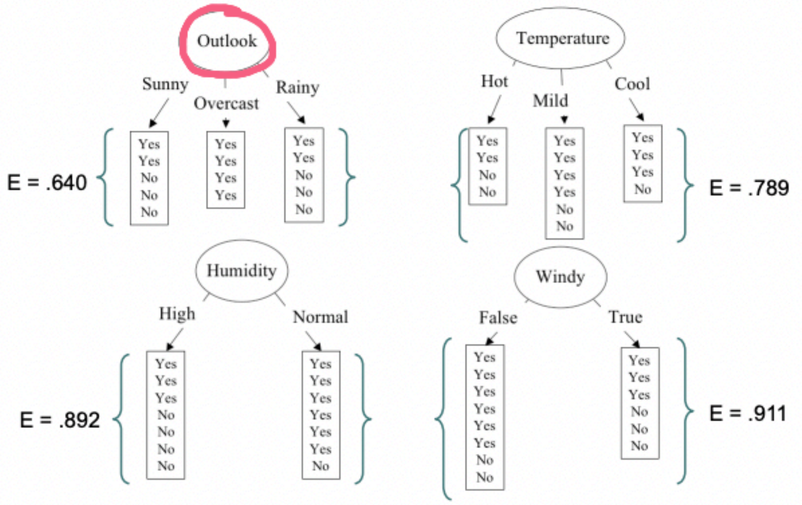
Topics:

- [Decision Trees] (Contd..) → thoroughly...
- Questions tab
- Chat for conversations
- Bagging → bootstrapped - aggregation
- Random forest → FAANG
- Code

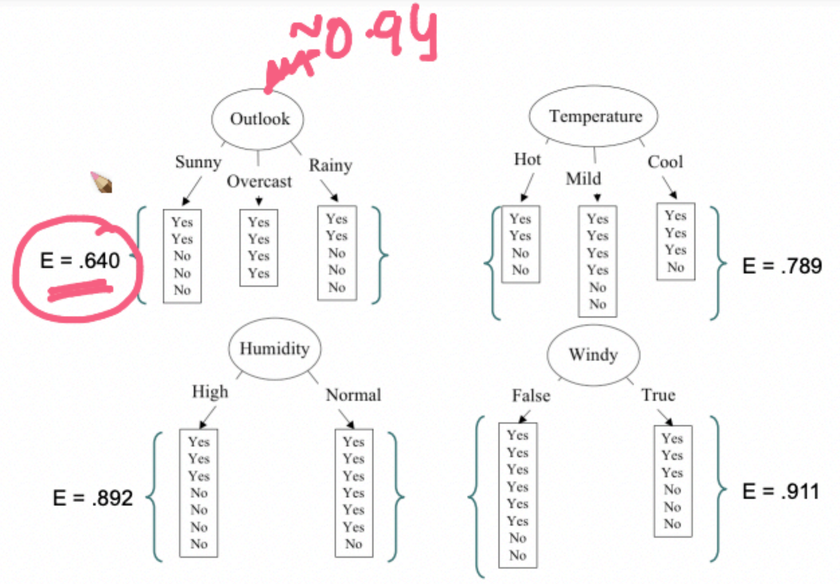
Y

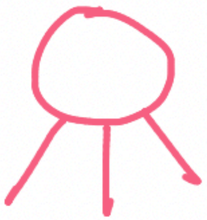
Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

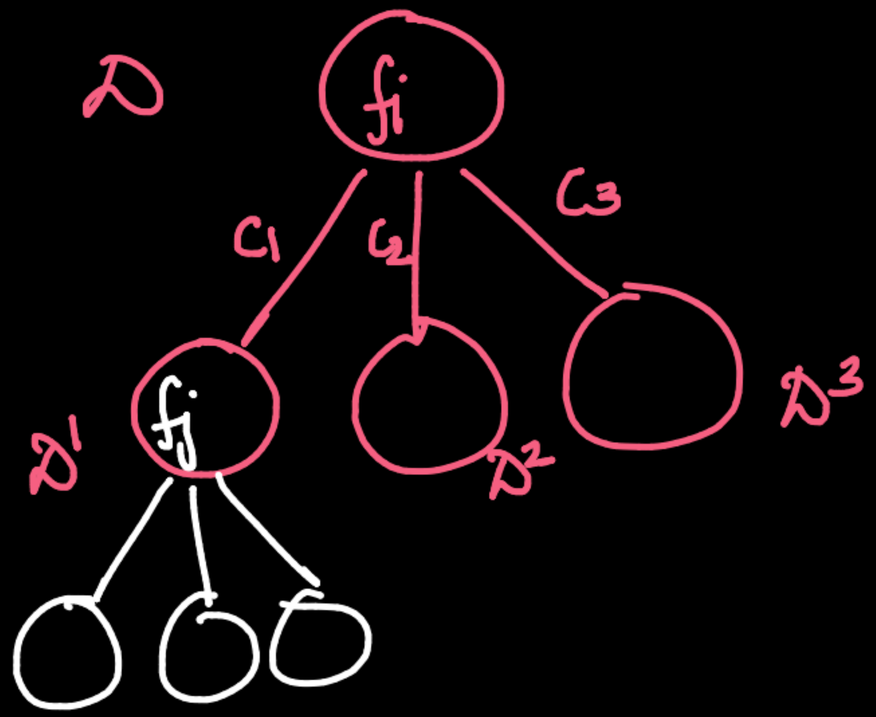


IG(Y, outlook) = weighted entropy @ child nodes − Entropy @ parent node

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No







- pure-nodes

only $y_i = 1$
or
$y_i = 0$

* @ every node of the DT; we have to try every feature & Split to pick the best decision to split on

IG

Recursive Partitioning

D1

Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes

Outlook

D2

Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes

D3

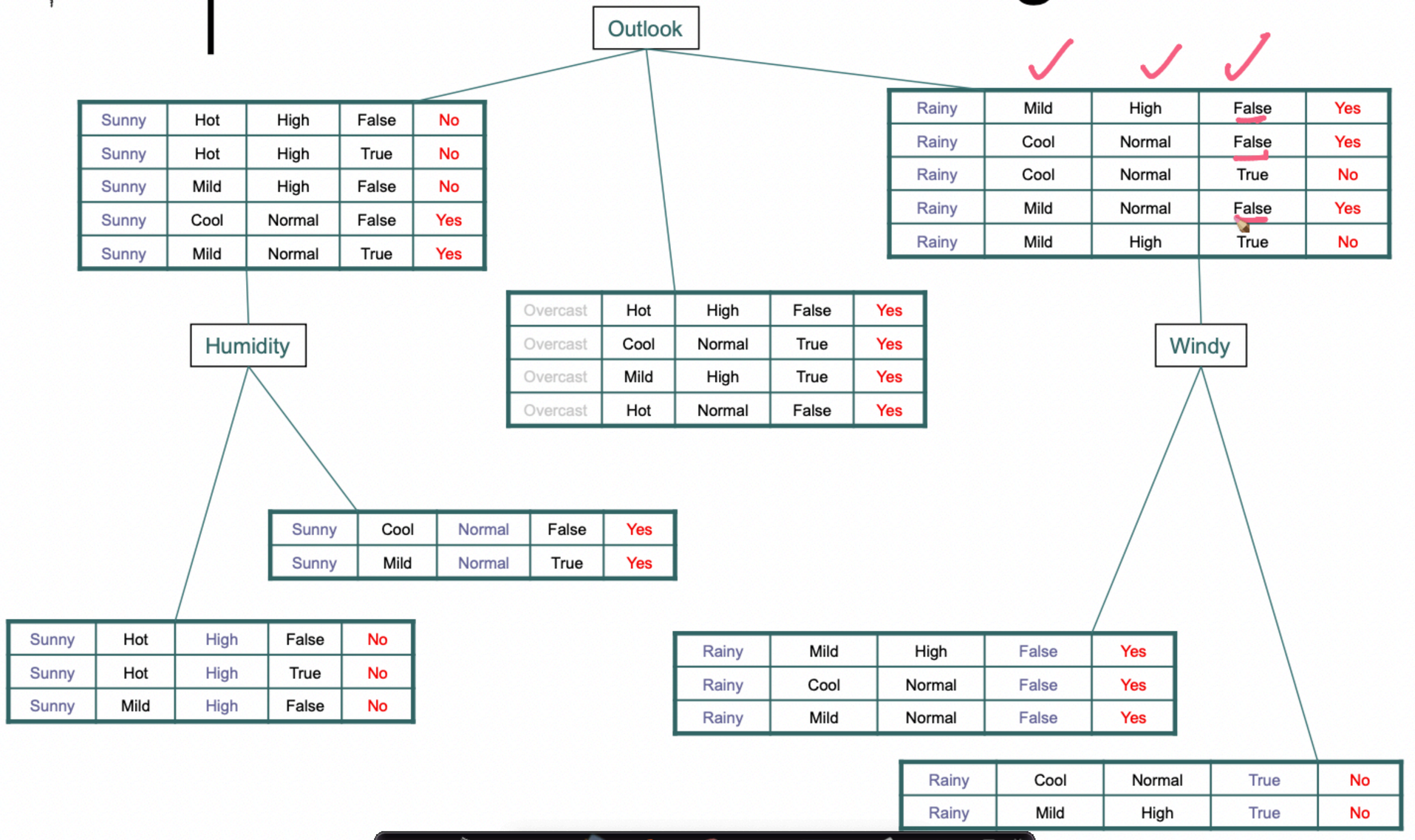
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Rainy	Mild	Normal	False	Yes
Rainy	Mild	High	True	No

Humidity

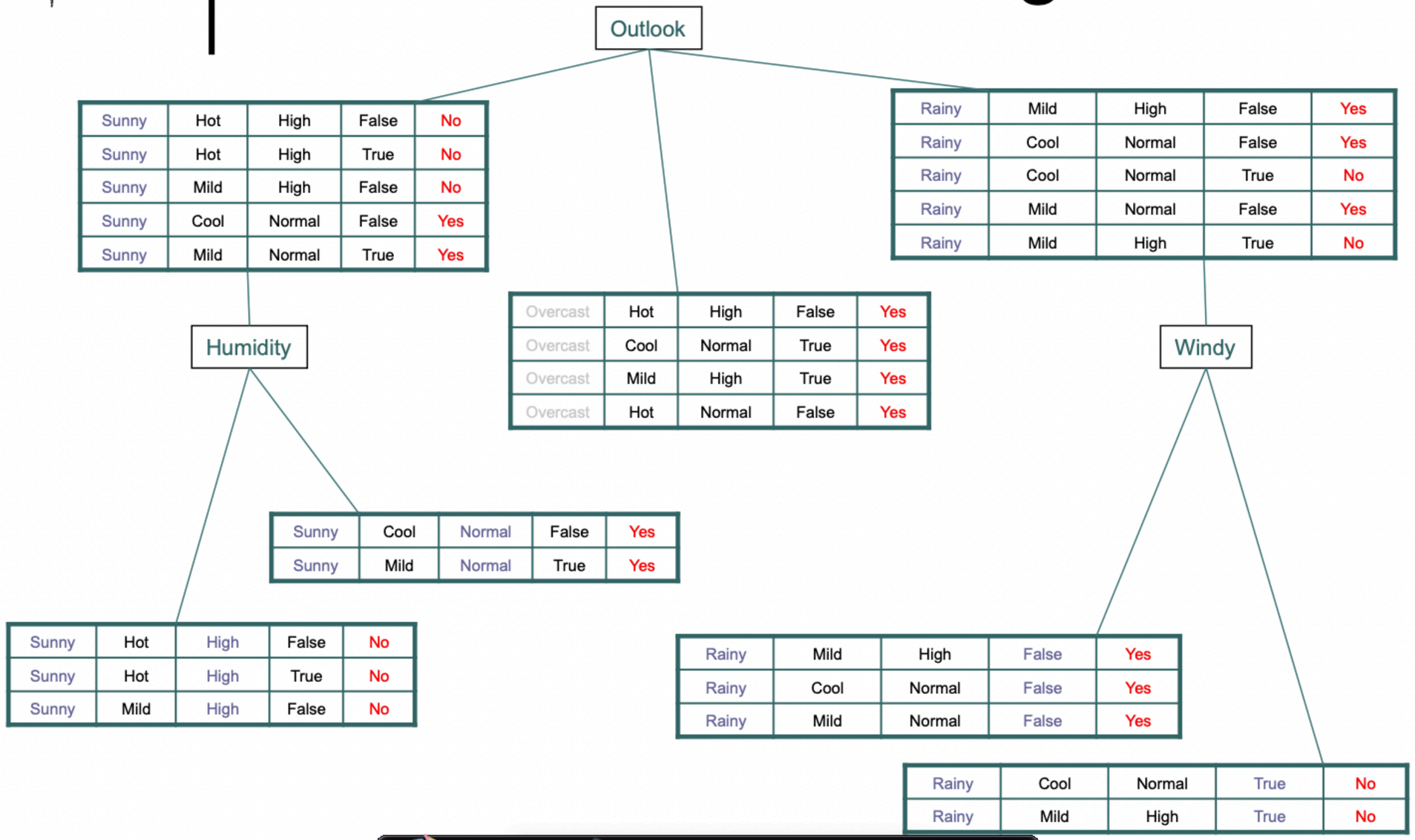
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes

Sunny	Hot	High	False	N
Sunny	Hot	High	True	N

Recursive Partitioning



Recursive Partitioning



Q lots of features; d is high : d>100

time to build the trees ↑

RF
GBDT

disb. computing
or multi-processing

d=1000
avoid DT

sklearn.ensemble.RandomForestClassifier

class sklearn.ensemble.RandomForestClassifier(*n_estimators=100*, ***, *criterion='gini'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features='auto'*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *bootstrap=True*, *oob_score=False*, *n_jobs=None*, *random_state=None*, *verbose=0*, *warm_start=False*, *class_weight=None*, *ccp_alpha=0.0*, *max_samples=None*) [source]

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree.

Read more in the User Guide.

Parameters:

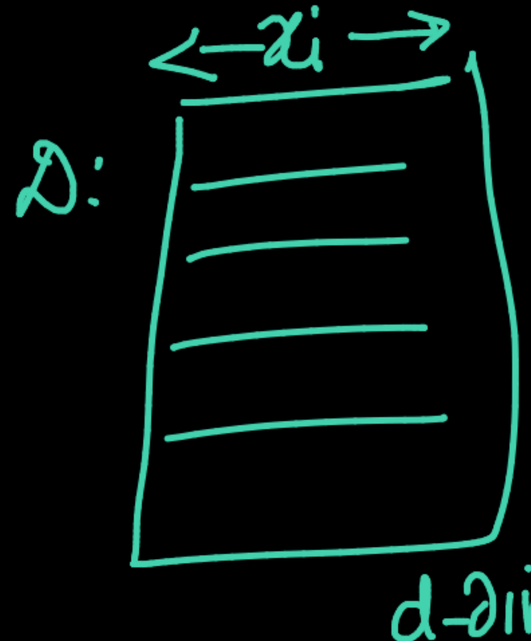
n_estimators : *int, default=100*
The number of trees in the forest.

Changed in version 0.22: The default value of `n_estimators` changed from 10 to 100 in 0.22.

criterion : *{"gini", "entropy"}, default="gini"*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Note: this parameter is tree-specific.

Q

PCA : dimensionality reduction →



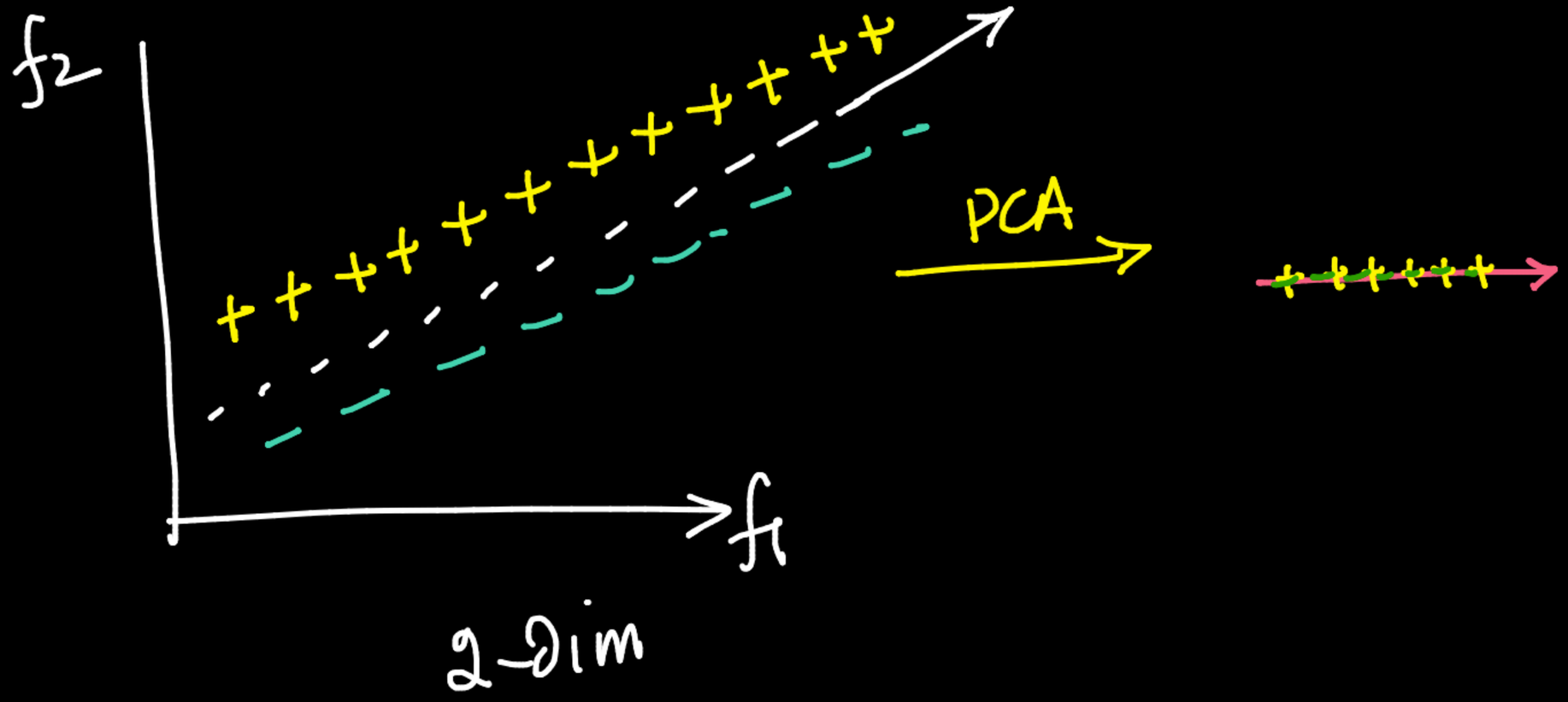
PCA

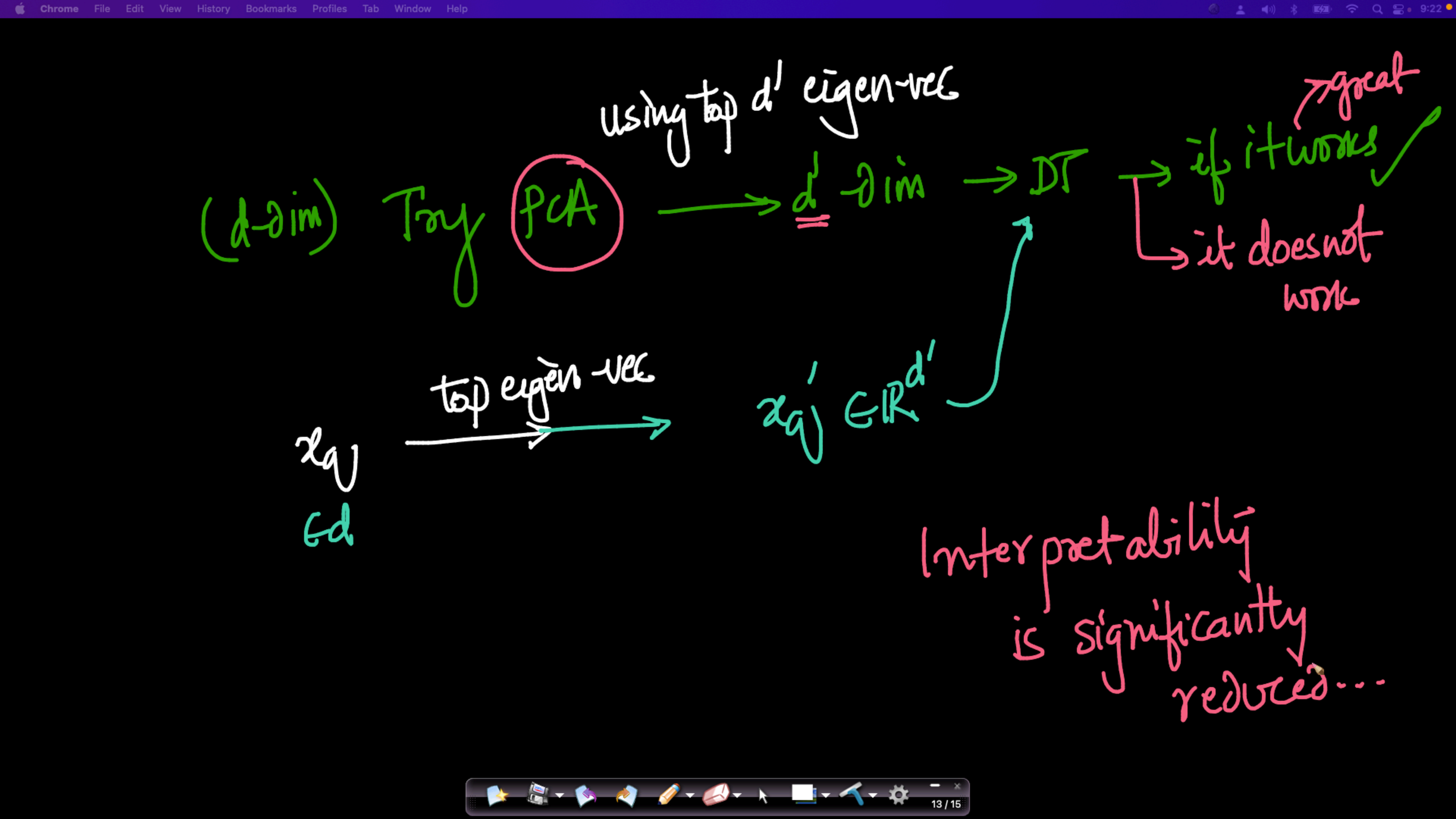
do NOT look @ y_i's

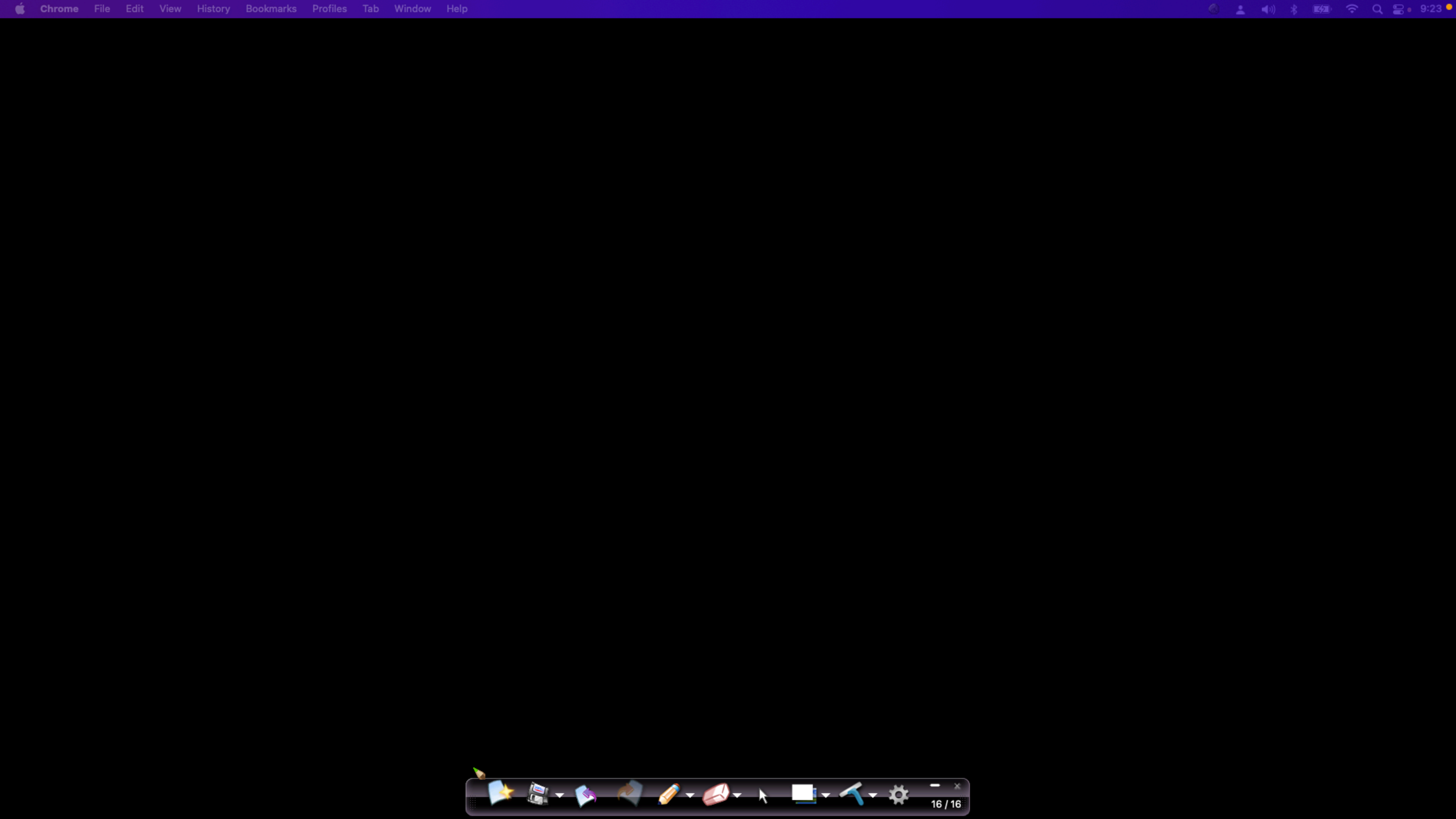
d-dim < d

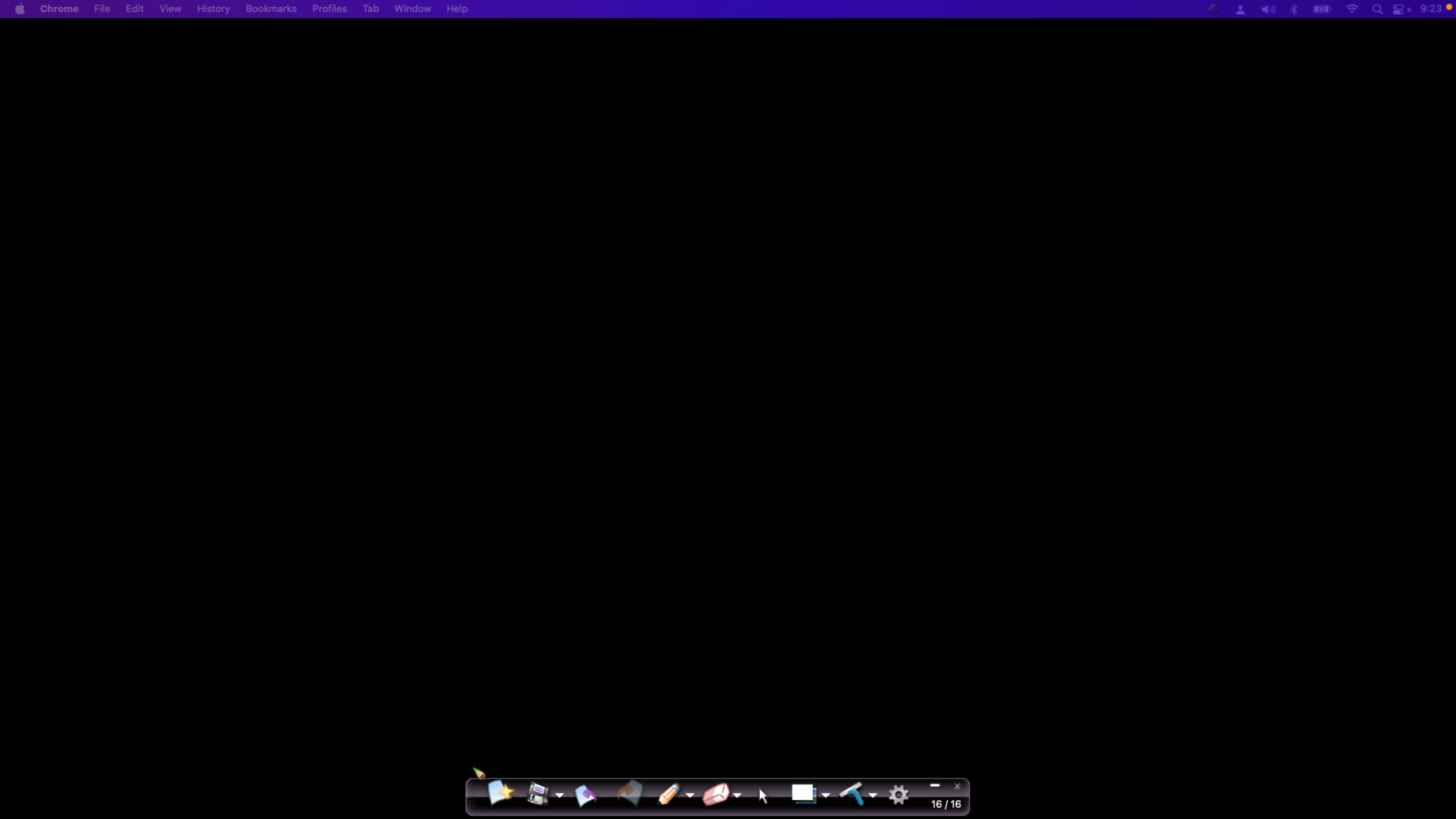
there exist dim red. tech that consider y_i's also

extreme case




Chrome File Edit View History Bookmarks Profiles Tab Window Help
9:22
(d-dim) Try PCA
using top d' eigen-vec
d'-dim
DT
if it works
great
it does not work
x_q $\in d$
top eigen-vec
$x_q' \in \mathbb{R}^{d'}$
Interpretability is significantly reduced...
13 / 15


Chrome File Edit View History Bookmarks Profiles Tab Window Help
9:23
16 / 16


Chrome File Edit View History Bookmarks Profiles Tab Window Help
9:23
16 / 16

I_G: Info gain

"Gini" Impurity

↓

Similar to Entropy

$$GI(Y) = 1 - \sum_{i=1}^{K} (p(y_i))^2$$

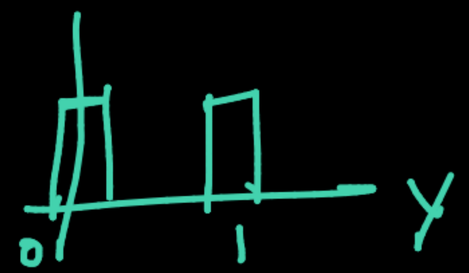
2-class:-

$$GI(Y) = 1 - \left[(p(y_i=1))^2 + (p(y_i=0))^2 \right]$$

Case 1:

$P(y+) = 0.5$

$P(y-) = 0.5$



$\log_2$

$H(Y) = 1$ high

$GI(Y) = 1 - (0.25 + 0.25) = 0.5$ high

Case 2:

$P(Y_+) = 1$

$P(Y_-) = 0$

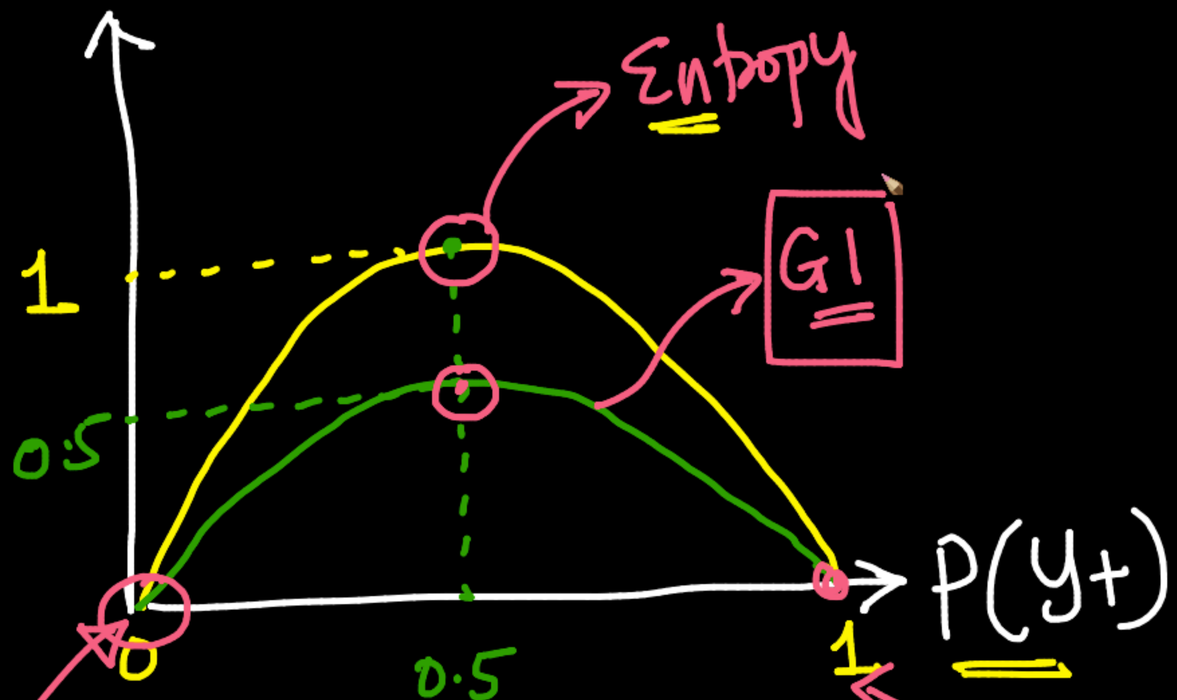
Small

$H(Y) = 0$: Min Entropy & max purity

$GI(Y) = 1 - (1+0) = 0$

Small

2-class



$-\lg_2$ in $H(Y)$

$P(y-) = 1 - P(y+)$

$P(y+) = 0$
$P(y-) = 1$

$P(y+) = 1$
$P(y-) = 0$

$P(y+) = 1/2 = P(y-)$

Code

$H(Y)$ vs $Gi(Y)$

K = #classes

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

$$1-\sum_{i=1}^{K} (p(y_i))^2$$

#classes ✓

sklearn

(*) log is computationally more expensive than squaring

sklearn.tree.DecisionTreeClassifier

class sklearn.tree.DecisionTreeClassifier(*, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy"}, default="gini"*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

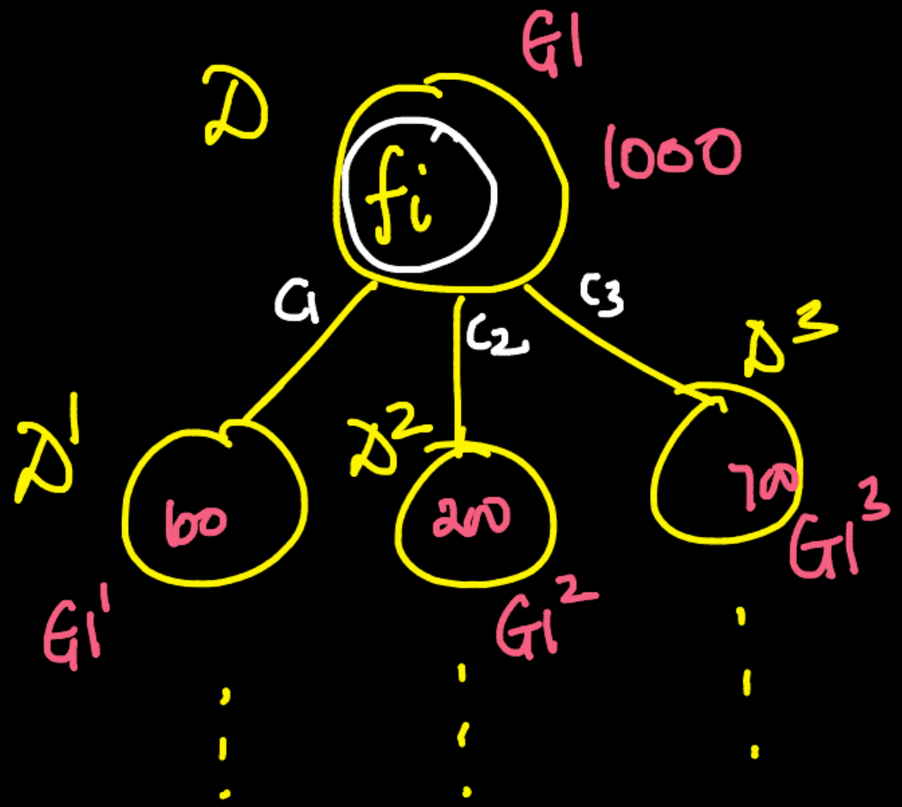
splitter : *{"best", "random"}, default="best"*
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int, default=None*
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int or float, default=2*
The minimum number of samples required to split an internal node:

multi-class - classfn:

1, 2, 3



Entropy or GI

& ΔHor ΔGI

→ Split on features

Weighted G.I of the child nodes

→ Entropy ✓

— G.I of the parent node

↳ Entropy ✓

sklearn.tree.DecisionTreeClassifier

class sklearn.tree.DecisionTreeClassifier(*, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy"}, default="gini"*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *{"best", "random"}, default="best"*
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

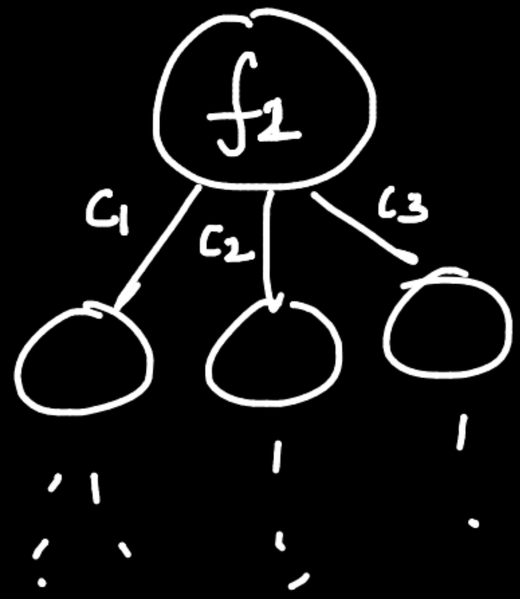
max_depth : *int, default=None*
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int or float, default=2*
The minimum number of samples required to split an internal node:

Numerical features

f1: Categorical feature: c_1, c_2, c_3

↓ Splitting criterion

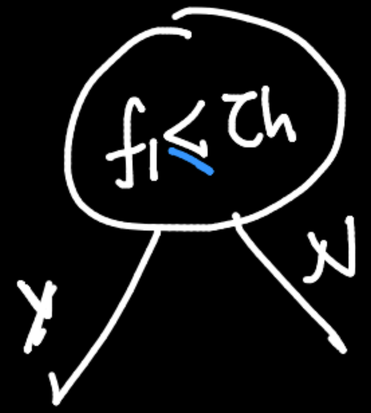


IG^2

numeric

f_2	f_1	y_i
C_1	2.2	1
	2.6	1
C_2	3.5	0
C_1	3.8	0
C_2	4.6	1
C_3	5.3	0
C_1		

f_i : numerical → n-ways



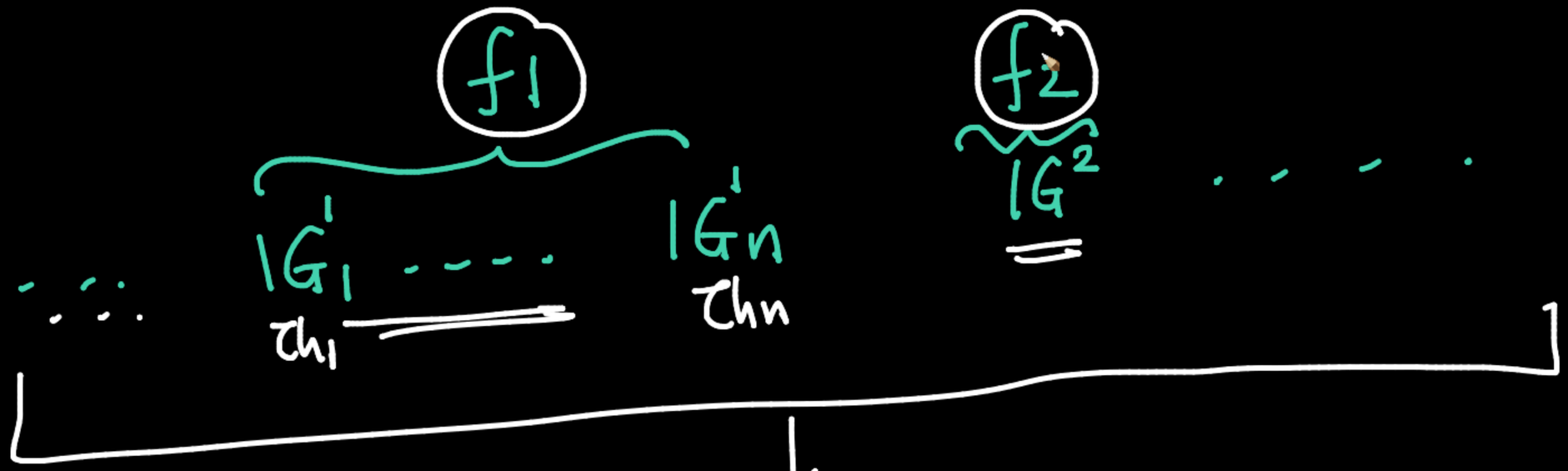
(1) $th_1 = 2.2$ → IG_1

(2) $th_2 = 2.6$ → IG_2

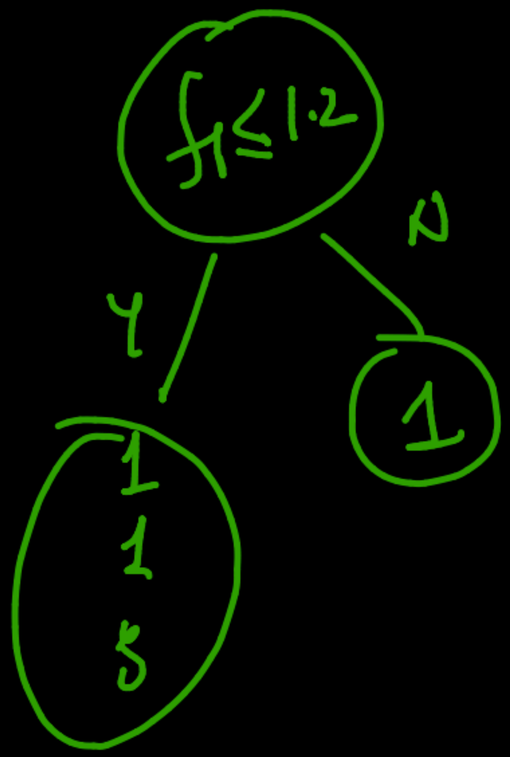
⋮

(n) $th_n = 5.3$

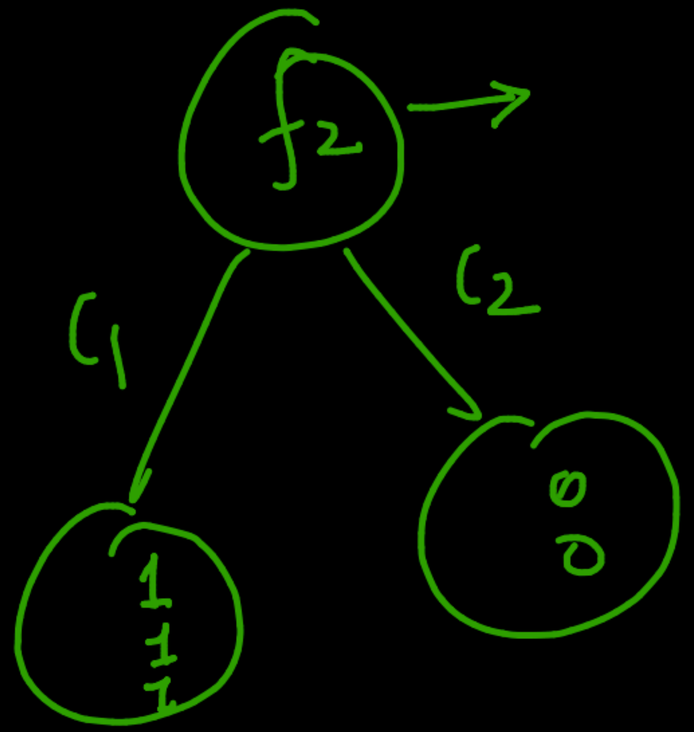
IG_n



Choose whichever split has max IG

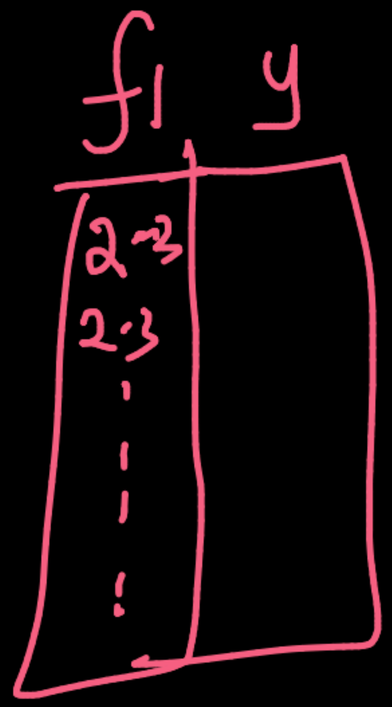


f_1	f_2	y_i
1.2	c_1	1
1.6	c_1	1
0.8	c_1	1
0.3	c_2	0
1.4	c_3	0



$IG^1_{10} = IG^2 = \text{minimal}$

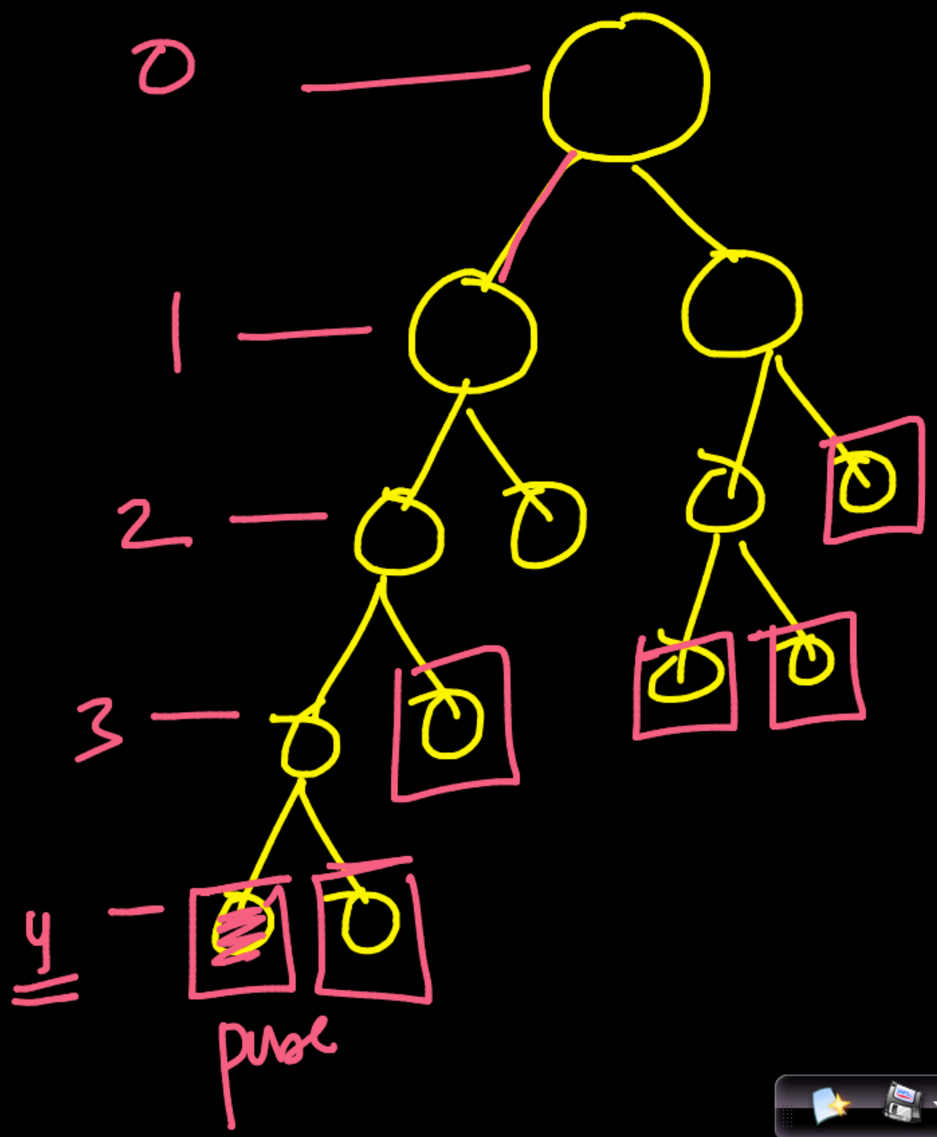
Pick any of these equal IG splits. ✓



more computationally efficient

rules bin it carefully

simplest $Q_1, Q_2, Q_3 Q_4$

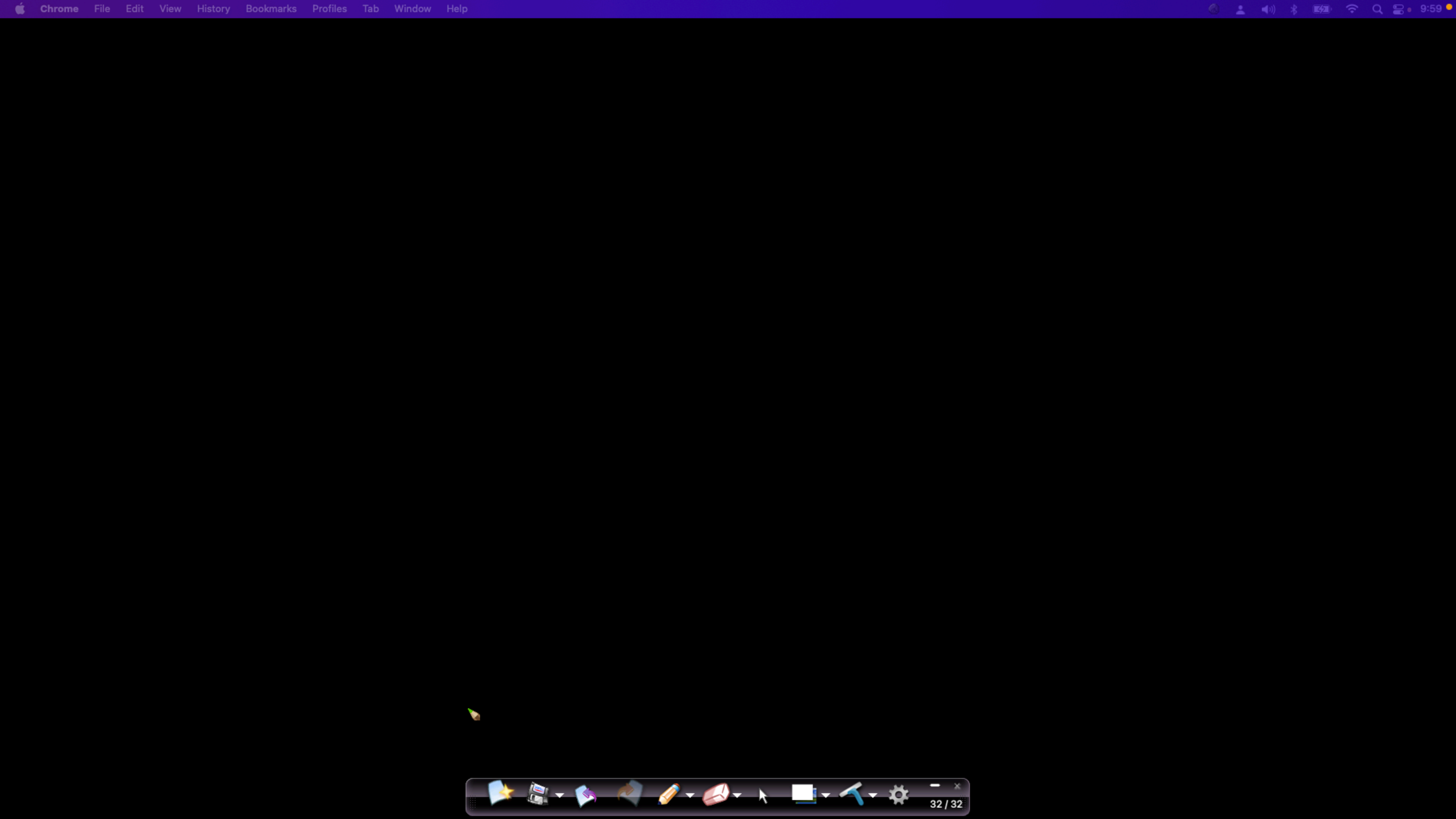


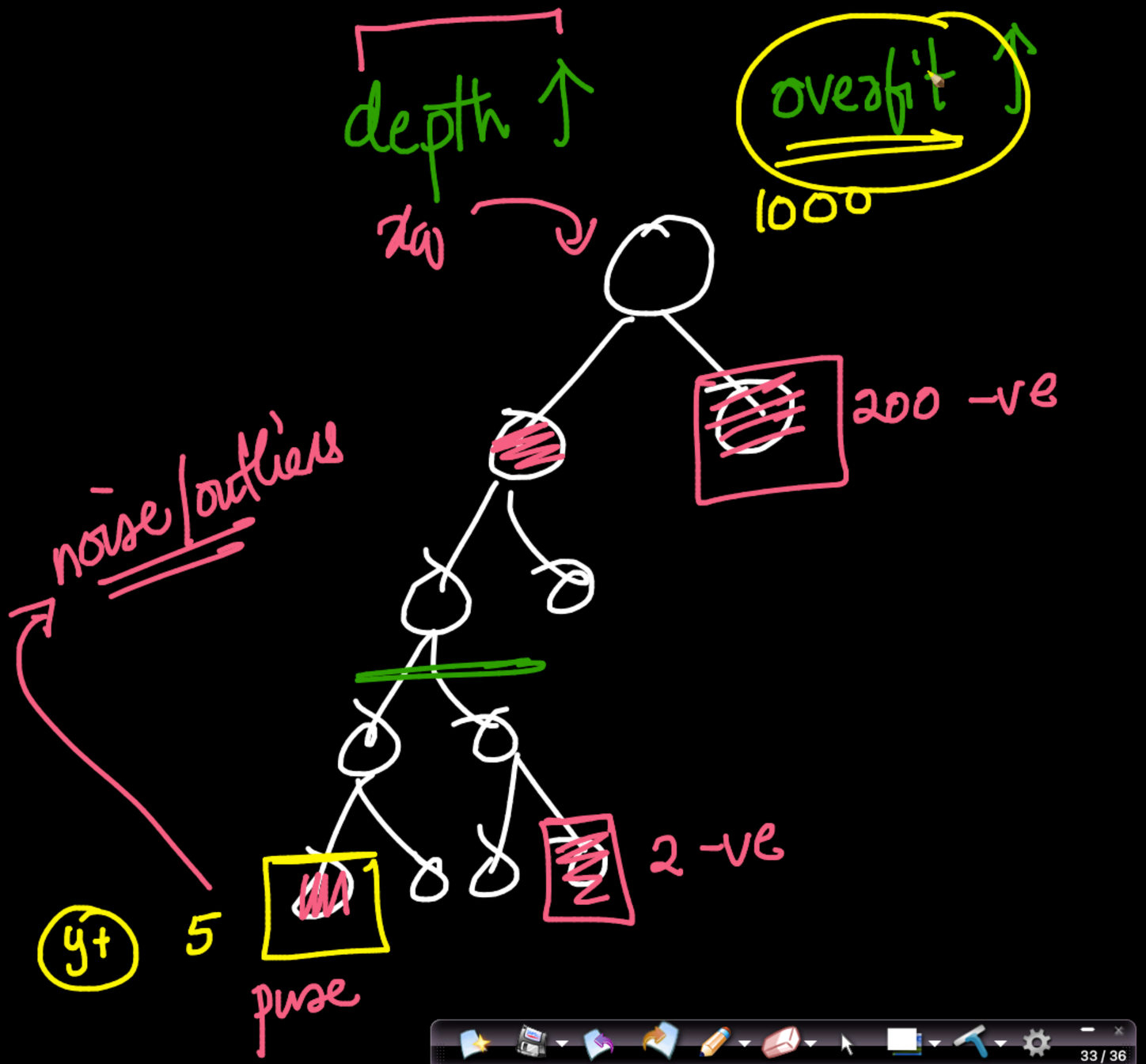
Overfit vs underfit

↓

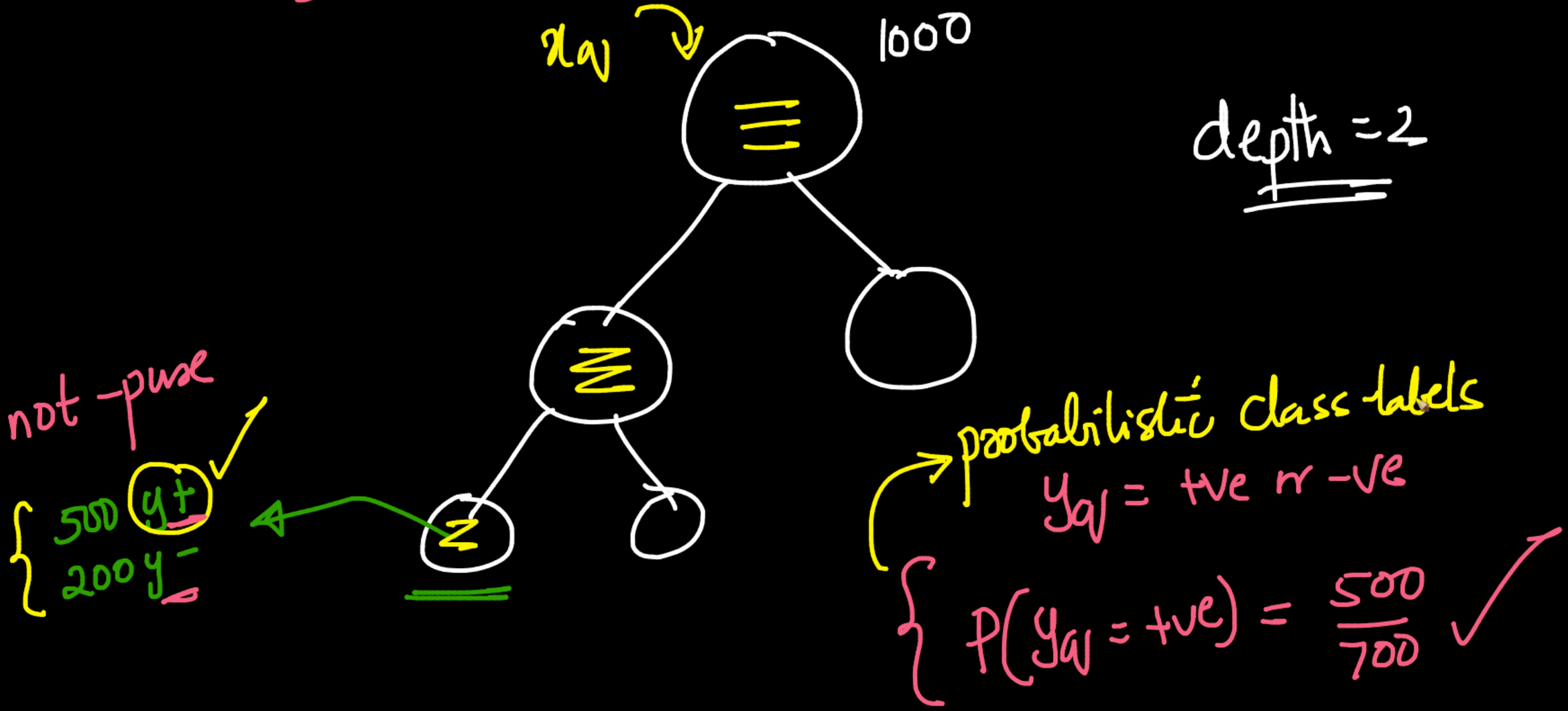
many splits

depth of a tree = 4

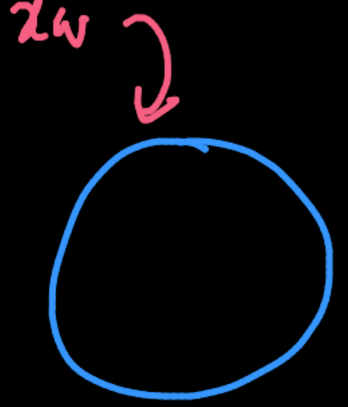

Chrome File Edit View History Bookmarks Profiles Tab Window Help
9:59
32 / 32


depth ↑
overfit ↑
1000
200 -ve
noise/outliers
2 -ve
5
pure
33 / 36

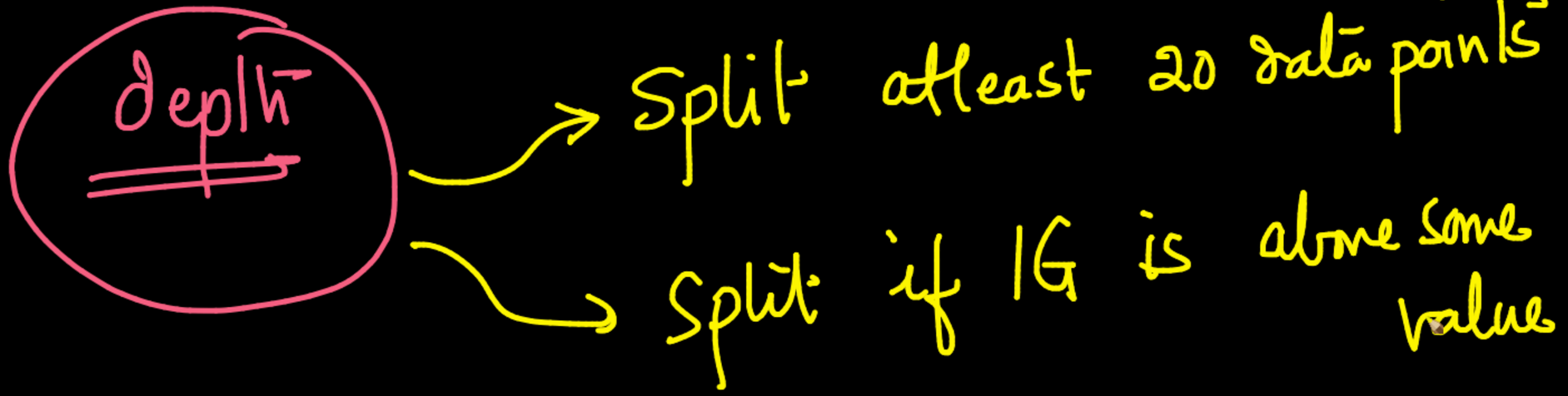
Shallow - Trees :-



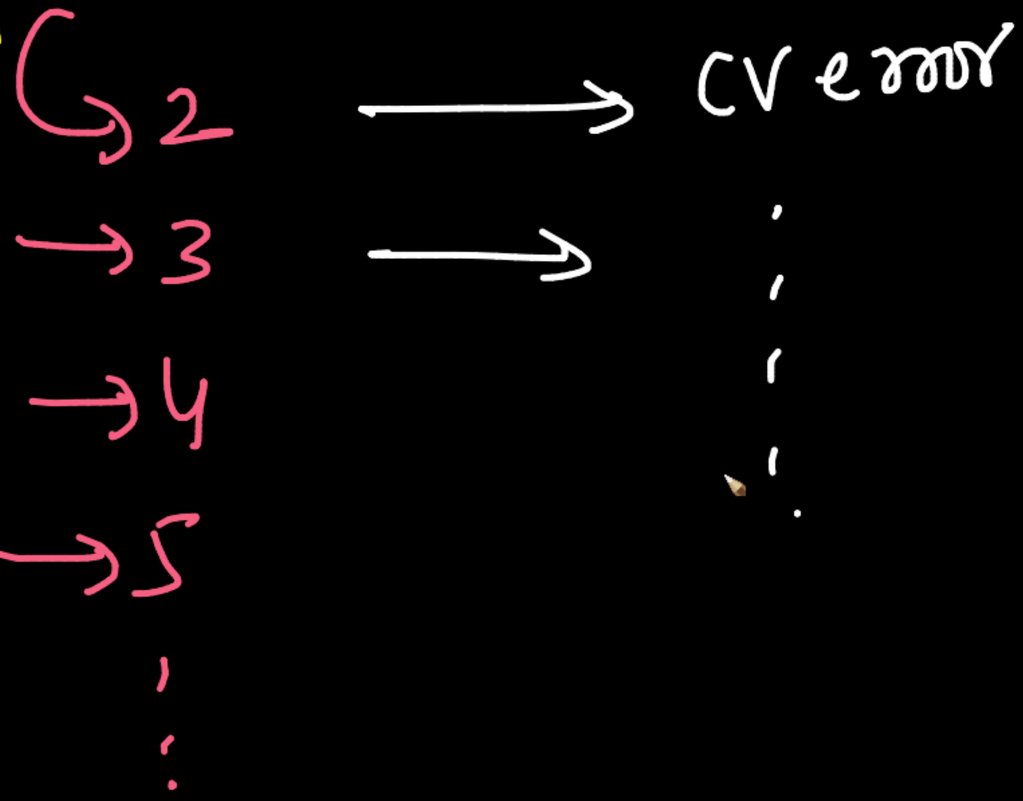
depth = 0

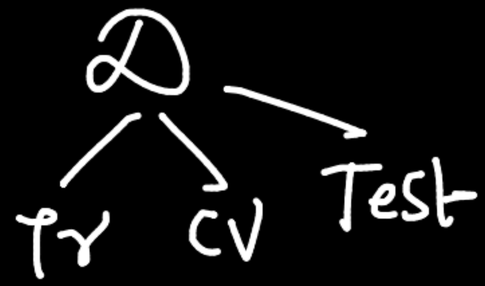


D = $y_+ : y_-$: 55% : 45%


depth
Split atleast 20 data points
Split if IG is above some value

Depth : hyperparam





depth = 2 pure nodes → Great ✓

class sklearn.tree.DecisionTreeClassifier(*, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy"}, default="gini"*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *{"best", "random"}, default="best"*
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

Extremely Random forest

max_depth : *int, default=None*
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int or float, default=2*
The minimum number of samples required to split an internal node:

`it` as the minimum number.
fraction and `ceil(min_samples_split *`

class sklearn.tree.DecisionTreeClassifier(*, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy"}, default="gini"*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *{"best", "random"}, default="best"*
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int, default=None*
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

class sklearn.tree.DecisionTreeClassifier(*, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy"}, default="gini"*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *{"best", "random"}, default="best"*
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int, default=None*
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

class sklearn.tree.DecisionTreeClassifier(*, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy"}, default="gini"*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *{"best", "random"}, default="best"*
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int, default=None*
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

class sklearn.tree.DecisionTreeClassifier(*, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*) [source]

A decision tree classifier.

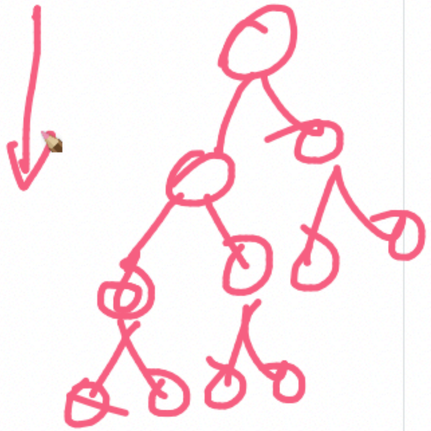
Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy"}, default="gini"*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *{"best", "random"}, default="best"*
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int, default=None*
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.



class sklearn.tree.DecisionTreeClassifier(*, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy"}, default="gini"*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *{"best", "random"}, default="best"*
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int, default=None*
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

scikit learn

Prev Up Next

scikit-learn 1.0.2
Other versions

Please cite us if you use the software.

sklearn.tree.DecisionTreeClassifier
Examples using sklearn.tree.DecisionTreeClass:

f1 f2 ... fd

`n_samples`) are the minimum number of samples for each node.

Changed in version 0.18: Added float values for fractions.

min_weight_fraction_leaf : *float, default=0.0*

The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

max_features : *int, float or {"auto", "sqrt", "log2"}, default=None*

The number of features to consider when looking for the best split:

- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a fraction and `int(max_features * n_features)` features are considered at each split.
- If "auto", then `max_features=sqrt(n_features)`.
- If "sqrt", then `max_features=sqrt(n_features)`.
- If "log2", then `max_features=log2(n_features)`.
- If None, then `max_features=n_features`.

Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features.

...e or None, default=None

sklearn.tree.DecisionTreeClassifier

class sklearn.tree.DecisionTreeClassifier(*, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *class_weight=None*, *ccp_alpha=0.0*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy"}, default="gini"*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *{"best", "random"}, default="best"*
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int, default=None*

hen nodes are expanded until all leaves are

sklearn.ensemble.RandomForestClassifier

class sklearn.ensemble.RandomForestClassifier(*n_estimators=100*, ***, *criterion='gini'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features='auto'*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *bootstrap=True*, *oob_score=False*, *n_jobs=None*, *random_state=None*, *verbose=0*, *warm_start=False*, *class_weight=None*, *ccp_alpha=0.0*, *max_samples=None*) [source]

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree.

Read more in the User Guide.

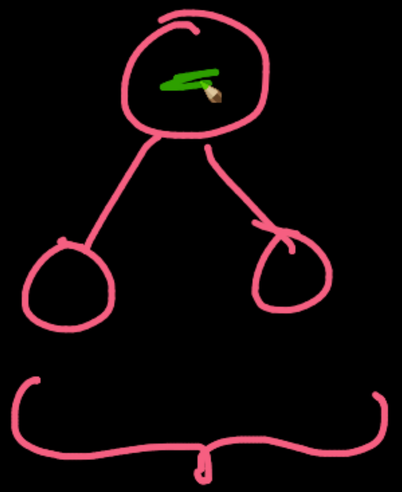
Parameters:

n_estimators : *int, default=100*
The number of trees in the forest.

...ue of `n_estimators` changed from 10 to

decision stump:

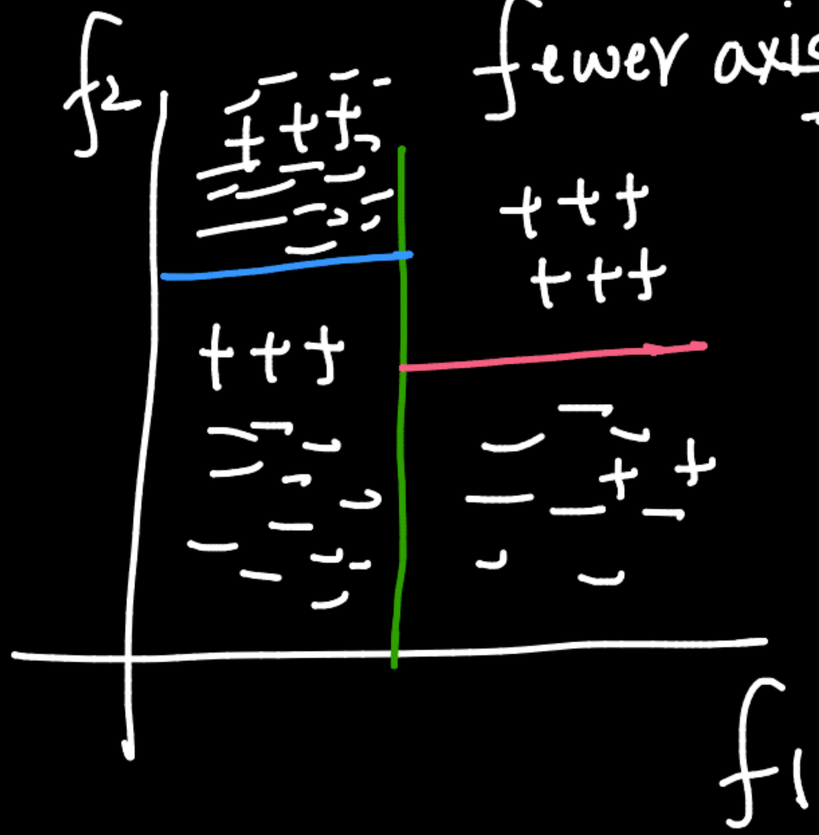
↳ DT of depth = 1



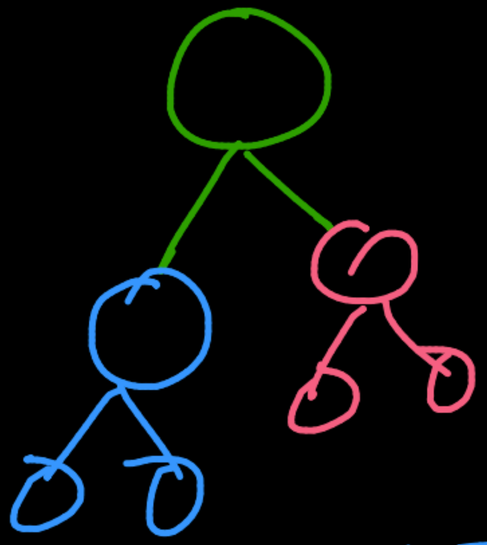
Shallow tree: depth is small

Deep tree: depth is large..

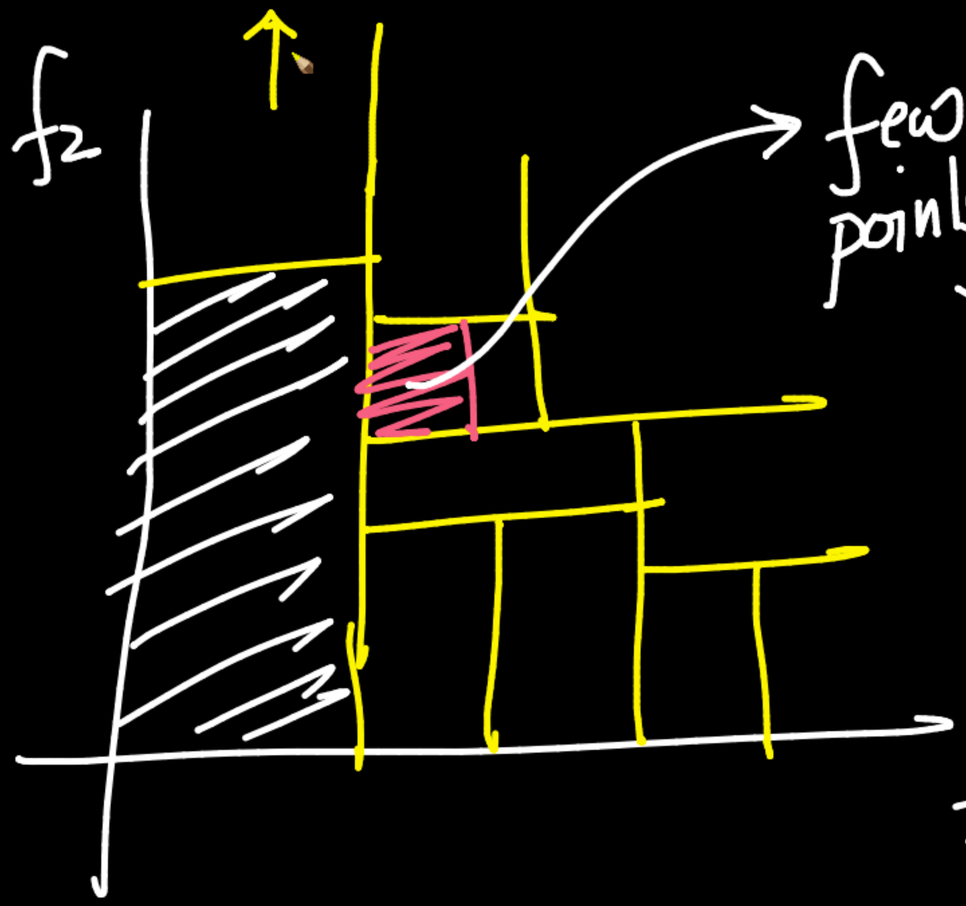
Geom:



fewer axis ∥ planes ← depth is low



Shallow-bet



depth is high

could be noise / outlier

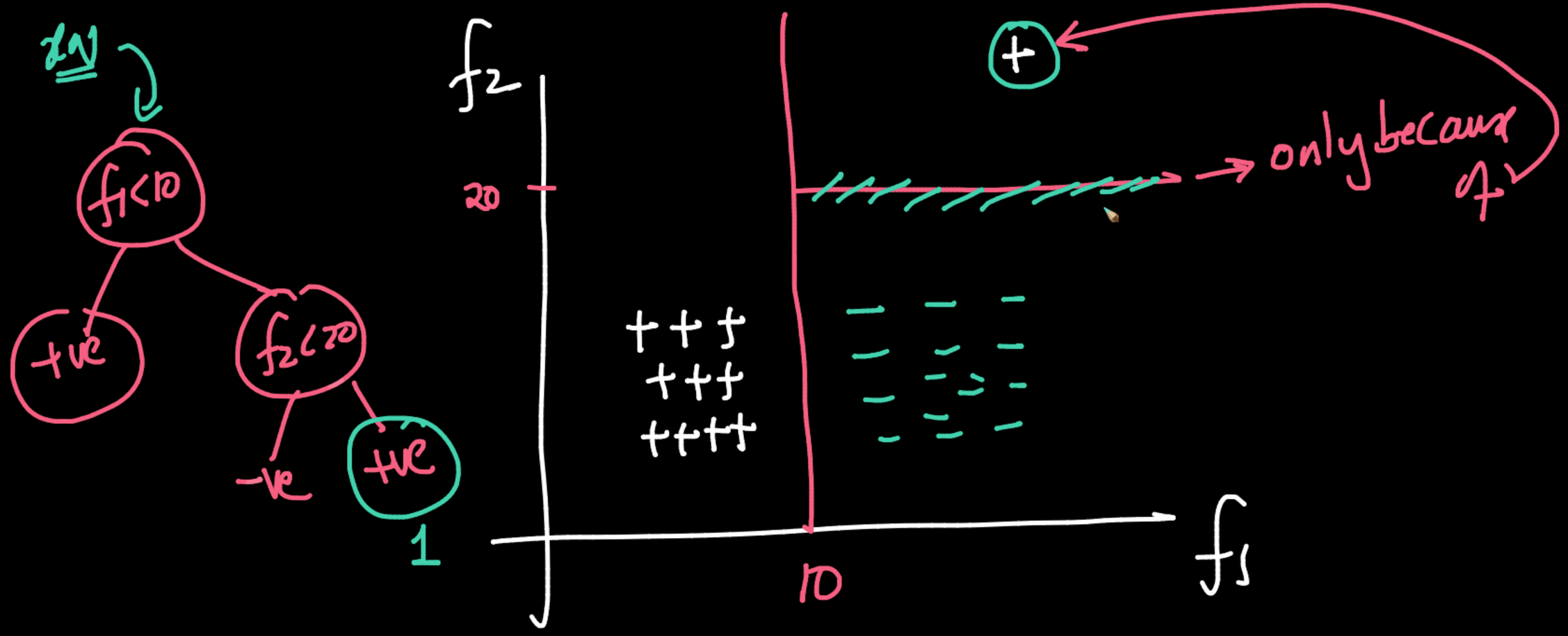
higher chance of overfitting

2D – rectangle

3D – cuboid

d-dim: hyper cuboid

Outliers ⟶ impact when depth is large


f1<10
+ve
f2<20
-ve
+ve
1
f2
20
+++
+++
++++
10
f1
+
only because of

- non-linear data
- 'd' is not too large

Tabular-data

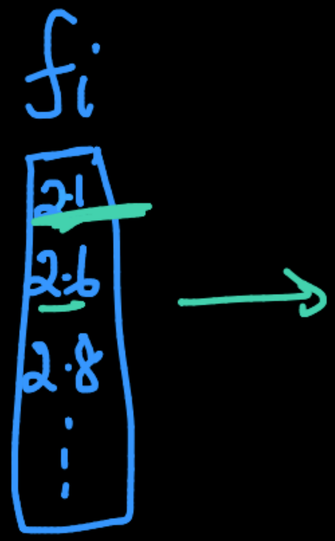
DT → RF & GBDT

Q

Linear & logistic & PCA → Standardize the data

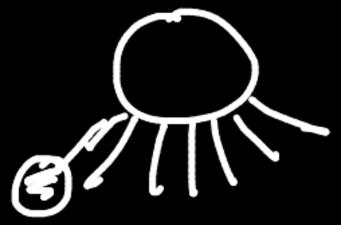
Is Standardization of data needed for DT

doesn't impact IG



Q

Categorical features: Zipcode (10,000) → encoding for DT

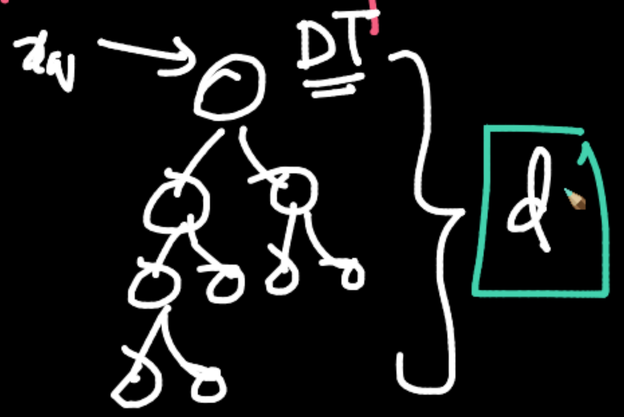


- no encoding ✗ → (entropy messed up)
- Target encoding ✓
- one hot encoding ✗ → dt

[Zipcode → City

(4) appropriate feature encoding also depends on Model using

Runtime - complx



Time : $O(depth) \rightarrow (\lg m)$

Space: $O(m)$

m = # nodes

n = # data pts

depth best using CV → DT is very efficient @ runtime

RF / GBDT

(100's of DTs)

10K QPS

(≤ 2msec)

Python — sklearn... ~20ms — V.V slow

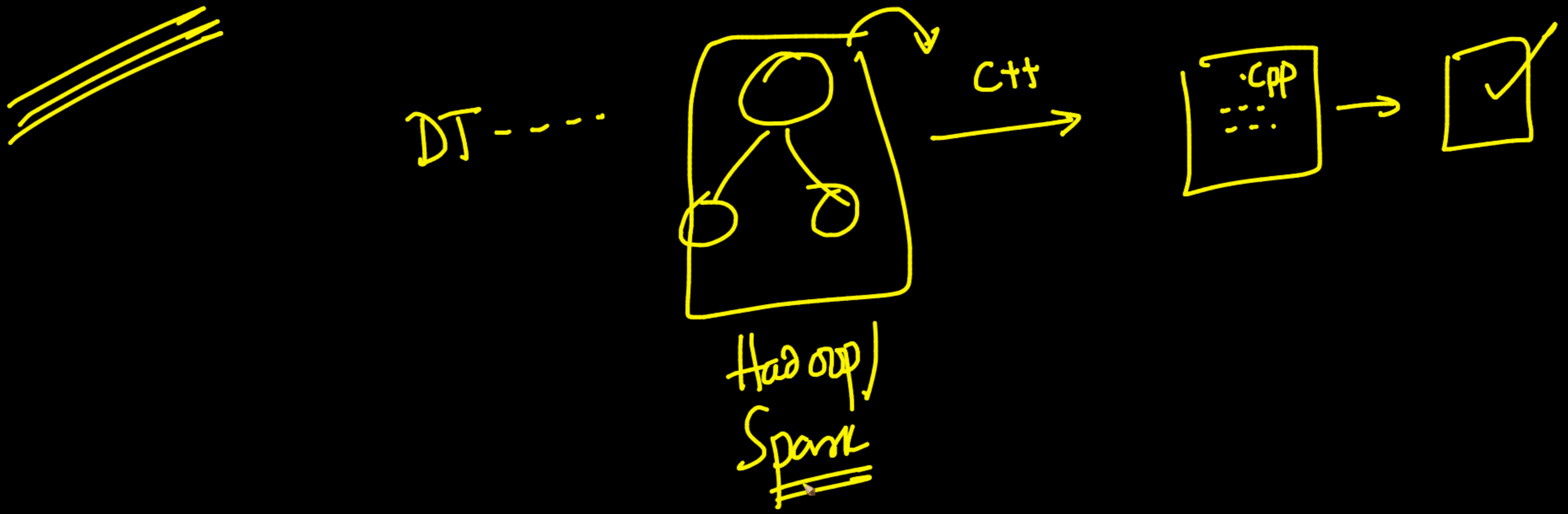
Java — if...else...

~5msec

C++ → if...else

1-3ms

✓ FBlearner → feed-ranking


DT - - - -
C++
.cpp
Hadoop/
Spark

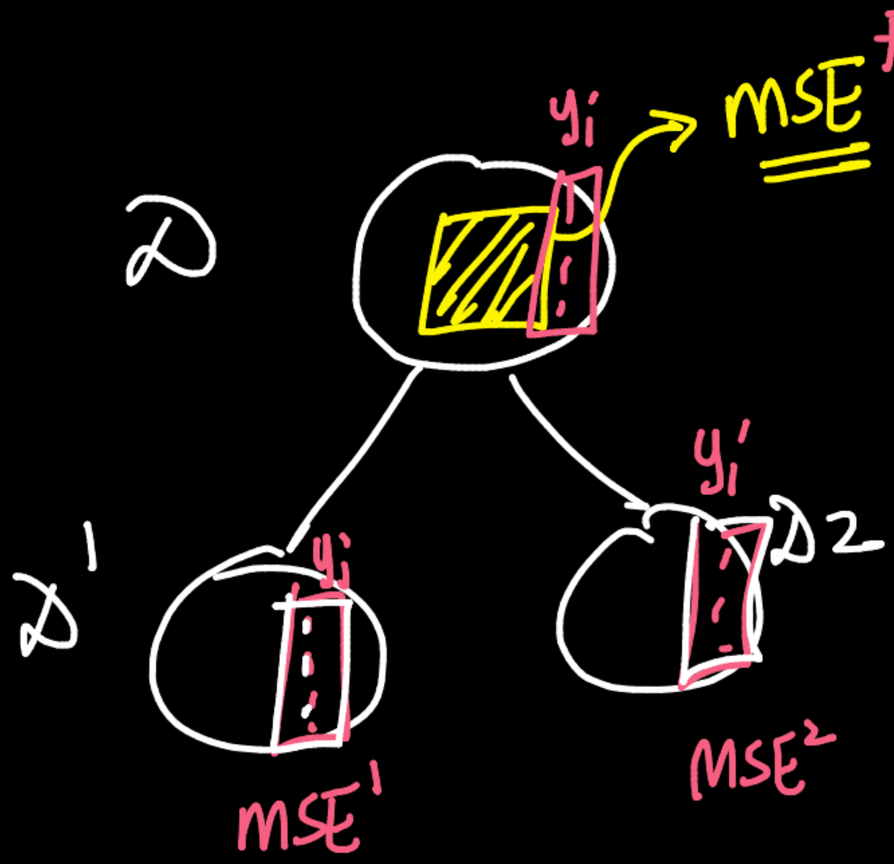
Regression on DT

- Entropy & GI → alternative → MSE or MAE
- leaf-node

y_i's

y_q : → mean/median

Classfn :- $y_+ : 500$; $y_- : 200$ → +ve or $\frac{5}{7}$



$y_i \in \mathbb{R}$

[all y_i's are same @ a node

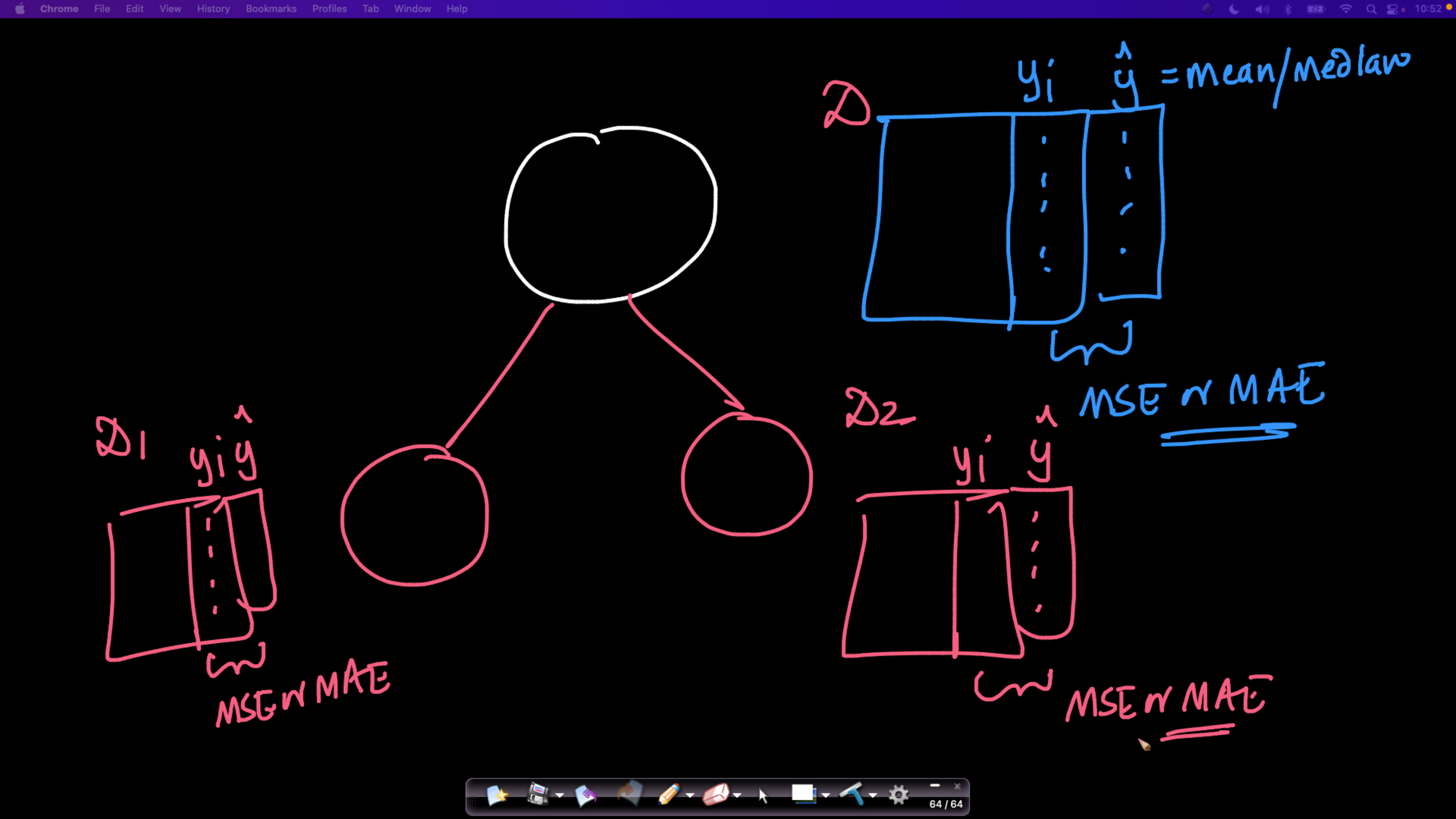
↓

MSE or MAE = 0

{ Weighted MSE of child nodes − MSE of parent node

MSE : lowest ⟶ all y_i's are same

MSE : Max ⟶ y_i's are very divergent

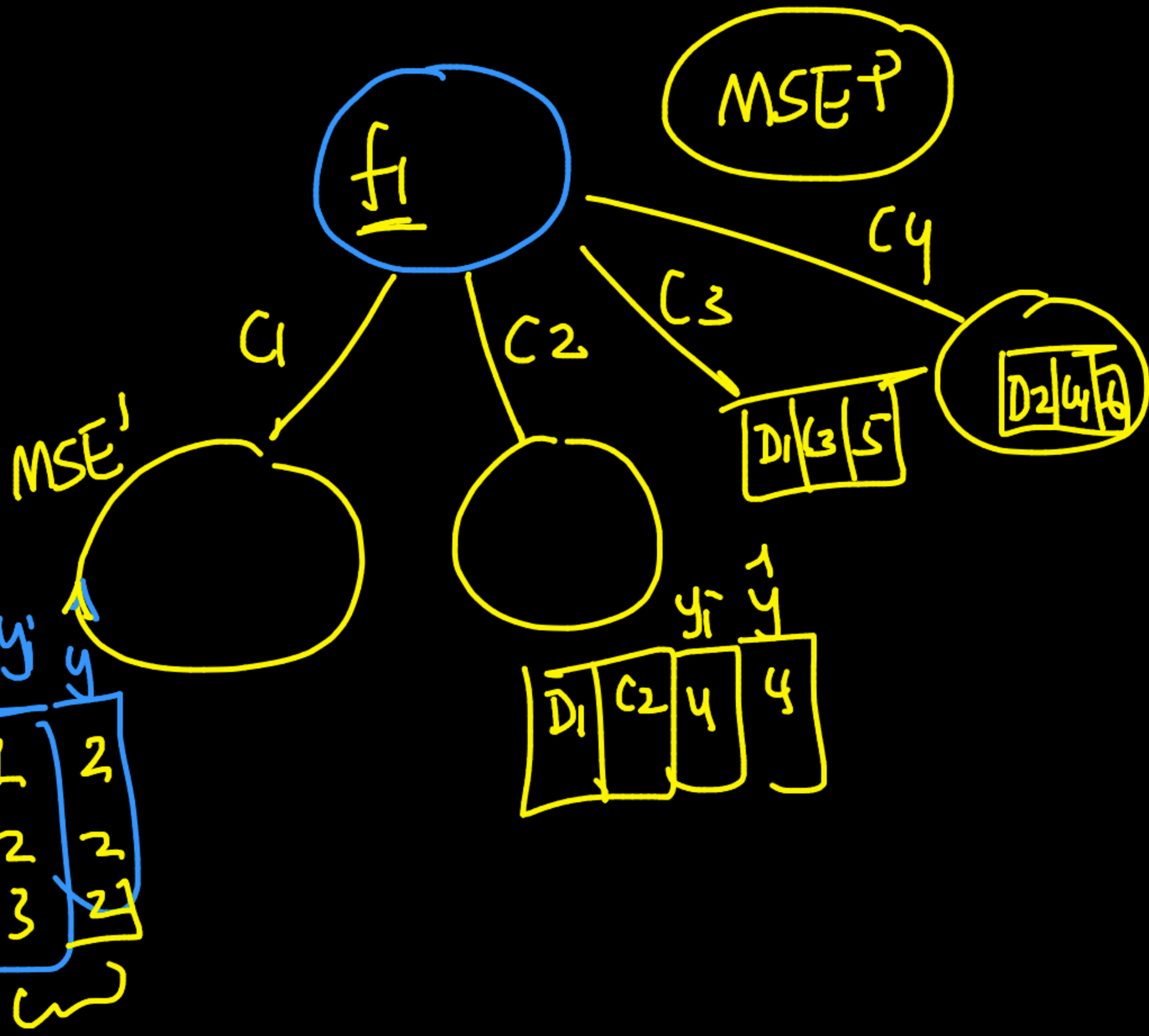

D
y_i
$\hat{y}$ = mean/median
MSE or MAE
D_1
y_i $\hat{y}$
MSE or MAE
D_2
y_i $\hat{y}$
MSE or MAE

f_2	f_1	y_i	$\hat{y}$
D_1	C_1	1	3.5
D_2	C_1	2	3.5
D_3	C_1	3	3.5
D_1	C_2	4	3.5
D_1	C_3	5	3.5
D_2	C_4	6	3.5

$n=6$

$\Delta MSE \rightarrow f_1$

$\rightarrow f_2$



f_1	f_2	y_i	$\hat{y}$
D_1	C_1	1	2
D_2	C_1	2	2
D_3	C_1	3	2

Q

Entropy or G.I ?

$$-\sum_{i=1}^{K} \log\left(\tilde{P}(y_i)\right)\, \tilde{P}(y_i)$$

Q

MSE or MAE ? ✓

MSE → outlier, impact is high

MAE → outlier resistant

Var vs MAD ✓

Imabalanced data in DT → Class weights : Upsampling SMOTE

↳ rebalance it

entropy or G·I

p(yi)

$y_+ : 99\%$.

$y_- : 1\%$.

Multi-class -classfn:-

→ entropy & G.I ✓

→

leaf-node

y_1: 300; y_2: 10; y_3: 50

↓

Majority-class

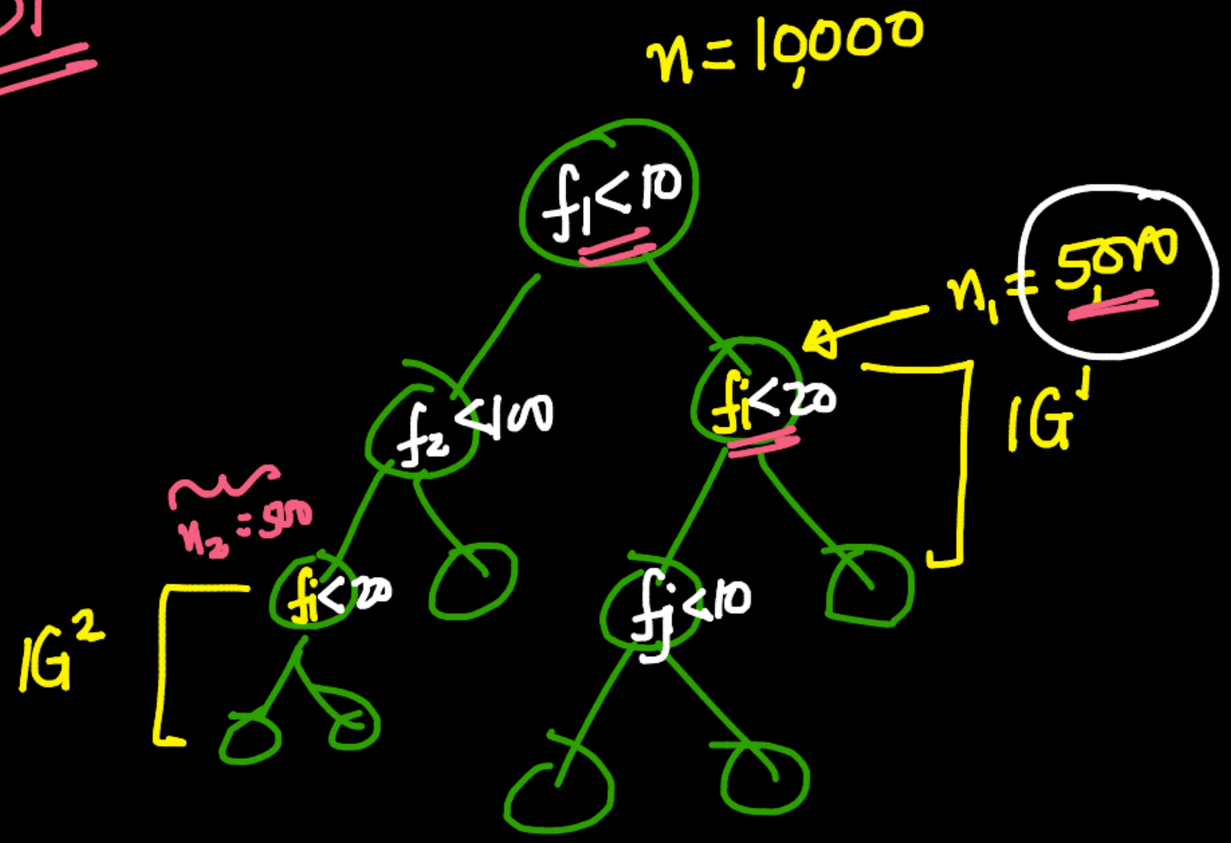
Interpretable:-

if ... else ...

}

→ non-expert

DT



Feature Importance

→ IG ✓

→ order of nodes

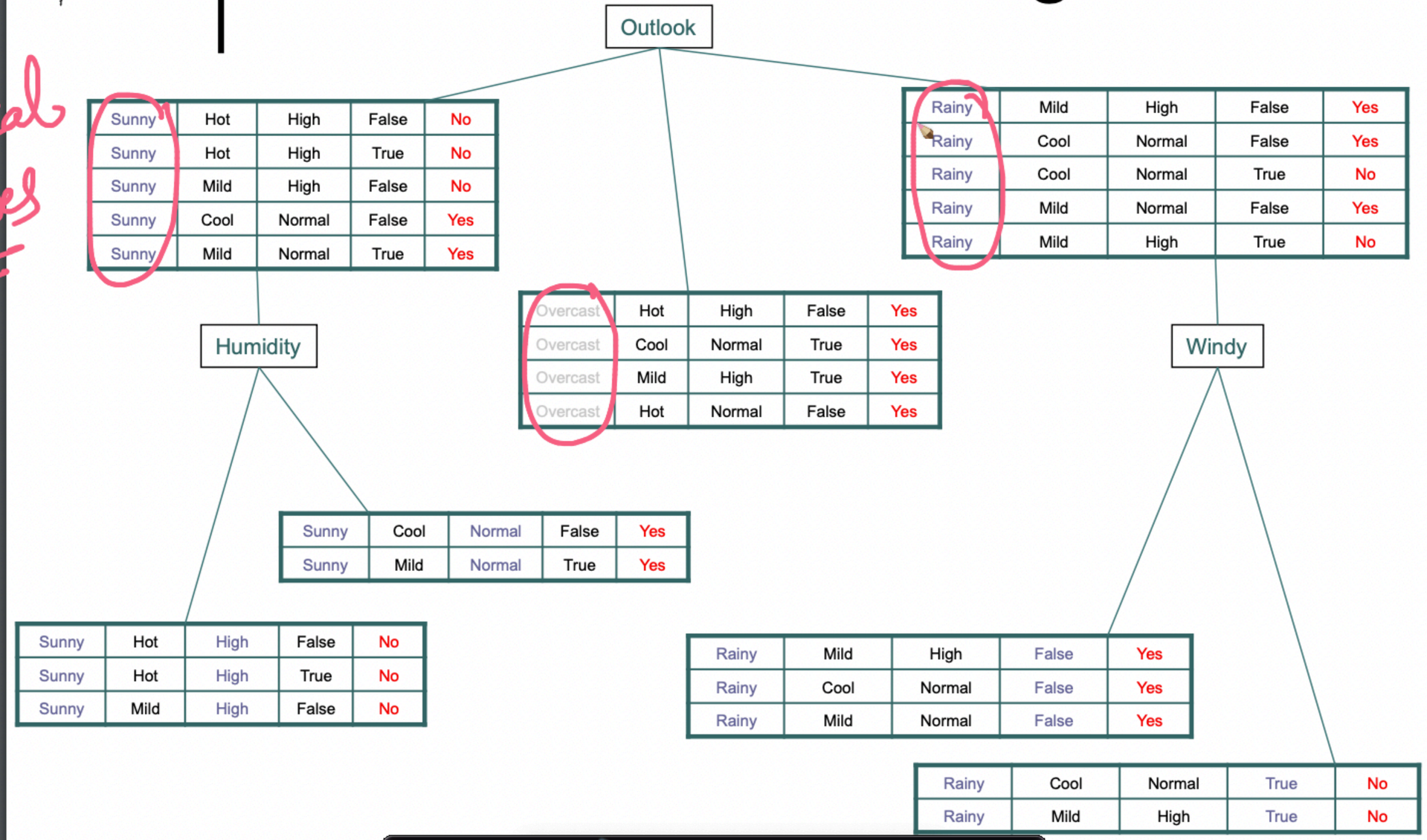
→ normalized info-gain ...

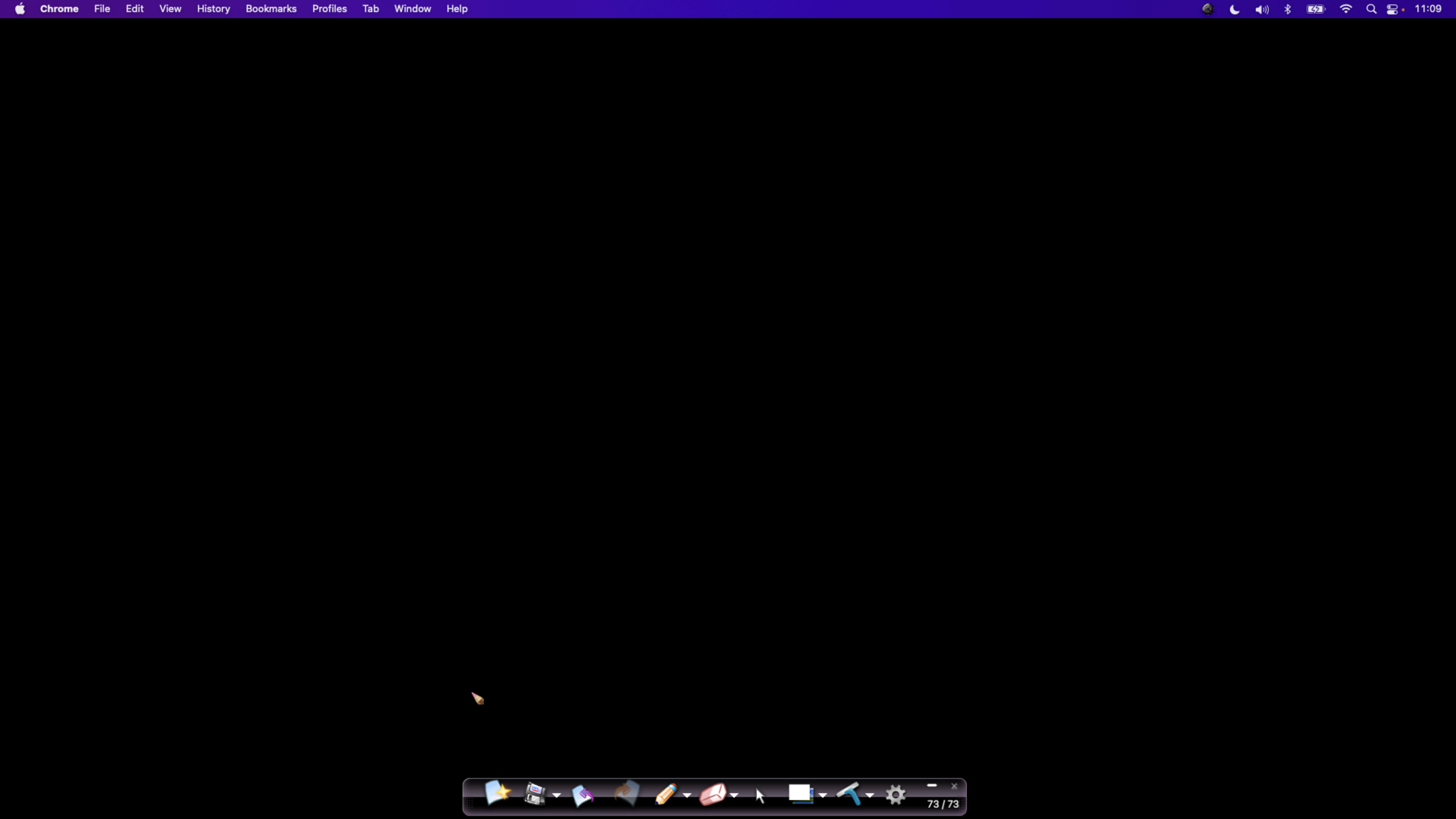
fi : $IG^1 * \frac{5000}{10000}$ + ✓

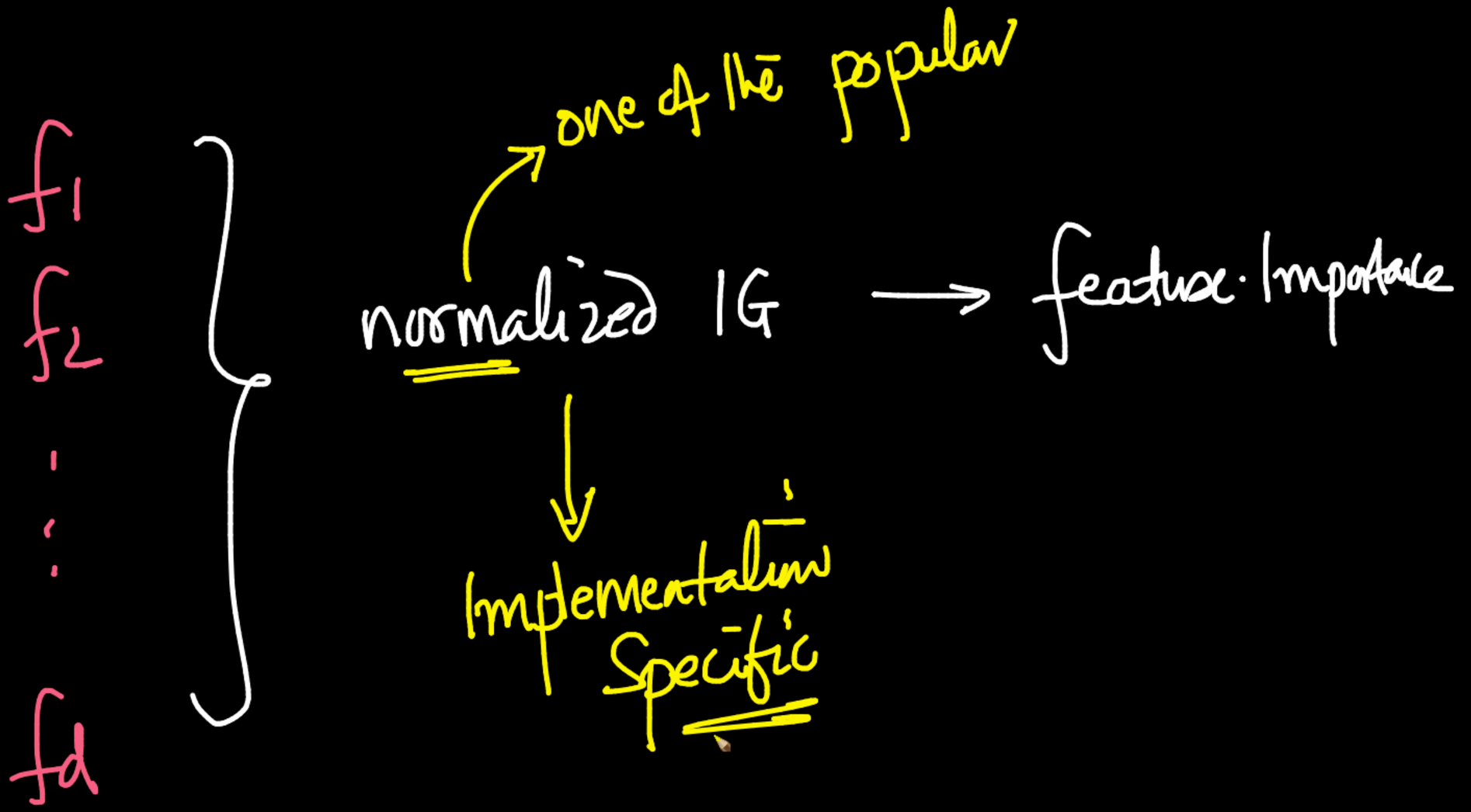
$IG^2 * \frac{500}{10000}$

Recursive Partitioning

Categorical features



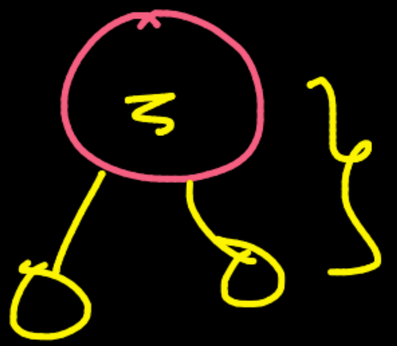

Chrome File Edit View History Bookmarks Profiles Tab Window Help
11:09
73 / 73



FYI

Optimal DT → NP-Complete (expo. hard time complx)

→ greedy approxi malem



DT:

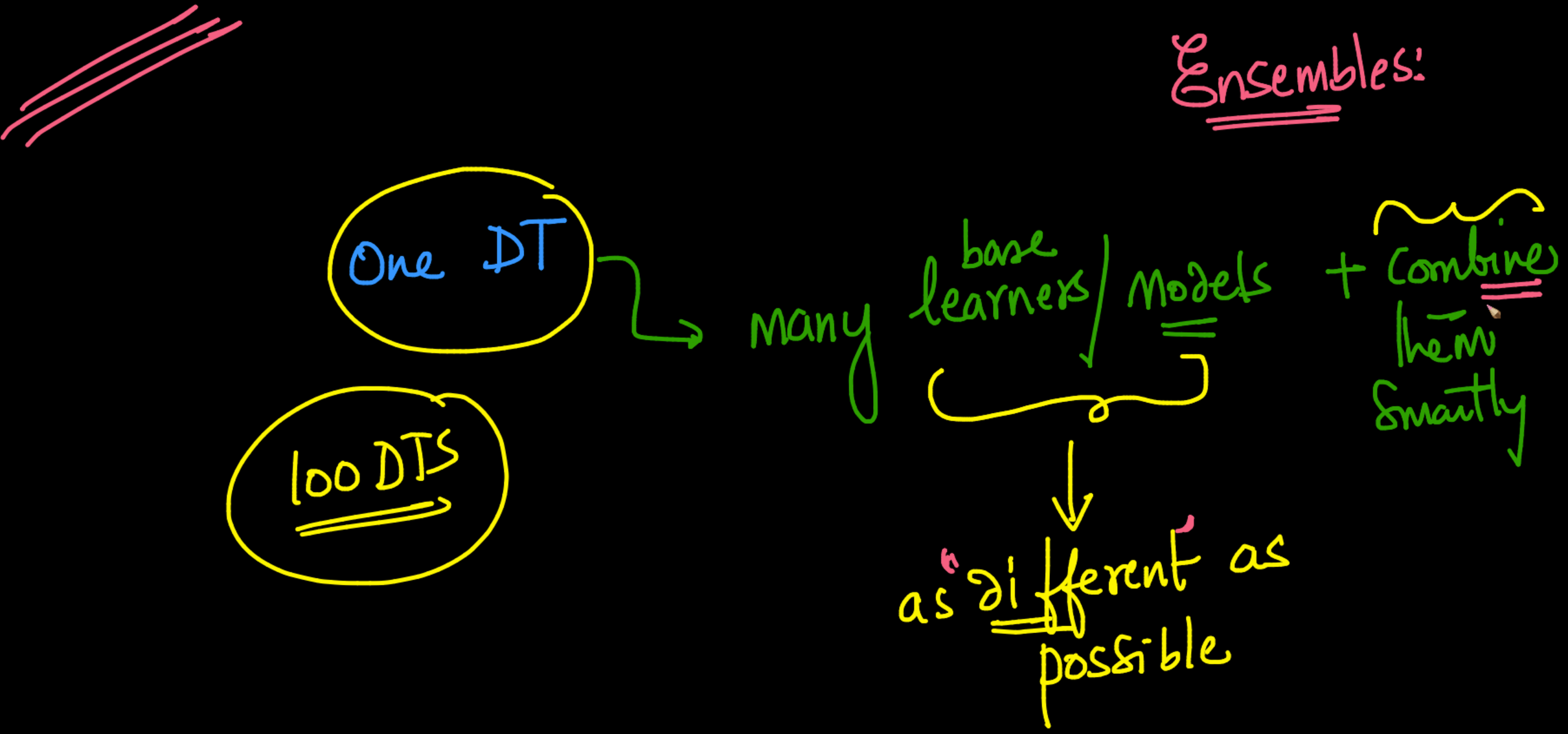
→ d is not too large

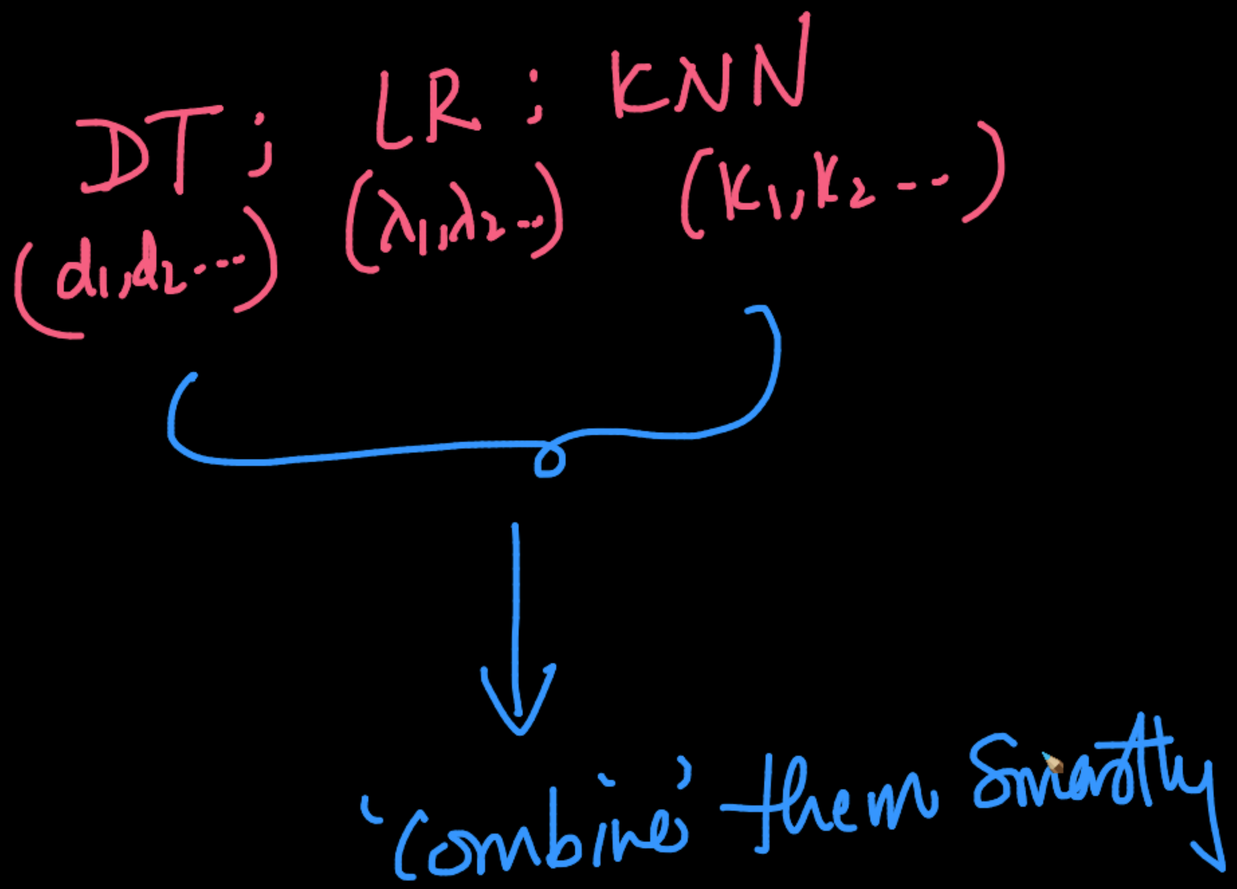
→ low runtime complx

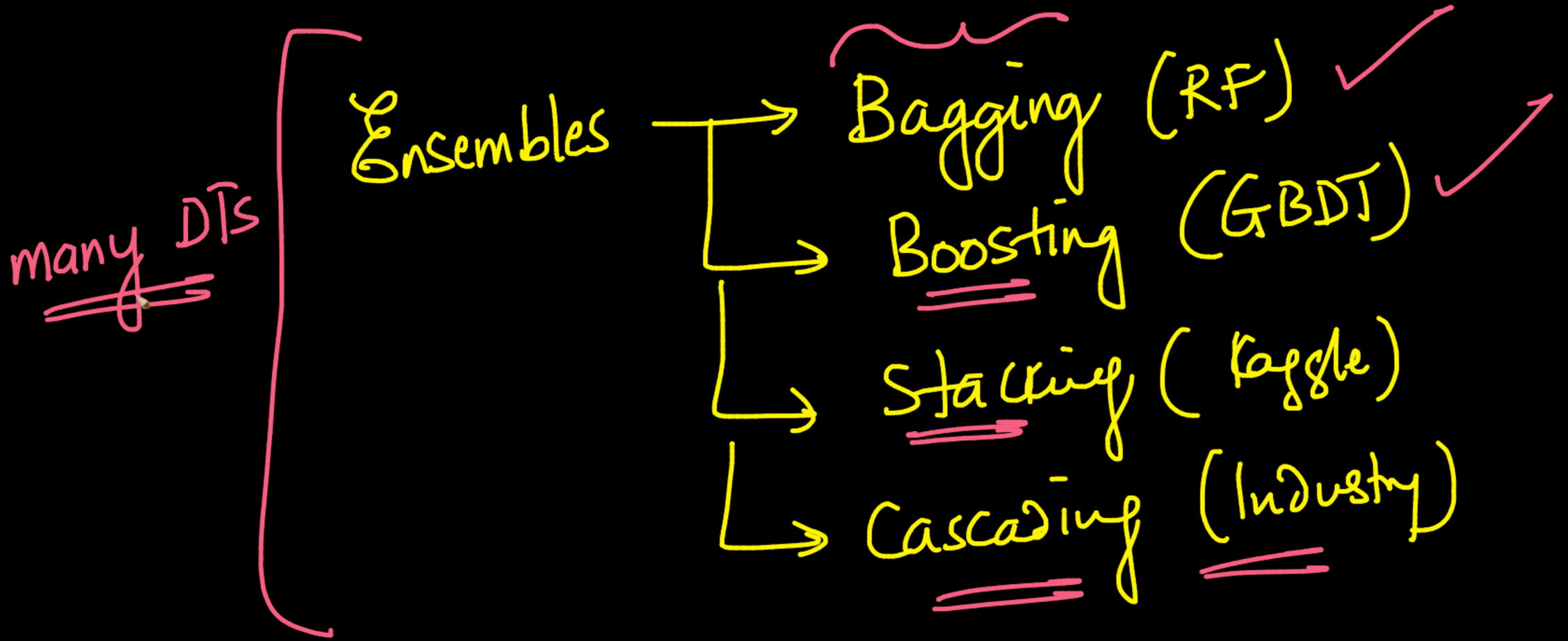
→ no standardization needed

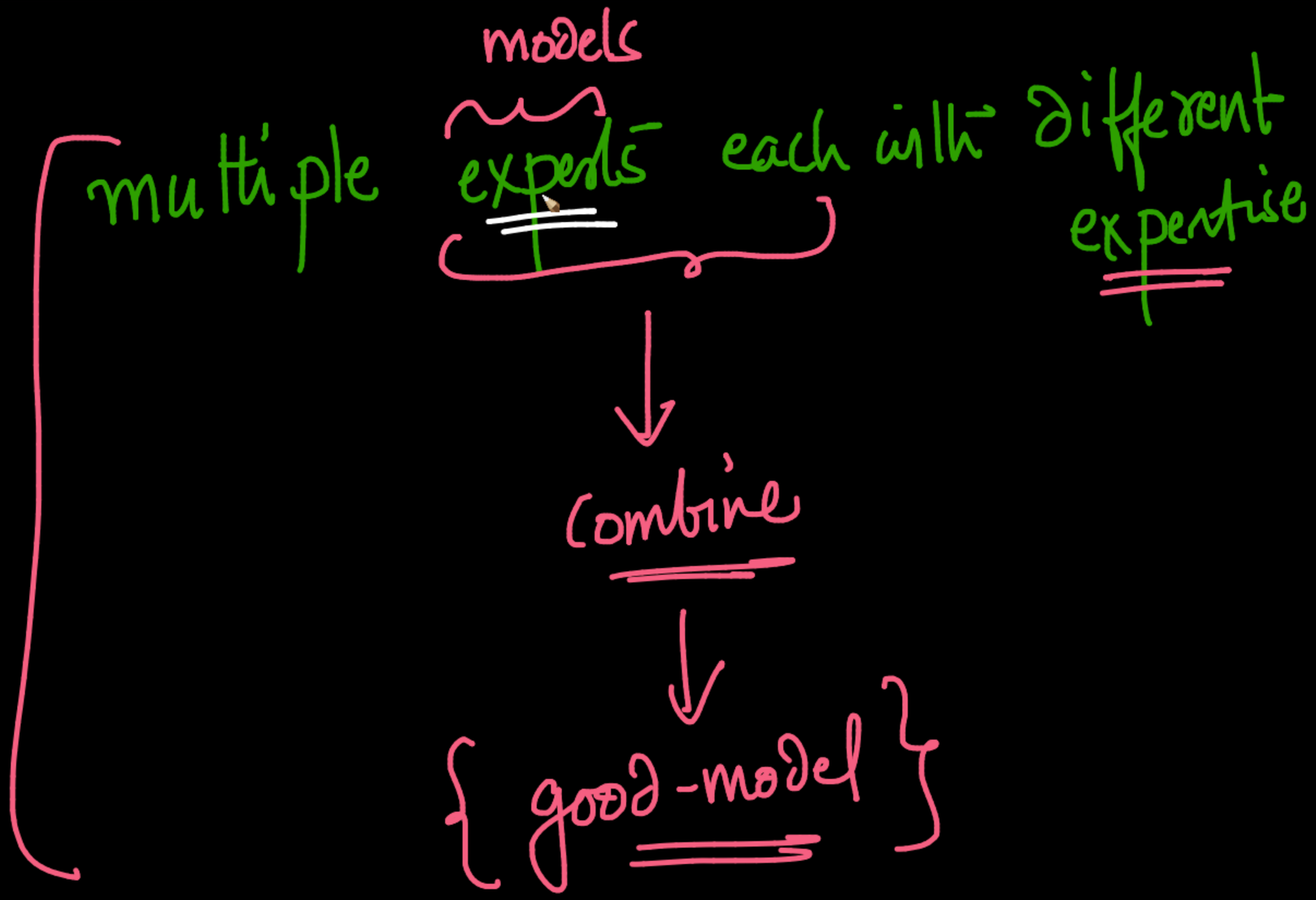
→ Depth is the key hyperparam

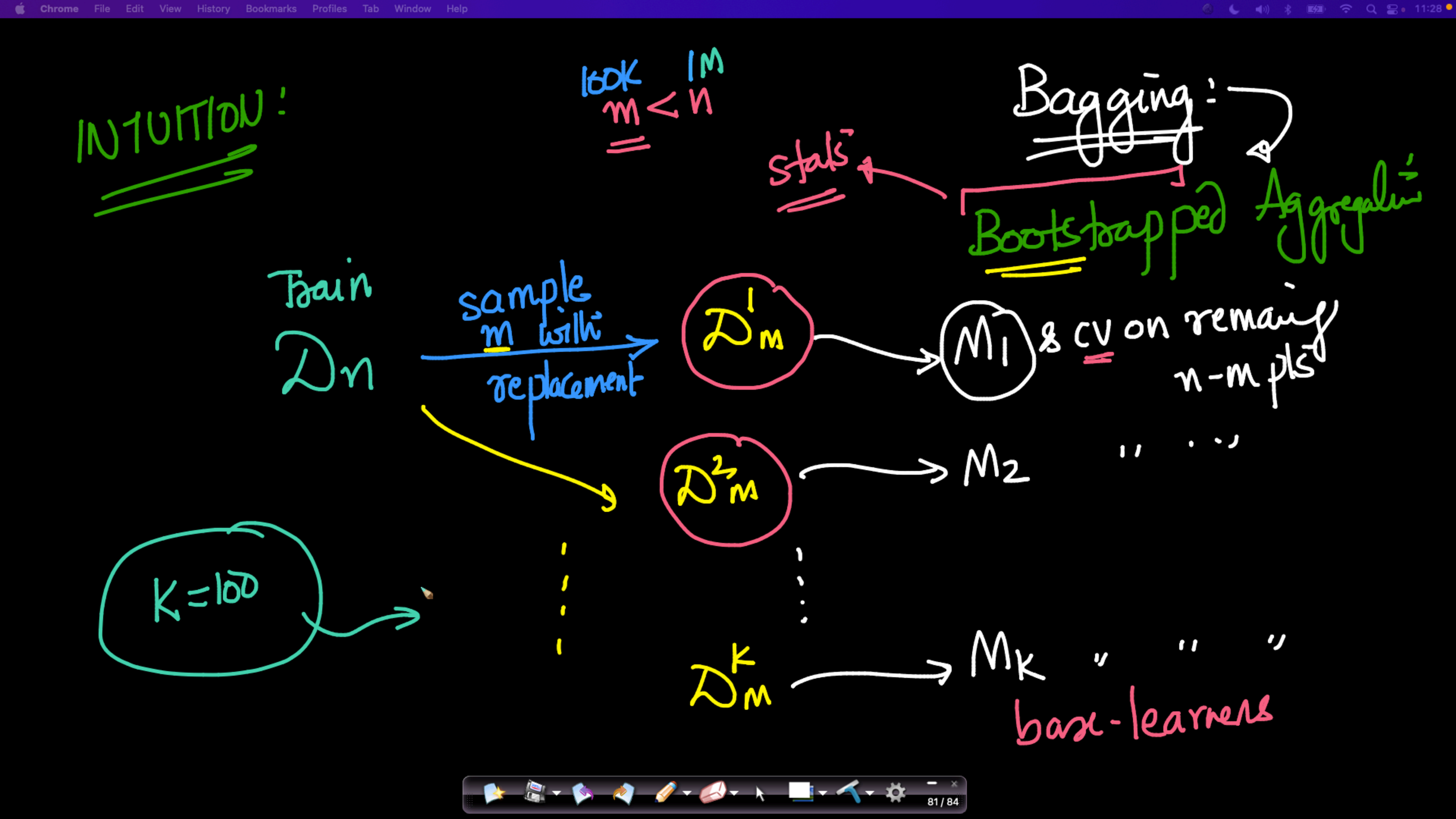
Ensembles:

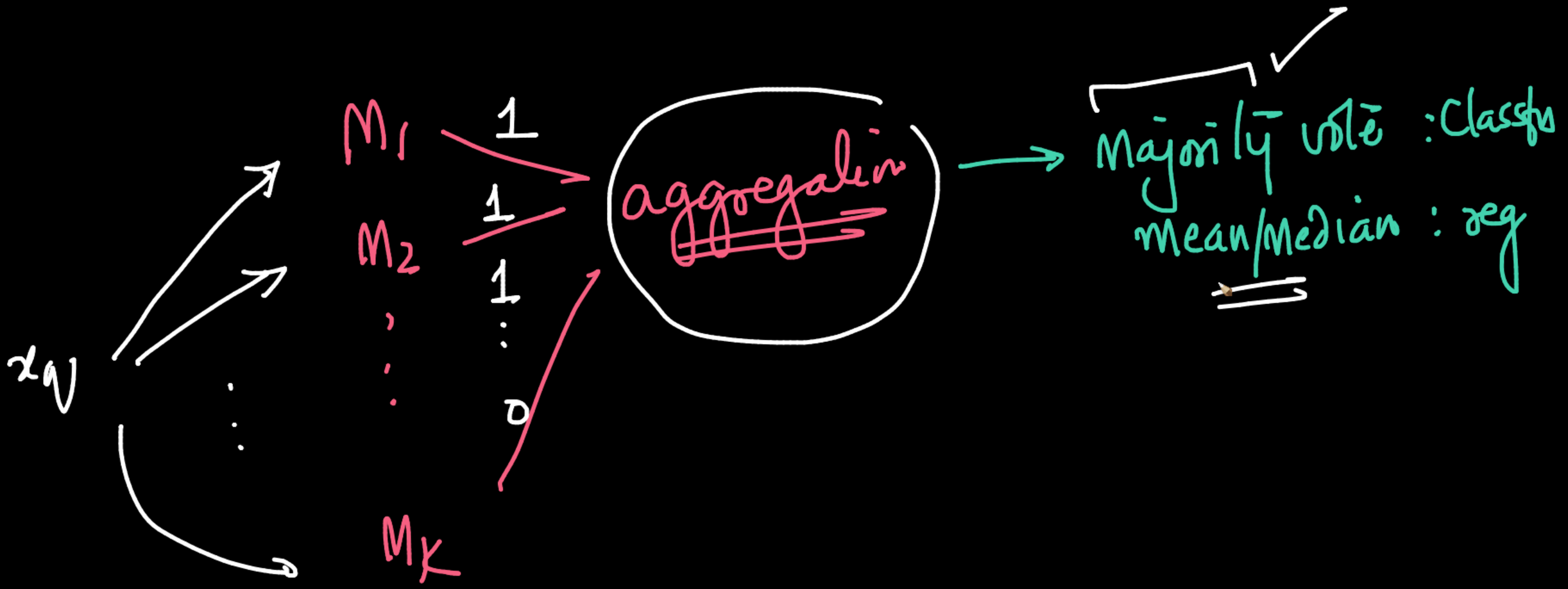





many DTs
Ensembles
Bagging (RF)
Boosting (GBDT)
Stacking (Kaggle)
Cascading (Industry)




INTUITION:
look IM
m < n
Bagging:
stats
Bootstrapped Aggregating
Train
D_n
sample m with replacement
D_m^1
M_1
& cv on remaining n-m pts
D_m^2
M_2
D_m^k
M_k
base-learners
K=100



DT

Challenge: $M_1, M_2 \ldots M_K$ are different from one another?

$K \approx$ 100's

RF

ideas:

- different sets of features for each tree : feature sampling → $d' < d$
- tune the depth
- M_i & M_j would be more diff if m is smaller

already happening in rf sampling

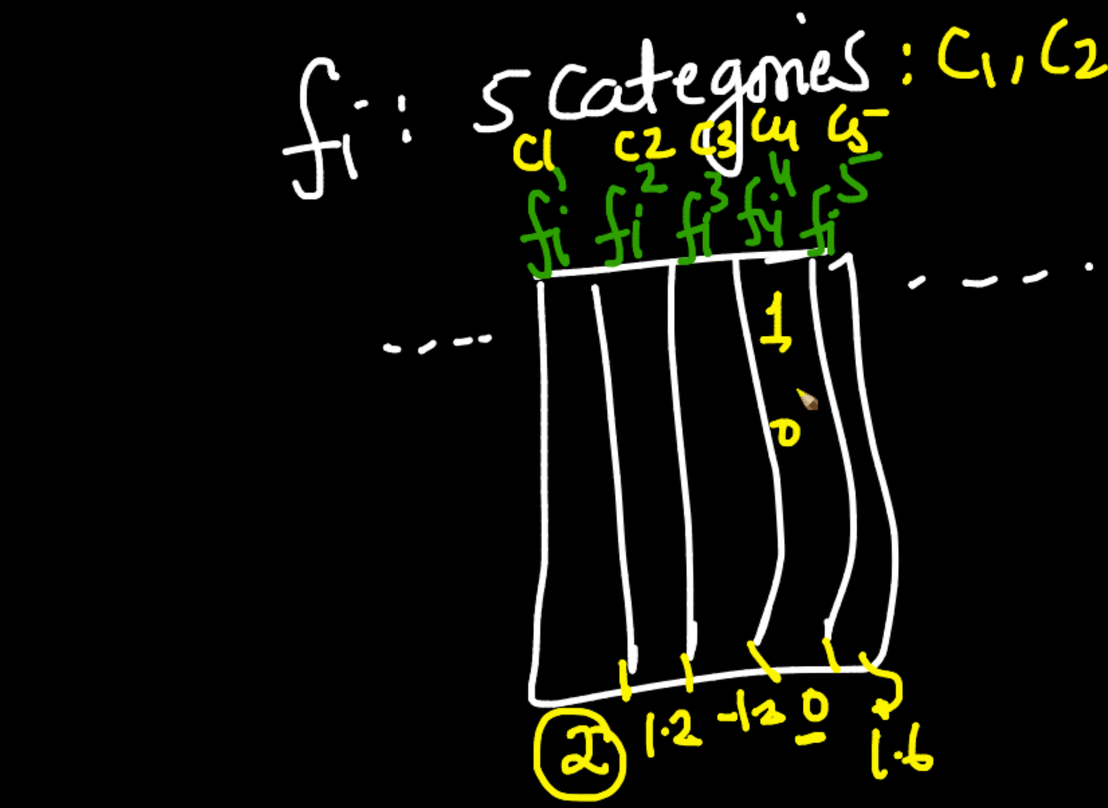
Logistic Reg → OHE

5-15 categories

$C_1 = 1$ = True

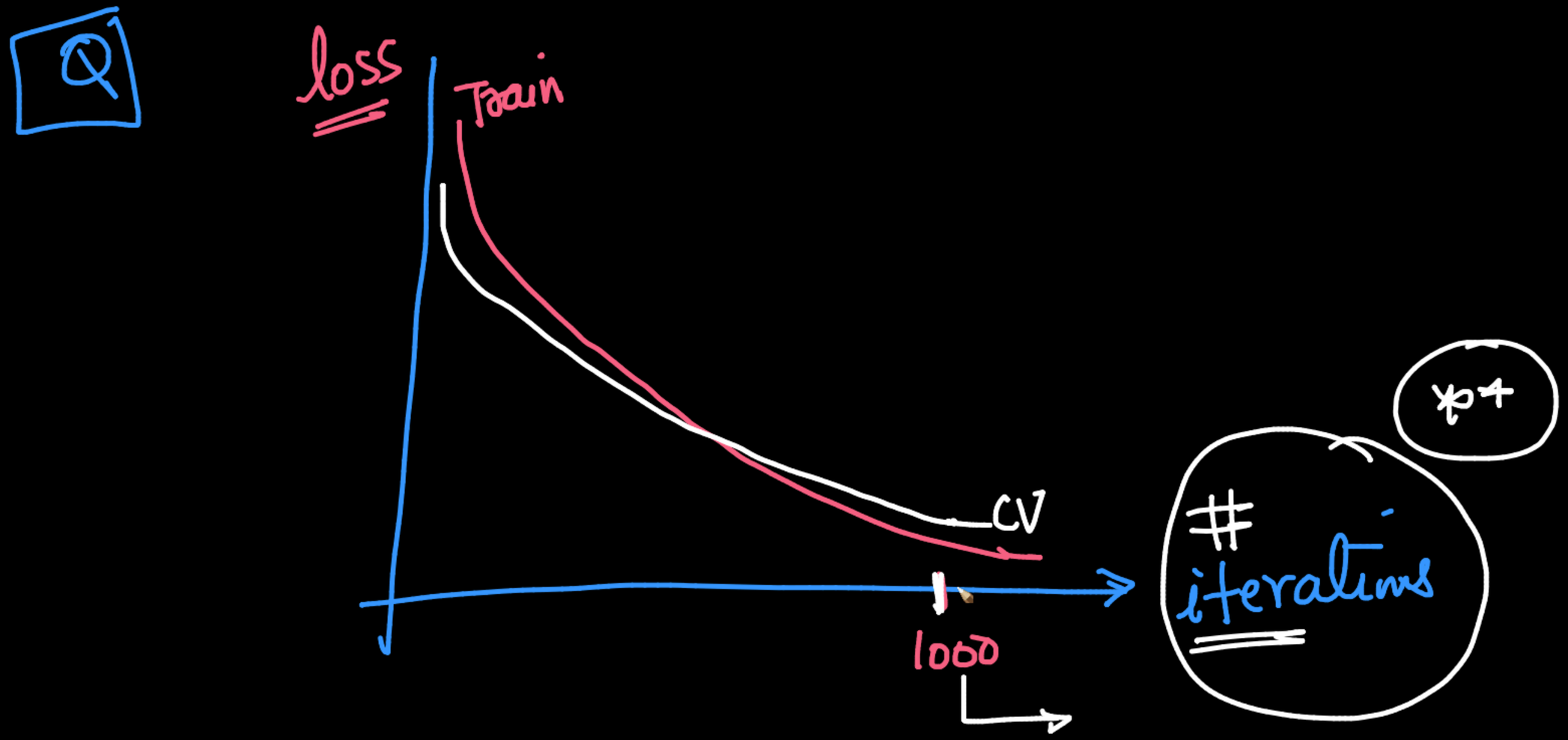
$W = 2$ →

fi : 5 categories : $C_1, C_2 \ldots C_5$



$C_3 = 1$ = True

$W = -1.2$

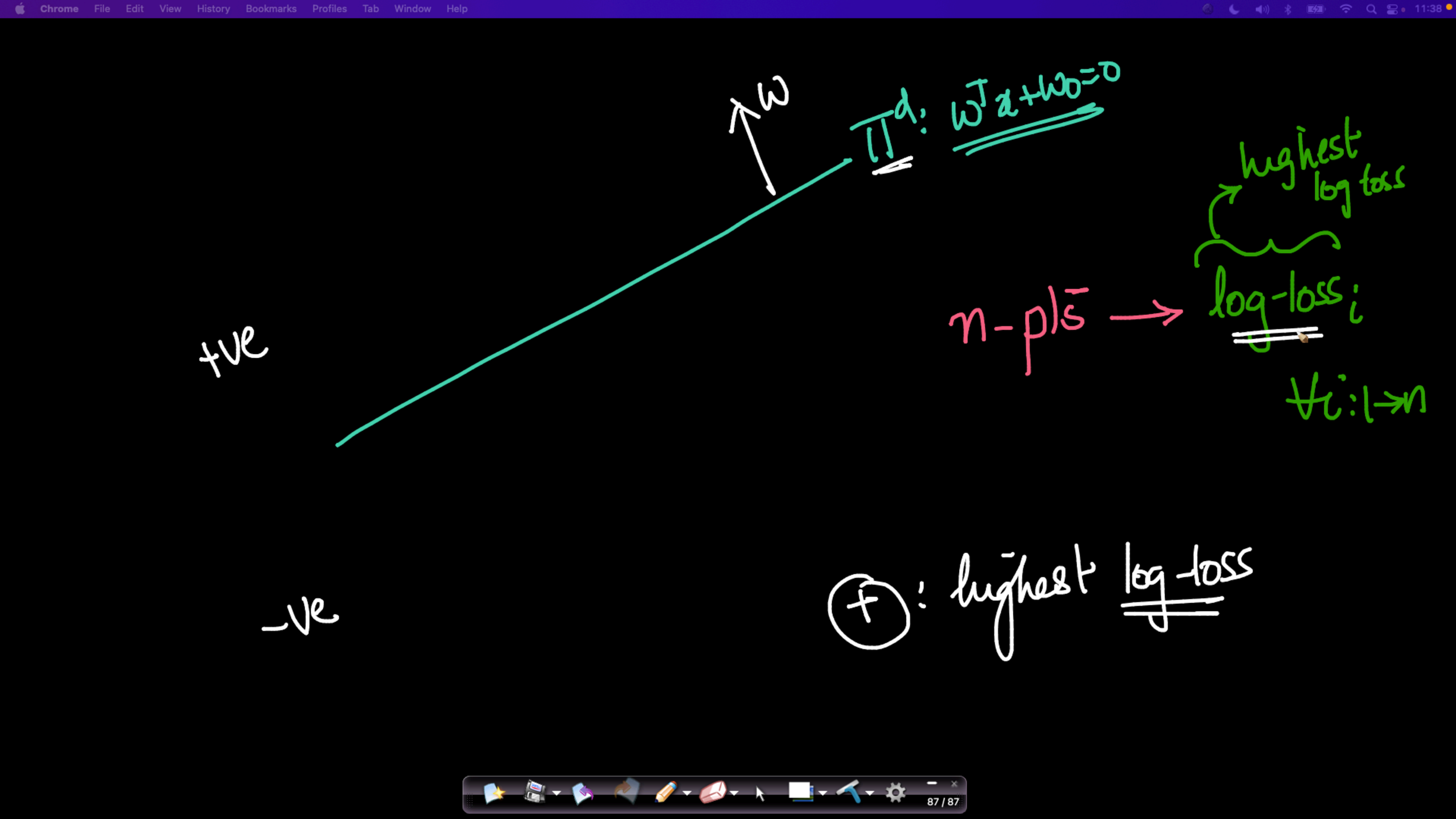

Q
loss
Ttrain
CV
1000
iterations
yp+

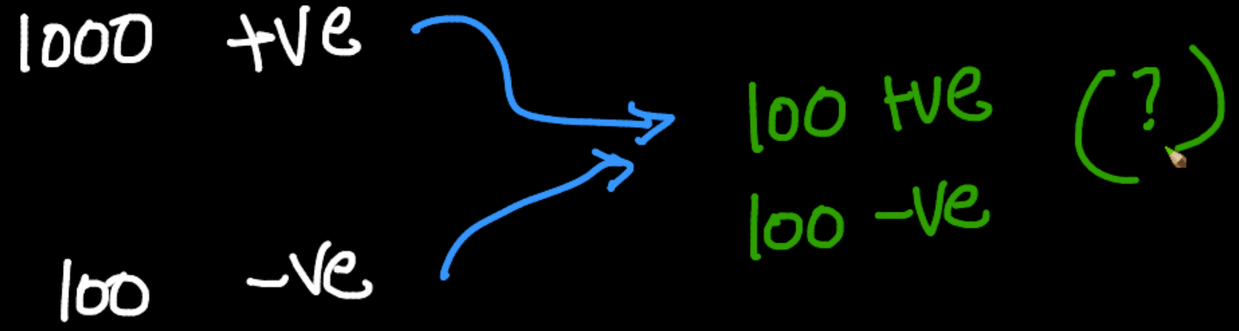
Regression using DT

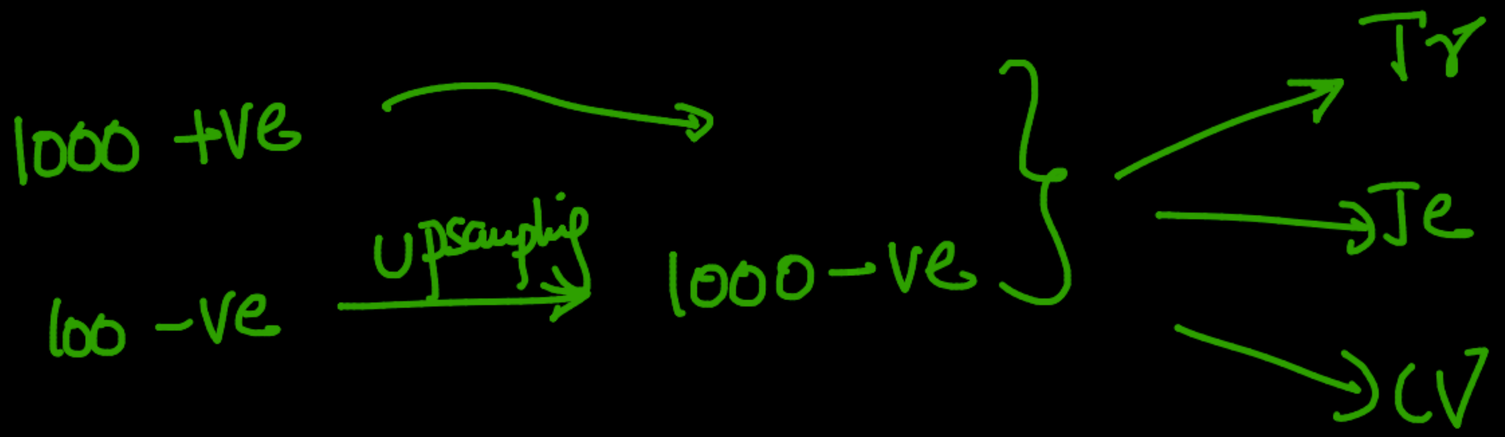
MSE or MAE

Variance

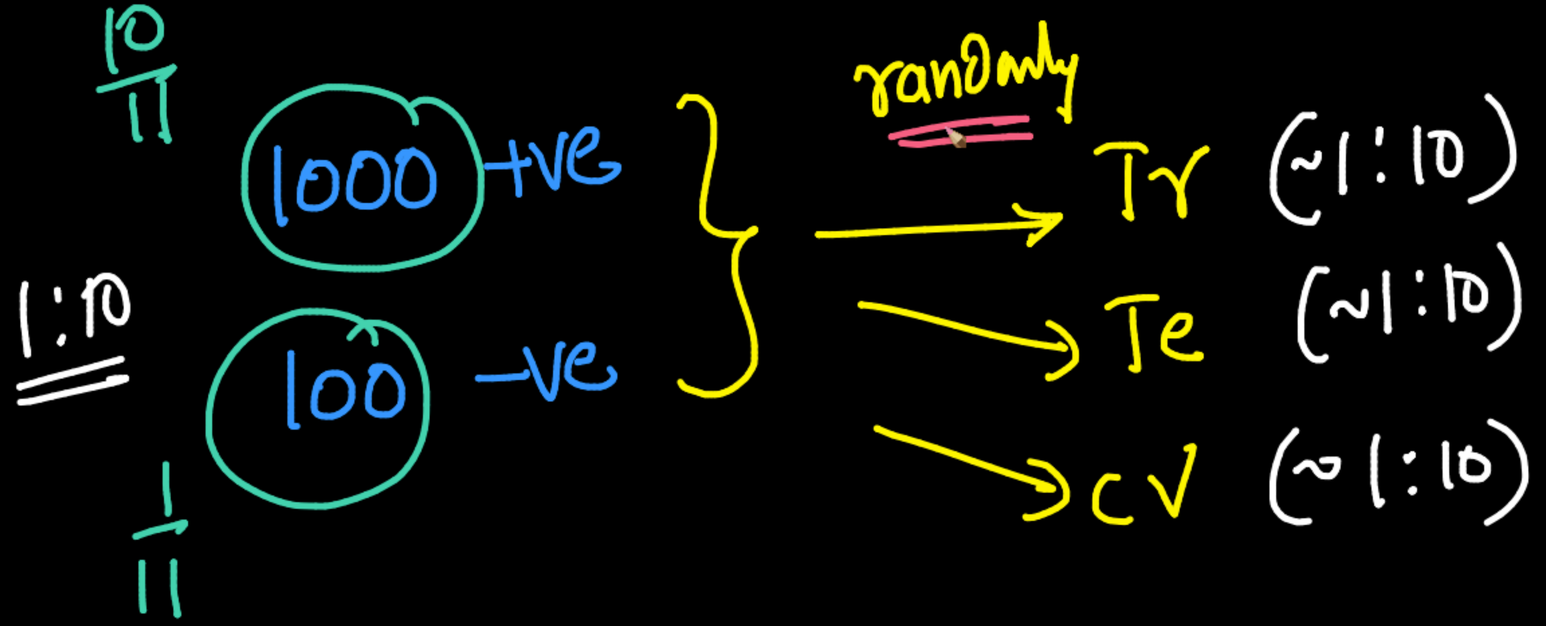
MAD

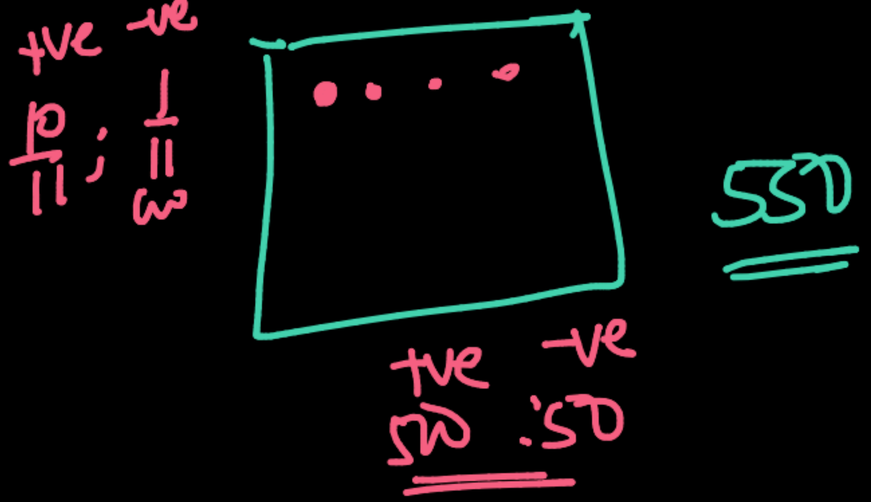

w
Π^d: w^T x + w0 = 0
highest log loss
+ve
n-pts → log-loss_i
∀i: 1→n
-ve
+: highest log-loss


1000 +ve
100 -ve
100 +ve
100 -ve
(?)



Class-weights


10/11
1000 +ve
1:10
100 -ve
1/11
randomly
Tr (~1:10)
Te (~1:10)
CV (~1:10)


+ve -ve
10/11 ; 1/11
550
+ve -ve
500 :50

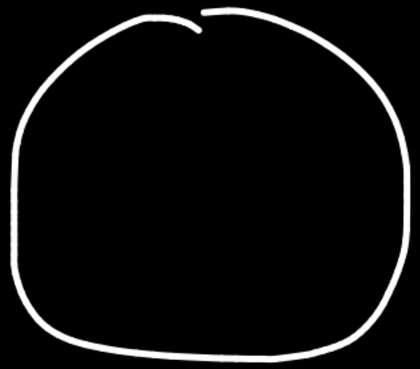
extreme

1 : 1000

+ve : 10000

-ve : 10

V. few



Bernoulli

+ve : 1000 / 1001

-ve : 1/1001

$X \sim Binomial(n = 500; p = \frac{1}{1001})$

$P(X = 1) =$

R^2: adjusted R^2

cover it later

Recursive Partitioning

Outlook

Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes

Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes

Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Rainy	Mild	Normal	False	Yes
Rainy	Mild	High	True	No

Humidity

Windy

Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes

Sunny	Hot	High		
Sunny	Hot	High	True	No

			False	Yes