데이터 과학 외전

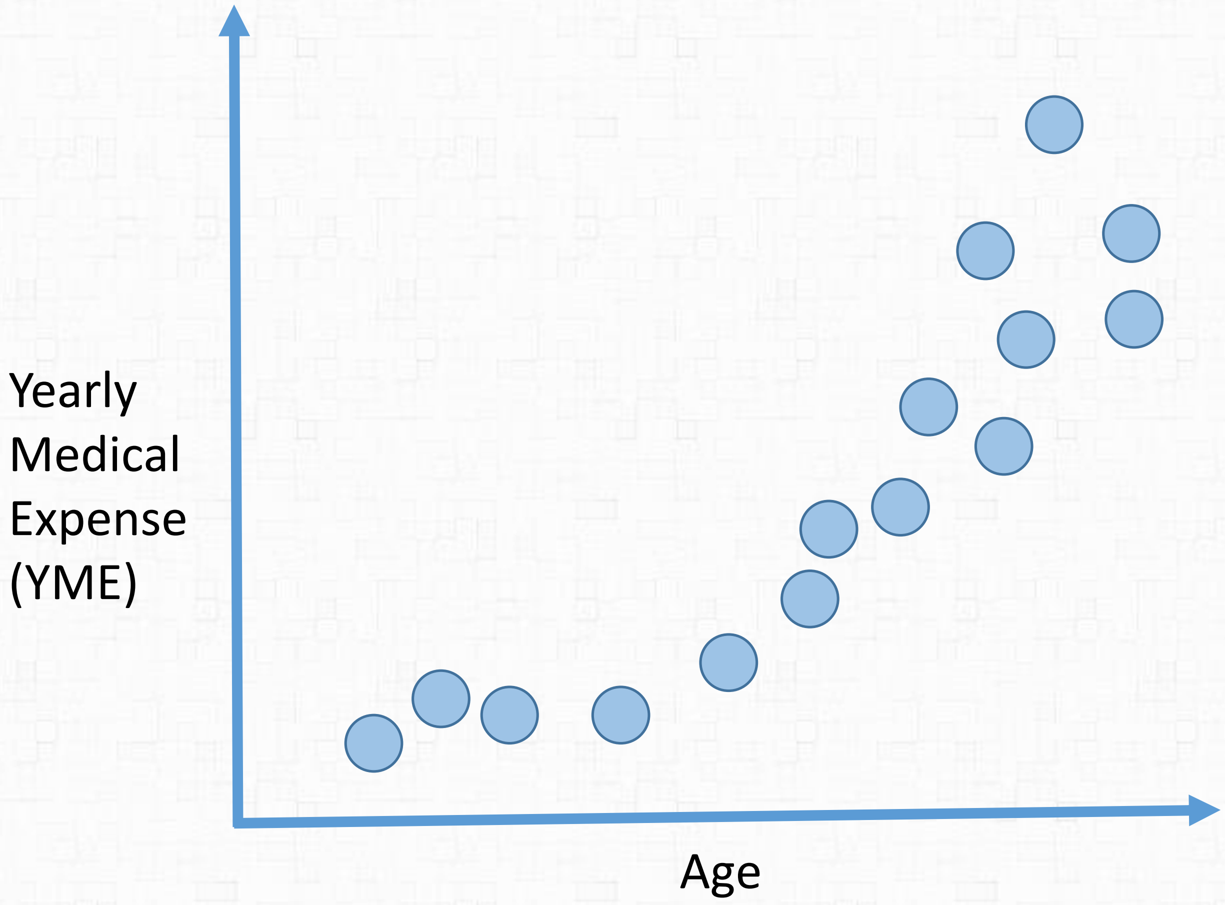
Day 4 – 앙상블 알고리즘

Contents

Bias And Variance

사람들의 나이와 연간의료비용을 데이터로 나타내었다

- 나이가 어리거나 젊을 때는 의료비용의 증가가 없거나 거의 미미하다
- 중년 이상이 되면서 의료비용이 서서히 증가하기 시작한다
- 노년 이상이되면서 의료비용의 증가 폭이 더욱 커진다

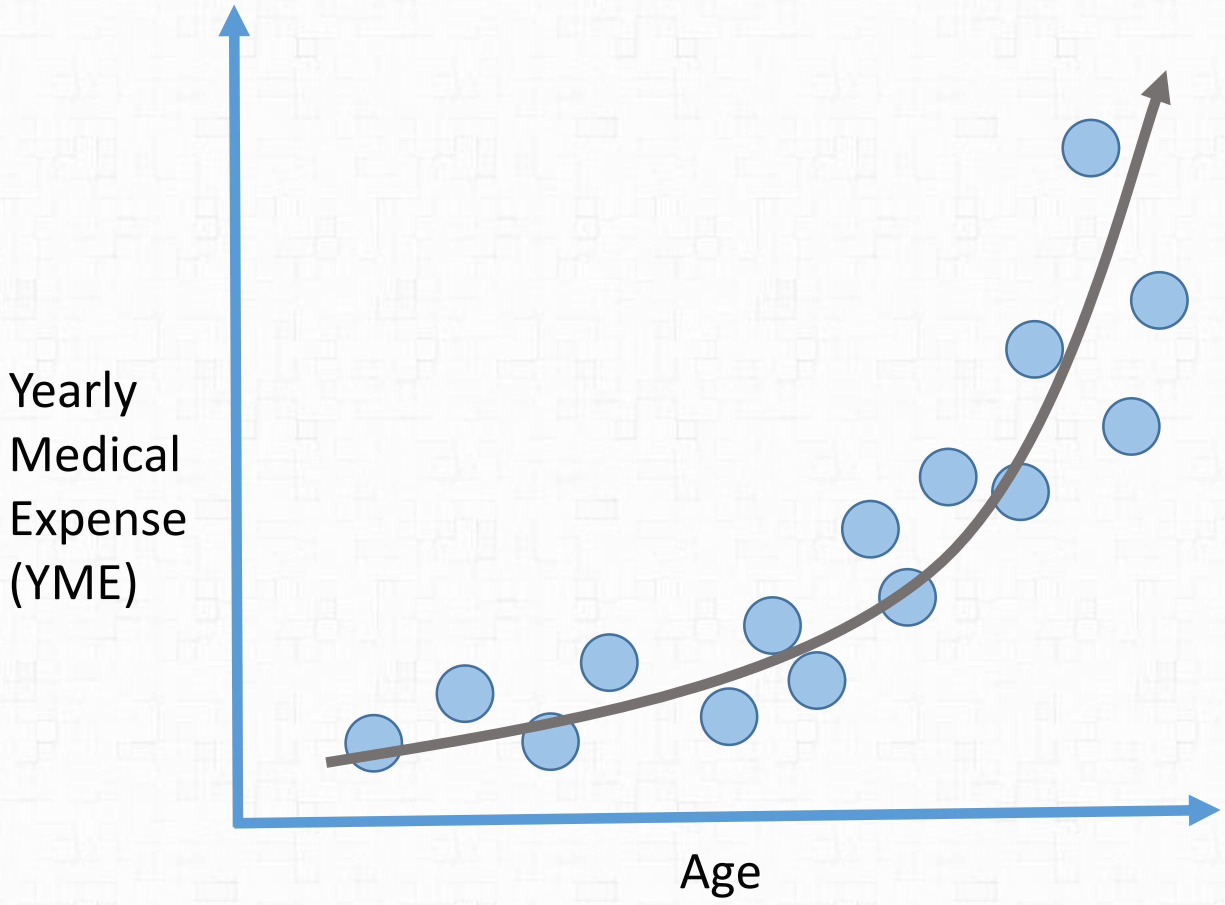


이상적으로는

의료비용과 나이의 관계를 결정하는
법칙 = 수식 = 모델
을 알고자 함

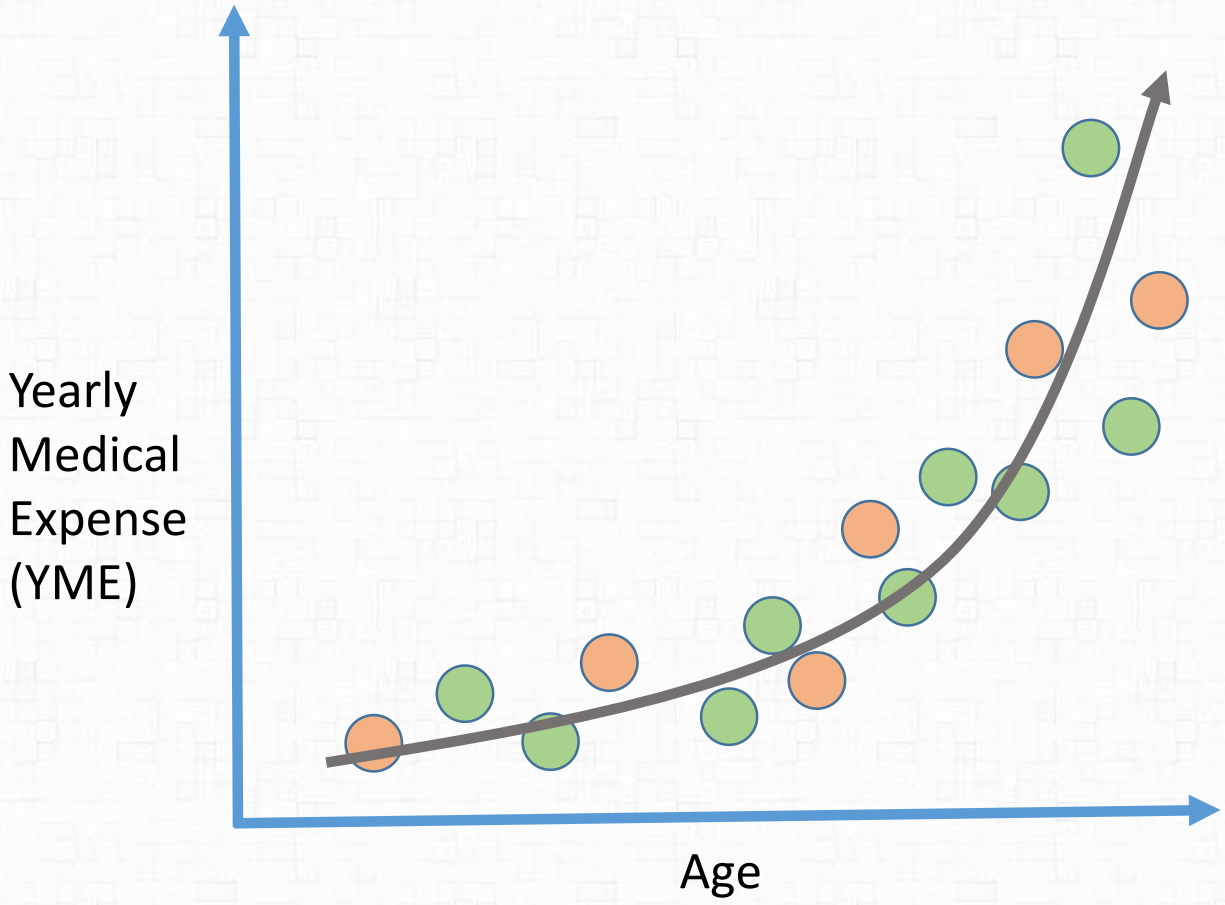
“True” relationship

나이와 의료비용의 관계를 sample data로
부터 추정하는 과정 = data modeling



데이터를 모델을 학습(추정)하기 위한 데이터(**학습데이터**)

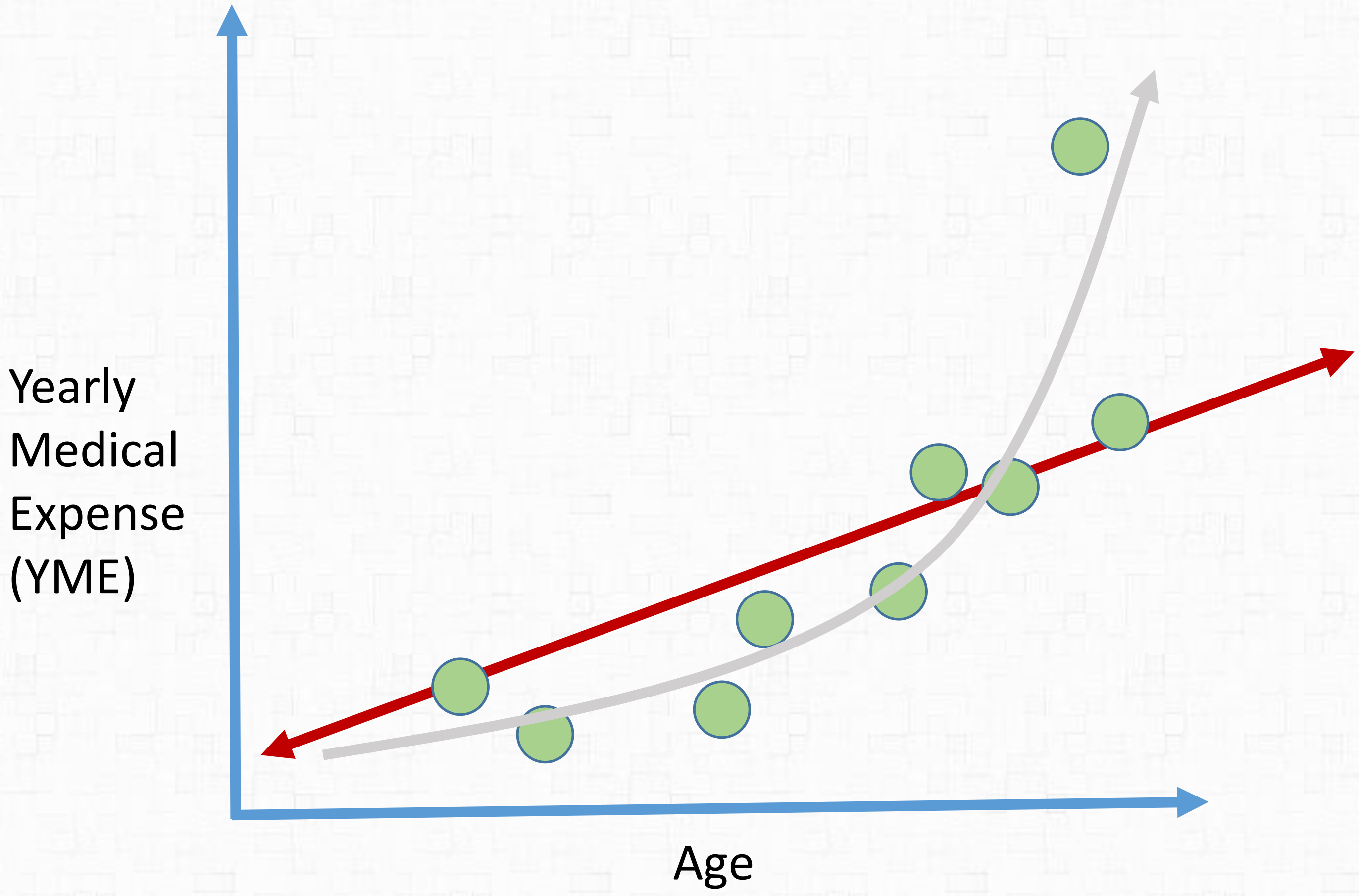
모델을 검증하기 위한 데이터(**테스트 데이터**)로 구분한다



First Try: Linear Regression

$$y = \alpha_0 + \alpha_1 x$$

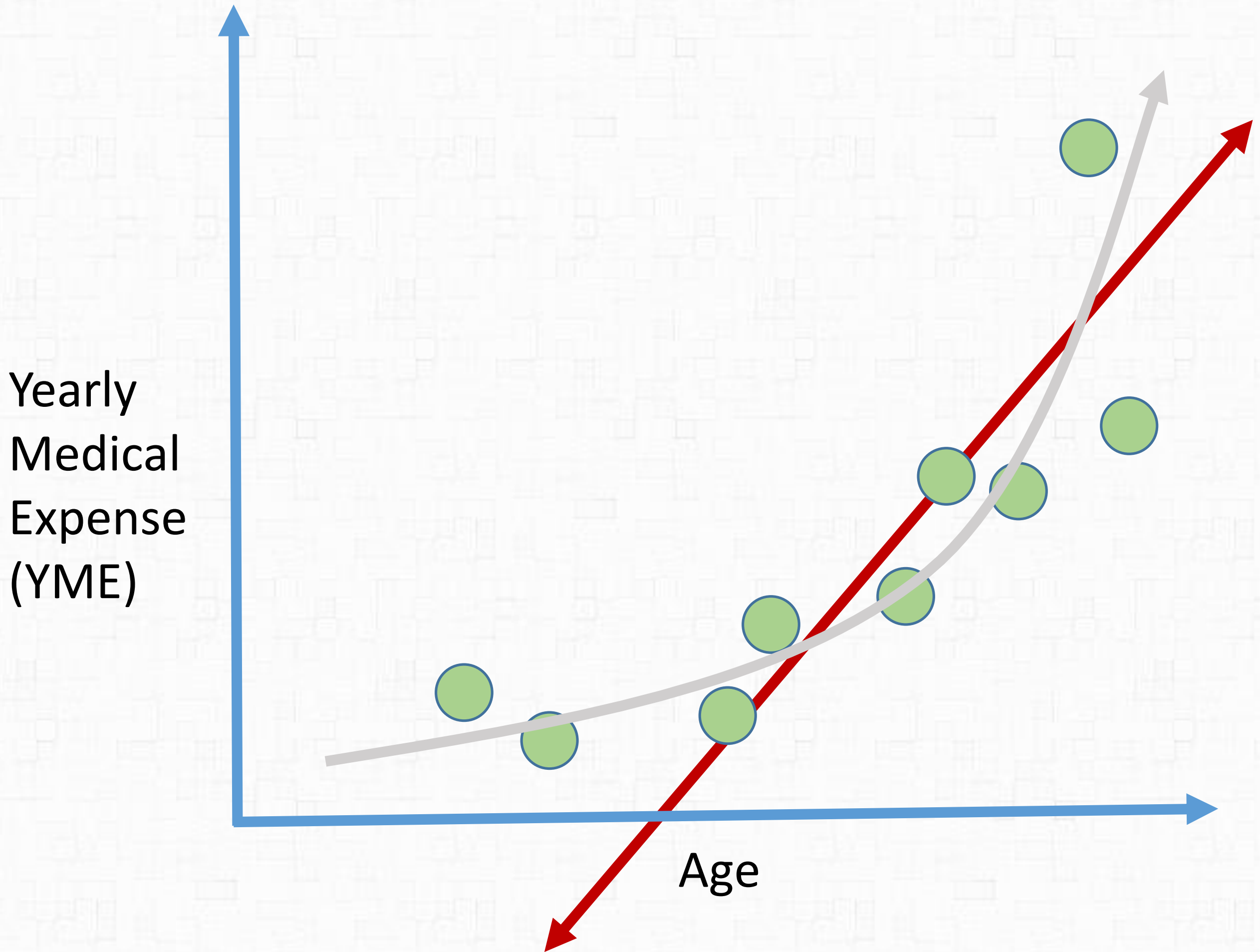
y : YME
x : Age



First Try: Linear Regression

$$y = \alpha_0 + \alpha_1 x$$

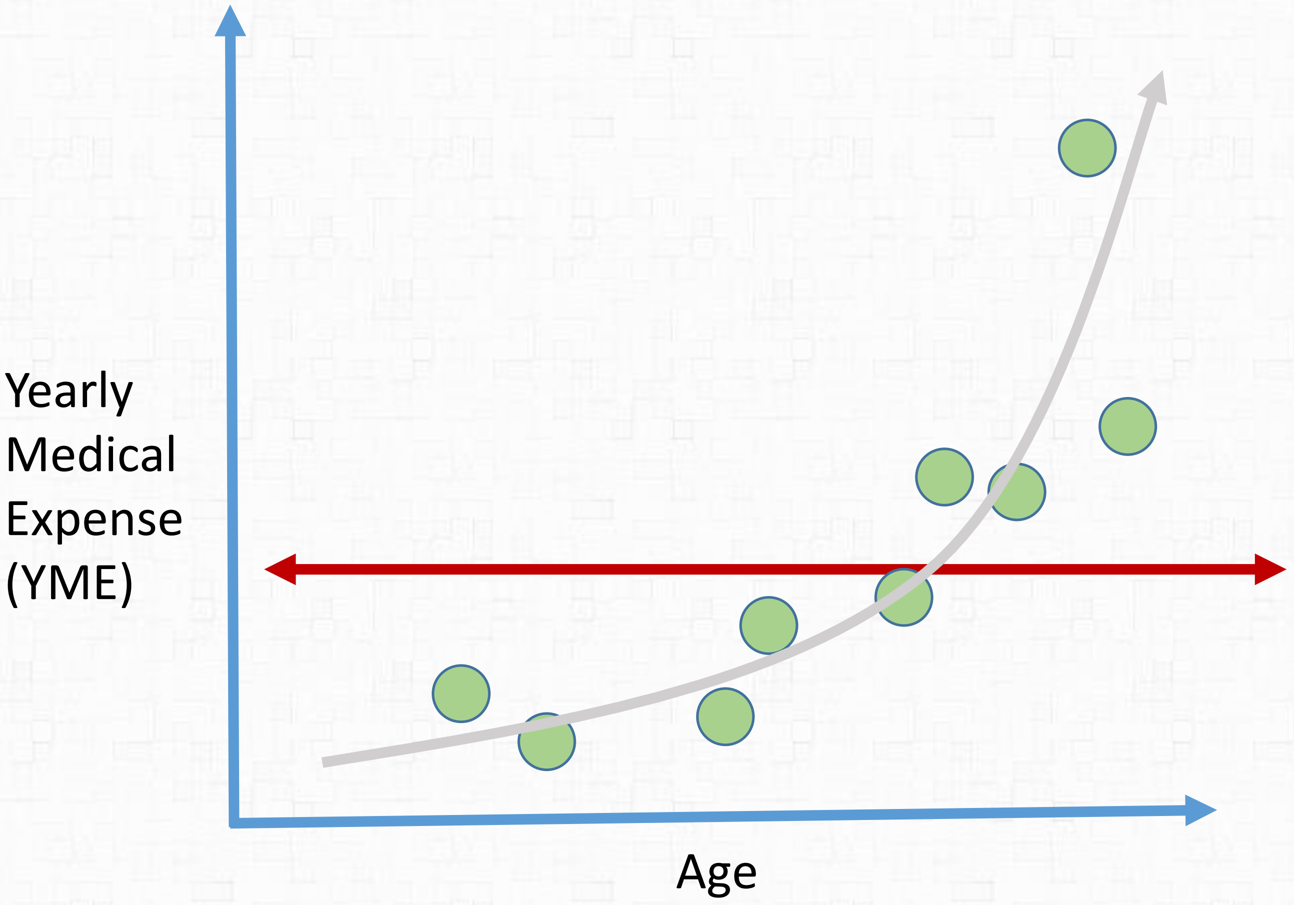
y : YME
x : Age



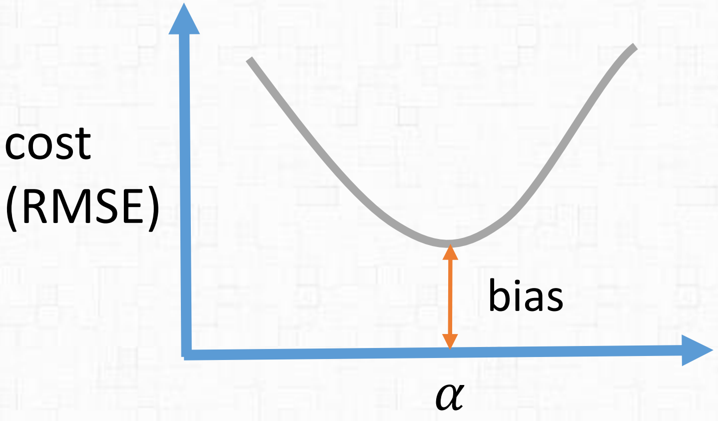
First Try: Linear Regression

$$y = \alpha_0 + \alpha_1 x$$

y : YME
x : Age

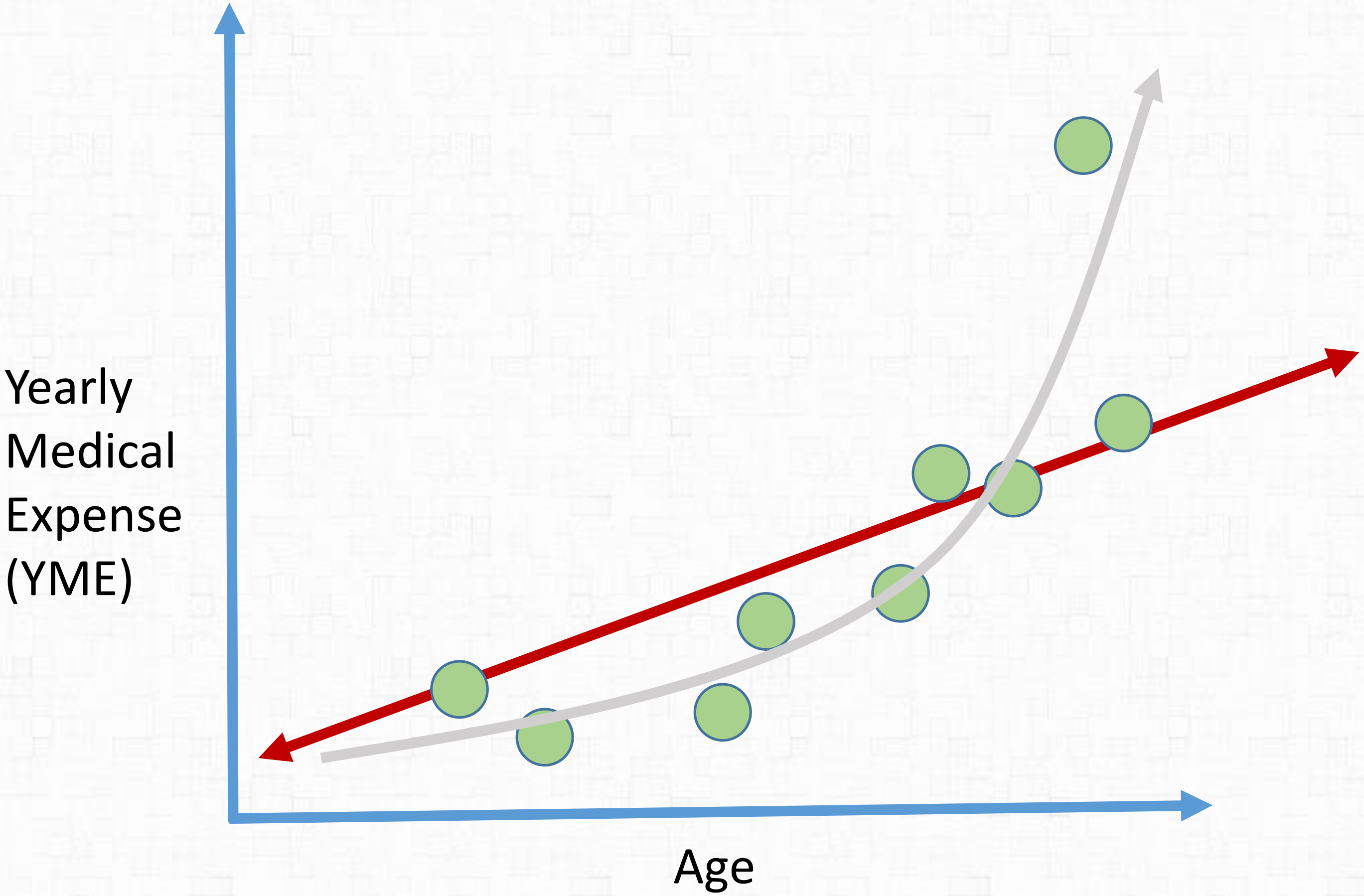


First Try: Linear Regression



$$y = \alpha_0 + \alpha_1 x$$

y : YME
x : Age



Linear Regression
will never capture
the "true relationship"

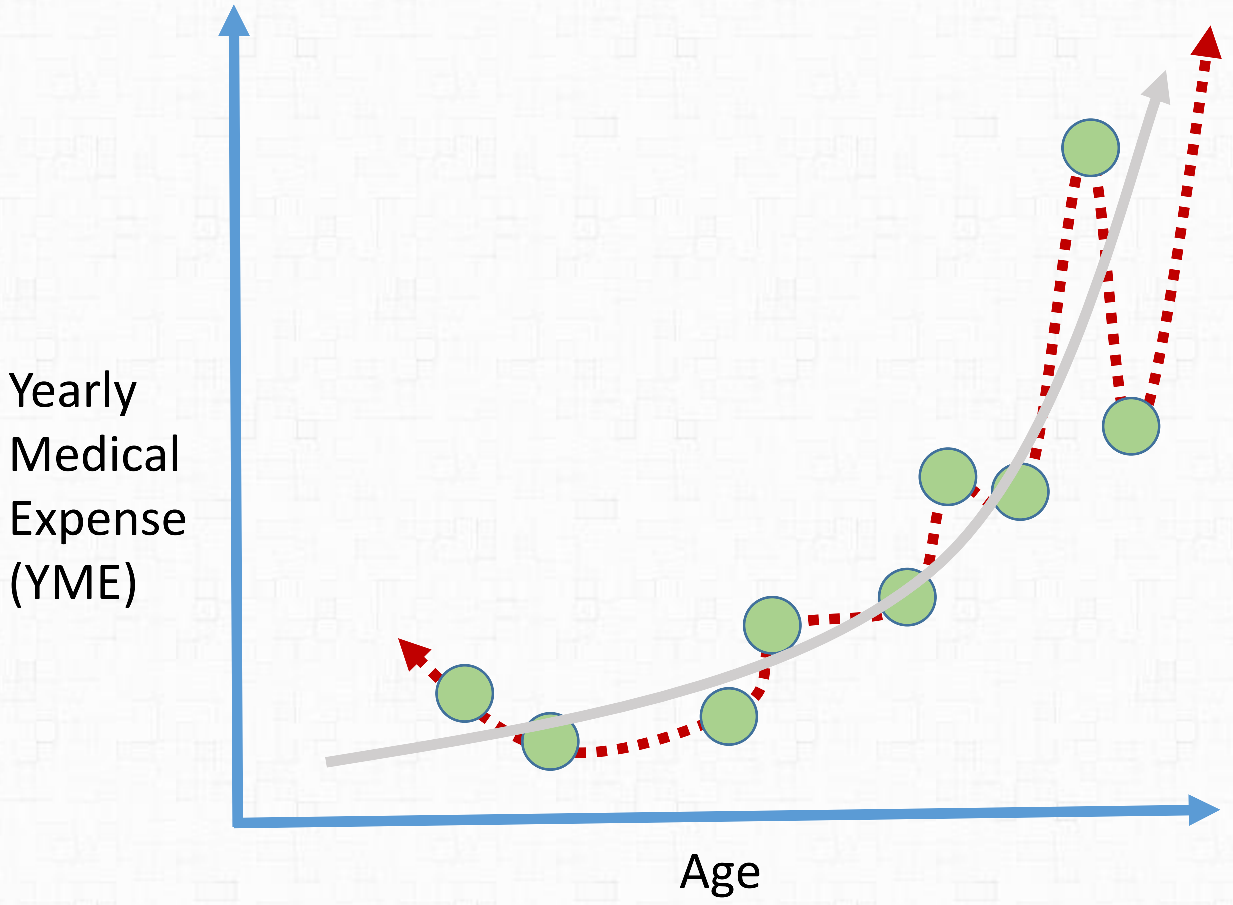
= Bias

Second Try: Polynomial Regression

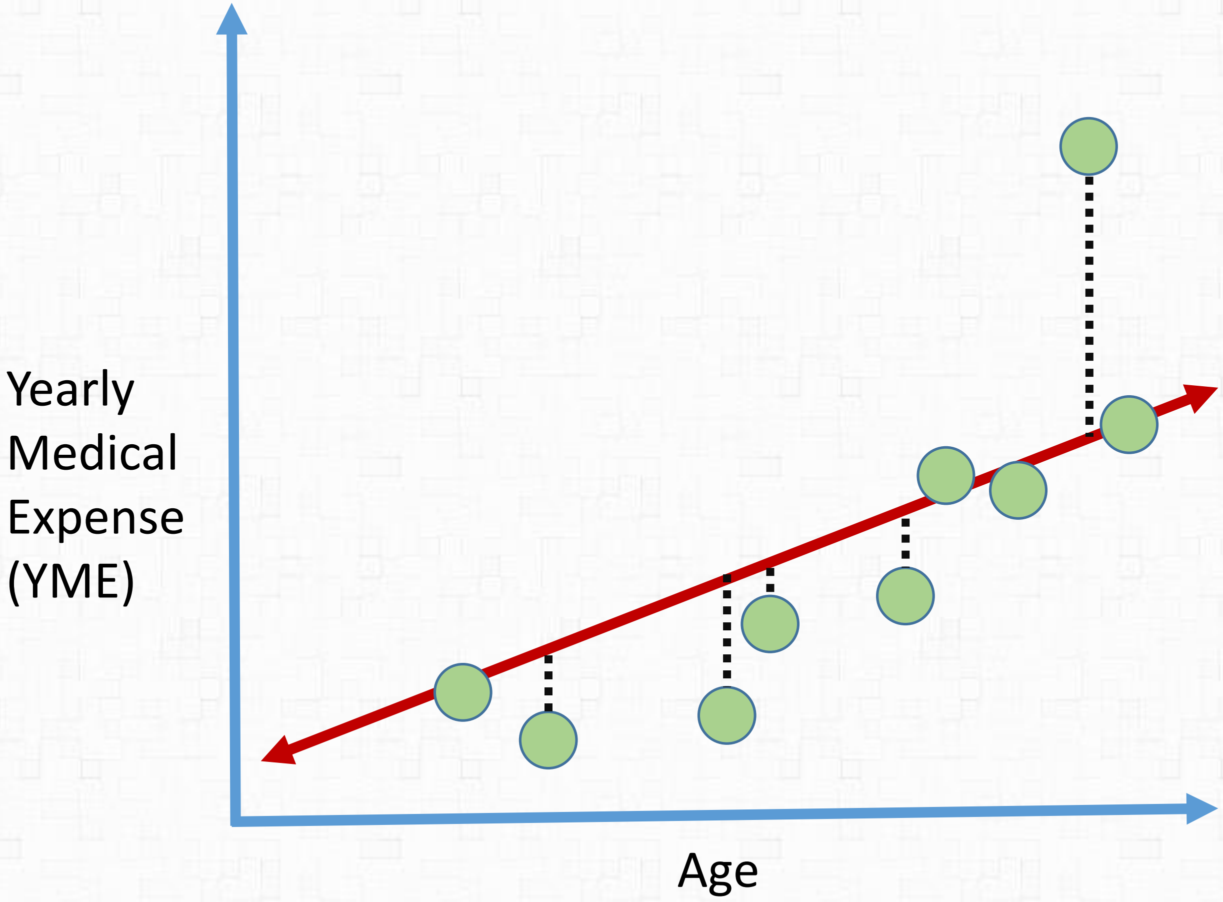
$$y = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3 + \alpha_4 x^4 + \alpha_5 x^5$$

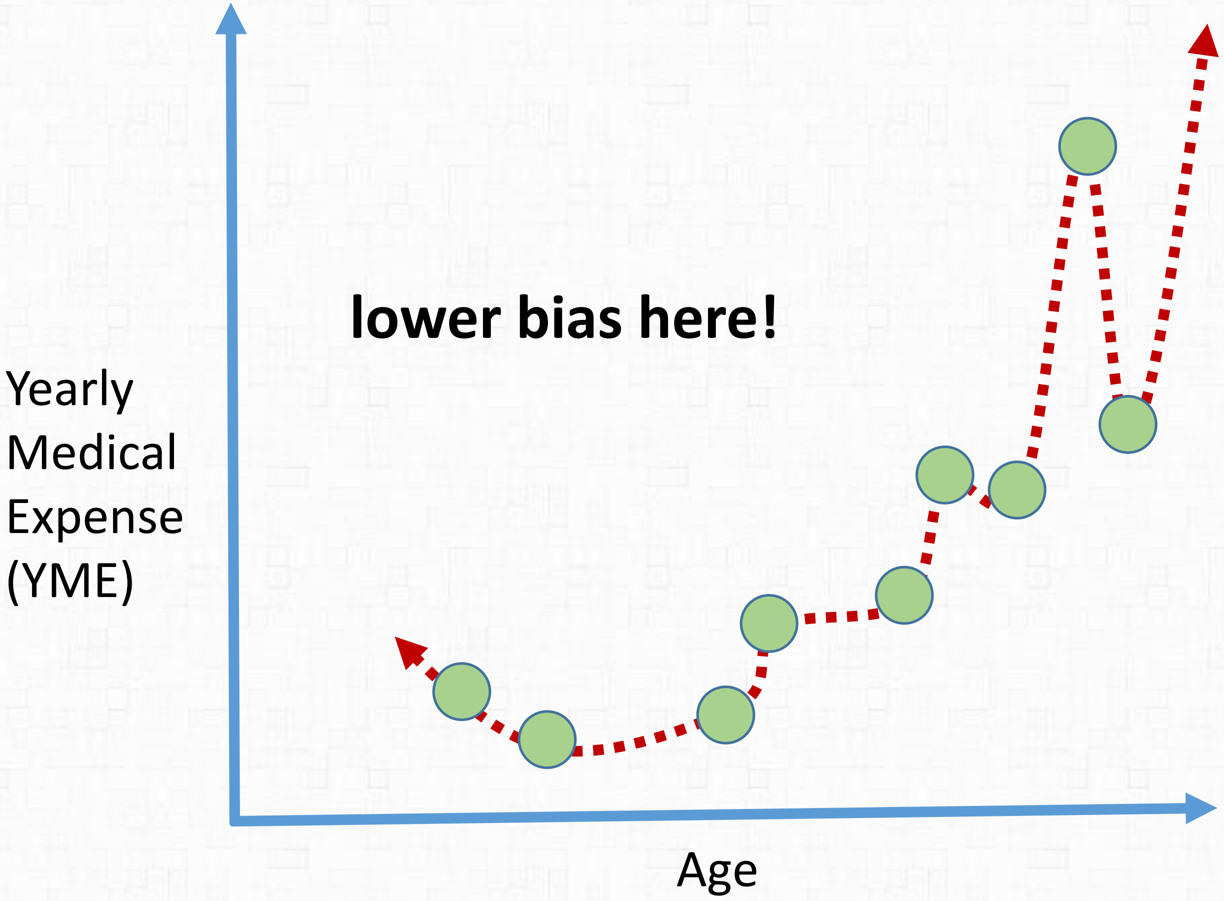
y : YME
x : Age

Model is flexible to capture the non-linearity



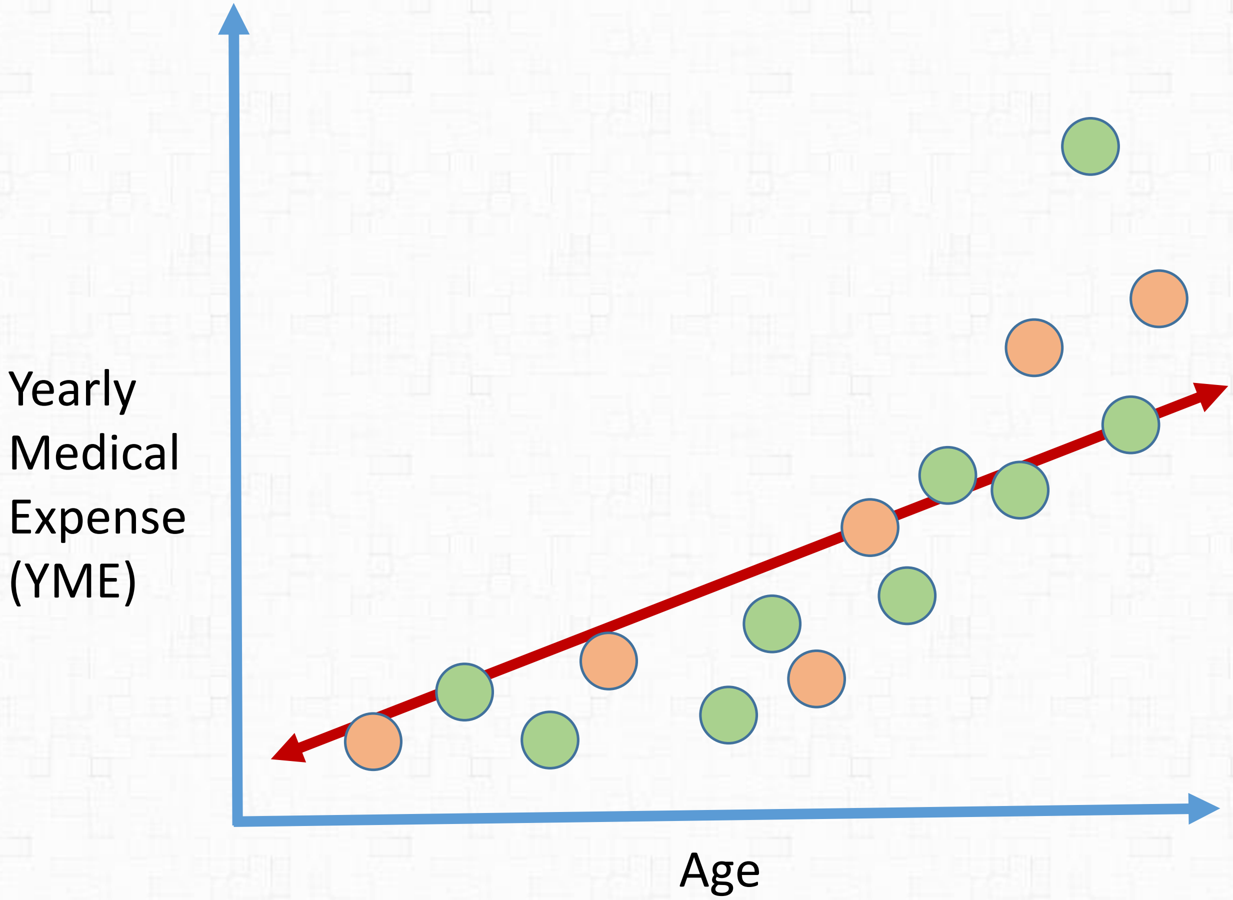
Second Try: Polynomial Regression

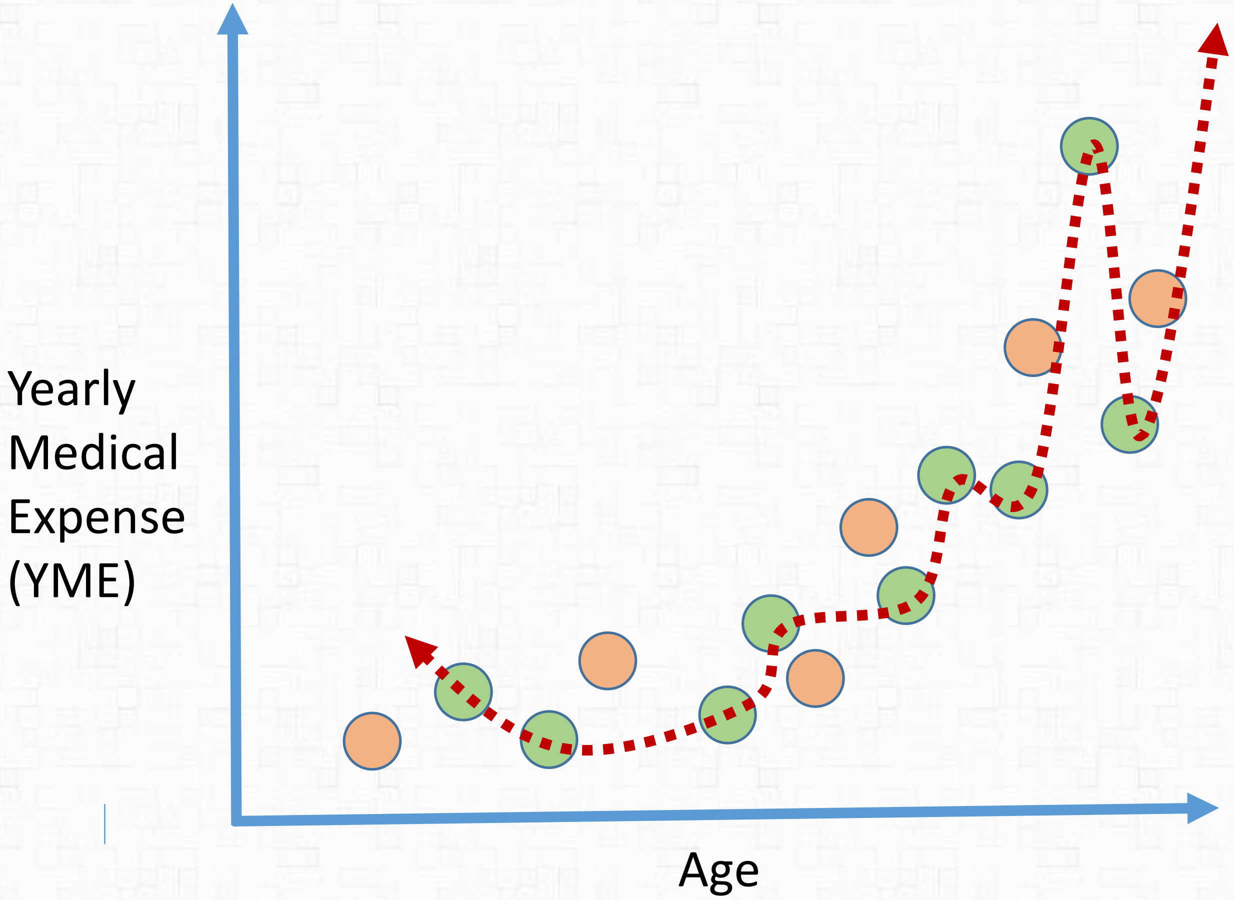




Second Try: Polynomial Regression

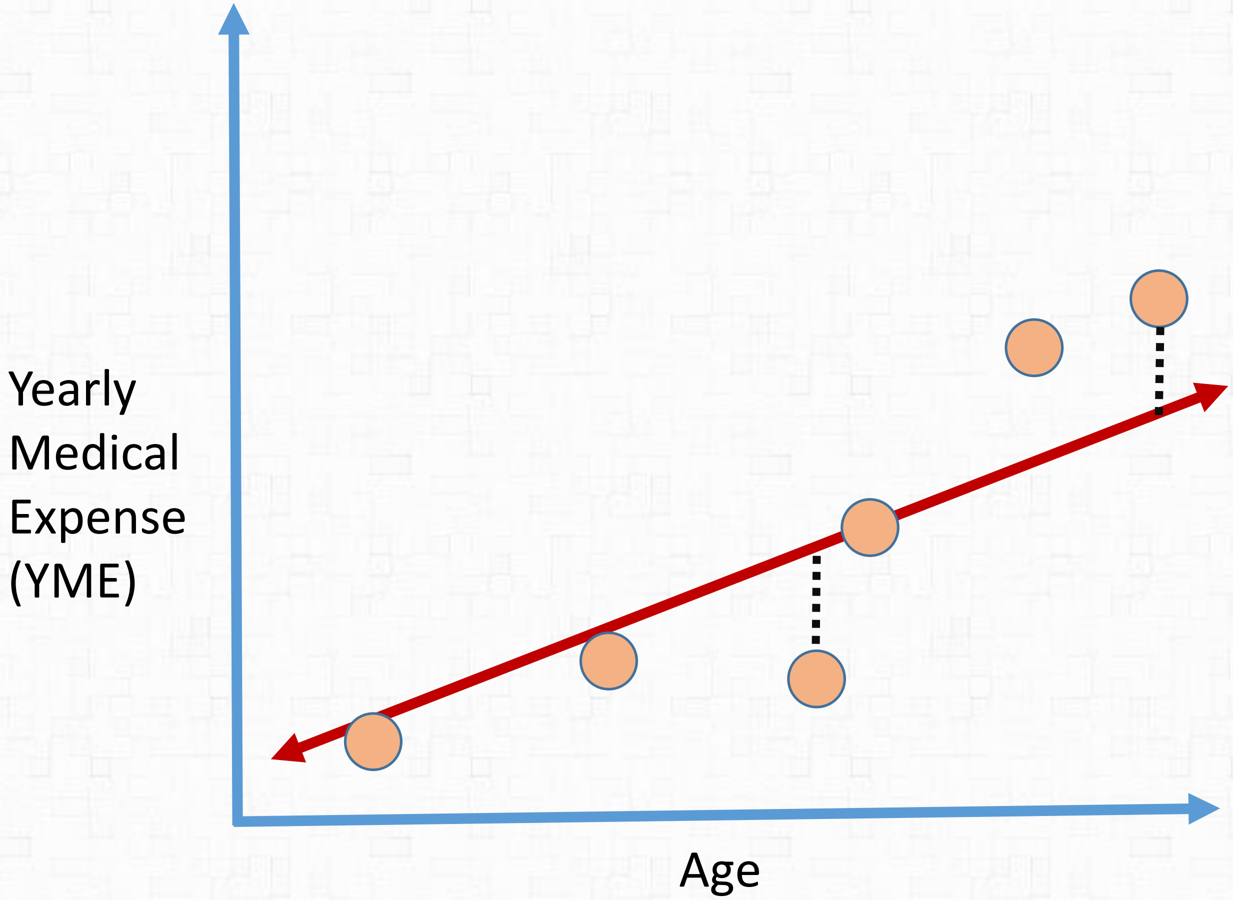
what about training set

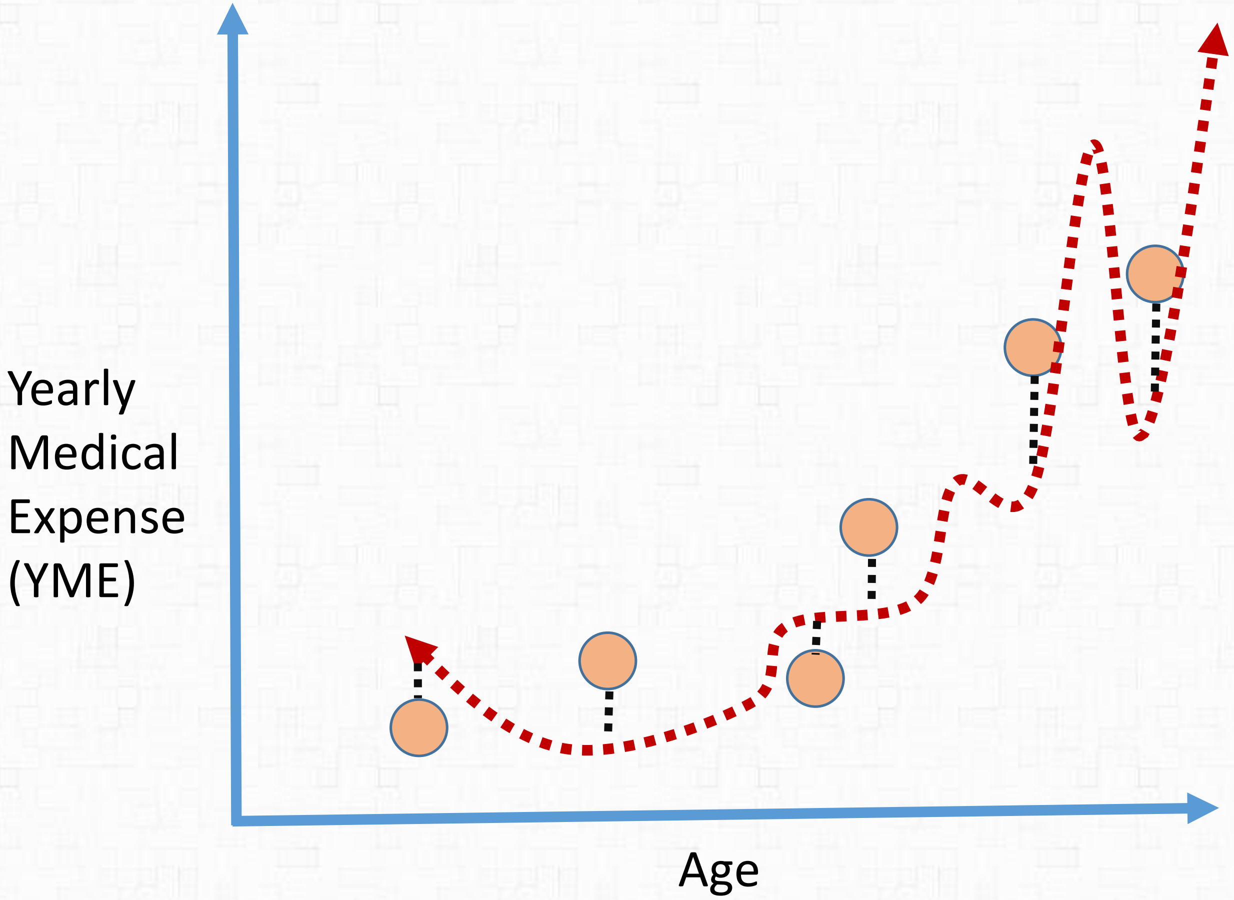




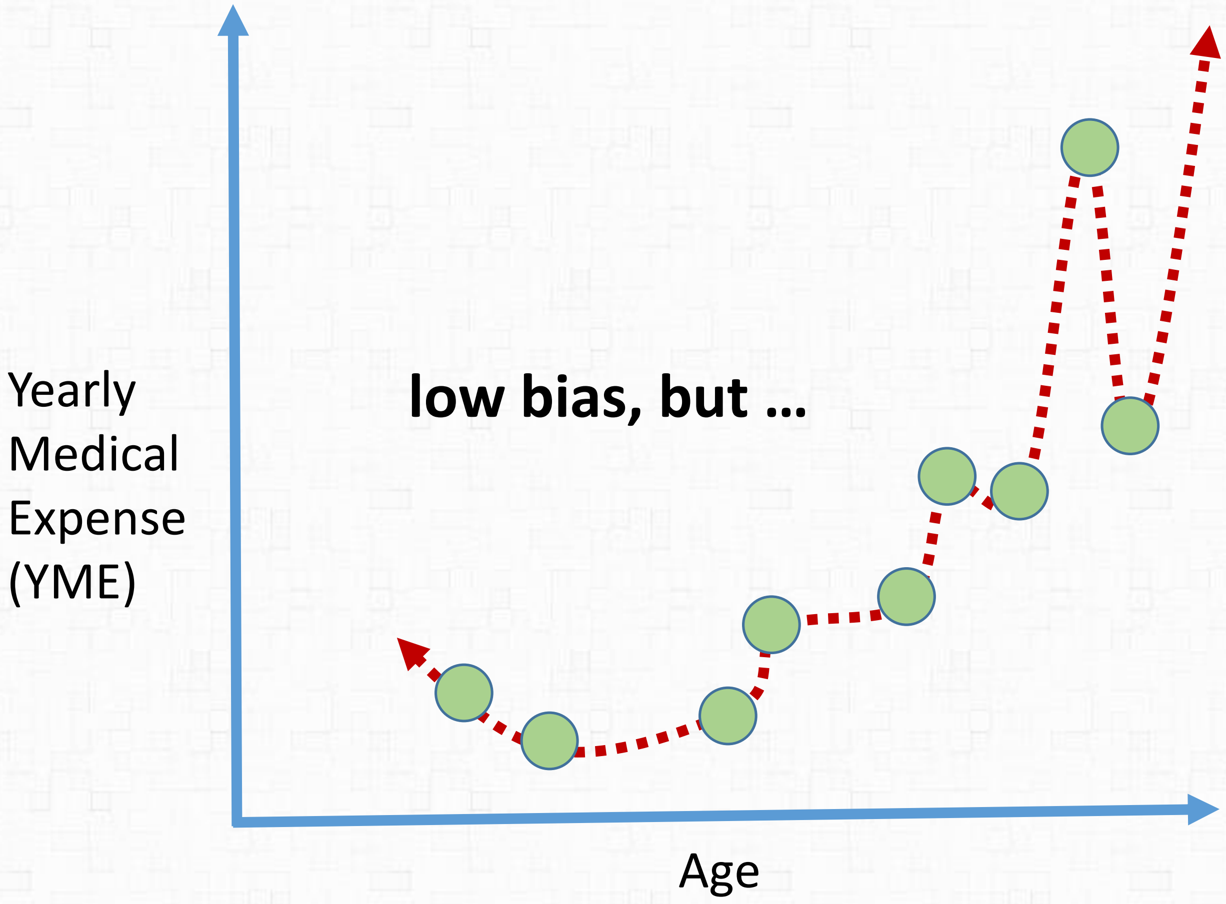
Second Try: Polynomial Regression

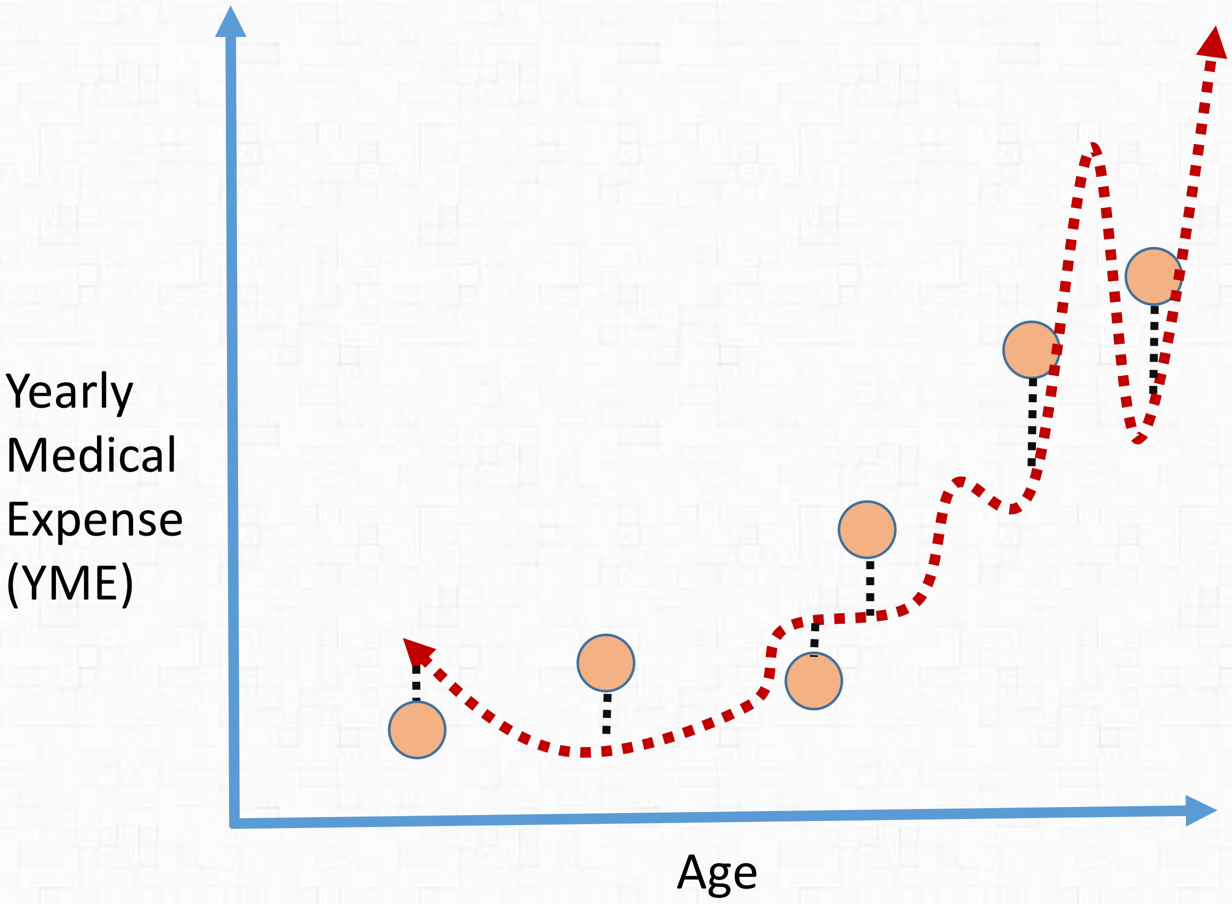
what about training set





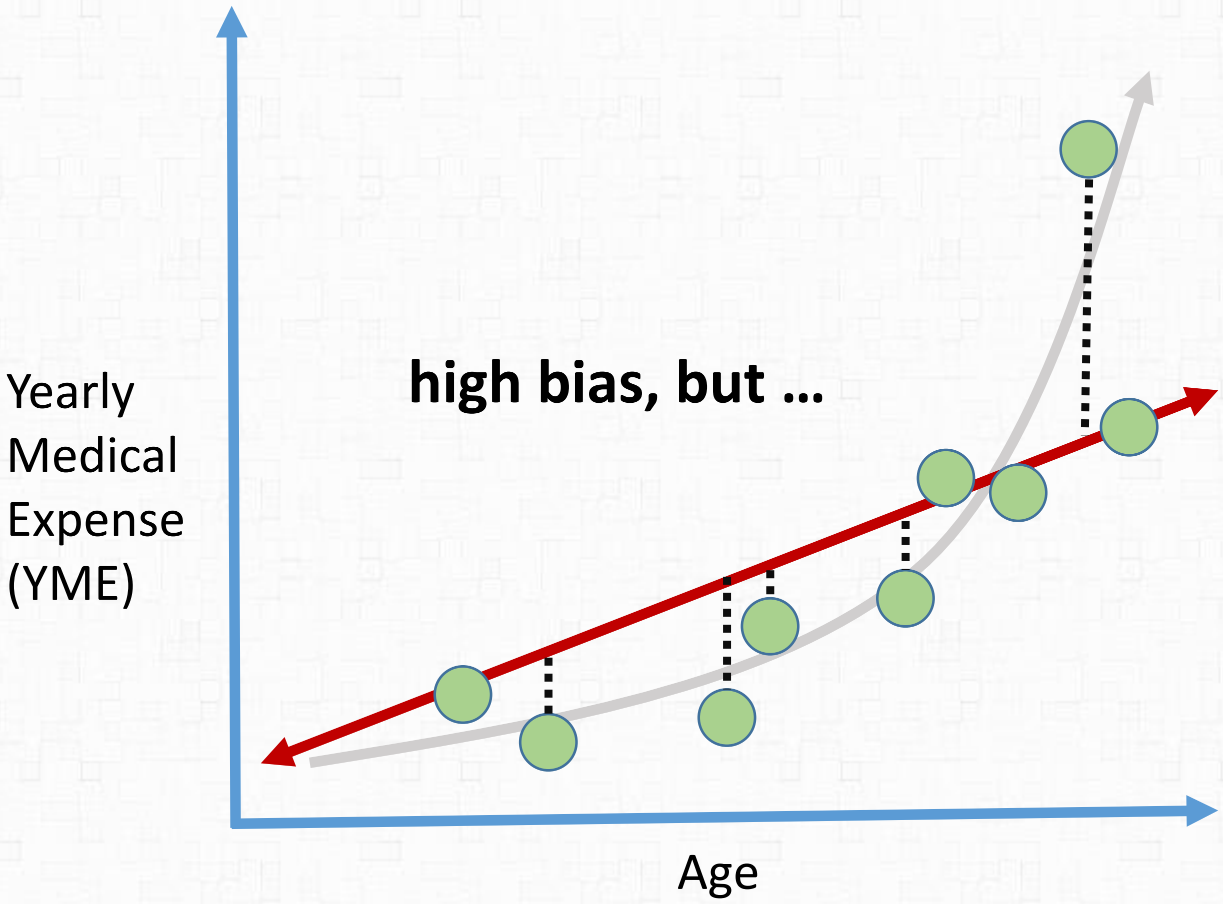
The difference in fit between two datasets is call **Variance**

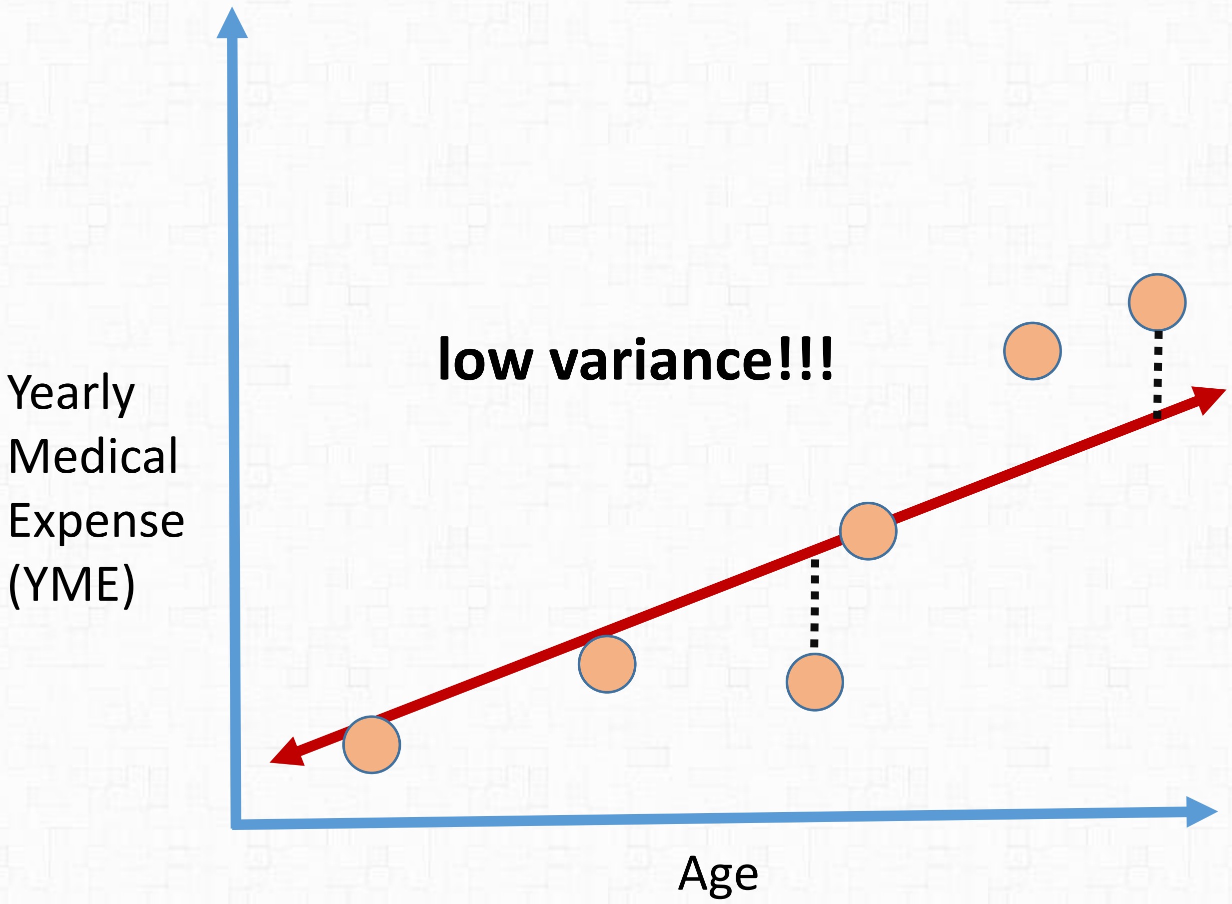




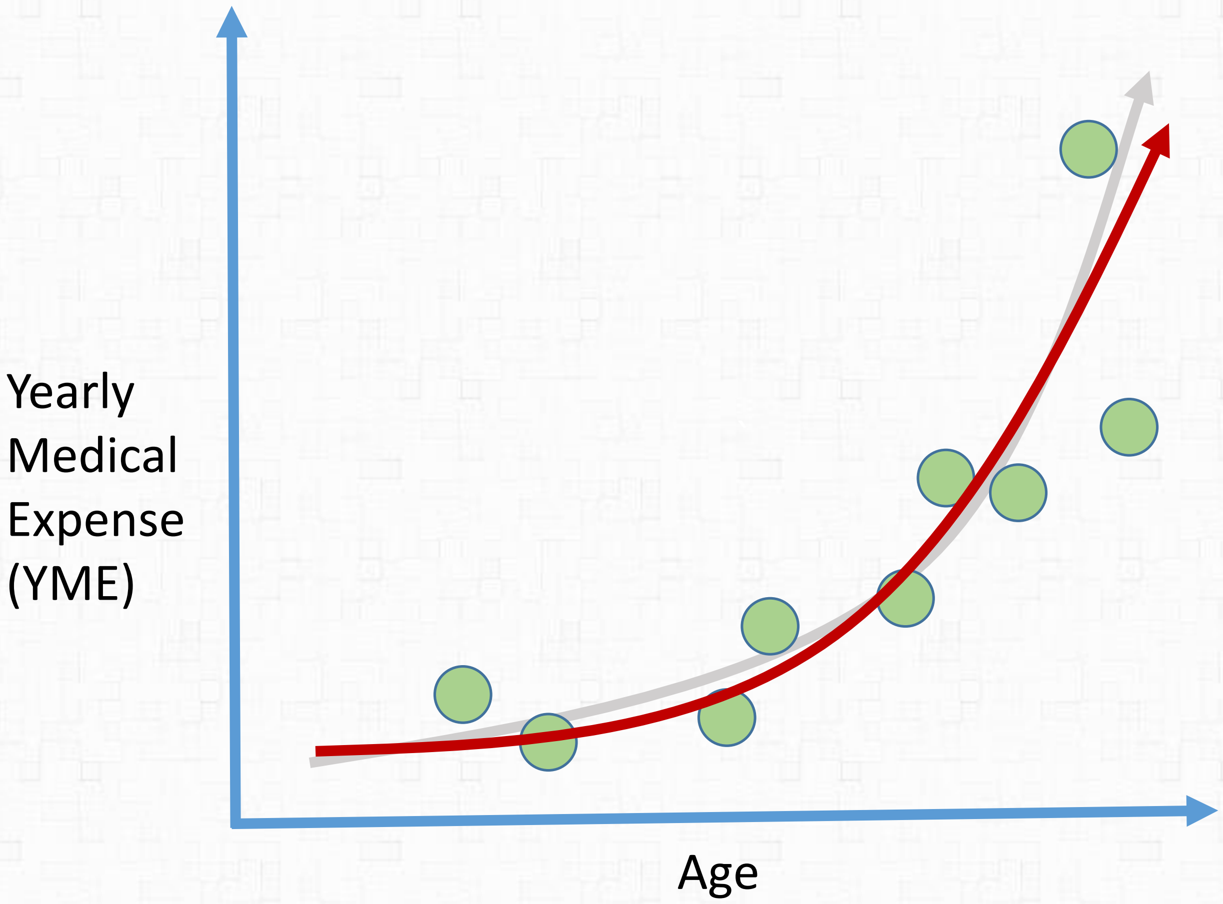
high variance!!

low variance model consistently give a good prediction (even though it is not a great prediction)

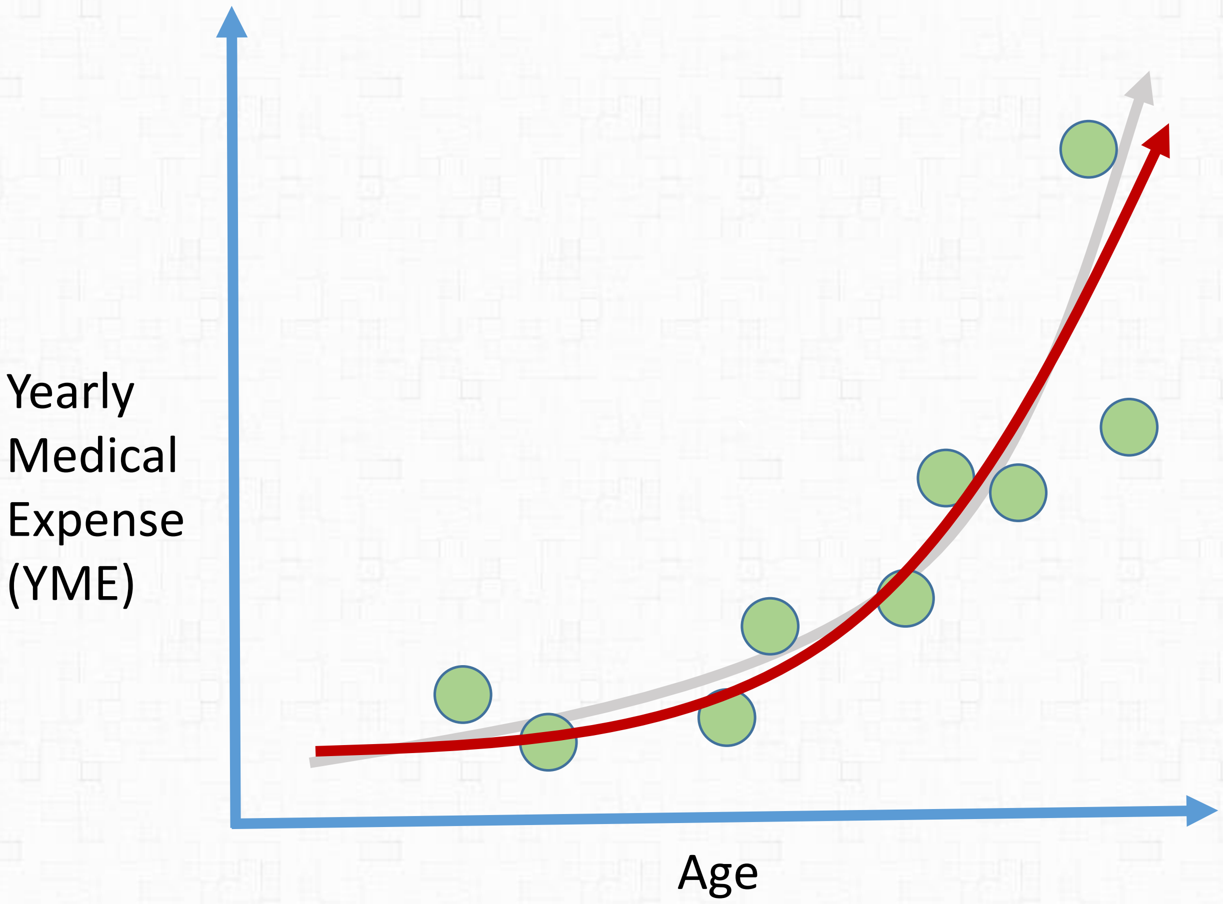


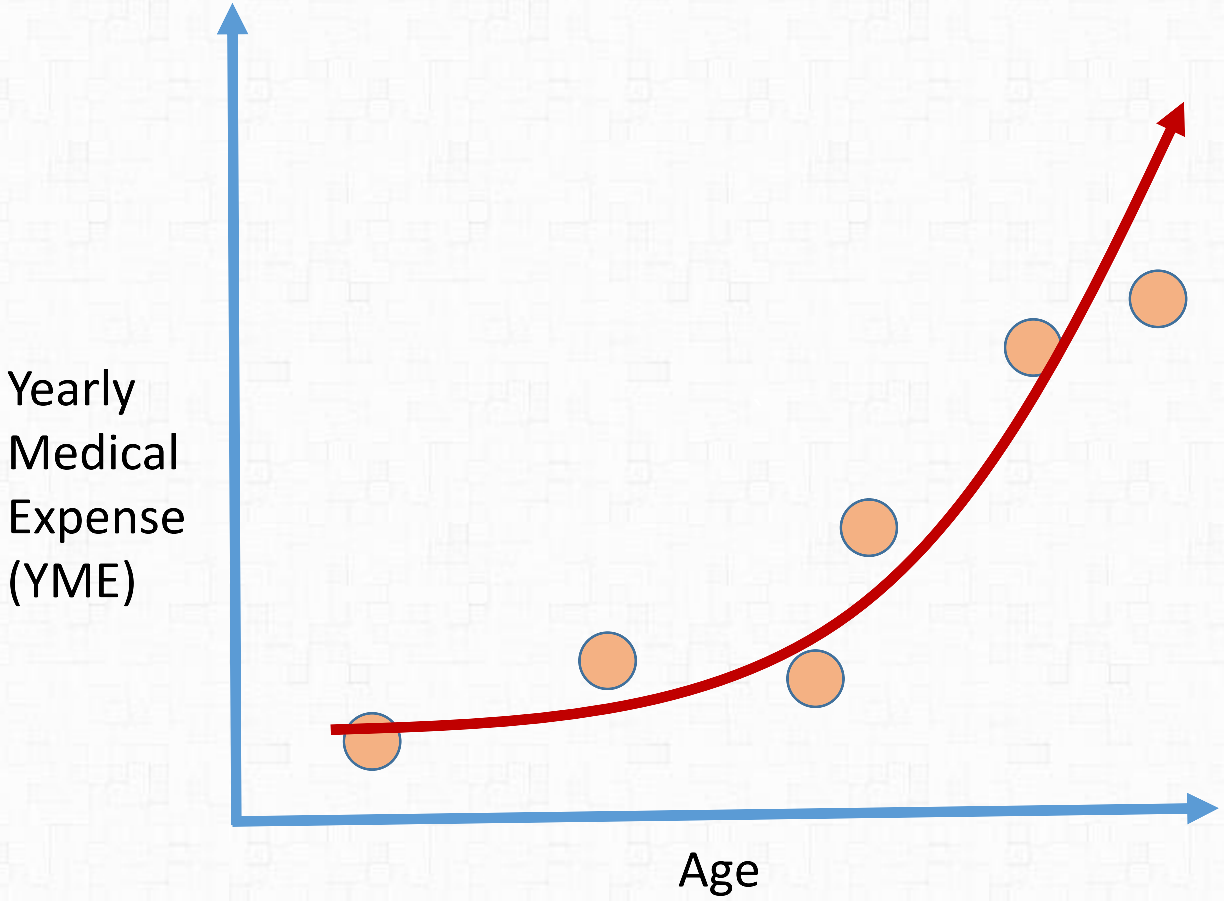


the ideal model has **low bias,**
which accurately model the true relationship



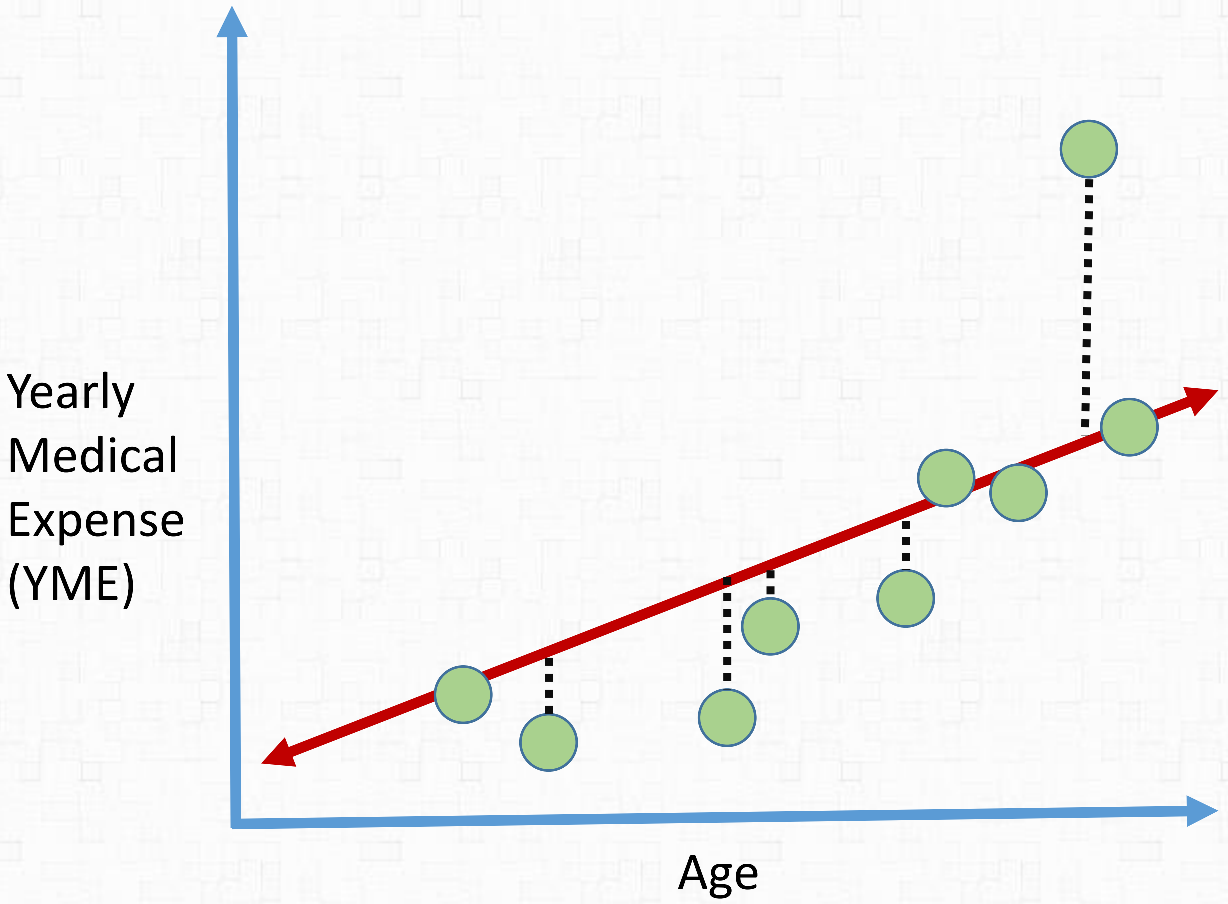
... ant it has **low variance**,
by making consistence predictions across different datasets

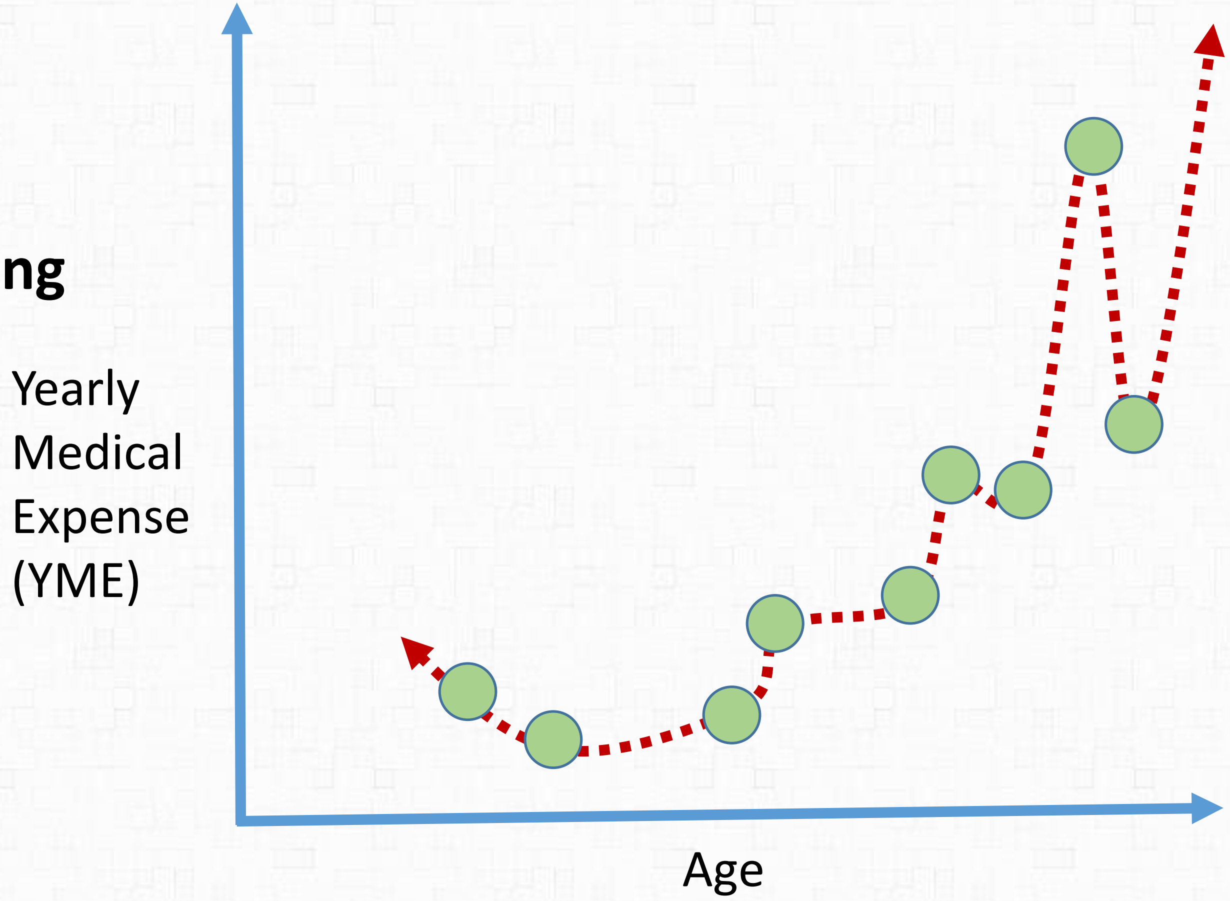




The key is to find a balance between a simple model and a complex model

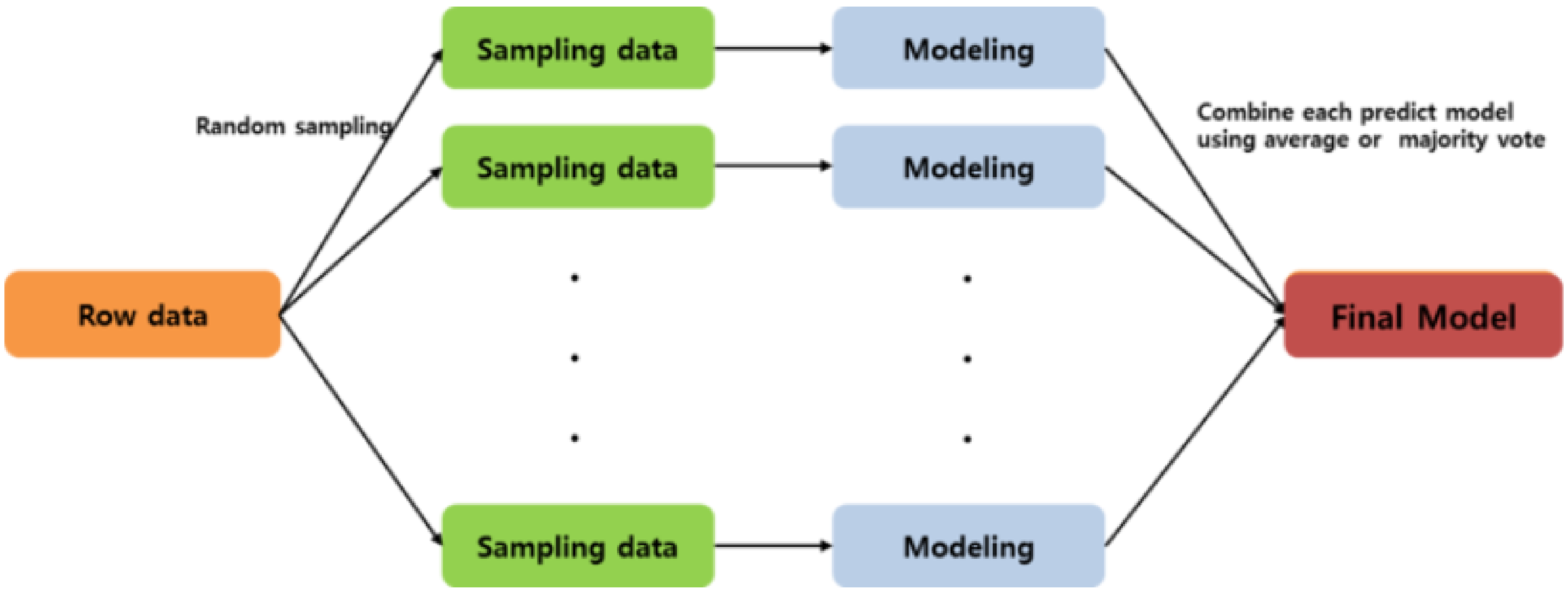
The commonly used methods are: regularization, boosting and bagging



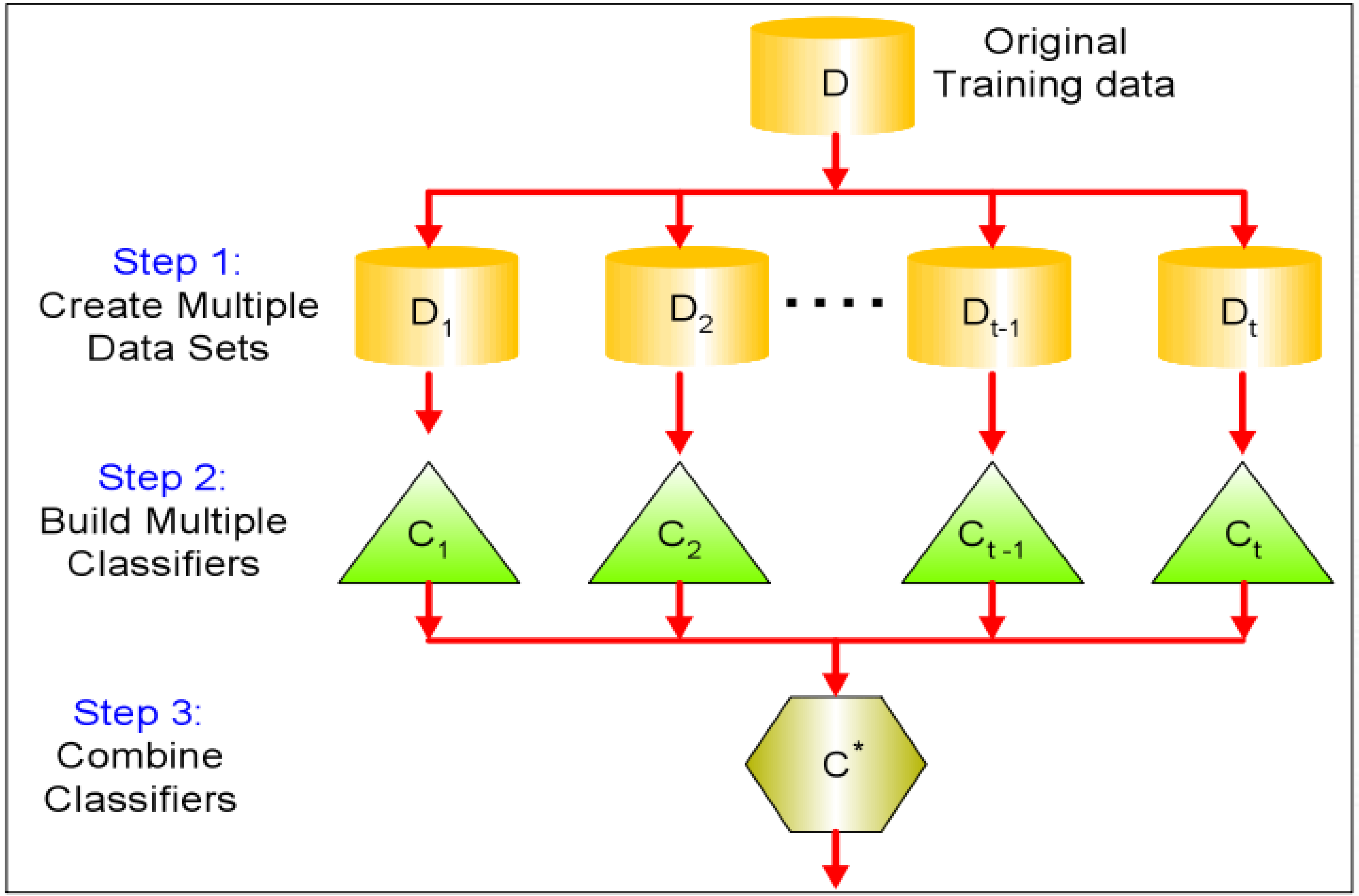


Bagging and Boosting

Bagging


Row data
Random sampling
Sampling data
Sampling data
Sampling data
Modeling
Modeling
Modeling
Combine each predict model
using average or majority vote
Final Model

Bagging



D
Original Training data
Step 1:
Create Multiple Data Sets
D_1
D_2
D_{t-1}
D_t
Step 2:
Build Multiple Classifiers
C_1
C_2
C_{t-1}
C_t
Step 3:
Combine Classifiers
C^*

Example of Bagging

- Sampling with replacement

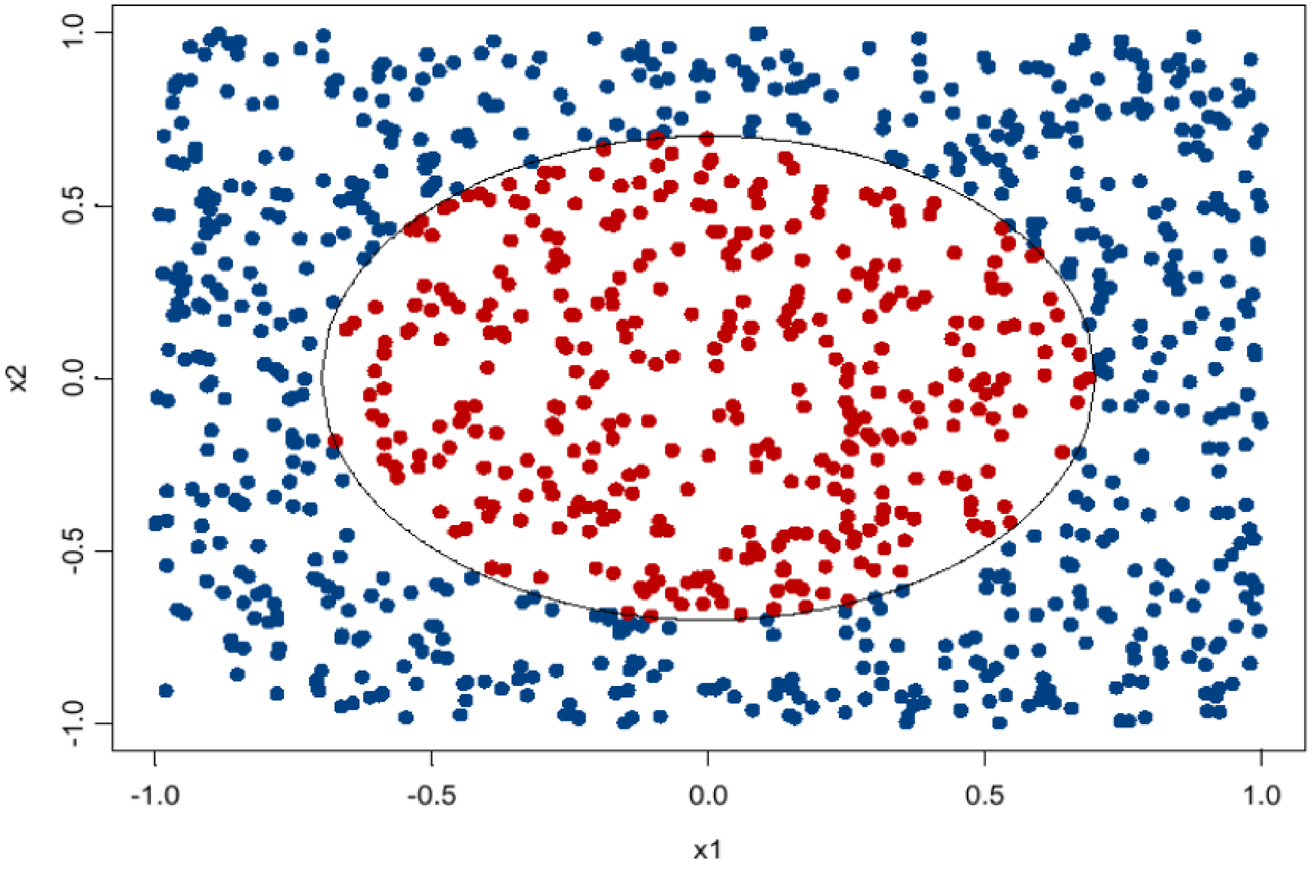
Training Data

Data ID

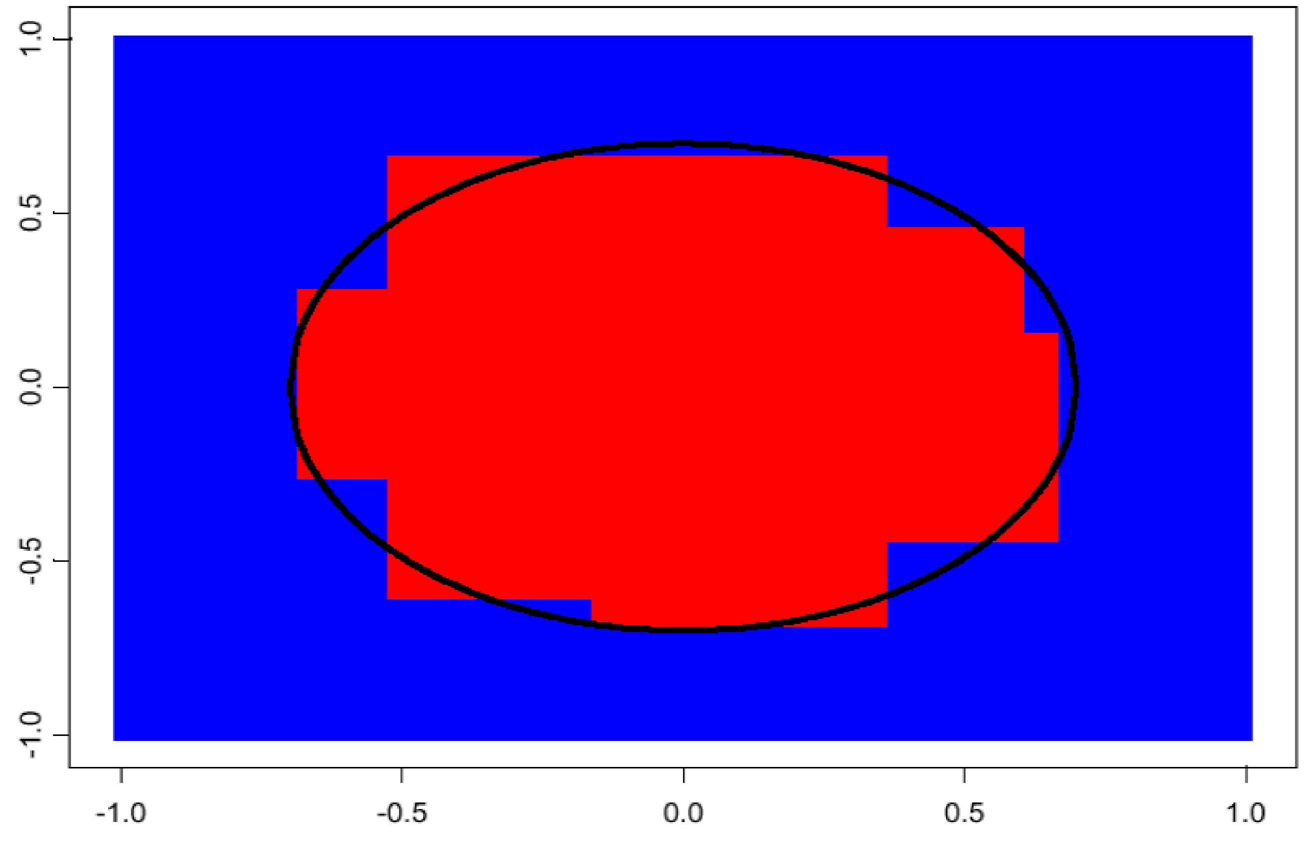
Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Build classifier on each bootstrap sample
- Each data point has probability $(1 – 1/n)^n$ of being selected as test data
- Training data = $1- (1 – 1/n)^n$ of the original data

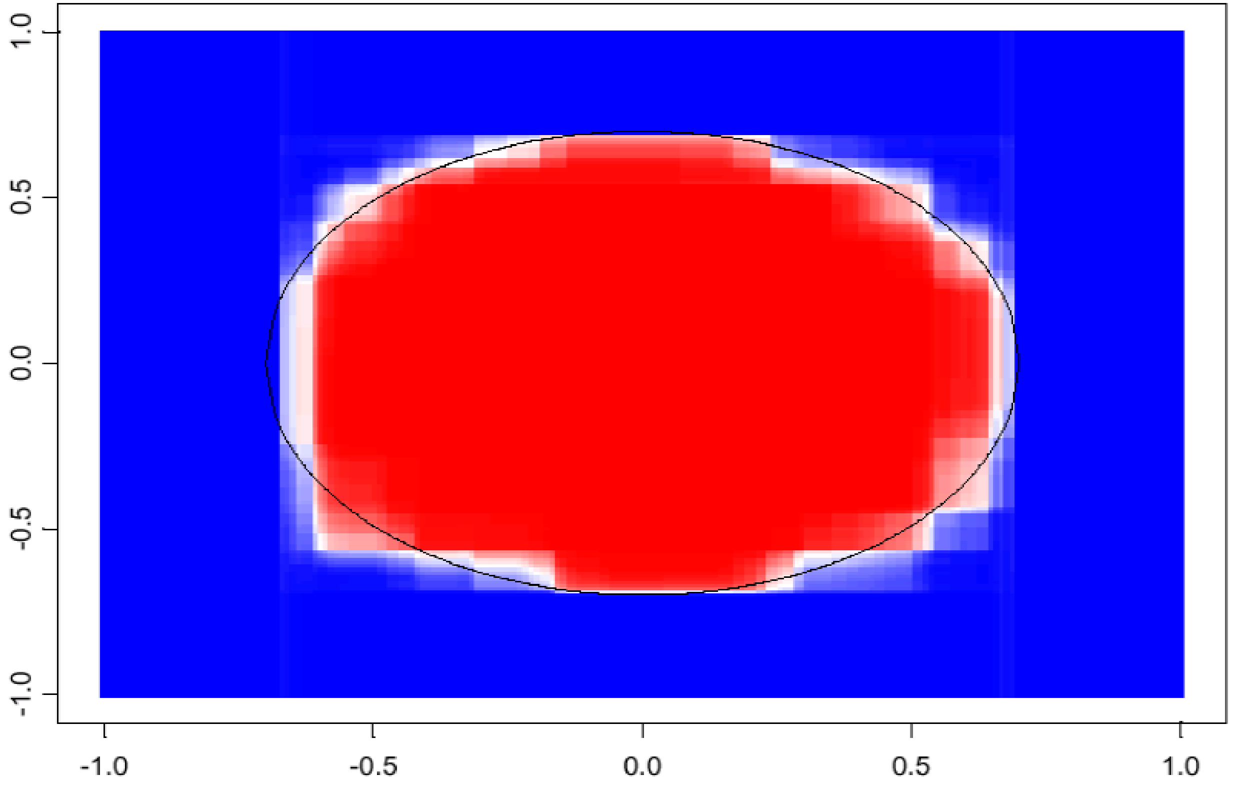
Example of Bagging



Example of Bagging



100 bagged trees



shades of blue/red indicate strength of vote for particular classification

Bagging / Resampling

- It reduces variance of model complexity
- It builds various models with many different samplings
- It reduces bias b/c it combines all models
- Aggregating all the models, reduce variance

Practice - Bagging

```
spamD <- read.table('spamD.tsv',header=T,sep='\t')

spamD[1:5, c(1:5,58:59)]

##   word.freq.make word.freq.address word.freq.all word.freq.3d
## 1           0.00              0.64          0.64            0
## 2           0.21              0.28          0.50            0
## 3           0.06              0.00          0.71            0
## 4           0.00              0.00          0.00            0
## 5           0.00              0.00          0.00            0
##   word.freq.our spam rgroup
## 1          0.32 spam     52
## 2          0.14 spam     91
## 3          1.23 spam     49
## 4          0.63 spam     88
## 5          0.63 spam     73

spamTrain <- subset(spamD,spamD$rgroup>=10)
spamTest <- subset(spamD,spamD$rgroup<10)
```

```
# Note 1:
#   Load the data and split into training (90% of data)
#   and test (10% of data) sets.
```

Building Tree model

```
# Note 2:
#   Use all the features and do binary classification,
#   where TRUE corresponds to spam documents.
```

```
spamVars <- setdiff(colnames(spamD),list('rgroup','spam'))
spamFormula <- as.formula(paste('spam=="spam"',
                        paste(spamVars,collapse=' + '),sep=' ~ '))
```

```
# Note 4:
#   A function to calculate and return various measures
#   on the model: prediction accuracy, and f1, which is the
#   harmonic mean of precision and recall.
```

```
accuracyMeasures <- function(pred, truth, name="model") {
  ctable <- table(truth=truth,
                  pred=(pred>0.5))
  accuracy <- sum(diag(ctable))/sum(ctable)
  precision <- ctable[2,2]/sum(ctable[,2])
  recall <- ctable[2,2]/sum(ctable[2,])
  f1 <- 2*precision*recall/(precision+recall)
  data.frame(model=name, accuracy=accuracy, f1=f1)
}
```

```
# Note 6:
#   Convert the class probability estimator into a
#   classifier by labeling documents that score greater
than 0.5 as
#   spam.
```

```r
library(rpart)
treemodel <- rpart(spamFormula, spamTrain)

accuracyMeasures(predict(treemodel, newdata=spamTrain),
                 spamTrain$spam=="spam",
                 name="tree, training")

##            model  accuracy      f1
## 1 tree, training 0.9104514 0.88337

accuracyMeasures(predict(treemodel, newdata=spamTest),
                 spamTest$spam=="spam",
                 name="tree, test")

##        model  accuracy        f1
## 1 tree, test 0.8799127 0.8414986
```

bagging

```
ntrain <- dim(spamTrain)[1]
n <- ntrain
ntree <- 100
```

```
# Note 1:
#   Use bootstrap samples the same size as the training
#   set, with 100 trees.
```

```
samples <- sapply(1:ntree,
                 FUN = function(iter)
                   {sample(1:ntrain, size=n, replace=T)})
```

```
# Note 2:
#   Build the bootstrap samples by sampling the row indices
of spamTrain with replacement. Each
#   column of the matrix samples represents the row indices
into spamTrain
#   that comprise the bootstrap sample.
```

```
treelist <-lapply(1:ntree,
                  FUN=function(iter)
                  {samp <- samples[,iter];
                   rpart(spamFormula, spamTrain[samp,])})
```

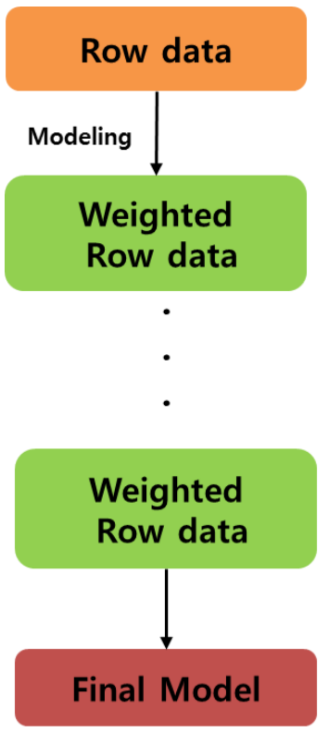
```
# Note 3:
#   Train the individual decision trees and return them
#   in a list. Note: this step can take a few minutes.
```

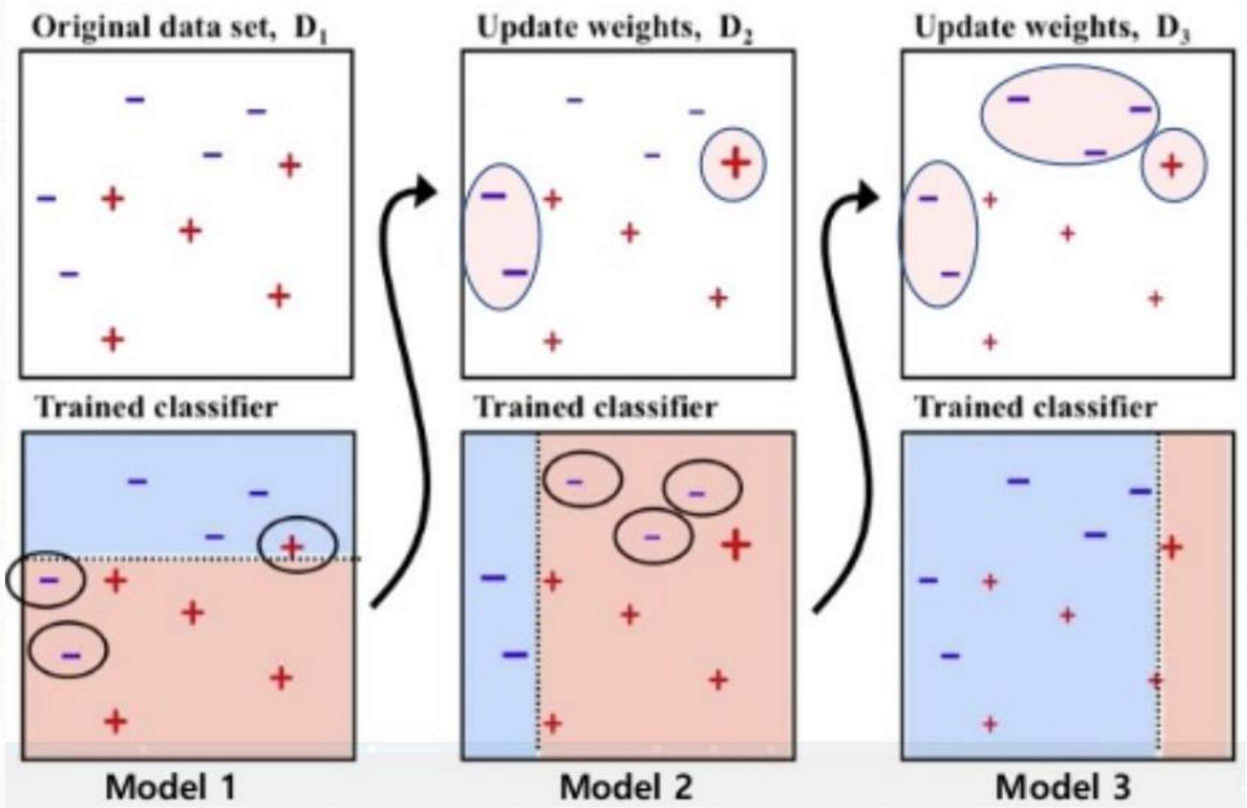
```
predict.bag <- function(treelist, newdata) {
  preds <- sapply(1:length(treelist),
                  FUN=function(iter) {
                    predict(treelist[[iter]], newdata=newdata)})
  predsums <- rowSums(preds)
  predsums/length(treelist)
}
```

```
# Note 4:
#   predict.bag assumes the underlying classifier returns decision probabilities, not
#   decisions.
```

```
accuracyMeasures(predict.bag(treelist, newdata=spamTrain),
                 spamTrain$spam=="spam",
                 name="bagging, training")

##                model  accuracy        f1
## 1 bagging, training 0.9227613 0.8990536

accuracyMeasures(predict.bag(treelist, newdata=spamTest),
                 spamTest$spam=="spam",
                 name="bagging, test")

##            model  accuracy        f1
## 1 bagging, test 0.9126638 0.8809524
```

Boosting (C5.0)





Boosting

- Aggregate the result of boosted model

알고리즘	특징	비고
AdaBoost	• 다수결을 통한 정답 분류 및 오답에 가중치 부여	
GBM	• Loss Function의 gradient를 통해 오답에 가중치 부여	gradient_boosting.pdf
Xgboost	• GBM 대비 성능향상 • 시스템 자원 효율적 활용 (CPU, Mem) • Kaggle을 통한 성능 검증 (많은 상위 랭커가 사용)	2014년 공개 boosting-algorithm-xgboost
Light GBM	• Xgboost 대비 성능향상 및 자원소모 최소화 • Xgboost가 처리하지 못하는 대용량 데이터 학습 가능 • Approximates the split (근사치의 분할)을 통한 성능 향상	2016년 공개 light-gbm-vs-xgboost

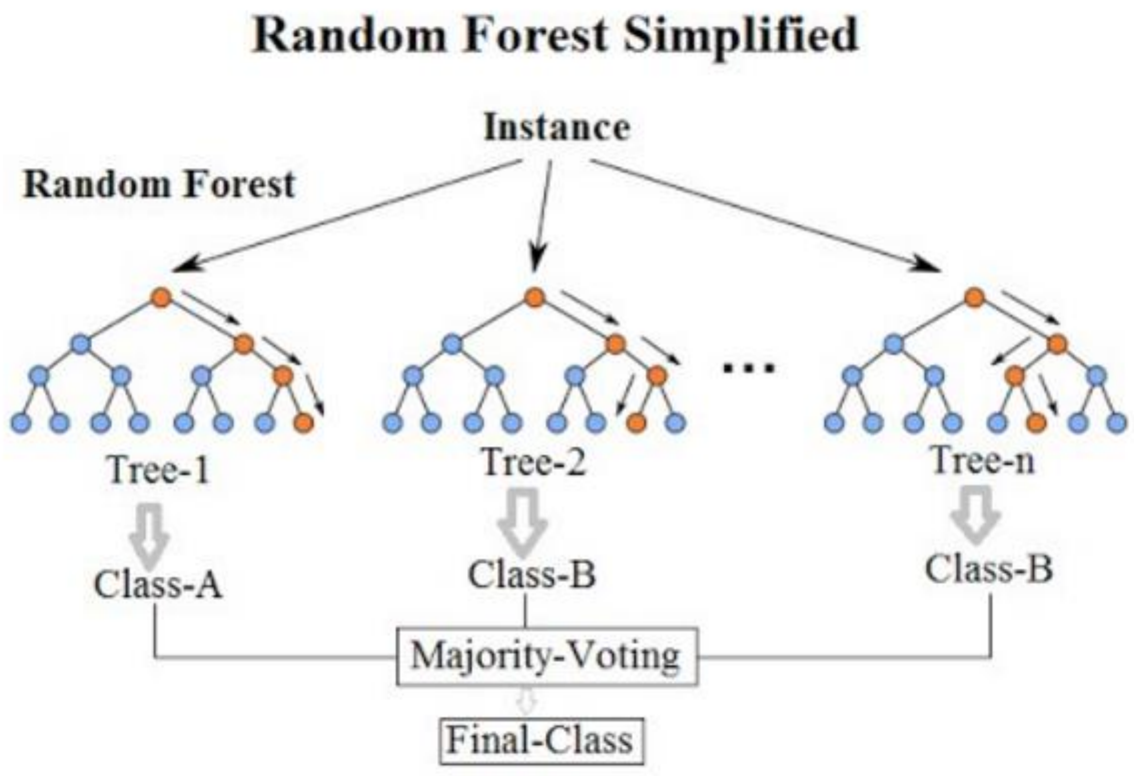
Bagging v.s. Boosting

비교	Bagging	Boosting
특징	병렬 앙상블 모델 (각 모델은 서로 독립적)	연속 앙상블 (이전 모델의 오류를 고려)
목적	Variance 감소	Bias 감소
적합한 상황	복잡한 모델 (High variance, Low bias)	Low variance, High bias 모 델
대표 알고리즘	Random Forest	Gradient Boosting, AdaBoost
Sampling	Random Sampling	Random Sampling with weight on error

Random Forest

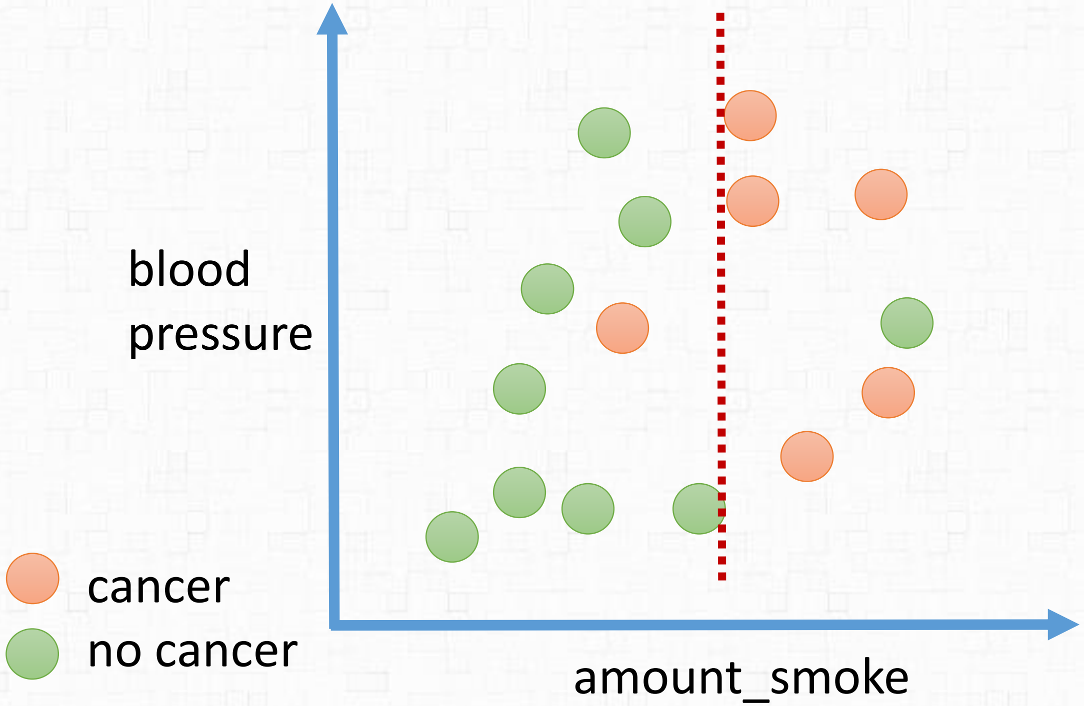
Random Forest

- Tree-based Enssenble model with Bagging



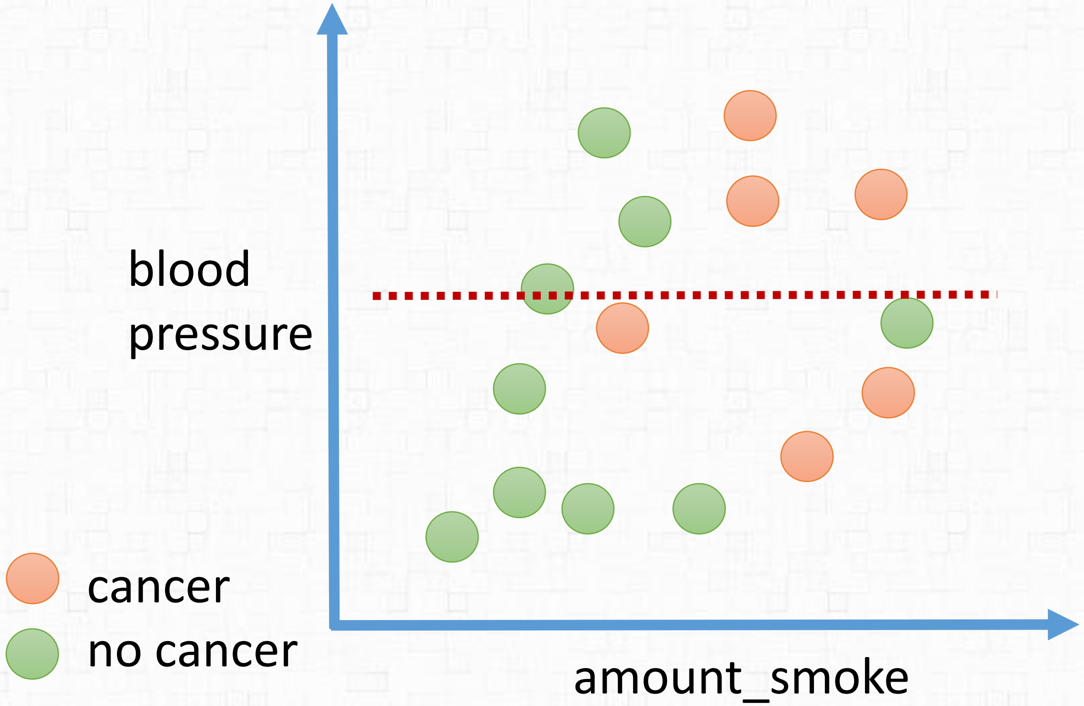
Random Forest – Tree Correlation

- Bagging reduces variance by aggregating many different models
- Bagging samples **observations** not features
- If certain features have high predictive power then all the tree models look very similar
  - Eg. Smoker variable of insurance dataset
  - High correlation in trees
- Random Forest
  - Select **features** randomly and build trees



Random Forest – Tree Correlation

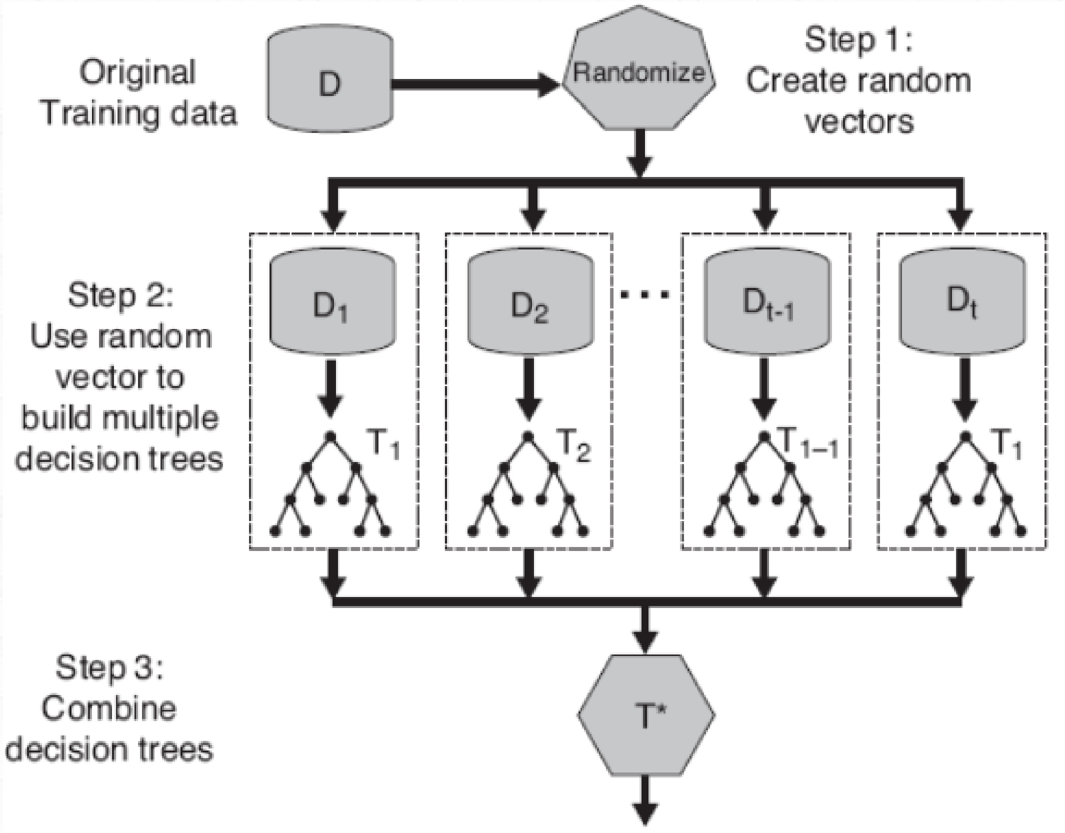
- Bagging reduces variance by aggregating many different models
- Bagging samples **observations** not features
- If certain features have high predictive power then all the tree models look very similar
  - Eg. Smoker variable of insurance dataset
  - High correlation in trees
- Random Forest
  - Select **features** randomly and build trees



Random Forests

- Ensemble method specifically designed for decision tree classifiers
- Introduce two sources of randomness:
  - “Bagging” and “Random input vectors”
  - **Bagging method**: each tree is grown using a bootstrap sample of training data
  - **Random vector method**: At each node, best split is chosen from a random sample of m attributes instead of all attributes

Random Forests



```
library(randomForest)

set.seed(5123512)
fmodel <- randomForest(x=spamTrain[,spamVars],
        y=spamTrain$spam,
        ntree=100,
        nodesize=7,
        importance=T)
```

```
# Note 4:
#   Use 100 trees to be compatible with our bagging
#   example. The default is 500 trees.
```

```
# Note 5:
#   Specify that each node of a tree must have a minimum
#   of 7 elements, to be compatible with the default minimum node size that rpart()
#   uses on this training set.
```

```
# Note 6:
#   Tell the algorithm to save information to be used for
#   calculating variable importance (we’ll see this later).
```

```
accuracyMeasures(predict(fmodel,
   newdata=spamTrain[,spamVars],type='prob')[,'spam'],
   spamTrain$spam=="spam",name="random forest, train")

##                    model  accuracy        f1
## 1 random forest, train 0.9884142 0.9851943

accuracyMeasures(predict(fmodel,
   newdata=spamTest[,spamVars],type='prob')[,'spam'],
   spamTest$spam=="spam",name="random forest, test")

##                   model  accuracy        f1
## 1 random forest, test 0.9497817 0.9340974
```

Estimated Error Rate

1. At each bootstrap iteration, predict the data not in the bootstrap sample (what Breiman calls "out-of-bag", or OOB, data) using the tree grown with the bootstrap sample.
2. Aggregate the OOB predictions. (On the average, each data point would be out-of-bag around 36% of the times, so aggregate these predictions.) Calcuate the error rate, and call it the OOB estimate of error rate.

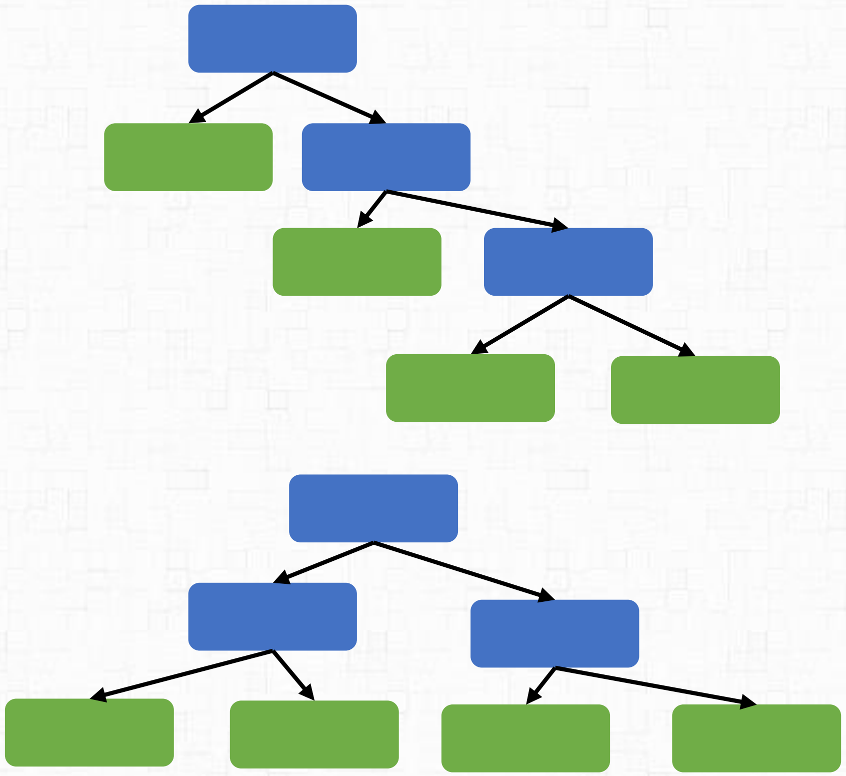
Variable importance

- Make permutation on certain variable p of OOB data while other variables unchanged
- Compare prediction accuracy on permuted data and unchanged data

Adaboost and XGBoost

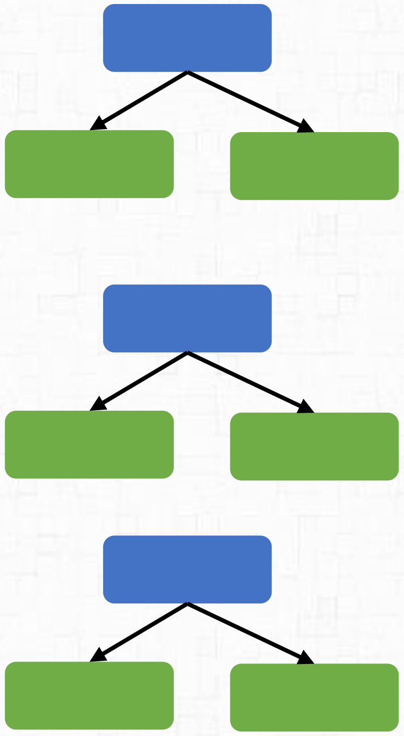
Random Forest

uses a set of fully grown trees



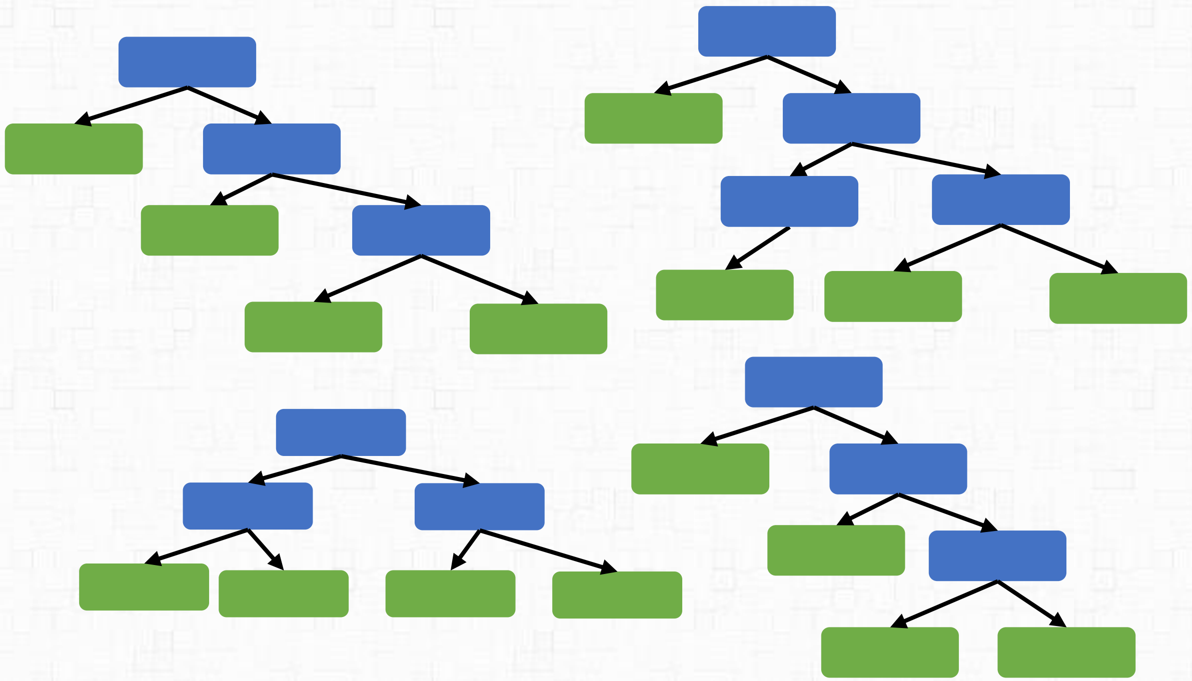
Adaboost

uses a set of stumps
– simplified decision tree (weak learner)



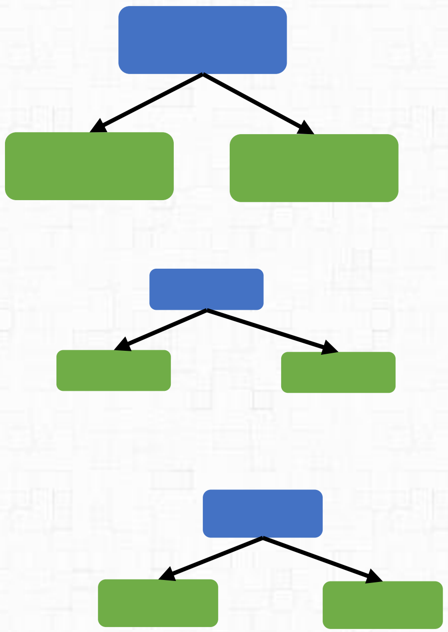
Random Forest

all trees are equally important
to make a final decision



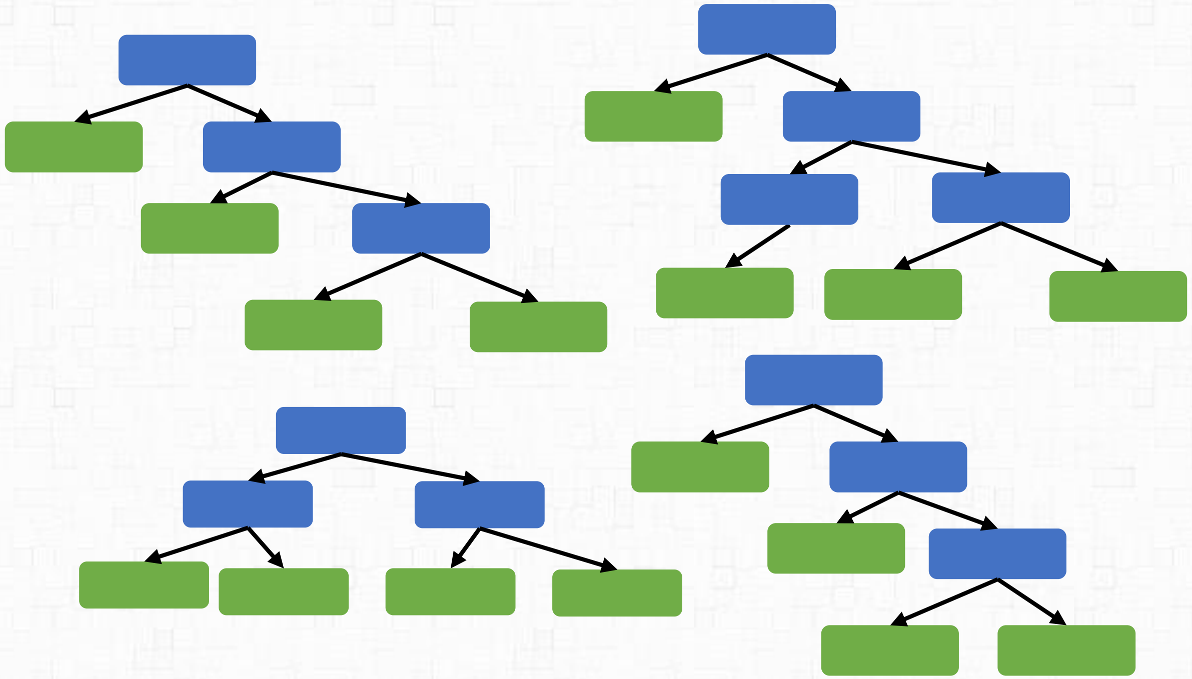
Adaboost

some stumps are more important
than others
when making a final decision



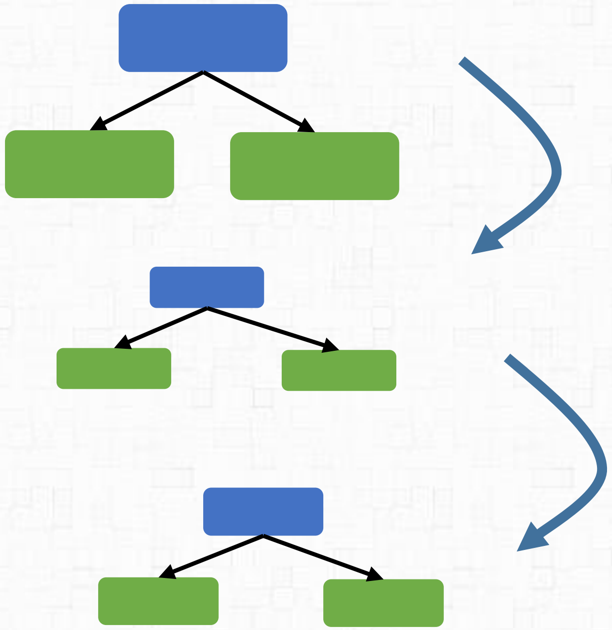
Random Forest

makes all tree independently



Adaboost

makes trees sequentially



Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

Chest Pain

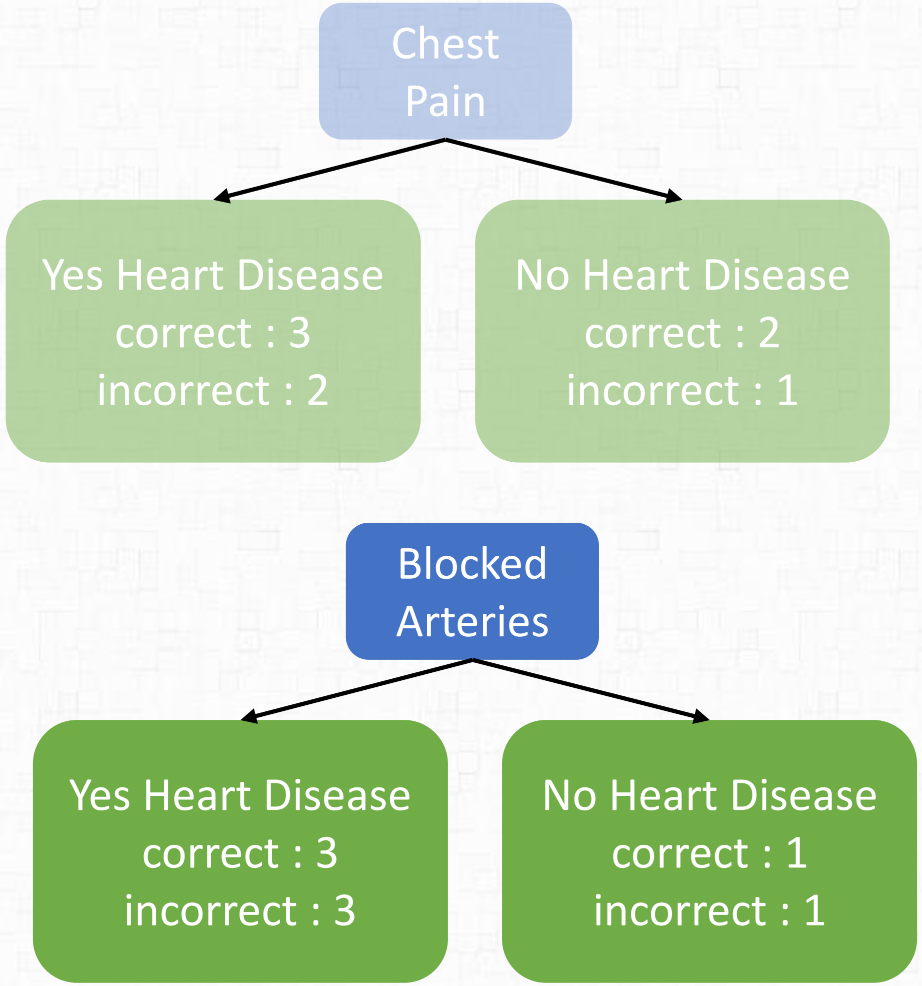
Yes Heart Disease
$\boxed{\text{correct : 3}}$
incorrect : 2

No Heart Disease
correct : 2
incorrect : 1

$$\text{G.I.} = \frac{5}{8}\left[1 - \left\{\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2\right\}\right] + \frac{3}{8}\left[1 - \left\{\left(\frac{2}{3}\right)^2 + \left(\frac{1}{3}\right)^2\right\}\right]$$
$$= 0.47$$

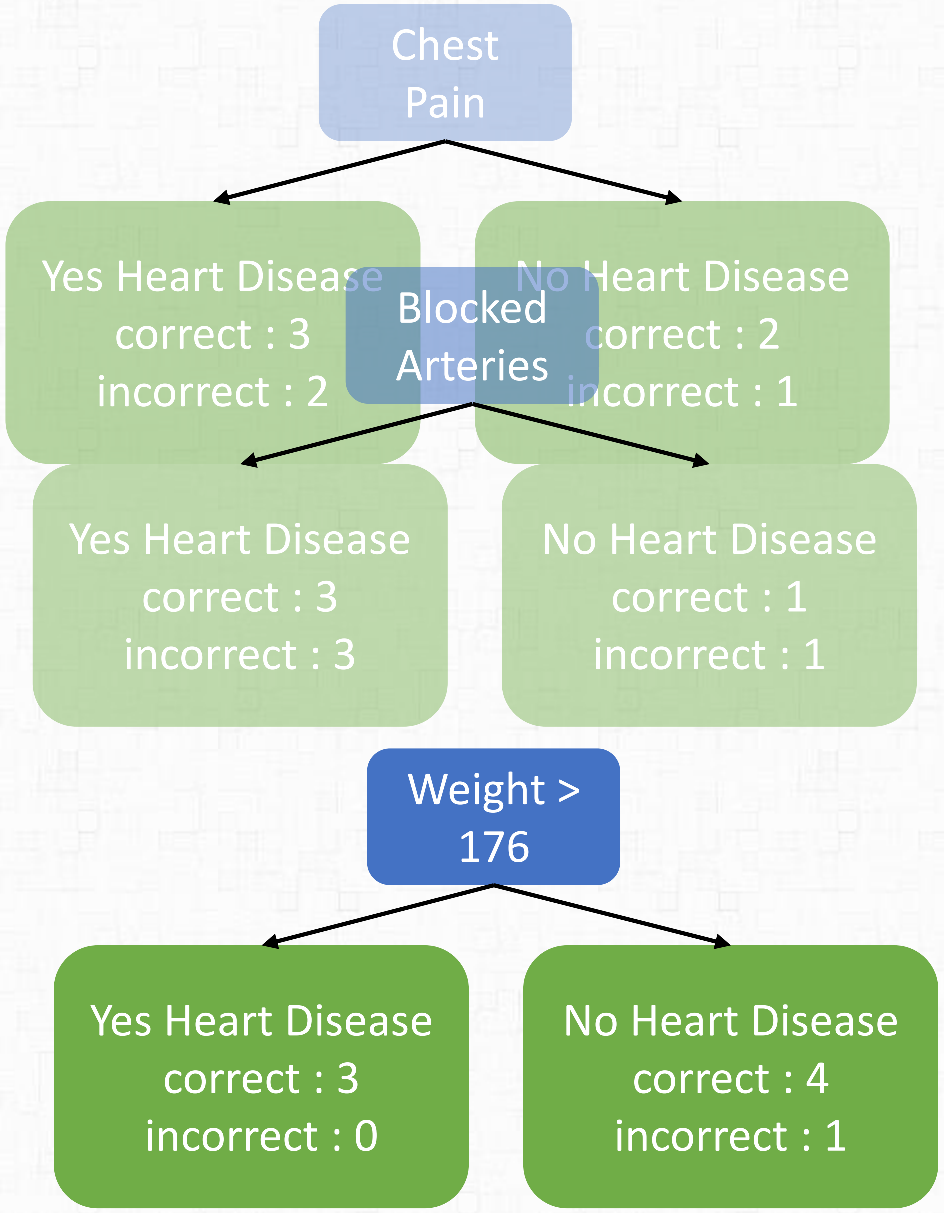
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
Yes	Yes	205	Yes
No	Yes	180	Yes
Yes	No	210	Yes
Yes	Yes	167	Yes
No	Yes	156	No
No	Yes	125	No
Yes	No	168	No
Yes	Yes	172	No

Sample Weight
1/8
1/8
1/8
1/8
1/8
1/8
1/8
1/8



$$\text{G.I.} = \frac{6}{8}\left[1 - \left\{\left(\frac{3}{6}\right)^2 + \left(\frac{3}{6}\right)^2\right\}\right] + \frac{2}{8}\left[1 - \left\{\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2\right\}\right]$$
$$= 0.5$$

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8



$$\text{G.I.} = \frac{3}{8}\left[1-\left\{\left(\frac{3}{3}\right)^2+\left(\frac{0}{3}\right)^2\right\}\right]+\frac{5}{8}\left[1-\left\{\left(\frac{4}{5}\right)^2+\left(\frac{1}{5}\right)^2\right\}\right] = 0.2$$

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

Weight > 176

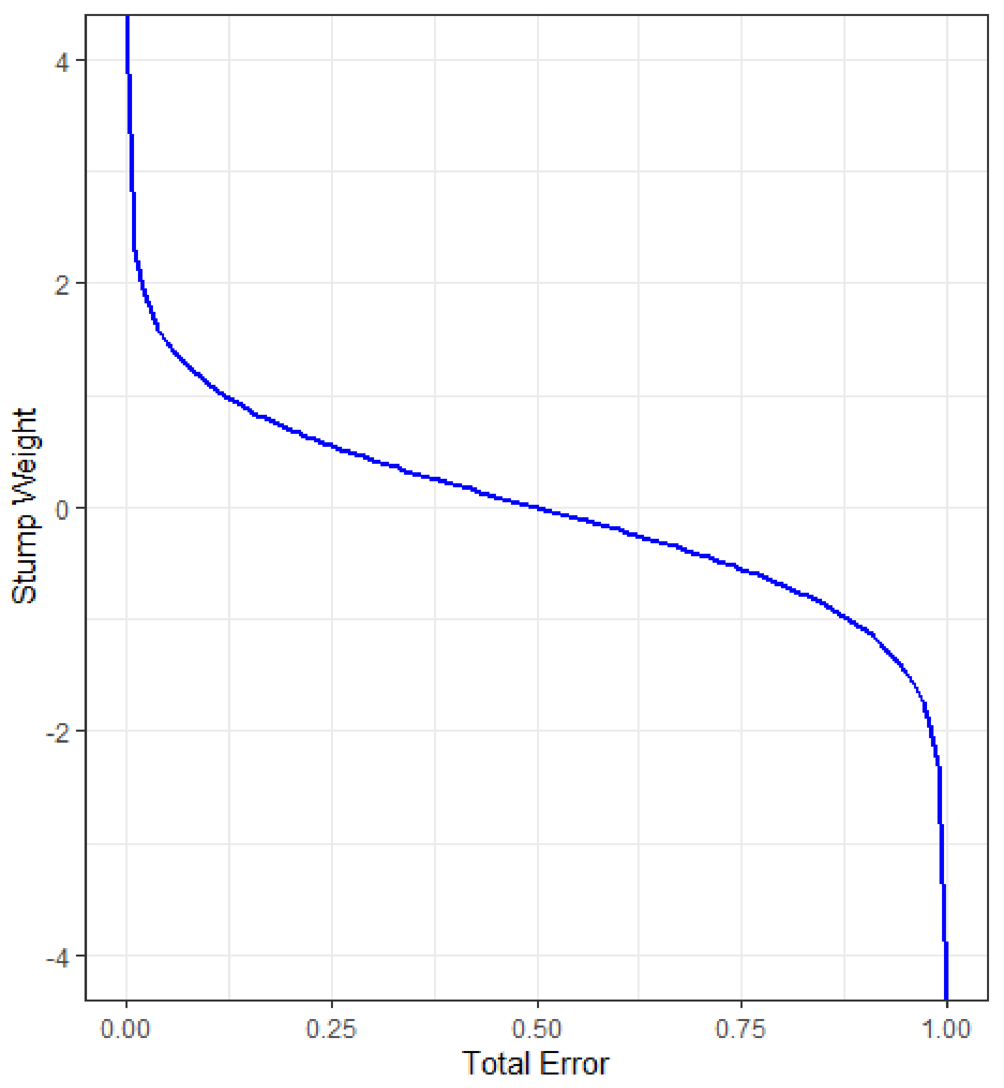
Yes Heart Disease
correct : 3
incorrect : 0

No Heart Disease
correct : 4
incorrect : 1

Total Error: the sum of weights for *incorrectly* classified samples
In the example, Total Error is 1/8

Stumps Weight:
Amount of weight that the stump contributes to the final decision

$$\frac{1}{2}\log\left(\frac{1 - total\ error}{total\ error}\right)$$



Weight > 176

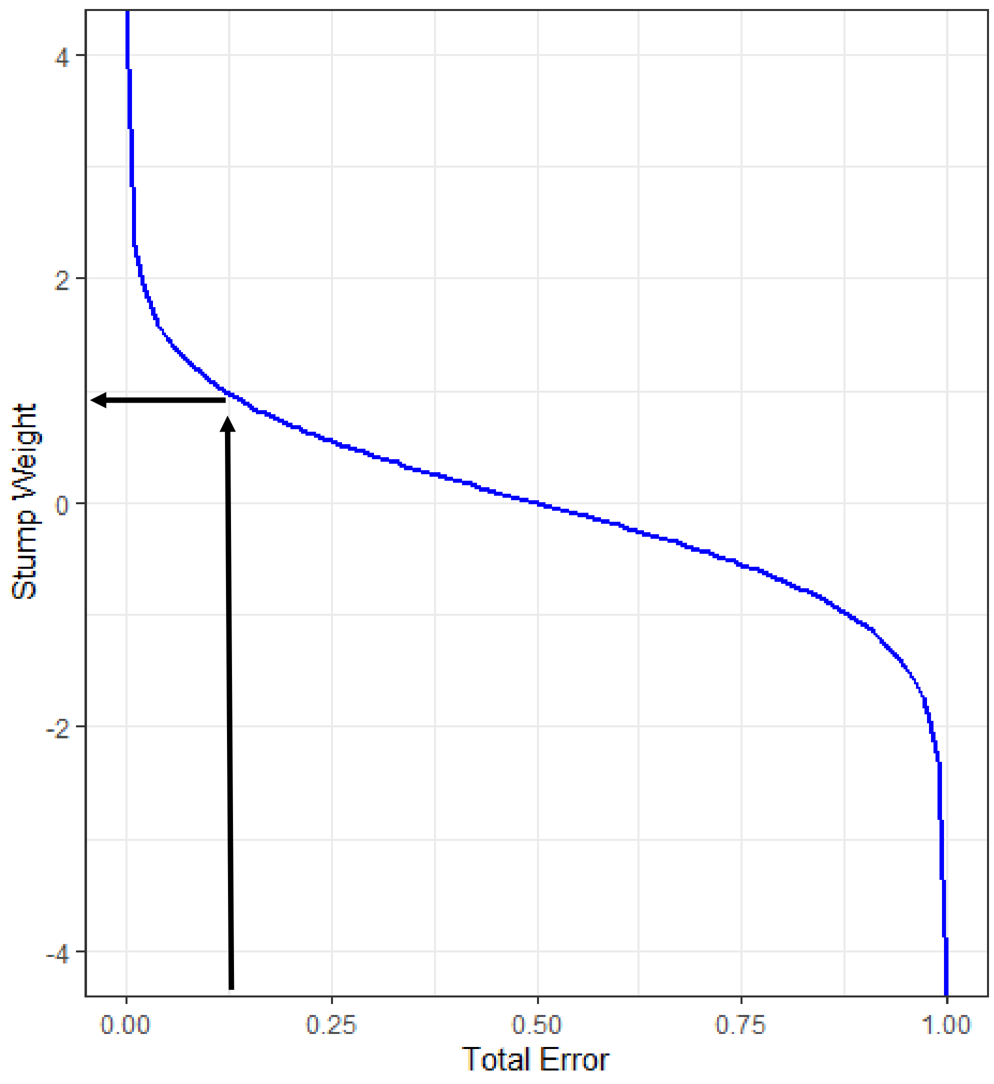
Yes Heart Disease
correct : 3
incorrect : 0

No Heart Disease
correct : 4
incorrect : 1

Total Error: the sum of weights for *incorrectly* classified samples
In the example, Total Error is 1/8

Stumps Weight:
Amount of weight that the stump contributes to the final decision

$$\frac{1}{2}\log\left(\frac{1 - total\ error}{total\ error}\right)$$



Weight > 176

Yes Heart Disease
correct : 3
incorrect : 0

No Heart Disease
correct : 4
incorrect : 1

Total Error: the sum of weights for *incorrectly* classified samples
In the example, Total Error is 1/8

Stumps Weight:
Amount of weight that the stump contributes to the final decision

$$\frac{1}{2}\log\left(\frac{1-\frac{1}{8}}{\frac{1}{8}}\right) = 0.97$$

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

New Sample Weight:

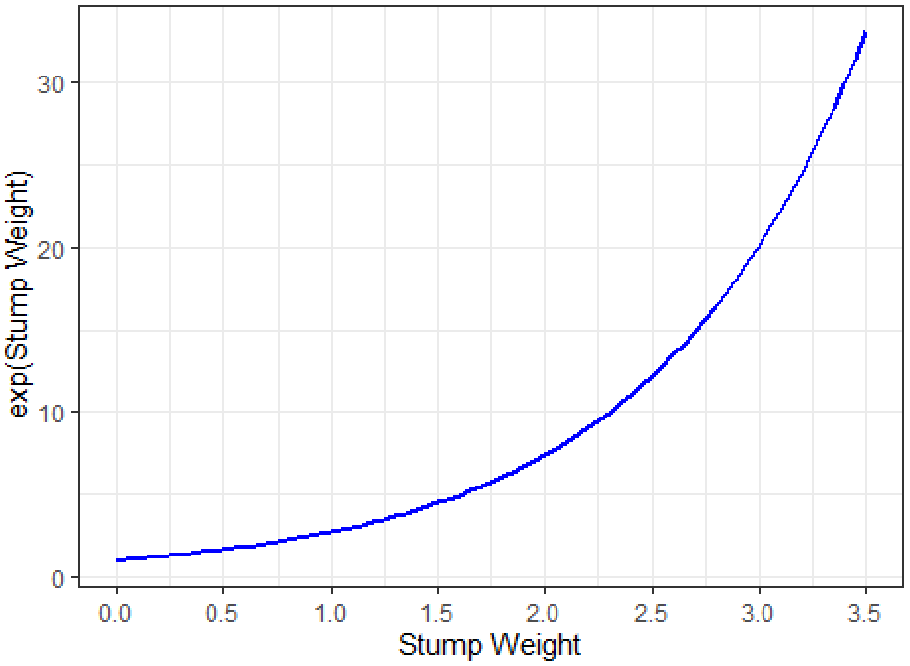
Incorrectly classified samples :

$$sample\ weight\ \times e^{stump\ weight}$$

$$= \frac{1}{8} \times e^{0.97} = \frac{1}{8} \times 2.64 = 0.33$$

Correctly classified samples :

$$sample\ weight\ \times e^{stump\ weight}$$



Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

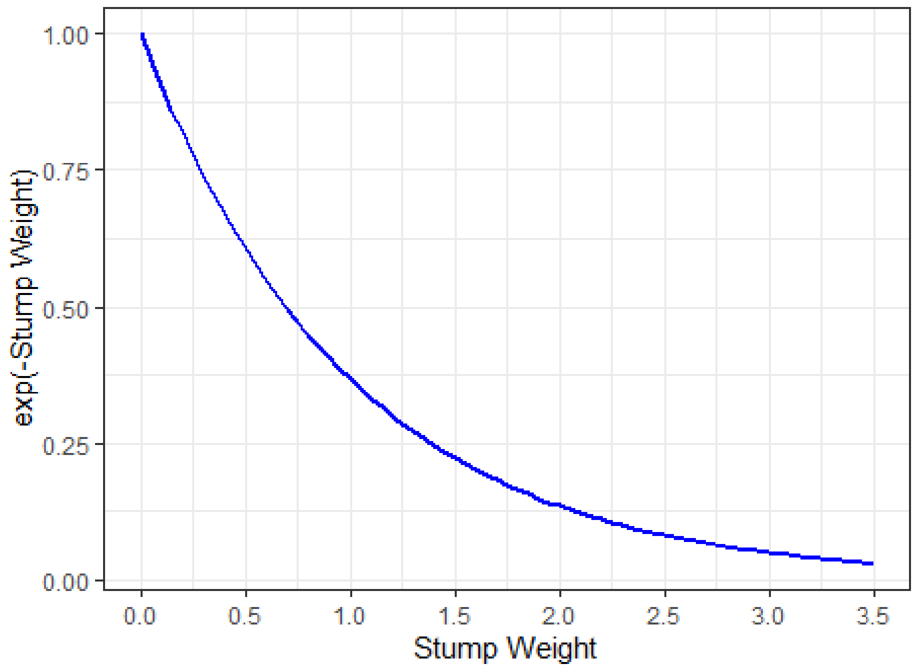
New Sample Weight:

Incorrectly classified samples :

$$sample\ weight \ \times e^{stump\ weight}$$
$$= \frac{1}{8} \times e^{0.97} = \frac{1}{8} \times 2.64 = 0.33$$

Correctly classified samples :

$$sample\ weight \ \times e^{stump\ weight}$$
$$= \frac{1}{8} \times e^{-0.97} = \frac{1}{8} \times 0.38 = 0.05$$



Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	Norm. Sample Weight
Yes	Yes	205	Yes	0.05	0.07
No	Yes	180	Yes	0.05	0.07
Yes	No	210	Yes	0.05	0.07
Yes	Yes	167	Yes	0.33	0.49
No	Yes	156	No	0.05	0.07
No	Yes	125	No	0.05	0.07
Yes	No	168	No	0.05	0.07
Yes	Yes	172	No	0.05	0.07

New Sample Weight:

Incorrectly classified samples :

$$sample\ weight\ \times e^{stump\ weight}$$

$$= \frac{1}{8} \times e^{0.97} = \frac{1}{8} \times 2.64 = 0.33$$

Correctly classified samples :

$$sample\ weight\ \times e^{stump\ weight}$$

$$= \frac{1}{8} \times e^{-0.97} = \frac{1}{8} \times 0.38 = 0.05$$

sum of new sample weights = 0.68

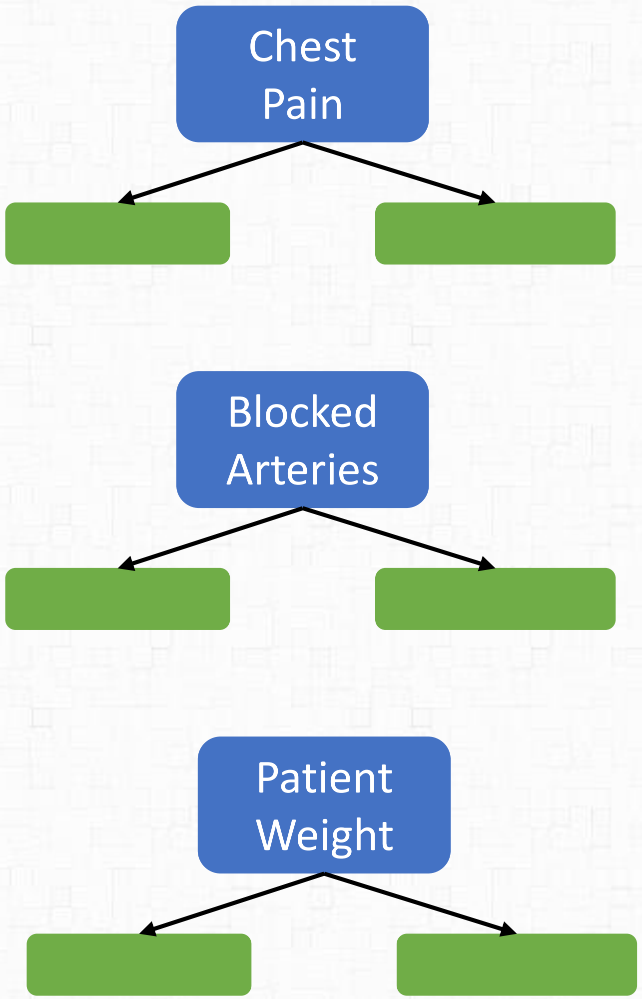
normalize the new weight dividing by 0.68

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	**Yes**	**167**	**Yes**	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

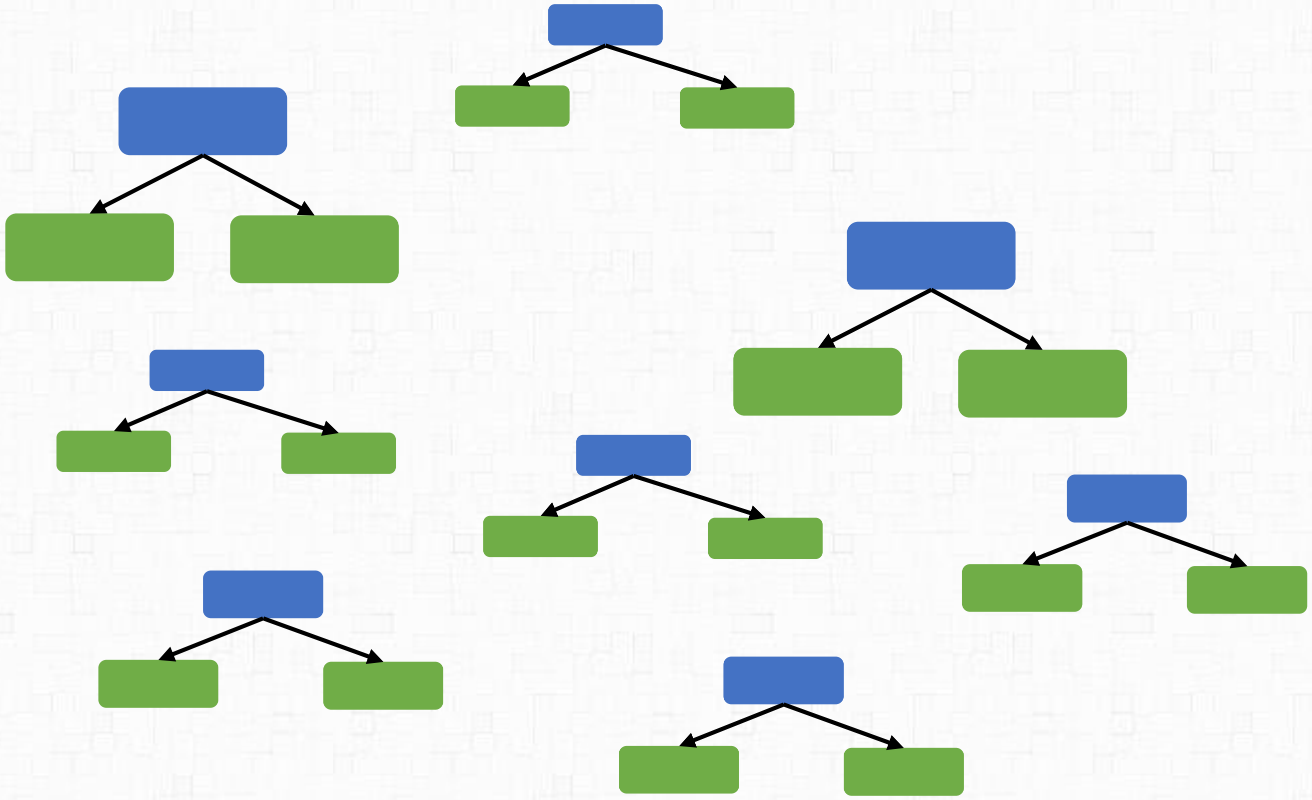
make a new sample
with the same sample size
with respect to sample weight

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
No	Yes	156	No
Yes	**Yes**	**167**	**Yes**
No	Yes	125	No
Yes	**Yes**	**167**	**Yes**
Yes	**Yes**	**167**	**Yes**
Yes	Yes	172	No
Yes	**Yes**	**167**	**Yes**
Yes	Yes	167	Yes

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
No	Yes	156	No	1/8
Yes	Yes	167	Yes	1/8
No	Yes	125	No	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	172	No	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	167	Yes	1/8

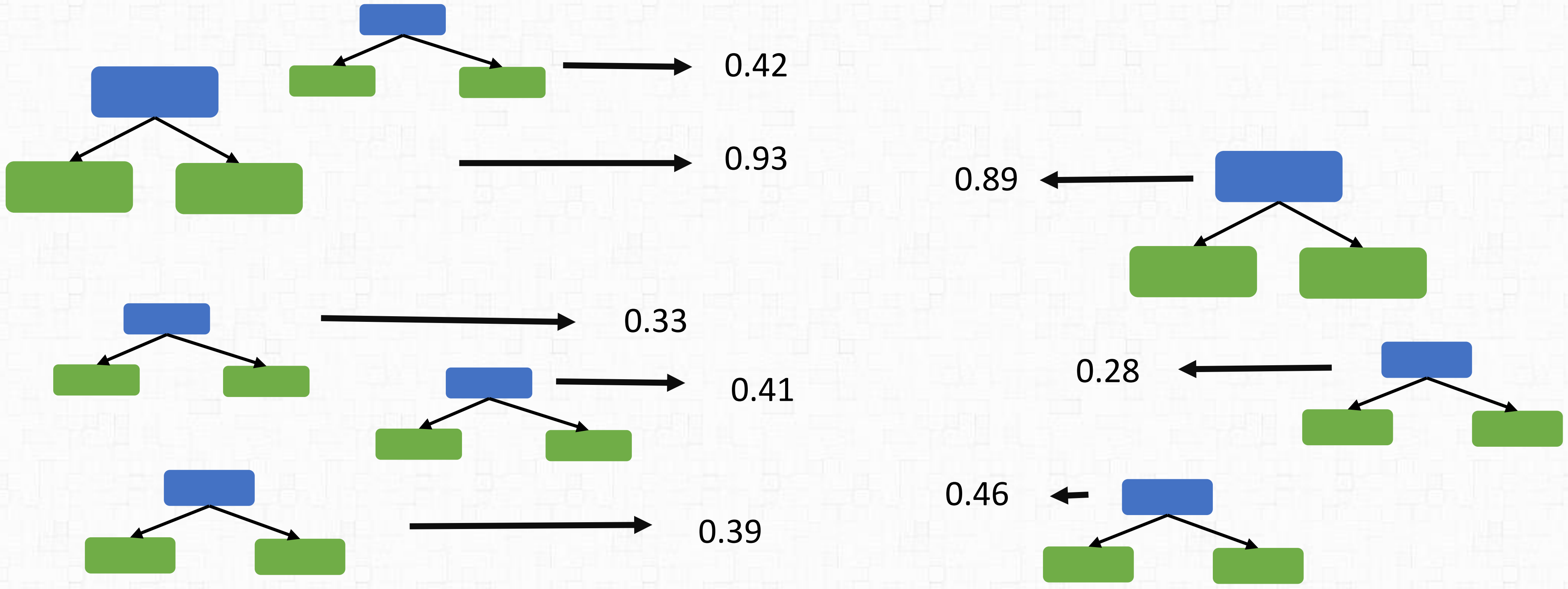


Now we have a forest of stumps...



Heart Disease !

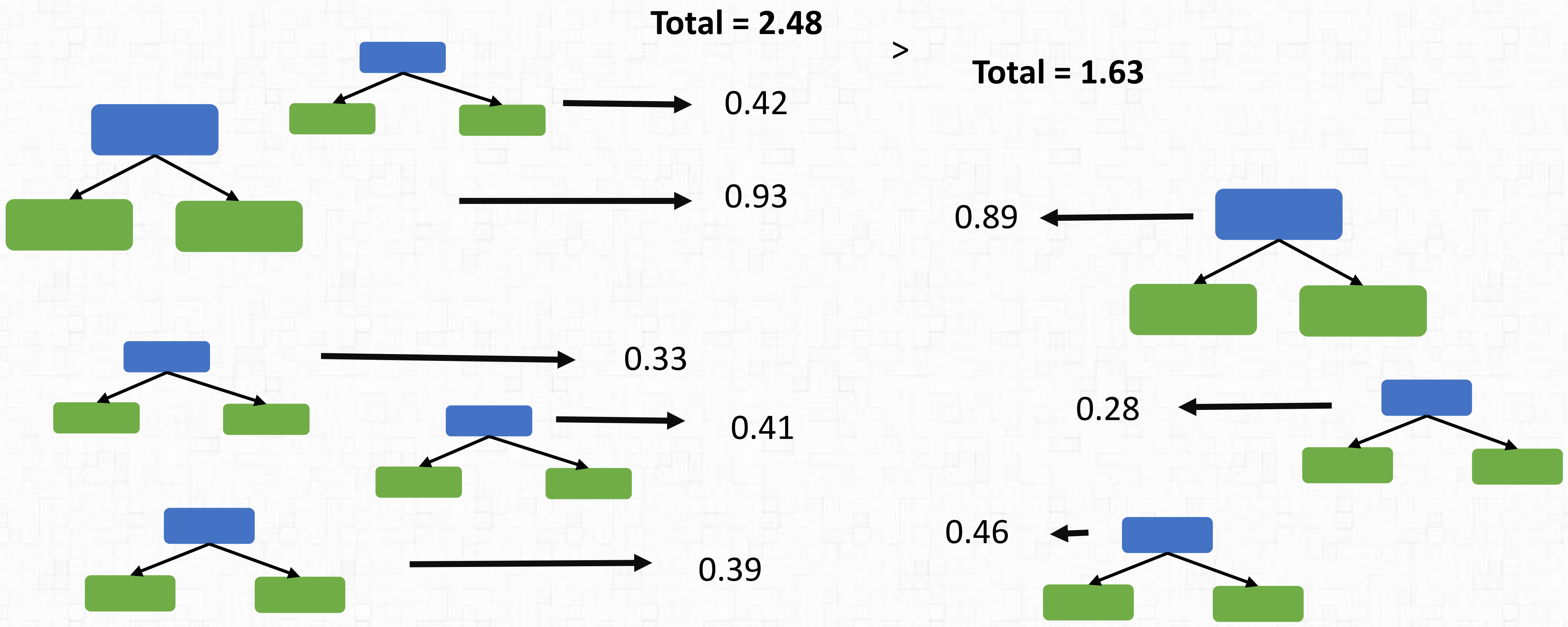
No Heart Disease !



Final Call
The patient has Heart Disease !

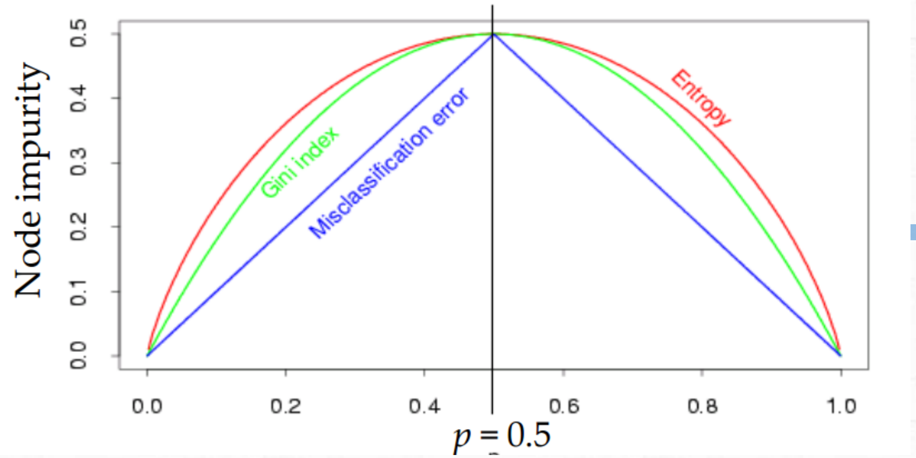
Heart Disease !

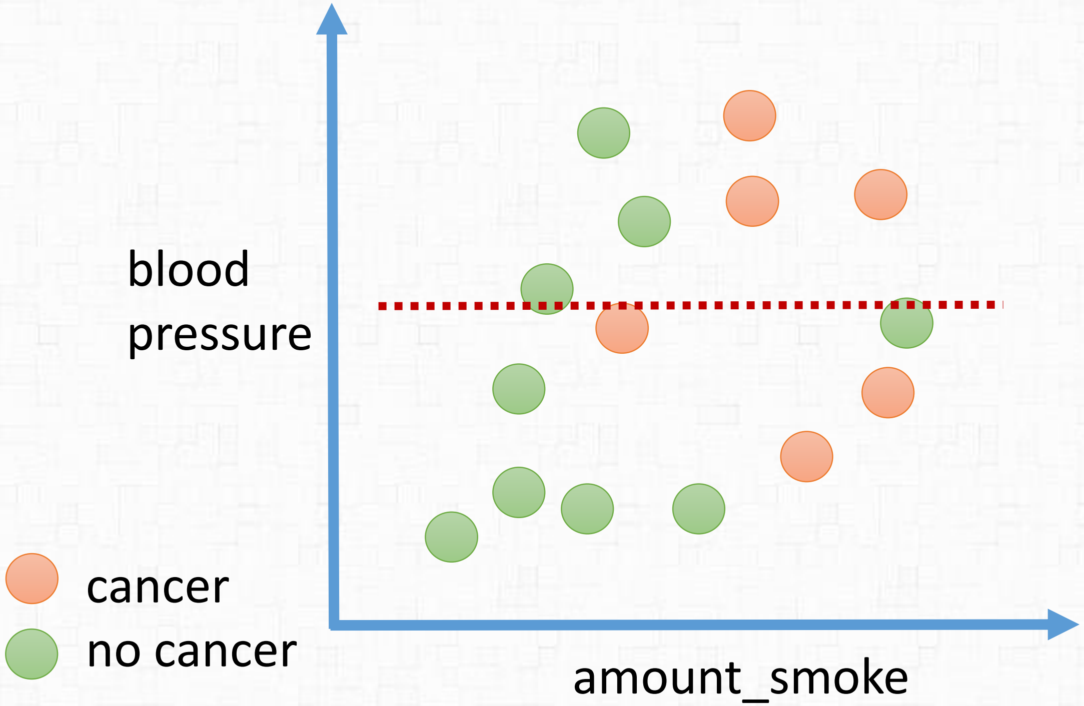
No Heart Disease !

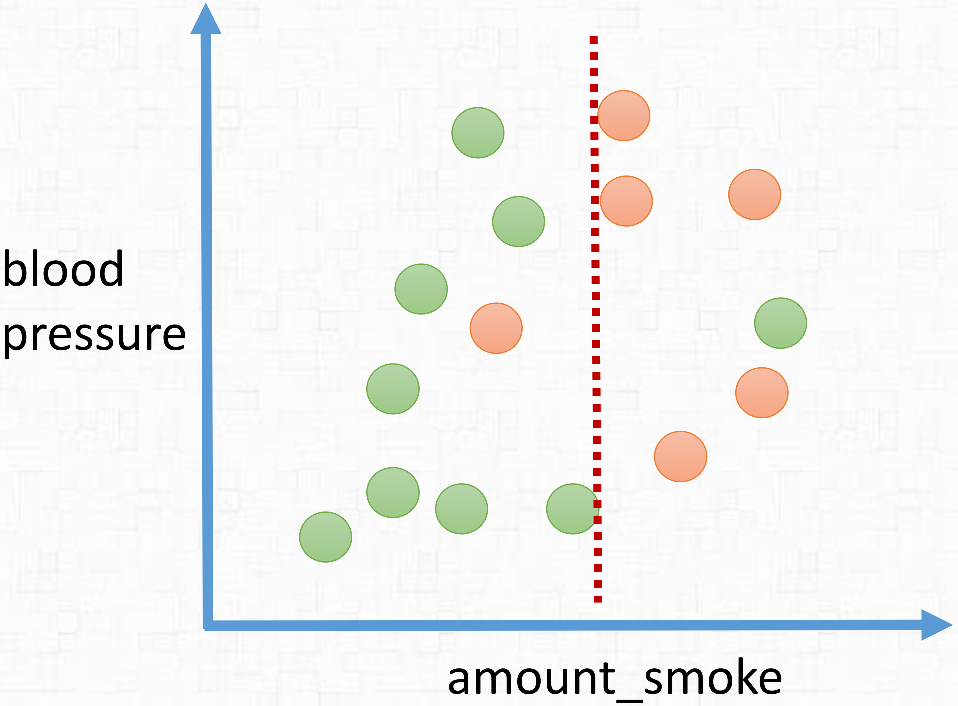


Gini Index (G. I.)

$$G.I(A) = \sum_{i=1}^{d} \left(R_i \left(1 - \sum_{k=1}^{m} p_{ik}^2 \right) \right)$$

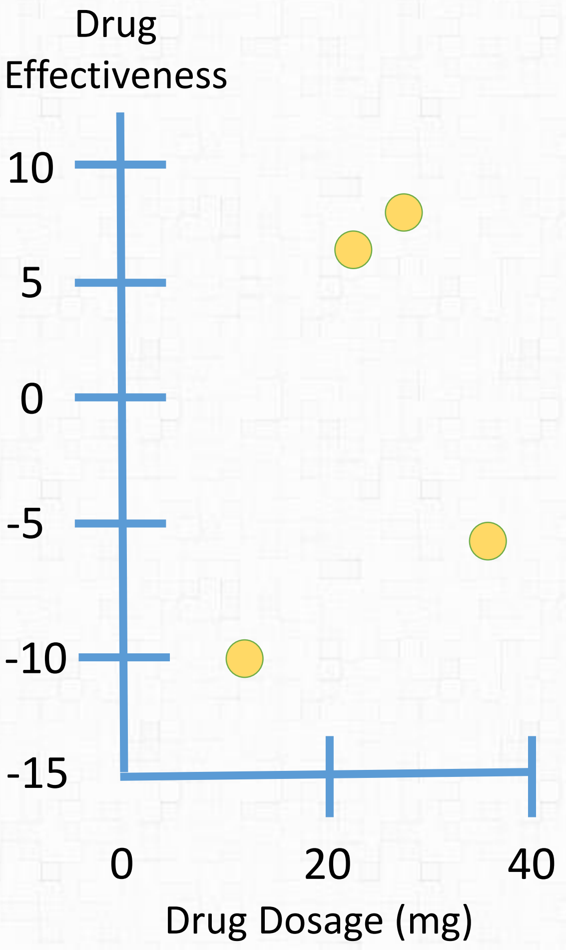






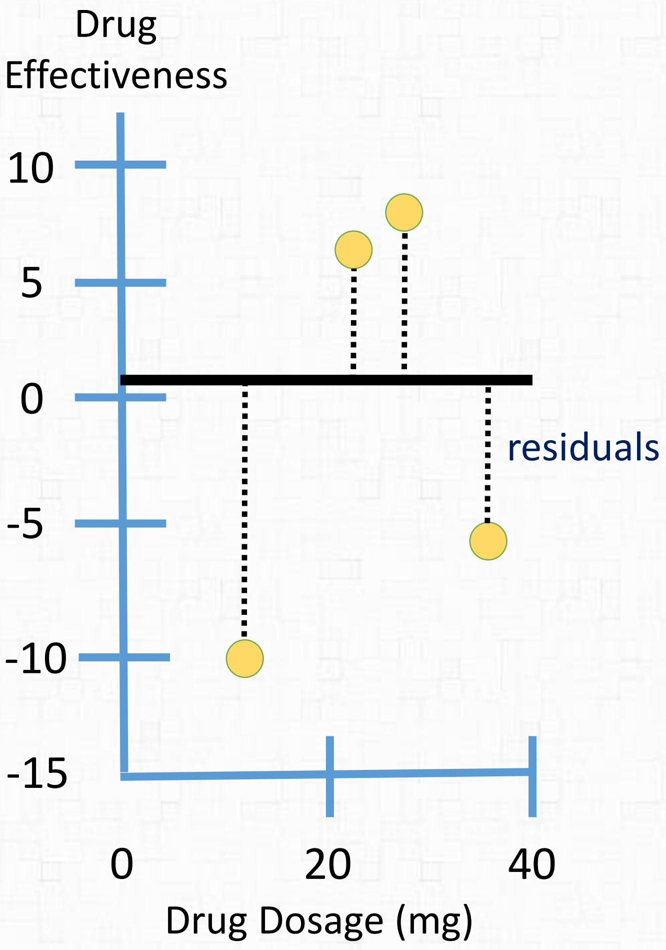
XGBoost for Regression

Initial Prediction: 0.5



XGBoost for Regression

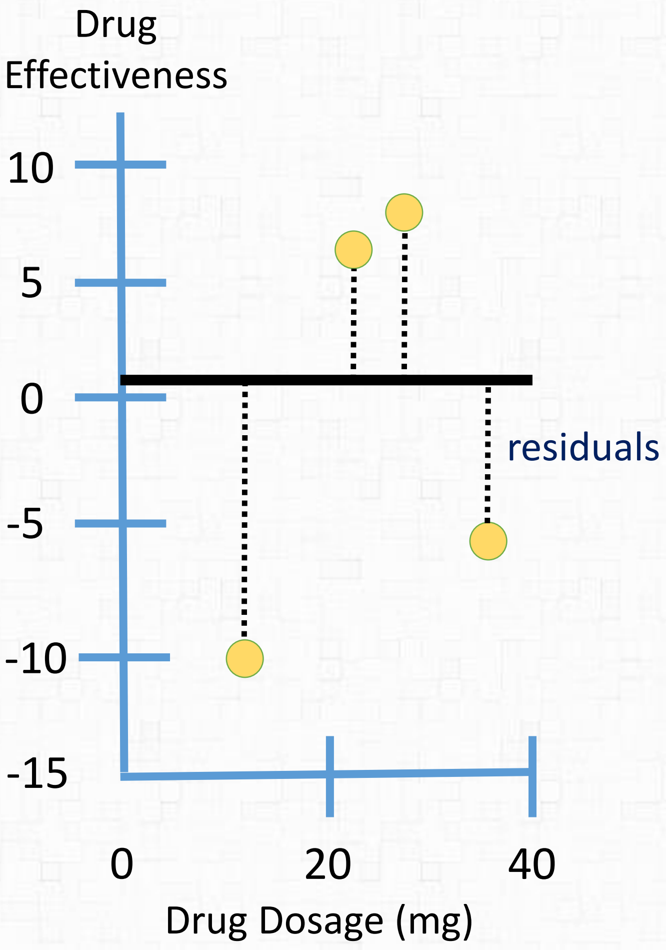
Initial Prediction: 0.5



-10.5, 6.5, 7.5, -7.5

From residuals,
We build the first **XGBoost Tree**

XGBoost for Regression



Initial Prediction: 0.5

-10.5, 6.5, 7.5, -7.5

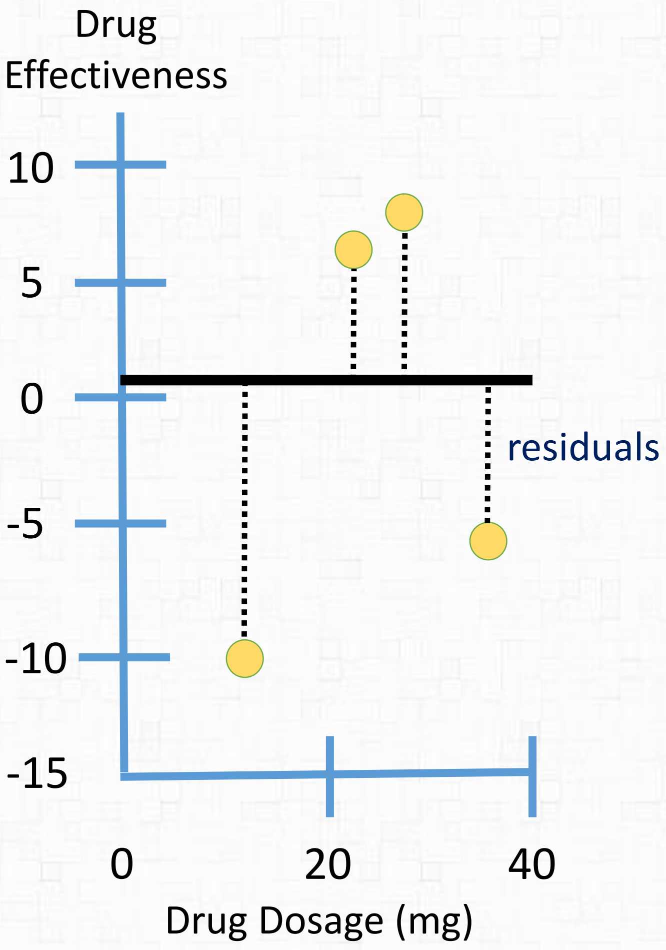
From residuals,
We build the first **XGBoost Tree**

$$\text{Simliarity Score} = \frac{\text{Sum of Residual, Squared}}{\text{Number of Residuals} + \lambda(\text{lambda})}$$

λ is a regularization term (to avoid overfitting)
Let's assume λ = 0

XGBoost for Regression

Initial Prediction: 0.5



From residuals,
We build the first **XGBoost Tree**

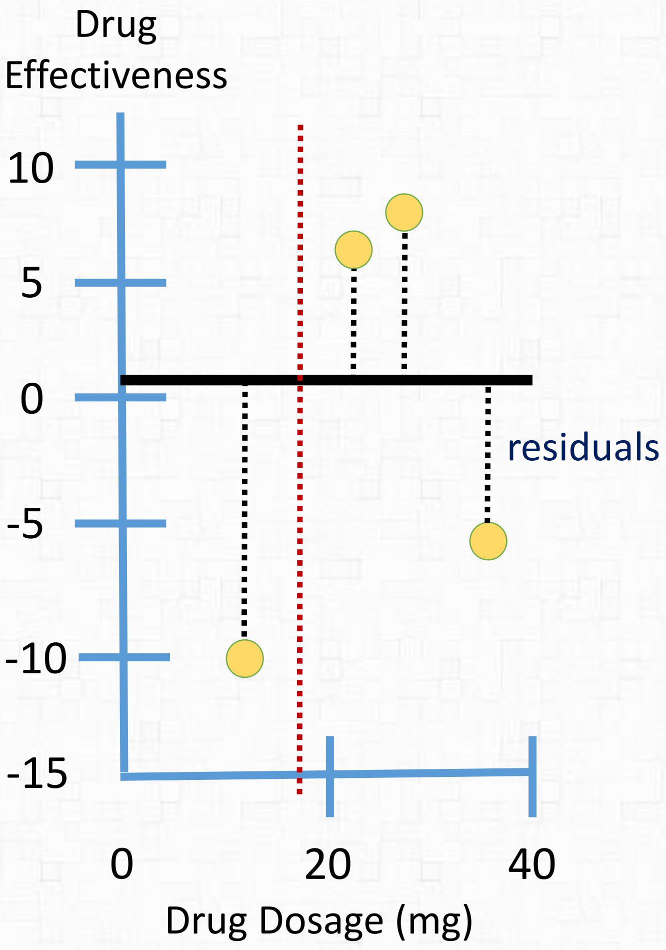
-10.5, 6.5, 7.5, -7.5

$\text{Simliarity} = 4$

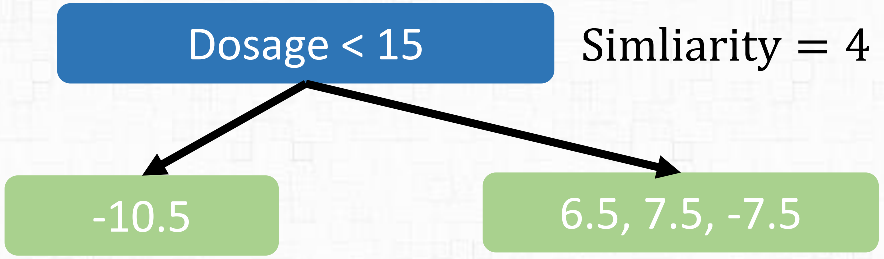
$$\text{Simliarity Score} = \frac{(-10.5 + 6.5 + 7.5 + -7.5)^2}{4 + \mathbf{0}}$$
$$= \frac{(-4)^2}{4+0} = 4$$

λ is a regularization term (to avoid overfitting)
Let's assume λ = 0

XGBoost for Regression



Initial Prediction: 0.5



$\text{Simliarity} = 4$

$$\text{Simliarity Score} = \frac{(-10.5)^2}{1 + \mathbf{0}}$$

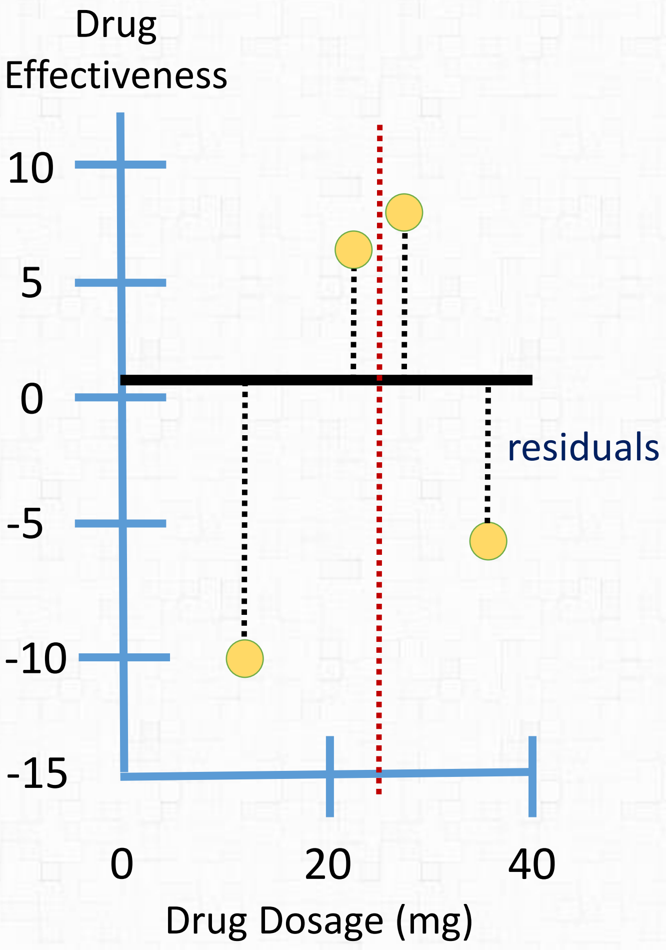
$$= 110.25$$

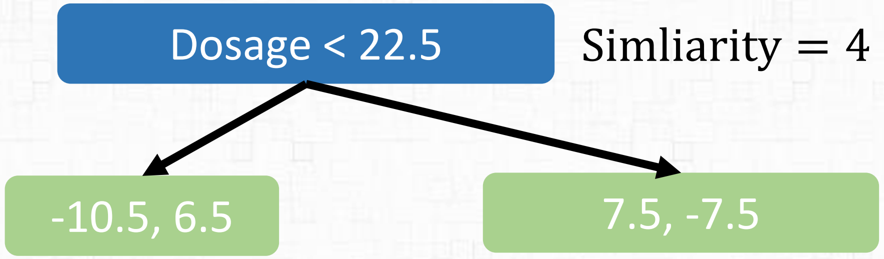
$$\text{Simliarity Score} = \frac{(6.5 + 7.5 + -7.5)^2}{3 + \mathbf{0}}$$

$$= 14.08$$

$$\text{Gain} = \text{Leftsimiliar}_{\text{ity}} + \text{Rightsi}_{\text{miliarity}} - \text{Rootsi}_{\text{miliarity}} = 120.33$$

XGBoost for Regression





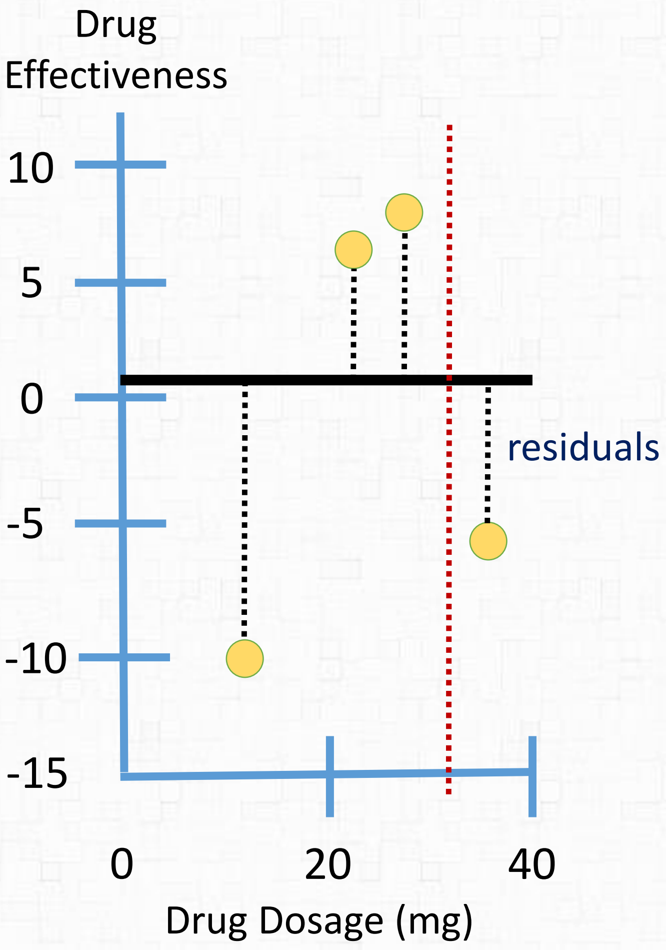
$\text{Simliarity} = 4$

$\text{Simliarity Score} = 8$

$\text{Simliarity Score} = 0$

$$\text{Gain} = 8 + 0\ - 4 = 4$$

XGBoost for Regression



Dosage < 30

$\text{Simliarity} = 4$

-10.5, 6.5, 7.5

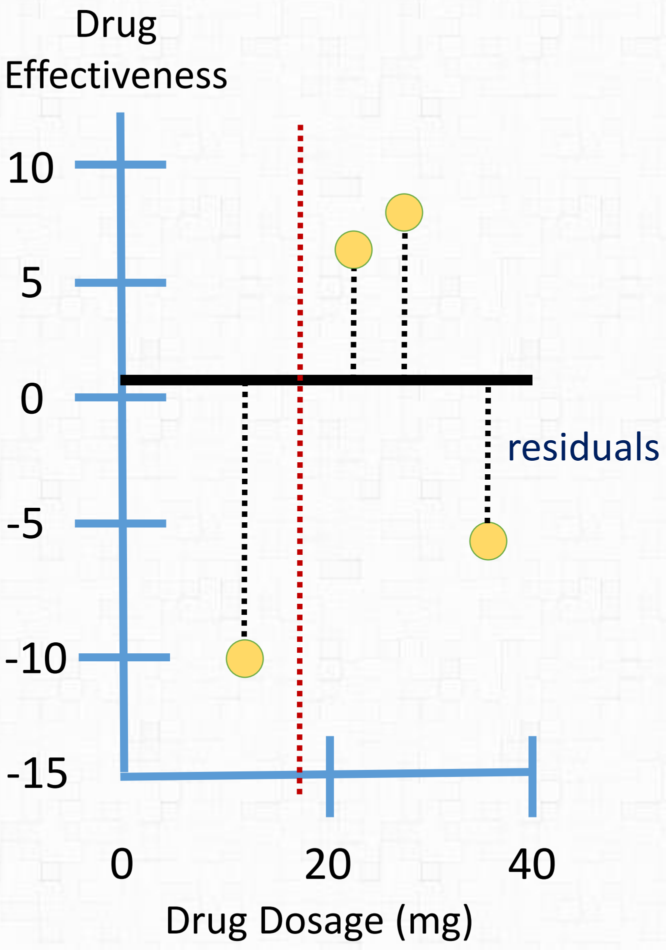
-7.5

$\text{Simliarity Score} = 4.08$

$\text{Simliarity Score} = 56.25$

$$\text{Gain} = \text{Leftsimiliar}_{\text{ity}} + \text{Rightsi}_{\text{miliarity}} - \text{Rootsi}_{\text{miliarity}} = 56.33$$

XGBoost for Regression



Initial Prediction: 0.5

Dosage < 15

$\text{Simliarity} = 4$

-10.5

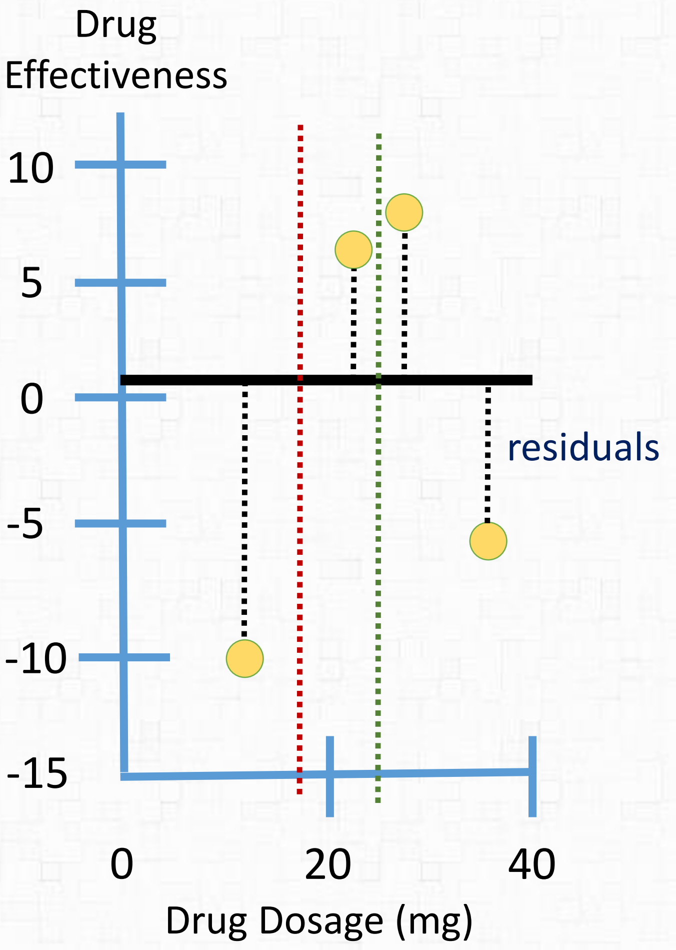
6.5, 7.5, -7.5

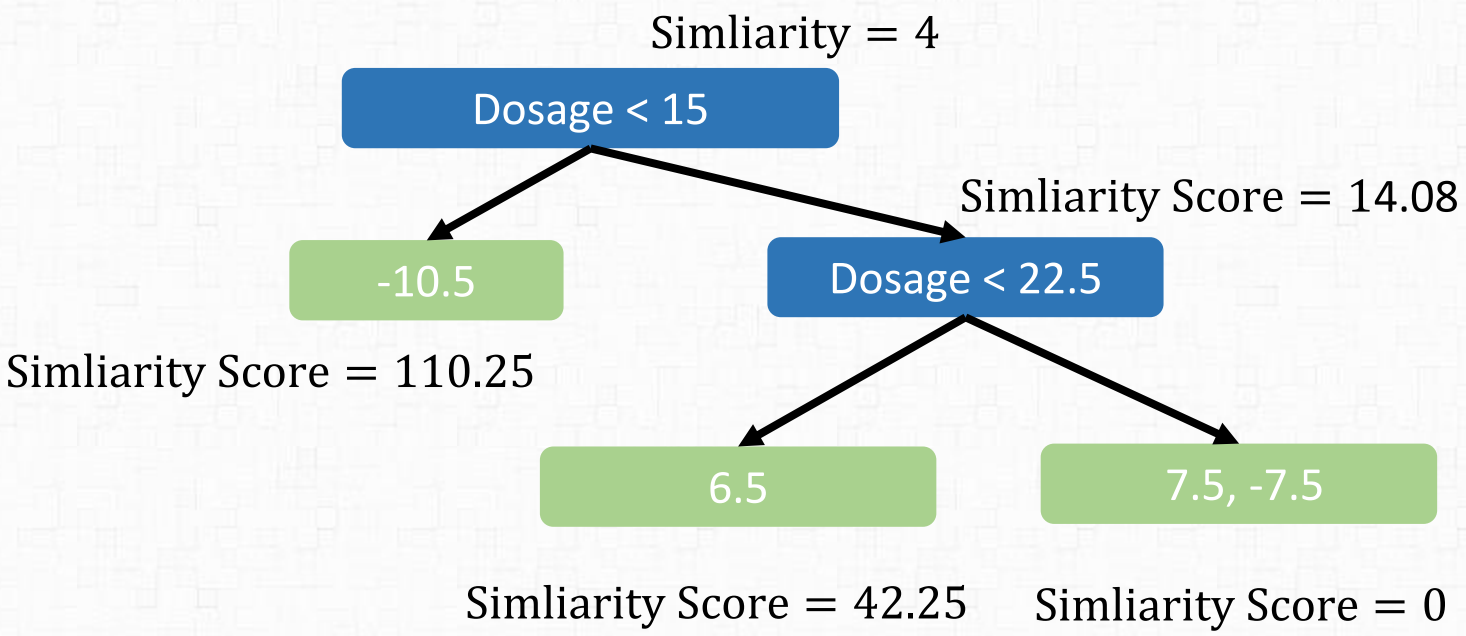
$\text{Simliarity Score} = 110.25$

$\text{Simliarity Score} = 14.08$

$\mathbf{Largest\ Gain, 120.33}$

XGBoost for Regression

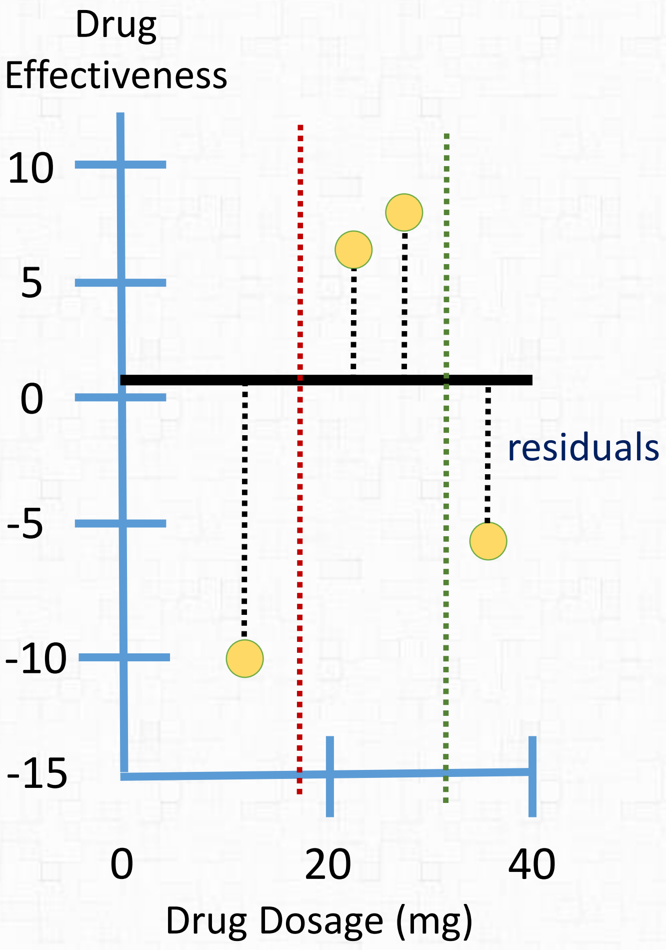


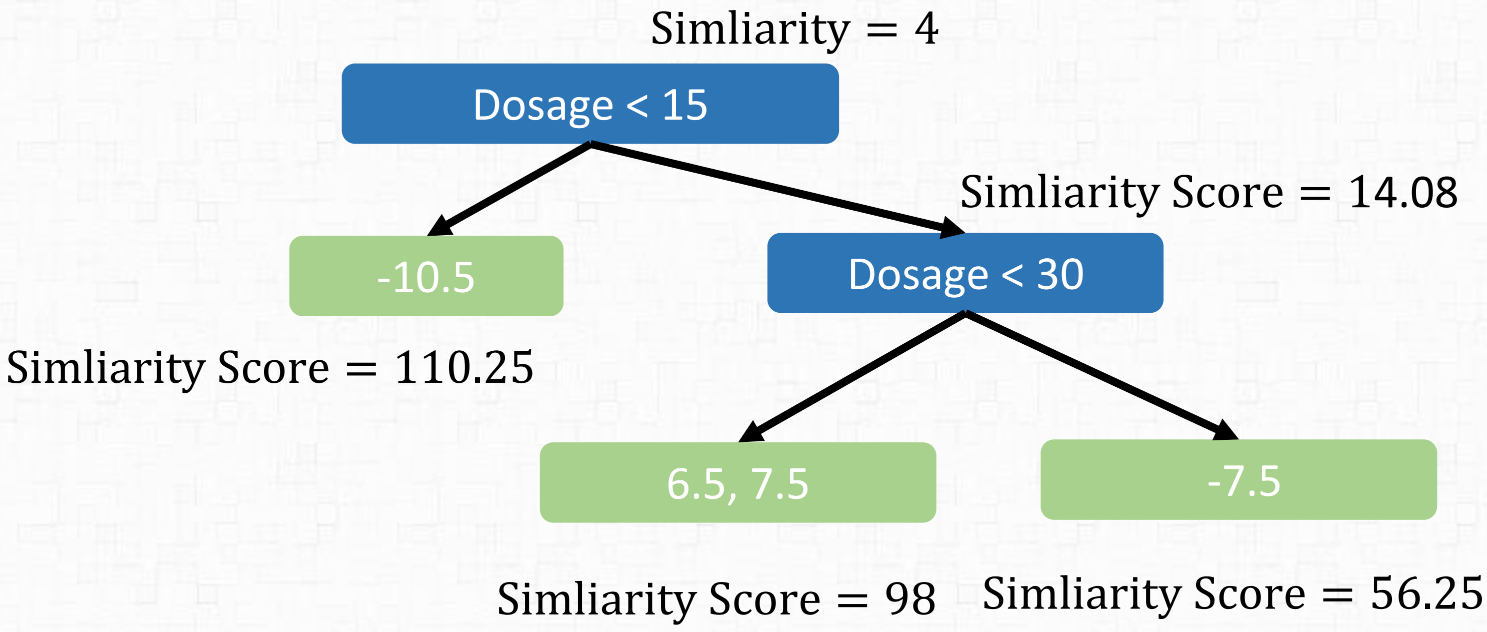


$$\text{Gain} = \text{Leftsimiliar}_{\text{ity}} + \text{Rightsi}_{\text{miliarity}} - \text{Rootsi}_{\text{miliarity}} =$$

$$42.25 + 0 - 14.08 = 28.17$$

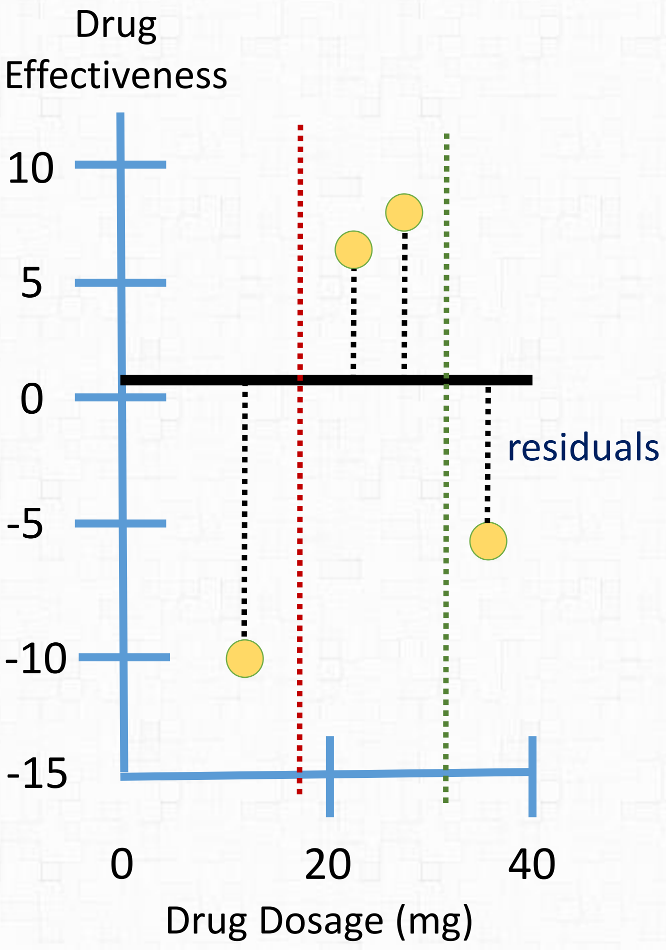
XGBoost for Regression

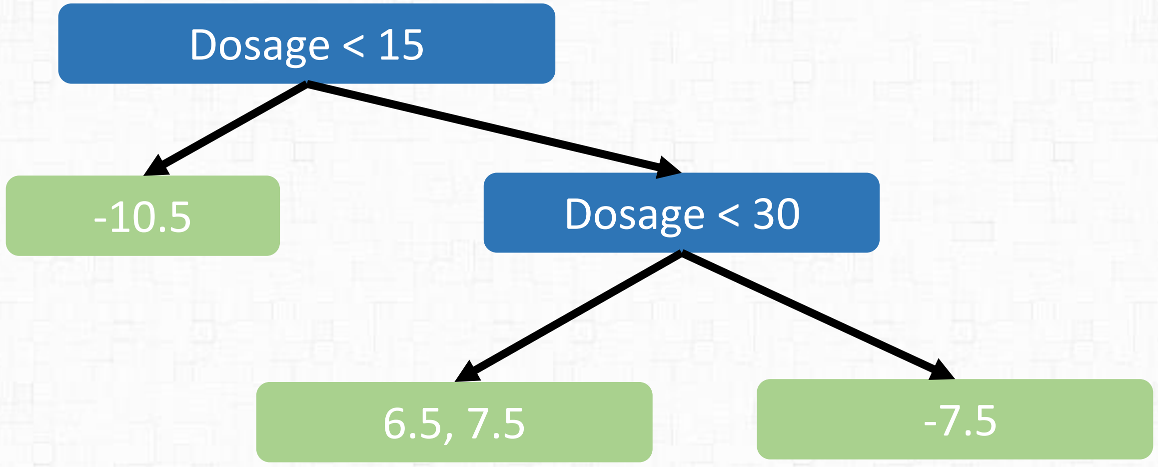




$$\text{Gain} = 98 + 56.25 - 14.08 = 140.17$$

XGBoost for Regression

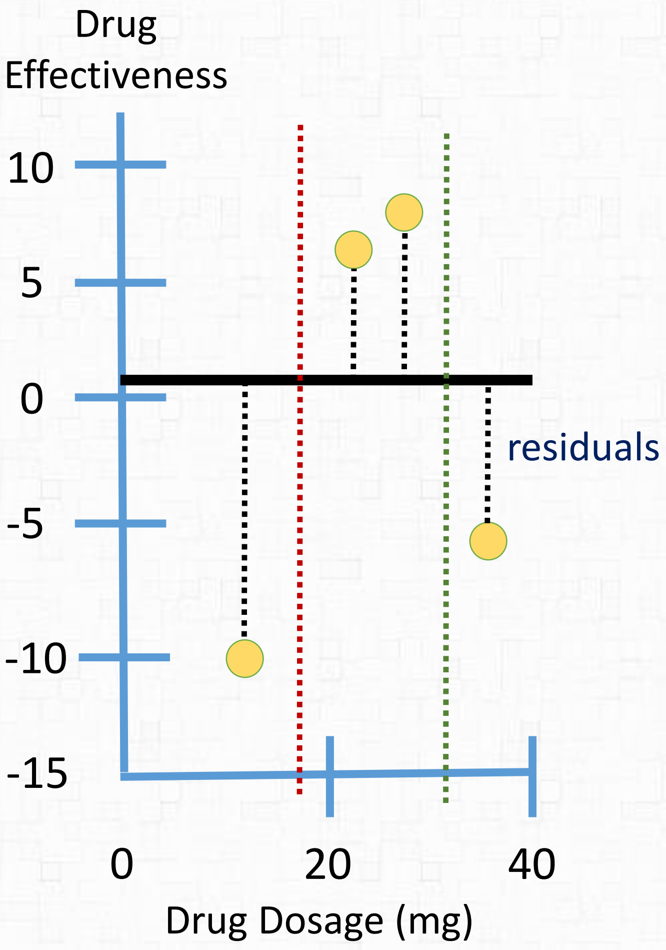


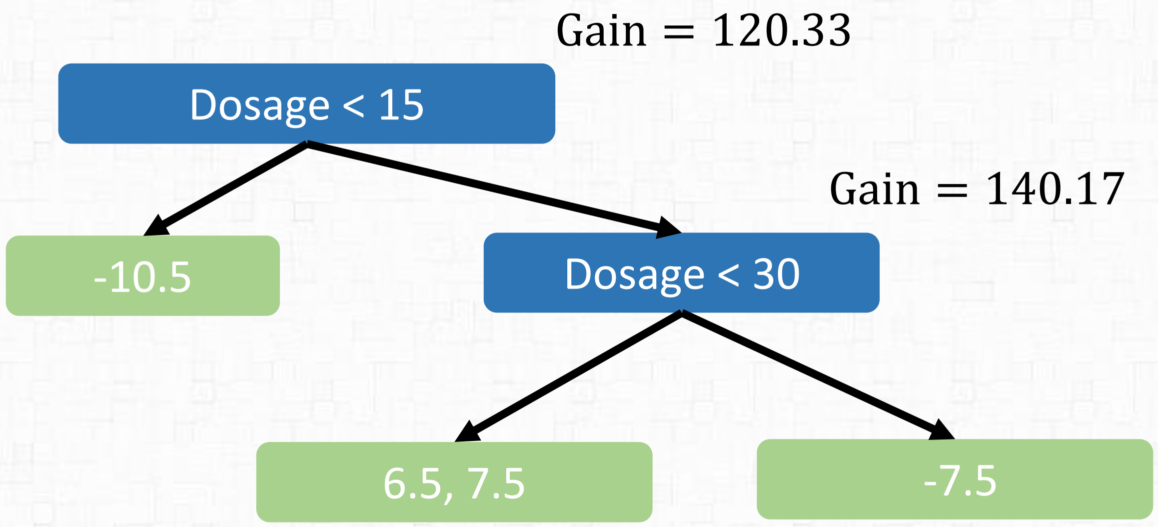


Limiting max depth of XGBoost Tree as 2,
We stop branching out the tree here.

By default, max depth is 6 levels

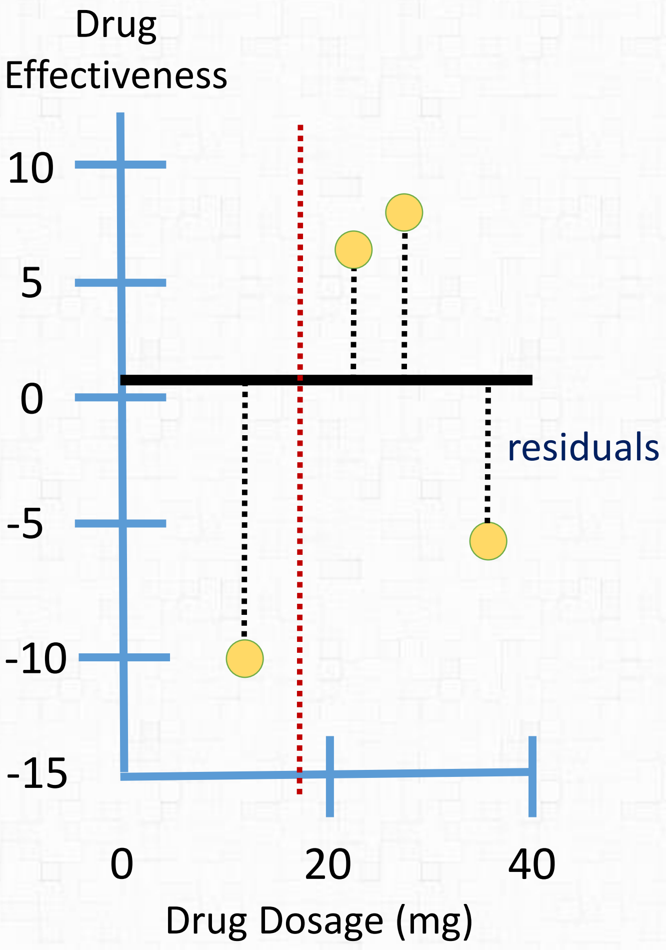
XGBoost for Regression

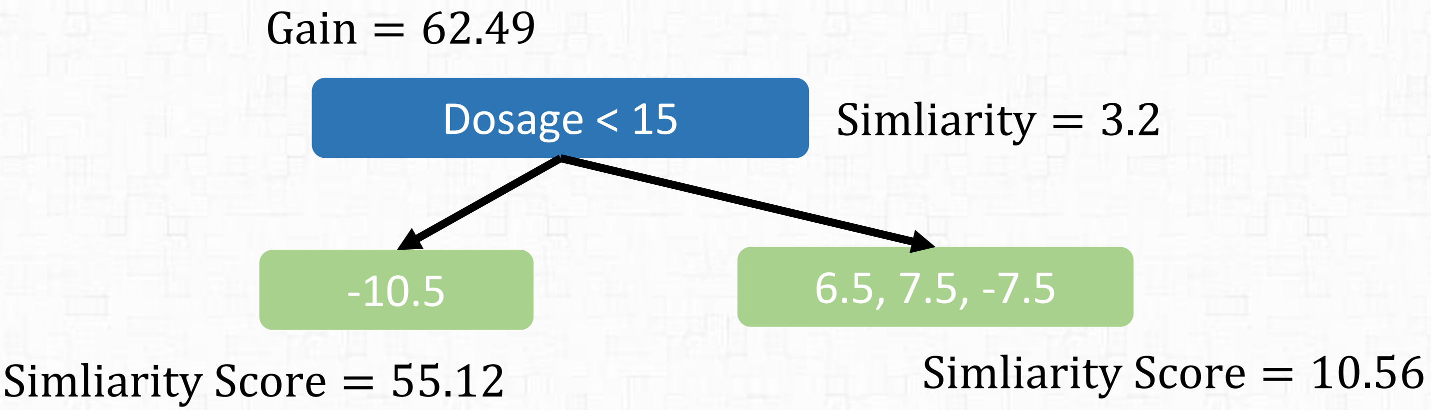




We prune the tree comparing Gain and γ(gamma)
(starting from leaf nodes)
γ is threshold for Gain
when the max gain is smaller than γ, we prune that branch

Regularization term – λ



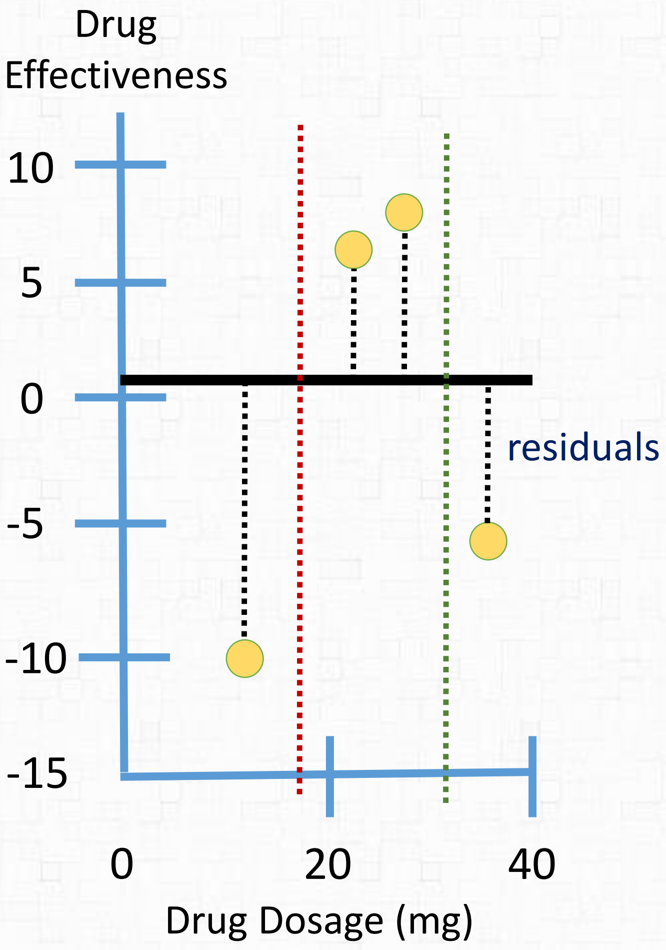


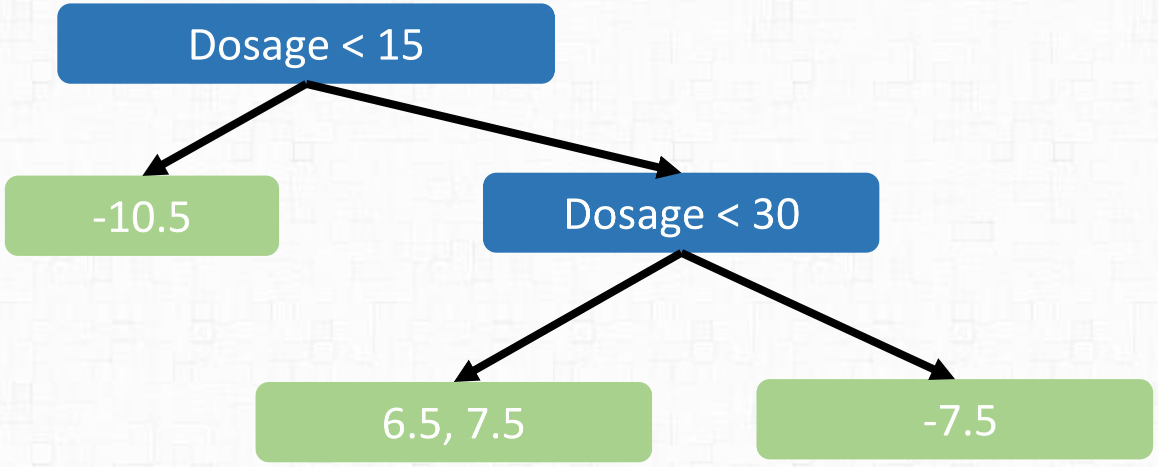
Now let's think about $\lambda > 0$
Let's assume λ = 1

For the smaller number of residuals, it decrease more of similarity score

Since it gives smaller Gain, Tree gets pruned more easily

XGBoost for Regression

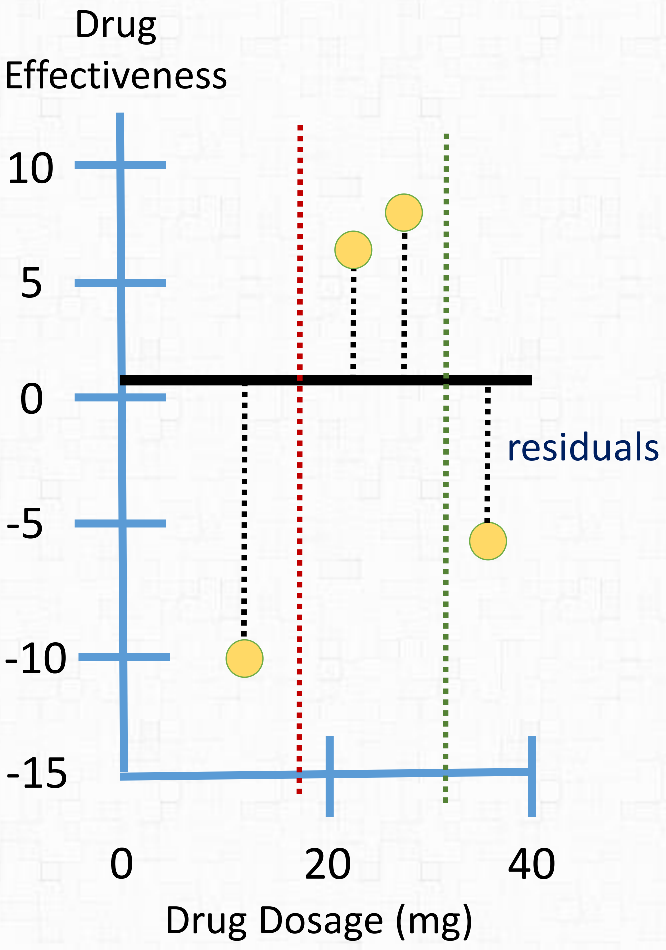


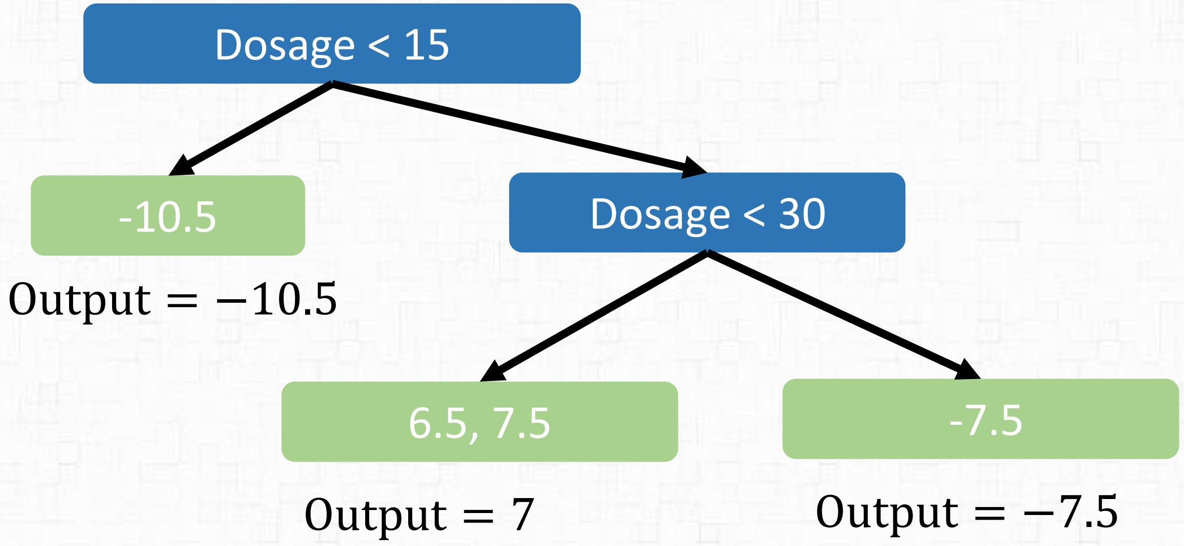


$$\text{Output Value} = \frac{\text{Sum of Residual}}{\text{Number of Residuals} + \lambda}$$

$$\text{Simliarity Score} = \frac{\text{Sum of Residual, Squared}}{\text{Number of Residuals} + \lambda}$$

XGBoost for Regression

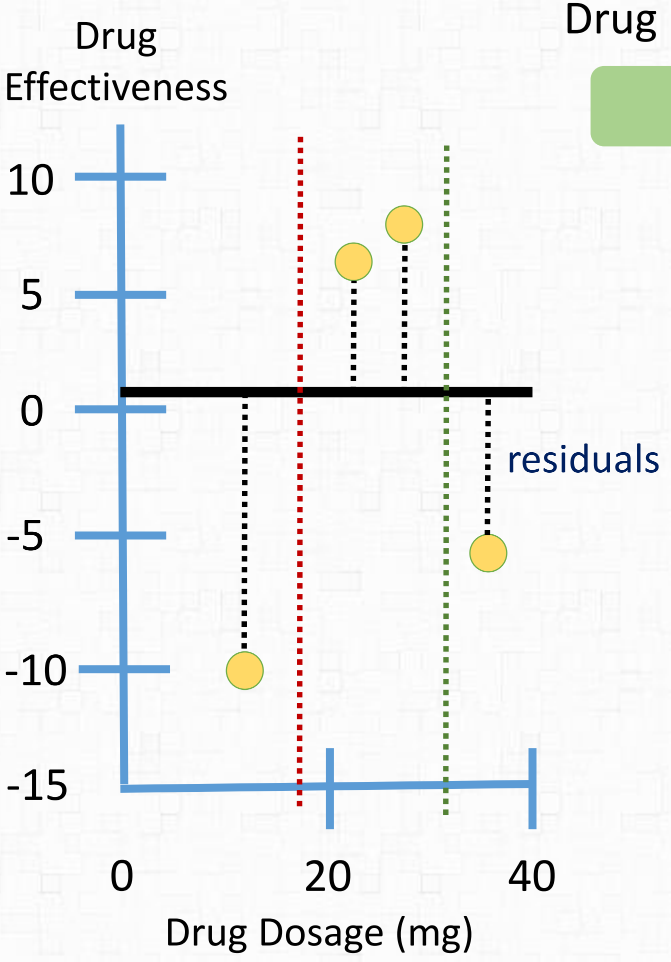




$$\text{Output Value} = \frac{\text{Sum of Residual}}{\text{Number of Residuals} + \lambda}$$

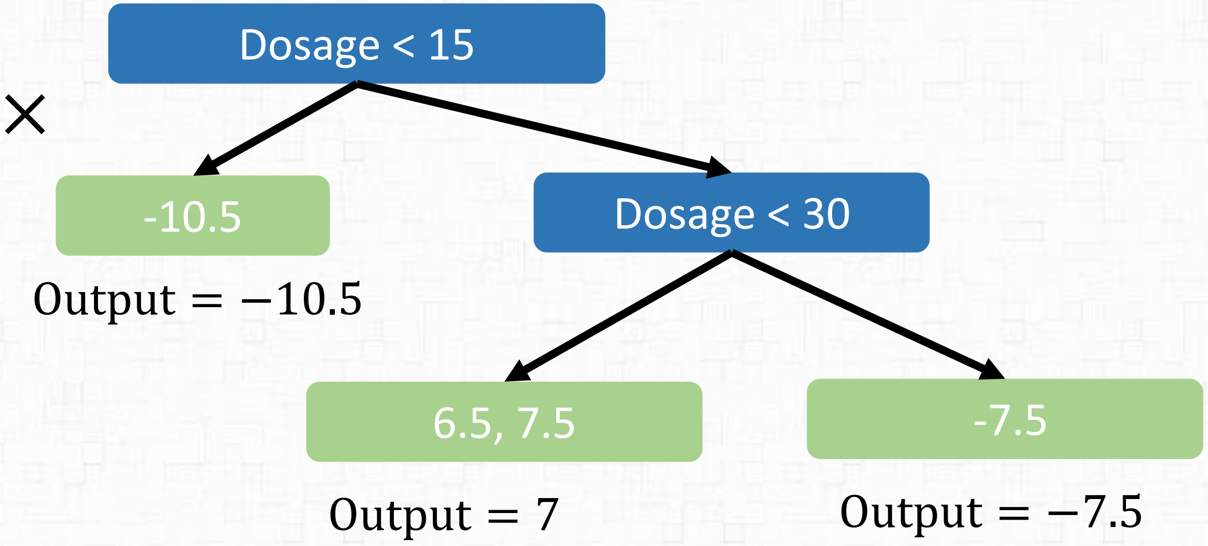
$$\text{Simliarity Score} = \frac{\text{Sum of Residual, Squared}}{\text{Number of Residuals} + \lambda}$$

XGBoost for Regression



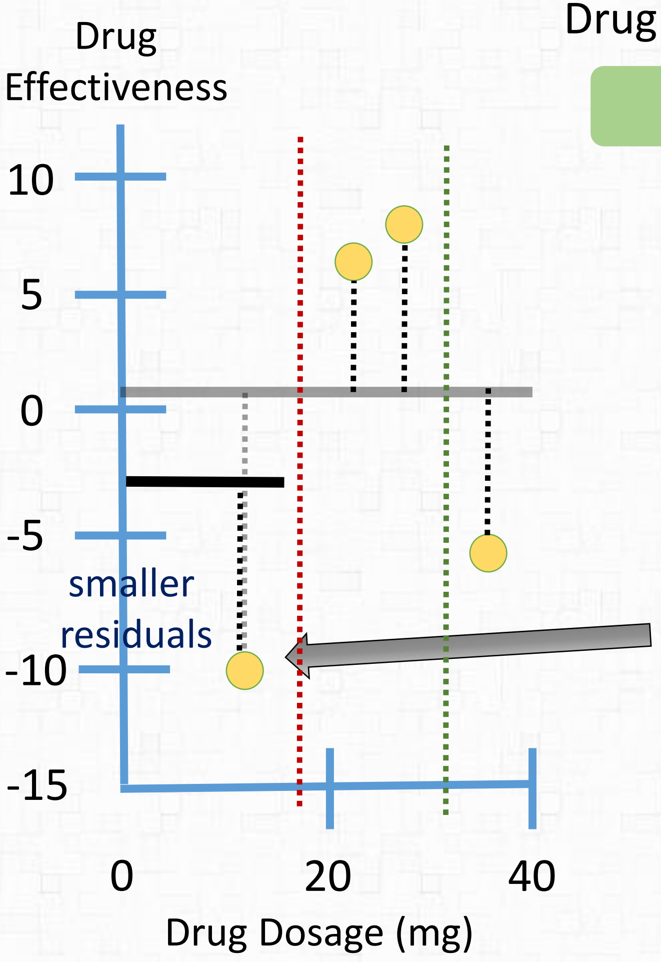
Predicted
Drug Effectiveness

0.5 + Learning Rate ×



Learning rate is called $\boldsymbol{\varepsilon}$(eta),
by default 0.3 In XGBoost

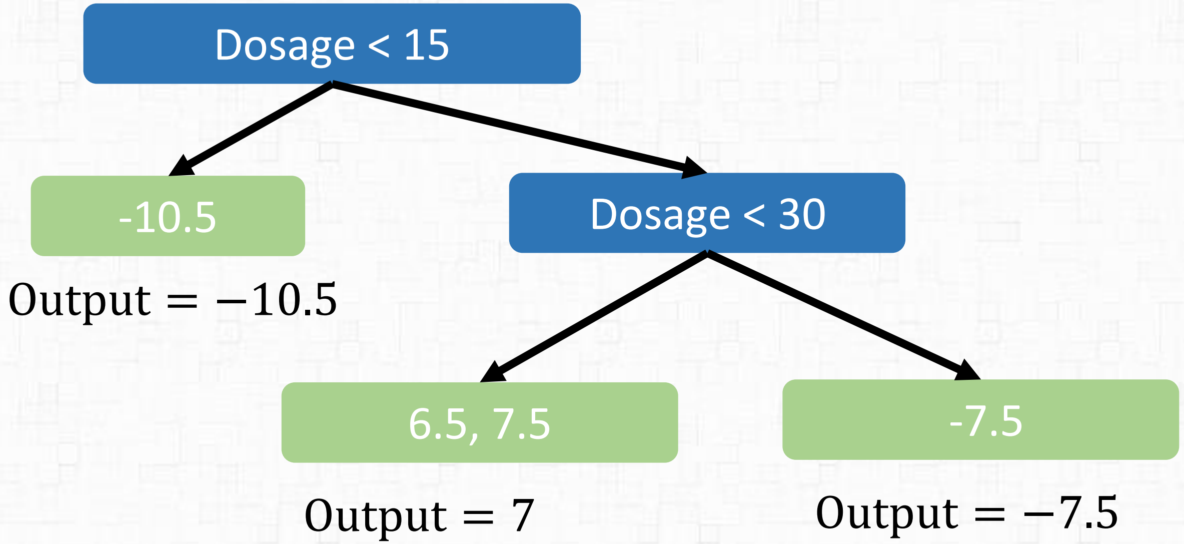
XGBoost for Regression



Predicted
Drug Effectiveness

0.5

+ 0.3 ×

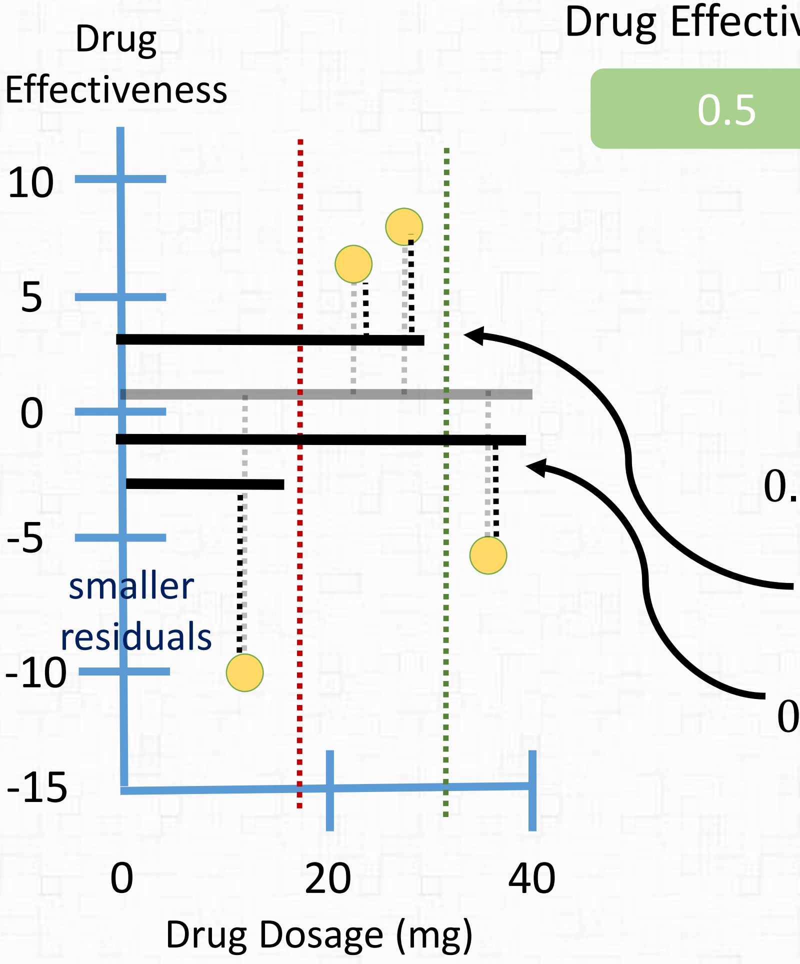


New predicted value is

$$0.5 + (0.3 \times -10.5) = -2.65$$

Learning rate is called $\boldsymbol{\varepsilon}$(eta), by default 0.3 In XGBoost

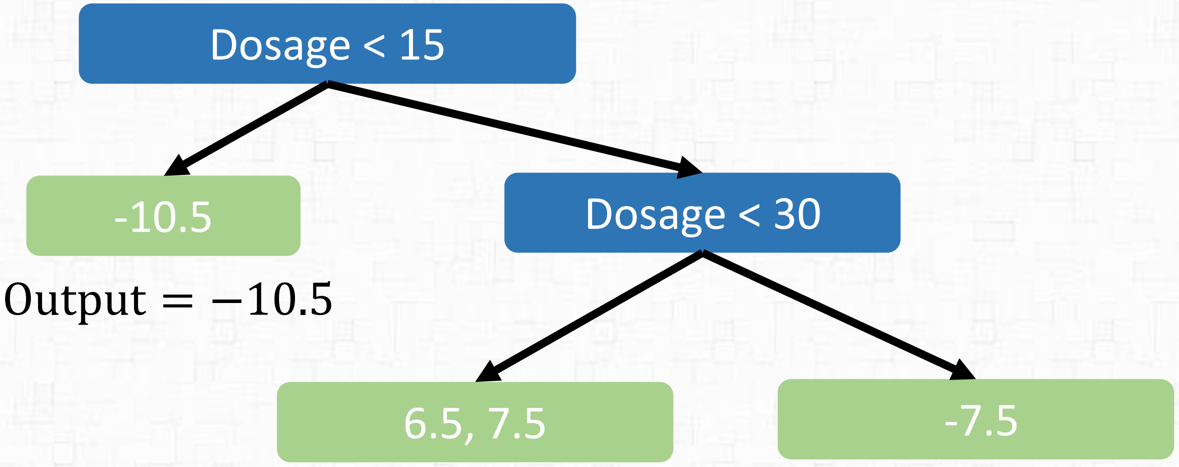
XGBoost for Regression



Predicted
Drug Effectiveness

0.5

+ 0.3 ×



$\text{Output} = -10.5$

$\text{Output} = 7$

$\text{Output} = -7.5$

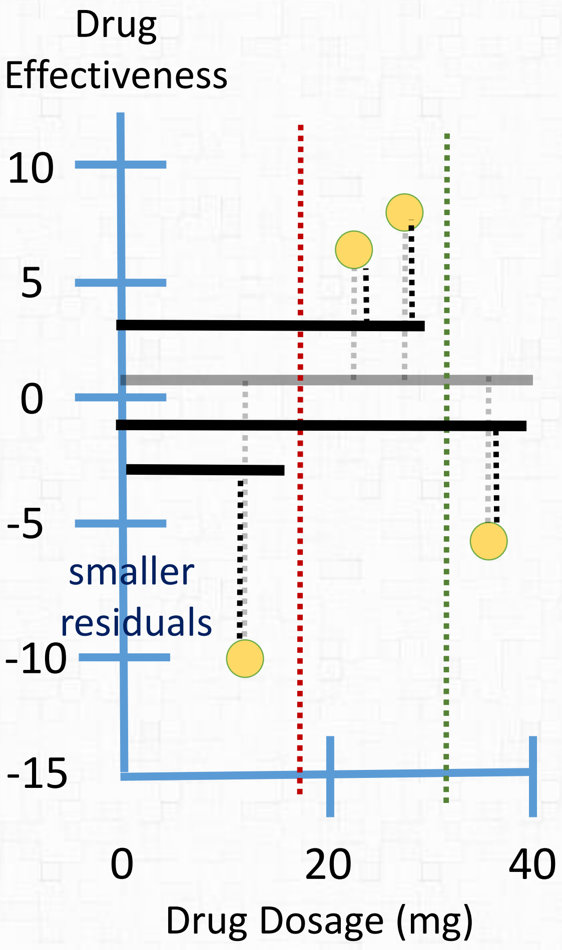
$0.5 + (0.3 \times -10.5) = -2.65$

$0.5 + (0.3 \times 7) = 2.6$

$0.5 + (0.3 \times -7.5) = -1.75$

Now we have smaller residuals in magnitude !!

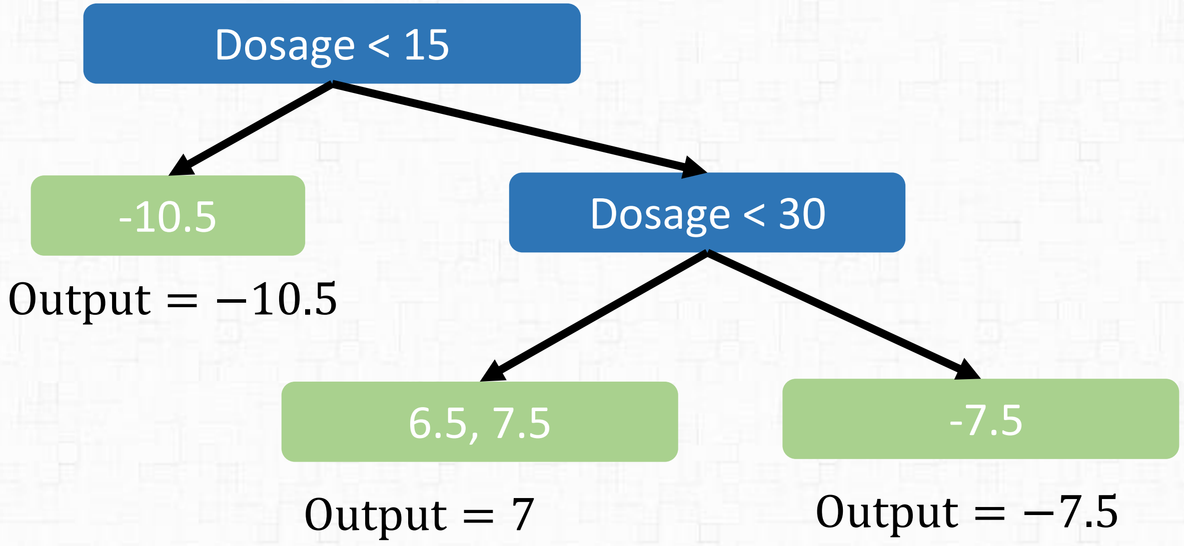
XGBoost for Regression



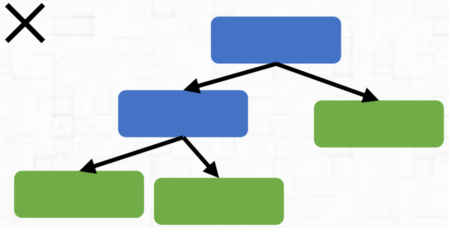
Predicted
Drug Effectiveness

0.5

$+ 0.3 \times$

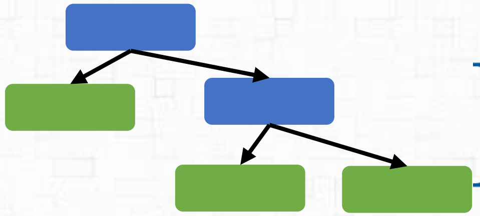


$+ 0.3 \times$



We keep building Trees
For the new residuals !!!

$+ 0.3 \times$



What to do next

- XGBoost for Classification, https://youtu.be/8b1JEDvenQU
- Mathematical details, https://youtu.be/ZVFeW798-2I
- Properties, https://youtu.be/oRrKeUCEbq8

References

- STATQUEST, https://statquest.org/