Ensembles:

↳ group of 'musicians'

ML: multiple models used together

$\{M_1, M_2, M_3, \ldots M_k\}$ → base models

↓ (combine)

M → more "powerful" model

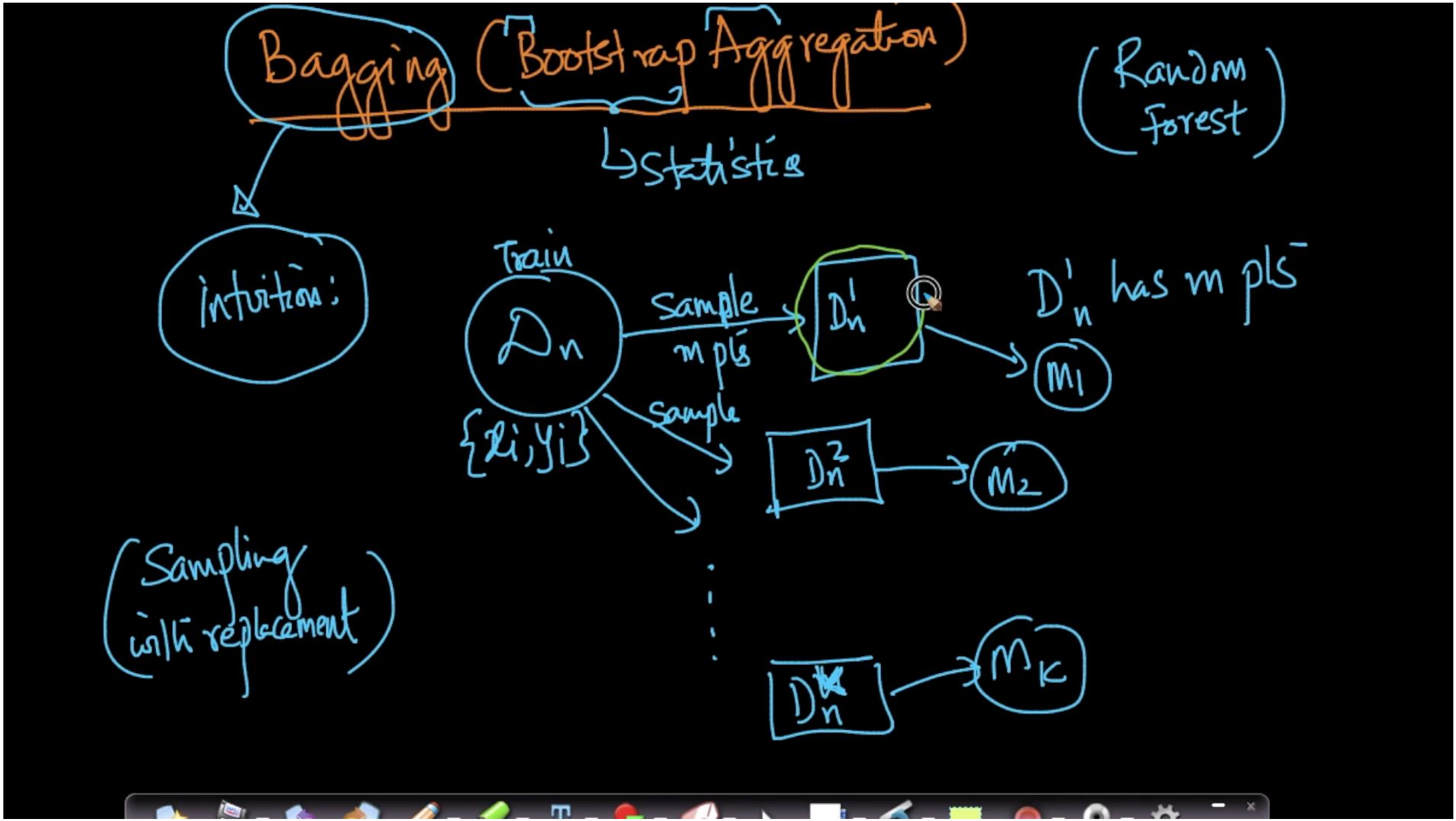
4 types:
- Bagging (Bootstrapped Aggregation)
- Boosting
- Stacking
- Cascading

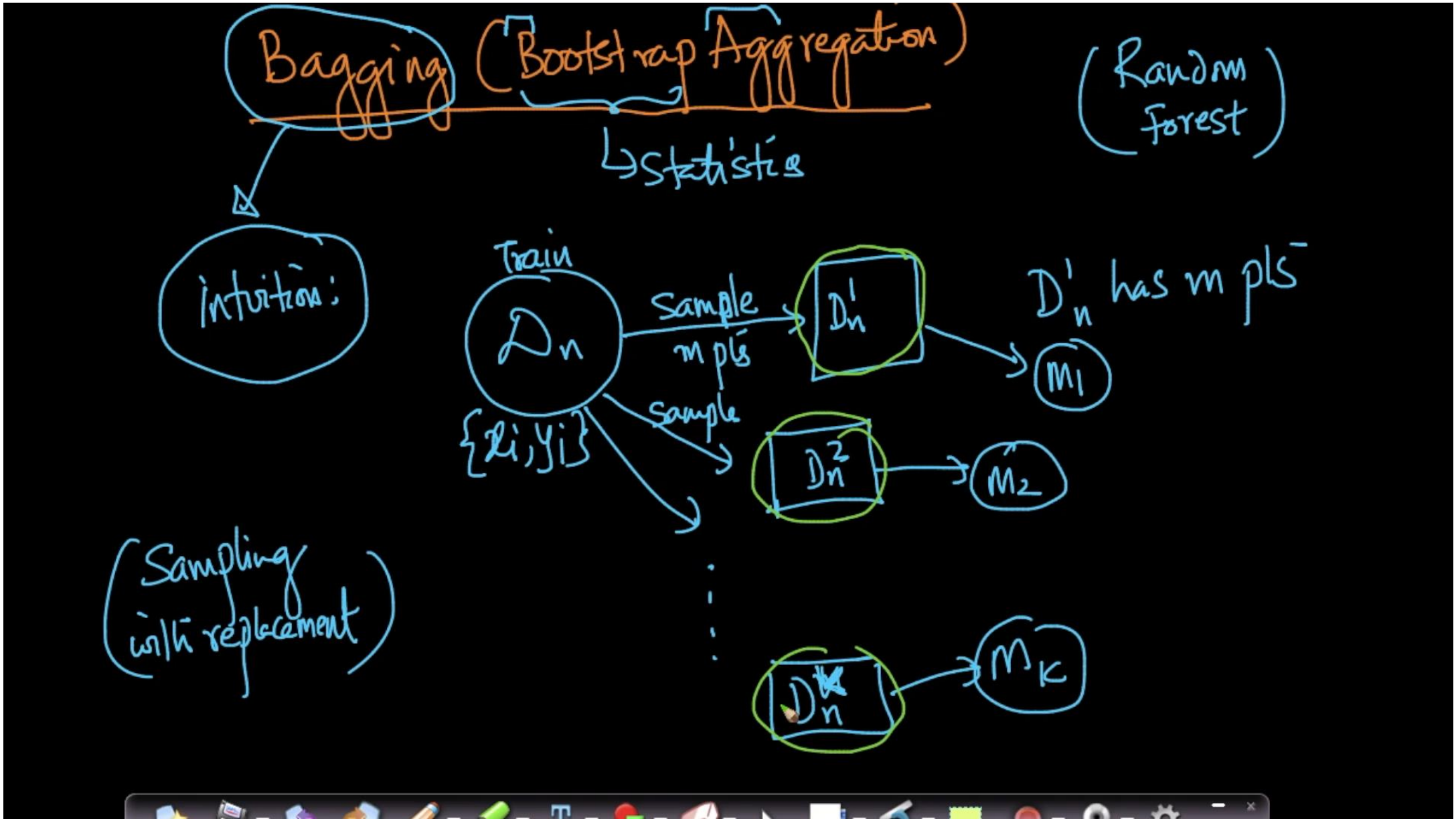
→ high-performing
→ v. powerful
→ Kaggle → most
→ very useful in real-world.

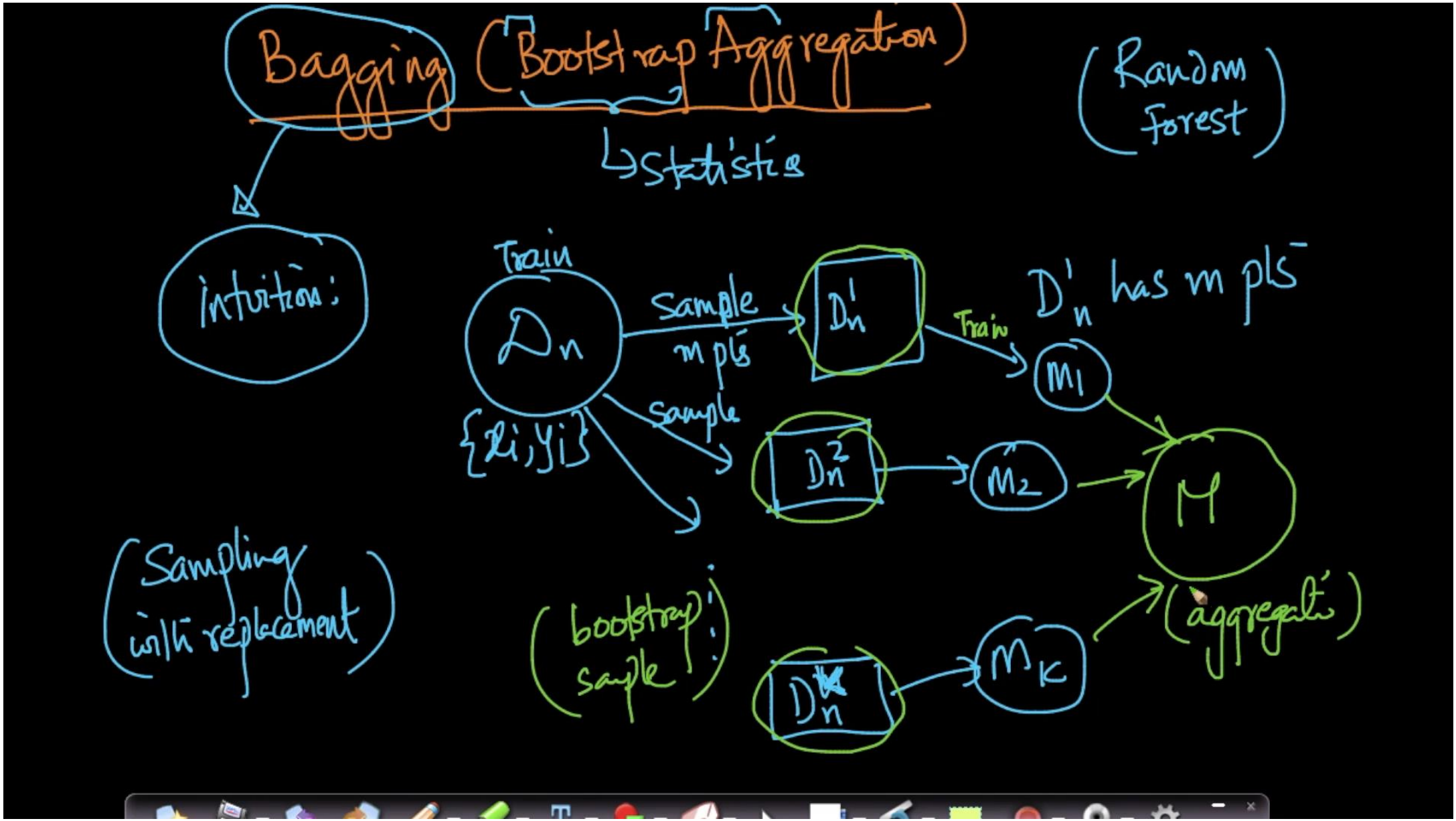
Key aspect: $M_1, M_2, M_3, \ldots M_k$

more different these models are, the better you can combine them: } generic

{ Problem :- (M_i :- expert)


Bagging (Bootstrap Aggregation)
↳ statistics
(Random Forest)
intuition:
Train
D_n
$\{x_i, y_i\}$
sample m pts
D_n^1
D_n' has m pts
m_1
sample
D_n^2
m_2
(Sampling with replacement)
D_n^k
m_k


Bagging (Bootstrap Aggregation)
↳ statistics
(Random Forest)
intuition:
Train
D_n
$\{x_i, y_i\}$
sample m pts
D_n^1
D_n' has m pts
m_1
sample
D_n^2
m_2
(Sampling with replacement)
D_n^k
m_k

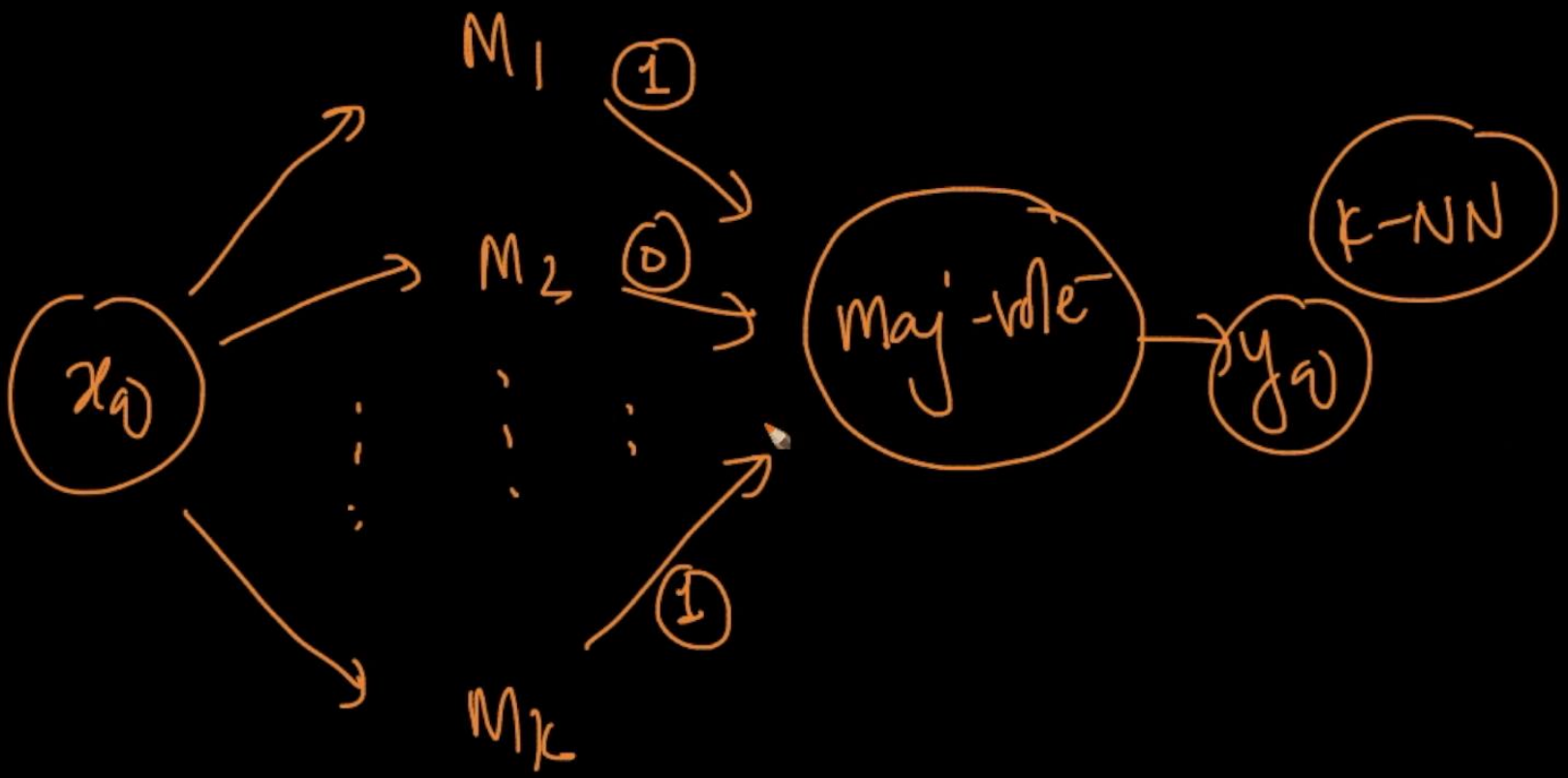

Bagging (Bootstrap Aggregation)
(Random Forest)
↳ statistics
intuition:
Train
D_n
sample m pts
D_n'
Train
D_n' has m pts
m_1
sample
D_n^2
m_2
M
{xi, yi}
(Sampling with replacement)
(bootstrap sample)
D_n^k
m_k
(aggregate)

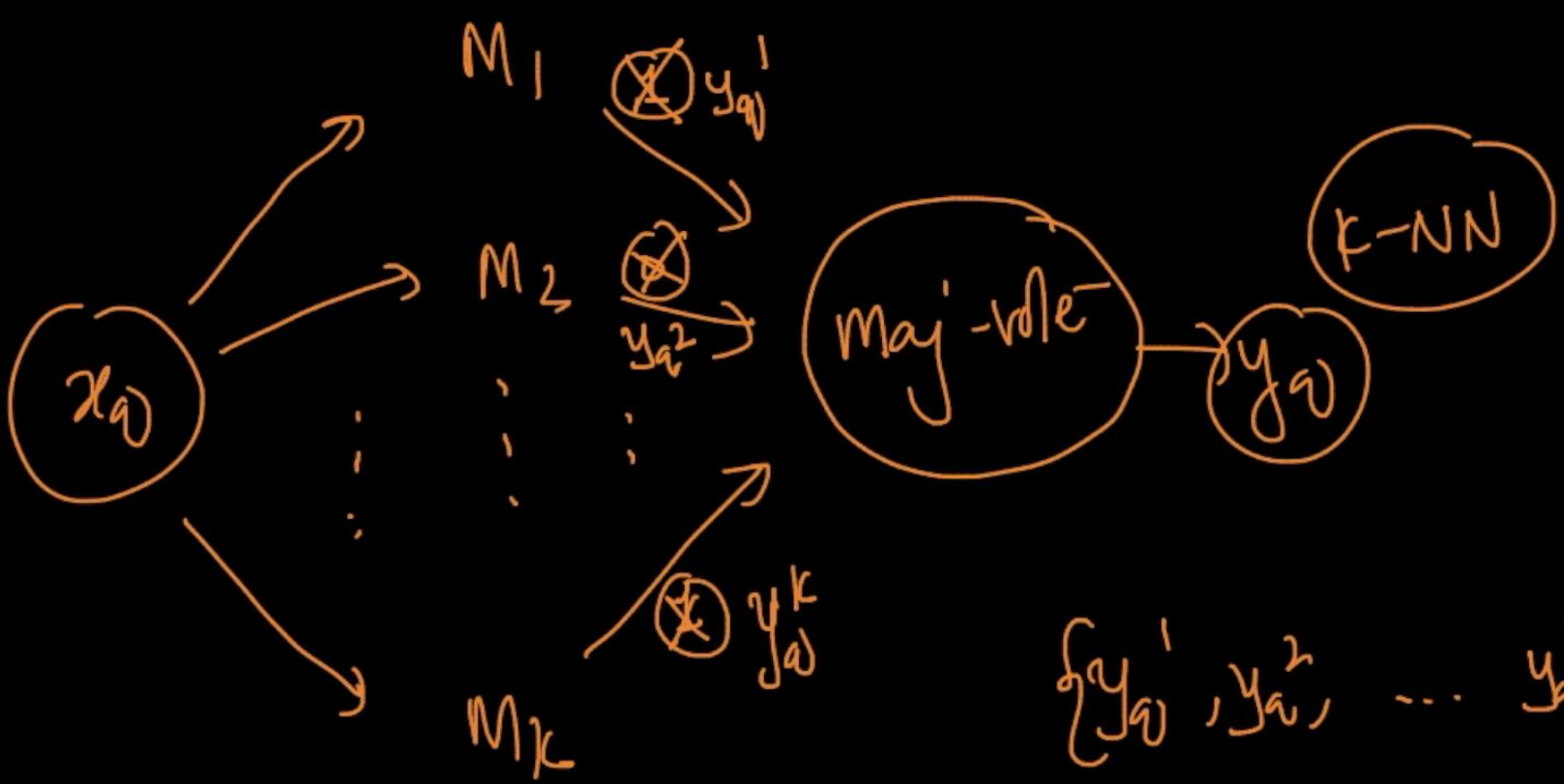
M_i is built using D_n^i of size m ($m \leq n$)

$\Rightarrow$ each model M_i has seen a different subset of data

Aggregation :-

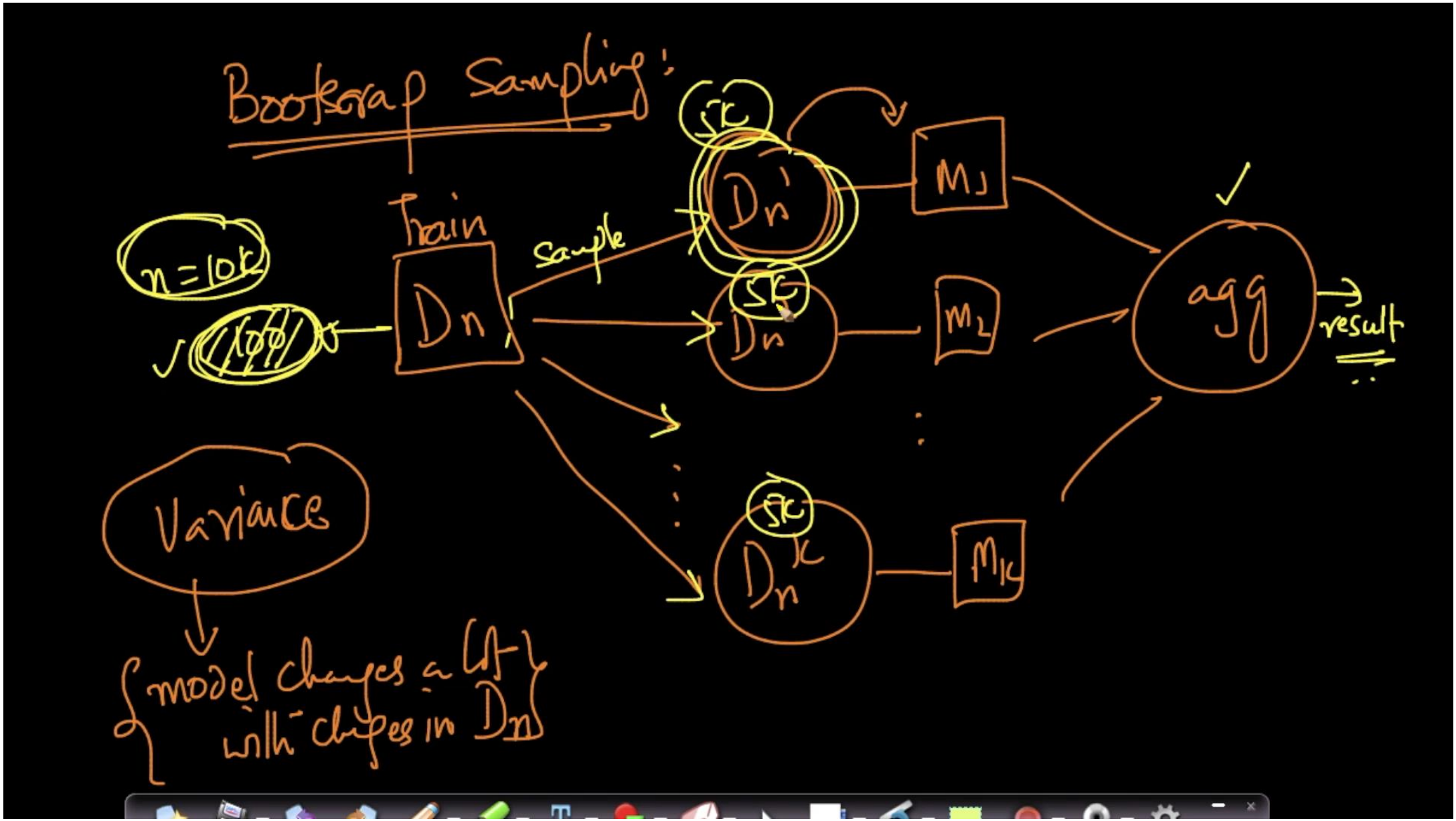
- Classification :- Majority vote ✓
- Regression :- mean | median

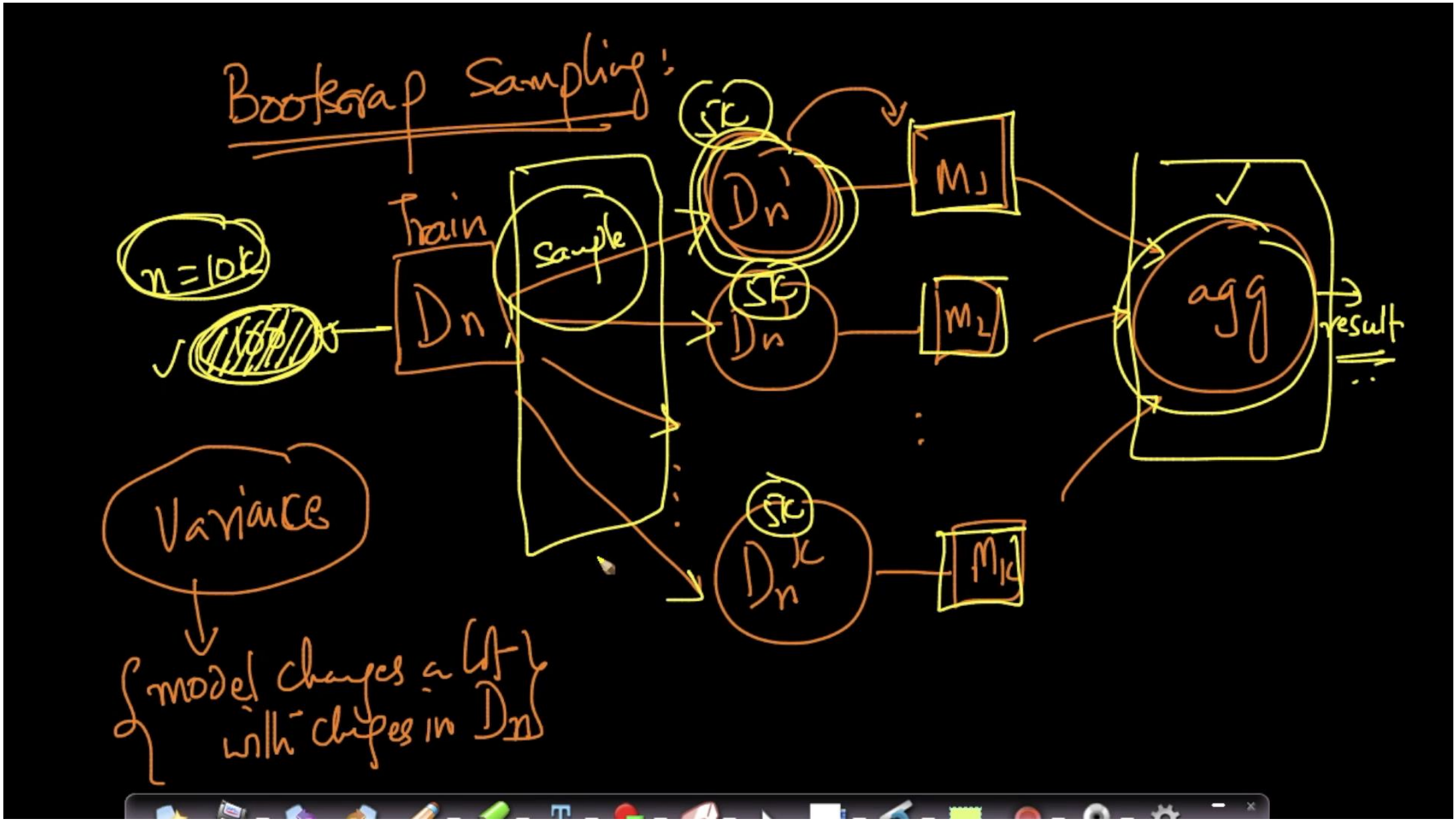

M_1
1
M_2
0
M_k
1
x_q
maj-vote
y_q
K-NN



$$\{y_{q0}^1, y_{q0}^2, \ldots y_{q0}^k\}$$

$$y_{q0} = \text{mean}\{y_{q0}^i\}$$


Bootstrap Sampling:
Train
sample
n = 10k
Dn
Variance
{model changes a lot with changes in Dn}
M1
M2
Mk
agg
result

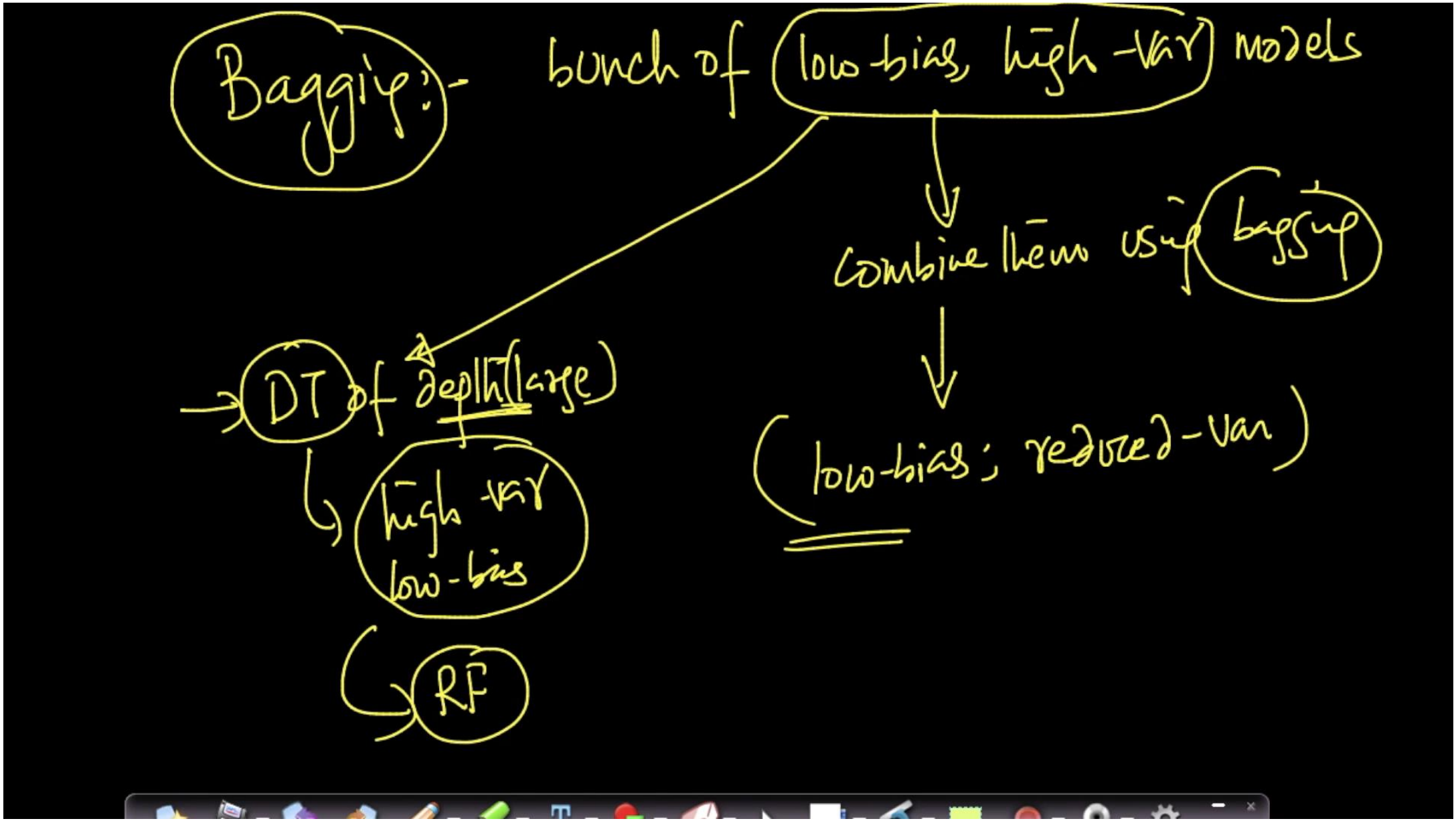

Bootstrap Sampling:
Train
Sample
n = 10k
Variance
{model changes a lot with changes in Dn}
agg
result

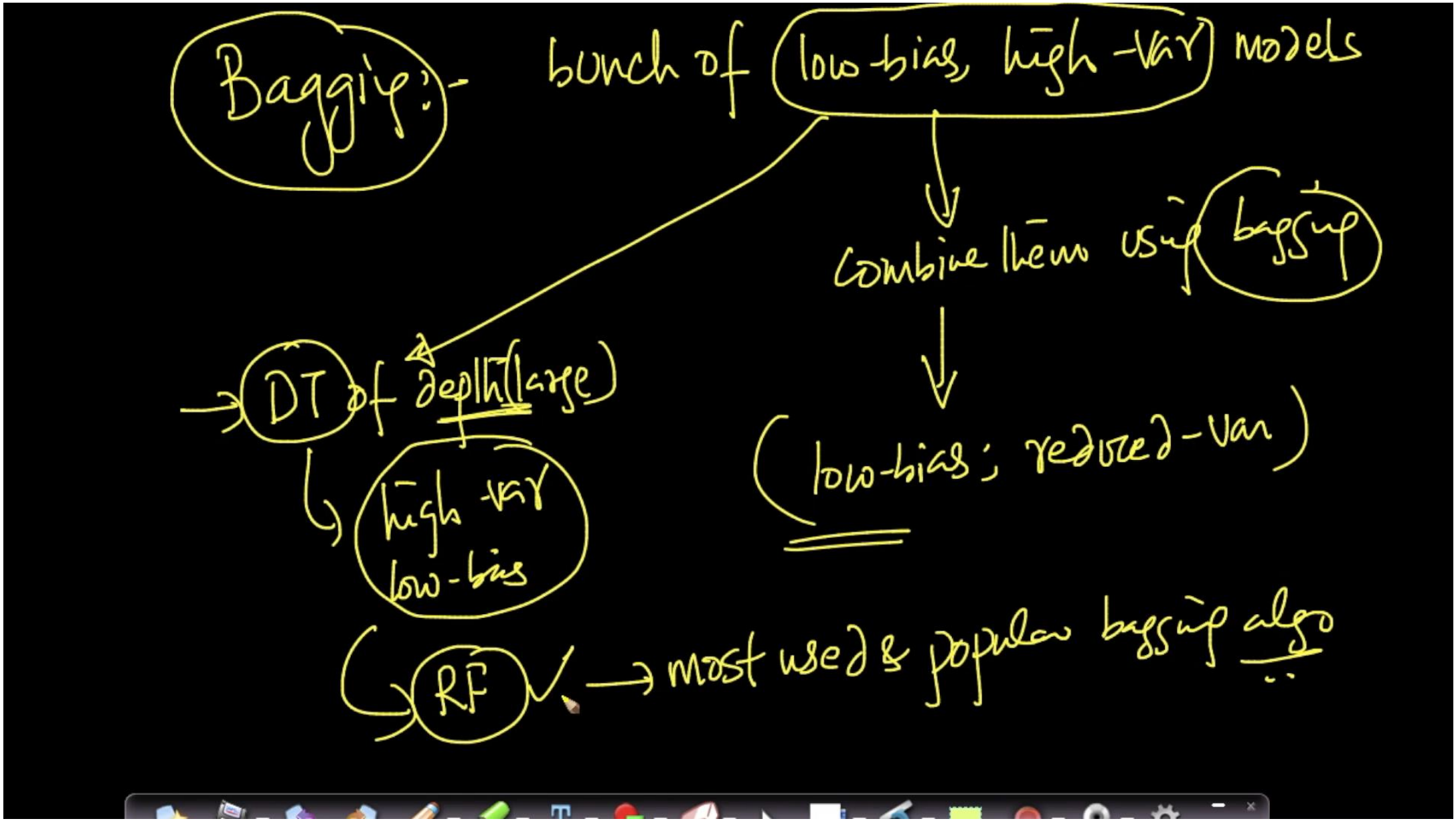
Bagging → can reduce variance in a model without impacting the bias

model error = $Bias^2 + Var$

base-model (M_i) :- low bias; high-Var model

Bagging (M_i's) → low-bias ; reduced variance


Bagging:-
bunch of
low-bias, high-var
models
Combine them using
bagging
DT of depth(large)
high-var
low-bias
RF
low-bias; reduced-var


Baggig:-
bunch of
low-bias, high-var
models
combine them using
bagging
DT of depth(large)
high-var
low-bias
(low-bias; reduced-var)
RF
→ most used & popular bagging algo

Random Forest (Bagging)

Decision Trees :-

"Random" Bootstrap Sampling

RF : DT + Bagging + col. sampling → feature bagging

DT ← base

Random Forest (Bagging)

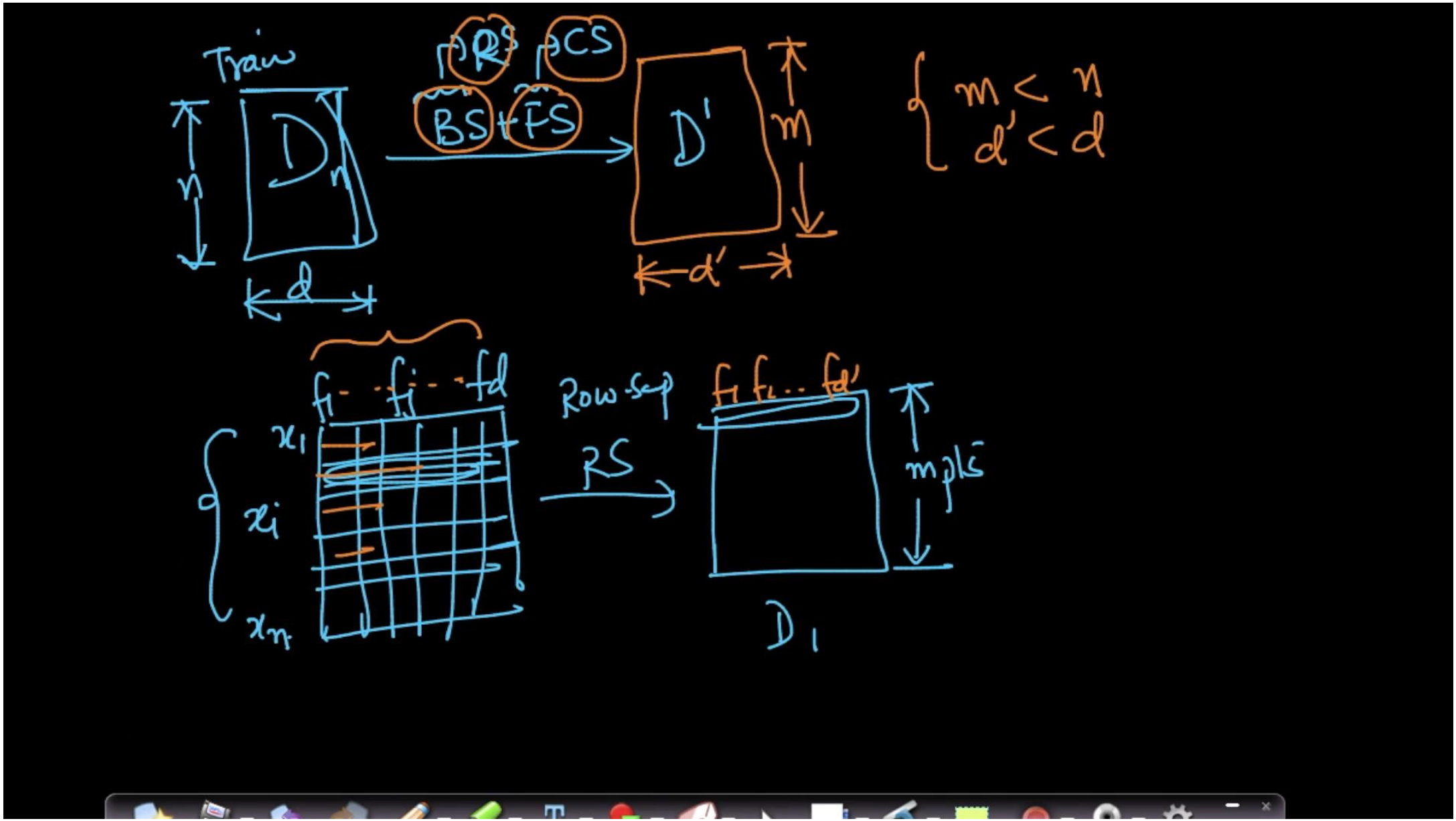
Decision Trees :-

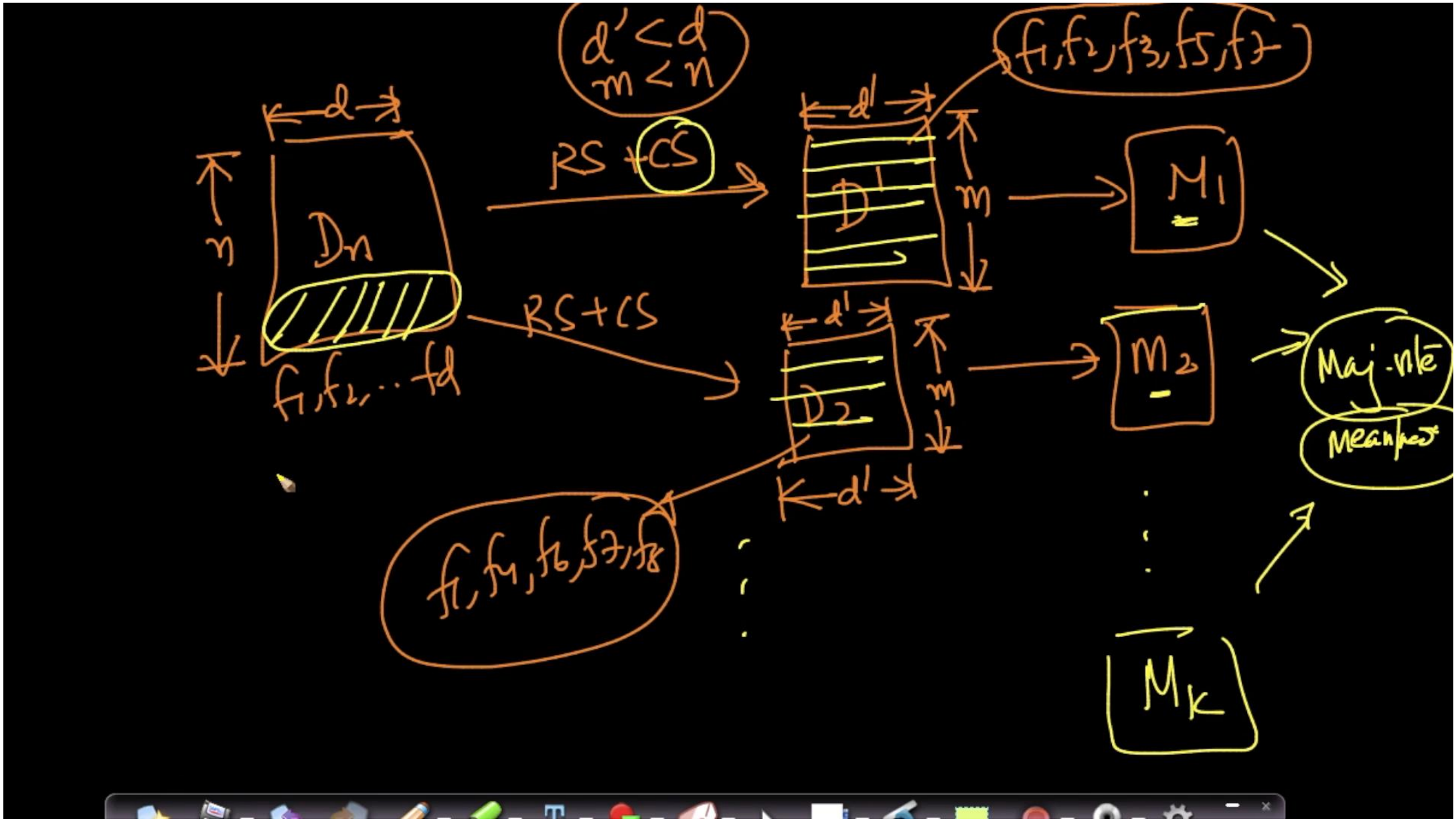
"Random" Bootstrap Sampling

RF : DT + Bagging + col. sampling → feature bagging

DT ↑ base

Bagging → Row. sampling w/ replacement

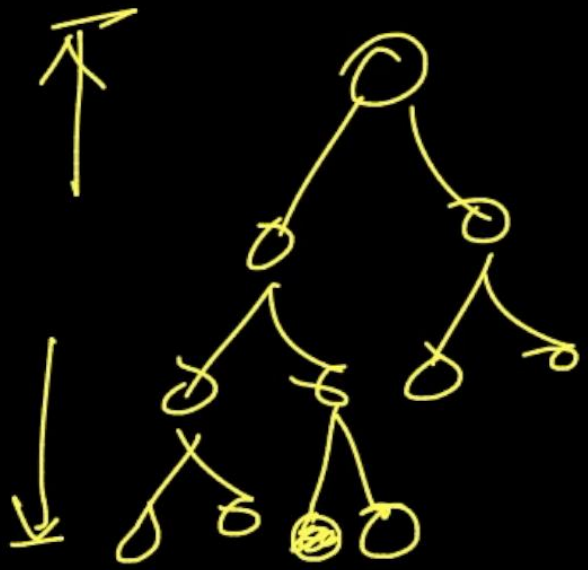

Train
D
n
d
RS
CS
BS
FS
D'
m
d'
$\begin{cases} m < n \\ d' < d \end{cases}$
f_1 ... f_i ... f_d
x_1
x_i
x_n
Row-sep
RS
f_1 f_2 ... $f_{d'}$
mpls
D_1

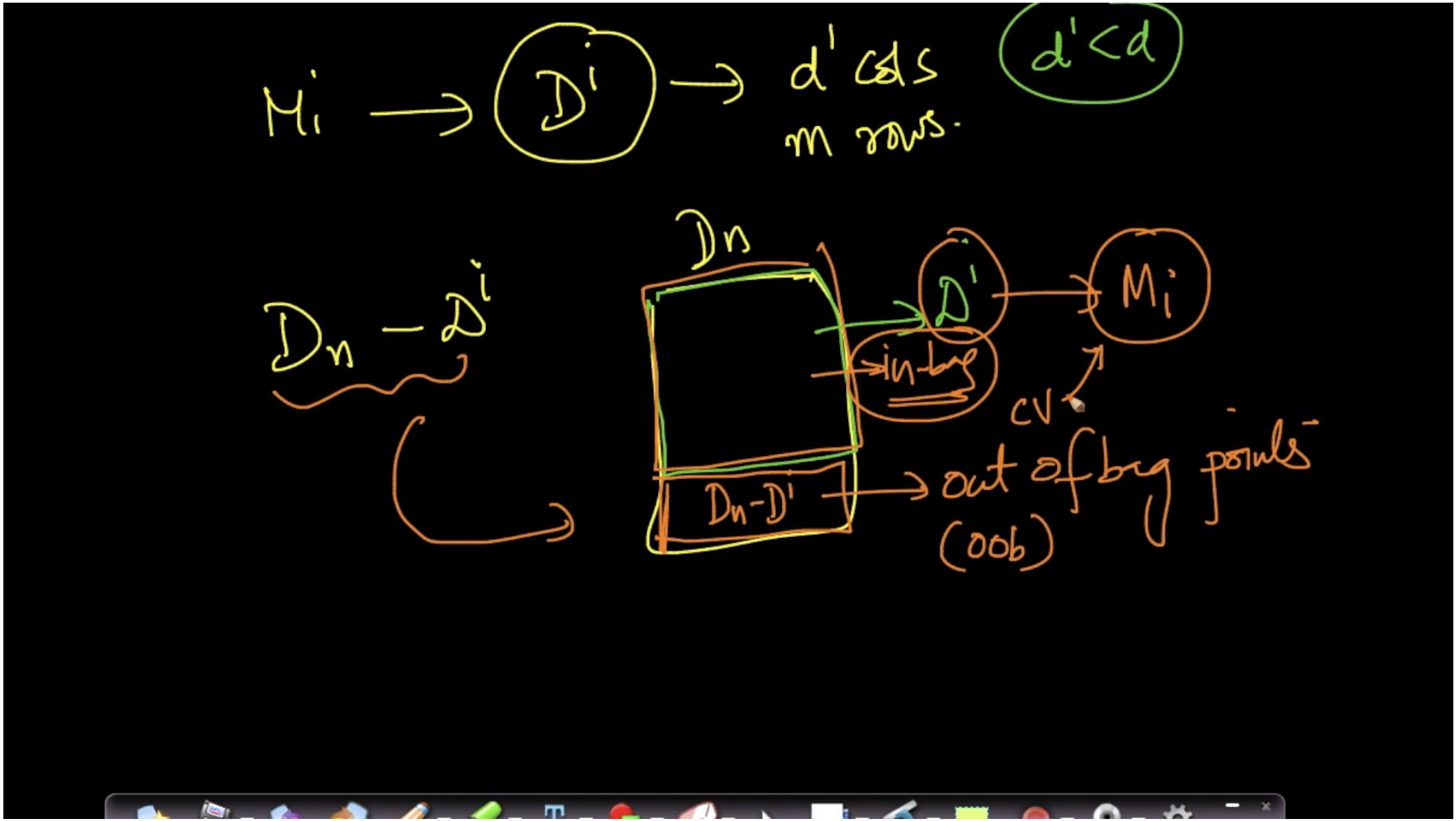

d' < d
m < n
d
n
Dn
f1, f2, ... fd
RS + CS
d'
m
D1
f1, f2, f3, f5, f7
M1
RS + CS
D2
M2
f1, f4, f6, f7, f8
Mk
Maj-vote
Mean/med

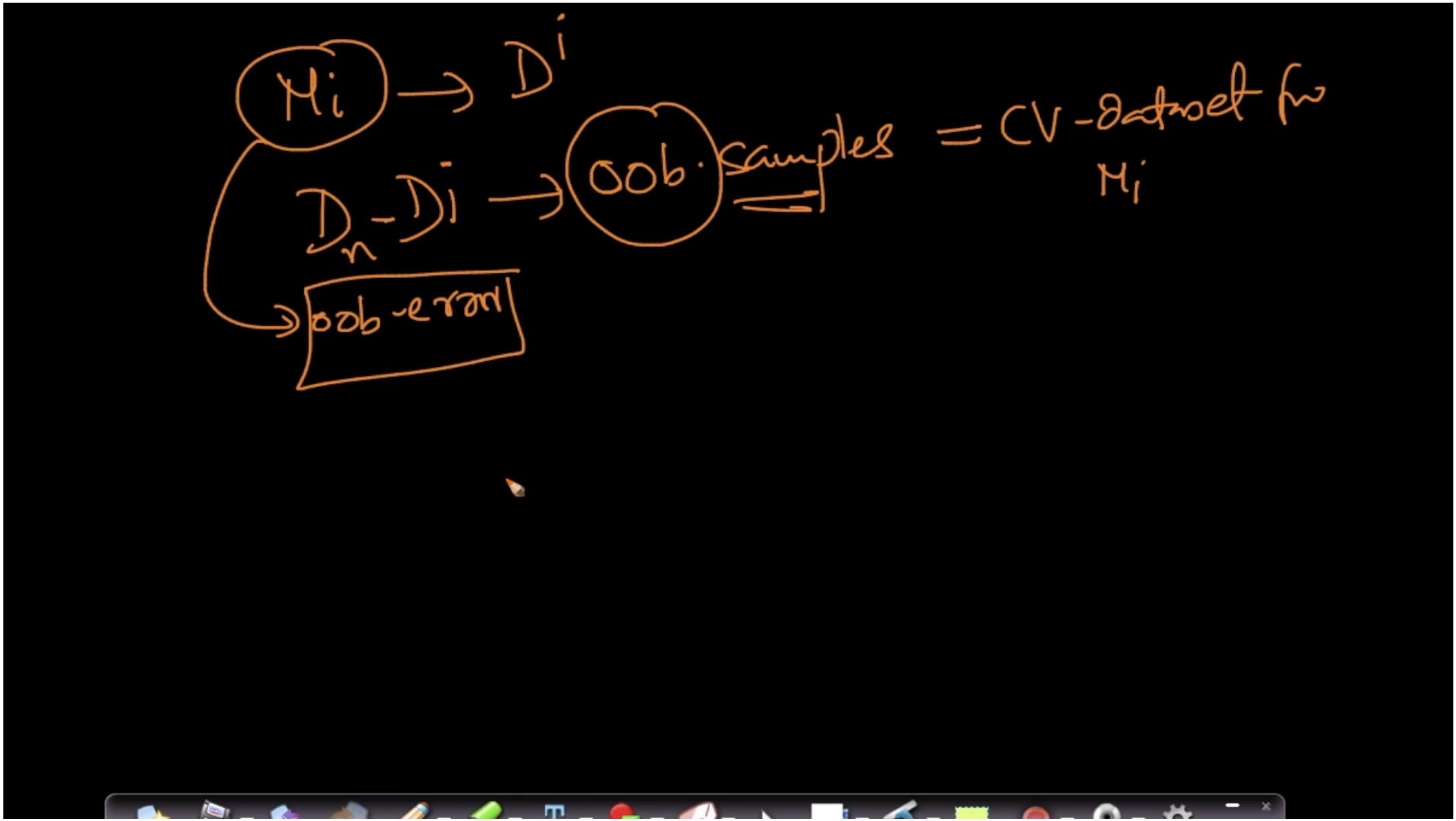
$H_i(S_i)$ — DT of reasonable depth

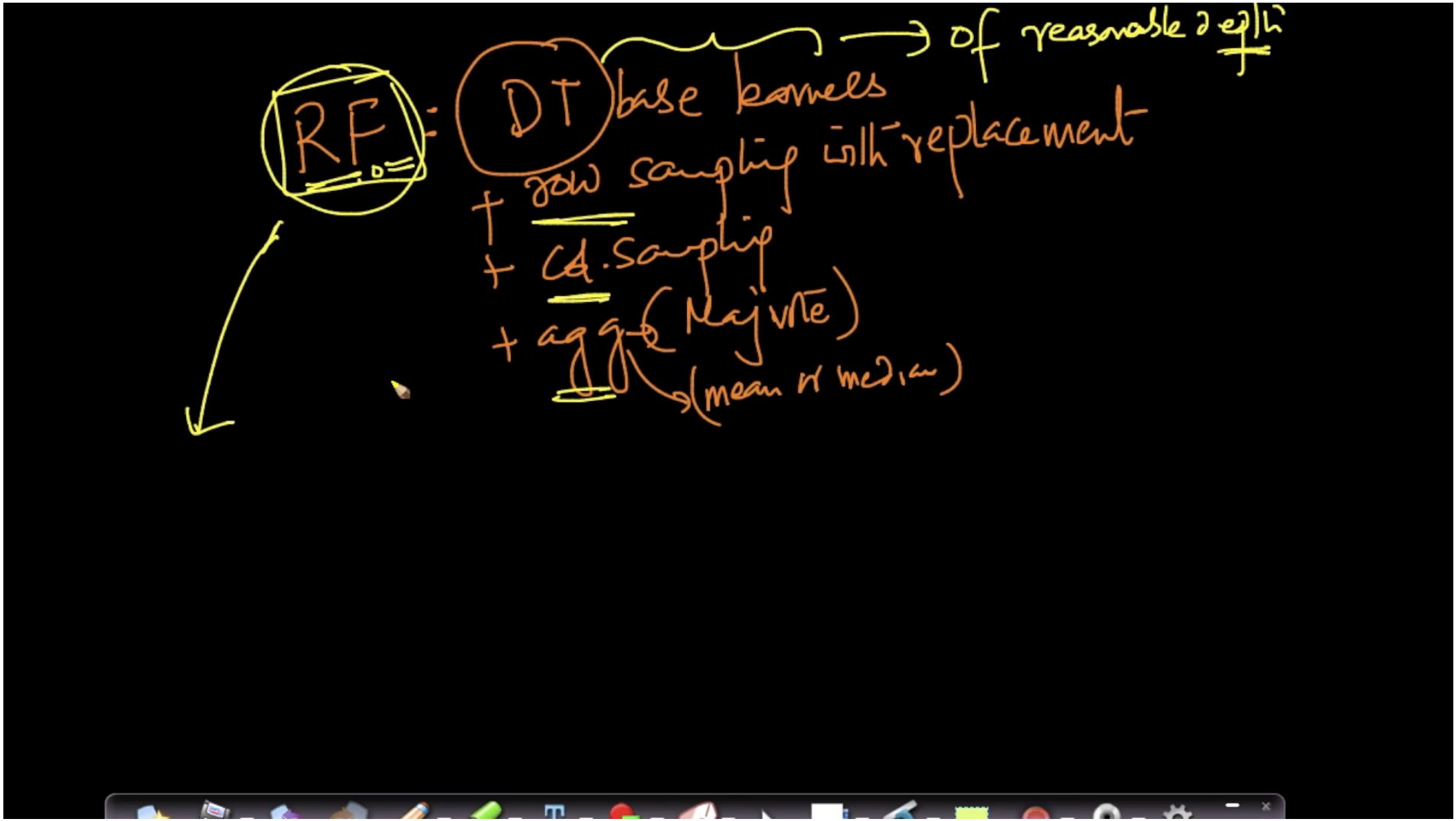
base learners: - low-bias, high-var

(reduces-variance)




Mi
Di
d' cols
m rows.
d'<d
Dn
Dn - Di
D'
Mi
in-bag
CV
Dn-D'
out of bag points
(oob)


Mi
Di
Dn - Di
oob samples
= CV-dataset for Mi
oob-error


RF
DT
base learners
of reasonable depth
+ row sampling with replacement
+ col. sampling
+ aggr (Majority)
(mean or median)

Bias - Variance tradeoff

RF: reduce variance

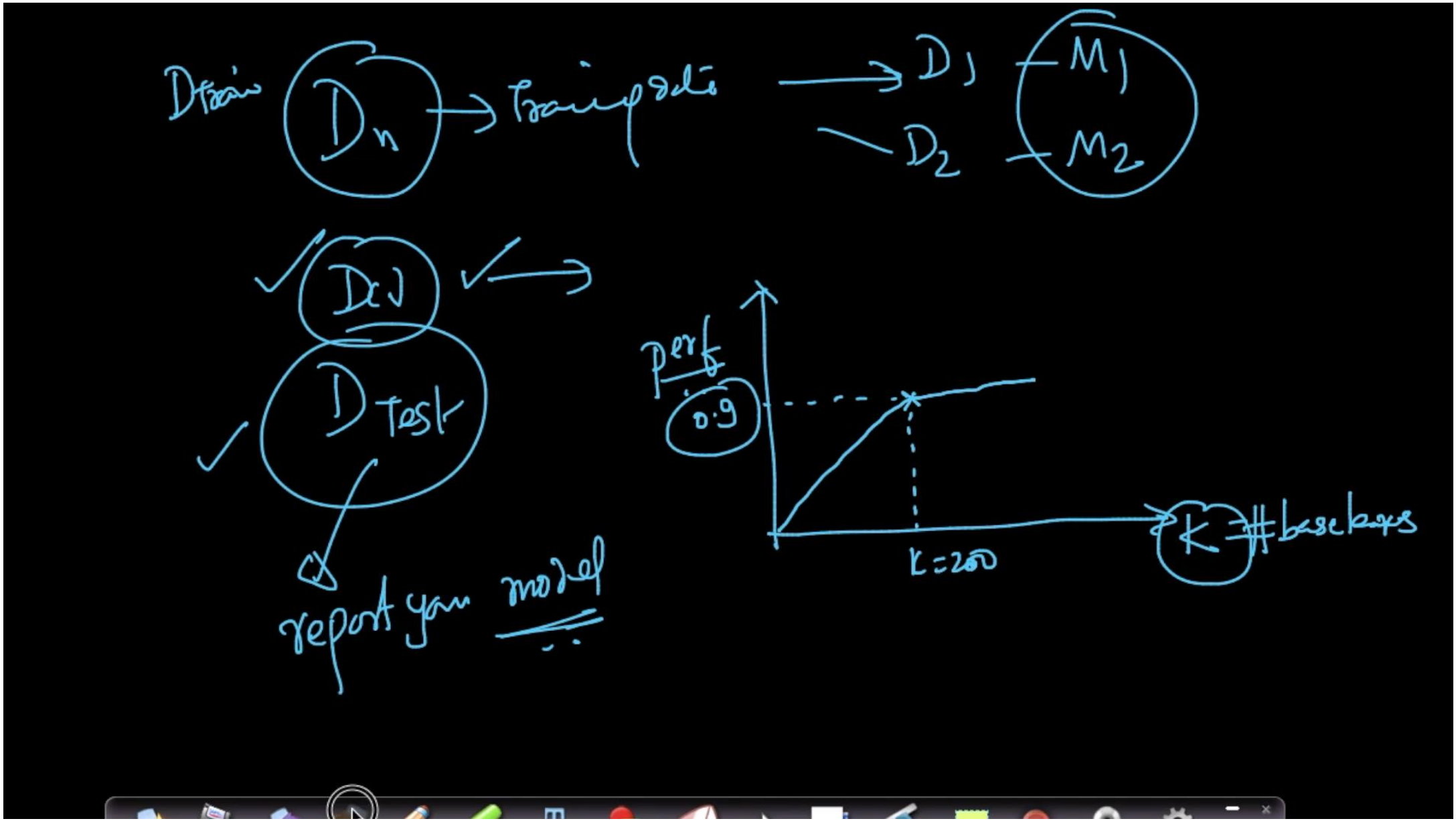
↳ low bias because base learners (M_i's) are low-bias

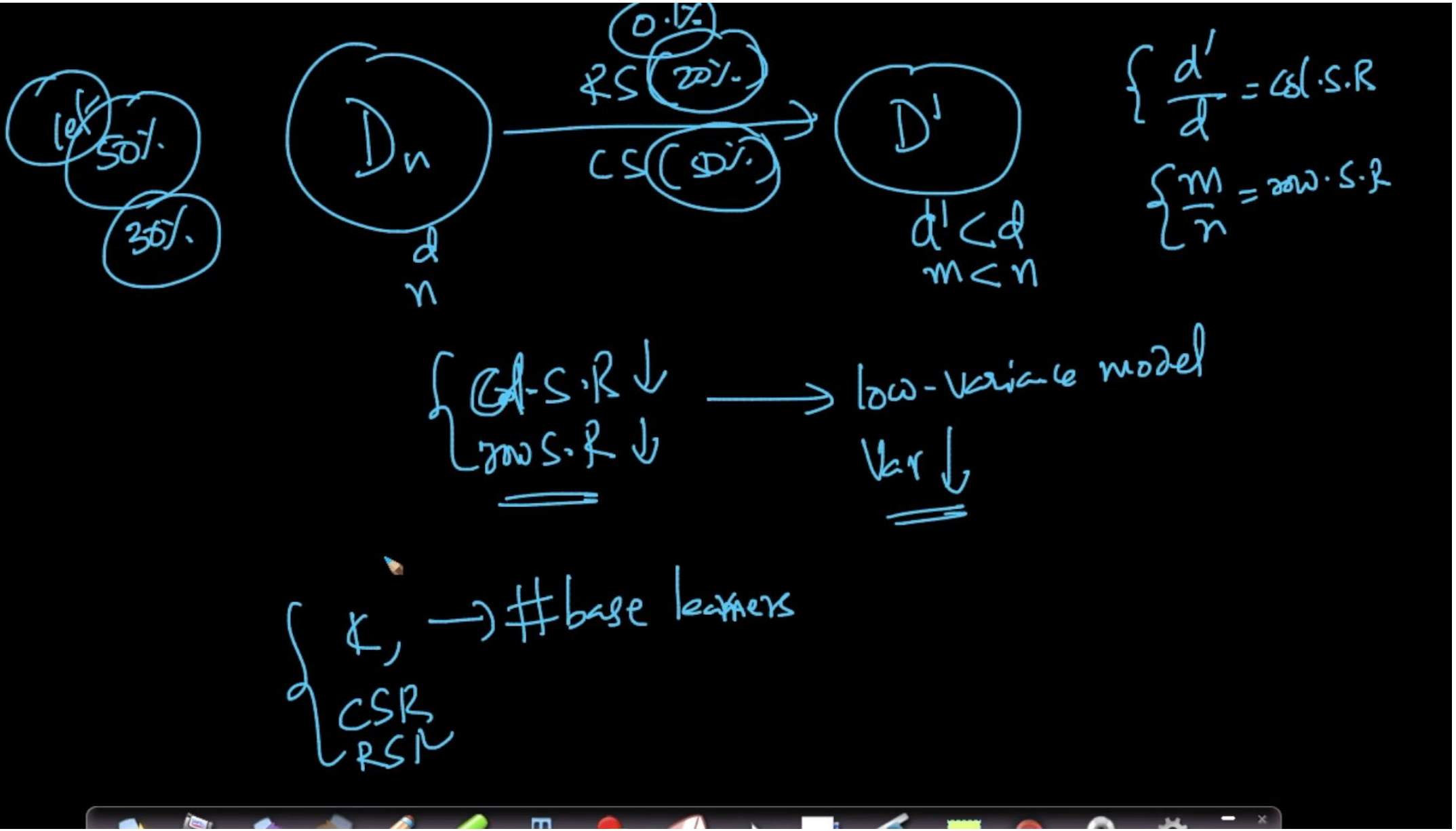
↳ Var

$M = agg(M_1, M_2, M_3, \ldots M_K)$ — 2000

$bias(M) = bias(M_i)$

$\begin{cases} K \uparrow ; \text{ Variance} \downarrow \\ K \downarrow ; \text{ Variance} \uparrow \end{cases}$


Dtrain
Dn
Training data
D1
D2
M1
M2
D(1)
D test
report your model
perf
0.9
K=200
K = # base boxes


10%
50%
35%
D_n
d
n
RS
20%
CS
50%
D'
$d' < d$
$m < n$
$\frac{d'}{d} = \text{col} \cdot S \cdot R$
$\frac{m}{n} = \text{row} \cdot S \cdot R$
Col-S.R ↓
row S.R ↓
low-variance model
Var ↓
K, → #base learners
CSR
RSR

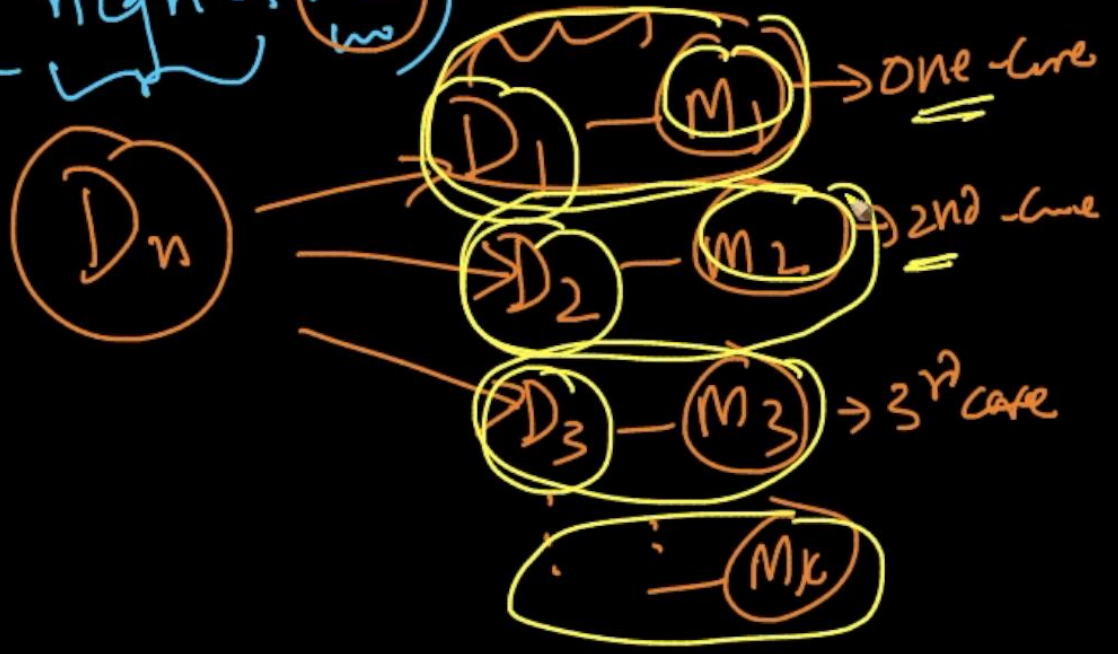
Train & Runtime Complexity

RF with K base-learners (DT)

multi-core

100-boxes

Train: $O(n \lg n\, d\, (*K))$

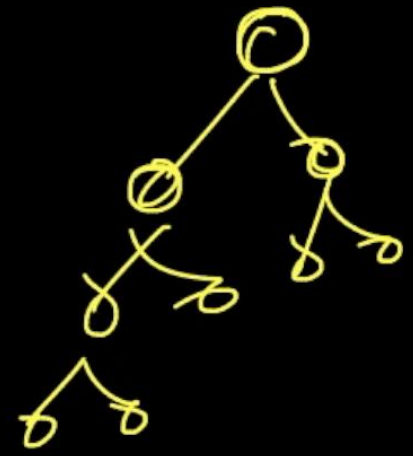


Trivially parallelizable

Runtime:-

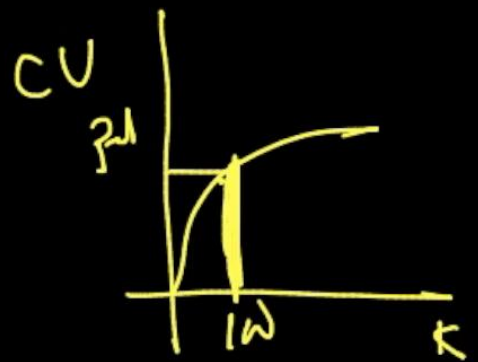
O(depth * k)

large (1 to 20)



Space:

O(DT * k)



RF with 200 trees of depth = 6 → ~3 ms

(if ... else)

$z_v \rightarrow y_v$

Extremely Randomized Trees:

→ Try out a random sample of possible values to determine τ

RF: → regular DT + Col. sampling; row sampling; agg

Mi →

{ Extreme Trees: →

fi < τ

Y N

DT/RF

↓

{ Try out all the possible values of fi to determine τ

(10 random values of fi)

RF/DT { fi: real valued values

Sort fi

n

Extreme Trees :-

Col sampling + r.s + agg → RF

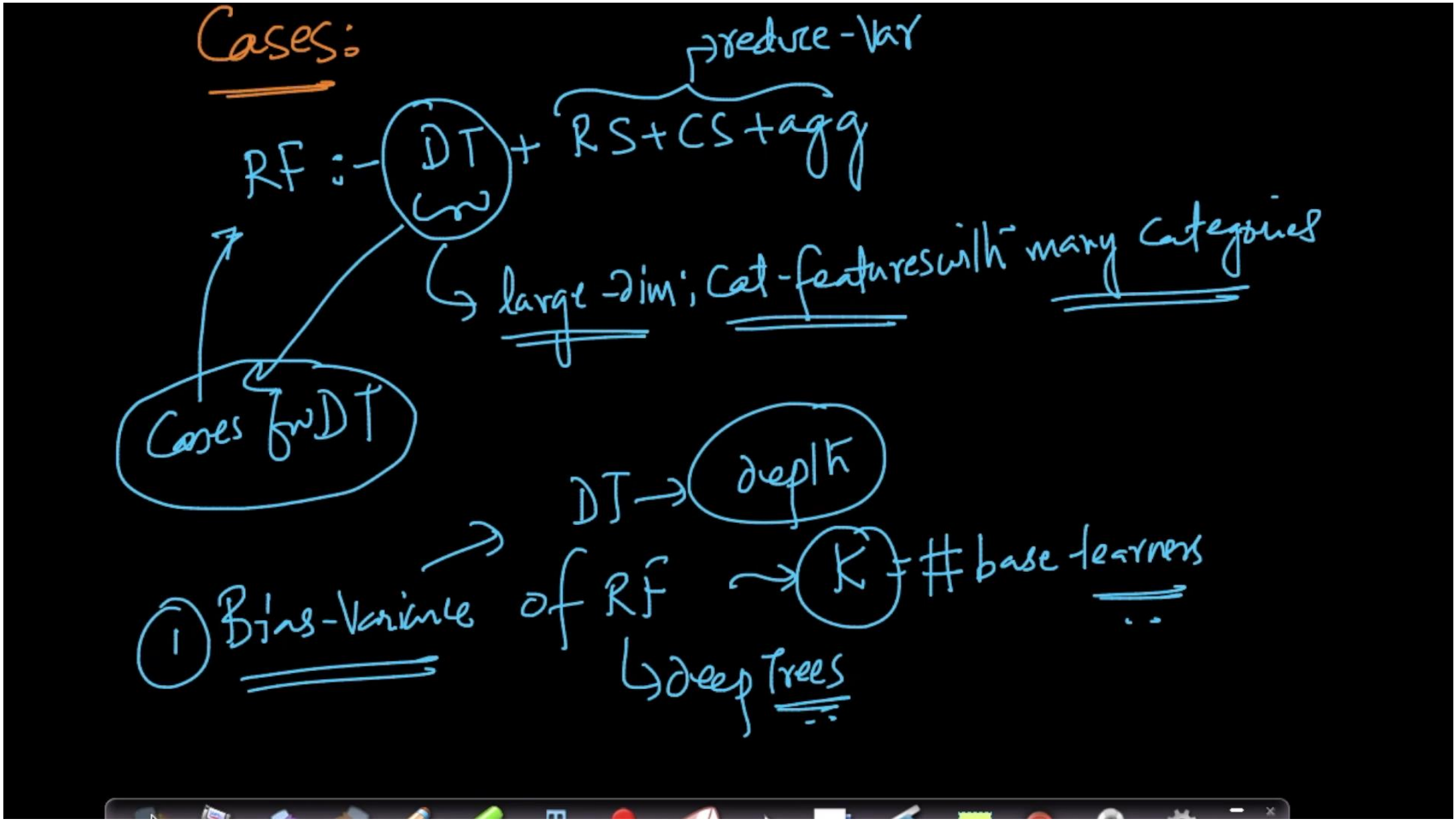
+ randomization when selecting 'c'

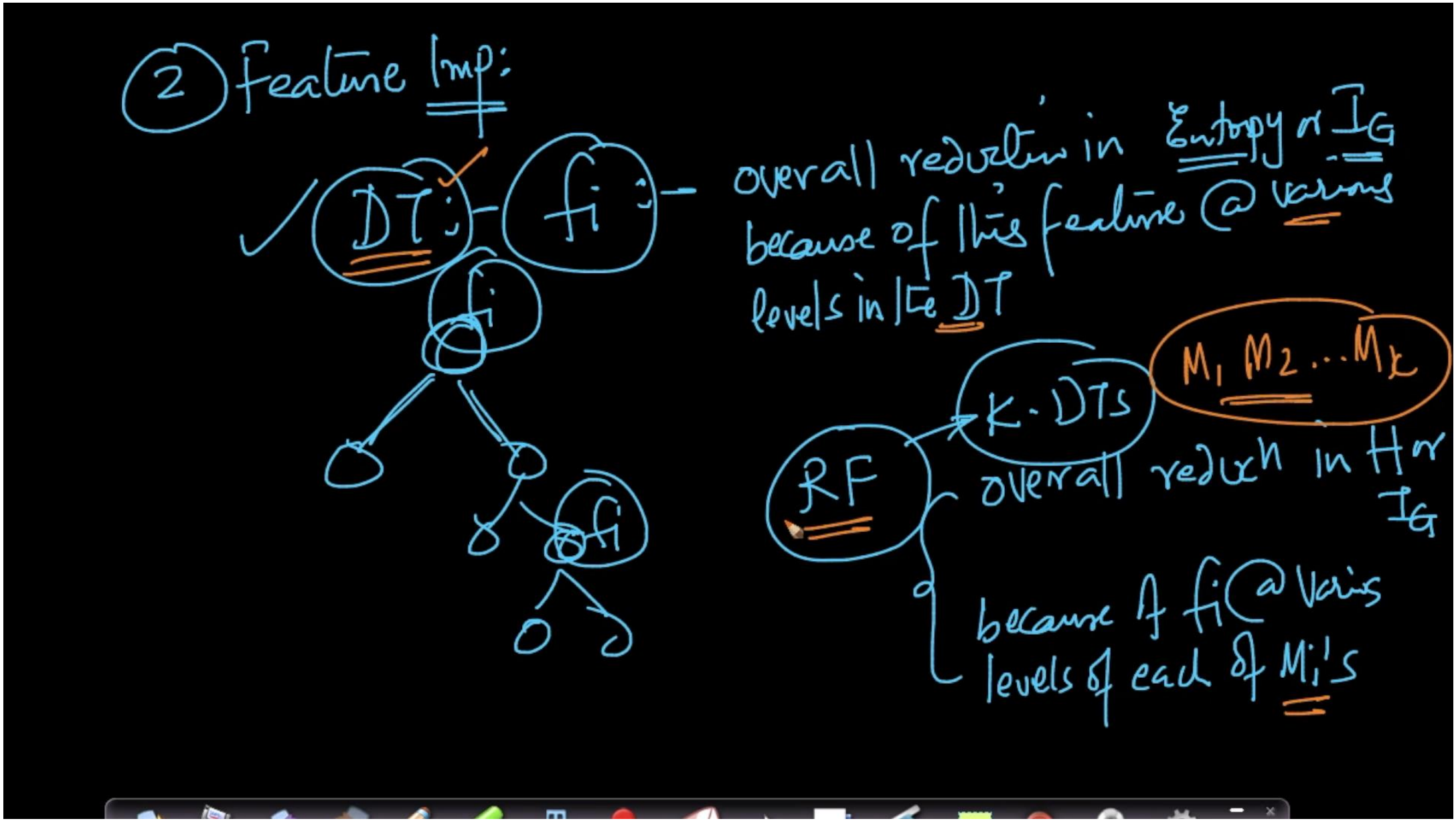
Randomization as a way to reduce variance

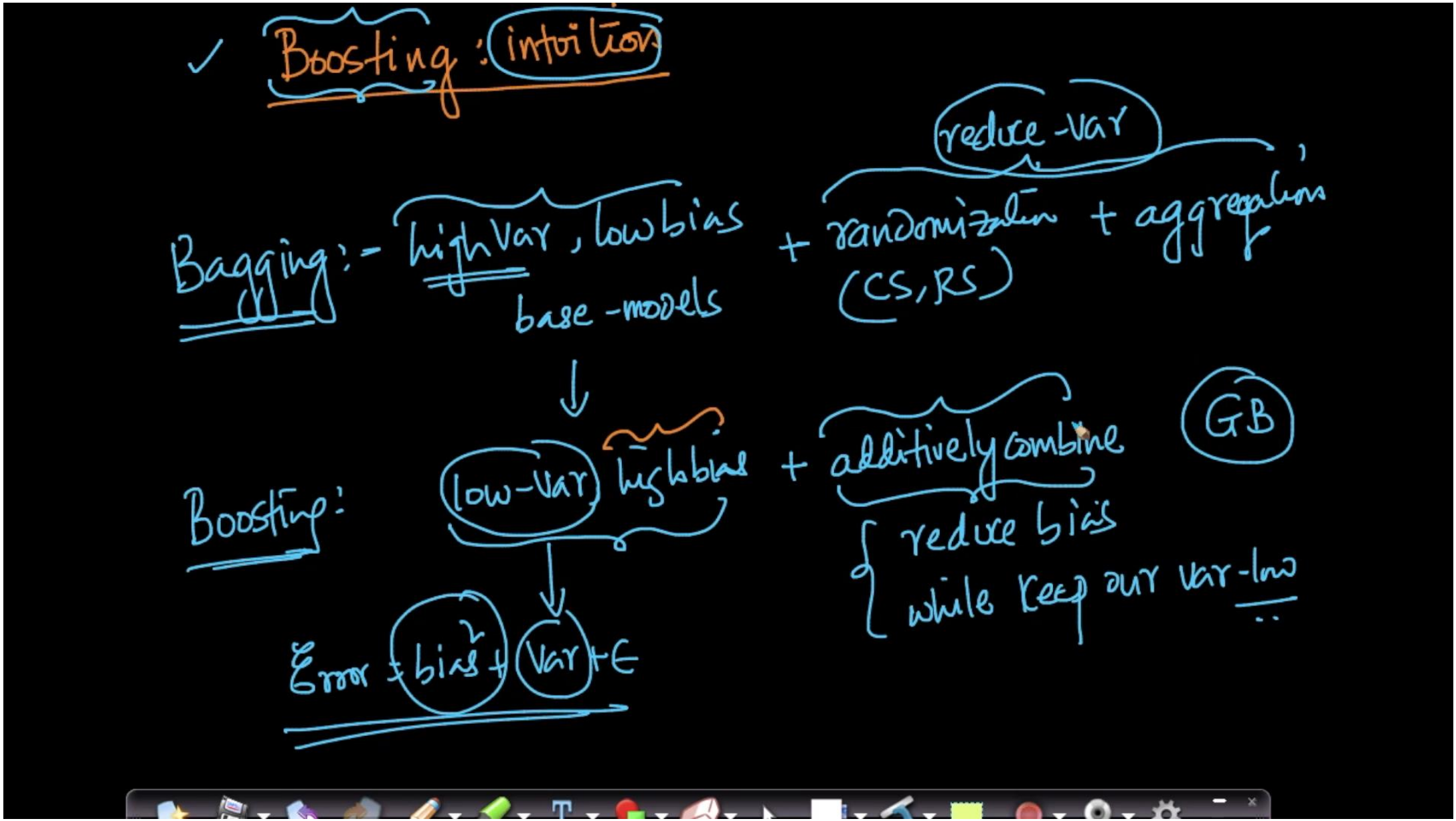
RF → C.S & r.s

ET → C.S + r.s + random 'c'

(reduce variance better than RF)


Cases:
RF :-
DT
+
RS + CS + agg
reduce - Var
large - dim; Cat - features with many categories
Cases for DT
DT → depth
1
Bias - Variance
of RF
K = # base learners
↳ deep Trees


2 Feature Imp:
DT
fi
overall reduction in Entropy or IG because of this feature @ various levels in the DT
RF
K-DTs
M1 M2 ... Mk
overall reduction in H or IG
because of fi @ various levels of each of Mi's


Boosting : intuition
Bagging :- high Var, low bias
base-models
+ randomization + aggregation
(CS, RS)
reduce-Var
Boosting :
low-Var high bias
+ additively combine
GB
reduce bias
while keep our var-low
Error = bias² + Var + ∈

Core-idea :- reduces bias

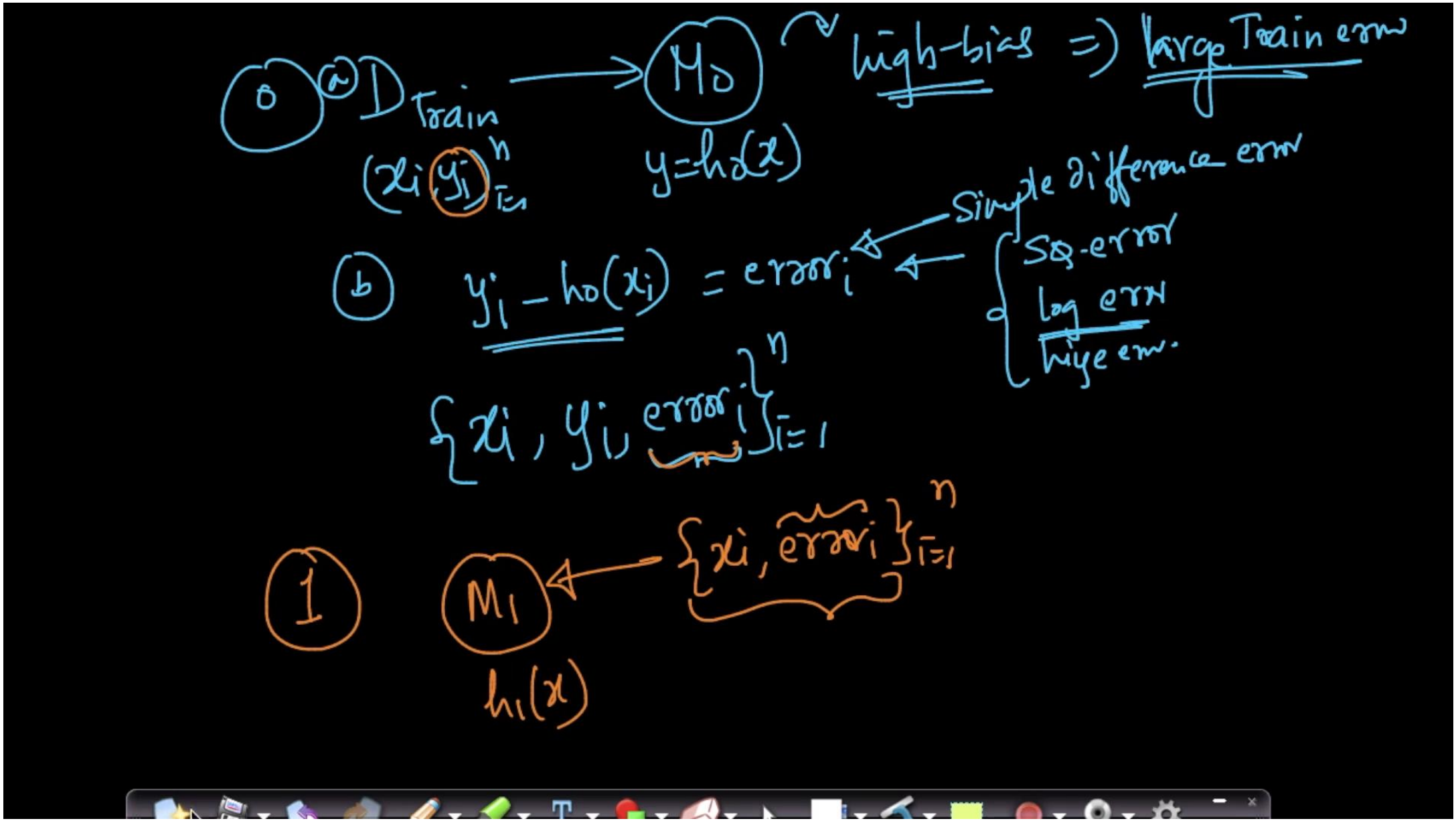
{classfn ✓
regssn ✓

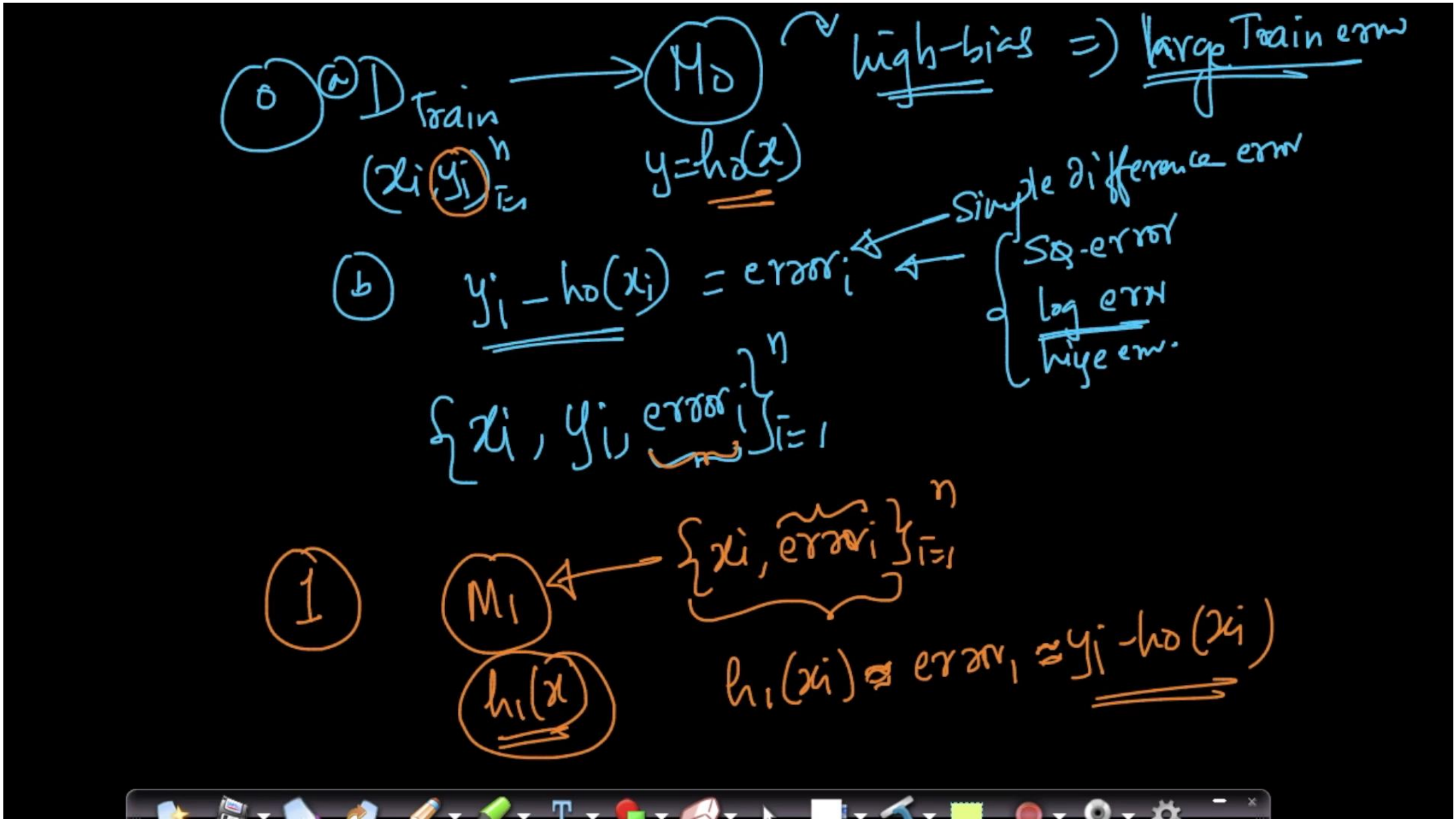
$D_{train} = \{x_i, y_i\}_{i=1}^{n}$

① $D_{train} \{x_i, y_i\}_{i=1}^{n} \rightarrow$ (M0)

↳ high-bias { low-var } { e.g: DT which is shallow }

high-bias ⇒ high Train-error


0
$\mathcal{D}_{Train}$
$(x_i, y_i)_{i=1}^{n}$
M_0
$y = h_0(x)$
high-bias ⇒ large Train error
b
$y_i - h_0(x_i) = error_i$
Simple difference error
Sq-error
log error
hinge error.
$\{x_i, y_i, error_i\}_{i=1}^{n}$
1
M_1
$\{x_i, error_i\}_{i=1}^{n}$
$h_1(x)$


0
$\mathcal{D}$
Train
$(x_i, y_i)_{i=1}^n$
M_0
$y = h_0(x)$
high-bias ⇒ large Train error
b
$y_i - h_0(x_i) = error_i$
Simple difference error
Sq-error
log error
hinge error.
$\{x_i, y_i, error_i\}_{i=1}^n$
1
M_1
$\{x_i, error_i\}_{i=1}^n$
$h_1(x)$
$h_1(x_i) \approx error_i \approx y_i - h_0(x_i)$

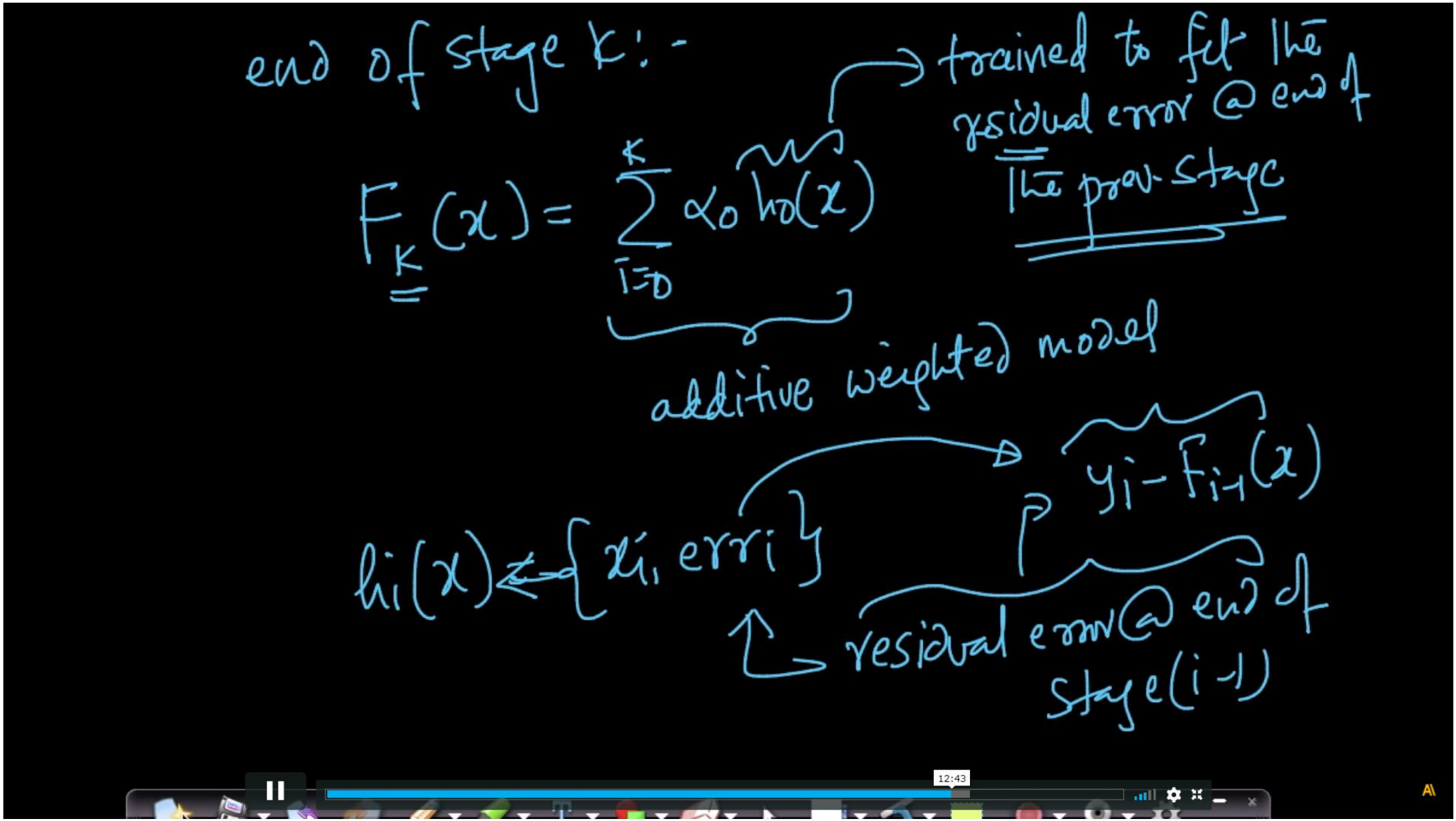
$F_1(x)$ = model at end of Stage 1

$F_1(x) = \alpha_0 h_0(x) + \alpha_1 h_1(x)$: - weighted sum of 2 - base models

(2) $\{x_i, err_i\} \longrightarrow m_2 \; h_2(x)$

$err_i = y_i - F_1(x_i)$

$F_2(x) = \alpha_0 h_0(x) + \alpha_2 h_2(x) + \alpha_1 h_1(x)$


end of stage K:-
$F_K(x) = \sum_{i=0}^{K} \alpha_0 h_0(x)$
trained to fit the residual error @ end of the prev. stage
additive weighted model
$h_i(x) \leftarrow \{x_i, err_i\}$
$y_i - F_{i-1}(x)$
residual error @ end of stage (i-1)

$k=0$

$f_k(x) = \sum_{i=0}^{k} \alpha_i h_i(x)$

$f_i(x) \rightarrow f_{i+1}(x)$

$h_{i+1}(x)$

ends up having a low residual error

Train error reduces ↓ ⇒ bias ↓

Gradient Boosted DT (GBDT) ✓ → internet

Adaboosting → images → face detection

Residuals, Loss-fns & Gradients

→ {Gradient Boosting}

$$F_k(x) = \sum_{i=0}^{k} \alpha_i h_i(x)$$

residual @ end of stage k → $err_i = y_i - F_k(x)$

↑ Residual

$M_{k+1} \leftarrow \{x_i, err_i\}$

{ Loss-minimization:- ✓ Logistic-loss ← classfn
lr. regr ← Sq. loss ← easier
SVM ← hinge-loss

regression:

$$\mathcal{L}(y_i, F_k(x_i)) = (y_i - \underbrace{F_k(x_i)}_{z_i})^2$$

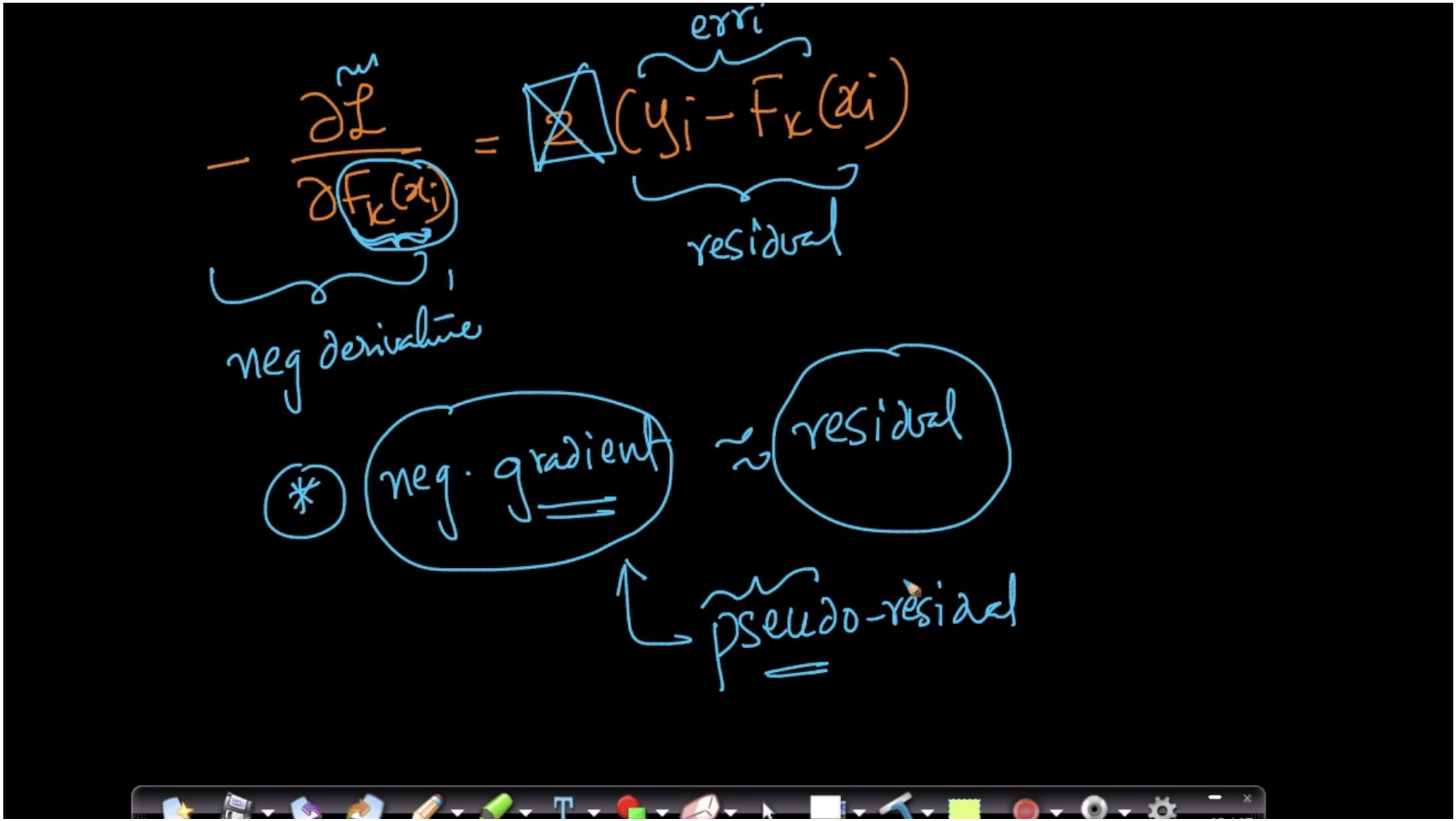
{ let $F_k(x_i) = z_i$

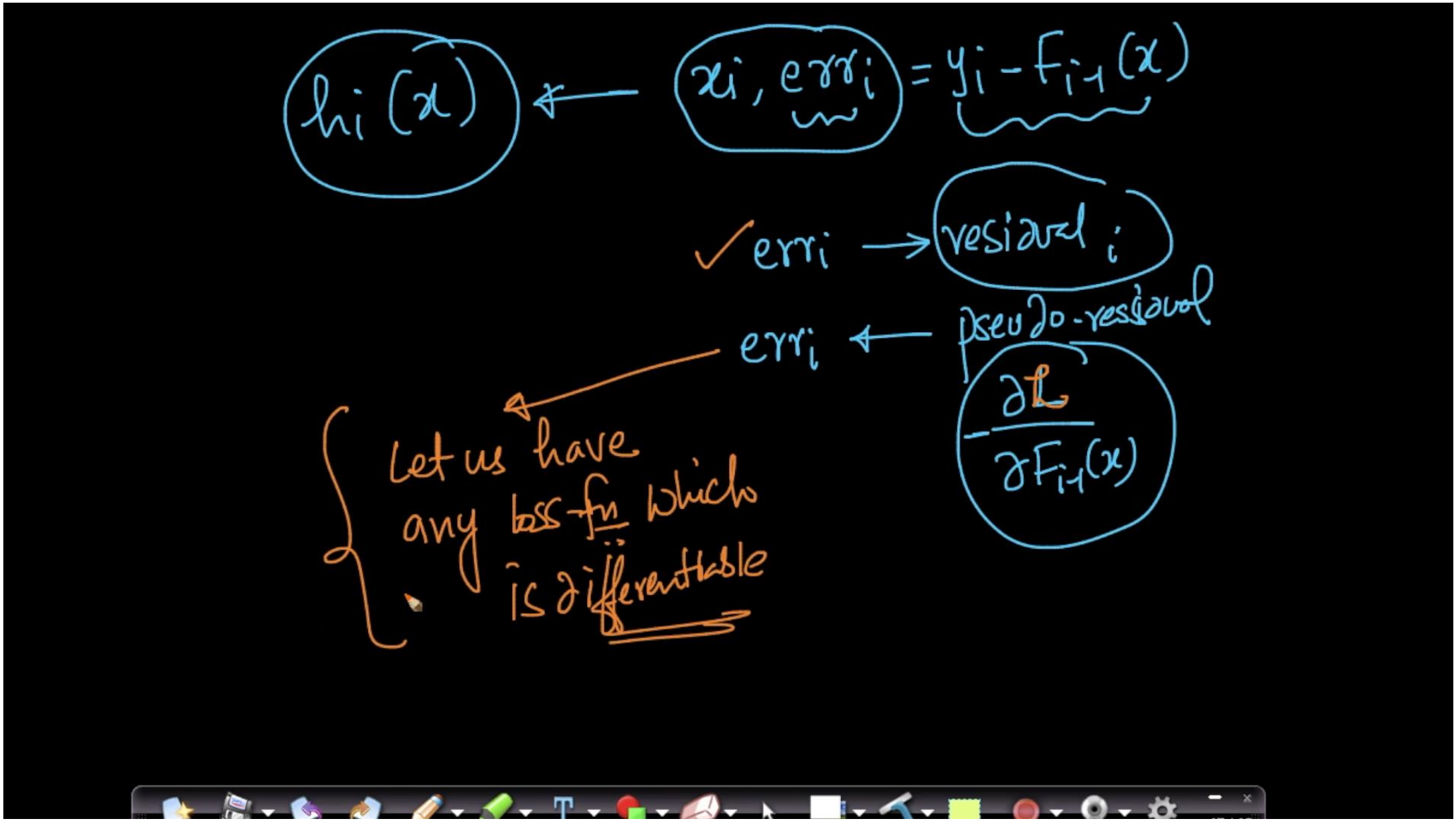
Sq loss

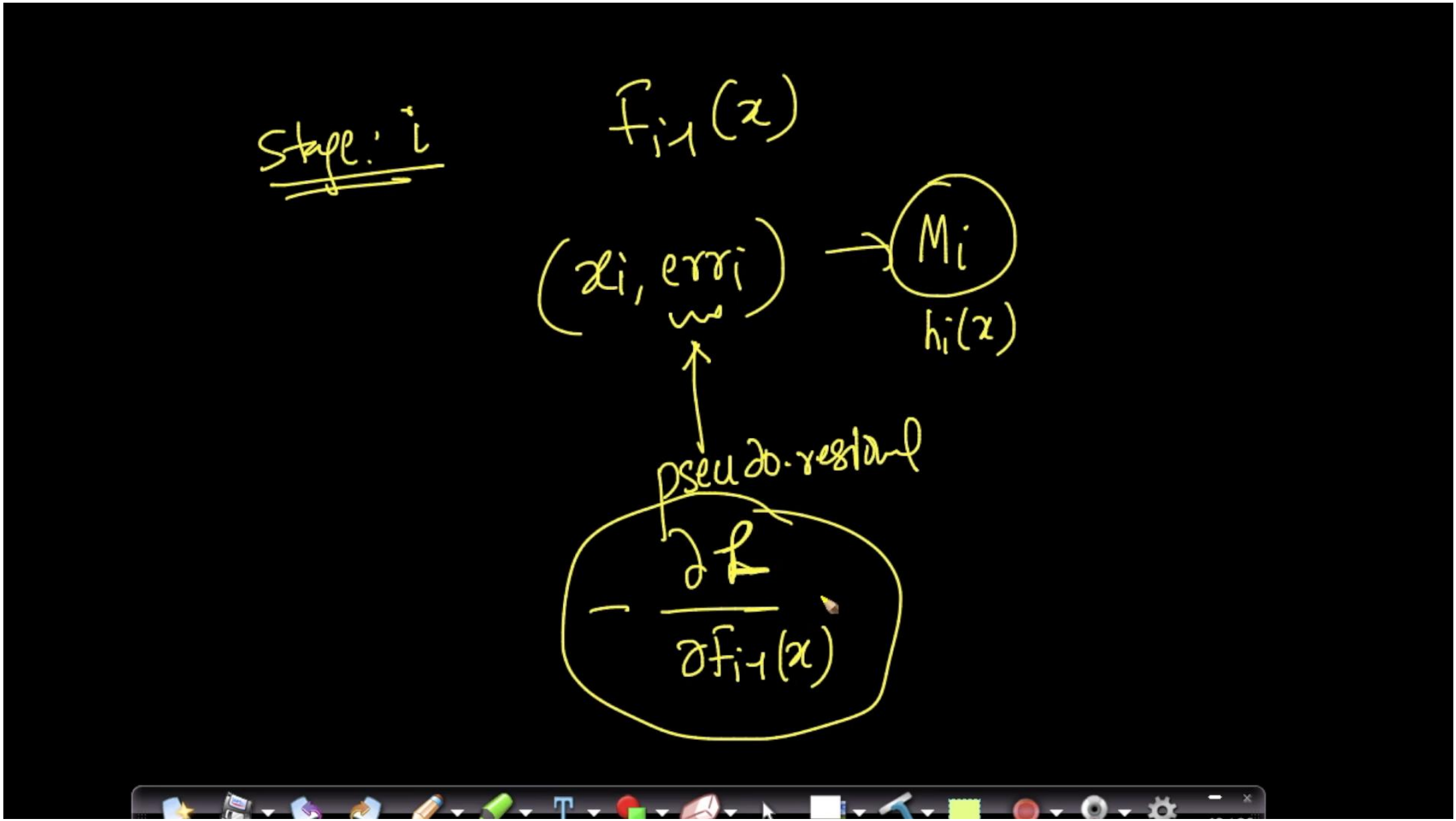
$$\frac{\partial \mathcal{L}}{\partial F_k(x_i)} = \frac{\partial \mathcal{L}}{\partial z_i} = \frac{\partial}{\partial z_i}(y_i - z_i)^2$$

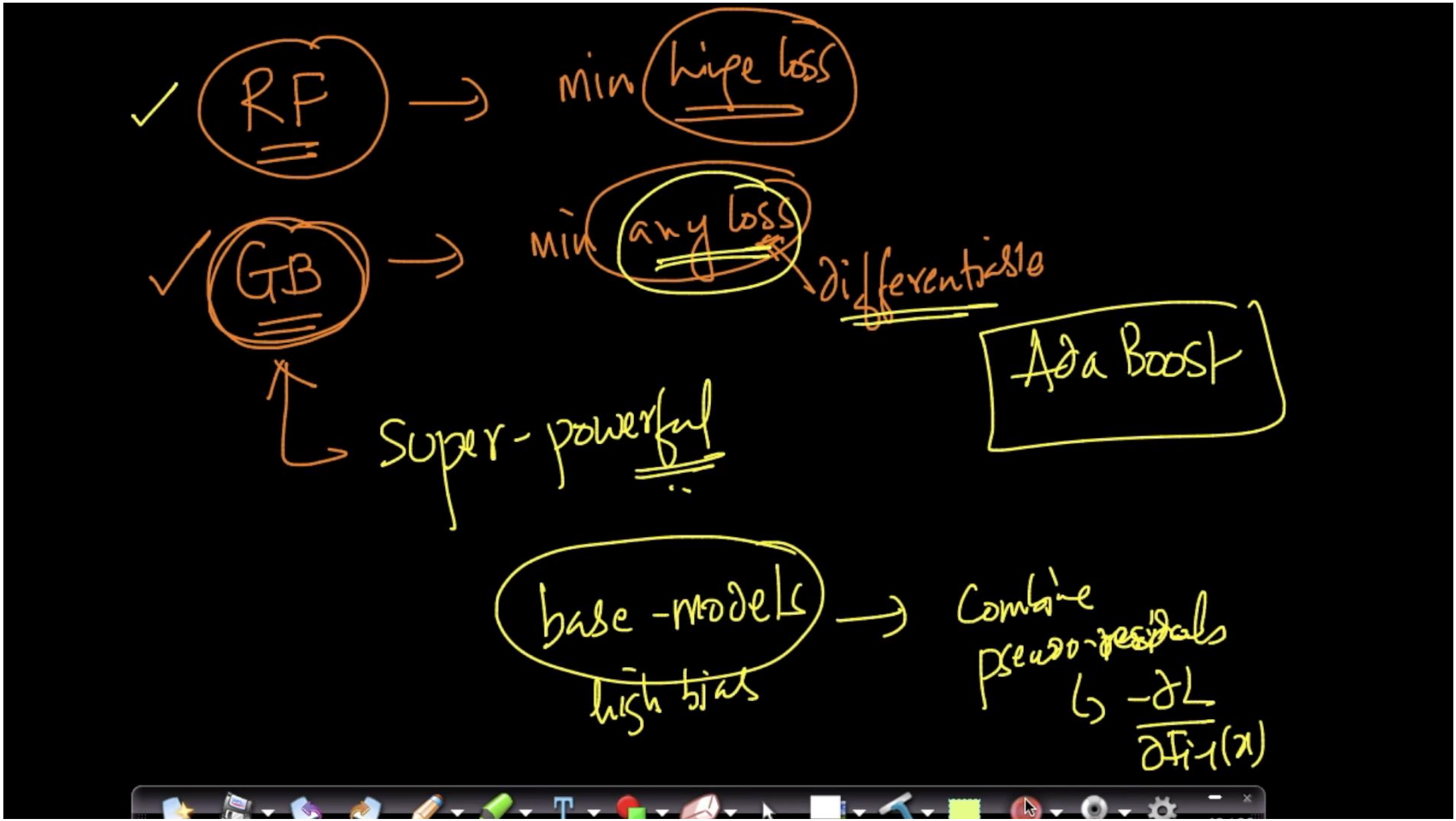
$$= (-1) * 2 * (y_i - z_i)$$

$$\partial L / \partial z_i = -2(y_i - z_i)$$


$-\frac{\partial \tilde{L}}{\partial F_k(x_i)} = 2(y_i - F_k(x_i))$
err_i
residual
neg derivative
(*) neg. gradient ≈ residual
pseudo-residual


$h_i(x)$
$x_i, err_i = y_i - F_{i-1}(x)$
err_i → residual $_i$
err_i ← pseudo-residual
$-\frac{\partial L}{\partial F_{i-1}(x)}$
Let us have any loss-fn which is differentiable


Step: i
$f_{i+1}(x)$
(x_i, err_i)
M_i
$h_i(x)$
pseudo-residual
$-\frac{\partial L}{\partial f_{i+1}(x)}$


RF
min
huge loss
GB
min
any loss
differentiable
Ada Boost
Super-powerful
base-models
high bias
Combine
pseudo-residuals
$-\frac{\partial L}{\partial F_{i-1}(x)}$

Contents - Google Docs | 3.2.4.3.5. sklearn.ensemble.Gr | Gradient boosting - Wikipedia | Chekuri Srikan...

Secure | https://en.wikipedia.org/wiki/Gradient_boosting

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

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

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

logistic-loss
SQ-loss
hinge loss

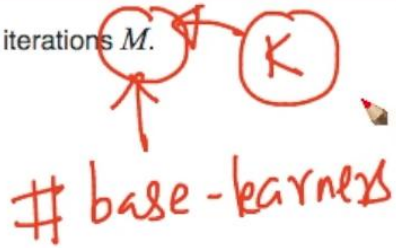
Gradient tree boosting [edit]

Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners. For this special

Contents - Google Docs | 3.2.4.3.5. sklearn.ensemble.Gr | Gradient boosting - Wikipedia | Chekuri Srikan...

Secure | https://en.wikipedia.org/wiki/Gradient_boosting

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.



Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

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

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Gradient tree boosting [edit]

Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners. For this special

Contents - Google Docs | 3.2.4.3.5. sklearn.ensemble.Gr | Gradient boosting - Wikipedia | Chekuri Srikan...

Secure | https://en.wikipedia.org/wiki/Gradient_boosting

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

(SQ-loss) ← constant

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

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

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Gradient tree boosting [edit]

Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners. For this special

02:07 / 18

APPLIED COURSE

Contents - Google Docs | 3.2.4.3.5. sklearn.ensemble.Gr | Gradient boosting - Wikipedia | Chekuri Srikan...

Secure | https://en.wikipedia.org/wiki/Gradient_boosting

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

zeroth model

$\hat{y}_i = \text{mean}(y_i)$

$\arg\min_{\gamma} \sum_{i=1}^{n} (y_i - \gamma)^2 \Rightarrow \gamma = \text{mean}(y_i)$

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

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

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Gradient tree boosting [edit]

Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners. For this special

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

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

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Gradient tree boosting [edit]

Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners. For this special

Contents - Google Docs | 3.2.4.3.5. sklearn.ensemble.Gr | Gradient boosting - Wikipedia | Chekuri Srikan...

Secure | https://en.wikipedia.org/wiki/Gradient_boosting

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

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

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_\gamma \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Gradient tree boosting [edit]

Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners. For this special

Contents - Google Docs | 3.2.4.3.5. sklearn.ensemble.Gr | Gradient boosting - Wikipedia | Chekuri Srikan...

Secure | https://en.wikipedia.org/wiki/Gradient_boosting

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For $m = 1$ to M:
   1. Compute so-called *pseudo-residuals*:

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

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Gradient tree boosting [edit]

Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners. For this special

Contents - Google Docs | 3.2.4.3.5. sklearn.ensemble.Gr | Gradient boosting - Wikipedia | Chekuri Srikan...

Secure | https://en.wikipedia.org/wiki/Gradient_boosting

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

$F_m(x)$

2. For m = 1 to M:

   1. Compute so-called *pseudo-residuals*:

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

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

$\{ F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$

3. Output $F_M(x)$.

Gradient tree boosting [edit]

Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners. For this special

Contents - Google Docs | 3.2.4.3.5. sklearn.ensemble.Gr | Gradient boosting - Wikipedia | Chekuri Srikan...

Secure | https://en.wikipedia.org/wiki/Gradient_boosting

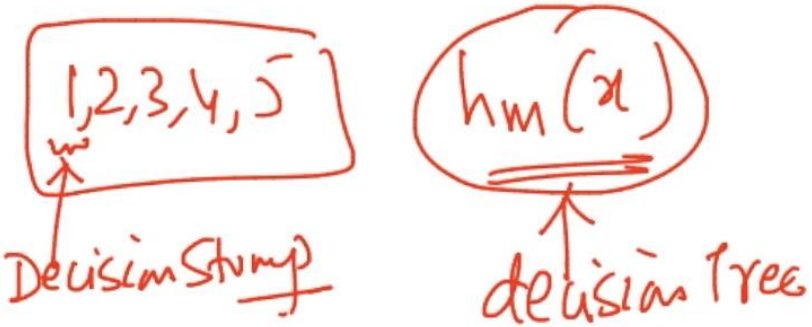
Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:



1. Compute so-called *pseudo-residuals*:

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

2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

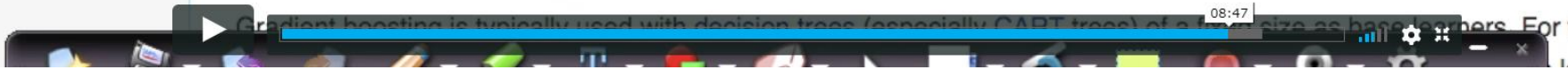
4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Gradient tree boosting [edit]

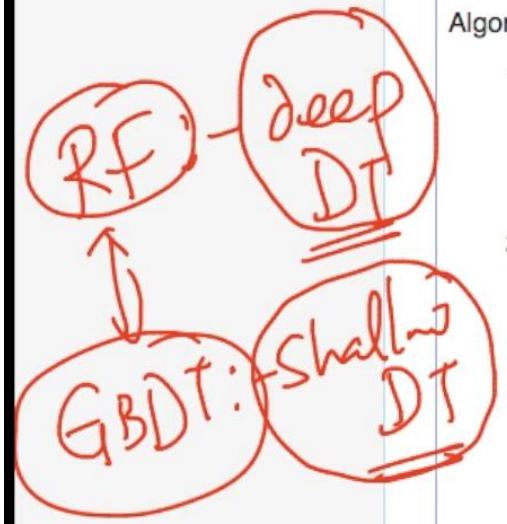
Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners. For this special



APPLIED COURSE

Contents - Google Docs | 3.2.4.3.5. sklearn.ensemble.Gr | Gradient boosting - Wikipedia | Chekuri Srikan...

Secure | https://en.wikipedia.org/wiki/Gradient_boosting



Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

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

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

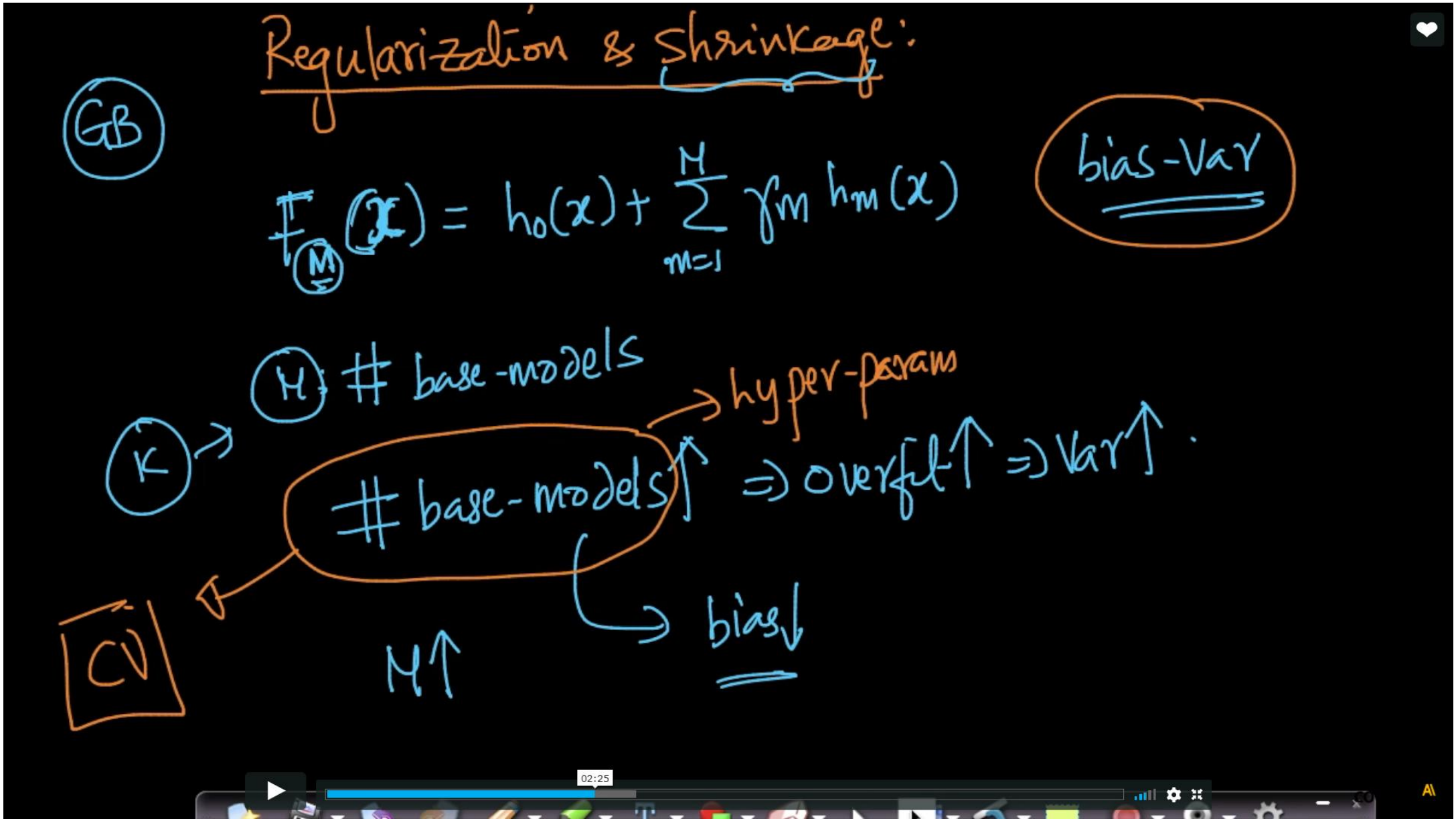
3. Output $F_M(x)$.

Gradient tree boosting [edit]

Gradient boosting is typically used with decision trees (especially CART trees) of a fixed ... base learners. For this spe... learner

09:21

APPLIED COURSE


Regularization & Shrinkage:
GB
$F_M(x) = h_0(x) + \sum_{m=1}^{M} \gamma_m h_m(x)$
bias-Var
M: # base-models
K
base-models
hyper-params
⇒ overfit ↑ ⇒ Var ↑
CV
M ↑
bias ↓
02:25

Regularization [edit]

Fitting the training set too closely can lead to degradation of the model's generalization ability. Several so-called regularization techniques reduce this overfitting effect by constraining the fitting procedure.

One natural regularization parameter is the number of gradient boosting iterations M (i.e. the number of trees in the model when the base learner is a decision tree). Increasing M reduces the error on training set, but setting it too high may lead to overfitting. An optimal value of M is often selected by monitoring prediction error on a separate validation data set. Besides controlling M, several other regularization techniques are used.

GB

$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$

Shrinkage [edit]

An important part of gradient boosting method is regularization by shrinkage which consists in modifying the update rule as follows:

$$F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m h_m(x), \quad 0 < \nu \leq 1,$$

where parameter ν is called the "learning rate".

ν is small

0.0001

$\nu \approx 0.1$

↑ hyperpara → CV

Empirically it has been found that using small learning rates (such as $\nu < 0.1$) yields dramatic improvements in model's generalization ability over gradient boosting without shrinking ($\nu = 1$).[7] However, it comes at the price of increasing computational time both during training and querying: lower learning rate requires more iterations.

Stochastic gradient boosting [edit]

Soon after the introduction of gradient boosting Friedman proposed a minor modification to the algorithm, motivated by Breiman's bagging method.[3] Specifically, he proposed that at each iteration of the algorithm, a base learner should be fit on a

Regularization [edit]

Fitting the training set too closely can lead to degradation of the model's generalization ability. Several so-called regularization techniques reduce this overfitting effect by constraining the fitting procedure.

One natural regularization parameter is the number of gradient boosting iterations M (i.e. the number of trees in the model when the base learner is a decision tree). Increasing M reduces the error on training set, but setting it too high may lead to overfitting. An optimal value of M is often selected by monitoring prediction error on a separate validation data set. Besides controlling M, several other regularization techniques are used.

Shrinkage [edit]

An important part of gradient boosting method is regularization by shrinkage which consists in modifying the update rule as follows:

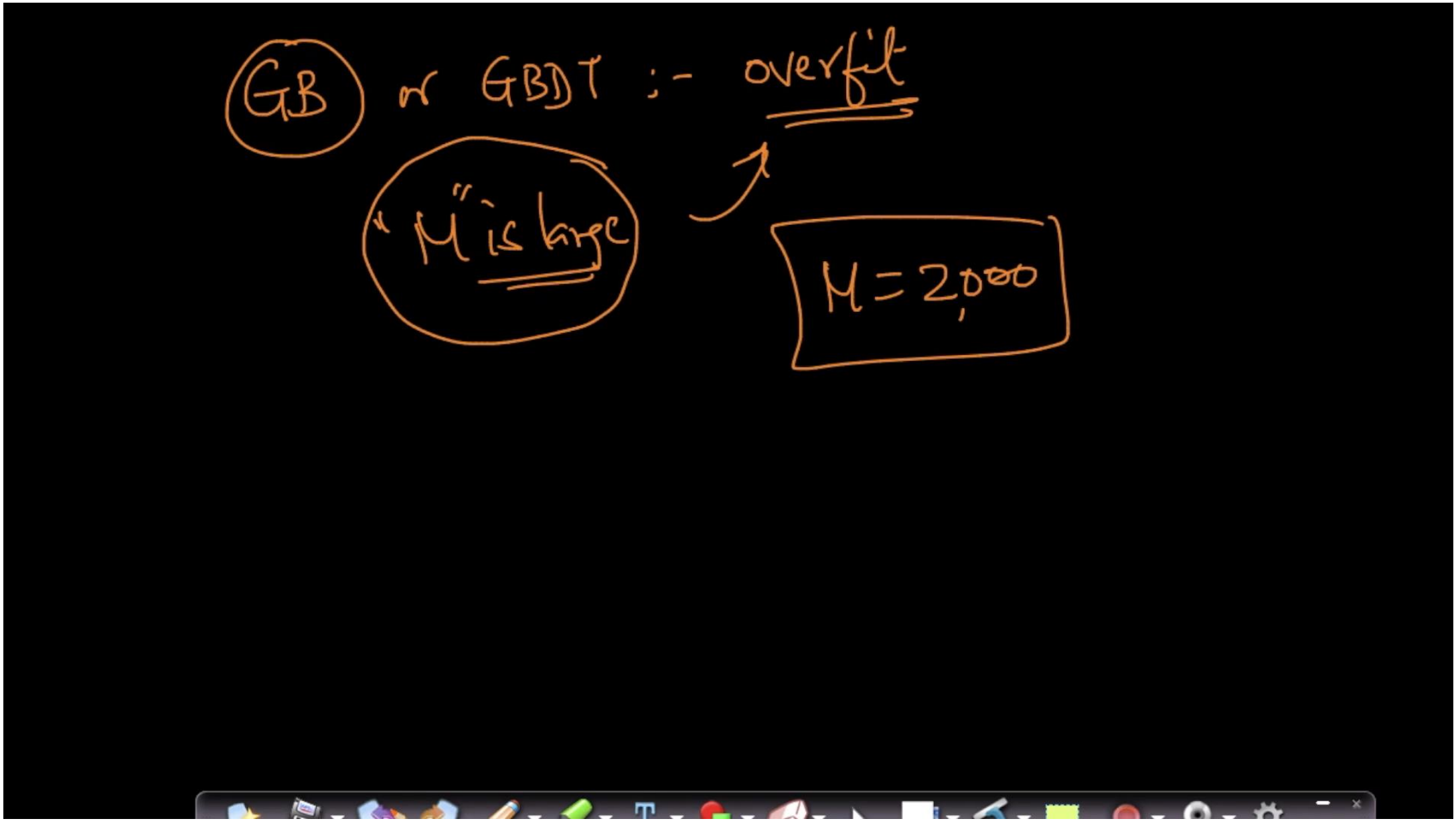
$$F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m h_m(x), \quad 0 < \nu \leq 1,$$

where parameter ν is called the "learning rate".

Empirically it has been found that using small learning rates (such as $\nu < 0.1$) yields dramatic improvements in model's generalization ability over gradient boosting without shrinking ($\nu = 1$).[7] However, it comes at the price of increasing computational time both during training and querying: lower learning rate requires more iterations.

Stochastic gradient boosting [edit]

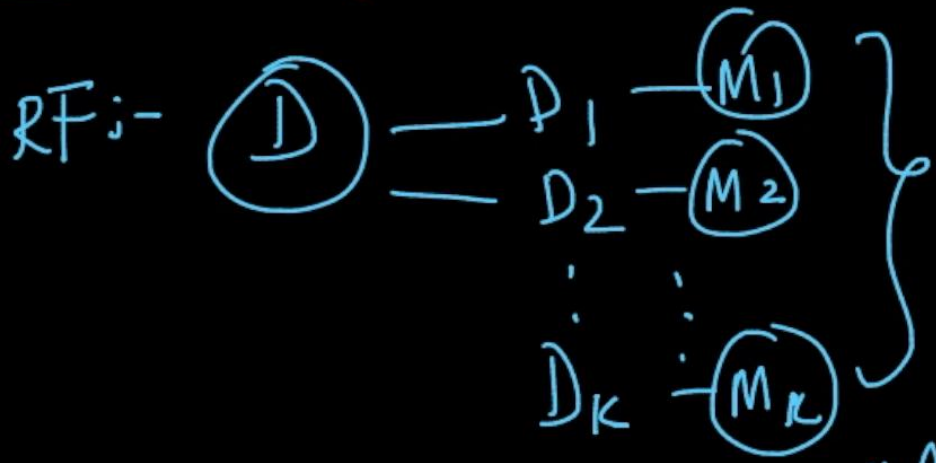
Soon after the introduction of gradient boosting Friedman proposed a minor modification to the algorithm, motivated by Breiman's bagging method.[3] Specifically, he proposed that at each iteration of the algorithm, a base learner should be fit on a


GB or GBDT :- overfit
"M" is large
M = 2,000

Train & Run time complexity (GB)

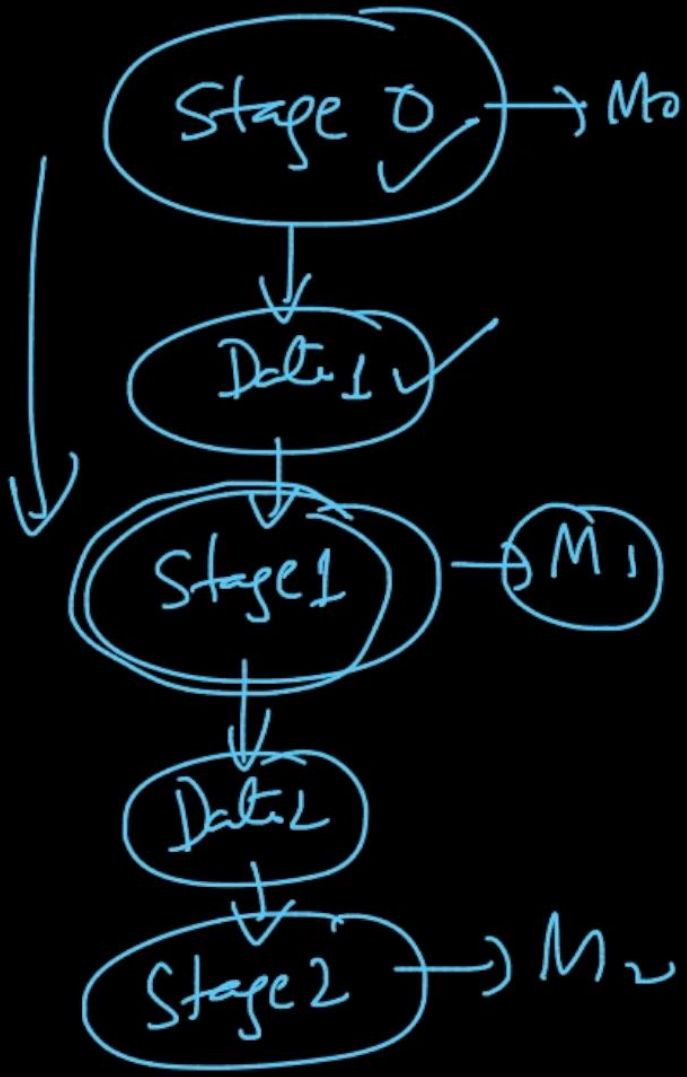
Train: $O(n \lg n \, d * M)$ M : #base-learners

RF:-

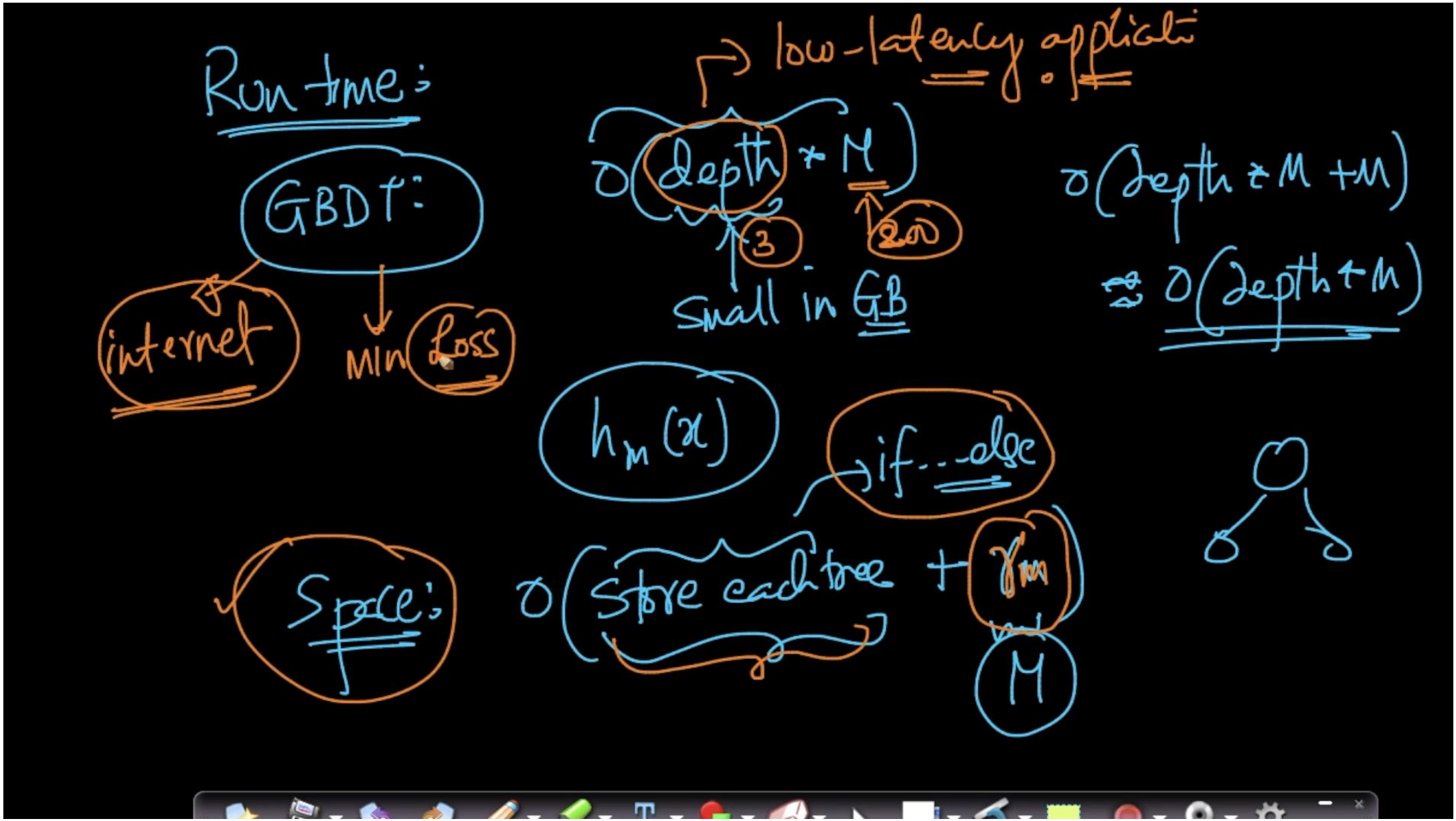


Trivially parallelizable

GBDT:- not easy to parallelize:-



{ GBDT take more time to train than RF


Run time:
GBDT:
internet
Min Loss
low-latency applicati
O(depth * M)
3
200
small in GB
O(depth * M + M)
≈ O(depth * M)
$h_m(x)$
if...else
Space:
O(store each tree + γ_m
M

Contents - Google Docs | 3.2.4.3.5. sklearn.ensemble | Python API Reference — xg | xgboost/sklearn_examples | How to Develop Your First | Chekuri Srikan...

xgboost.readthedocs.io/en/latest/python/python_api.html

XGBoost | Get Started | Tutorials | How To | Packages | Knobs | Search

Scikit-Learn API

Scikit-Learn Wrapper interface for XGBoost.

class `xgboost.` **`XGBRegressor`** ***(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True, objective='reg:linear', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1, max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)***

Bases: `xgboost.sklearn.XGBModel` , `object`

Implementation of the scikit-learn API for XGBoost regression.

Parameters

max_depth : *int*

Maximum tree depth for base learners.

learning_rate : *float*

Boosting learning rate (xgb's "eta")

n_estimators : *int*

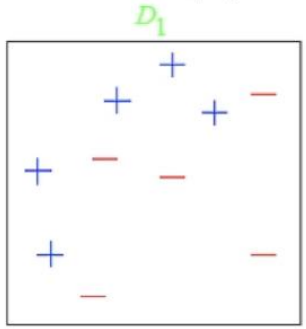
Number

Contents - Google Docs | CIS520 Machine Learning | Python API Reference — xg | xgboost/sklearn_examples | How to Develop Your First | Chekuri Srikan...

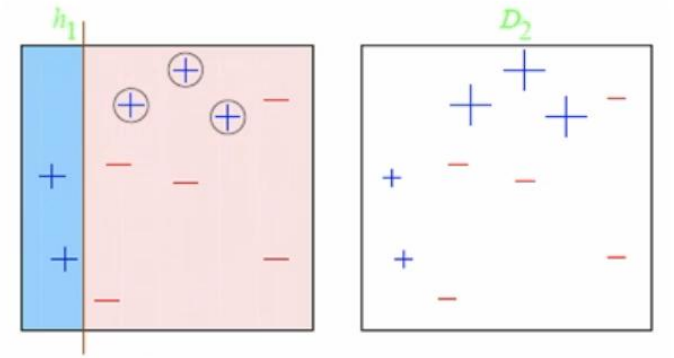
Secure | https://alliance.seas.upenn.edu/~cis520/wiki/index.php?n=lectures.boosting

A toy example from Schapire's tutorial

Consider this toy dataset in 2D and assume that our weak classifiers are decision stumps (vertical or horizontal half-planes):



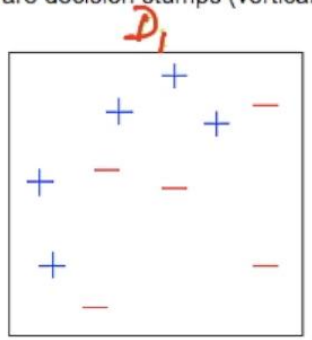
The first round:



The second round:

A toy example from Schapire's tutorial

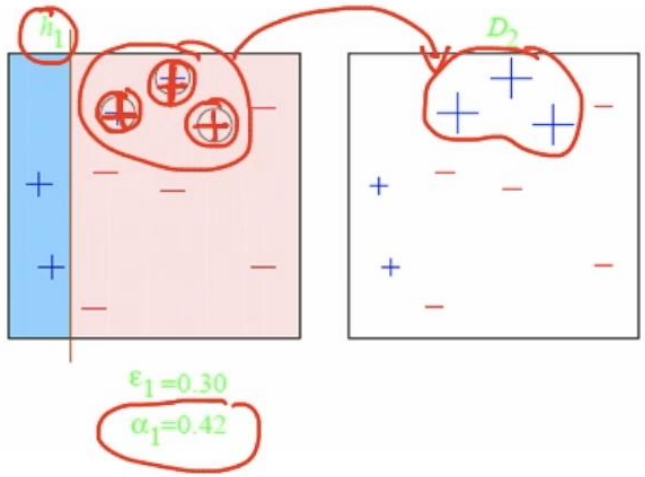
Consider this toy dataset in 2D and assume that our weak classifiers are decision stumps (vertical or horizontal half-planes):



DT. with depth = 1
↓
Decision Stump

The first round:

$\alpha_1 h_1(x)$

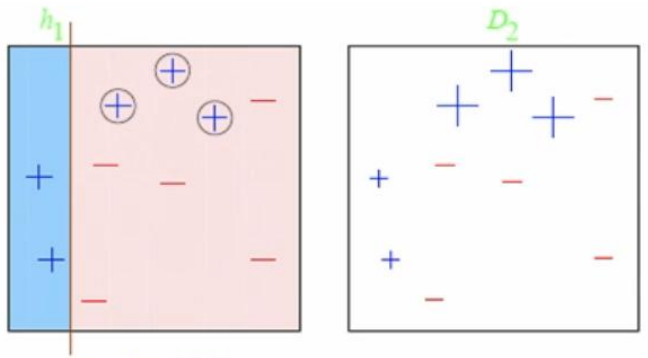


weighted models
↓
upsample

The second round:

The first round:

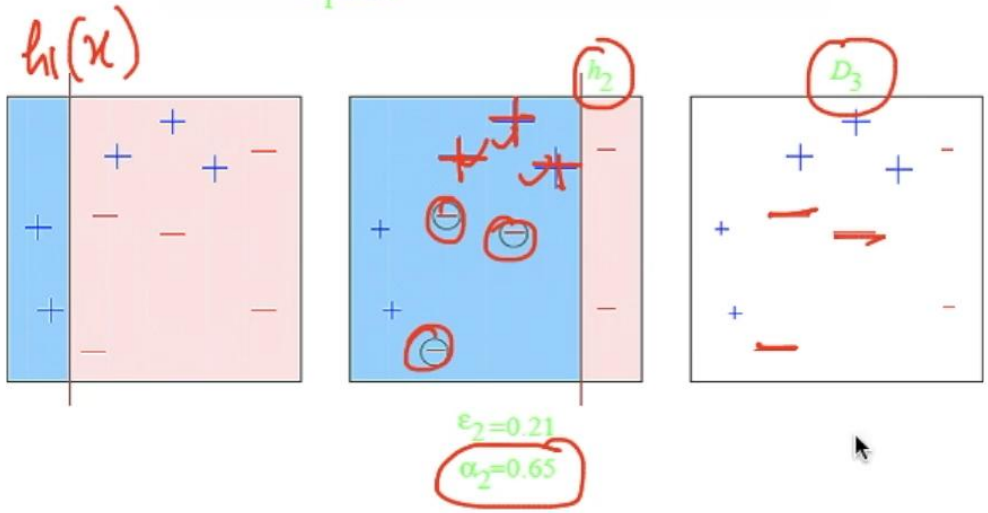
$\alpha_1 h_1$



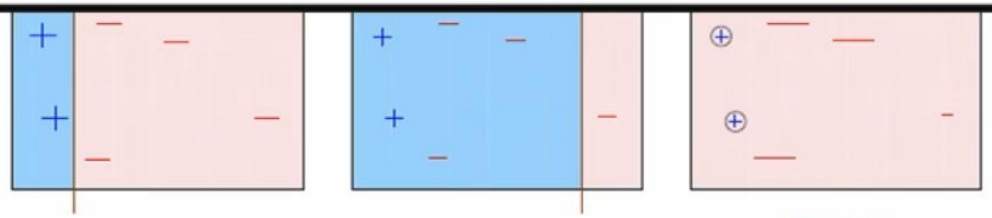
exponentially

The second round:

$\alpha_2 h_2$



The third round:



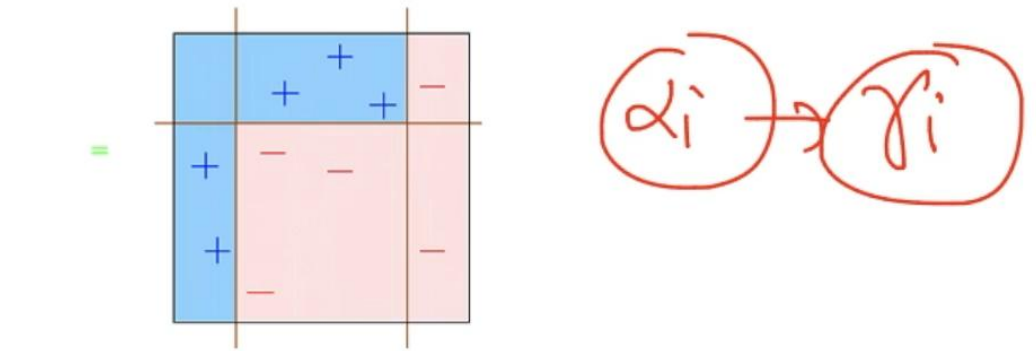
$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

The final classifier

$F_3(x) = \alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3$

$H_{\text{final}} = \text{sign}$ (α_1 0.42 + α_2 0.65 + α_3 0.92)



$\alpha_i \rightarrow \gamma_i$

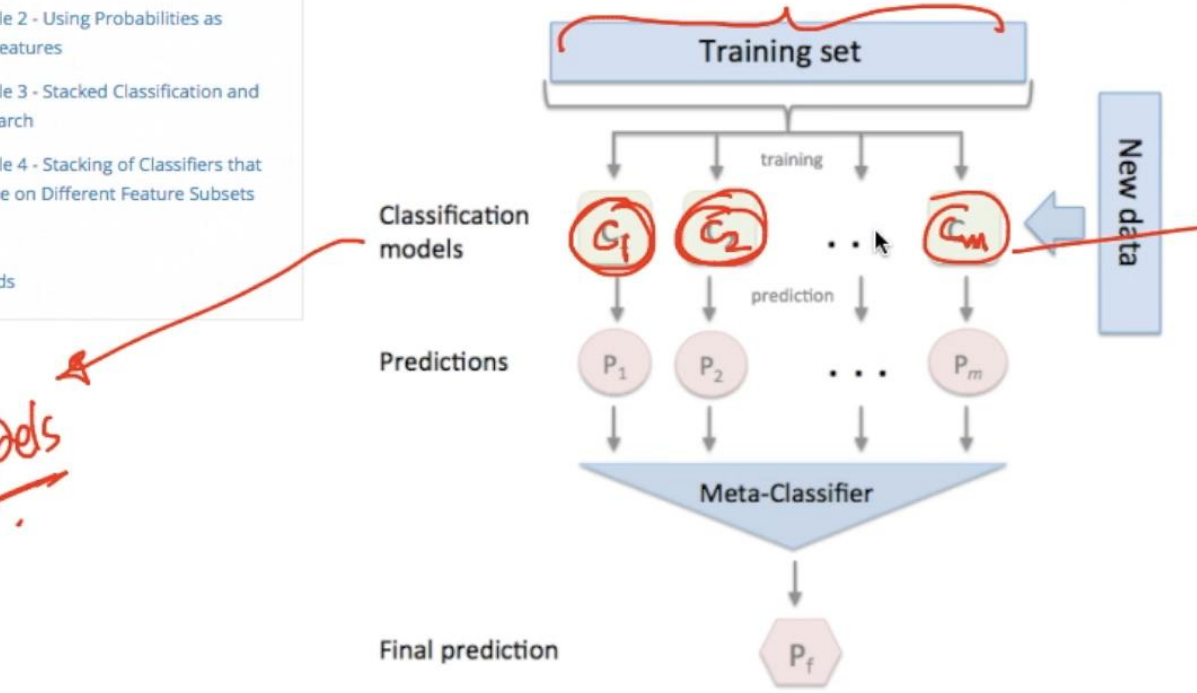
Contents - Google Docs | StackingClassifier - mlxtend | Chekuri Srikan...

Secure | https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

mlxtend | Home | User Guide | API | Installation | About | Search | Previous | Next | GitHub

Overview

Stacking is an ensemble learning technique to combine multiple classification models via a meta-classifier. The individual classification models are trained based on the complete training set; then, the meta-classifier is fitted based on the outputs -- meta-features -- of the individual classification models in the ensemble. The meta-classifier can either be trained on the predicted class labels or probabilities from the ensemble.



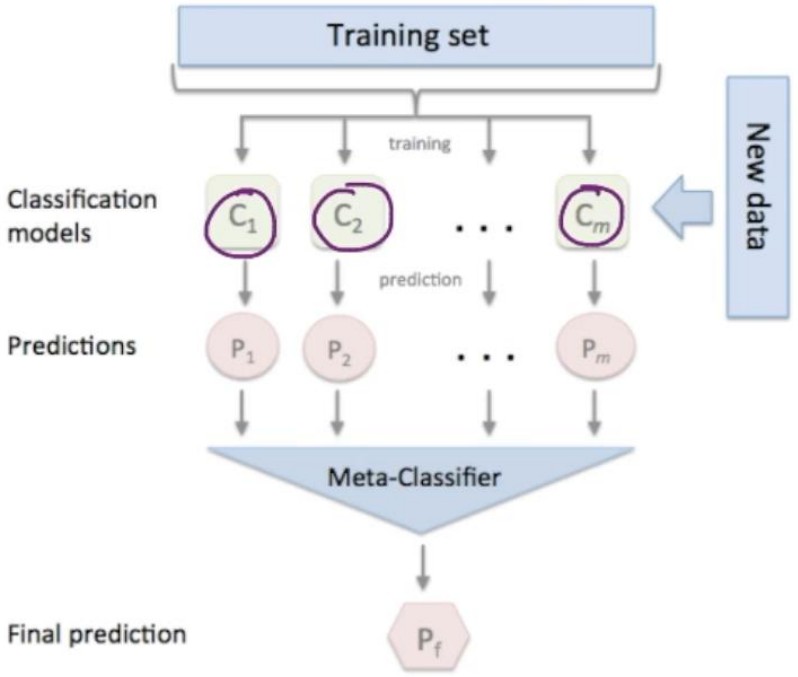
Contents - Google Docs | StackingClassifier - mlxtend | Chekuri Srikan...

Secure | https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

mlxtend | Home | User Guide | API | Installation | About | Search | Previous | Next | GitHub

Overview

Stacking is an ensemble learning technique to combine multiple classification models via a meta-classifier. The individual classification models are trained based on the complete training set; then, the meta-classifier is fitted based on the outputs -- meta-features -- of the individual classification models in the ensemble. The meta-classifier can either be trained on the predicted class labels or probabilities from the ensemble.



stacking :-

C_1 : LR-SUM
C_2 : RBF-SUM
C_3 : GBDT(200)
C_4 : NB
C_5 : K-NN (K=5)
C_6 : K-NN (K=3)

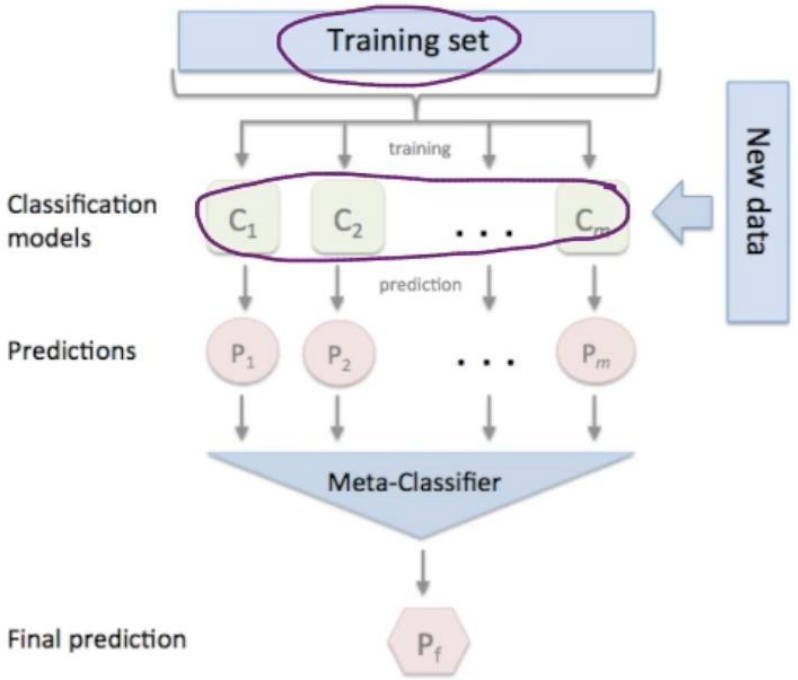
Contents - Google Docs | StackingClassifier - mlxtend | Chekuri Srikan...

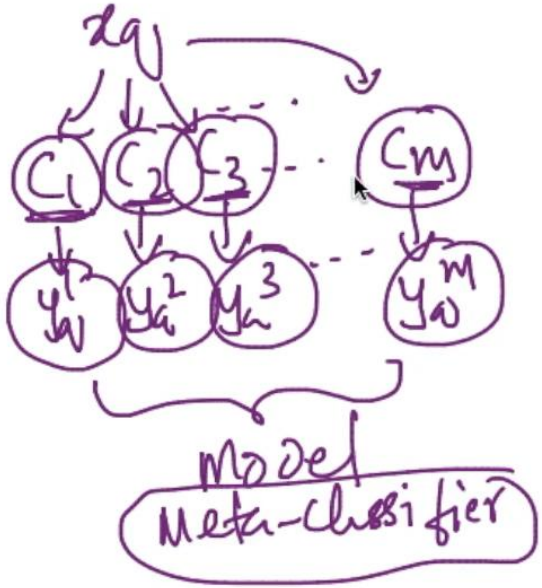
Secure | https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

mlxtend | Home | User Guide | API | Installation | About | Search | Previous | Next | GitHub

Overview

Stacking is an ensemble learning technique to combine multiple classification models via a meta-classifier. The individual classification models are trained based on the complete training set; then, the meta-classifier is fitted based on the outputs -- meta-features -- of the individual classification models in the ensemble. The meta-classifier can either be trained on the predicted class labels or probabilities from the ensemble.





RF
C → high-var low-bias
agg → Maj-vote

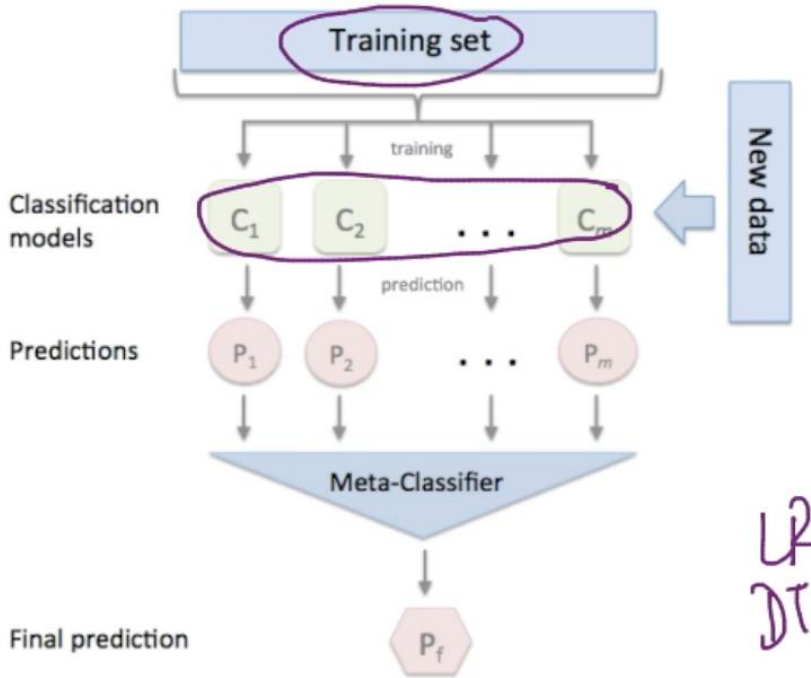
Contents - Google Docs | StackingClassifier - mlxtend | Chekuri Srikan...

Secure | https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

mlxtend | Home | User Guide | API | Installation | About | Search | Previous | Next | GitHub

Overview

Stacking is an ensemble learning technique to combine multiple classification models via a meta-classifier. The individual classification models are trained based on the complete training set; then, the meta-classifier is fitted based on the outputs -- meta-features -- of the individual classification models in the ensemble. The meta-classifier can either be trained on the predicted class labels or probabilities from the ensemble.



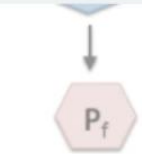
RF
C → high-var low-bias
agg → low-var

LR
DT ..
Model
Meta-classifier

Contents - Google Docs | StackingClassifier - mlxtend | Chekuri Srikan...

Secure | https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

mlxtend | Home | User Guide | API | Installation | About | Search | Previous | Next | GitHub



The algorithm can be summarized as follows (source: [1]):

Algorithm 19.7 Stacking

Input: Training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{m}$ $(\mathbf{x}_i \in \mathbb{R}^n, y_i \in \mathcal{Y})$
Output: An ensemble classifier H

1: Step 1: Learn first-level classifiers
2: **for** $t \leftarrow 1$ to T **do**
3: Learn a base classifier h_t based on $\mathcal{D}$
4: **end for**
5: Step 2: Construct new data sets from $\mathcal{D}$
6: **for** $i \leftarrow 1$ to m **do**
7: Construct a new data set that contains $\{\mathbf{x}'_i, y_i\}$, where $\mathbf{x}'_i = \{h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_T(\mathbf{x}_i)\}$
8: **end for**
9: Step 3: Learn a second-level classifier
10: Learn a new classifier h' based on the newly constructed data set
11: **return** $H(\mathbf{x}) = h'(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_T(\mathbf{x}))$

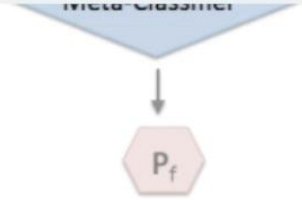
References

- [1] Tang, J., S. Alelyani, and H. Liu. "Data Classification: Algorithms and Applications." Data Mining and Knowledge Discovery Series, CRC Press (2015): pp. 498-500.
- [2] Wolpert, David H. "Stacked generalization." Neural networks 5.2 (1992): 241-259.

Contents - Google Docs | StackingClassifier - mlxtend | Chekuri Srikan...

Secure | https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

mlxtend | Home | User Guide | API | Installation | About | Search | Previous | Next | GitHub



The algorithm can be summarized as follows (source: [1]):

Algorithm 19.7 Stacking

Input: Training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{m}$ $(\mathbf{x}_i \in \mathbb{R}^n, y_i \in \mathcal{Y})$
Output: An ensemble classifier H

Step 1: Learn first-level classifiers
for $t \leftarrow 1$ to T **do**
    Learn a base classifier h_t based on $\mathcal{D}$
end for
Step 2: Construct new data sets from $\mathcal{D}$
for $i \leftarrow 1$ to m **do**
    Construct a new data set that contains $\{\mathbf{x}'_i, y_i\}$, where $\mathbf{x}'_i = \{h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_T(\mathbf{x}_i)\}$
end for
Step 3: Learn a second-level classifier
Learn a new classifier h' based on the newly constructed data set
return $H(\mathbf{x}) = h'(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_T(\mathbf{x}))$

References

- [1] Tang, J., S. Alelyani, and H. Liu. "Data Classification: Algorithms and Applications." Data Mining and

Contents - Google Docs | StackingClassifier - mlxtend | Chekuri Srikan...

Secure | https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

The algorithm can be summarized as follows (source: [1]):

Algorithm 19.7 Stacking

Input: Training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{m}$ ($\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \mathcal{Y}$)
Output: An ensemble classifier H

1: Step 1: Learn first-level classifiers
2: **for** $t \leftarrow 1$ to T **do**
3: Learn a base classifier h_t based on $\mathcal{D}$
4: **end for**
5: Step 2: Construct new data sets from $\mathcal{D}$
6: **for** $i \leftarrow 1$ to m **do**
7: Construct a new data set that contains $\{\mathbf{x}'_i, y_i\}$, where $\mathbf{x}'_i = \{h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_T(\mathbf{x}_i)\}$
8: **end for**
9: Step 3: Learn a second-level classifier
10: Learn a new classifier h' based on the newly constructed data set
11: **return** $H(\mathbf{x}) = h'(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_T(\mathbf{x}))$

References

- [1] Tang, J., S. Alelyani, and H. Liu. "Data Classification: Algorithms and Applications." Data Mining and Knowledge Discovery Series, CRC Press (2015): pp. 498-500

Contents - Google Docs | StackingClassifier - mlxtend | Installation - mlxtend | Chekuri Srikan...

Secure | https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/


```
iris = datasets.load_iris()
X, y = iris.data[:, 1:3], iris.target
```

```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from mlxtend.classifier import StackingClassifier
import numpy as np

clf1 = KNeighborsClassifier(n_neighbors=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
lr = LogisticRegression()
sclf = StackingClassifier(classifiers=[clf1, clf2, clf3],
                          meta_classifier=lr)

print('3-fold cross validation:\n')

for clf, label in zip([clf1, clf2, clf3, sclf],
                      ['KNN',
                       'Random Forest',
                       'Naive Bayes',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, X, y,
                                             cv=3, scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

$h_1(x) = clf1 \rightarrow KNN$

$h_2(x) = clf2 \rightarrow RF$

$h_3(x) = clf3 \rightarrow GNB$

$h'(x) = LR$

```
3-fold cross validation:
```

Contents - Google Docs | StackingClassifier - mlxtend | Installation - mlxtend | Chekuri Srikan...

Secure | https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

mlxtend Home User Guide API Installation About Search Previous Next GitHub

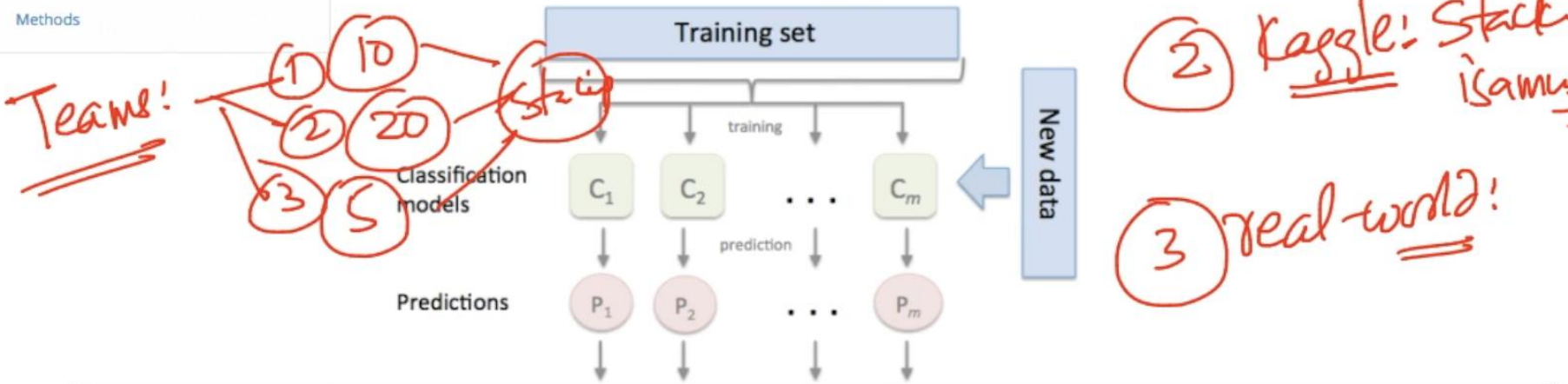
StackingClassifier

An ensemble-learning meta-classifier for stacking.

```
from mlxtend.classifier import StackingClassifier
```

Overview

Stacking is an ensemble learning technique to combine multiple classification models via a meta-classifier. The individual classification models are trained based on the complete training set; then, the meta-classifier is fitted based on the outputs -- meta-features -- of the individual classification models in the ensemble. The meta-classifier can either be trained on the predicted class labels or probabilities from the ensemble.



1 $h_1(x), h_2(x), h_3(x)$ more different the base models, the better it is

Teams!

2 Kaggle: Stacking is a must!

3 real-world!

Contents - Google Docs | StackingClassifier - mlxtend | Installation - mlxtend | Chekuri Srikan...

Secure | https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

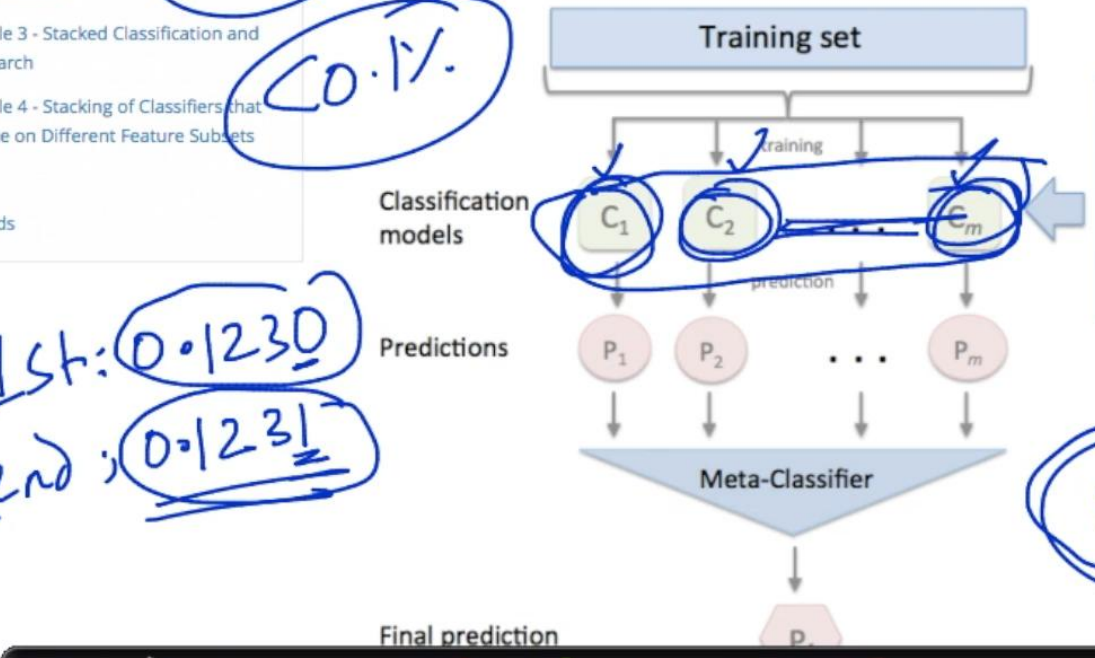
mlxtend | Home | User Guide | API | Installation | About | Search | Previous | Next | GitHub

```
from mlxtend.classifier import StackingClassifier
```

- StackingClassifier
- Overview
- References
- Example 1 - Simple Stacked Classification
- Example 2 - Using Probabilities as Meta-Features
- Example 3 - Stacked Classification and GridSearch
- Example 4 - Stacking of Classifiers that Operate on Different Feature Subsets
- API
- Methods

Overview

Stacking is an ensemble learning technique to combine multiple classification models via a meta-classifier. The individual classification models are trained based on the complete training set; then, the meta-classifier is fitted based on the outputs -- meta-features -- of the individual classification models in the ensemble. The meta-classifier can either be trained on the predicted class labels or probabilities from the ensemble.



Competitions → Kaggle

Stacking being used less often than boosting (GB) & bagging (RF)

1/230

CO·1%

log loss
→ 1st: 0.1230
→ 2nd: 0.1231

0·1%

Contents - Google Docs | StackingClassifier - mlxtend | Installation - mlxtend | Chekuri Srikan...

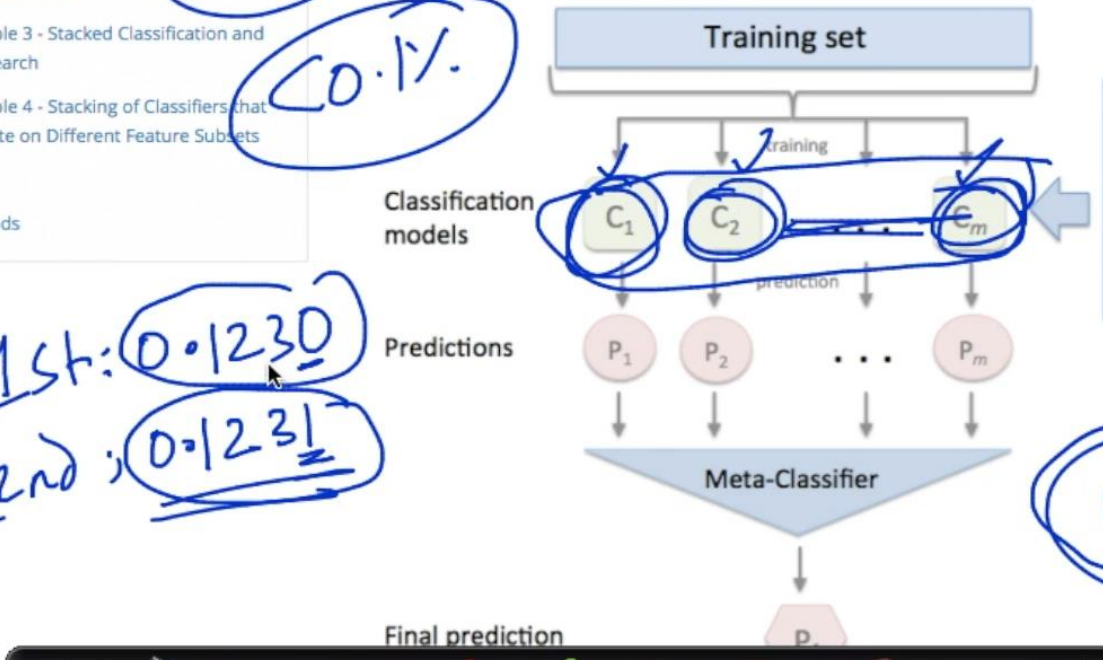
Secure | https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

mlxtend | Home | User Guide | API | Installation | About | Search | Previous | Next | GitHub

from mlxtend.classifier import StackingClassifier

Overview

Stacking is an ensemble learning technique to combine multiple classification models via a meta-classifier. The individual classification models are trained based on the complete training set; then, the meta-classifier is fitted based on the outputs -- meta-features -- of the individual classification models in the ensemble. The meta-classifier can either be trained on the predicted class labels or probabilities from the ensemble.



Competitions → Kaggle

Stacking being used less often than boosting (GB) & bagging (RF)

log loss
1st: 0.1230
2nd: 0.1231

0.1%

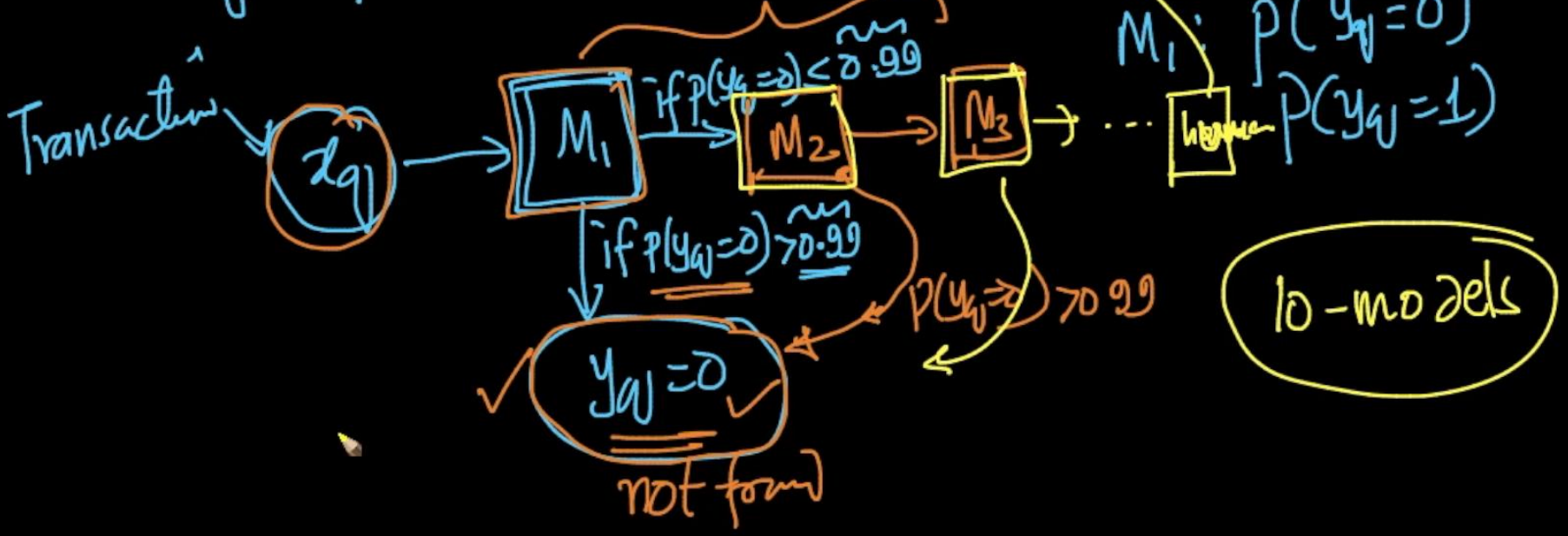
0.1%

"Cascading" models:

→ Bagging, boosting, Stacking

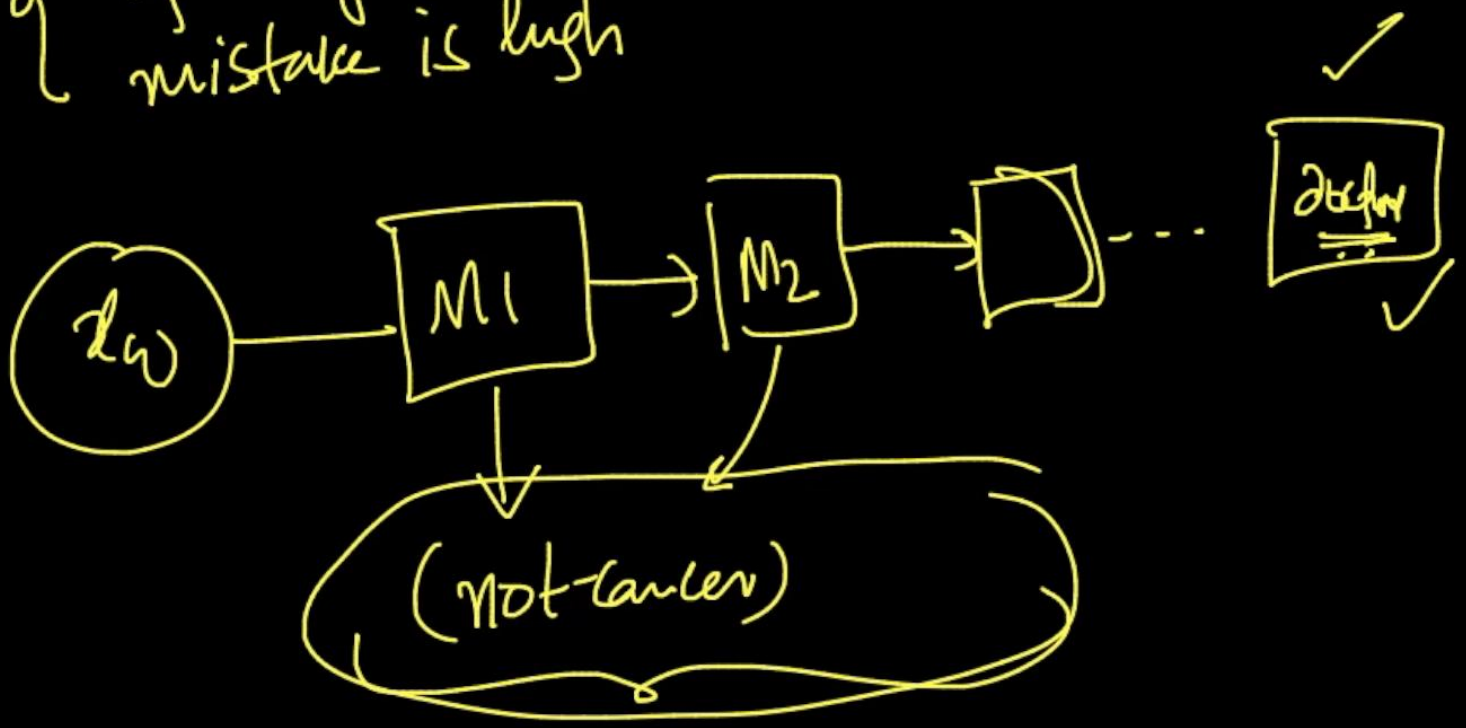
→ e.g:- predicts {credit card transaction} is fraudulent or not

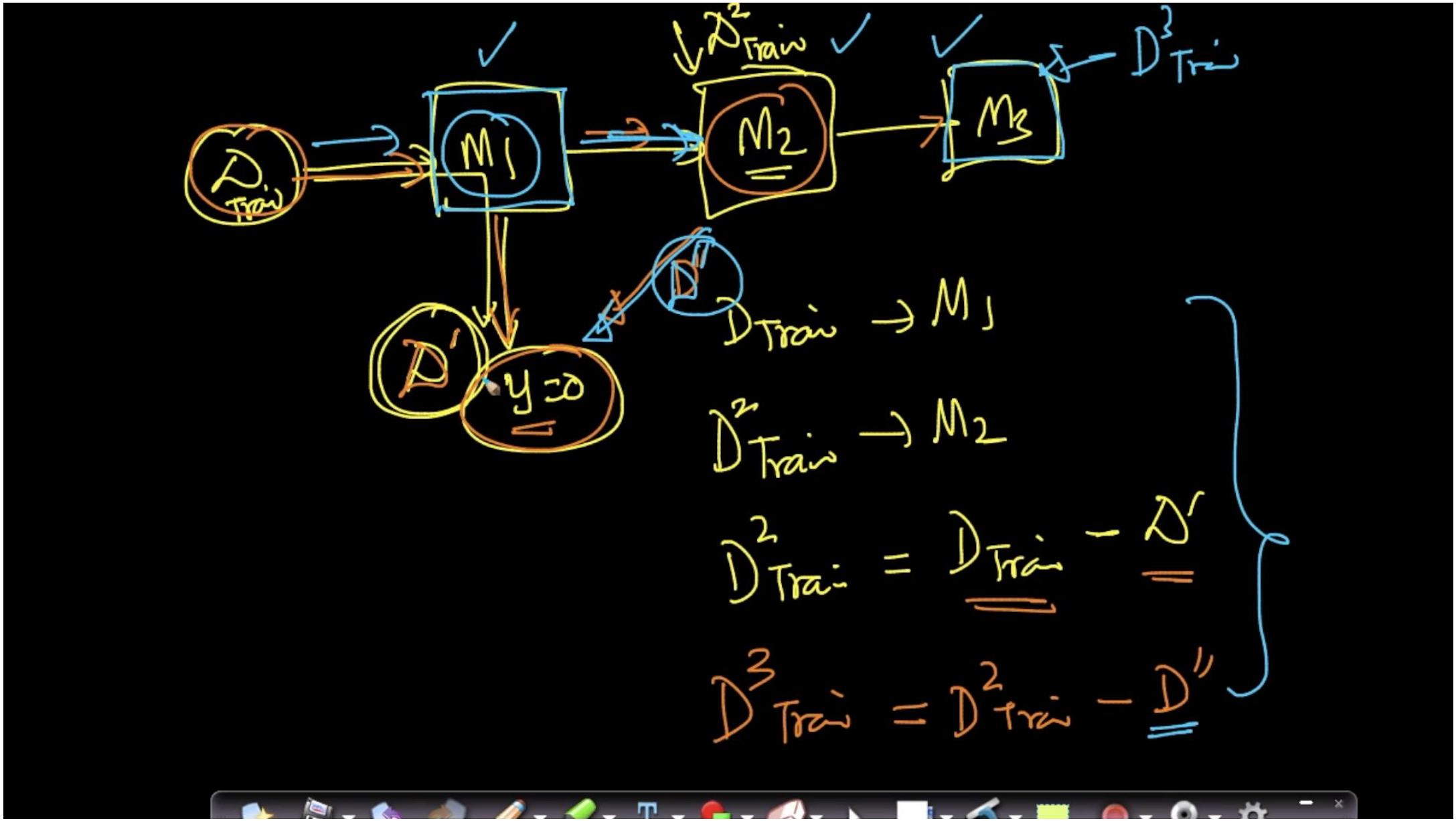
{0 → not fraud, 1 → fraud}

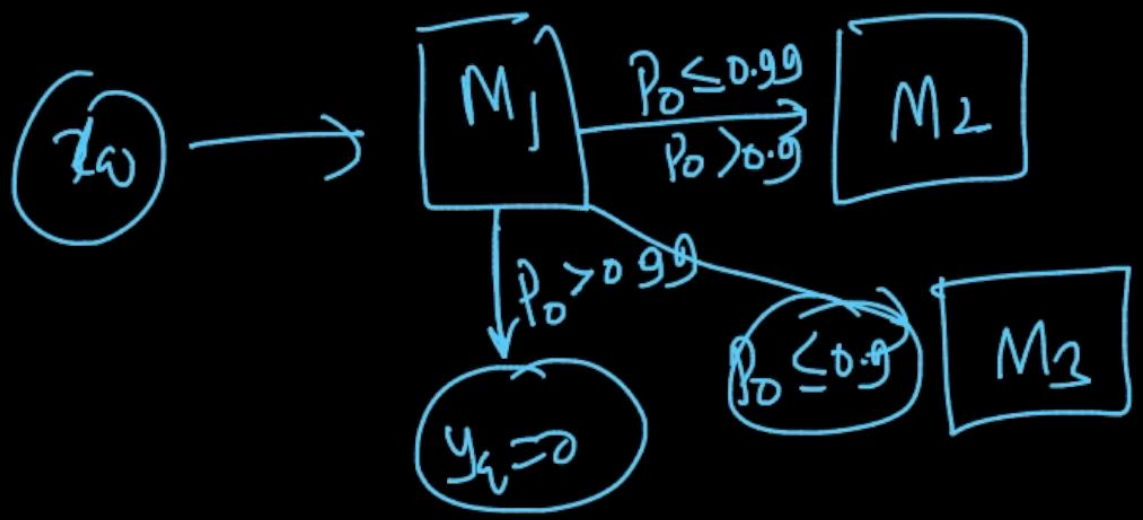


(Cascade-model:)

{ Typically used when the cost of making a mistake is high




D_{Train}
M_1
M_2
M_3
D^2_{Train}
D^3_{Train}
D'
$y=0$
$D_{Train} \rightarrow M_1$
$D^2_{Train} \rightarrow M_2$
$D^2_{Train} = D_{Train} - D'$
$D^3_{Train} = D^2_{Train} - D''$



$P_0 = P(y_4 = 0)$

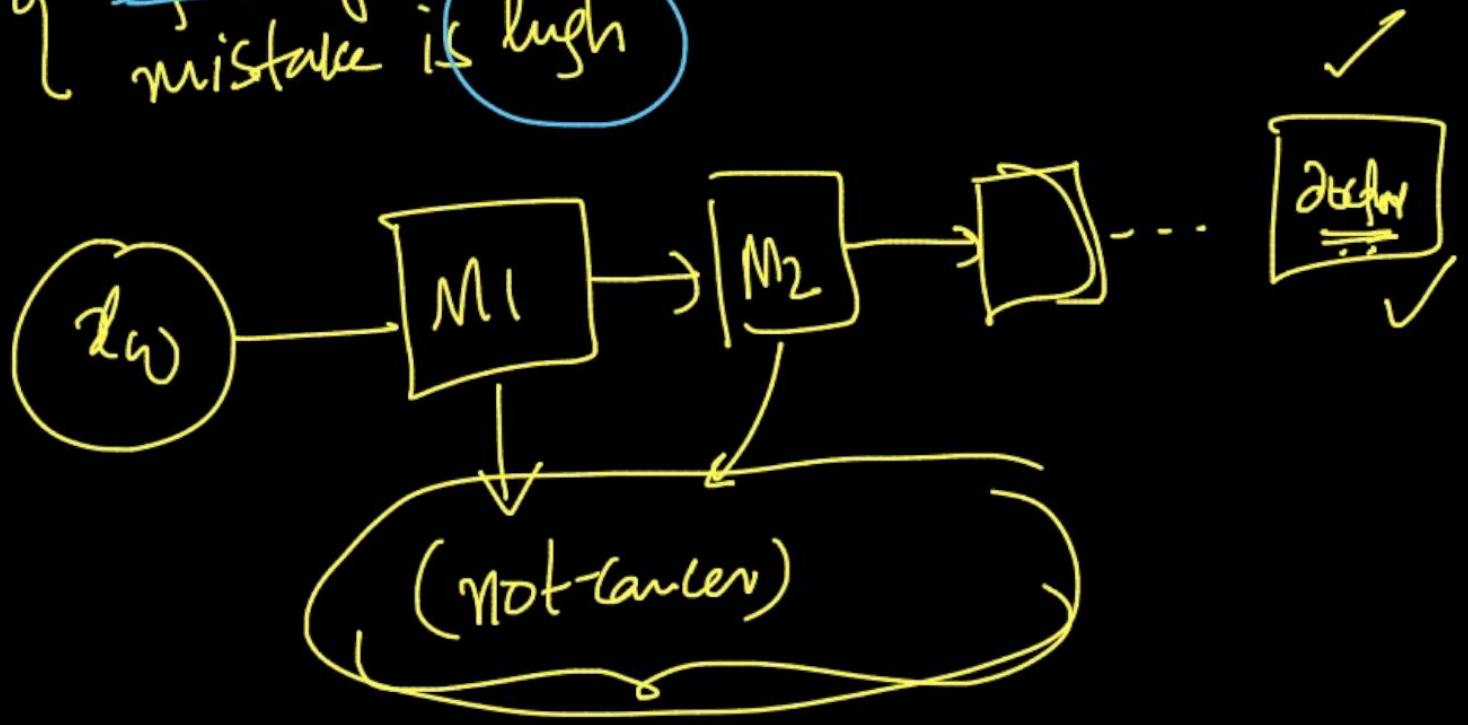
$P_1 = P(y_4 = 1)$

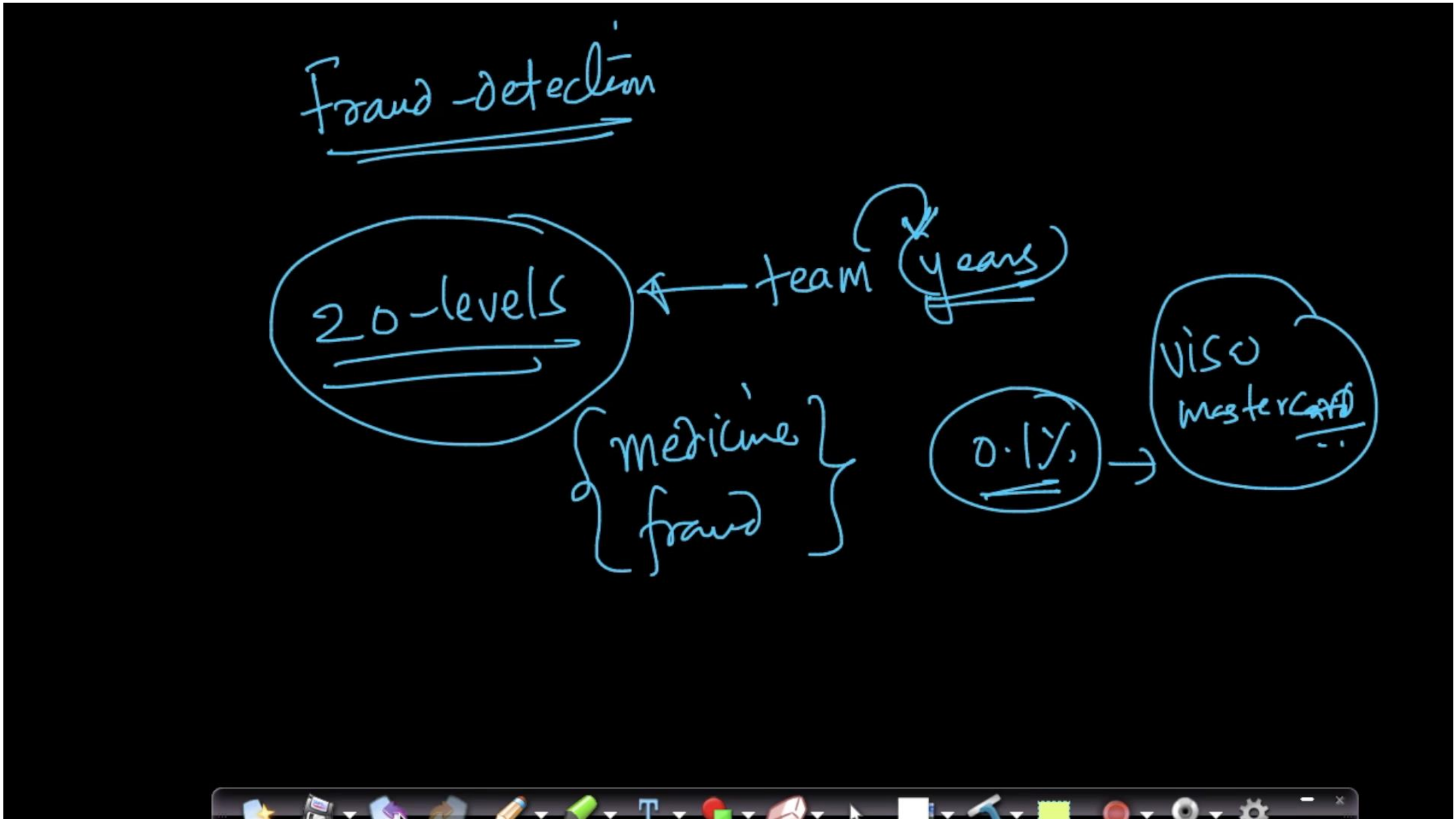
```
if P0 > 0.99
    y4 = 0
else
{   if P0 > 0.9
        use M2
    else
        use M3
}
```

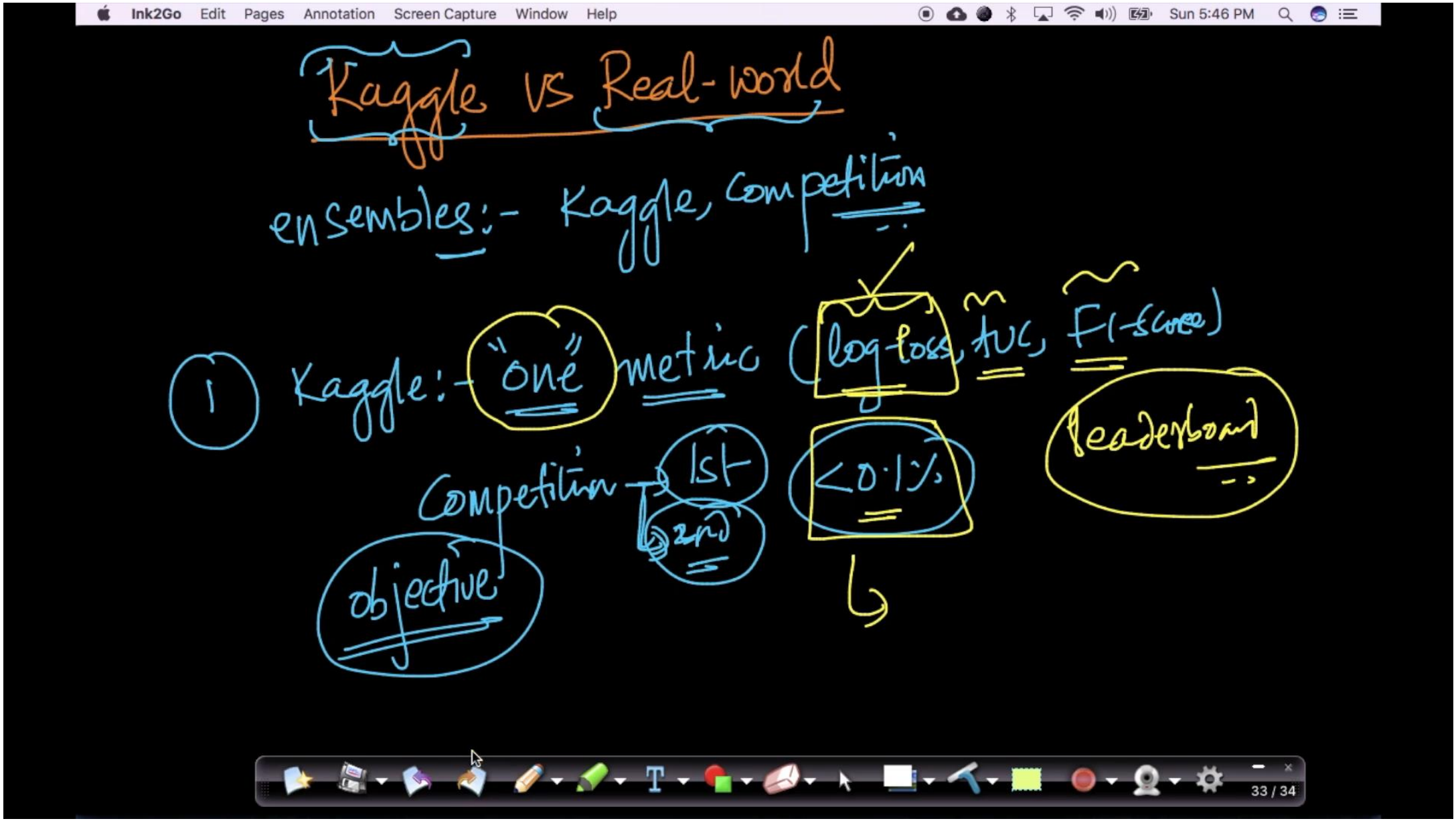
(Cascade-model:)

✓ { Typically used when the cost of making a mistake is high

✓



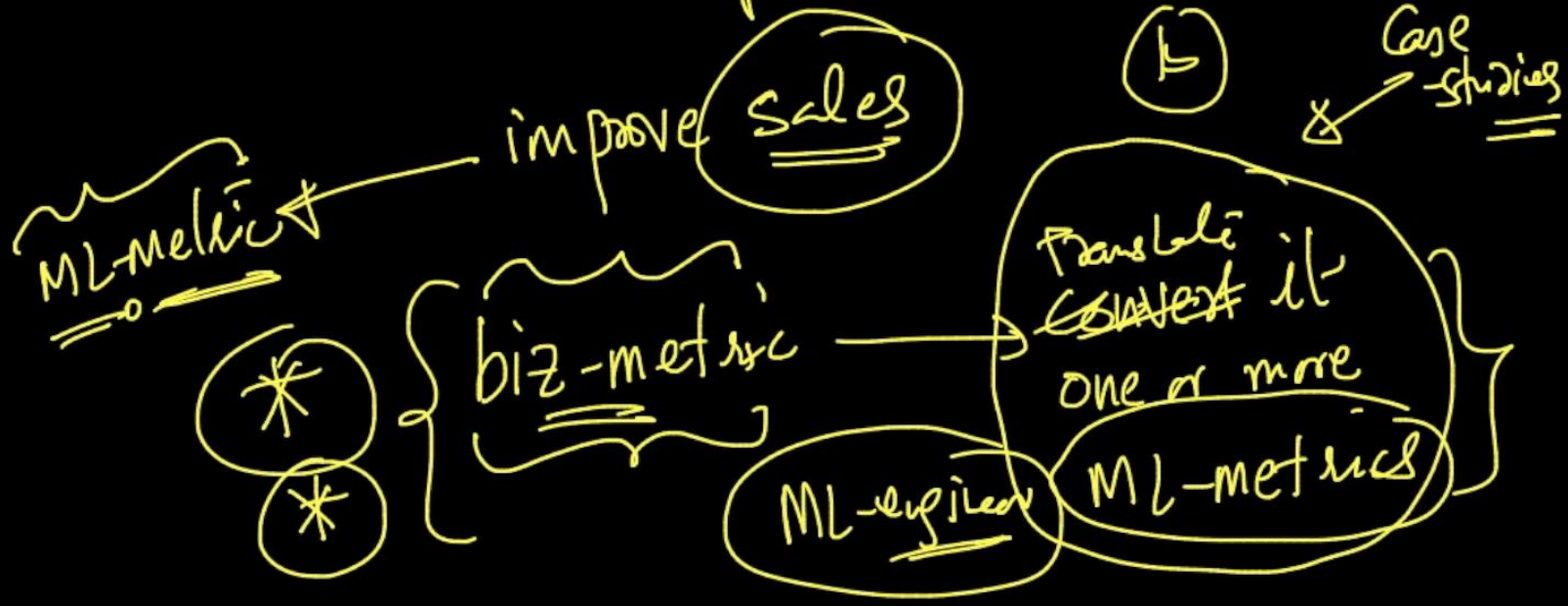

Fraud detection
20 levels
team
years
medicine
fraud
0.1%
visa
MasterCard

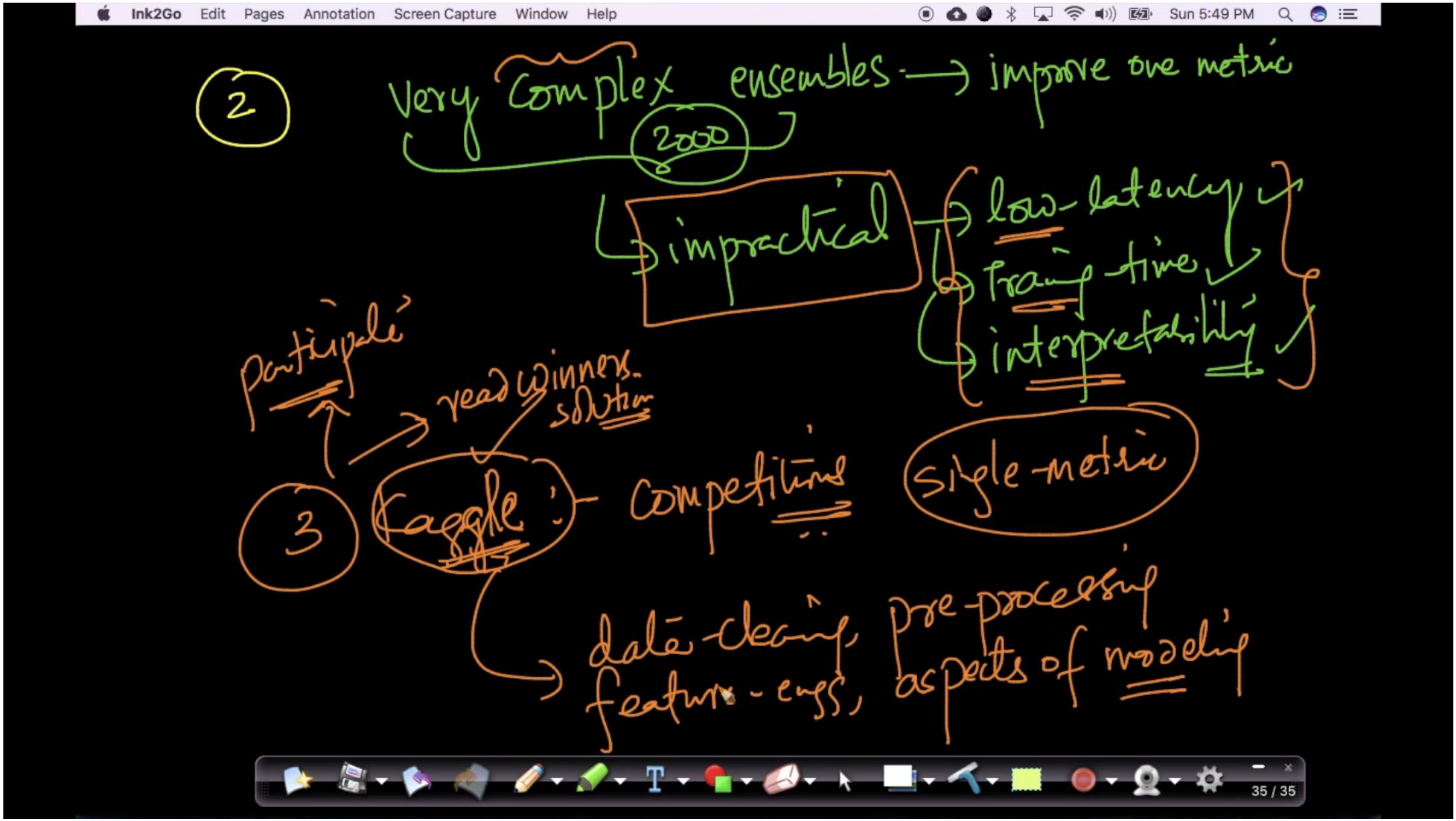

Ink2Go Edit Pages Annotation Screen Capture Window Help
Sun 5:46 PM
Kaggle vs Real-world
ensembles:- Kaggle, Competition
1
Kaggle:- "one" metric
log-loss, AUC, F(1-score)
leaderboard
Competition
1st
2nd
<0.1%
objective
33 / 34

real-world :- multiple metrics

(a)

→ primary metric
↳ secondary metric




Ink2Go Edit Pages Annotation Screen Capture Window Help
Sun 5:49 PM
2
Very Complex ensembles → improve one metric
2000
impractical
low-latency
train-time
interpretability
participate
read winners solutions
3
Kaggle
competitions
single-metric
data-cleaning, pre-processing
feature-engg, aspects of modeling
35 / 35

Loss-minimization : Hinge-loss

log-reg :- logistic loss + reg

lr-reg :- lr. loss + reg

sum :- hinge loss + reg

$= y_i (\vec{w} x_i + b) = z_i$

$z_i > 0$:- x_i is correctly classified

$z_i < 0$:- x_i is incorrectly classified