

week 5

overview

- every classifier = a rule that turns a row's features into a class
- learning = pick the rule from a family of rules that makes the fewest mistakes on train
- the real goal = few mistakes on rows we have not seen →

generalization

- three choices define any model → model shape and loss and optimiser
- this week
 - a family of rules = boxes then trees
 - how to score a rule = confusion matrix and metrics
 - how a tree picks its rule = impurity → entropy or gini
 - how to stop it memorising = pruning

classification

- supervised task that predicts a class label for a row
 - malignant or benign → classification cuz the answer is a label
- $X = \{ (x_t, r_t) \}$ for $t = 1 \dots N$
 - $r = 1$ if positive sample
 - $r = 0$ if negative sample
- $x = x_1 x_2 \dots x_n$ = input variables = features = the columns
 - breast cancer table → radius mean and texture mean ... compactness mean
- $r = y$ = target = output variable
- N = number of rows and t = which row
- positive = the class we're looking for and not the good one
 - malignant = positive and benign = negative
 - spam = positive and normal mail = negative
- flow → training set → learning algorithm → hypothesis $h(x)$ → $h(x)$ maps a new row's features to a target
- types
 - binary → 2 classes → malignant or benign → metrics use one positive class
 - multiclass → more than 2 classes and one per row → Iris setosa or versicolor or virginica → metrics per class then averaged
 - multilabel → a row can have several labels → a film is action and comedy → not on the slides

hypothesis

- hypothesis h = one guessed rule that splits positive from negative
- H = hypothesis class = every rule of that shape we could pick
- learning = search H for the best h
- slide example $\rightarrow$ texture mean x_2 and compactness mean x_5
 $(te_1 \leq \text{texture} \leq te_2) \text{ AND } (co_1 \leq \text{compactness} \leq co_2)$
 $h(x) = 1$ if x is inside the box
 $h(x) = 0$ if x is outside
- H here = every axis aligned rectangle $\rightarrow$ 4 numbers te_1, te_2, co_1, co_2
 pick one h



- error of h = number of training rows it gets wrong
 $E(h(x)) = \sum 1(h(x_t) \neq r_t)$
- $1(\dots) = 1$ when the statement is true and 0 when false
 - 10 rows and the box misplaces 2 $\rightarrow E = 2$
 error rate = E / N
 accuracy = $1 - E / N$
- why the choice of H matters
 - choosing H = an assumption $\rightarrow$ we bet malignant cells sit inside a range of both features
 truth is a box $\rightarrow$ enough data finds it
 - truth is a circle $\rightarrow$ no box fits $\rightarrow$ some error stays no matter how much data
 - bigger $H \rightarrow$ fits more patterns but also fits noise $\rightarrow$ overfit risk
- the box as a tree $\rightarrow$ 4 tests in a row $\rightarrow$ a depth 4 decision tree
 $\text{texture} \geq te_1 \rightarrow \text{texture} \leq te_2 \rightarrow \text{compact} \geq co_1 \rightarrow \text{compact} \leq co_2$
 $\rightarrow$ positive
 - so every leaf of a tree is one box in feature space

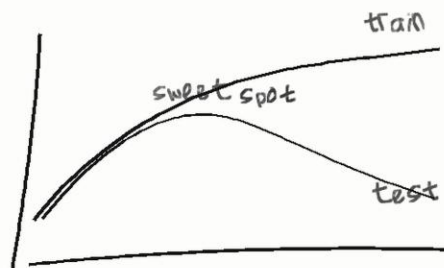
errors in the hypothesis

- C = the true class region and h = our box
- false positive = actually negative but h says positive
 - benign patient told malignant $\rightarrow$ extra tests and stress
- false negative = actually positive but h says negative
 - malignant patient sent home $\rightarrow$ the worse mistake here
- TP and TN = right
- S = most specific h = tightest box around all the + points
- G = most general h = biggest box that still has no - inside
- every box between S and G has zero training error
- margin = gap between S and G

- pick the h halfway $\rightarrow$ a new point that lands a little off still gets the right side
 - like building a road in the middle of a gap and not on the cliff edge $\rightarrow$ analogy
- Slide correction
 - slide says $\rightarrow FP = C \downarrow$ but $h \downarrow$ and $FN = C \downarrow$ but $h \downarrow$
 - correct $\rightarrow FP = C \downarrow$ and $h \downarrow \rightarrow FN = C \downarrow$ and $h \downarrow \rightarrow$ a false positive is a negative case we called positive
 - in the exam $\rightarrow$ use the words actually negative predicted positive and add that the slide swaps C and h

generalization and overfitting

- generalization = doing well on rows the model never saw
- overfitting = fits train including the noise $\rightarrow$ worse on new rows
- underfitting = too simple for the real pattern $\rightarrow$ bad everywhere
- why it happens
 - train error always falls as the model gets more complex $\rightarrow$ more freedom to bend around each row
 - test error falls then rises $\rightarrow$ once the real pattern is caught the extra freedom only fits noise
 - noise is different in new data $\rightarrow$ fitting it is pure loss



- expected error on new data has three parts

$$\text{error} = \text{bias}^2 + \text{variance} + \text{noise}$$
- bias = how far the average model is from the truth $\rightarrow$ too simple = high bias
- variance = how much the model changes on another sample $\rightarrow$ too complex = high variance
- noise = randomness in the labels $\rightarrow$ no model removes it
 - one deep tree $\rightarrow$ bias low and variance high
 - one split stump $\rightarrow$ bias high and variance low
 - pruning $\rightarrow$ less variance and more bias
- how to spot it

train	test	means
1.00	0.72	overfit
0.60	0.58	underfit
0.86	0.83	good

- slide factors $\rightarrow$ complexity of the model and amount of training data and generalization error on new data
- more data with variety $\rightarrow$ harder to memorise $\rightarrow$ the gap closes
- duplicated rows add no information $\rightarrow$ they only count the same row twice

test set and leakage

- train error is too optimistic $\rightarrow$ θ was chosen to fit exactly those rows
- only rows the model never touched show how it does on new rows
- the test set must look like the future $\rightarrow$ else the score lies
- data leakage = information from test rows reaching training $\rightarrow$ test score too high $\rightarrow$ worse when deployed
 - scaler fit on all rows before the split $\rightarrow$ test min and max leak in
 - a feature made from the target $\rightarrow$ days in hospital to predict admission
 - the same patient in train and test
- rule $\rightarrow$ split first $\rightarrow$ fit everything on train $\rightarrow$ use test once at the end
- baseline = score of always guessing the biggest class $\rightarrow$ the model must beat it
 - Orthopedic $\rightarrow 150 / 310 = 0.48 \rightarrow$ the tree at 0.72 is a real gain

three decisions for any model

- data $X = \{x_i, y_i\}$ is iid = independent and identically distributed
 - independent $\rightarrow$ one row does not affect another
 - identically distributed $\rightarrow$ train rows and future rows come from the same population
 - the only reason old rows say anything about new ones
- model $g(x; \theta) \rightarrow$ shape of the rule and $\theta =$ its parameters
- loss $L(r, g) \rightarrow$ cost of one prediction

$$E_{\theta}(X) = \sum L(r_i, g(x_i, \theta))$$
- optimiser $\rightarrow$ method that finds the best θ

$$\theta^* = \operatorname{argmin}_{\theta} \text{ over } \theta \text{ of } E_{\theta}(X)$$
- argmin = the θ that gives the smallest E and not the smallest E itself

	model	loss	optimiser
box	rectangle	0 or 1	pick between S and G
tree	tree of tests	impurity	greedy splits

- any new model $\rightarrow$ answer the three questions $\rightarrow$ then you know how it learns

confusion matrix

- counts of predicted vs actual
- slide layout $\rightarrow$ rows = predicted and columns = actual

	actual 1	actual 0
pred 1	TP	FP
pred 0	FN	TN

- sklearn `ConfusionMatrix` puts actual on rows and predicted on columns $\rightarrow$ the opposite way $\rightarrow$ check before reading off FP and FN
- name rule $\rightarrow$ second letter = what we predicted and first letter = right or wrong
 - FN = false + negative $\rightarrow$ we said negative and were wrong $\rightarrow$ actually positive
- example $\rightarrow$ 100 patients and 10 have malignant cancer
 - model flags 8 and 6 of them really are
 - TP 6 and FP 2 and FN 4 and TN 88
 - Check $\rightarrow$ TP + FN = 10 actual positives and FP + TN = 90 actual negatives
- card fraud $\rightarrow$ positive = fraud
 - TP $\rightarrow$ fraud and flagged $\rightarrow$ money saved
 - TN $\rightarrow$ normal and passed $\rightarrow$ customer happy
 - FP $\rightarrow$ normal but blocked $\rightarrow$ customer annoyed
 - FN $\rightarrow$ fraud but passed $\rightarrow$ money lost
- every metric below = one count divided by a row or a column
- = a conditional probability $P(A|B)$ = share of the B rows that are also A

accuracy

- asks $\rightarrow$ of all predictions what share is right
- $$\text{accuracy} = (TP + TN) / \text{total} = P(\text{predicted} = \text{actual})$$
$$= (6 + 88) / 100 = 0.94$$
- fails when one class is rare
 - always say benign $\rightarrow (0 + 90) / 100 = 0.90$ and finds zero cancers
- so always compare with the baseline = share of the biggest class

precision

- asks $\rightarrow$ when I say positive how often am I right
- $$\text{precision} = TP / (TP + FP) = P(\text{actual} + | \text{predicted} +)$$
$$= 6 / 8 = 0.75$$
- denominator = everything I flagged
- 0.75 $\rightarrow$ 1 in 4 alarms is false
- matters when a false alarm is costly

- spam filter $\rightarrow$ FP = a real mail in the spam folder $\rightarrow$ a missed bill or job offer is worse than one spam in the inbox
- court verdict $\rightarrow$ FP = an innocent person jailed

recall or sensitivity

- asks $\rightarrow$ of the real positives how many did I catch

$$\text{recall} = TP / (TP + FN) = R_{\text{predicted}} + 1 \text{ actual} +$$

$$= 6 / 10 = 0.60$$
- denominator = every real positive
- 0.60 $\rightarrow$ 4 of 10 cancers missed
- matters when a miss is costly $\rightarrow$ cancer screening or fraud
- precision and recall pull against each other
 - flag everyone $\rightarrow$ recall 1.0 and precision = base rate 0.10
 - flag only the surest one $\rightarrow$ precision maybe 1.0 and recall 0.1
 - so neither alone means anything $\rightarrow$ look at both or at F1

specificity

- asks $\rightarrow$ of the real negatives how many did I clear

$$\text{specificity} = TN / (TN + FP) = R_{\text{predicted}} - 1 \text{ actual} -$$

$$= 88 / 90 = 0.98$$
- = recall of the negative class

$$\text{false positive rate} = FP / (FP + TN) = 1 - \text{specificity}$$
- Slide Correction
 - slide says $\rightarrow$ specificity = $TN / (TN + FN)$
 - correct $\rightarrow$ $TN / (TN + FP) \rightarrow$ the bottom must be all actual negatives and FN rows are actual positives $\rightarrow$ 88 / 90 not 88 / 92
 - in the exam $\rightarrow$ write $TN / (TN + FP)$ and note the slide version mixes in FN

F1

- asks $\rightarrow$ one number that is high only if precision and recall are both high

$$F1 = 2 \times P \times R / (P + R)$$

$$= 2 \times 0.75 \times 0.60 / 1.35 = 0.67$$
- harmonic mean = 1 over the average of 1 over each

$$F1 = 2 / (1/P + 1/R)$$

$$F1 = 2TP / (2TP + FP + FN)$$

$$= 12 / (12 + 2 + 4) = 0.67$$
- TN never appears $\rightarrow$ F1 ignores the easy negatives $\rightarrow$ works for rare classes
- why harmonic $\rightarrow$ $1/P$ blows up when P is small so the smaller one dominates

- P 1.0 and R 0.1 $\rightarrow$ normal mean 0.55 but F1 0.18
- 0 to 1 and higher is better
- Slide Correction
 - slide says $\rightarrow$ the higher F1 the higher accuracy
 - correct $\rightarrow$ F1 and accuracy are different numbers $\rightarrow$ always
 - benign model $\rightarrow$ accuracy 0.90 and F1 0
 - in the exam $\rightarrow$ say higher F1 = better balance of precision and recall and not higher accuracy

multiclass metrics

- K classes $\rightarrow$ each class takes a turn as positive vs all the rest $\rightarrow$ then average
- lab Iris matrix in sklearn layout $\rightarrow$ rows actual and columns predicted

actual	set	ver	vir
setosa	19	0	0
versicolor	0	12	0
virginica	0	1	13

accuracy = diagonal / total = 44 / 45 = 0.98

- sklearn layout $\rightarrow$ precision = diagonal / its column and recall = diagonal / its row
 - versicolor P = 12 / 13 = 0.92 and R = 12 / 12 = 1.00
 - virginica P = 13 / 13 = 1.00 and R = 13 / 14 = 0.93

macro = $(m_1 + m_2 + \dots + m_n) / K$

weighted = $\sum (\text{support of class} / N) \times m$

- support = actual rows of that class in the test set $\rightarrow$ 19 12 14
- macro P = $(1.00 + 0.92 + 1.00) / 3 = 0.97$
- macro $\rightarrow$ every class counts the same $\rightarrow$ shows a failing small class
- weighted $\rightarrow$ big classes count more $\rightarrow$ sits near accuracy
- macro far below weighted $\rightarrow$ a small class is bad
 - Orthopedic $\rightarrow$ Hernia recall 0.33 $\rightarrow$ macro recall 0.66 vs weighted 0.72

micro $\rightarrow$ add TP and FP and FN over all classes first $\rightarrow$ compute once

- Iris $\rightarrow$ TP 44 and FP 1 and FN 1

micro P = $44 / (44 + 1) = 0.98$

- single label multiclass $\rightarrow$ each wrong row = one FP for the class it got and one FN for its real class $\rightarrow$ micro P = micro R = micro F1 = accuracy

micro $\rightarrow$ every row counts the same and macro $\rightarrow$ every class counts the same

- binary $\rightarrow$ report the positive class itself and not an average

log loss

- asks $\rightarrow$ how much probability did the model put on the right answer

$$\log \text{loss} = -(1/n) \sum [y \log(\hat{y}) + (1-y) \log(1-\hat{y})]$$

$\hat{y}$ = predicted probability of class 1 and $\log$ = natural log

- one term is alive per row

$$y = 1 \rightarrow \text{cost} = -\log(\hat{y})$$

$$y = 0 \rightarrow \text{cost} = -\log(1-\hat{y})$$

- where it comes from

- the model says $P(y = 1) = \hat{y} \rightarrow$ so the probability it gave to what really happened is

$$P(y) = \hat{y}^y \times (1-\hat{y})^{(1-y)}$$

rows are iid $\rightarrow$ chance of the whole dataset = product over rows = likelihood

- best model = the one that makes the real labels most likely

- take the log $\rightarrow$ product becomes sum $\rightarrow$ minimise the negative mean $\rightarrow$ log loss

y	$\hat{y}$	cost
1	0.9	0.11
1	0.5	0.69
1	0.1	2.30
1	0.01	4.61

- 3 rows $y = 1 \ 0 \ 1$ and $\hat{y} = 0.9 \ 0.2 \ 0.6$

$$\text{likelihood} = 0.9 \times 0.8 \times 0.6 = 0.432$$

$$\log \text{loss} = -\log(0.432) / 3 = 0.28$$

- 0 = perfect and no top $\rightarrow$ lower is better

- 0.69 = a coin flip on 2 classes $\rightarrow$ beat that

- confident and wrong costs the most

- an unpruned tree gives pure leaves $\rightarrow \hat{y} = 1.0$ or $0.0 \rightarrow$ one wrong row costs $\log(0) = \infty \rightarrow$ sklearn clips $\hat{y}$ a bit away from 0

- label based metrics $\rightarrow$ accuracy precision recall F1 $\rightarrow$ use predict

- probability based $\rightarrow$ log loss $\rightarrow$ uses predict_proba $\rightarrow$ two models with the same accuracy can differ a lot here

- Slide Correction

$$\text{slide says} \rightarrow \log \text{loss} = -(1/n) \sum (y \log(\hat{y}) + (1-y) \log(1-\hat{y}))$$

- correct $\rightarrow$ the second term is $(1-y) \log(1-\hat{y}) \rightarrow$ without the log it is not a likelihood

- in the exam $\rightarrow$ write the version with both logs

- Slide correction

$$\text{slide says} \rightarrow 0 \leq \log \text{loss} \leq 1$$

- correct $\rightarrow 0$ to $\infty \rightarrow y = 1$ and $\hat{y} = 0.01$ all costs 4.61

- in the exam $\rightarrow$ say lower is better with 0 as perfect and no upper limit

entropy

- entropy = how mixed a group is = average surprise of its labels

- surprise of an event with probability p

$$\text{surprise} = -\log_2(p)$$

- why log $\rightarrow$ independent events multiply their p but surprise should add

$\rightarrow$ log turns product into sum

- why base 2 $\rightarrow$ a fair coin flip = exactly 1 bit

$$p=1 \rightarrow 0 \text{ bits}$$

$$p=1/8 \rightarrow 3 \text{ bits}$$

- entropy = expected surprise

$$\text{Info}(D) = -\sum p \log_2(p) \text{ over each class}$$

$$4 \text{ buy} \rightarrow -\log_2(4/7) = 0.81 \text{ bits each}$$

$$3 \text{ no} \rightarrow -\log_2(3/7) = 1.22 \text{ bits each}$$

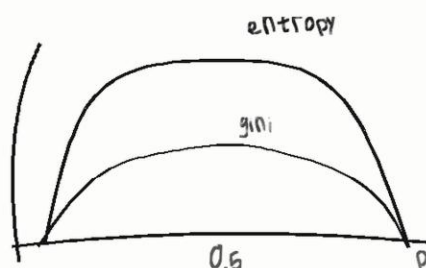
$$\text{Info}(D) = (4/7)(0.81) + (3/7)(1.22) = 0.985$$

- 0 = pure

- max = $\log_2(K)$ at equal shares $\rightarrow$ 2 classes 1.0 and 3 classes 1.585

- why max at equal shares $\rightarrow$ no class is easier to guess $\rightarrow$ every outcome equally surprising

$$50/50 \rightarrow 1.0 \text{ and } 90/10 \rightarrow 0.47 \text{ and } 100/0 \rightarrow 0$$



- calculator $\rightarrow \log_2(x) = \log(x) / \log(2)$

- $0 \times \log(0)$ counts as 0

information gain

- how much a split lowers entropy $\rightarrow$ ID3 picks the highest gain

$$\text{Info_A}(D) = \sum (|D_j| / |D|) \times \text{Info}(D_j)$$

$$\text{Gain}(A) = \text{Info}(D) - \text{Info_A}(D)$$

- D_j = rows that go down branch j and $|D_j|$ = how many

- why the weights $\rightarrow$ a random row lands in branch j with probability $|D_j| / |D| \rightarrow \text{Info_A}$ = expected entropy left after asking A

- gain = what asking A tells you about the class

- never negative $\rightarrow$ a split can not make you know less on average

- 0 when every branch keeps the parent mix

- 14 rows 8 yes 6 no $\rightarrow$ split into 4 3 and 4 3 $\rightarrow$ each child 0.985 $\rightarrow$ gain 0

the 7 customers

id	age	student	credit	buy
1	young	yes	excellent	yes
2	young	no	fair	no
3	middle	no	excellent	yes
4	senior	no	excellent	yes
5	senior	no	fair	no
6	senior	no	fair	no
7	middle	no	fair	yes
8	middle	yes	excellent	?

- count first
 - all $\rightarrow$ 4 buy and 3 no
 - young 1 1 and middle 2 0 and senior 1 2
 - student yes 1 0 and student no 3 3
 - excellent 3 0 and fair 1 3

$$\text{Info}(D) = 0.985$$

$$\text{Info}_{\text{age}} = (2/7)(1.0) + (2/7)(0) + (3/7)(0.918) = 0.679$$

$$\text{Gain}(\text{age}) = 0.985 - 0.679 = 0.31$$

$$\text{Info}_{\text{student}} = (1/7)(0) + (6/7)(1.0) = 0.857$$

$$\text{Gain}(\text{student}) = 0.13$$

$$\text{Info}_{\text{credit}} = (3/7)(0) + (4/7)(0.811) = 0.464$$

$$\text{Gain}(\text{credit}) = 0.52$$

- credit is not computed on the slides and it wins $\rightarrow$ credit = root

lookup

counts	entropy	gini
1 1	1.000	0.500
1 2	0.918	0.444
1 3	0.811	0.375
2 3	0.971	0.480
1 4	0.722	0.320
3 4	0.986	0.490
2 5	0.863	0.408
5 9	0.940	0.459
4 0	0	0

gain ratio

- C4.5 fix for the bias of gain toward attributes with many values
- why needed
 - split on customer id $\rightarrow$ 7 groups of 1 $\rightarrow$ all pure $\rightarrow$ gain = 0.985 = the max
 - a new customer id matches no branch $\rightarrow$ the split predicts nothing
 - more branches = smaller groups = purer by chance

$$\text{SplitInfo}_{\text{ACD}} = -\sum (|D_j|/|D|) \log_2(|D_j|/|D|)$$

$$\text{GainRatio}(A) = \text{Gain}(A) / \text{SplitInfo}_A(D)$$

- SplitInfo = entropy of the branch sizes and ignores the class
- many equal small branches $\rightarrow$ SplitInfo up to $\log_2(V) \rightarrow$ divides the gain down
- $\text{SplitInfo}(\text{age}) = \text{entropy of sizes } 2 \ 2 \ 3 = 1.557$
- $\text{GainRatio}(\text{age}) = 0.31 / 1.557 = 0.20$
- id $\rightarrow \text{SplitInfo} = \log_2(7) = 2.81 \rightarrow \text{ratio} = 0.985 / 2.81 = 0.35$
- credit $\rightarrow \text{SplitInfo} = 0.985 \rightarrow \text{ratio} = 0.52 / 0.985 = 0.53$
- credit still wins and id is pushed down
- weak spot $\rightarrow$ a tiny branch makes SplitInfo near 0 $\rightarrow$ the ratio blows up $\rightarrow$ C4.5 only compares ratios among attributes with at least average gain
- Slide Correction
 - slide says $\rightarrow \text{SplitInfo}(\text{age}) = 1.62$
 - correct $\rightarrow 1.557 \rightarrow 2 \times (2/7)(1.807) + (3/7)(1.222) \rightarrow$ the ratio still rounds to 0.20
 - in the exam $\rightarrow$ show 1.557 and the same 0.20

gini index

- CART impurity $\rightarrow$ pick the split with the lowest weighted gini
- $\text{Gini}(D) = 1 - \sum p^2$ over each class
- $\text{Gini}_A(D) = (|D_1|/|D|) \text{Gini}(D_1) + (|D_2|/|D|) \text{Gini}(D_2)$
- reduction = $\text{Gini}(D) - \text{Gini}_A(D)$
- where it comes from $\rightarrow$ pick 2 rows at random with replacement
 - chance both are class i = $p \times p \rightarrow$ chance they match = $\sum p^2$
 - chance they differ = $1 - \sum p^2 = \text{gini}$
 - = chance you label a row wrong if you guess at random in the class shares
- 4 yes 3 no $\rightarrow 2 \times (4/7)(3/7) = 24/49 = 0.49$
- 0 = pure and max = $1 - 1/K$ at equal shares $\rightarrow$ 2 classes 0.5 and 3 classes 0.667
- CART only splits in 2 $\rightarrow$ try every 2 group option
- nominal with v values $\rightarrow 2^{(v-1)} - 1$ options
- $v = 3 \rightarrow 3$ and $v = 4 \rightarrow 7$
- why binary only
 - any multi way split = several binary ones $\rightarrow$ no power lost
 - multi way splits chop the rows fast $\rightarrow$ the many values bias again
 - one threshold on a number is alr a binary split $\rightarrow$ same code for both

worked

$$\text{Gini}(D) = 1 - (4/7)^2 - (3/7)^2 = 0.49$$

$$\text{young vs rest} = (2/7)(0.5) + (5/7)(0.48) = 0.486$$

$$\text{senior vs rest} = (3/7)(0.444) + (4/7)(0.375) = 0.406$$

$$\text{middle vs rest} = (2/7)(0) + (5/7)(0.48) = 0.343$$

$$\text{student} = (6/7)(0.5) = 0.429$$

$$\text{credit} = (4/7)(0.375) = 0.214$$

- Credit lowest $\rightarrow$ wins again

- best age split = middle vs rest $\rightarrow$ reduction $0.49 - 0.343 = 0.15$

- Slide Correction

- slide says $\rightarrow$ $Gini(D) = 0.51$ and young vs others = 0.62 and $Gini(\text{age}) = 0.51 - 0.34 = 0.17$

- correct $\rightarrow$ $Gini(D) = 1 - 16/49 - 9/49 = 0.49 \rightarrow$ young vs others = 0.486 $\rightarrow$ reduction 0.16

- in the exam $\rightarrow$ show 0.49 and 0.486 and say the slide prints 0.51 and 0.62

gini vs entropy

- same hump shape $\rightarrow$ pick the same split nearly always $\rightarrow$ gini has no log so it is faster

- why not just count errors $\rightarrow$ 1 - biggest p misses progress

- 400 400 $\rightarrow$ split into 300 100 and 100 300 $\rightarrow$ error 0.25

- 400 400 $\rightarrow$ split into 200 400 and 200 0 $\rightarrow$ error 0.25 too

- gini 0.375 vs 0.333 and entropy 0.811 vs 0.689 $\rightarrow$ both prefer the second cuz it made a pure node

decision tree

- decision tree induction = learning a tree from rows that all have labels

- root = first test and internal node = a test on one attribute

- branch = one outcome of a test and leaf = a class

- predict = start at the root $\rightarrow$ follow the matching branch $\rightarrow$ read the leaf

- why it works $\rightarrow$ each test cuts the rows into purer groups $\rightarrow$ keep cutting until each group is one class $\rightarrow$ the path to a leaf is an if then rule

- like 20 questions $\rightarrow$ ask the question that cuts the possibilities most $\rightarrow$ analogy

- ID3 = Iterative Dichotomiser $\rightarrow$ information gain $\rightarrow$ multiway splits

- C4.5 = successor of ID3 $\rightarrow$ gain ratio $\rightarrow$ also numbers and missing values

- CART = Classification and Regression Tree $\rightarrow$ gini $\rightarrow$ binary splits $\rightarrow$ also regression

- sklearn DecisionTreeClassifier = CART $\rightarrow$ default criterion gini

attribute types

- nominal → names with no order → student yes no or colour
 - split → one branch per value in ID3 or any 2 group option in CART
- ordinal → names with an order → age young middle senior
 - split → keep the order → young vs middle and senior is fine and young and senior vs middle is not
 - v values → v - 1 order keeping cuts
- numeric → petal width → threshold → petal width ≤ 0.80
 - sort the values and try a cut between each pair of neighbours → n values give n - 1 cuts
 - only the order matters → scaling never changes a tree
- borderline → zip code is digits but nominal
- customer id → every value different → gain loves it and it predicts nothing → drop it

learning process

start with partition D and the attribute list and a selection method

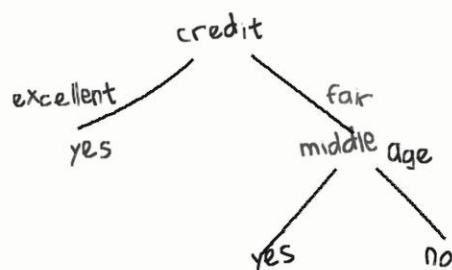
- make node N for D
- all rows one class → N becomes a leaf of that class
- else → score every attribute → take the best splitting criterion
- label N with that test and grow one branch per outcome
- repeat on each branch with only its rows → recursion
- stop when a rule hits
 - node is pure
 - no attributes left → leaf = majority class
 - branch got no rows → leaf = majority class of the parent
 - a pre pruning limit → max depth or min rows
- greedy → best split now and never undo → fast but maybe not the best tree overall
- Iris all 150 rows → root = petal width ≤ 0.80

$$\text{root gini} = 1 - 3 \times (1/3)^2 = 0.667$$

$$\text{after} = (50/150)(0) + (100/150)(0.5) = 0.333$$

the 7 customers as a tree

- credit at root
- excellent → rows 1 3 4 → all yes → leaf yes
- fair → rows 2 5 6 7 → 1 yes 3 no → split by age
 - middle → row 7 → yes and the rest → no → pure → stop



- customer 8 → excellent → buy yes

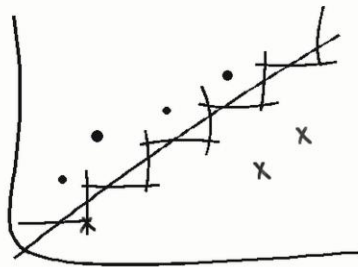
- the slide tree with age at the root comes from a bigger textbook table and not from these 7 rows

reading a fitted tree

- leaf probability = class shares in the leaf $\rightarrow$ 8 yes 2 no $\rightarrow P(\text{yes}) = 0.8 \rightarrow$ predict_proba
- depth $d \rightarrow$ at most 2^d leaves
- feature importance = total impurity drop from splits on that feature weighted by rows reaching each node $\rightarrow$ scaled to sum to 1
 - Iris tree $\rightarrow$ petal width 0.92
- hyperparameters
 - criterion $\rightarrow$ gini or entropy
 - max_depth and min_samples_split and min_samples_leaf $\rightarrow$ pre pruning
 - ccp_alpha $\rightarrow$ post pruning

good and bad

- good $\rightarrow$ readable rules and no scaling and handles numbers and categories and fast predict
- overfits $\rightarrow$ no limit grows until every leaf is pure $\rightarrow$ memorises noise
- unstable $\rightarrow$ change a few rows and the root can change
- splits are along one feature $\rightarrow$ a diagonal border needs a staircase of many cuts



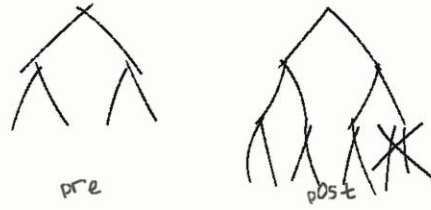
- greedy $\rightarrow$ misses a pair of splits that only work together

regression trees

- CART also predicts numbers $\rightarrow$ DecisionTreeRegressor
- impurity = variance of $y \rightarrow$ split = biggest drop in squared error
- a leaf predicts the mean y of its rows $\rightarrow$ leaf with 10 12 14 $\rightarrow$ 12

tree pruning

- removing the least reliable branches $\rightarrow$ smaller tree $\rightarrow$ easier to read and better on new rows
- why $\rightarrow$ deep branches rest on few rows $\rightarrow$ a leaf made from 1 odd row fits noise



pre pruning

- stop early → a test says the split is not worth it → node becomes a leaf
- the leaf holds the majority class or the class probabilities
- tests → gain below a threshold or depth limit or too few rows
- cheap but can stop too early
 - XOR → each feature alone gives gain 0 but both together split perfectly → pre pruning stops at the root

post pruning

- grow the full tree → cut subtrees from the bottom up
- a cut subtree becomes a leaf with the majority class of its rows
- keep the cut if error on held out data does not get worse
- sklearn cost complexity pruning

$$R_{\alpha}(T) = R(T) + \alpha * (\text{number of leaves})$$
- $R(T)$ = training error or impurity → α = price per leaf
- $\alpha = 0$ → full tree and α huge → one leaf → pick α with cross validation

pre vs post

- pre = cheap and may stop too early
- post = costs more and sees the whole tree first so usually better
- both trade a bit more bias for much less variance

exam check

- confusion matrix given → find TP FP FN TN → check which axis is actual → plug in
- which metric → name the costly mistake
 - disease or fraud → FN costly → recall
 - spam or verdict → FP costly → precision
 - rare class → F1 or recall and not accuracy
- derive a metric → write it as $P(\text{something} | \text{something})$
- entropy or gain → counts per branch first → lookup → weighted sum
- gain ratio → the fix for many valued attributes
- gini → binary splits → list every 2 group option → pick lowest
- build a tree → best root → repeat on each impure branch → stop when pure

- predict a new row $\rightarrow$ walk it down the tree
- overfit or underfit $\rightarrow$ train vs test
- pre vs post pruning $\rightarrow$ stop early vs cut after
- log loss $\rightarrow$ from likelihood $\rightarrow$ lower is better
- describe any model $\rightarrow$ model and loss and optimiser
- mix ups
 - precision = predicted positive side and recall = actual positive side
 - specificity = recall of the negative class
 - gain $\rightarrow$ highest and gini $\rightarrow$ lowest
 - ID3 gain and C4.5 gain ratio and CART gini
 - sklearn rows = actual and slide rows = predicted

conceptual questions

- binary vs multiclass $\rightarrow$ 2 classes vs more than 2 with one per row $\rightarrow$ multiclass metrics per class then averaged
- hypothesis $\rightarrow$ one rule $h \rightarrow H =$ every rule of that shape $\rightarrow H$ is an assumption and if the truth is not in H some error stays
- generalization $\rightarrow$ doing well on unseen rows $\rightarrow$ train error can not show it cuz θ was chosen to fit those rows
- overfit $\rightarrow$ train high and test low $\rightarrow$ too complex or too little data $\rightarrow$ underfit $\rightarrow$ both low $\rightarrow$ too simple
- more data helps $\rightarrow$ new variety is harder to memorise $\rightarrow$ duplicates add nothing
- iid $\rightarrow$ rows independent and from the same population as future rows $\rightarrow$ without it train says nothing about test
- FP vs FN $\rightarrow$ said positive and wrong vs said negative and wrong $\rightarrow$ cancer FN worse $\rightarrow$ spam FP worse cuz a real mail is hidden
- S and G $\rightarrow$ tightest and biggest zero error box $\rightarrow$ margin = gap $\rightarrow$ pick the middle so small noise does not flip the label
- accuracies on rare classes $\rightarrow$ 10 cancers in 100 $\rightarrow$ always benign = 0.90 and 0 recall
- precision vs recall $\rightarrow$ of my alarms how many real vs of the real ones how many caught $\rightarrow$ flag more = recall up precision down
- F1 harmonic $\rightarrow$ the smaller of P and R dominates $\rightarrow$ P 1.0 R 0.1 $\rightarrow$ mean 0.66 but F1 0.18
- specificity $\rightarrow$ TN / (TN + FP) $\rightarrow$ recall of the negative class $\rightarrow$ FPR = 1 - specificity
- averages $\rightarrow$ macro = each class equal $\rightarrow$ weighted = by support $\rightarrow$ micro = pool counts = accuracy in single label
- log loss $\rightarrow$ scores the probability $\rightarrow$ -log of the probability given to the truth $\rightarrow$ from maximum likelihood $\rightarrow$ confident and wrong is huge
- three decisions $\rightarrow$ model and loss and optimiser $\rightarrow$ tree = tree of tests and impurity and greedy splits

- tree building $\rightarrow$ pure $\rightarrow$ leaf $\rightarrow$ else best split $\rightarrow$ branch per outcome $\rightarrow$ recurse $\rightarrow$ stop at pure or no attributes or no rows or a limit
- entropy $\rightarrow$ average surprise $\rightarrow$ 0 when pure $\rightarrow$ max at equal shares cuz guessing is hardest
- information gain $\rightarrow$ entropy before minus weighted entropy after $\rightarrow$ never negative $\rightarrow$ 0 when each branch keeps the parent mix
- gain ratio $\rightarrow$ gain loves many valued attributes like id $\rightarrow$ divide by split info
- gini $\rightarrow$ chance two random rows differ $\rightarrow$ same picks as entropy nearly always $\rightarrow$ faster
- why not plain error rate $\rightarrow$ misses progress like making a pure node $\rightarrow$ 400 400 example
- CART binary $\rightarrow$ any multi way split = several binary ones $\rightarrow$ v values give $2^{(v-1)}$ options
- ID3 gain multi way $\rightarrow$ C4.5 gain ratio and numbers $\rightarrow$ CART gini binary and regression
- numeric attribute $\rightarrow$ sort and try a threshold between neighbours $\rightarrow$ scaling keeps the order so the tree is the same
- attribute types $\rightarrow$ nominal any grouping $\rightarrow$ ordinal keep the order $\rightarrow$ numeric threshold
- greedy $\rightarrow$ best split now and never undo $\rightarrow$ may miss the best tree $\rightarrow$

XOR

- trees overfit $\rightarrow$ grow until pure $\rightarrow$ leaves from 1 row fit noise $\rightarrow$ pruning cuts unreliable branches
- pre vs post $\rightarrow$ stop early and cheap but may stop too soon vs grow then cut and costlier but better
- leaf $\rightarrow$ majority class $\rightarrow$ probability = class shares $\rightarrow$ 8 yes 2 no $\rightarrow$ yes with 0.8
- tree good $\rightarrow$ readable and no scaling and fast $\rightarrow$ bad $\rightarrow$ overfits and unstable and staircase borders and greedy

support $\rightarrow$ actual rows per class $\rightarrow$ macro far below weighted $\rightarrow$ a small class is failing

- test set $\rightarrow$ kept apart so the score is honest $\rightarrow$ leakage = test info in training $\rightarrow$ score too high
- random_state $\rightarrow$ fixes the split and the trees random choices $\rightarrow$ without it Iris gave 1.00 or 0.978
- sklearn matrix $\rightarrow$ rows actual and columns predicted $\rightarrow$ precision = diagonal / column and recall = diagonal / row
- root by hand $\rightarrow$ count classes per branch $\rightarrow$ entropy or gini per branch $\rightarrow$ weighted sum $\rightarrow$ best gain or lowest gini
- feature importance $\rightarrow$ total impurity drop from splits on that feature weighted by rows $\rightarrow$ sums to 1
- regression tree $\rightarrow$ variance replaces impurity $\rightarrow$ leaf predicts the mean

lab code

- Iris → 150 flowers and 4 measurements and 3 species
- Orthopedic → 310 patients and 6 spine angles and 3 classes
- every model → make it then fit then predict then score

imports

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

- 1 numpy → arrays and maths
- 2 pandas → tables = DataFrame
- 3 matplotlib.pyplot → plots
- 4 seaborn → statistical plots on top of matplotlib

load

```
url = "https://raw.githubusercontent.com/
elfin/datasets/main/Iris.csv"
data = pd.read_csv(url)
data = data.drop(columns='Id')
data.head()
```

- 1 pd.read_csv → reads the csv from the link into a table
- 2 drop(columns='Id') → removes the row number column → an id is not a feature and only helps memorise rows
- 3 head → first 5 rows

explore

```
data.info()
data.describe()
sns.countplot(data["Species"], palette="Set2")
plt.title("Species")
plt.show()
```

- 1 info → column names and non null counts and dtypes → 150 rows and no missing values
 - 2 describe → count mean std min quartiles max per column
 - 3 countplot → rows per class → Iris 50 50 50 balanced
 - 4 title and show → label and draw the plot
- Orthopedic → Spondylolisthesis 150 and Normal 100 and Hernia 60 → not balanced → accuracy alone misleads
- ```
color_list = ['red' if i == 'Iris-setosa' else
 'blue' if i == 'Iris-versicolor' else
 'green'
 for i in data.loc[:, 'Species']]
```

```
pd.plotting.scatter_matrix(
 data.loc[:, data.columns != 'Species'],
 c=color_list, figsize=[15,15], diagonal='hist')
sns.pairplot(data=data, hue='Species')
```

- 1 color\_list → one colour per row from its species
  - 2 data.loc[:, data.columns != 'Species'] → every column except Species
  - 3 scatter\_matrix → every feature against every other and histograms on the diagonal
  - 4 pairplot → the same grid in seaborn coloured by hue
- read → petal length and petal width split setosa off cleanly → expect a very accurate tree

### preprocess

```
X, y = data.loc[:, data.columns != 'Species'], data.loc[:, 'Species']
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X))
```

- 1 loc → X = the 4 feature columns and y = Species
- 2 MinMaxScaler → squeezes each column into 0 to 1  
$$x' = (x - \min) / (\max - \min)$$
- 3 fit\_transform → learns each columns min and max then applies the formula
- 4 pd.DataFrame → wraps the array back into a table → column names become 0 1 2 3

### split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
 X_scaled, y, test_size = 0.3, random_state = 32)
```

- 1 train\_test\_split → shuffles and splits the rows
- 2 test\_size = 0.3 → Iris 105 train and 45 test → Orthopedic 217 and 93
- 3 random\_state = 32 → seed so the same split comes out every run

### decision tree

```
from sklearn import tree
dtree = tree.DecisionTreeClassifier()
dtree.fit(X_train, y_train)
prediction = dtree.predict(X_test)
print('Prediction: {}'.format(prediction))
print('Decision tree accuracy is:', dtree.score(X_test, y_test))
```

- 1 DecisionTreeClassifier → empty CART tree → gini and no depth limit  
→ no pruning
- 2 fit → grows the tree on the training rows

- 3 predict → runs each test row down the tree → one species per row
- 4 format → puts the array into the printed text
- 5 score → accuracy on the test rows
- lab → Iris 0.978 = 44 of 45 → Orthopedic 0.72

### plot the tree

```
plt.figure(figsize=(15,15))
tree.plot_tree(dtree)
plt.show()

import graphviz
dot_data = tree.export_graphviz(dtree, out_file=None,
 feature_names=X.columns.values.tolist(),
 class_names=y.unique(),
 filled=True, rounded=True, special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

1 figure → 15 by 15 inch canvas

2 plot\_tree → draws every node with its test and gini and samples and value

3 export\_graphviz → writes the tree as DOT text → out\_file=None returns the text

4 feature\_names → real column names cuz X\_scaled lost them

5 class\_names → class names for the boxes

6 filled → colours each node by its majority class → rounded = round boxes

7 graphviz.Source → draws the DOT text

- reading a node → samples = rows reaching it → value = rows per class → gini = impurity → class = majority

- Lab Correction → class\_names=y.unique() lists classes in order of first appearance but sklearn sorts them → dtree.classes\_

- Iris → same order by luck

- Orthopedic → Normal and Spondylolisthesis are swapped in the picture

- fix → class\_names=dtree.classes\_

### confusion matrix and report

```
from sklearn.metrics import confusion_matrix
cf_matrix = confusion_matrix(y_test, prediction)
sns.heatmap(cf_matrix, annot=True)
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, prediction))
```

1 confusion\_matrix → rows = actual and columns = predicted → classes sorted

2 heatmap → colours the matrix → annot=True writes the numbers

3 classification\_report → precision recall F1 support per class plus accuracy and macro and weighted avg

- lab Iris → 1 mistake → one virginica called versicolor

- lab Orthopedic → order Hernia Normal Spondylolisthesis

[ 7, 14, 0],

[ 7, 26, 1],

[ 0, 4, 34]]

- Hernia recall  $7/21 = 0.33$  → 14 hernias called Normal → those two classes overlap

- Spondylolisthesis F1 0.93 → easy class

- accuracy 0.72 and macro F1 0.67 and weighted F1 0.71

## random forest and adaboost

- theory → week 6

from sklearn.ensemble import RandomForestClassifier

randforest = RandomForestClassifier()

randforest.fit(X\_train, y\_train)

randforest.score(X\_test, y\_test)

tree.export\_graphviz(randforest.estimators\_[1], ...)

from sklearn.ensemble import AdaBoostClassifier

adaBoost = AdaBoostClassifier(n\_estimators=100)

adaBoost.fit(X\_train, y\_train)

adaBoost.score(X\_test, y\_test)

1 RandomForestClassifier → 100 trees each on a bootstrap sample

→ predict = majority vote

2 estimators\_ → the list of fitted trees → [1] = the second one → all 100 can not be drawn

3 AdaBoostClassifier → stumps trained one after another →

n\_estimators=100 rounds → default 50

4 fit and predict and score → same calls as the tree

- lab → forest Iris 0.978 and Orthopedic 0.82 → adaboost Iris 1.00 and Orthopedic 0.78

## exercise 2

- same calls on the Orthopedic file with target class\_status

- the code cells are empty and the outputs are kept → copy

exercise 1 and change url and Species → class\_status

- Lab Correction → no random\_state on the tree or forest → the score changes each run

- one tree → sklearn shuffles the feature order at every node → ties go to the first one tried

- Iris tree on the same split gave 1.00 or 0.978

- Lab Correction → scaling does nothing for trees → the Orthopedic tree averages 0.788 over 50 splits with or without MinMaxScaler

- Lab correction → the scaler is fit on all rows before the split → leakage → harmless for trees but wrong for scaled models
- Lab correction → 0.72 is one split → over 50 splits the Orthopedic tree ranges 0.70 to 0.88
- Lab Correction → countplot with palette and no hue gives a Future Warning → `sns.countplot(x="Species", hue="Species", data=data)`

## good practice

- not on the slides
- split with `stratify=y` so each class keeps its share  
`train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)`
- set `random_state` on the model too
- skip the scaler for trees
- prune by cross validation on train only  
`GridSearchCV(tree.DecisionTreeClassifier(random_state=0),  
{'max_depth': [2, 3, 4, 5, None]}, cv=5)`
- Orthopedic 5 fold CV → no limit 0.81 and depth 3 0.82 and depth 4 0.83 and depth 2 0.77
- compare models by CV mean and not one test split
- unbalanced classes → report macro F1 → `class_weight='balanced'` makes missed Hernia rows count more

## week 6

### Overview

- one deep tree → low bias and high variance → unstable and overfits
- fix 1 → many trees in parallel on shuffled data and vote → bagging → random forest → lowers variance
- fix 2 → trees in sequence each fixing what the last missed → boosting → adaboost then XGBoost → lowers bias
- fix 3 → different models and a final model on top of them → stacking
- lab also adds → filling missing values and SMOTE for unbalanced classes and GridSearchCV tuning

### ensembles

- ensemble = many models vote → the group beats one model
- why it works → each voter right more than half the time and their mistakes differ → the majority is right more often  
3 voters at 0.7 each →  $P(2 \text{ or } 3 \text{ right}) = 0.784$   
5 voters → 0.837

26 voters  $\rightarrow 0.98$

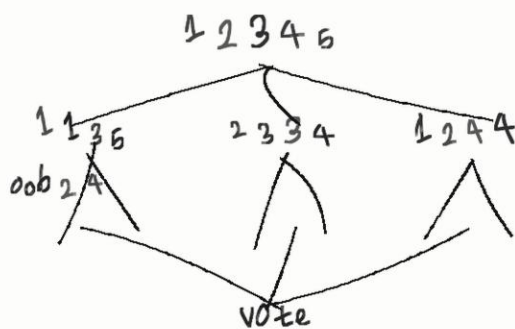
- fails when voters make the same mistakes  $\rightarrow$  3 copies of one tree = one tree
- so the whole game = make the members different
- types
  - bagging  $\rightarrow$  same model on bootstrap samples in parallel  $\rightarrow$  equal vote  $\rightarrow$  random forest
  - boosting  $\rightarrow$  models in sequence each fixing the last ones mistakes  $\rightarrow$  weighted sum  $\rightarrow$  adaboost and XGBoost
  - stacking  $\rightarrow$  different model types  $\rightarrow$  a final model learns how to combine them

## random forest

- extended decision tree  $\rightarrow$  builds many trees in training  $\rightarrow$  final answer = majority vote
- slide factors  $\rightarrow$  the forest is good when each tree is strong and the trees are not correlated
- supervised  $\rightarrow$  classification by majority vote  $\rightarrow$  regression by averaging the trees  $\rightarrow$  RandomForestRegressor

### how it works

- draw a random sample of rows  $\rightarrow$  bootstrap
- build a tree on each sample  $\rightarrow$  each gives a prediction
- collect the votes
- most votes = the final prediction



### diversity

- bagging = bootstrap aggregating  $\rightarrow$  each tree draws  $N$  rows with replacement from the  $N$  rows
  - the data is not chopped into separate chunks  $\rightarrow$  every tree sees a full size sample with repeats
- $P(\text{a row is never drawn}) = (1 - 1/N)^N \rightarrow 0.368$ 
  - so each tree misses about  $1/3$  of the rows
- feature randomness  $\rightarrow$  at each split the tree may only choose from a random subset of features  $\rightarrow$  default  $\sqrt{(\text{features})}$ 
  - Pima 8 features  $\rightarrow \sqrt{8}$  rounded down = 2 candidates per split
  - stops every tree from starting with the same strong feature  $\rightarrow$

lower correlation

- why lower correlation matters  $\rightarrow$  averaging  $B$  trees each with variance  $v$  and correlation  $c$   
$$\text{variance} = c \times v + (1 - c) \times v / B$$
- more trees shrink only the second part  $\rightarrow$  the first part needs lower  $c$   
 $c = 0.5$  and  $B = 100 \rightarrow 0.506v$   
 $c = 0.2$  and  $B = 100 \rightarrow 0.208v$
- Slide Correction
  - slide says  $\rightarrow$  each tree can pick only from a random subset of features
  - correct  $\rightarrow$  a new random subset is drawn at every split and not once per tree  $\rightarrow$  sklearn `max_features` works per split
  - in the exam  $\rightarrow$  say per split and mention the slide words it per tree

### out of bag error

- each tree leaves out about  $1/3$  of the rows  $\rightarrow$  those are out of bag = OOB for that tree
- send each OOB row down the trees that never saw it  $\rightarrow$  vote  $\rightarrow$  compare with its true class
- share of wrong votes over all rows = OOB error
- = a free test score with no separate validation set
  - Pima  $\rightarrow$  OOB accuracy 0.766 vs 5 fold CV 0.751  $\rightarrow$  close

### reading the fitted forest

- feature importance = how much the nodes that use a feature lower impurity across all trees
  - Pima  $\rightarrow$  Glucose 0.24 and BMI 0.18 and Age 0.14  $\rightarrow$  the rest under 0.13
- use it to drop weak features  $\rightarrow$  fewer features  $\rightarrow$  less overfitting
- the single readable tree is lost  $\rightarrow$  importance is what you read instead

### hyperparameters

- `n_estimators`  $\rightarrow$  number of trees  $\rightarrow$  default 100  $\rightarrow$  more = steadier and slower and never more overfit
- `max_features`  $\rightarrow$  features tried per split  $\rightarrow$  sqrt or log2  $\rightarrow$  lower = less correlated trees
- `max_depth` and `min_samples_leaf`  $\rightarrow$  how deep each tree grows  $\rightarrow$  default no limit
- `oob_score=True`  $\rightarrow$  also computes the OOB score
- `n_jobs`  $\rightarrow$  cores used  $\rightarrow$  the real speed setting
- `random_state`  $\rightarrow$  same forest every run

- Slide Correction
  - slide says  $\rightarrow$  to increase speed use random state and oob score
  - correct  $\rightarrow$  speed =  $n\_jobs \rightarrow$  random\_state = repeatable runs  $\rightarrow$  oob\_score = a free validation score that costs a little extra time
  - in the exam  $\rightarrow$  say  $n\_jobs$  for speed and give what the other two really do

### why bias stays and variance drops

- every tree is built the same way from the same kind of sample  $\rightarrow$  their average prediction = one tree's average prediction
- averaging moves nothing closer to the truth  $\rightarrow$  bias stays  $\rightarrow$  it only removes the wobble  $\rightarrow$  variance drops

### good and bad

- good  $\rightarrow$  handles many input variables  $\rightarrow$  feature importance  $\rightarrow$  avoids overfitting  $\rightarrow$  no scaling needed
- bad  $\rightarrow$  slower than one tree  $\rightarrow$  100 trees to store and to run
- Slide Correction
  - slide says  $\rightarrow$  cannot handle the noise of data and too slow for real time
  - correct  $\rightarrow$  the vote averages out much of the noise so it handles noise better than one tree or boosting  $\rightarrow$  but deep trees still fit label noise one by one
  - correct  $\rightarrow$  predict = depth many tests  $\times$  100 trees  $\rightarrow$  milliseconds per row  $\rightarrow$  slow only with very many trees or tight limits
  - in the exam  $\rightarrow$  give the slide parts and add the reason  $\rightarrow$  each deep tree fits noise and many trees cost time

### adaboost

- in the week 5 lab and not on the slides
- small trees one after another  $\rightarrow$  each new one focuses on rows the earlier ones got wrong  $\rightarrow$  weighted vote
- base learner = stump = tree with 1 split  $\rightarrow$  weak = barely better than guessing
- start  $\rightarrow$  every row weight  $w = 1/N$
- each round
  - err = sum of weights of wrong rows
  - $\alpha = 0.5 \times \ln((1 - \text{err}) / \text{err})$
  - wrong rows  $\rightarrow w \times e^{\alpha}$
  - right rows  $\rightarrow w \times e^{-\alpha}$
- divide all  $w$  by their sum so they add to 1
- $$f(x) = \text{sign}(\sum \alpha \times h(x))$$

- why  $\alpha$  looks like this
  - $\text{err} < 0.5 \rightarrow \alpha > 0 \rightarrow$  better than guessing gets a say
  - $\text{err small} \rightarrow \alpha \text{ big} \rightarrow$  accurate stumps vote louder
  - $\text{err} = 0.5 \rightarrow \alpha = 0 \rightarrow$  a coin flip gets no say
- after the update the wrong rows hold exactly half the weight  $\rightarrow$  the last stump now scores err 0.5  $\rightarrow$  the next stump must be different

### worked

- 10 rows and the stump misses 3
  - $\text{err} = 3 \times 0.1 = 0.3$
  - $\alpha = 0.5 \times \ln(0.7 / 0.3) = 0.42$
  - wrong  $\rightarrow 0.1 \times 1.528 = 0.1528 \rightarrow 3 \text{ rows} = 0.458$
  - right  $\rightarrow 0.1 \times 0.655 = 0.0655 \rightarrow 7 \text{ rows} = 0.458$
  - total = 0.917
  - after dividing  $\rightarrow$  wrong 0.167 each and right 0.071 each
- each wrong row now weighs over 2x a right row
- lowers bias  $\rightarrow$  many weak rules add up to shapes one stump can not draw
- weakspot  $\rightarrow$  noisy labels keep getting wrong  $\rightarrow$  their weight explodes  $\rightarrow$  the model chases noise
- hyperparameters  $\rightarrow$   $n_{\text{estimators}}$   $\rightarrow$  default 50  $\rightarrow$  learning\_rate shrinks each  $\alpha$

## gradient boosting

- not on the slides  $\rightarrow$  it is the idea XGBoost is built on
- adaboost reweights rows  $\rightarrow$  gradient boosting fits the leftover error directly
- squared loss  $\rightarrow$  leftover = residual =  $y - \hat{y}$
- log loss  $\rightarrow$  leftover =  $y - p \rightarrow$  the gap between the true label and the predicted probability
- $\hat{y}(t) = \hat{y}(t-1) + \text{eta} \times f_t(x)$
- eta = learning rate  $\rightarrow$  each tree only moves part of the way



### worked

- 3 rows  $y = 2 \ 4 \ 9 \rightarrow$  start  $\hat{y} = 0 \rightarrow \text{eta} = 0.5 \rightarrow$  each tree has 1 split
- round 1  $\rightarrow$  residuals  $2 \ 4 \ 9 \rightarrow$  tree splits rows 1 2 vs 3  $\rightarrow$  leaves 3 and 9

$$\hat{y} = 0 + 0.5 \times (3 \ 3 \ 9) = 1.5 \ 1.5 \ 4.5$$

$$\text{residuals} = 0.5 \ 2.5 \ 4.5$$

- round 2  $\rightarrow$  tree splits row 1 vs rows 2 3  $\rightarrow$  leaves 0.5 and 3.5

$$\hat{y} = 1.5 \ 1.5 \ 4.5 + 0.5 \times (0.5 \ 3.5 \ 3.5) = 1.75 \ 3.25 \ 6.25$$

$$\text{residuals} = 0.25 \ 0.75 \ 2.75$$

- sum of squared residuals  $\rightarrow$  101 then 26.75 then 8.19  $\rightarrow$  shrinks every round

- why residuals  $\rightarrow$  the gradient of the squared loss  $0.5(y - \hat{y})^2$  is  $-(y - \hat{y})$

$\rightarrow$  stepping against the gradient = adding the residual

- why eta  $\rightarrow$  small steps and more trees overfit less than a few big steps

## xgboost

- XGBoost = gradient boosting + regularization + speed tricks

- slide  $\rightarrow$  strong on tabular data and wins many Kaggle competitions

## how it works

- train the first tree on the data

- compute the gradients of the loss for every row  $\rightarrow$  how the loss would change if the prediction moved a bit

- train the next tree to fit those gradients  $\rightarrow$  with a penalty on tree complexity

- scale the new trees prediction by the learning rate and add it to the model

- repeat for a set number of rounds or until early stopping sees no more improvement

- final output = sum of all trees  $\rightarrow$  each one a small correction

- Slide correction

- slide says  $\rightarrow$  the first tree is trained on the data

- correct  $\rightarrow$  the model starts from a constant prediction  $\rightarrow$  base\_score  $\rightarrow$  and even the first tree fits the gradients around it and not the raw labels

- in the exam  $\rightarrow$  say start from a constant then every tree fits gradients

## model

$$\hat{y} = \sum f_k(x) \text{ over } k \text{ trees}$$

$$\theta = [f_1 \ f_2 \ \dots \ f_K]$$

- the parameters are the trees themselves  $\rightarrow$  not a vector of numbers

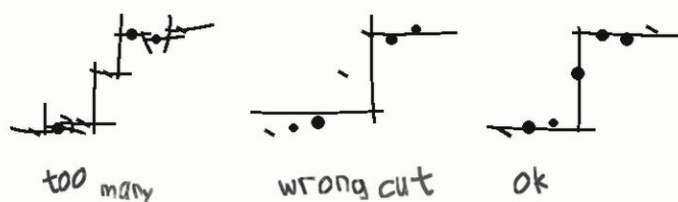
## objective

$$obj = \sum L(y, \hat{y}) + \sum \omega(f_k)$$

- first sum  $\rightarrow$  loss over the  $N$  rows  $\rightarrow$  fit the data
- second sum  $\rightarrow$  penalty over the  $K$  trees  $\rightarrow$  keep each tree simple  
 $\text{omega}(f) = \gamma * T + 0.5 * \lambda * \sum w^2$
- $T$  = number of leaves  $\rightarrow$   $\gamma$  = price per leaf
- $w$  = leaf weights = the values the leaves output  $\rightarrow$   $\lambda$  = L2 penalty  $\rightarrow$  reg\_alpha adds an L1 penalty
- Slide Correction
  - slide says  $\rightarrow$   $\text{omega}$  can be the number of nodes in the tree
  - correct  $\rightarrow$  number of leaves  $T$   $\rightarrow$  priced by  $\gamma$   $\rightarrow$  plus the L2 or L1 size of the leaf weights
  - in the exam  $\rightarrow$  write  $\text{omega} = \gamma * T + 0.5 \lambda \sum w^2$  and name the 3 options from the slide with leaves for nodes

### why regularization

- slide example  $\rightarrow$  one tree on a single variable  $x$



- too many splits  $\rightarrow$   $\text{omega}$  high  $\rightarrow$  follow every point  $\rightarrow$  overfits
- wrong split point  $\rightarrow$   $L$  high  $\rightarrow$  misses the jump
- good balance  $\rightarrow$  one split at the jump  $\rightarrow$  low  $L$  and low  $\text{omega}$
- same idea as pruning with  $\text{ccp\_alpha}$   $\rightarrow$  bss plus a price per leaf

### how it learns

- gradient descent needs numbers to nudge  $\rightarrow$  a tree is a structure of splits  $\rightarrow$  no gradient for which split to pick
- so  $\rightarrow$  additive training  $\rightarrow$  keep the trees already built fixed and add one new tree per round  
 $\hat{y}(0) = 0$   
 $\hat{y}(1) = \hat{y}(0) + f_1(x)$   
 $\hat{y}(t) = \hat{y}(t-1) + f_t(x)$
- each new tree is chosen to lower obj. the most given the trees already there

### leaf weights and splits

- not on the slides  $\rightarrow$  shows how  $\lambda$  and  $\gamma$  act
- per row  $\rightarrow$   $g$  = first derivative of the loss  $\rightarrow$   $h$  = second derivative
  - squared loss  $\rightarrow$   $g = \hat{y} - y$  and  $h = 1$
  - log loss  $\rightarrow$   $g = p - y$  and  $h = p(1 - p)$
- $G$  and  $H$  = sums of  $g$  and  $h$  over the rows in a leaf  
 best leaf weight  $w = -G / (H + \lambda)$ 
  - squared loss  $\rightarrow$   $w = \text{sum of residuals} / (\text{rows} + \lambda)$

- leaf with 1 row and residual 4.5  $\rightarrow \lambda = 0 \rightarrow 4.5 \rightarrow \lambda = 1 \rightarrow 2.25 \rightarrow \lambda$  pulls leaves toward 0
- gain =  $0.5 \times [G_L^2 / (H_L + \lambda) + G_R^2 / (H_R + \lambda) - G^2 / (H + \lambda)] - \gamma$
- keep a split only if gain  $> 0 \rightarrow \gamma$  = the least improvement a split must bring  $\rightarrow$  built in pruning

## hyperparameters

- n\_estimators  $\rightarrow$  rounds  $\rightarrow$  lab default 100
- learning\_rate  $\rightarrow$  shrink per tree  $\rightarrow 0.01$  to  $0.3 \rightarrow$  lower needs more trees
- max\_depth  $\rightarrow$  tree size  $\rightarrow 3$  to  $6$  typical
- gamma  $\rightarrow$  min gain per split  $\rightarrow$  higher = fewer splits
- reg\_lambda and reg\_alpha  $\rightarrow$  L2 and L1 on leaf weights
- subsample and colsample\_bytree  $\rightarrow$  share of rows and columns per tree  $\rightarrow$  forest style randomness
- early\_stopping\_rounds  $\rightarrow$  stop when the validation score stops improving
- scale\_pos\_weight  $\rightarrow$  weight for the positive class  $\rightarrow$  unbalanced data without SMOTE
- missing values  $\rightarrow$  no filling needed  $\rightarrow$  at each split XGBoost tries sending the missing rows left and then right and keeps the side with the higher gain  $\rightarrow$  that side = the default direction
- sklearn 1.4 and newer trees and forests also accept NaN  $\rightarrow$  older versions and many other models need filling first

## good and bad

- good  $\rightarrow$  high accuracy on tabular data  $\rightarrow$  handles missing values  $\rightarrow$  built in regularization  $\rightarrow$  scales to big data  $\rightarrow$  feature importance
- bad  $\rightarrow$  needs careful tuning  $\rightarrow$  can overfit without regularization  $\rightarrow$  weaker on high dimensional sparse data  $\rightarrow$  slow to train on large data  $\rightarrow$  hard to interpret
- tuning check on Pima 5 fold CV
  - default  $\rightarrow 0.736$
  - max\_depth 3 and learning\_rate 0.05 and 200 trees  $\rightarrow 0.764$
  - random forest default  $\rightarrow 0.761$
  - so untuned XGBoost lost to the forest and tuned it just beats it

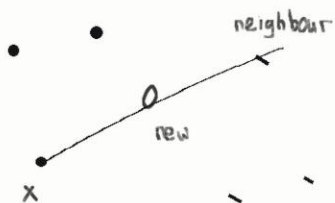
## smote

- in the lab and not on the slides
- unbalanced classes  $\rightarrow$  Pima 600 no and 268 diabetes  $\rightarrow$  the model leans to the big class
- SMOTE = Synthetic Minority Over-sampling Technique  $\rightarrow$  makes new

minority rows

$$\text{new} = x + u \times (\text{neighbour} - x)$$

- $x$  = a minority row  $\rightarrow$  neighbour = one of its 5 nearest minority rows  $\rightarrow$   
 $u$  = random 0 to 1
- = a random point on the line between two real minority rows

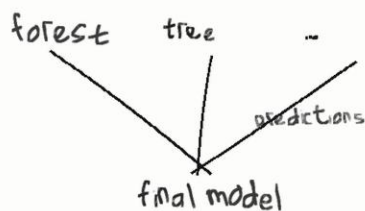


- apply to the training rows only  $\rightarrow$  the test set must stay like the real world
- Pima over 20 splits  $\rightarrow$  forest class 1 recall 0.57  $\rightarrow$  0.68 with SMOTE  $\rightarrow$  accuracy 0.763  $\rightarrow$  0.766
  - diabetes  $\rightarrow$  a missed patient costs more  $\rightarrow$  the trade is worth it
- other options  $\rightarrow$  `class_weight='balanced'` in sklearn  $\rightarrow$   
`scale_pos_weight` in XGBoost  $\rightarrow$  no fake rows needed
- weak spot  $\rightarrow$  new points between noisy or outlier rows land in the wrong class region

## stacking

- in the lab and not on the slides

base models predict  $\rightarrow$  a final model learns from those predictions how to combine them



- sklearn `StackingClassifier`  $\rightarrow$  base models predict on rows they were not trained on  $\rightarrow$  5 fold cross validation inside
  - why  $\rightarrow$  predictions on their own training rows look too good  $\rightarrow$  the final model would trust the wrong model
- then the base models are refit on all training rows for later use
- lab  $\rightarrow$  random forest and a tree as base and XGBoost as final  $\rightarrow$  accuracy 0.72  $\rightarrow$  no better than the forest alone 0.71 to 0.73
  - stacking helps when the base models make different kinds of mistakes  $\rightarrow$  two tree models make similar ones
- which error it lowers  $\rightarrow$  depends on the base models  $\rightarrow$  many high variance models  $\rightarrow$  mainly variance like bagging  $\rightarrow$  simple biased models of different kinds  $\rightarrow$  the final model can also fix bias by weighting the one that is right in each region

## comparing ensembles

|           | forest    | adaboost        | xgboost          |
|-----------|-----------|-----------------|------------------|
| trees     | parallel  | sequence        | sequence         |
| each tree | deep      | stump           | small            |
| rows      | bootstrap | reweighted      | all or subsample |
| fits      | labels    | weighted labels | gradients        |
| lowers    | variance  | bias            | bias             |

- forest  $\rightarrow$  good default with little tuning  $\rightarrow$  noisy data  $\rightarrow$  want OOB score
- boosting  $\rightarrow$  want the best score and can tune  $\rightarrow$  clean labels
- stacking  $\rightarrow$  base models of different kinds that make different mistakes
- cost  $\rightarrow$  forest trees build in parallel  $\rightarrow$  boosting trees wait for each other  $\rightarrow$  prediction is fast for all

## exam check

- forest steps  $\rightarrow$  bootstrap  $\rightarrow$  tree per sample with random features per split  $\rightarrow$  vote
- two kinds of diversity  $\rightarrow$  bagging and feature randomness  $\rightarrow$  why  $\rightarrow$  lower correlation
- OOB  $\rightarrow$  rows a tree never saw  $\rightarrow$  free test score
- feature importance  $\rightarrow$  impurity drop summed over trees  $\rightarrow$  use to drop features
- bagging vs boosting  $\rightarrow$  parallel and variance vs sequence and bias
- adaboost round by hand  $\rightarrow$  err  $\rightarrow$   $\alpha$   $\rightarrow$  reweight  $\rightarrow$  normalise
- gradient boosting  $\rightarrow$  each tree fits residuals  $\rightarrow$  learning rate scales the step
- XGBoost objective  $\rightarrow$  loss + omega  $\rightarrow$  omega = gamma \* T + 0.5 lambda  $\sum w^2$   $\rightarrow$  why  $\rightarrow$  stop trees fitting noise
- why not gradient descent on trees  $\rightarrow$  splits are not numbers  $\rightarrow$  additive training
- SMOTE  $\rightarrow$  new point between two minority rows  $\rightarrow$  train only
- which model  $\rightarrow$  see comparing ensembles
- mix ups
  - bootstrap = rows with replacement and feature randomness = columns per split
  - OOB error  $\neq$  test error  $\rightarrow$  close but computed on training rows each tree skipped
  - learning rate in boosting  $\neq$   $\alpha$  in adaboost
  - gamma = price per leaf and lambda = shrink on leaf values

## conceptual questions

- why a forest beats one tree  $\rightarrow$  averaging trees with different mistakes cancels variance  $\rightarrow$  bias stays the same
- what makes a good forest  $\rightarrow$  strong trees and low correlation between them
- bagging  $\rightarrow$  each tree on  $N$  rows drawn with replacement  $\rightarrow$  about  $1/3$  left out
- feature randomness  $\rightarrow$  new random subset at every split  $\rightarrow$  stops all trees using the same top feature
- why more trees do not overfit  $\rightarrow$  they only shrink the variance part  $\rightarrow$   $c \times v$  stays
- OOB error  $\rightarrow$  vote of the trees that skipped a row  $\rightarrow$  no validation set needed  $\rightarrow$  Pima 0.766 vs CV 0.761
- feature importance  $\rightarrow$  impurity drop per feature over all trees  $\rightarrow$  Pima Glucose first
- forest weak spots  $\rightarrow$  slower and less readable than one tree  $\rightarrow$  deep trees still fit label noise
- adaboost  $\rightarrow$  reweight missed rows  $\rightarrow$   $\alpha = 0.5 \ln((1 - \text{err}) / \text{err})$   $\rightarrow$  sum of weighted stumps
- adaboost vs gradient boosting  $\rightarrow$  reweight rows vs fit residuals
- gradient boosting  $\rightarrow$  each tree fits  $y - \hat{y} \rightarrow \hat{y} += \text{eta} \times \text{tree} \rightarrow$  residuals shrink
- XGBoost vs plain gradient boosting  $\rightarrow$  regularized objective and second derivatives and missing value handling and speed
- omega  $\rightarrow$  gamma per leaf and lambda on leaf weights  $\rightarrow$  stops trees from chasing every point
- why additive training  $\rightarrow$  a tree is not a vector of numbers  $\rightarrow$  add one tree at a time
- learning rate  $\rightarrow$  small steps  $\rightarrow$  needs more trees  $\rightarrow$  overfits less
- early stopping  $\rightarrow$  stop adding trees when validation stops improving
- XGBoost weak spots  $\rightarrow$  tuning needed and overfits if not regularized and hard to read
- SMOTE  $\rightarrow$   $x + u(\text{neighbour} - x)$   $\rightarrow$  only on train  $\rightarrow$  raises minority recall
- SMOTE vs class weights  $\rightarrow$  fake rows vs heavier errors on real rows
- stacking  $\rightarrow$  final model trained on out of fold predictions of the base models
- when stacking helps  $\rightarrow$  base models of different kinds
- forest vs boosting choice  $\rightarrow$  forest for little tuning or noisy labels  $\rightarrow$  boosting for the top score with tuning
- baseline on Pima  $\rightarrow$  always no diabetes  $\rightarrow$  0.661  $\rightarrow$  models add about 0.1

lab code

- Pima Indian diabetes → 768 patients and 8 features and Outcome ↓  
= diabetes
- the notebook keeps the outputs but most code cells are empty →  
the calls below fill them

## load and check

```
url = "https://raw.githubusercontent.com/trdsaputr1/
sample/refs/heads/main/pima_indian_dataset"
data = pd.read_csv(url)
data.info()
data.isnull().sum()
```

- 1 `pd.read_csv` → reads the file into a table
- 2 `info` → 768 rows → a few columns have 766 or 767 non null
- 3 `isnull().sum()` → missing per column → Pregnancies 1 Glucose 1  
SkinThickness 1 BMI 1

## fill missing

```
data = data.fillna(data.median())
data.describe()
```

- 1 `fillna` → replaces each missing value → median of its column is one  
choice → the cell is empty in the notebook
  - 2 `describe` → checks the result → min of Glucose and BMI and Insulin is  
0
- Lab Correction → 0 means missing in this dataset → Insulin 374  
zeros and SkinThickness 226 and BloodPressure 35 and BMI 11 and  
Glucose 5 → nobody has BMI 0 → replace 0 with NaN then fill

## split

```
train, test = train_test_split(data, test_size = 0.3,
random_state = 32)
```

```
X_train = train.drop(columns='Outcome')
```

```
y_train = train['Outcome']
```

```
X_test = test.drop(columns='Outcome')
```

```
y_test = test['Outcome']
```

- 1 `train_test_split` on the whole table → 537 train and 231 test
  - 2 `drop` → features and ['Outcome'] → target
- class counts → test 147 no and 84 yes

## SMOTE

```
from imblearn.over_sampling import SMOTE
```

```
smote = SMOTE(random_state=42)
```

```
X_train_balanced, y_train_balanced = smote.fit_resample(X_train,
y_train)
```

- 1 `SMOTE` → the oversampler → `random_state` fixes the new points
- 2 `fit_resample` → adds synthetic diabetes rows until both classes

match → train only

## random forest

```
randforest = RandomForestClassifier()
randforest.fit(X_train_balanced, y_train_balanced)
prediction = randforest.predict(X_test)
print(classification_report(y_test, prediction))
```

- 1 RandomForestClassifier → 100 trees default
  - 2 fit on the balanced train
  - 3 predict on the real untouched test
  - 4 classification\_report → per class precision recall F1
- lab → accuracy 0.71 → class 1 recall 0.60 and F1 0.60

## tuning

```
param_grid = {'n_estimators': [25, 50, 75, 100, 200],
 'max_depth': [None, 5, 10, 20],
 'max_features': ['sqrt', 'log2']}
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(rf, param_grid, cv=5)
grid_search.fit(X_train_balanced, y_train_balanced)
grid_search.best_params_
y_pred = grid_search.best_estimator_.predict(X_test)
```

- 1 param\_grid →  $5 \times 4 \times 2 = 40$  combinations
  - 2 GridSearchCV → trains every combination with 5 fold CV → 200 fits
  - 3 best\_params\_ → max\_depth None and max\_features sqrt and n\_estimators 50
  - 4 best\_estimator\_ → the winning model re-fit on all of train
- lab → accuracy 0.73 → class 1 recall 0.60 and F1 0.62
- Lab Correction → SMOTE runs before the grid search → the CV folds hold synthetic rows built from rows in other folds → CV scores look better than they are → put SMOTE inside an imblearn Pipeline so it runs on each training fold only

## XGBoost

```
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(X_train_balanced, y_train_balanced)
y_pred = xgb.predict(X_test)
```

- 1 XGBClassifier → the printed setup shows 100 trees and learning\_rate 0.1 → the default with nothing set is 0.3
  - 2 fit → adds trees one by one on the balanced train
  - 3 predict → sums the trees then thresholds the probability at 0.5
- lab → accuracy 0.73 → class 1 recall 0.64 and F1 0.64 → best recall

of the lab

## stacking

```
estimators = [
 ('rf', RandomForestClassifier(n_estimators=10,
 random_state=42)),
 ('tree', make_pipeline(StandardScaler(),
 tree.DecisionTreeClassifier()))]
```

```
clf = StackingClassifier(estimators = estimators,
 final_estimator = XGBClassifier())
clf.fit(X_train_balanced, y_train_balanced)
y_pred_stack = clf.predict(X_test)
```

1 estimators → list of name and model pairs → a 10 tree forest and a scaled tree

2 make\_pipeline → runs StandardScaler then the tree as one model

3 StackingClassifier → base models feed out of fold predictions to final\_estimator

4 fit → 5 fold inside for the base predictions then refit

5 predict → base models predict then XGBoost combines

- lab → accuracy 0.72 → class 1 recall 0.58 and F1.060

- Lab Correction → StandardScaler before a tree does nothing → trees ignore scale

## what the lab scores mean

the lab split is a hard one → over 20 splits the forest with SMOTE averages 0.756 → ranges 0.706 to 0.792

- baseline always no = 0.651 → every model adds about 0.1

- the gaps between the lab models 0.71 to 0.73 are smaller than the split to split spread → one split can not rank them

## good practice

- not on the slides

- treat 0 as missing in Glucose BloodPressure SkinThickness Insulin BMI

- split with stratify=y

- SMOTE inside the CV loop → imblearn Pipeline

from imblearn.pipeline import Pipeline

Pipeline([('smote', SMOTE()), ('rf', RandomForestClassifier())])

- compare models by cv mean and spread and by class 1 recall

- XGBoost → tune max\_depth and learning\_rate with early stopping

## comparing the models

## which model when

| model    | lowers   | scaling   | read it    |
|----------|----------|-----------|------------|
| tree     | nothing  | no        | the rules  |
| forest   | variance | no        | importance |
| adaboost | bias     | no        | importance |
| xgboost  | bias     | no        | importance |
| stacking | depends  | per model | hard       |

- one tree → you must explain every decision → small data → a quick first model
- Random forest → a strong default with little tuning → noisy labels → want a free OOB score
- adaboost → weak simple learners and clean labels
- xgboost → the best score on tabular data when you can tune → missing values left in
- stacking → base models of different kinds that make different mistakes

## how each one is trained and scored

- tree → greedy splits by gini or entropy → prune by depth or  $ccp\_alpha$
- forest → many deep trees on bootstrap rows with random features per split → vote
- adaboost → stumps in sequence on reweighted rows → weighted vote
- xgboost → trees in sequence on gradients with a penalty per leaf → sum
- all of them → score with the confusion matrix and precision recall F1 → compare by CV mean against the baseline