multi-model

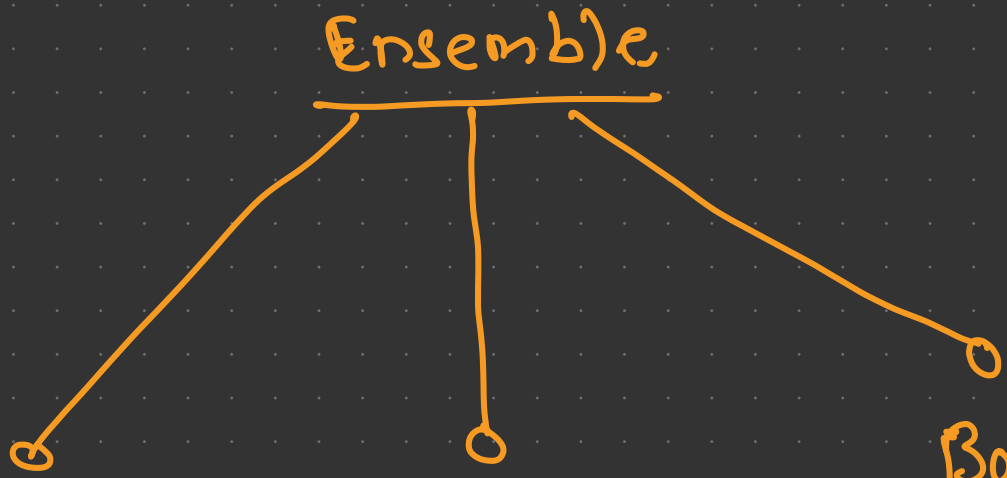
Ensemble Learning

dependent on multiple models

Overview

↳ model → algorithm

- Ensemble is the art of combining diverse set of learners (individual models) together to improvise on the stability and predictive power of the model
- Primarily used to improve the (classification, prediction, function approximation, etc.) performance of a model, or reduce the likelihood of an unfortunate selection of a poor one
- Other applications of ensemble learning include assigning a confidence to the decision made by the model, selecting optimal (or near optimal) features, data fusion, incremental learning, nonstationary learning and error-correcting



Bagging

- improves stability
- improves confidence
- Random forest

DT

Stacking

Boosting

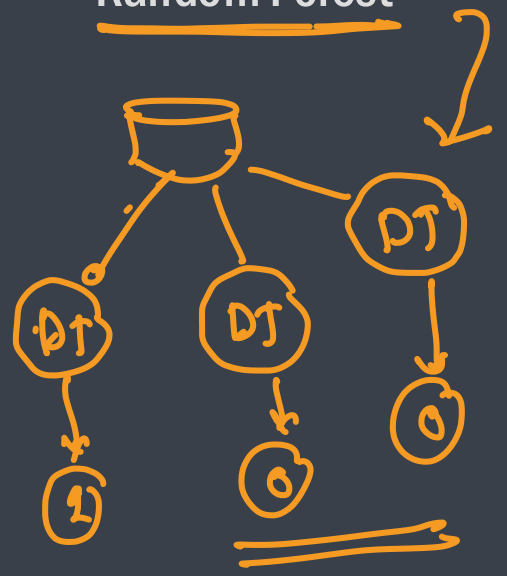
- improves accuracy
- Gradient Boost
- CatBoost
- AdaBoost
- XGBoost ***

Bagging

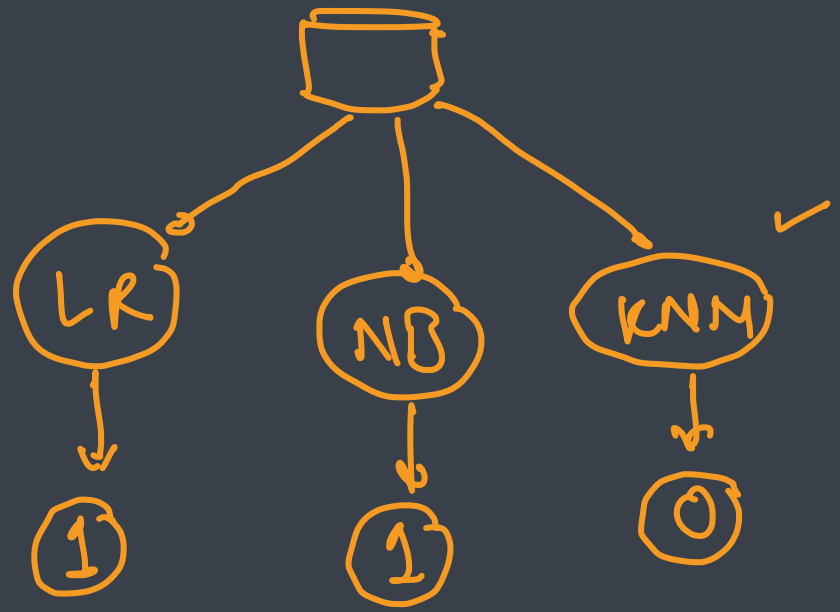
improving stability

Bagging

- Bagging tries to implement similar learners on small sample populations and then takes a mean of all the predictions
- In generalized bagging, you can use different learners on different population
- This helps us to reduce the variance error
- Algorithm
  - Random Forest



Similar learners

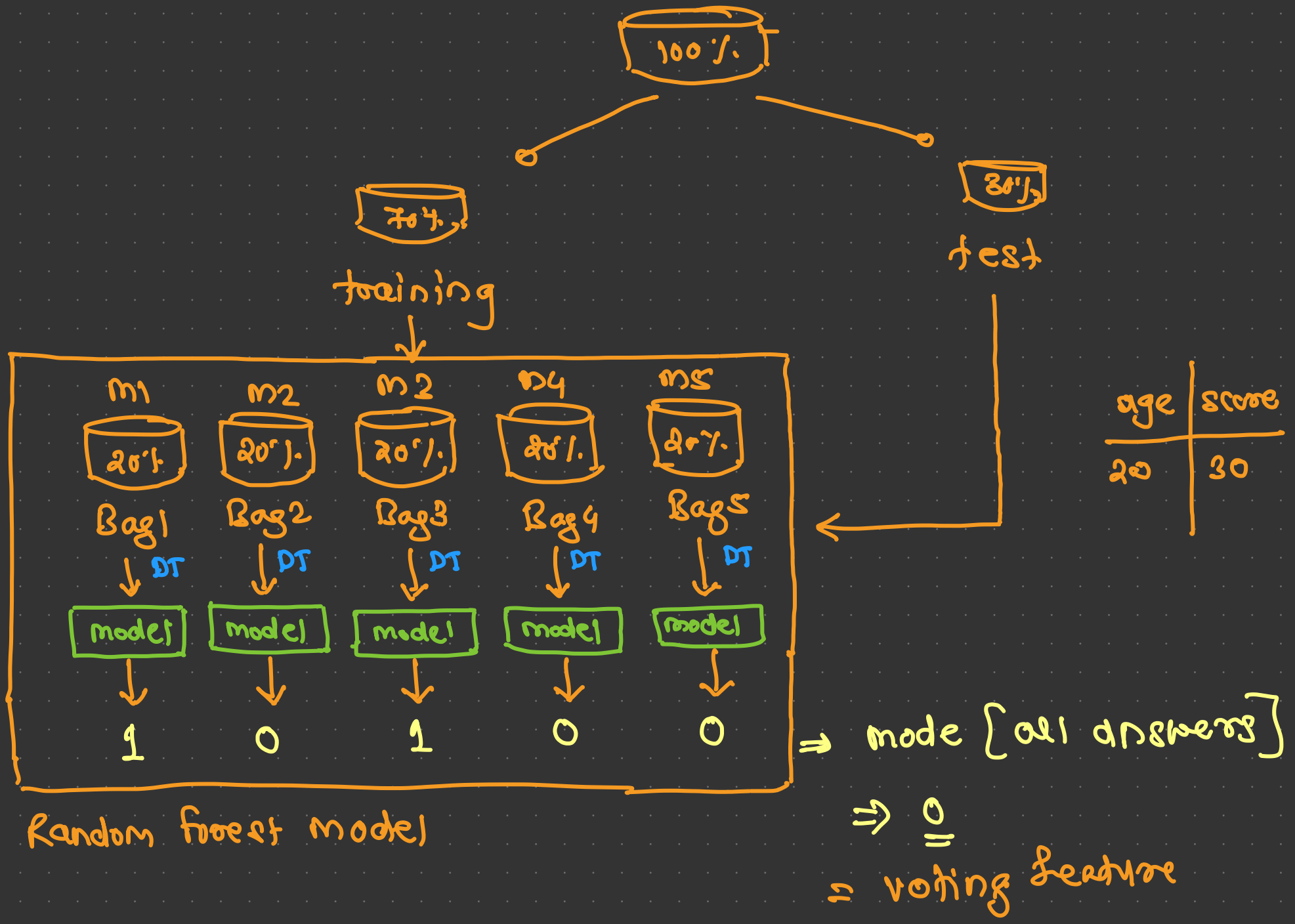


different learner

Regression = mean

classification = mode

= 1


100 %.
70%.
training
30%.
test
m1
m2
m3
m4
m5
20%.
20%.
20%.
40%.
20%.
Bag1
Bag2
Bag3
Bag4
Bag5
DT
DT
DT
DT
DT
model
model
model
model
model
1
0
1
0
0
Random forest model
age
score
20
30
⇒ mode [all answers]
⇒ 0
= voting feature

split training data into Bags

x1	x2	x3
20	10	5
30	5	3
40	9	8
50	7	2

data

* Vertically
  - x1 and x2
  - x2 and x3

Column based splitting

* horizonally

x1	x2	x3
20	10	5
30	5	3

x1	x2	x3
40	9	8
50	7	2

row based spliting

Random Forest → Trees

- Random forest classifier creates a set of decision trees from randomly selected subset of training set
- It then aggregates the votes from different decision trees to decide the final class of the test object
- This works well because a single decision tree may be prone to a noise, but aggregate of many decision trees reduce the effect of noise giving more accurate results

How does it work?

- Suppose training set is given as : [X1, X2, X3, X4] with corresponding labels as [L1, L2, L3, L4], random forest may create three decision trees taking input of subset for example,

[X1, X2, X3]
[X1, X2, X4]
[X2, X3, X4]

vertical

- So finally, it predicts based on the majority of votes from each of the decision trees made

Case Study

- A certain country has a population of 118 million
- Salary data is collected with following attributes
  - Age, Gender, Education, Residence, Industry
- Salary bands :
  - Band 1 : Below $40,000
  - Band 2: $40,000 – 150,000
  - Band 3: More than $150,000

Case Study

- Following are the outputs of the 5 different CART model.

Tree 1

	Salary Band	1	2	3
Age	Below 18	90%	10%	0%
	19-27	85%	14%	1%
	28-40	70%	23%	7%
	40-55	60%	35%	5%
	More than 55	70%	25%	5%

Tree 2

	Salary Band	1	2	3
Gender	Male	70%	27%	3%
	Female	75%	24%	1%

Tree 3

	Salary Band	1	2	3
Education	<=High School	85%	10%	5%
	Diploma	80%	14%	6%
	Bachelors	77%	23%	0%
	Master	62%	35%	3%

Tree 4

	Salary Band	1	2	3
Residence	Metro	70%	20%	10%
	Non-Metro	65%	20%	15%

Tree 5

	Salary Band	1	2	3
Industry	Finance	65%	30%	5%
	Manufacturing	60%	35%	5%
	Others	75%	20%	5%

Case Study

- Using these 5 CART models, we need to come up with single set of probability to belong to each of the salary classes
- For simplicity, we will just take a mean of probabilities in this case study. Other than simple mean, we also consider vote method to come up with the final prediction.
- To come up with the final prediction let's locate the following profile in each CART model :
  - 1. Age : 35 years
  - 2. Gender : Male
  - 3. Highest Educational Qualification : Diploma holder
  - 4. Industry : Manufacturing
  - 5. Residence : Metro

Case Study

- For each of these CART model, following is the distribution across salary bands :

CART	Band	1	2	3
Age	28-40	70%	23%	7%
Gender	Male	70%	27%	3%
Education	Diploma	80%	14%	6%
Industry	Manufacturing	60%	35%	5%
Residence	Metro	70%	20%	10%
Final probability		70%	24%	6%

- The final probability is simply the average of the probability in the same salary bands in different CART models
- As you can see from this analysis, that there is 70% chance of this individual falling in class 1 (less than $40,000) and around 24% chance of the individual falling in class 2

Advantages of Random Forest algorithm

tree is very complex = too many level

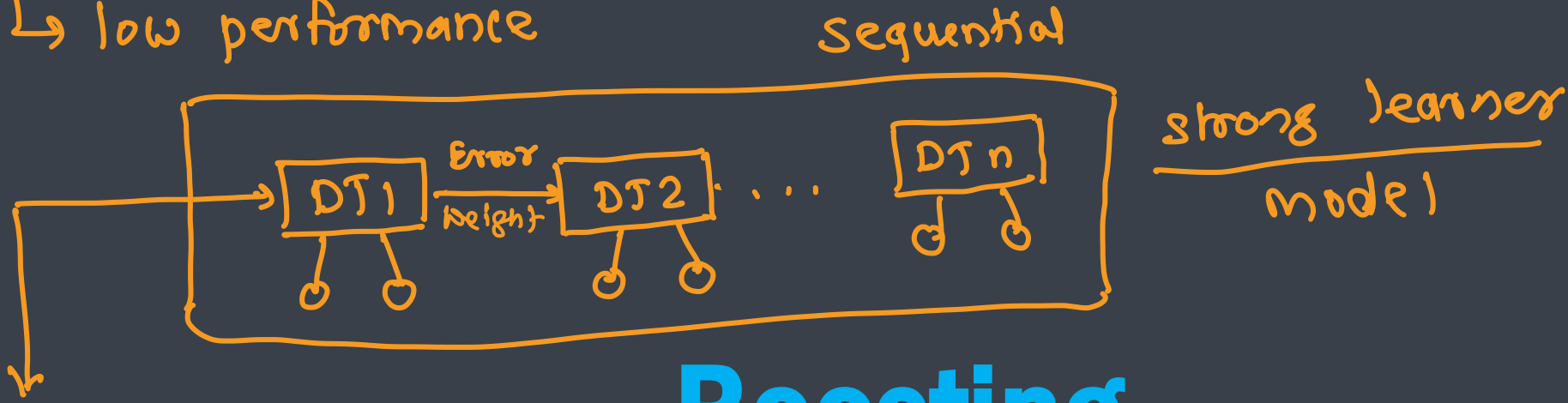
- For applications in classification problems, Random Forest algorithm will avoid the overfitting problem
- For both classification and regression task, the same random forest algorithm can be used
- The Random Forest algorithm can be used for identifying the most important features from the training dataset, in other words, feature engineering.

Applications of Random Forest → for all types of classification

- For the application in banking, Random Forest algorithm is used to find loyal customers, which means customers who can take out plenty of loans and pay interest to the bank properly, and fraud customers, which means customers who have bad records like failure to pay back a loan on time or have dangerous actions.
- For the application in medicine, Random Forest algorithm can be used to both identify the correct combination of components in medicine, and to identify diseases by analyzing the patient's medical records.
- For the application in the stock market, Random Forest algorithm can be used to identify a stock's behavior and the expected loss or profit.
- For the application in e-commerce, Random Forest algorithm can be used for predicting whether the customer will like the recommend products, based on the experience of similar customers.

Weak learner → model which has NOT learned much from training

↳ low performance



Decision tree stump

Depth = 1

Boosting

improves accuracy

Weak learners → strong learner

models

Decision tree

Boosting

$y = \beta_1 x_1 + \beta_2 x_2 \ldots \beta_n x_n + \beta_0$

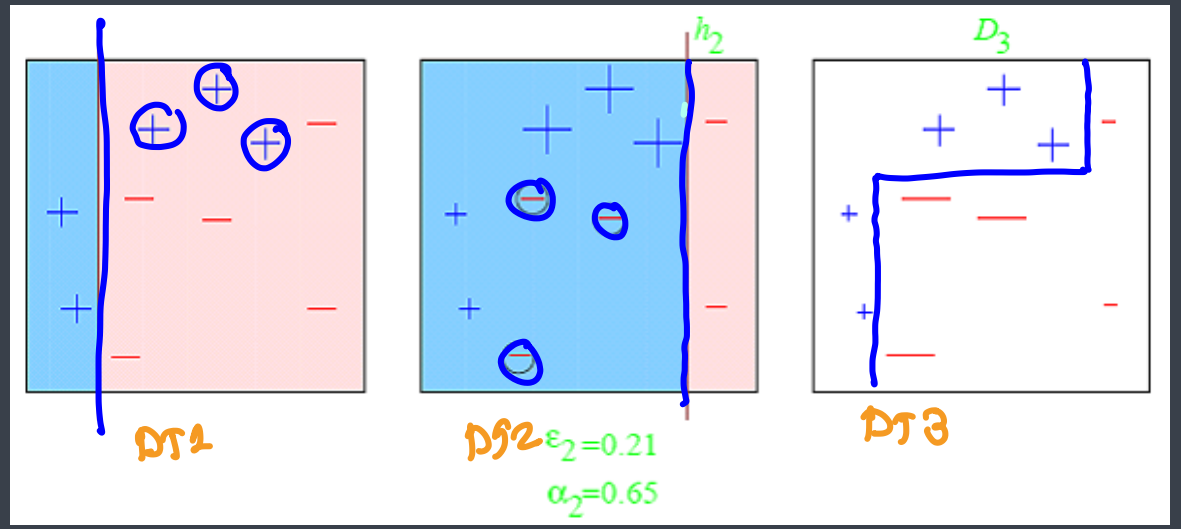
high error / low accuracy ⇒ low error / high accuracy

- Boosting refers to a family of algorithms that are able to convert weak learners to strong learners
- Boosting is an iterative technique which adjust the weight of an observation based on the last classification
- If an observation was classified incorrectly, it tries to increase the weight of this observation and vice versa
- Boosting in general decreases the bias error and builds strong predictive models
- Algorithms (Ag Boost)
  - AdaBoost
  - Gradient Boosting
  - eXtreme Gradient Boosting

$f = \alpha_1(m_1) + \alpha_2(m_2) + \ldots + \alpha_n(m_n)$

$m_1, m_2 \ldots m_n \rightarrow$ models (weak learners) — Decision Tree stump

$\alpha_1, \alpha_2 \ldots \alpha_n \rightarrow$ weights



CatBoost — very less or no data cleansing

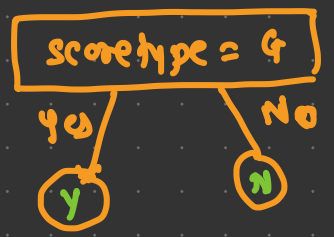
- CatBoost or Categorical Boosting is an open-source boosting library developed by Yandex
- It is designed for use on problems like regression and classification having a very large number of independent features
- Catboost is a variant of gradient boosting that can handle both categorical and numerical features
- It does not require any feature encodings techniques like One-Hot Encoder or Label Encoder to convert categorical features into numerical features → Decision Tree
- It also uses an algorithm called symmetric weighted quantile sketch(SWQS) which automatically handles the missing values in the dataset to reduce overfitting and improve the overall performance of the dataset

CatBoost Features

- Built-in Method for handling categorical features
  - CatBoost can handle categorical features without any feature encoding
- Built-in methods for Handling missing values
  - Unlike other Models, CatBoost can easily handle any missing values in the dataset
- Automatic feature scaling
  - CatBoost internal scales all the columns to the same scaling whereas in other models we need to convert columns extensively
- Built-in cross-validation
  - CatBoost internally applies a cross-validation method to choose the best hyperparameters for the model.
- Regularizations
  - CatBoost supports both L1 and L2 regularization methods to reduce overfitting
- It can be used in both Python and R language

x_1 Salary	x_2 Score type	y approved	$\hat{y}$	weight	
≥ 50k	A	Y	Y	1/4	✓
≥ 50k	B	N	N	1/4	✓
< 50k	A	N	Y	1/4	✗
< 50k	—	Y	N	1/4	✗

Decision Tree Stump



1 yes / 1 No

1 yes / 1 No

$$\text{Total Error} = \frac{1}{4} + \frac{1}{4} = \frac{2}{4}$$

(TE)

$$\text{final weight} = \frac{1}{2}\log\left[\frac{1-TE}{TE}\right]$$

$= \Rightarrow$

XGBoost

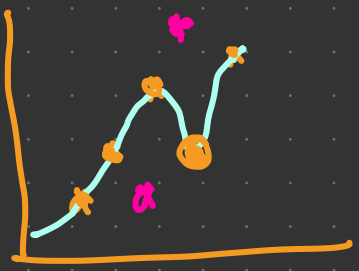
- XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework
- XGBoost algorithm was developed as a research project at the University of Washington
- Since its introduction, this algorithm has not only been credited with winning numerous Kaggle competitions but also for being the driving force under the hood for several cutting-edge industry applications
- As a result, there is a strong community of data scientists contributing to the XGBoost open source projects with ~350 contributors and ~3,600 commits on GitHub

Overfitting : training accuary is very good → 80'/.
testing accuray is very bad → 30'/.

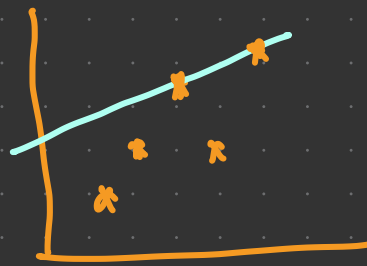
model tries to learn from all or most of the training data

y_pred = model.predict(x_test) ← test

y_pred = model.predict(x_train) ← train



overfitting



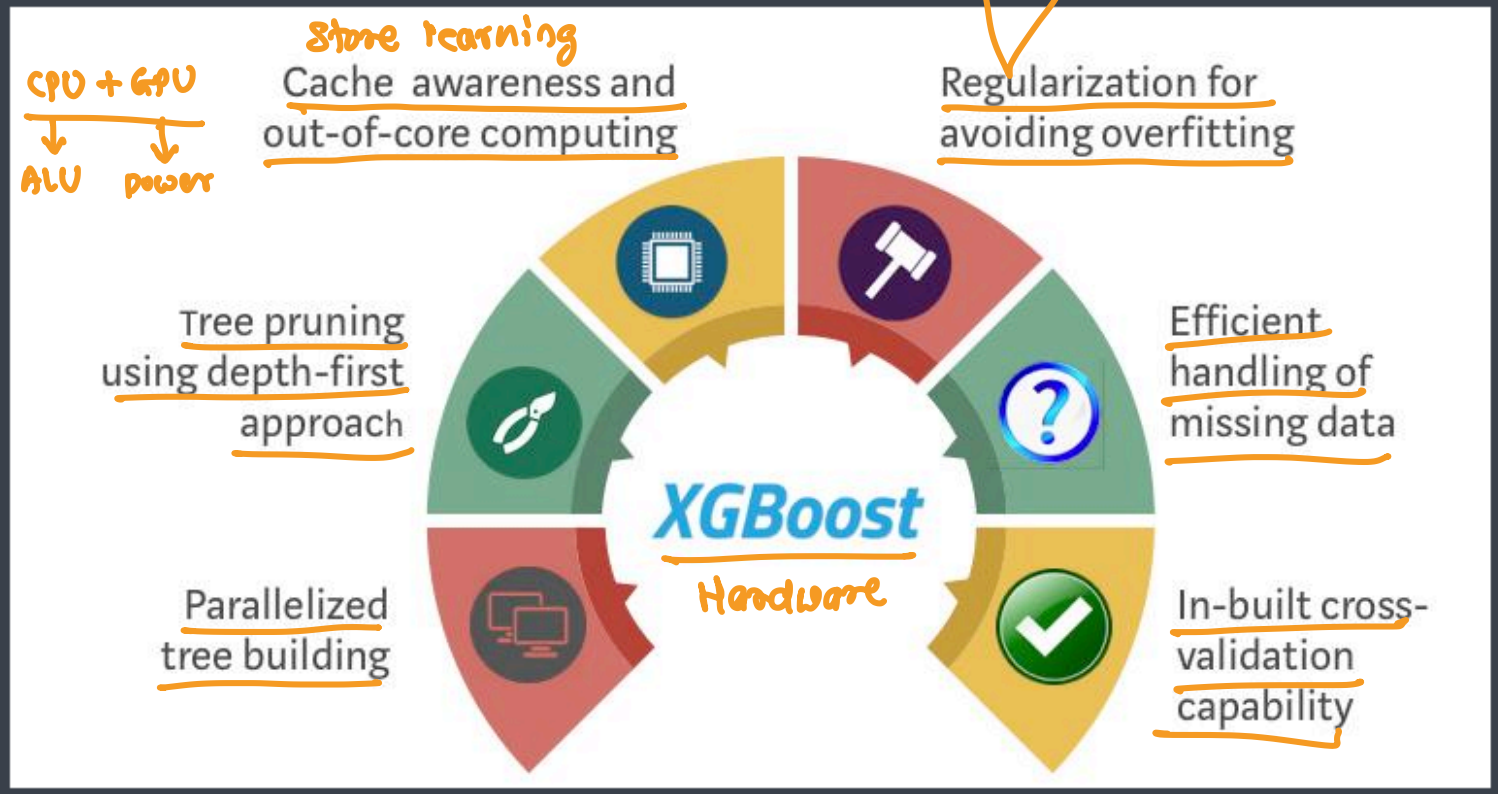
underfitting

training accuracy → Very good Bad

testing accuracy → very Bad Bad

Why does it perform so well?

- XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners using the gradient descent architecture
- However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.



System Optimization

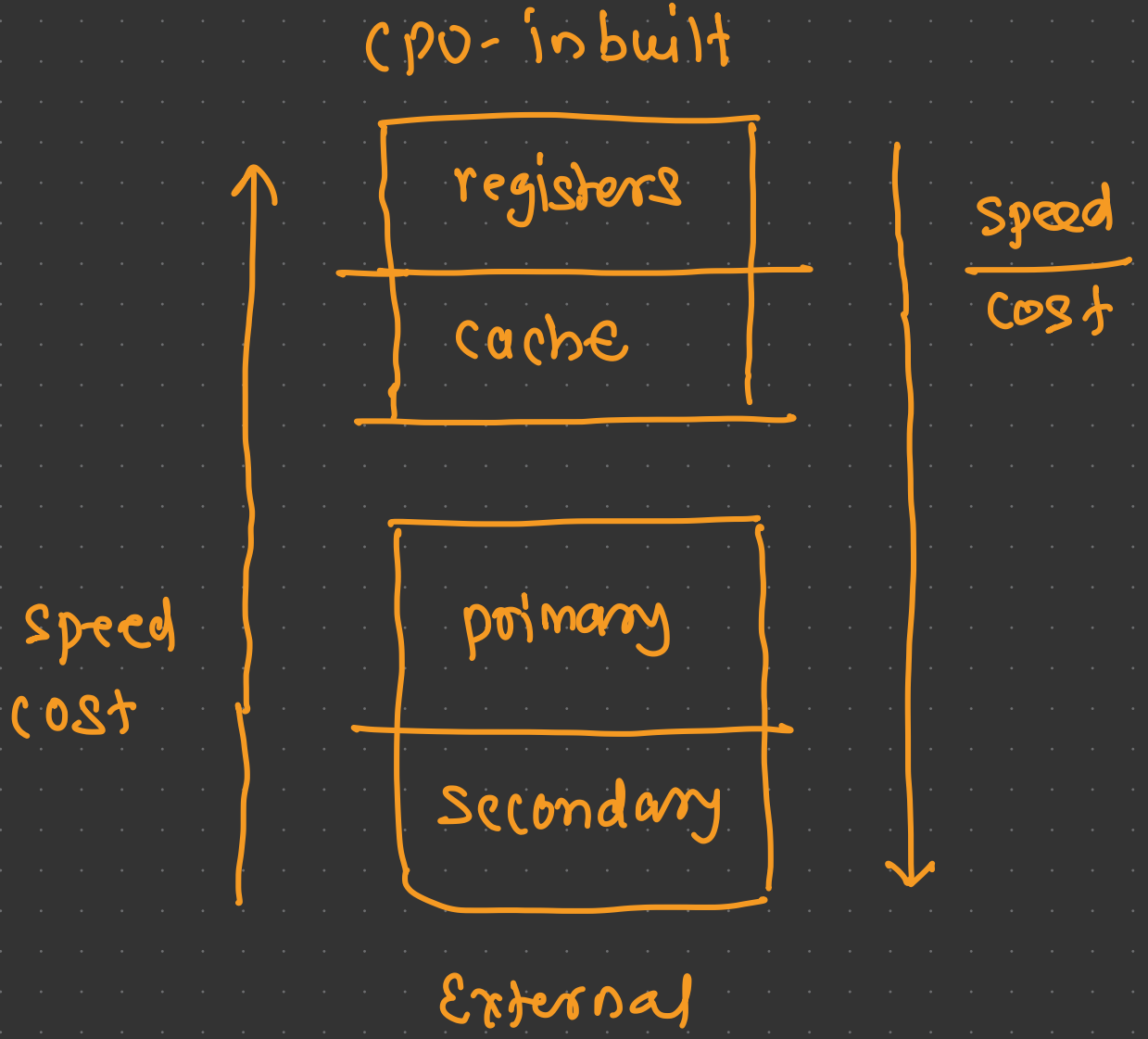
- **Parallelization**
  - XGBoost approaches the process of sequential tree building using parallelized implementation
  - This is possible due to the interchangeable nature of loops used for building base learners; the outer loop that enumerates the leaf nodes of a tree, and the second inner loop that calculates the features
  - This nesting of loops limits parallelization because without completing the inner loop (more computationally demanding of the two), the outer loop cannot be started
  - Therefore, to improve run time, the order of loops is interchanged using initialization through a global scan of all instances and sorting using parallel threads
  - This switch improves algorithmic performance by offsetting any parallelization overheads in computation
- **Tree Pruning**
  - The stopping criterion for tree splitting within GBM framework is greedy in nature and depends on the negative loss criterion at the point of split
  - XGBoost uses ‘max_depth’ parameter as specified instead of criterion first, and starts pruning trees backward
  - This ‘depth-first’ approach improves computational performance significantly.

System Optimization

- **Hardware Optimization**
  - This algorithm has been designed to make efficient use of hardware resources
  - This is accomplished by cache awareness by allocating internal buffers in each thread to store gradient statistics
  - Further enhancements such as ‘out-of-core’ computing optimize available disk space while handling big data-frames that do not fit into memory.

Memory

- primary
- secondary



Benefits

- **Parallel Computing**: It is enabled with parallel processing (using OpenMP); i.e., when you run xgboost, by default, it would use all the cores of your laptop/machine.
- **Regularization**: I believe this is the biggest advantage of xgboost. GBM has no provision for regularization. Regularization is a technique used to avoid overfitting in linear and tree-based models.
- **Enabled Cross Validation**: In R, we usually use external packages such as caret and mlr to obtain CV results. But, xgboost is enabled with internal CV function (we'll see below).
- **Missing Values**: XGBoost is designed to handle missing values internally. The missing values are treated in such a manner that if there exists any trend in missing values, it is captured by the model.
- **Flexibility**: In addition to regression, classification, and ranking problems, it supports user-defined objective functions also. An objective function is used to measure the performance of the model given a certain set of parameters. Furthermore, it supports user defined evaluation metrics as well.

Benefits

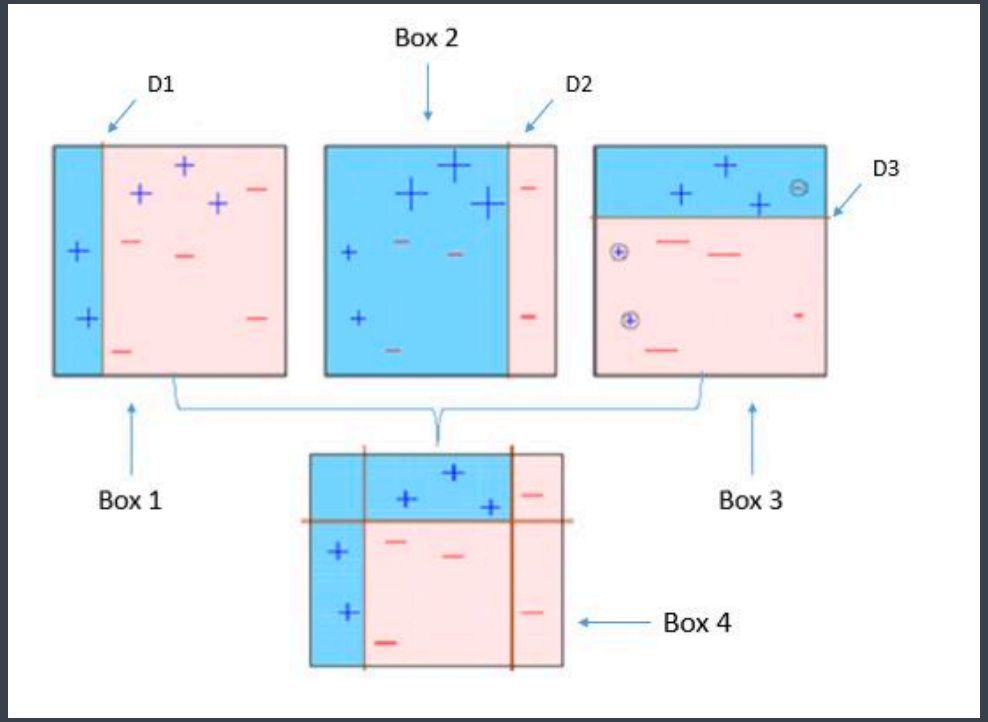
- **Availability**: Currently, it is available for programming languages such as R, Python, Java, Julia, and Scala.
- **Save and Reload**: XGBoost gives us a feature to save our data matrix and model and reload it later. Suppose, we have a large data set, we can simply save the model and use it in future instead of wasting time redoing the computation.
- **Tree Pruning**: Unlike GBM, where tree pruning stops once a negative loss is encountered, XGBoost grows the tree upto max_depth and then prune backward until the improvement in loss function is below a threshold.

How does it work?

- It combines a set of weak learners and delivers improved prediction accuracy
- At any instant t, the model outcomes are weighed based on the outcomes of previous instant t-1
- The outcomes predicted correctly are given a lower weight and the ones miss-classified are weighted higher
- Note that a weak learner is one which is slightly better than random guessing

How does it work?

- **1. Box 1**: The first classifier (usually a decision stump) creates a vertical line (split) at D1. It says anything to the left of D1 is + and anything to the right of D1 is -. However, this classifier misclassifies three + points.
- **Note** a Decision Stump is a Decision Tree model that only splits off at one level, therefore the final prediction is based on only one feature.
- **2. Box 2**: The second classifier gives more weight to the three + misclassified points (see the bigger size of +) and creates a vertical line at D2. Again it says, anything to the right of D2 is - and left is +. Still, it makes mistakes by incorrectly classifying three - points.
- **3. Box 3**: Again, the third classifier gives more weight to the three - misclassified points and creates a horizontal line at D3. Still, this classifier fails to classify the points (in the circles) correctly.
- **4. Box 4**: This is a weighted combination of the weak classifiers (Box 1,2 and 3). As you can see, it does a good job at classifying all the points correctly.



Stacking

Stacking

- Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor
- The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model as features
- The base level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous