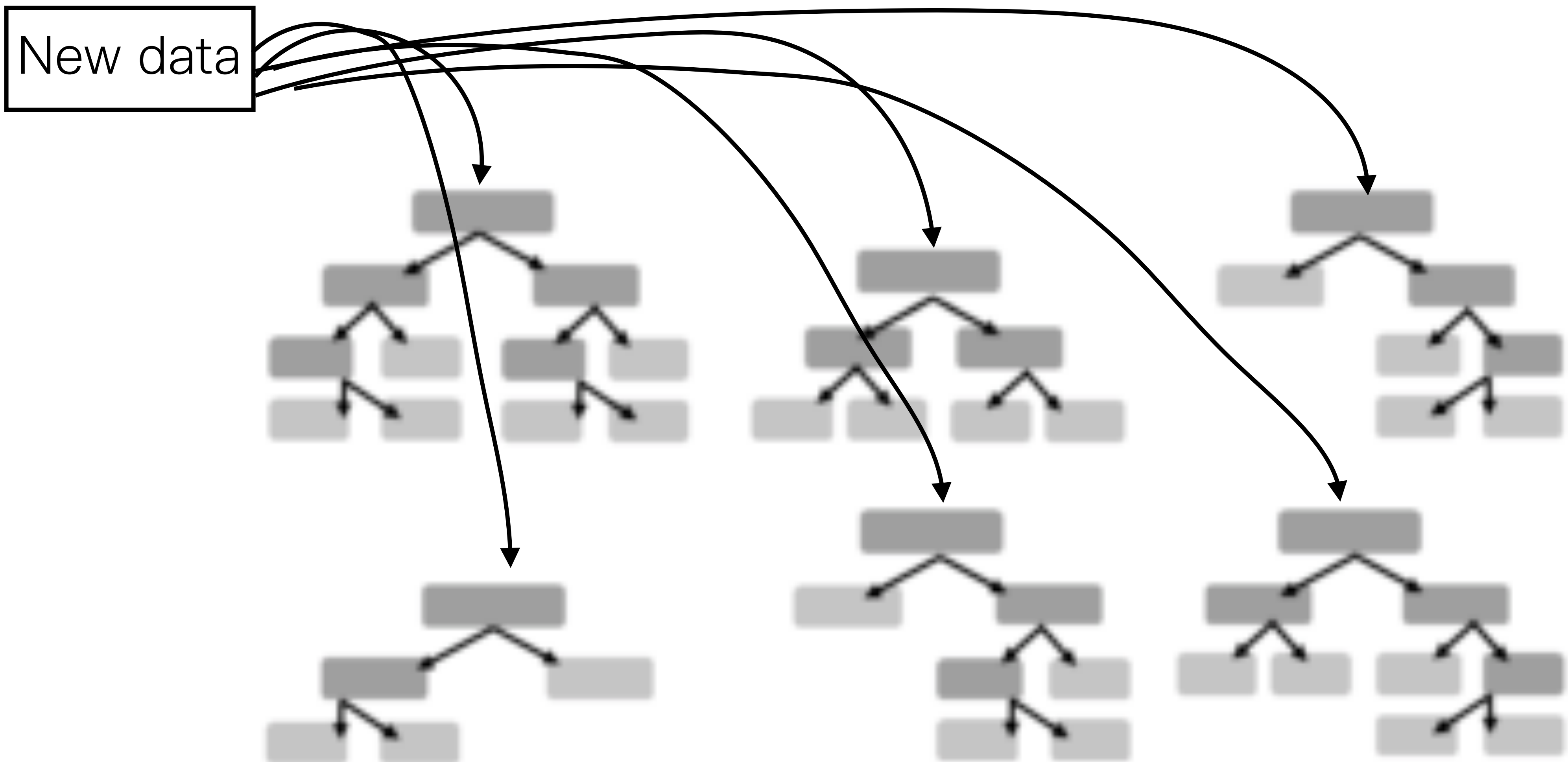


Forest of stumps made with AdaBoost, some stumps get more weight in the final classification than others.

Main concepts behind AdaBoost



Random Forest



Deposit	
Yes	No
5	1

A Formal view on AdaBoost

1. Given a training dataset (X, y) which contains data points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$
2. For iteration $t = 1,2,...,T$
   - Construct distribution of W_t on $\{1,\ 2,\ 3,...,\ m\}$, where $W_t(i)$ is the weight attributed to subject i on the iteration t
     - Let $W_1(i) = 1/m$
   - Build up a classifier $M_t : X \to \{0,1\}$
   - Compute the error associated to the classifier, let $e_t(i) = 1$ if $M_t(i) \neq y_i$, and $e_t(i) = 0$ if $M_t(i) = y_i$

http://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf

In a random forest, each time you make a tree, you make a full sized tree.

Out-of-bag error vs Out-sample error

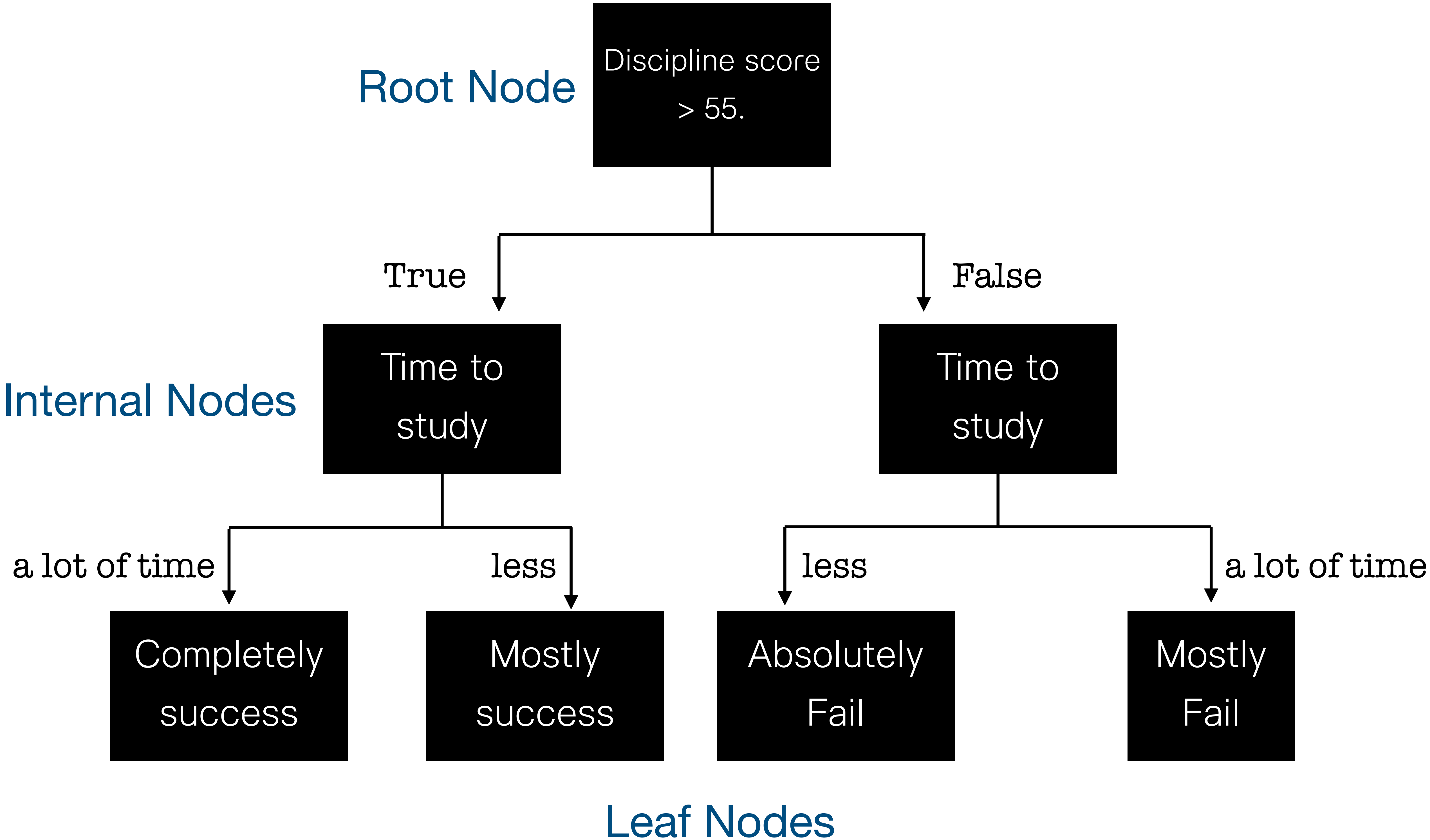
- OOB error only estimate errors (not AUC, sens, spec,…)
- Can't compare OBB error to other types of models

Estimate the accuracy of a random forest

- Out of bag error

1. Create a “bootstrapped” dataset

Tree models


Root Node
Discipline score
> 55.
True
False
Internal Nodes
Time to study
Time to study
a lot of time
less
less
a lot of time
Completely success
Mostly success
Absolutely Fail
Mostly Fail
Leaf Nodes

Advantages

- Simple to understand, interpret, visualize
- Can handle both numerical and categorical features

No need to standardised or normalised

No need to create dummy variables

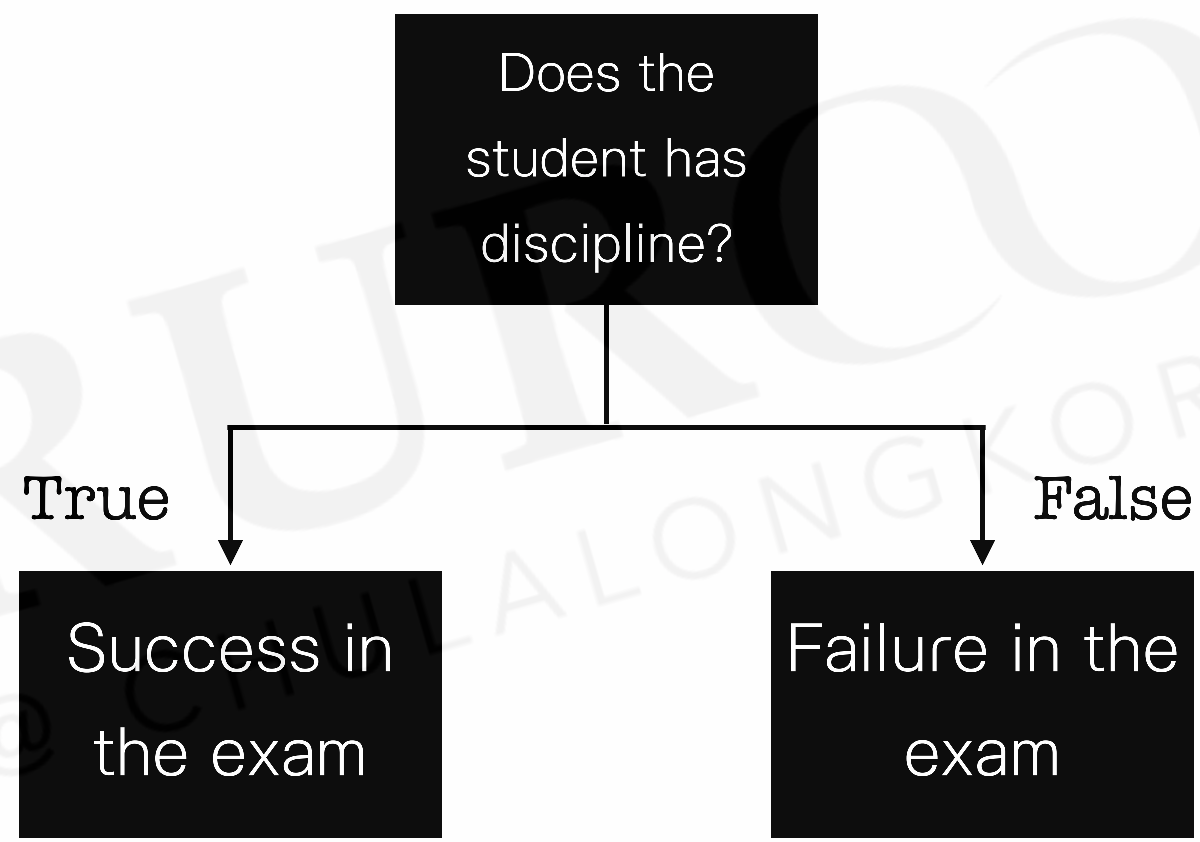
- Can handle missing data
- Robust to outliers, then trees require little or no data preparation
- Can model non-linearity in the data
- Trees can handle large datasets

Disadvantages

- Large trees can be hard to interpret.
- Trees models tend to perform high variance. Hence the models need to be tuned up by choosing the most appropriate hyperparameter.

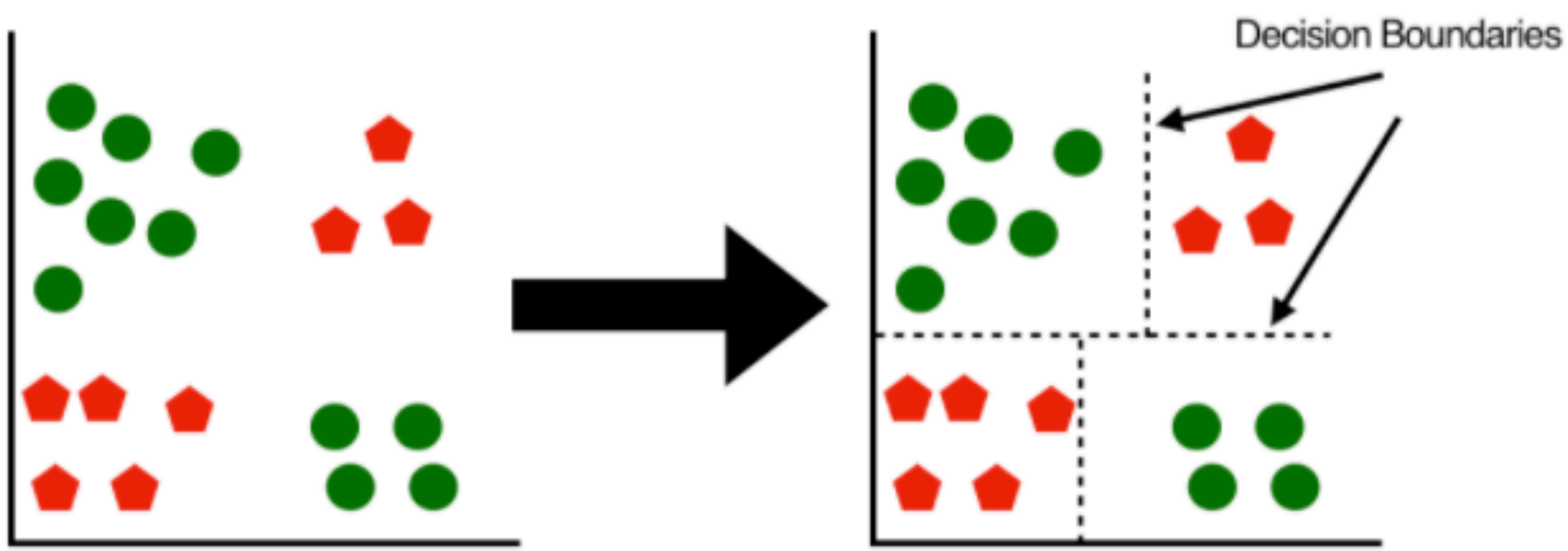
Tree-based models

- Classification trees
- Regression trees
- Bagged Trees
- Random Forests
- Boosted Trees (GBM)



Classification Trees

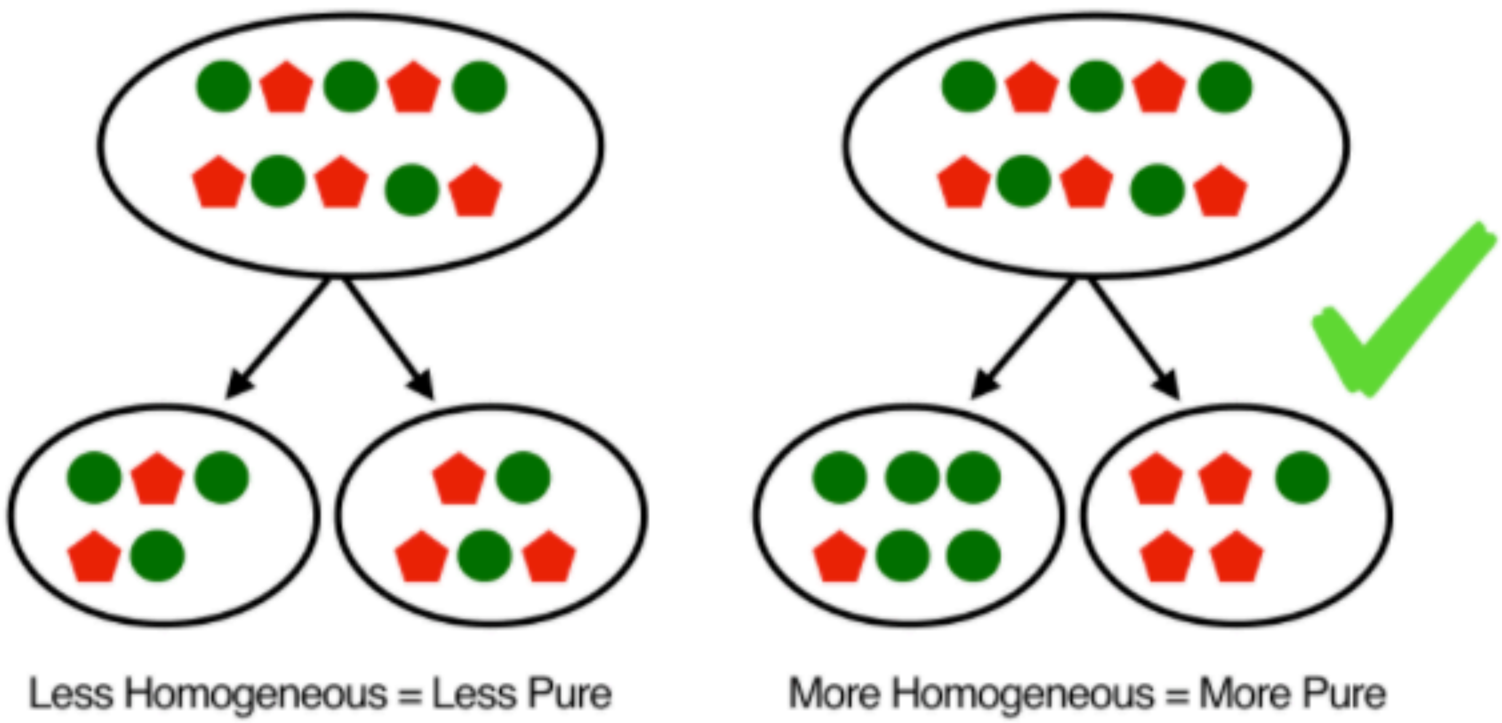
Split the data into "pure" regions



- Decision tree model makes classification decision based on the **decision boundaries.**

- The example above, there are 100% pure sub-region (only one class for each region)

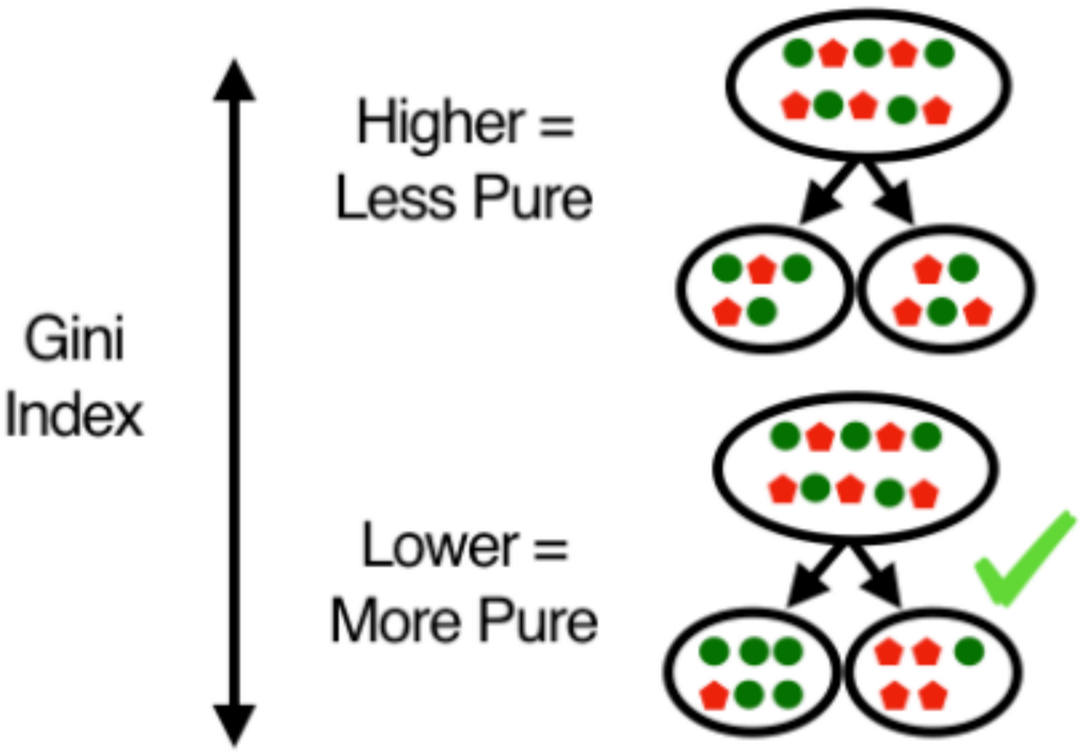
How to determine the best split?



Goal of classification tree is to partition data at a node into subset that are as pure as possible.

Impurity measure

- Measure of a node specifies how mixed the resulting subsets are.
- We want to fine a node specifies that minimised the impurity measure
- Common impurity measures
    - Entropy
    - Gini impurity
    - Misclassification rate

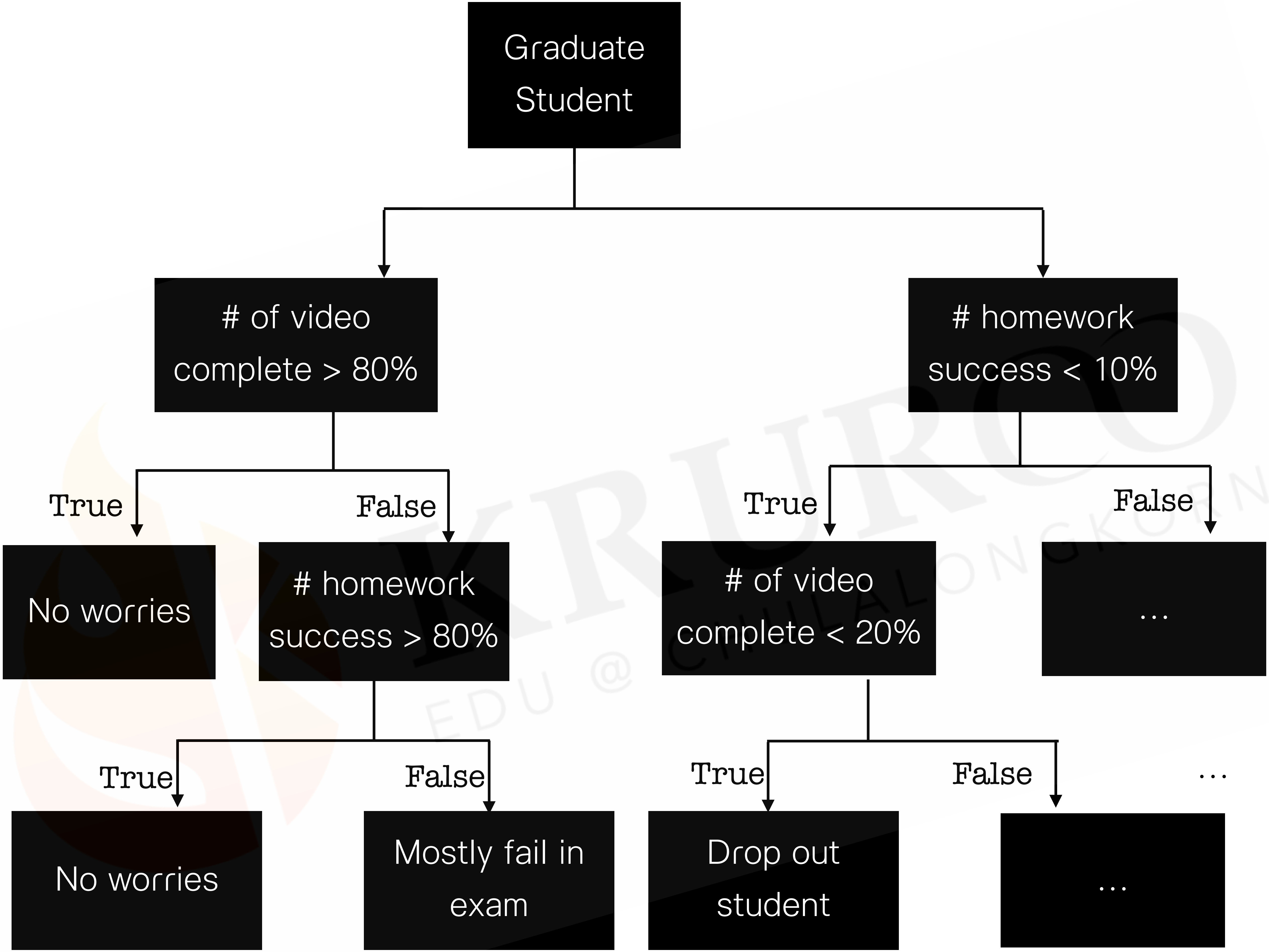


Show some calculation

Building up the model

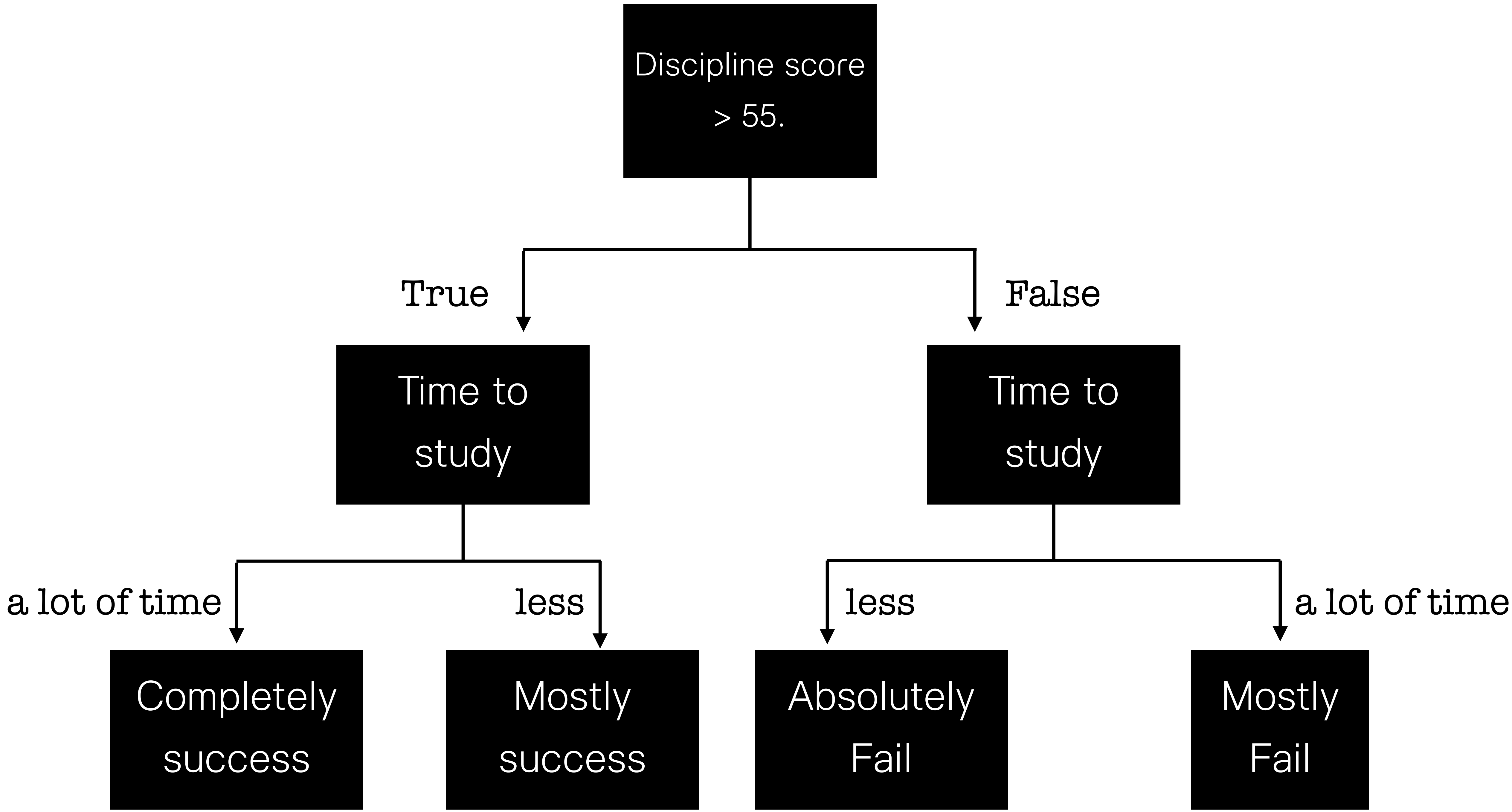
1. Split data into train and test dataset. (80/20)
2. Train classification tree using train dataset

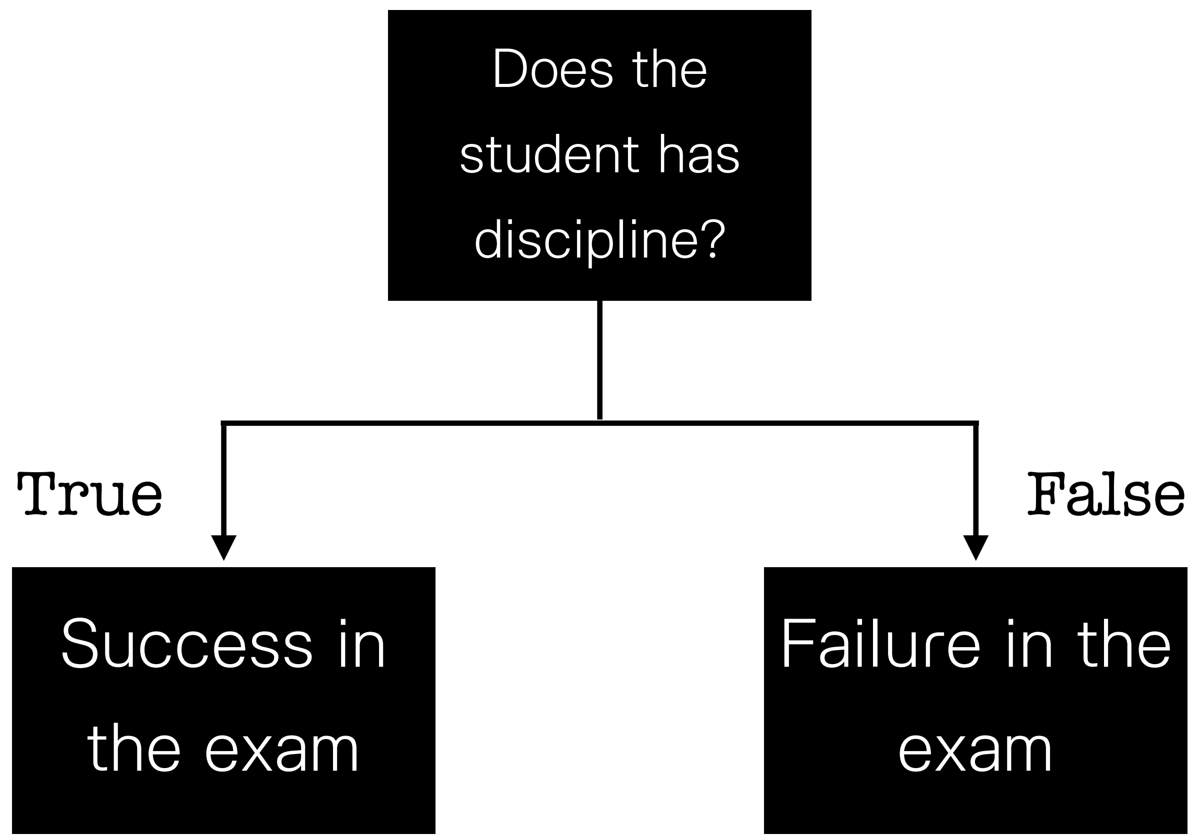
```
install.packages("rpart")
install.packages("rpart.plot")
library(rpart)
library(rpart.plot)

fit<-rpart(formula,data,method="class",
            control = rpart.control(cp, minsplit,maxdepth,xval),
            parms = list(split="information"))
            # For classification splitting, the list can contain any of: the vector of prior
            probabilities (component prior), the loss matrix (component loss) or the splitting
            index (component split). The priors must be positive and sum to 1. The loss matrix
            must have zeros on the diagonal and positive off-diagonal elements. The splitting
            index can be gini or information. The default priors are proportional to the data
            counts, the losses default to 1, and the split defaults to gini.
```


Graduate Student
# of video complete > 80%
homework success < 10%
True
False
No worries
homework success > 80%
True
False
# of video complete < 20%
...
True
False
No worries
False
Mostly fail in exam
True
Drop out student
False
...
...

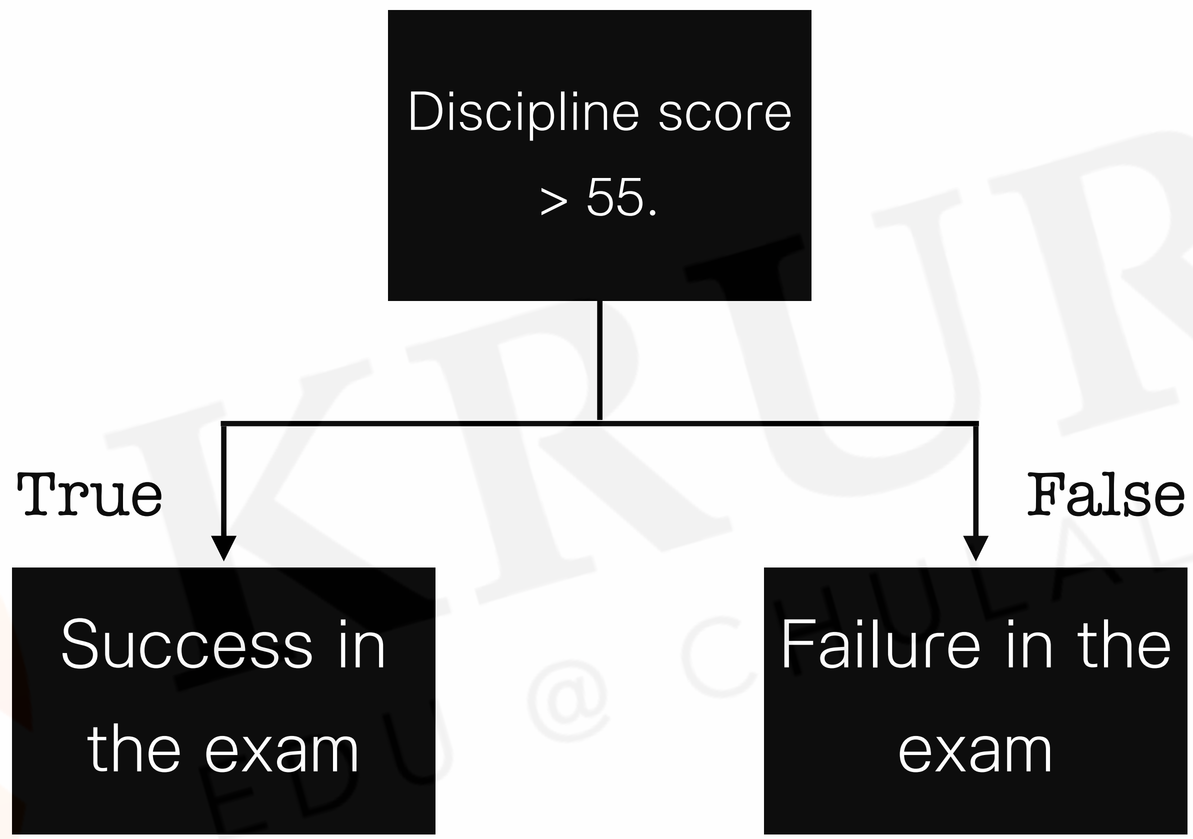
Apple

- Discipline = 70
- No time to study

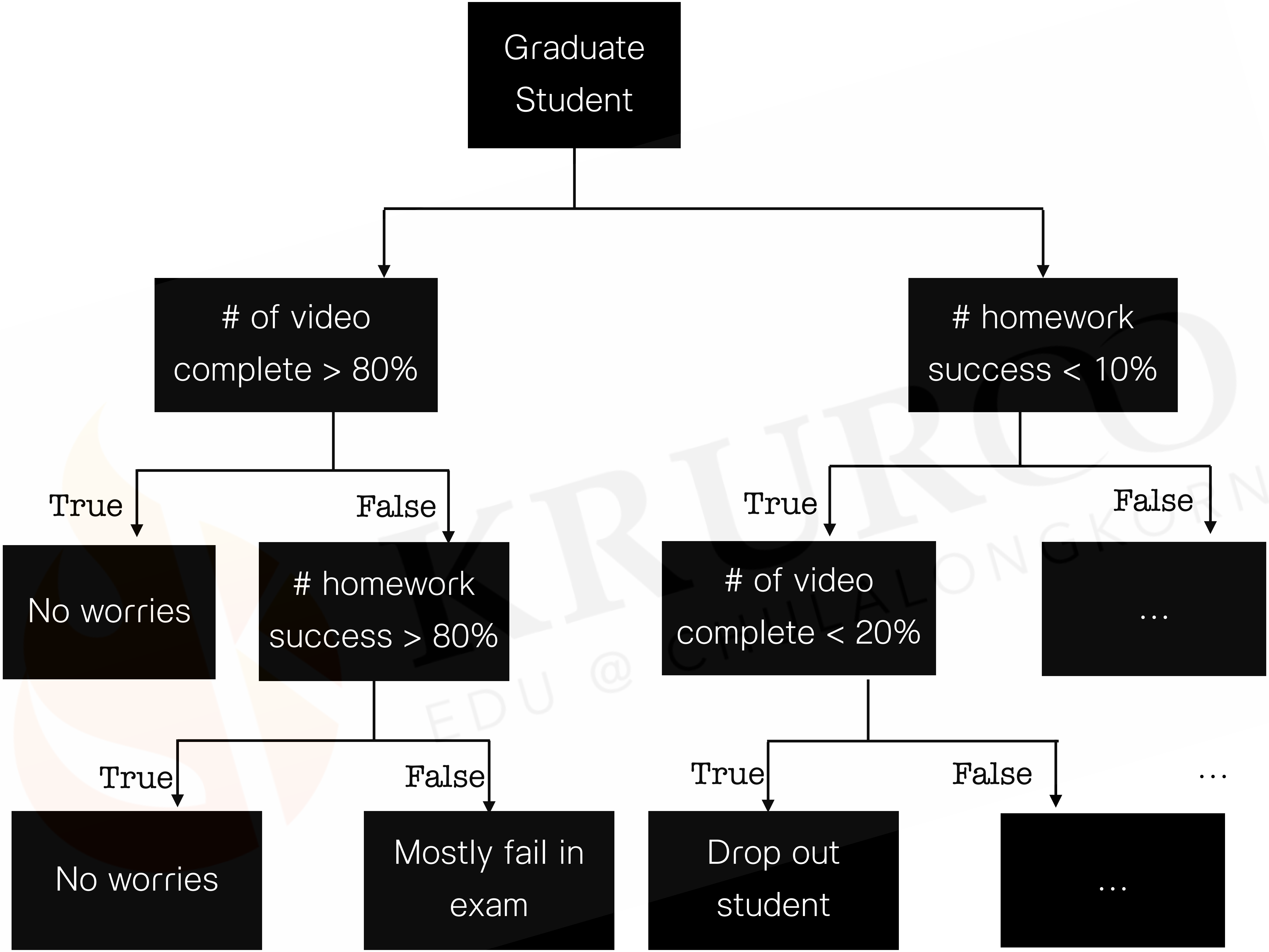




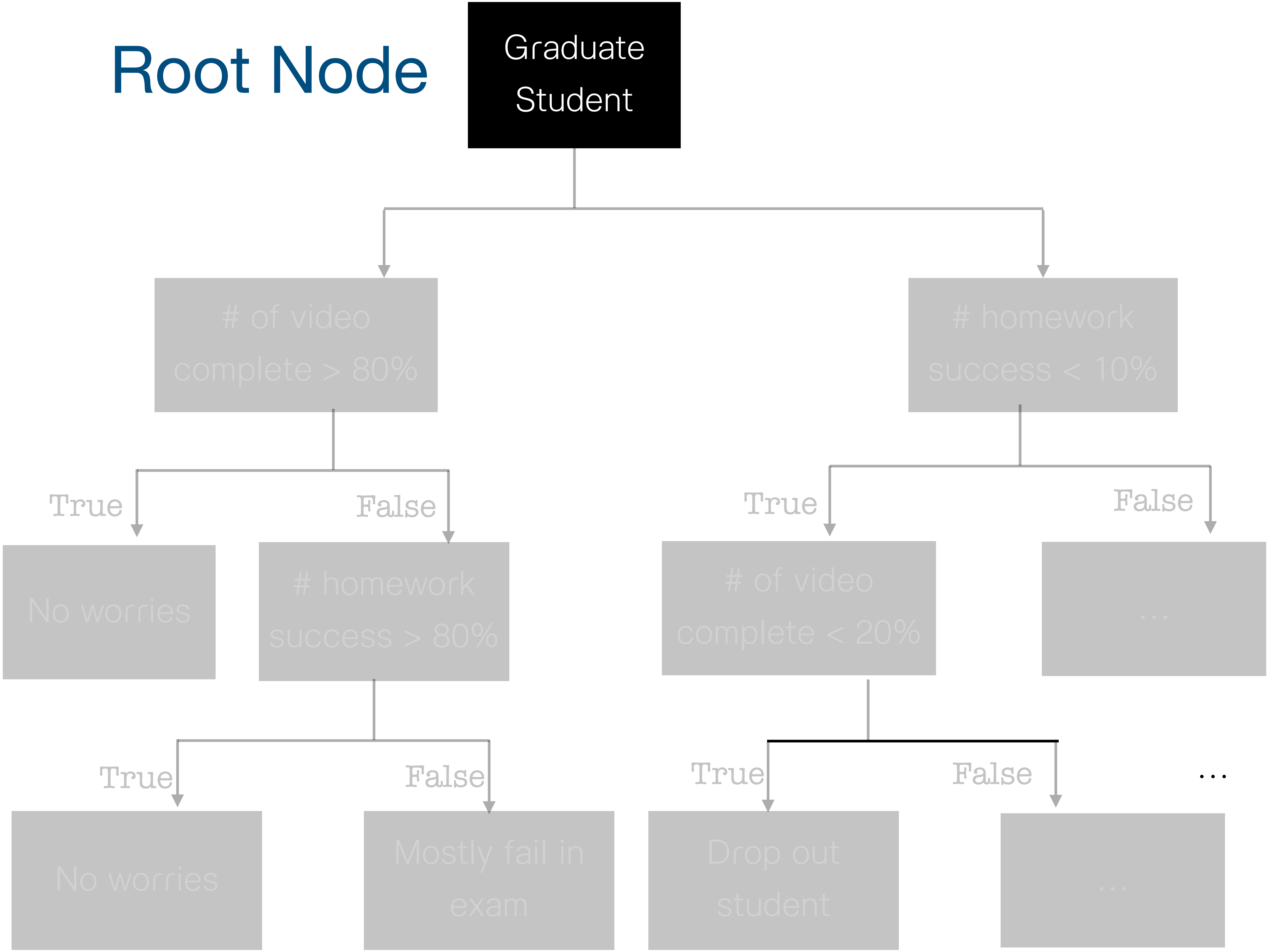
Decision tree ask a question, and then classifies the subject based on the answer.



Decision tree for numeric data.

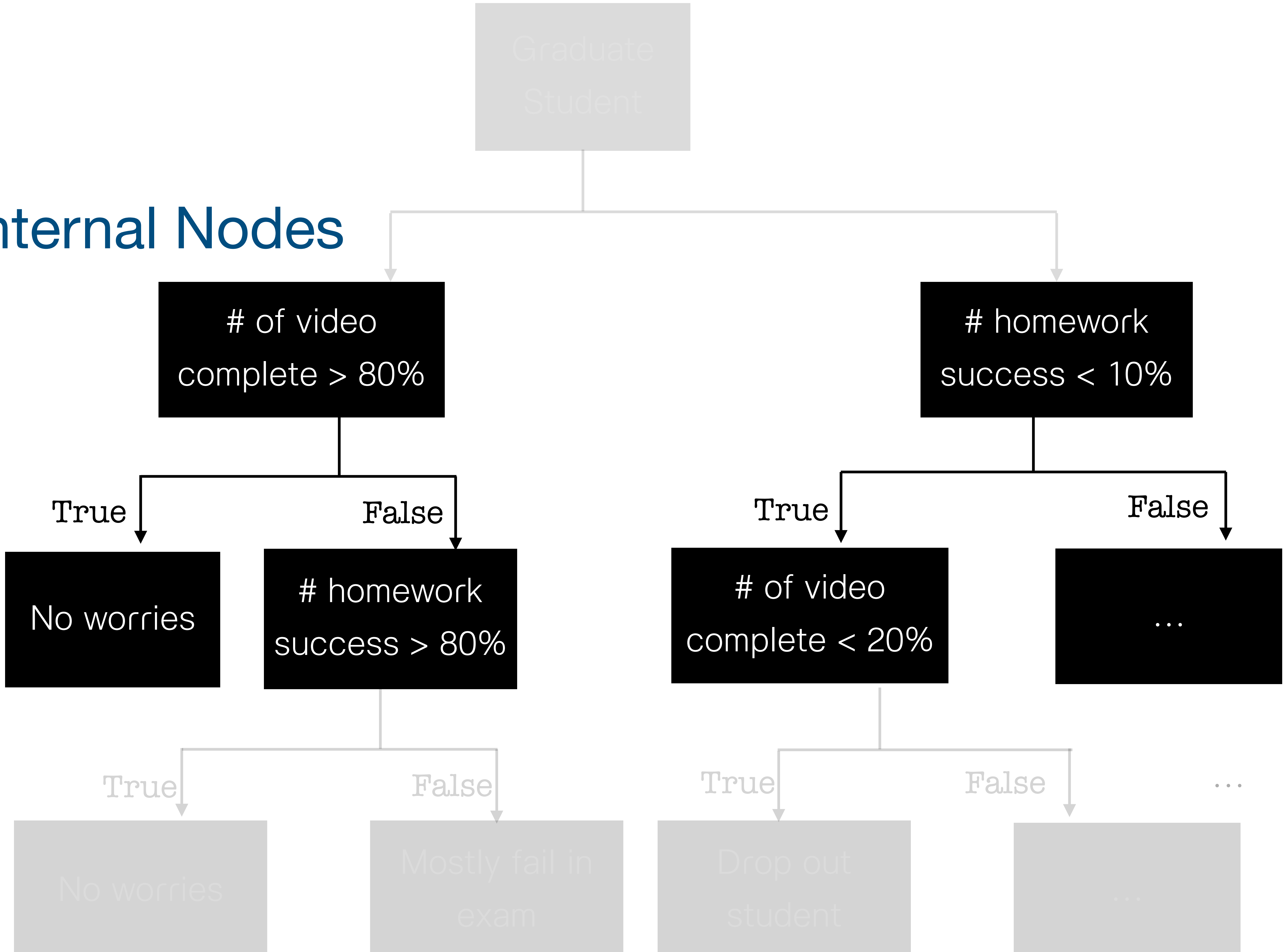

Graduate Student
# of video complete > 80%
homework success < 10%
True
False
No worries
homework success > 80%
True
False
# of video complete < 20%
...
True
False
No worries
Mostly fail in exam
True
False
Drop out student
...
...

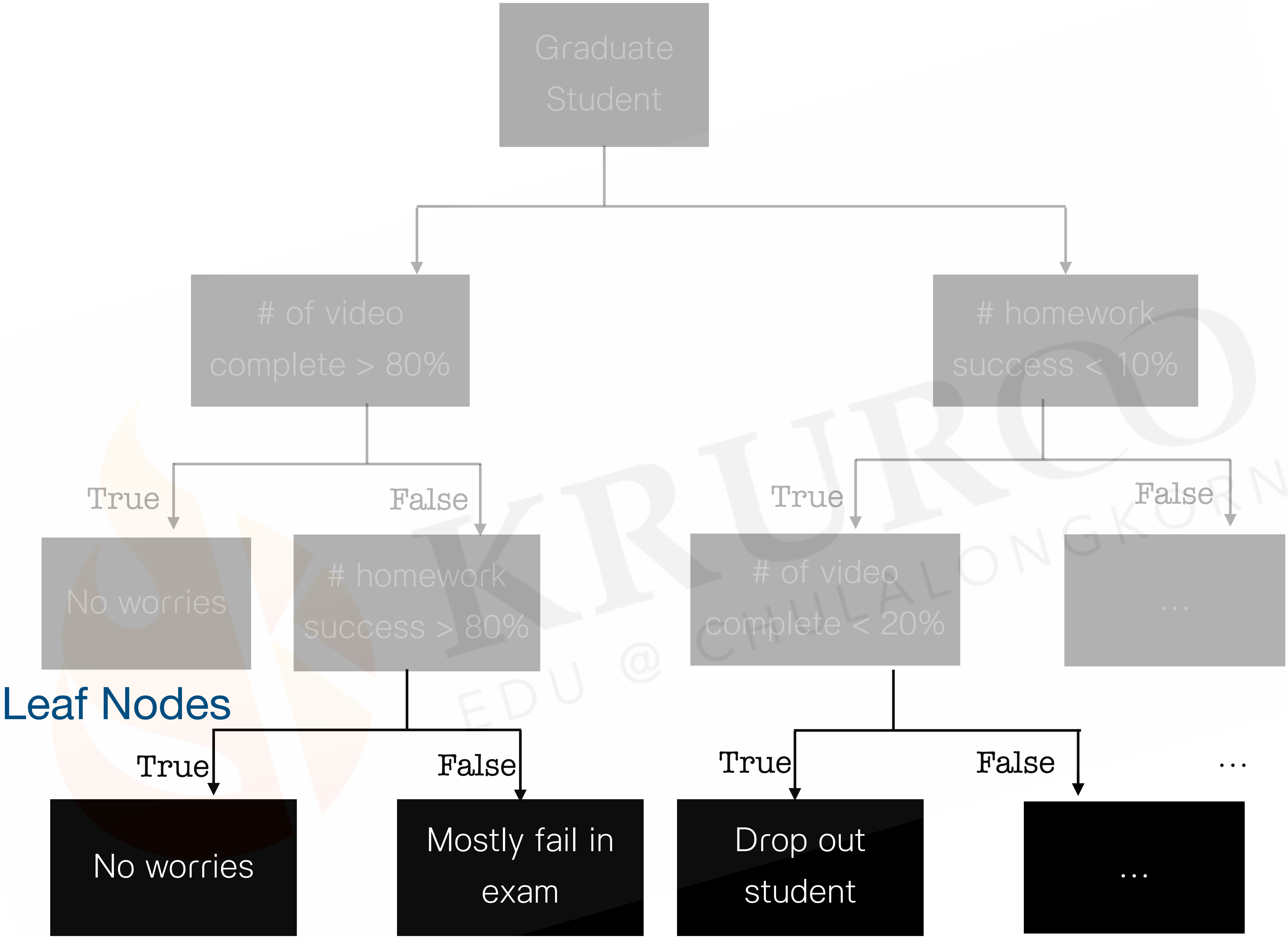
Root Node



Graduate Student
# of video complete > 80%
homework success < 10%
True
False
True
False
No worries
homework success > 80%
of video complete < 20%
…
True
False
True
False
…
No worries
Mostly fail in exam
Drop out student
…

Internal Nodes




Graduate Student
# of video complete > 80%
homework success < 10%
True
False
True
False
No worries
homework success > 80%
of video complete < 20%
...
Leaf Nodes
True
False
True
False
...
No worries
Mostly fail in exam
Drop out student
...