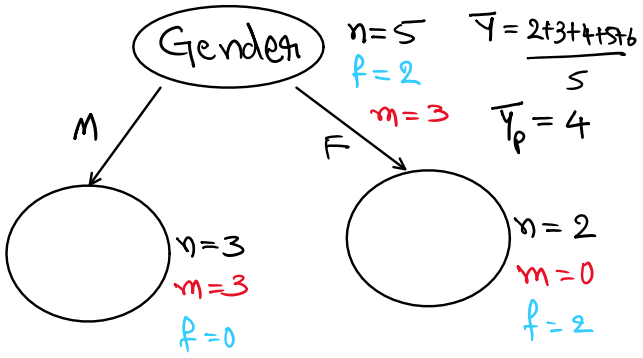
Bagging & Random Forest

13 March 2025 19:58

→ Decision Tree Regression

Gender	Edu	Salary
F	G	2
M	NG	3
F	NG	4
M	NG	5
M	G	6



$\bar{Y}_M = \frac{3+5+6}{3} = 4.67$ $\bar{Y}_F = \frac{2+4}{2} = 3$

→ If we use MSE as a metric, we will have these tables:-

MSE_M

Y	$\bar{Y}$
3	4.67
5	4.67
6	4.67

MSE_F

Y	$\bar{Y}$
2	3
4	3

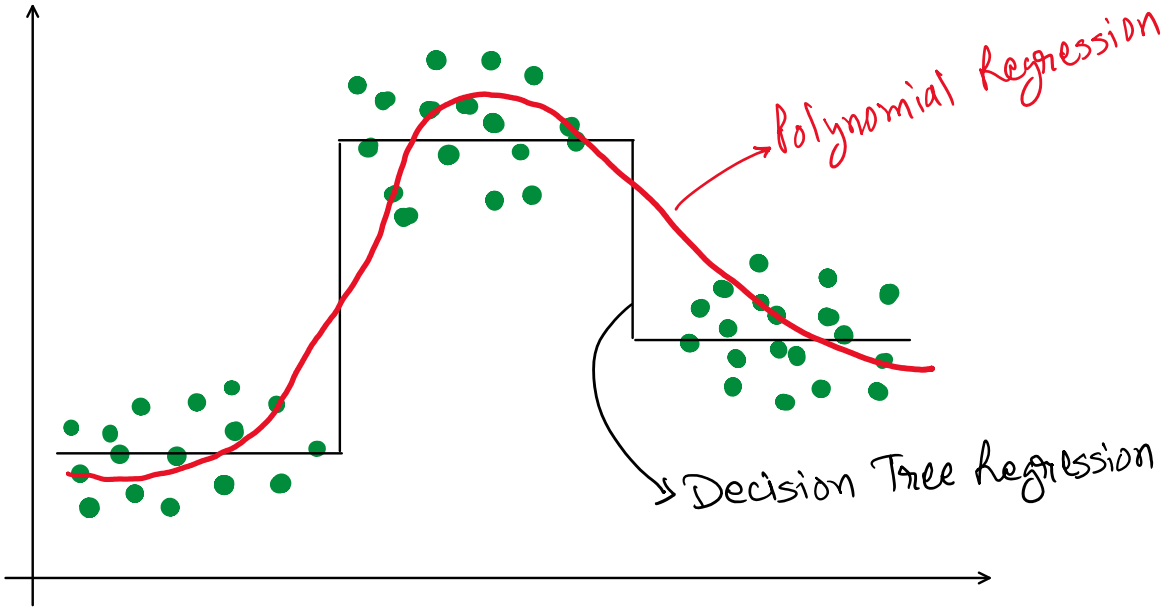
$$MSE_M = \frac{1}{3}\left[(3-4.67)^2 + (5-4.67)^2 + (6-4.67)^2\right]$$

$$MSE_F = \frac{1}{2}\left[(2-3)^2 + (4-3)^2\right]$$

$$MSE_C = \frac{3}{5} MSE_M + \frac{2}{5} MSE_F$$

$$MSE_P = \frac{1}{5}\left[(2-4)^2 + (3-4)^2 + (4-4)^2 + (5-4)^2 + (6-4)^2\right]$$

Visualization:



Problem: Can we get our model more accurate (accuracy > 90 - 95%) somehow?

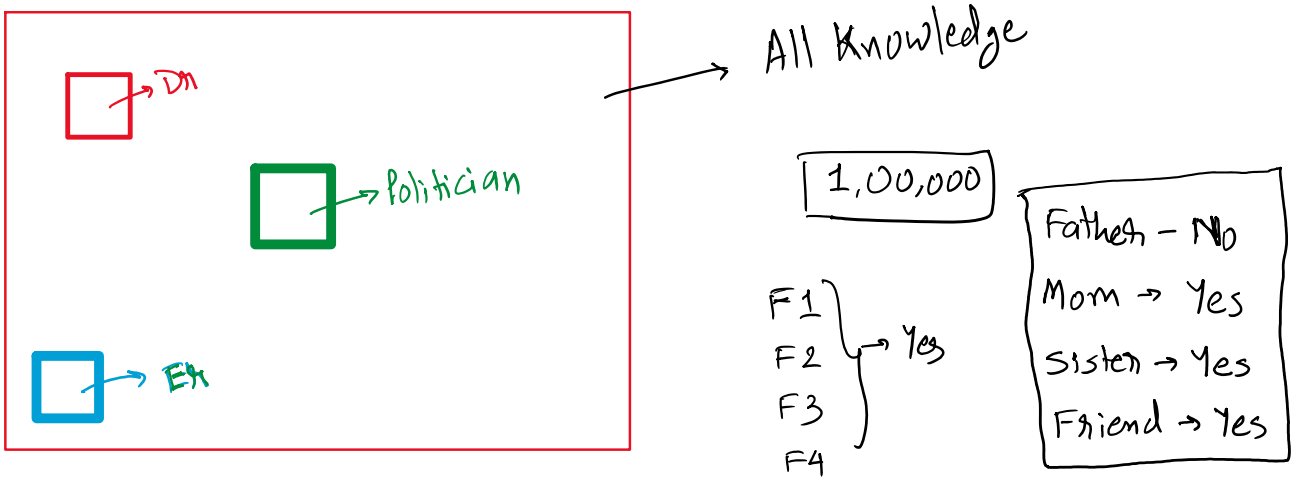
Suppose, we have a project on hand about waste water management. Which of these two teams will you choose for this purpose?

Team - 1: 6 members

1. Eng
2. Eng
3. Eng
4. Eng
5. Eng
6. Eng

Team - 2: 6 members

1. Eng
2. Doctor
3. Lawyer
4. Politician
5. Scientist
6. Finance Expert



Similarly, rather than training one single model on the entire data, can we train various models on small chunks of data which are "best" in their portion of data?

[m1, m2, … , m_k] => These models are called "Base Learners"
Where m1, m2, … can be any type of models e.g., m1 can be Logistic Regression, m2 may be KNN, m3 may be Decision Tree, m4 may be a different KNN, m5 may be another Decision Tree etc.
This type of machine learning is known as Ensemble Learning.
There are many techniques of Ensemble Learning but 4 of them are most popular, especially first two:

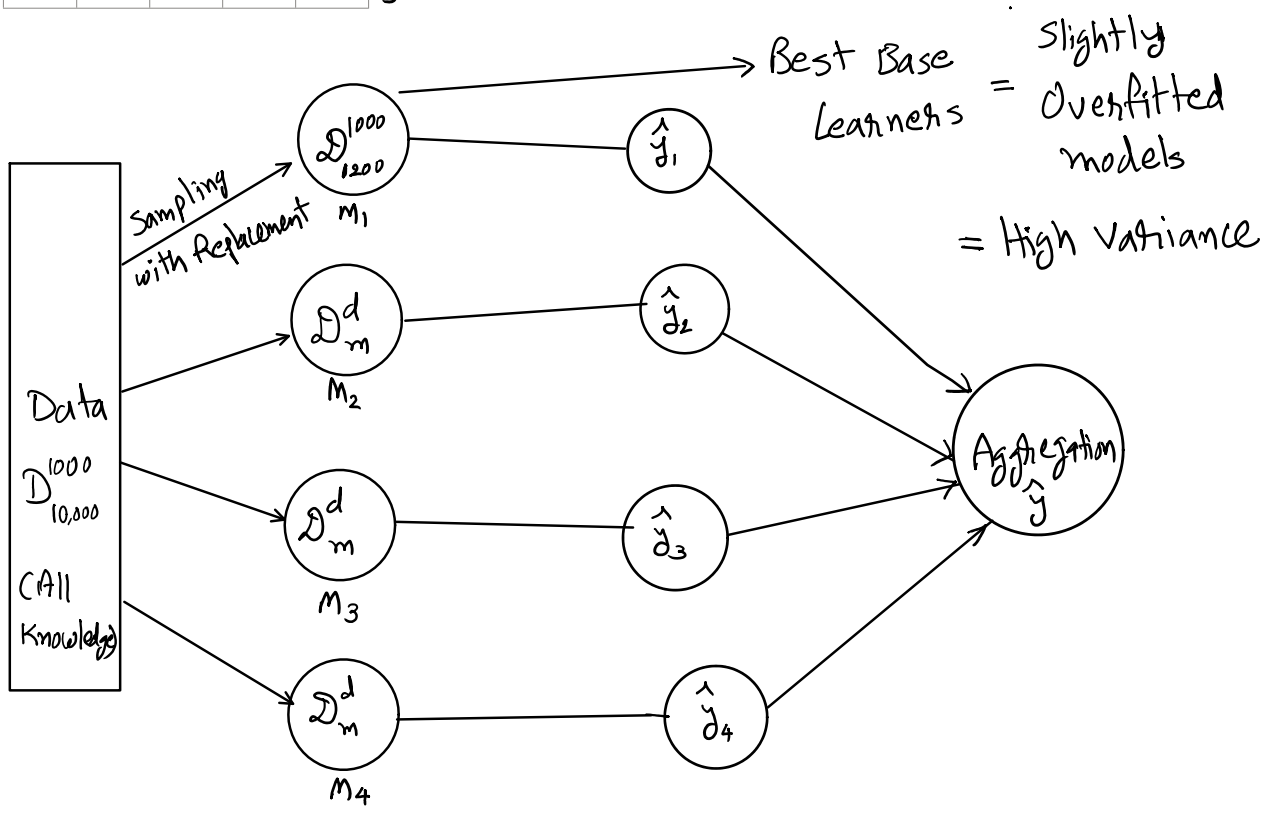
1. Bagging
2. Boosting
3. Stacking
4. Cascading

Bagging: Bootstrapped **Aggregation**

What was Bootstrapping?
Bootstrapping is sampling with replacement.

f1	f2	…	f_d	y

$\Rightarrow D_n^d$ (d: no. of features; n: no. of records)



Q: Then what type of aggregation will we use?
Ans: That depends on the type of the question.

If it is a classification problem (spam/not spam
M1 : -ve class
M2: +ve class
M3: -ve class
M4: -ve class
Solution: using Mode

If it's a regression problem (car price prediction)
M1: 5,50,000
M2: 6,70,000
M3: 4,90,000
M4: 7,10,000
Solution: using Mean

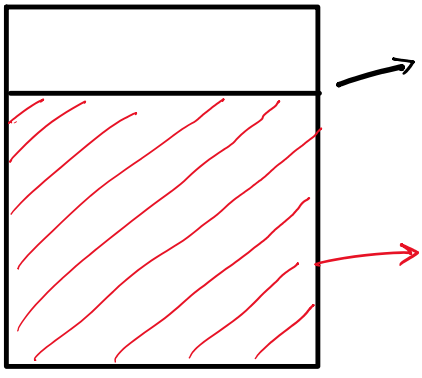
If these [m1, m2, m3, …, m_k] all are Decision Trees only then this Ensemble is known as **Random Forest.**

R.F. : Base Learners (DT) + Row Sampling of Dataset (with replacement) + Feature/Column sampling of Ds (Sampling w.r.t. features) (without replacement) + Aggregation

How can we validate (Hyper parameter tunning) Random Forest?

1. Tunning each base learner individually (we do not do this generally)
2. Tunning the entire (overall) model and not the base learners independently

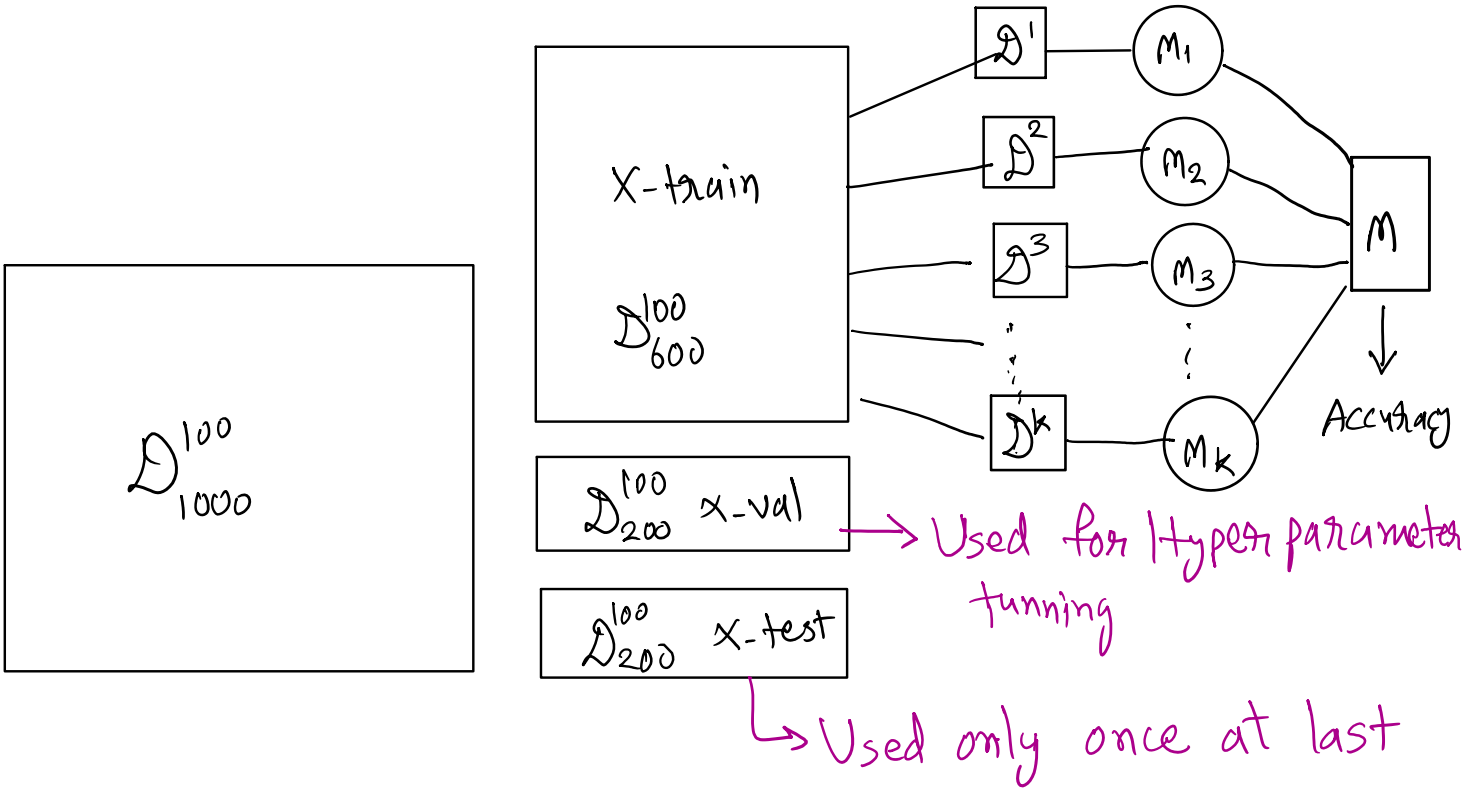
Tunning each base learner individually (we do not do this generally): We use **OOB points** as validation data



Sample data that is fed to a Base Learner 1. Can we say that we are creating a "Bag" of points on which our Base Learner 1 is trained?

Out of Bag data points (OOB points)

Tunning the entire (overall) model: max_depth = 7 , k = 100

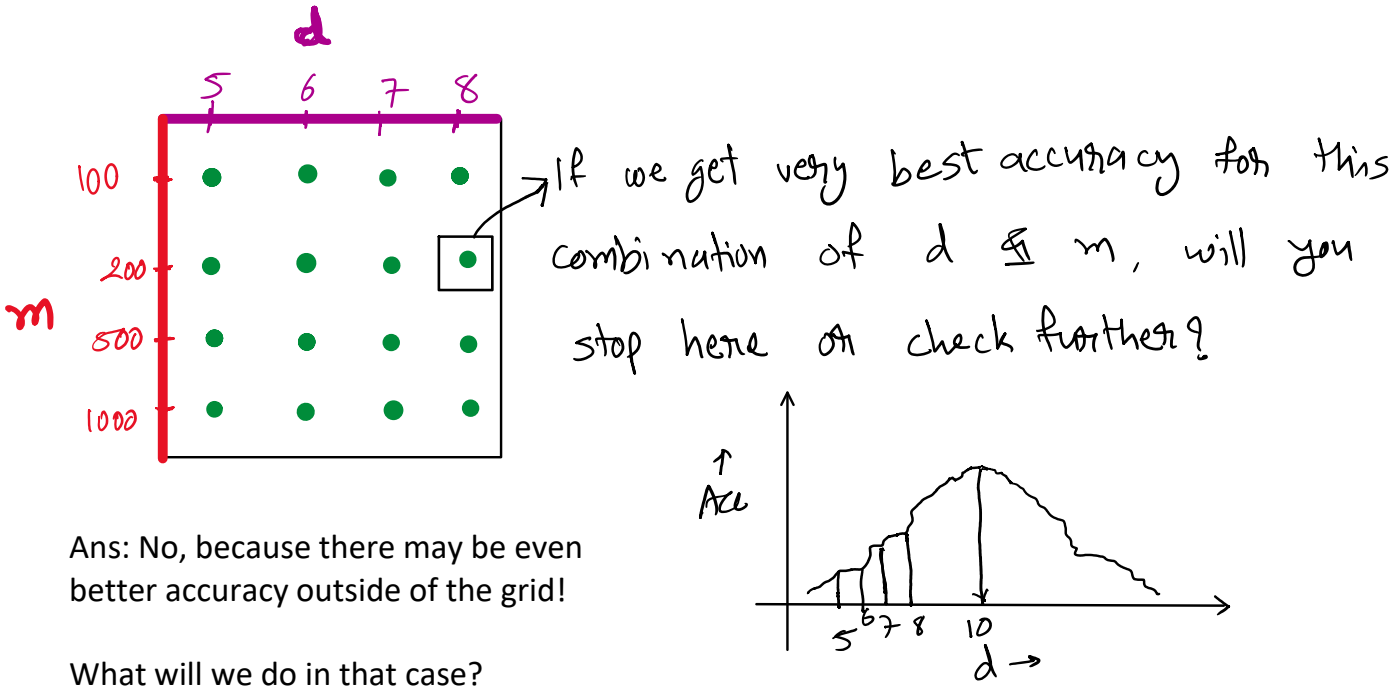


Hyperparameter Tunning:

Let's consider two Hyperparameters to tune- max_depth of each Base learner (call it 'd') and number of base learners (decision trees, say 'm') and d = [5, 6, 7, 8] & m = [100, 200, 500, 1000]

1. **Grid Search**
2. **Randomized Search**

Grid Search: We will make grid of all the possible combinations



Ans: No, because there may be even better accuracy outside of the grid!

What will we do in that case?
Ans: First we will see where that "best" accuracy is located. Is that point on the border of the grid or is it in the middle?

Case - 1: We get best accuracy on the border of our grid

Suppose in the above grid, we got best accuracy for m=200 & d=8. Therefore, it is clear that d=8 has higher accuracy than values 5, 6 or 7 hence we should search on the right side of d=8 that is, d=9, 10, 11 etc.

Similarly, if the best accuracy is at m=200 & d=5 then this value of d is already having higher accuracy value than d=6,7 or 8 hence, we look to the left side of d=5 that is, d=4, 3, 2 etc.

Now suppose we got best accuracy at m=100 & d=5 and we want to tune 'm' then it is already found that m=100 has better accuracy than m=200,500 & 1000 so we should look to the left side of m=100 (lower values of m like 80, 60, 50 etc.)

And if our best accuracy in the grid is at m=1000 & d=6 then we will look to the right side of m=1000 (more than 1000 that is, 1100, 1200, 1500 etc.) for even higher accuracy.

In short, **if we get best accuracy on the border of our grid then we continue searching for higher accuracy on left or right of the present values.**

Case - 2: We get best accuracy in our grid somewhere in the middle (not on the border)

Let's say we got the best accuracy in our grid at m=200 & d=5 and we want to fine tune 'm'. Now it is clear that m=200 has better accuracy than m=100 or m=500 but as we do not

know the distribution of accuracy between 100 & 500, it makes sense to look around m= 200 for better accuracy. So in a way, we zoom into this region to find better accuracy. and we calculate accuracy at m=190, m=210, m=220 etc. points.

In short, **if we get best accuracy somewhere in the middle of our grid (not on the borders) then we zoom into that region to look for better accuracy around that value of that hyperparameter.**