Ensembles: (DT)

- Bagging + RandomForest
  - Bootstrapped Aggregation
- = class starts @ 9:03 PM

```
[1]  Downloading...
     From: https://drive.google.com/uc?id=12Bh2AN8LcZAlg20ehpQrEWccUDaSdsOG
     To: /content/y_test.pickle
     100% 9.49k/9.49k [00:00<00:00, 14.3MB/s]
```

```
[2]  import pickle
     import pandas as pd
     # Load data (deserialize)
     df = pd.read_csv("HR-Employee-Attrition.csv")

     with open('preprocessed_X_sm.pickle', 'rb') as handle:
         X_sm = pickle.load(handle)

     with open('y_sm.pickle', 'rb') as handle:
         y_sm = pickle.load(handle)
```

depth | Train Acc | Val Acc

7 | 93.24 | 84.21 — gap

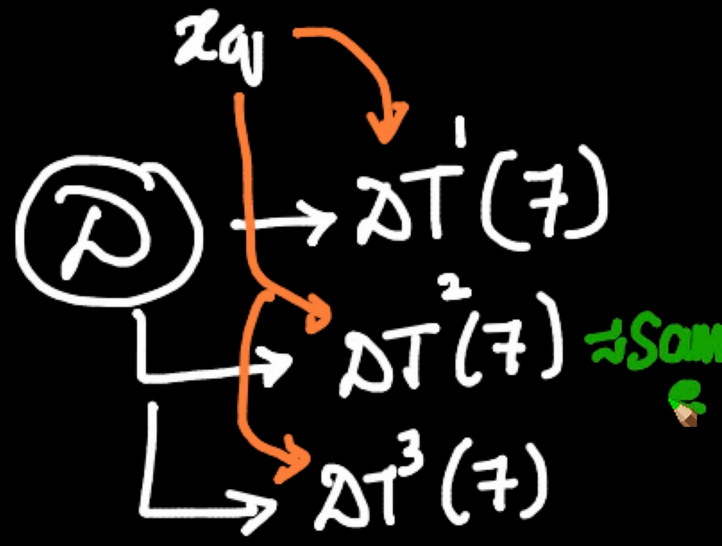
4 ← | 83.51 | 80.58 — less

```
# Using the best decision tree model as observed in the last lecture
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold, cross_validate

tree_clf = DecisionTreeClassifier(random_state=7, max_depth = 4)
kfold = KFold(n_splits=10)
cv_acc_results = cross_validate(tree_clf, X_sm, y_sm, cv = kfold, scoring = 'accuracy', return_train_score = True)

print(f"K-Fold Accuracy Mean: Train: {cv_acc_results['train_score'].mean()*100} Validation: {cv_acc_results['test_score'].mean()*100}")
print(f"K-Fold Accuracy Std: Train: {cv_acc_results['train_score'].std()*100} Validation: {cv_acc_results['test_score'].std()*100}")
```

```
K-Fold Accuracy Mean: Train: 83.50772540936214 Validation: 80.58049353701529
K-Fold Accuracy Std: Train: 0.7889219949763586 Validation: 6.337951865133366
```

One DT → base-learner

Ensembles



100's of Decision Trees ✓

↳ Combine multiple base learner

↓

'different'

DT ..... Ensembles → Bagging (RF) ✓

→ Boosting (GBDT) ✓

→ Stacking (Kaggle)

→ Cascading → industry

INTUITION:

look 1Million

$m < n$; $K = 100$

Bagging:

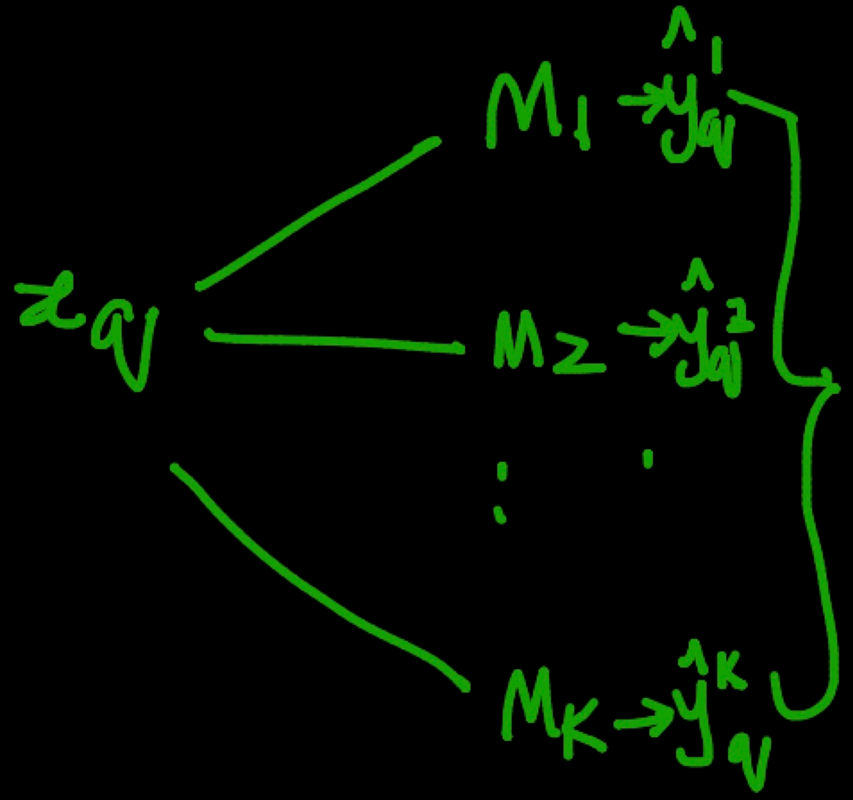
(Bootstrapping)

Train D_n → sample m pts with replacement

D_m^1 → M_1 baselearner — Tr: m, Val: n-m

D_m^2 → M_2

D_m^K → M_K



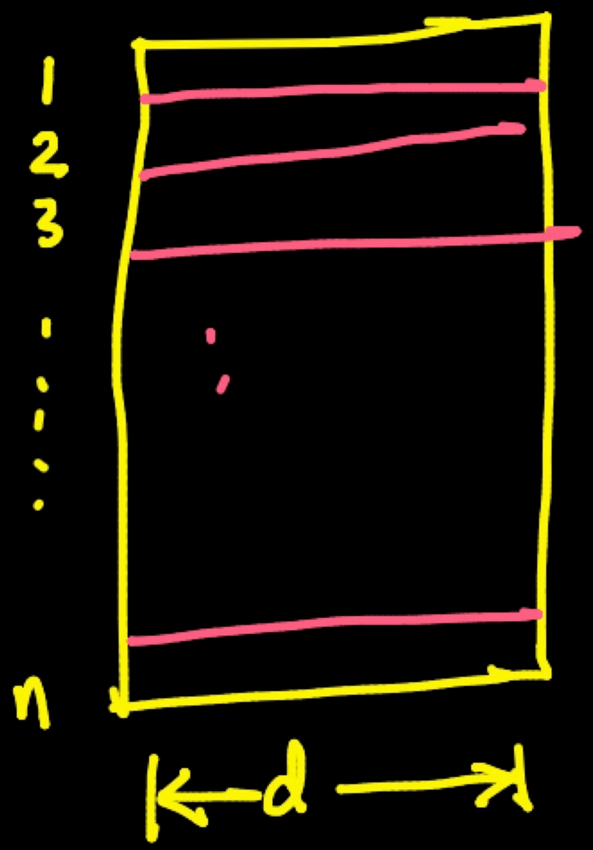
aggregation : (Classfn) Majority vote

(Reg: mean/median)

if $m << n$ then M_i's are more dissimilar

if $m \approx n$; $m < n$ M_i's tend to become more similar

$\mathcal{D}_{Tr}^{n}$

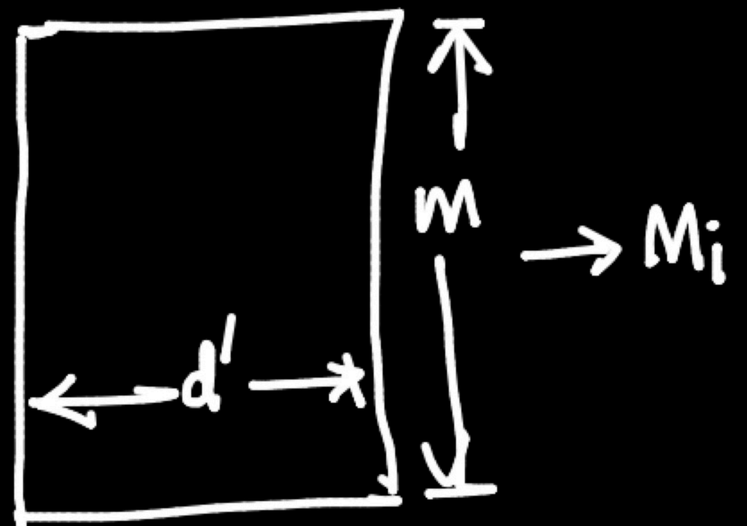


$m < n$
(m-rows)
row-sampling

Column-samp

$d' < d$

$\mathcal{D}'_m$



↳ different base-models

Random forest

- bootstrapping & Col. sampling
- Set of DTs

Tr
$\mathcal{D}_n$

r·s

c·s

Train

Single DT

$\mathcal{D}_m^1$

Same cols..

Val (OOB)

Tr
$\mathcal{D}_n - \mathcal{D}_m^1$

Model

M_1

Random Forest Training

r·s

c·s

$\mathcal{D}_m^2$ ~ $\mathcal{D}_n^{Tr} - \mathcal{D}_m^2$

M_2

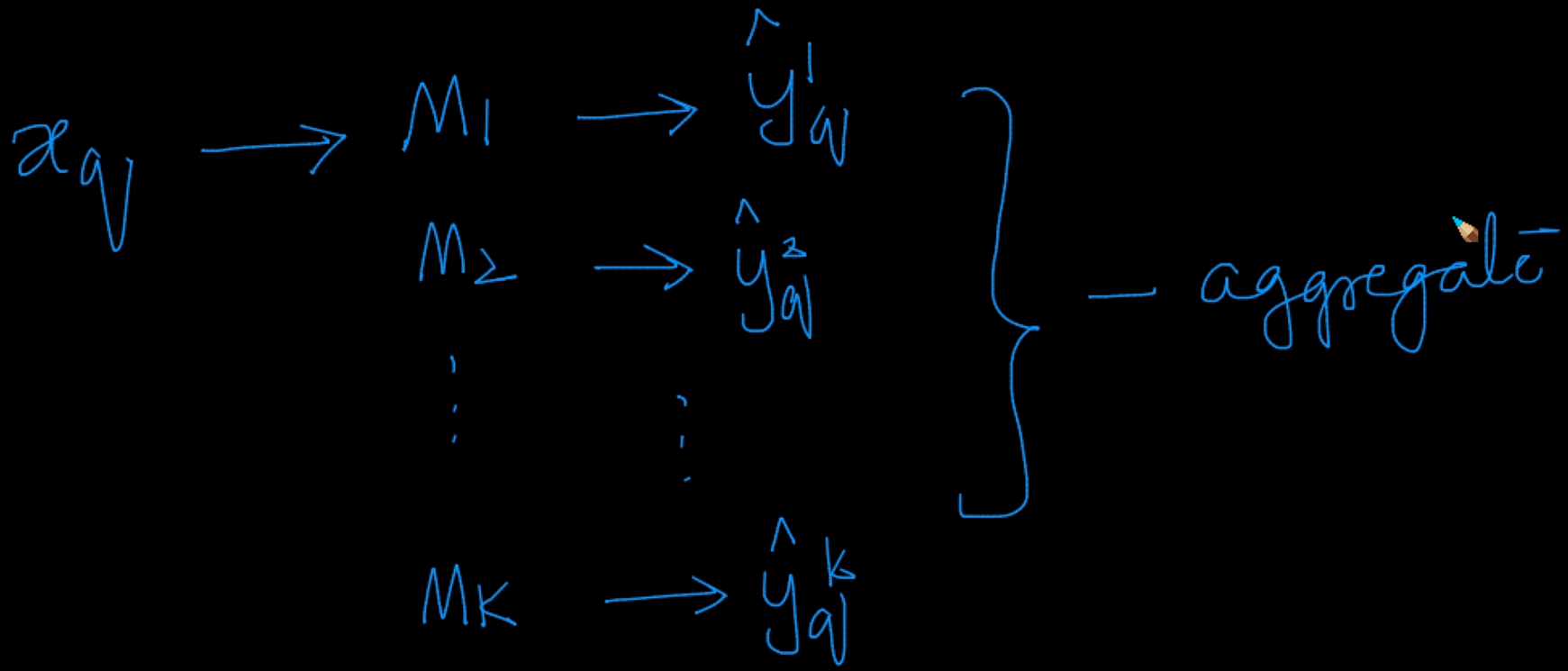
M_K

(Q) Can't we create dissimilar base-learners (DT)

using diff hyper-params

↳ can do it

Run-time



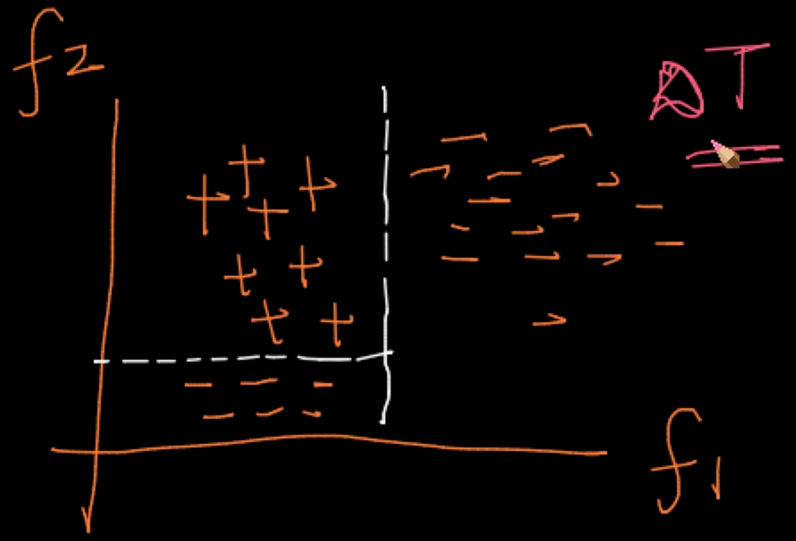
Practical

RF works better than DT (most cases)

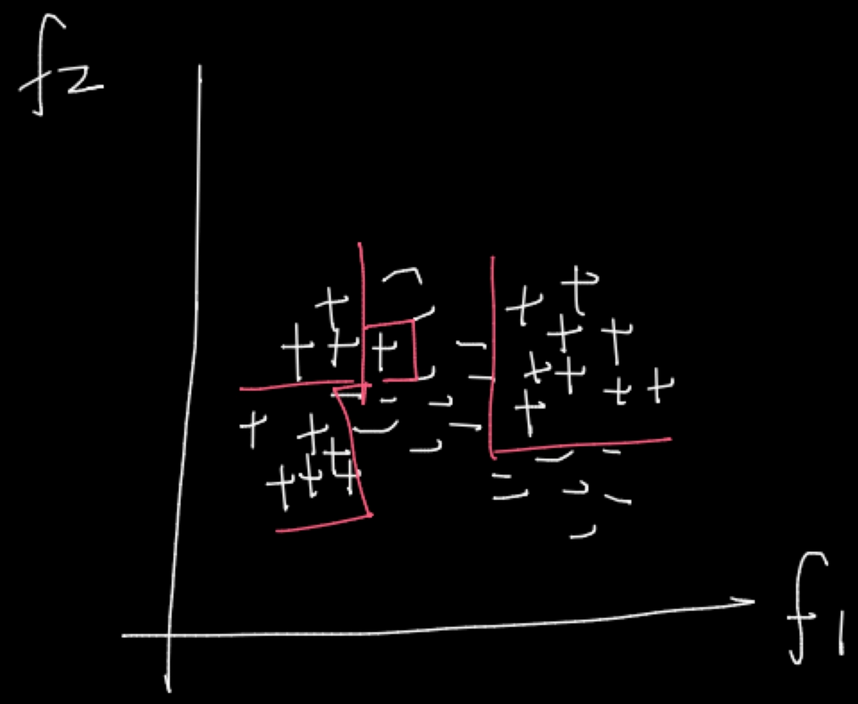
(Q) When would RF & DT have same performance?

→ n = m × Col-sampling

→ Small datasets



simple task



Complex data

Hyperparams

RF

- K = # base-learners / trees
- Col. Sampling ratio = d'/d
- Row. Sampling ratio = m/n

bias-variance tradeoff

[depth of DT → OOB-val data

$\mathcal{D}$ → $\mathcal{D}_{Tr}$ → Tr: $\mathcal{D}'_m$ & Val: $\mathcal{D}_{Tr} - \mathcal{D}'_m$ (OOB) → Base learners (M_i's)

$\mathcal{D}$ → $\mathcal{D}_{val}$ → hyperparam tune RF

$\mathcal{D}_{test}$

↳ evaluate on test data

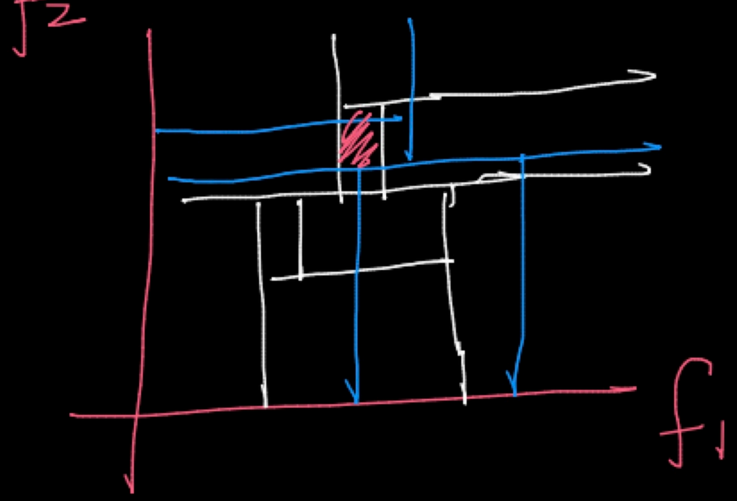
RF

each base learner is slightly deep (e.g = 7)

↓

slightly overfit

trained on a subset of rows & cols



Bagging (***)

slightly overfit base models → different from one another (random ~~zation~~)

→ aggregation

(majority vote
mean / median)

results in reduction of overfitting

e.g

100 pts mean

1 = ✓

10 — mean

(Q)

$$\text{depth} = \begin{cases} 1 \longrightarrow M1 \\ 2 \longrightarrow M2 \\ 2 \longrightarrow M3 \end{cases} \text{Underfitting}$$

$$\begin{cases} 3 \longrightarrow M4 \\ 3 \longrightarrow M4 \\ \vdots \end{cases} \text{overfitting}$$

***

randomizalim + agregalim

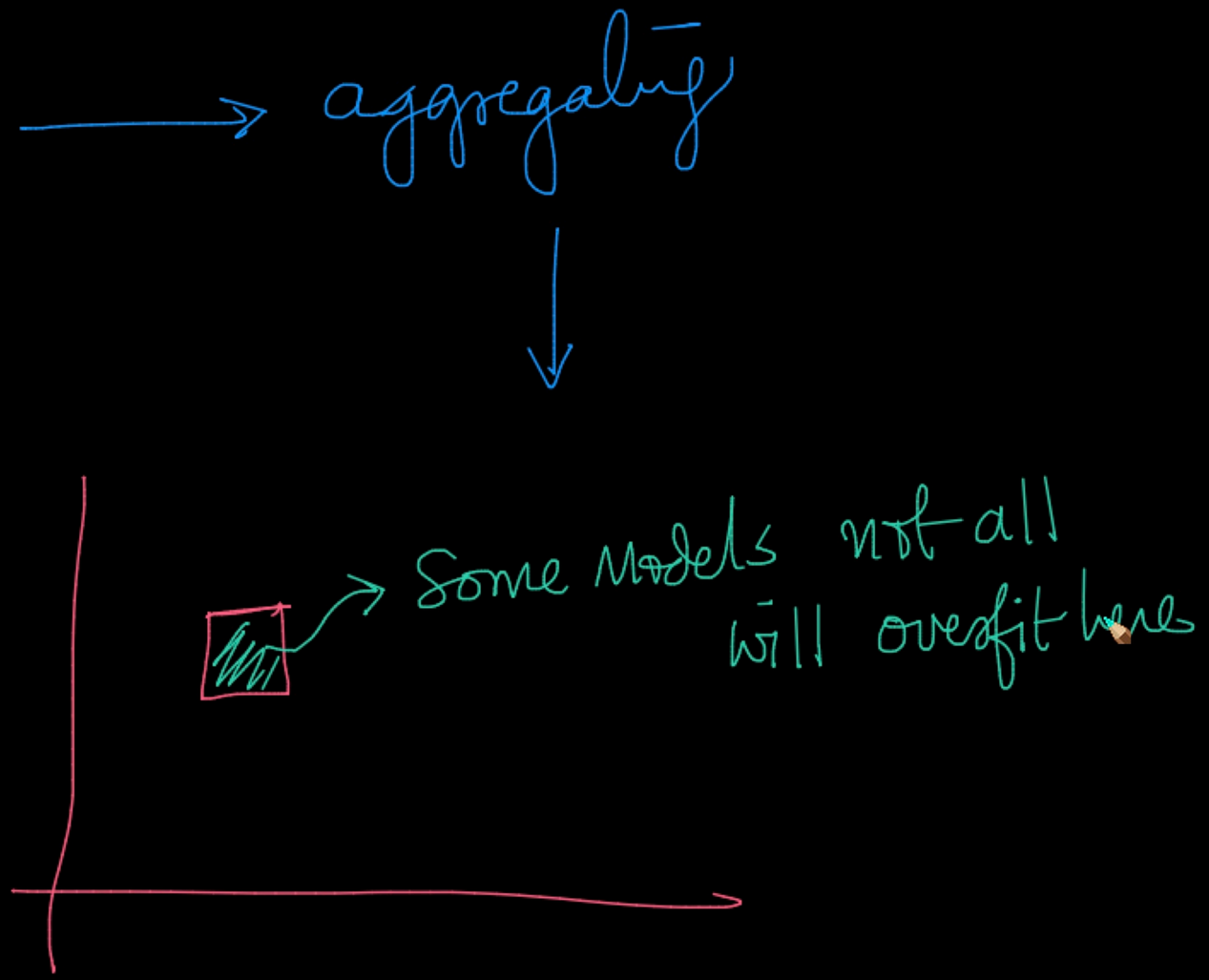
(y.s & c.s)

for

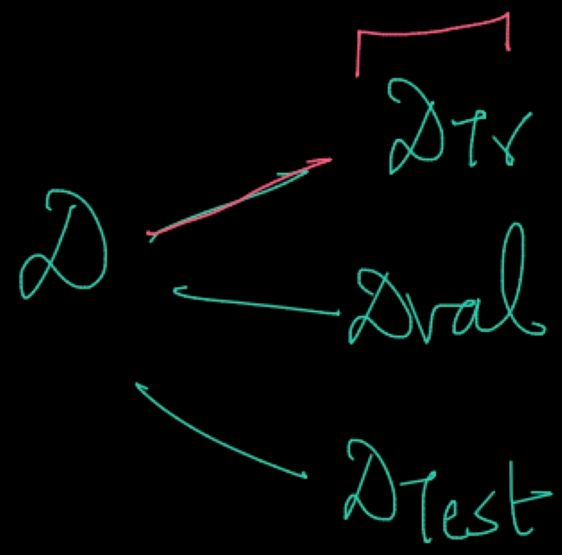
regularizalim

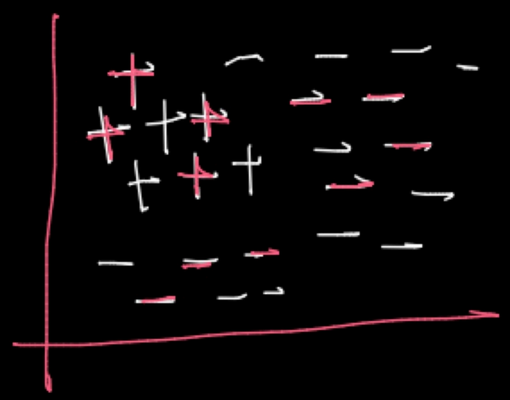
(A)

100's or 1000's slightly overfit models ⟶ aggregating



(2) No optimization





Lr- Regression

needs less data to train as

Compared to DT/RFs

RF → rigorous Math proof (Stanford Stats) early 2000's

lots of slightly overfit base learners (high-variance)

↓

aggregation : reduction in variance ⇒ less overfit

aggregation + randomization

↓

reduces overfitting

100 trials

obj: mean-height

1 random person = 155 cm → high variance (overfitting)

10 random people → avg → lower variance

Model → picks 1 person

random ; O/P height

155 cm

160 cm

175 cm

(Q) Linear / Logistic reg model as base learner

↳ Polynomial versions → slightly overfit of Linear Model

✓ DT → model interpretable

Statistical ML

(optimal Research Sessions)

Error model = $Bias^2$ + Var + irreducible error

tradeoff [var ↑ overfit
bias ↑ Underfit] tradeoff

INTUITION

variance → how much variability would exist in my model if I chose different subsets of D to train

D

or

bias → how simple the model is?

Linear Model → lowest bias

Non linear Model deep-DT high bias

RF:

Bagging: rs + CS

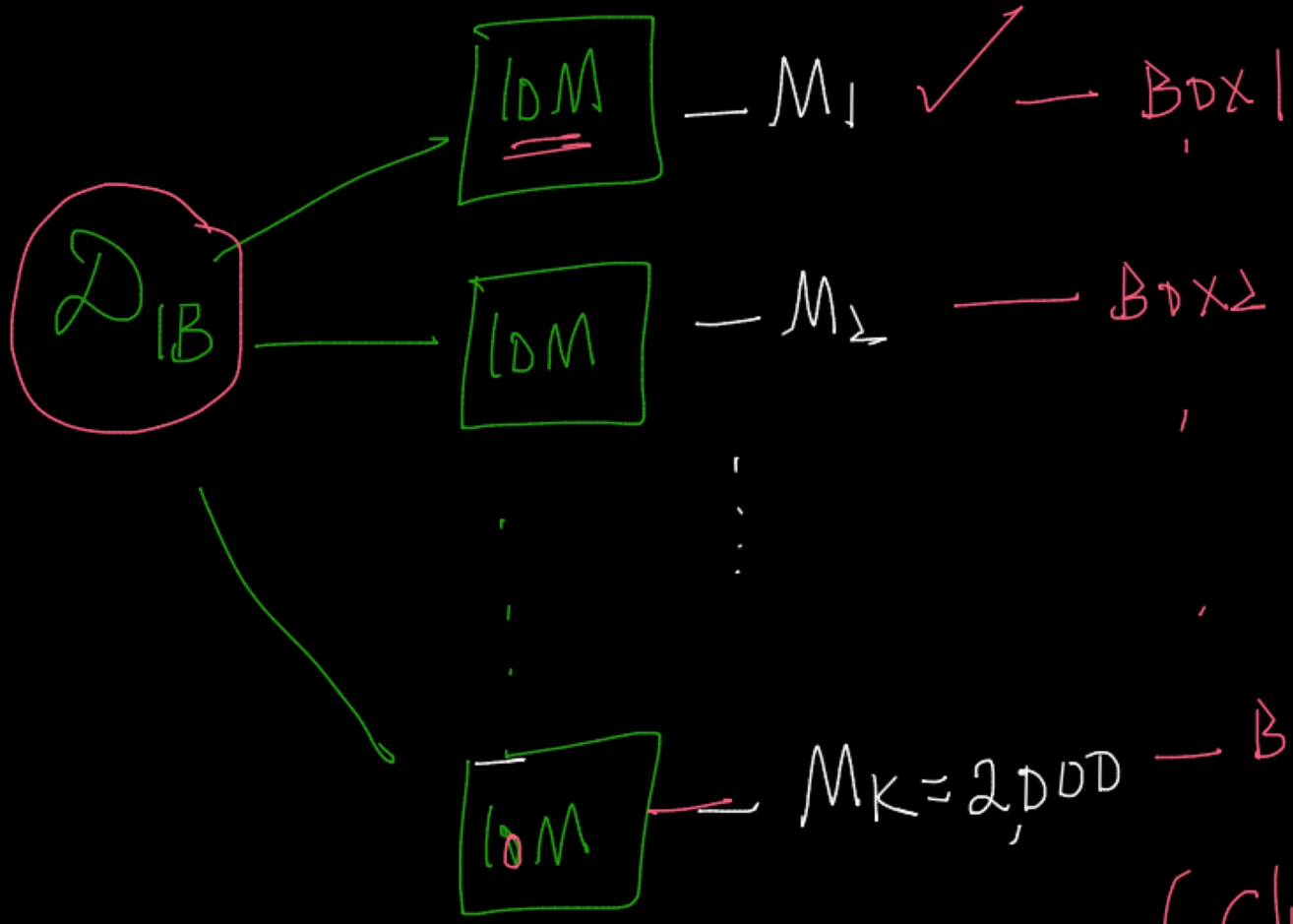
slightly overfit base-learners

aggregation to reduce variance while keeping the bias the same as base learners

MLOps (SparkML)

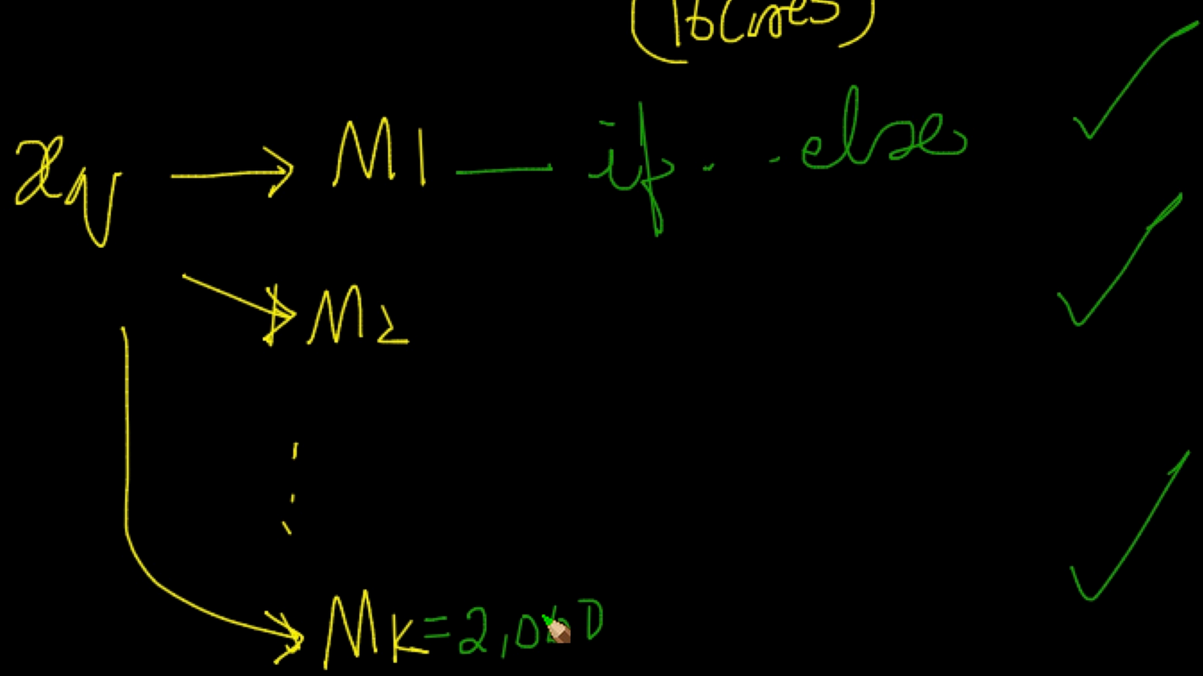
trivially parallelizable

Training



Server
Xeon
(16Cores)

Runtime



(A)

image
text
Speech
Video

} non-tabular → Deep learning

Tabular → Ensembles (RF / GBDT)

(Q) When not to use a RF?

↳ Decision Trees?

↳ too many features

Linearly Seperable

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='sqrt'*, *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***

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='sqrt', 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***

Prev Up Next

scikit-learn 1.2.1
Other versions

Please cite us if you use the software.

sklearn.ensemble.RandomFores
tClassifier
RandomForestClassifier
Examples using
sklearn.ensemble.RandomForestCl

ccp_alpha : *non-negative float, default=0.0*

Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than `ccp_alpha` will be chosen. By default, no pruning is performed. See Minimal Cost-Complexity Pruning for details.

New in version 0.22.

max_samples : *int or float, default=None*

If bootstrap is True, the number of samples to draw from X to train each base estimator.

- If None (default), then draw `X.shape[0]` samples.
- If int, then draw `max_samples` samples.
- If float, then draw `max_samples * X.shape[0]` samples. Thus, `max_samples` should be in the interval `(0.0, 1.0]`.

New in version 0.22.

Attributes:

`estimator_` : *DecisionTreeClassifier*

Estimator used to grow the ensemble.

`base_estimator_` : *DecisionTreeClassifier*

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='sqrt'*, *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***

+ Code + Text Saving failed since 20:32

RAM Disk

1s

```
print(f"K-Fold Accuracy Mean: Train: {cv_acc_results['train_score'].mean()*100} Validation: {cv_acc_r
print(f"K-Fold Accuracy Std: Train: {cv_acc_results['train_score'].std()*100} Validation: {cv_acc_r
```

```
K-Fold Accuracy Mean: Train: 93.24203753064435 Validation: 84.20945945945945
K-Fold Accuracy Std: Train: 0.8887486809601636 Validation: 8.453236055736403
```

5s

```
[9] from sklearn.ensemble import RandomForestClassifier
    from sklearn.model_selection import KFold, cross_validate

    tree_clf = RandomForestClassifier(random_state=7, max_depth=7, n_estimators=100)
    kfold = KFold(n_splits=10)
    cv_acc_results = cross_validate(tree_clf, X_sm, y_sm, cv = kfold, scoring = 'accuracy', return_trai

    print(f"K-Fold Accuracy Mean: Train: {cv_acc_results['train_score'].mean()*100} Validation: {cv_acc
    print(f"K-Fold Accuracy Std: Train: {cv_acc_results['train_score'].std()*100} Validation: {cv_acc_r
```

```
K-Fold Accuracy Mean: Train: 96.09794552823907 Validation: 89.78025851938895
K-Fold Accuracy Std: Train: 0.6293464780221507 Validation: 6.684410725948877
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

+ Code + Text Saving failed since 20:32

RAM Disk

[10] 1s

```
print(f"K-Fold Accuracy Mean: Train: {cv_acc_results['train_score'].mean()*100} Validation: {cv_acc
print(f"K-Fold Accuracy Std: Train: {cv_acc_results['train_score'].std()*100} Validation: {cv_acc_r
```

```
K-Fold Accuracy Mean: Train: 93.24203753064435 Validation: 84.20945945945945
K-Fold Accuracy Std: Train: 0.8887486809601636 Validation: 8.453236055736403
```

41s

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold, cross_validate

tree_clf = RandomForestClassifier(random_state=7, max_depth=7, n_estimators=1000)
kfold = KFold(n_splits=10)
cv_acc_results = cross_validate(tree_clf, X_sm, y_sm, cv = kfold, scoring = 'accuracy', return_trai

print(f"K-Fold Accuracy Mean: Train: {cv_acc_results['train_score'].mean()*100} Validation: {cv_acc
print(f"K-Fold Accuracy Std: Train: {cv_acc_results['train_score'].std()*100} Validation: {cv_acc_r
```

```
K-Fold Accuracy Mean: Train: 96.23023656852769 Validation: 90.37573443008226
K-Fold Accuracy Std: Train: 0.5861570220746954 Validation: 6.764979471421996
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

39 / 39

metric

K-fold CV

{ Val-acc-mean ↑
  Val-acc-Var ↓

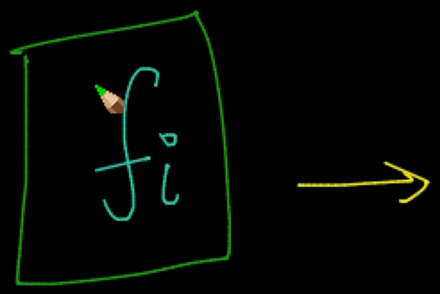
① [mean - 2√var = mean + 2√var]

② F1 Score: Pr↑ Re↑

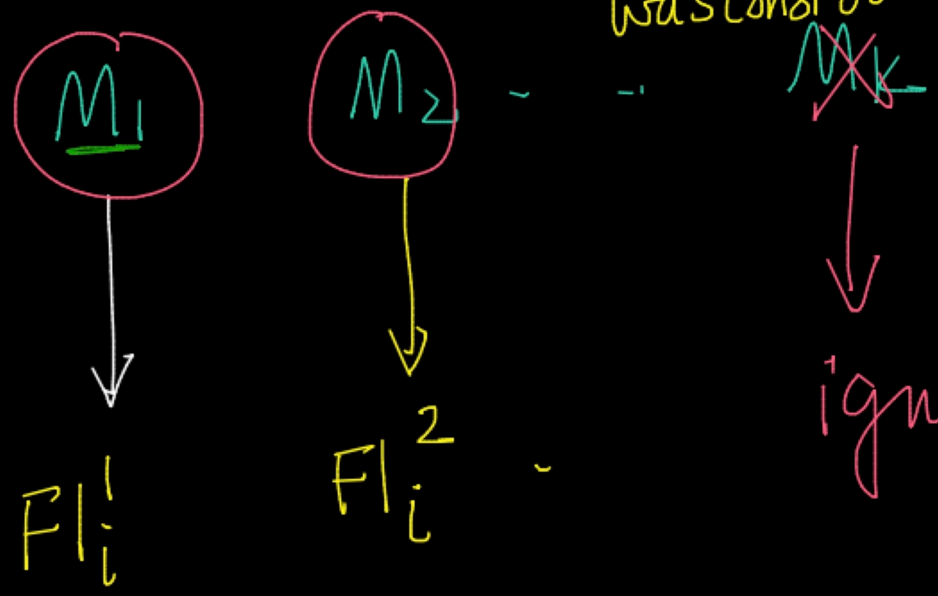
mean ↑
1/Var ↑ → F1 Score

Col. samp = 0.1

Feature Imp in RF



take models where fi was considered



- → mean ✓
- → weighted mean: Weight = OOB performance
- → medians

```
print('K-Fold Accuracy Std: Train: {cv_acc_results[ 'train_score' ].std()*100} Validation: {cv_acc_results[ 'test_score' ].std()*100} ')
```

[12] 41s

```
K-Fold Accuracy Mean: Train: 96.23023656852769 Validation: 90.37573443008226
K-Fold Accuracy Std: Train: 0.5861570220746954 Validation: 6.764979471421996
```

```
Gri# Feature importance
import numpy as np
import matplotlib.pyplot as plt

clf = RandomForestClassifier(random_state=7, max_depth=4, n_estimators=100)
clf.fit(X_sm, y_sm)
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1] # Sort feature importances in descending order
names = [X_sm.columns[i] for i in indices] # Rearrange feature names so they match the sorted feature importances
plt.figure(figsize=(15, 7)) # Create plot
plt.title("Feature Importance") # Create plot title
plt.bar(range(X_sm.shape[1]), importances[indices]) # Add bars
plt.xticks(range(X_sm.shape[1]), names, rotation=90) # Add feature names as x-axis labels
plt.show() # Show plot
```

hyper-params:

① # baselearners = K ↑ — Overfitting ↓

② m = 20% samples ↑ — Overfitting ↑

m ↑ ⇒ base models are more similar

(3) d' = # cols sampled ↑ ⇒ overfit ↑

(4) depth of base learners ↑ ⇒ overfit ↑

(not using OOB CV)

DT → not a traditional ML optmzn problem

$\min(\text{loss} + \alpha\ \text{reg})$

regul zn

MSE/log

min (loss + α) #leaves in my tree

d = 1, 2, 3, 4, ... to

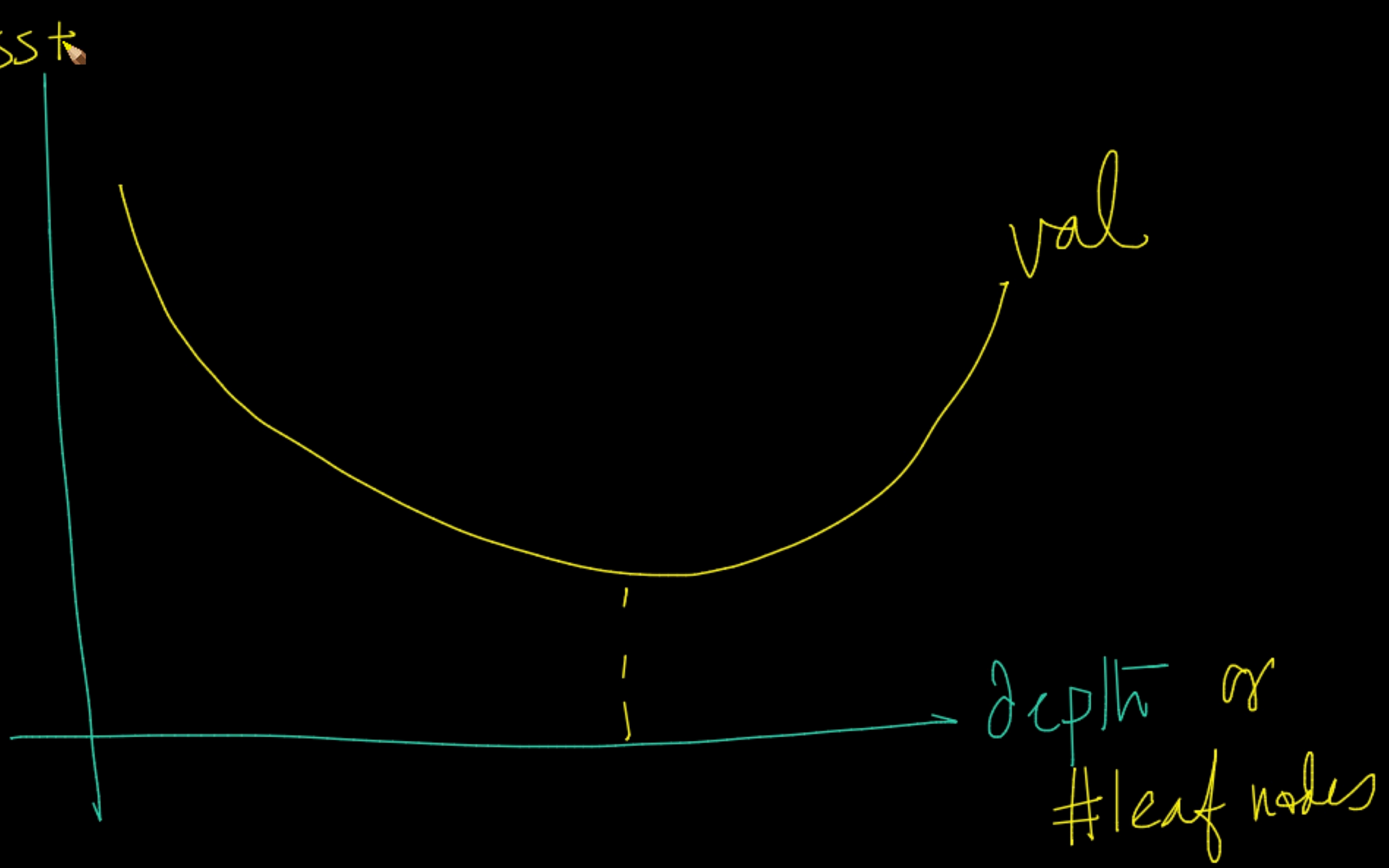
Changing depth

depth ↑

ND SGD

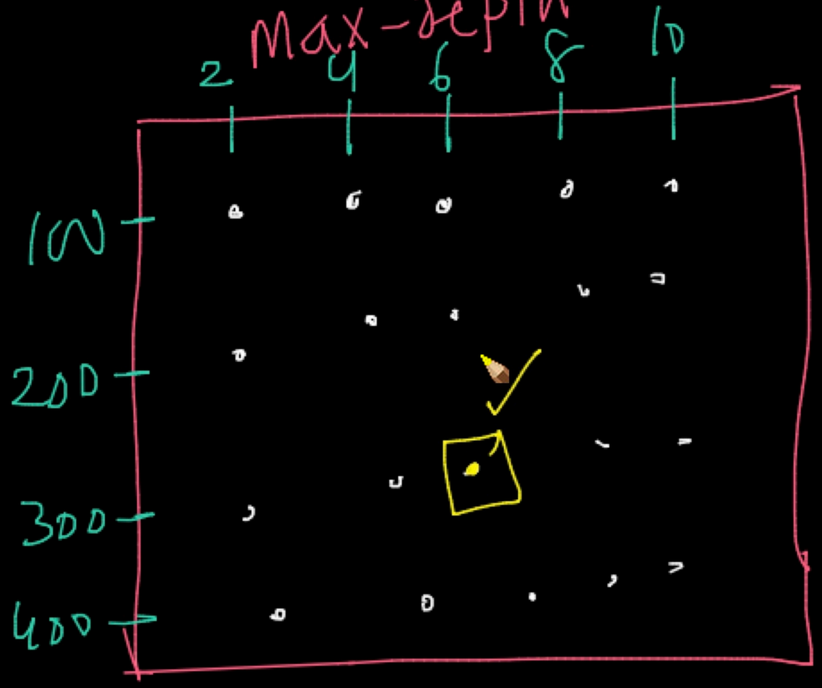
depth ↑ ⇒ #leaves ↑

⇒ (loss + α #leaves) ↑



Grid Search:

→ NOT using OOB-val



#base learners

#BL MaxDepth

M1 = (100, 2)
(100, 4)
(100, 6)

(brute force)

[14] 0s


```
# Defining Parametes
params = {
          'n_estimators' : [100,200,300,400],
          'max_depth' : [3,5,10],
          'criterion' : ['gini', 'entropy'],
          'bootstrap' : [True, False],
          'max_features' : [8,9,10]
         }
```

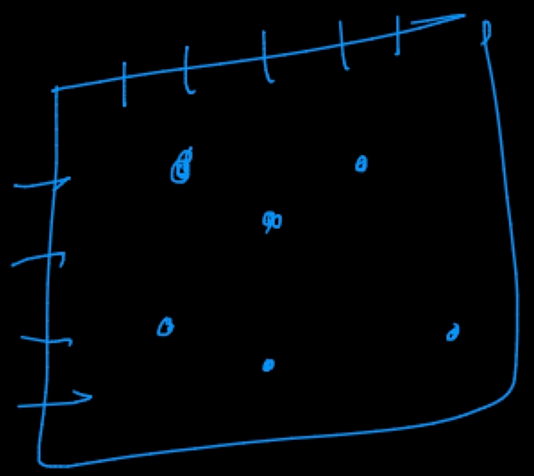
Chrome File Edit View History Bookmarks Profiles Tab Window Help 11:00

Grid Search — brute force (try all combinations)

X Binary Search → sorted

Randomized Search

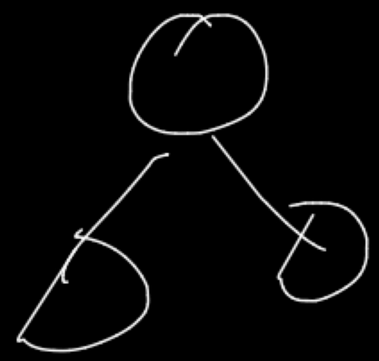
[random combinations]



50 / 50

Underfitting

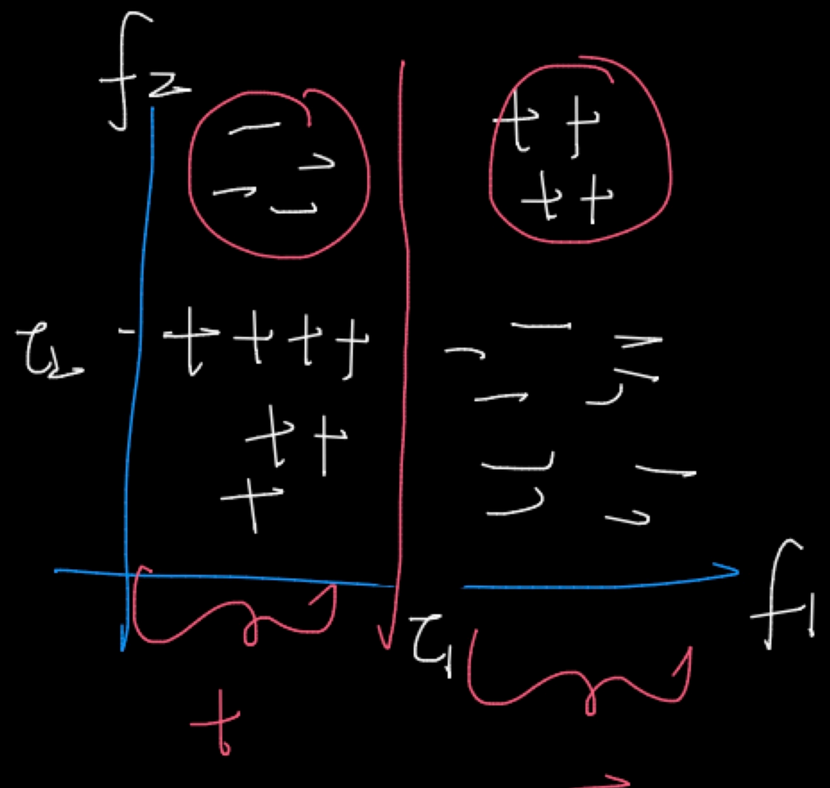
depth = 1



Simple & limited

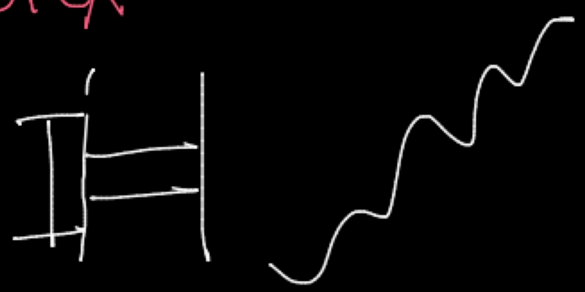
↓

T_{train} - error is high



INTUITION

- Simpler model → high bias → Underfit model
- Complex model → low bias



…value of the parameter being estimated. An estimator or decision rule with zero bias is called *unbiased*. In statistics, …s" is an *objective* property of an estimator. Bias is a distinct concept from consistency: consistent estimators converge in probability to the true value of the parameter, but may be biased or unbiased; see bias versus consistency for more.

All else being equal, an unbiased estimator is preferable to a biased estimator, although in practice, biased estimators (with generally small bias) are frequently used. When a biased estimator is used, bounds of the bias are calculated. A biased estimator may be used for various reasons: because an unbiased estimator does not exist without further assumptions about a population; because an estimator is difficult to compute (as in unbiased estimation of standard deviation); because a biased estimator may be unbiased with respect to different measures of central tendency; because a biased estimator gives a lower value of some loss function (particularly mean squared error) compared with unbiased estimators (notably in shrinkage estimators); or because in some cases being unbiased is too strong a condition, and the only unbiased estimators are not useful.

Bias can also be measured with respect to the median, rather than the mean (expected value), in which case one distinguishes *median*-unbiased from the usual *mean*-unbiasedness property. Mean-unbiasedness is not preserved under non-linear transformations, though median-unbiasedness is (see § Effect of transformations); for example, the sample variance is a biased estimator for the population variance. These are all illustrated below.

Definition [edit]

Suppose we have a statistical model, parameterized by a real number θ, giving rise to a probability distribution for observed data, $P_\theta(x) = P(x \mid \theta)$, and a statistic $\hat{\theta}$ which serves as an estimator of θ based on any observed data x. That is, we assume that our da… that is part of this

$O(K \cdot \overset{\max}{depth})$ — runtime Cmplx

$O(K * \text{timeCmpl for medT})$ — Traintime Cmplx

an squared error

$$e^{-4\lambda} - 2e^{\lambda(1/e^2-3)} + e^{\lambda(1/e^4-1)}$$

is smaller; compare the unbiased estimator's MSE of

$$1 - e^{-4\lambda}.$$

The MSEs are functions of the true value λ. The bias of the maximum-likelihood estimator is:

$$e^{-2\lambda} - e^{\lambda(1/e^2-1)}.$$

Maximum of a discrete uniform distribution [edit]

Main article: Maximum of a discrete uniform distribution

The bias of maximum-likelihood estimators can be substantial. Consider a case where *n* tickets numbered from 1 through to *n* are placed in a box and one is selected at random, giving a value *X*. If *n* is unknown, then the maximum-likelihood estimator of *n* is *X*, even though the expectation of *X* given *n* is only (*n* + 1)/2; we can be certain only that *n* is at least *X* and is probably more. In this case, the natural unbiased estimator is 2*X* – 1.

Median-unbiased estimators [edit]

The theory of median-unbiased estimators was revived by George W. Brown in 1947:[6]

An estimate 

the median of