

week 5

classification

- supervised task that predicts a class label for a row

- malignant or benign $\rightarrow$ classification cuz the answer is a label
- tumour size in mm $\rightarrow$ regression cuz the answer is a number
- training set = N samples and each one has features x and a label r
 $X = \{ (x_t, r_t) \} \text{ for } t = 1 \dots N$
 $r = 1$ if positive sample
 $r = 0$ if negative sample
- $x = x_1 x_2 \dots x_n$ = input variables = features = the columns
 - breast cancer table $\rightarrow$ radius mean and texture mean ... compactness mean
- $r = y$ = target = output variable
- positive = the class we're looking for and not the good one
 - malignant = positive and benign = negative
 - spam = positive and normal mail = negative
- flow $\rightarrow$ training set goes into the learning algorithm $\rightarrow$ it outputs a hypothesis $h(x) \rightarrow h(x)$ maps a new row's features to a target

hypothesis

- hypothesis h = one guessed rule that splits positive from negative
- slide example on 2 features $\rightarrow$ texture mean x_2 and compactness mean x_5
 $(te_1 \leq \text{texture} \leq te_2) \text{ AND } (co_1 \leq \text{compactness} \leq co_2)$
- inside the box $\rightarrow$ malignant and outside $\rightarrow$ benign
 $h(x) = 1$ if h says x is positive
 $h(x) = 0$ if h says x is negative
- H = hypothesis class = every box we could draw $\rightarrow$ learning = pick the best h out of H
- error of h = how many training rows it gets wrong
 $E(h|X) = \sum 1(h(x_t) \neq r_t)$
- $1(\dots)$ counts 1 when the statement is true and 0 when false
 - 10 rows and the box misplaces 2 $\rightarrow E = 2$
- many boxes give $E = 0$ on train $\rightarrow$ we must pick the one that works on new rows too $\rightarrow$ generalization

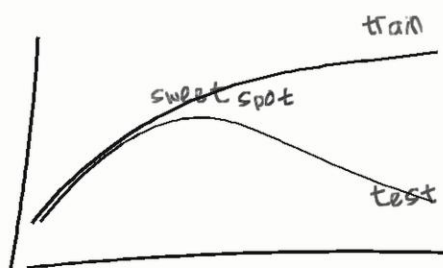


errors in hypothesis

- C = the true class $\rightarrow$ the real box and h = our box
- false positive = actually negative but predicted positive
 - benign patient told they have malignant cancer
 - extra tests and stress but nobody is harmed long term
- false negative = actually positive but predicted negative
 - malignant patient told they are fine
 - goes home untreated $\rightarrow$ the worse mistake in medicine
- true positive and true negative = predicted right
- S = most specific box = tightest box around the + points
- G = most general box = biggest box that still holds no - point
- any box between S and G has zero training error
- margin = distance between S and G
 - best h sits halfway $\rightarrow$ a new point that lands a bit off still gets the right label
 - bigger margin = more room for noise = better generalization
- CAREFUL the slide writes FP as $C = 1$ but $h = 0$ and FN as $C = 0$ but $h = 1$
- the right way $\rightarrow$ FP = $C = 0$ and $h = 1$ and FN = $C = 1$ and $h = 0$
- if asked $\rightarrow$ use the words actually negative predicted positive

overfitting vs underfitting

- more complex model $\rightarrow$ better on training data every time
- too complex $\rightarrow$ it memorises the training rows and noise $\rightarrow$ fails on new rows $\rightarrow$ lost generalization
 - tree grown until every leaf has 1 row $\rightarrow$ 100% on train and worse on test
- too simple $\rightarrow$ misses the real pattern $\rightarrow$ bad on train and test $\rightarrow$ underfit
 - tree with 1 split on 3 classes $\rightarrow$ can only ever name 2 of them
- sweet spot = where test accuracy peaks



- three factors $\rightarrow$ complexity of the model and amount of training data and the generalization error on new data
- more data helps cuz it brings variety $\rightarrow$ more cases the model must fit at once
- duplicating rows does not help $\rightarrow$ no new information and it just weighs those rows double

- how to spot it → compare train accuracy with test accuracy
 - train 1.00 and test 0.72 → overfit
 - train 0.60 and test 0.58 → underfit

confusion matrix

- table of predicted vs actual counts
- slide layout → rows = predicted and columns = actual

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

- CAREFUL sklearn confusion_matrix puts actual on the rows and predicted on the columns → the opposite way
- always check which axis is which before reading off FP and FN

metrics

- example → 100 patients and 10 have malignant cancer
- model flags 8 as malignant and 6 of them really are
- TP 6 and FP 2 and FN 4 and TN 88

accuracy

$$\text{accuracy} = (TP + TN) / \text{total}$$

$$= (6 + 88) / 100 = 0.94$$

- share of all predictions that are right
- trap → a model that says benign for everyone gets $90 / 100 = 0.90$ and finds zero cancers
- so accuracy lies when one class is rare

precision

$$\text{precision} = TP / (TP + FP)$$

$$= 6 / 8 = 0.75$$

- of the ones I flagged positive how many really are
- matters when a false alarm is costly
 - spam filter → a real mail sent to spam is bad → want high precision

recall or sensitivity

$$\text{recall} = TP / (TP + FN)$$

$$= 6 / 10 = 0.60$$

- of the real positives how many did I catch
- matters when missing one is costly
 - cancer screening → want high recall

specificity

$$\text{specificity} = \text{TN} / (\text{TN} + \text{FP})$$

$$= 88 / 90 = 0.98$$

- of the real negatives how many did I clear correctly
- = recall of the negative class
- CAREFUL the slide writes $\text{TN} / (\text{TN} + \text{FN})$
- that is wrong cuz FN rows are actual positives $\rightarrow$ specificity uses FP

F1

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

$$= 2 \times 0.75 \times 0.60 / 1.35 = 0.67$$

- harmonic mean of precision and recall $\rightarrow$ between 0 and 1 and higher is better
- why harmonic and not the normal mean $\rightarrow$ it drops hard if either one is low
 - $P = 1.0$ and $R = 0.1 \rightarrow$ normal mean 0.56 but $\text{F1} = 0.18$
 - so F1 is high only when both are high
- precision and recall pull against each other
 - flag more patients $\rightarrow$ recall up and precision down
- CAREFUL slide says higher F1 means higher accuracy $\rightarrow$ F1 and accuracy are different numbers
 - the 90% always benign model $\rightarrow$ accuracy 0.90 and F1 0

multiclass metrics

- more than 2 classes $\rightarrow$ treat each class as positive vs all the rest and compute the binary metrics per class
- accuracy = correct / total = diagonal / everything
- 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 = $(19 + 12 + 13) / 45 = 44 / 45 = 0.98$
- per class in sklearn layout $\rightarrow$ precision = diagonal / its column and recall = diagonal / its row
 - versicolor precision = $12 / 13 = 0.92$ cuz 1 virginica got called versicolor
 - versicolor recall = $12 / 12 = 1.00$
 - virginica precision = $13 / 13 = 1.00$ and recall = $13 / 14 = 0.93$
- macro avg = plain mean of the per class scores $\rightarrow$ every class counts the same
 - macro P = $(1.00 + 0.92 + 1.00) / 3 = 0.97$
- weighted avg = mean weighted by support $\rightarrow$ big classes count more

- support = how many actual rows that class has in the test set →
19 12 14
- macro much lower than weighted → the model is bad on a small class
 - Orthopedic lab → Hernia recall 0.33 drags macro recall to 0.66

log loss

- for models that output a probability $\hat{y}$ between 0 and 1 instead of a hard label
- $\log \text{loss} = -(1/n) \sum [y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$
- only one term is alive per row
 - $y = 1 \rightarrow \text{cost} = -\log(\hat{y})$
 - $y = 0 \rightarrow \text{cost} = -\log(1 - \hat{y})$
- $y = 1$ and $\hat{y} = 0.9 \rightarrow -\log(0.9) = 0.11 \rightarrow$ confident and right → small cost
- $y = 1$ and $\hat{y} = 0.5 \rightarrow 0.69 \rightarrow$ a coin flip
- $y = 1$ and $\hat{y} = 0.1 \rightarrow 2.30 \rightarrow$ confident and wrong → huge cost
- $y = 1$ and $\hat{y} = 0.01 \rightarrow 4.61 \rightarrow$ keeps growing toward ∞
- lower = better and 0 = perfect
- why use it → accuracy only sees right or wrong and log loss also sees how sure the model was
- CAREFUL the slide formula has $(1 - y)(1 - \hat{y})$ with no log → the second term needs $\log(1 - \hat{y})$
- CAREFUL the slide says $0 \leq \log \text{loss} \leq 1 \rightarrow$ really 0 to ∞ cuz $-\log(\hat{y})$ has no top

three decisions for any model

- data $X = \{x_t, r_t\}$ is iid = independent and identically distributed
 - independent → one row does not affect another
 - identically distributed → every row comes from the same population as the future rows
 - this is why a model learned on train can work on test at all
- to approximate r_t with $g(x_t, \theta)$ we choose three things
- model $g(x|\theta) \rightarrow$ the shape of the rule and θ = its parameters
- loss function $L \rightarrow$ how bad one prediction is
 $E(\theta|X) = \sum L(r_t, g(x_t, \theta))$
- optimisation → the method that finds the θ with the lowest total error
 $\theta^* = \text{argmin over } \theta \text{ of } E(\theta|X)$
- argmin = the θ that gives the smallest E and not the smallest E itself
- week 2 linear regression → model $\theta_0 + \theta_1 x$ and loss squared error and optimisation least squares
- decision tree → model a tree of tests and loss impurity → entropy or

gini and optimisation greedy search for the best split at each node

- hypothesis box $\rightarrow$ model rectangle and loss count of wrong rows and optimisation pick the box between S and G

decision tree

- decision tree induction = learning a tree from rows that all have class labels
- each internal node = a test on one attribute $\rightarrow$ eg age
- each branch = one outcome of that test $\rightarrow$ youth or middle age or senior
- each leaf = a class label $\rightarrow$ buy yes
- top node = root
- predicting = start at the root and follow the branch that matches the row until a leaf
- why it works $\rightarrow$ each test splits the rows into groups that are more pure than before $\rightarrow$ closer to one class each
- keep splitting until each group is one class $\rightarrow$ the path to the leaf is the rule
 - like the game 20 questions where each question is picked to cut the possibilities the most $\rightarrow$ analogy

history

- ID3 = Iterative Dichotomiser $\rightarrow$ uses information gain
- C4.5 $\rightarrow$ successor of ID3 $\rightarrow$ uses gain ratio
- CART = Classification and Regression Tree $\rightarrow$ uses gini $\rightarrow$ binary splits only $\rightarrow$ also does regression
- sklearn DecisionTreeClassifier = an optimised CART $\rightarrow$ default criterion gini

learning process

- start with data partition D and the attribute list and an attribute selection method
- make node N for D
- all rows in D the same class $\rightarrow$ N becomes a leaf with that class $\rightarrow$ stop
- else $\rightarrow$ run the selection method on every attribute $\rightarrow$ pick the best splitting criterion
- label N with that test and grow one branch per outcome
- repeat the same steps on each branch with only its rows $\rightarrow$ recursion
- stop when a stopping rule is met and return the tree
- stopping rules

- all rows in the node are one class
- no attributes left $\rightarrow$ leaf takes the majority class
- no rows reach that branch $\rightarrow$ leaf takes the majority class of the parent
- a pre pruning limit is hit $\rightarrow$ eg max depth
- the tree is greedy $\rightarrow$ takes the best split right now and never goes back
- fast but the final tree is not guaranteed to be the best possible tree

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	?

- 7 rows $\rightarrow$ buy 4 and no 3
- young 2 $\rightarrow$ buy 1 and no 1
- middle 2 $\rightarrow$ buy 2 and no 0
- senior 3 $\rightarrow$ buy 1 and no 2
- student yes 1 $\rightarrow$ buy 1 and student no 6 $\rightarrow$ buy 3 and no 3
- excellent 3 $\rightarrow$ buy 3 and fair 4 $\rightarrow$ buy 1 and no 3

entropy and information gain

- entropy = how mixed or random the classes in a group are

$$\text{Info}(D) = -\sum p \log_2(p) \text{ over each class}$$
- p = share of rows in that class
- meaning = the average number of yes or no questions needed to tell the class of a random row
 - all one class $\rightarrow$ 0 bits $\rightarrow$ nothing to ask
 - 50 50 $\rightarrow$ 1 bit $\rightarrow$ the max for 2 classes
- calculator $\rightarrow \log_2(x) = \log(x) / \log(2)$
- $0 \times \log(0)$ counts as 0
- after splitting on attribute A into v groups

$$\text{Info}_A(D) = \sum (|D_j| / |D|) \times \text{Info}(D_j)$$
- = weighted average entropy of the groups $\rightarrow |D_j| / |D|$ = share of rows that go to group j
- information gain = entropy before minus entropy after

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

- pick the attribute with the highest gain = the one that removes the most randomness

worked

$$\text{Info}(D) = -(4/7)\log_2(4/7) - (3/7)\log_2(3/7) = 0.985$$

- age

$$\text{Info}_{\text{age}} = (2/7)(1.0) + (2/7)(0) + (3/7)(0.918) = 0.286 + 0 + 0.394 = 0.679$$

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

- student

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

$$\text{Gain}(\text{student}) = 0.985 - 0.857 = 0.13$$

- credit → not on the slides

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

$$\text{Gain}(\text{credit}) = 0.985 - 0.464 = 0.52$$

- credit has the highest gain → credit is the root

entropy lookup

counts	entropy	gain
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	0.985	0.490
2 5	0.863	0.408
6 9	0.940	0.459
4 0	0	0

gain ratio

- problem → information gain favours attributes with many values
 - split on customer id → 7 groups of 1 row → each pure → Info = 0 → gain = 0.985 = the max
 - but id says nothing about a new customer → useless split
- C4.5 fixes it by dividing by the split info = the entropy of the group sizes themselves

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

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

- many small groups → big split info → ratio pushed down

worked

$$\text{SplitInfo}(\text{age}) = -(2/7)\log_2(2/7) - (2/7)\log_2(2/7) - (3/7)\log_2(3/7) = 1.557$$

$$\text{GainRatio}(\text{age}) = 0.31 / 1.557 = 0.20$$

slide prints 1.52 $\rightarrow$ the real value is 1.657 $\rightarrow$ the ratio still rounds to 0.20

- customer id $\rightarrow$ SplitInfo = $\log_2(7) = 2.81 \rightarrow$ ratio = $0.985 / 2.81 = 0.35$
- credit $\rightarrow$ SplitInfo = entropy of 3 and 4 = 0.985 $\rightarrow$ ratio = $0.62 / 0.985 = 0.53 \rightarrow$ still wins

gini index

- gini = impurity of D
 $Gini(D) = 1 - \sum p^2$ over each class
- meaning = chance that 2 rows picked at random from the group have different classes
 - pure group $\rightarrow 0$
 - 50 50 $\rightarrow 0.5 \rightarrow$ the max for 2 classes
- CART only does binary splits $\rightarrow$ an attribute with 3 values has 3 ways to make 2 groups
 $Gini_A(D) = (D_1/V(D)) Gini(D_1) + (D_2/V(D)) Gini(D_2)$
- pick the split with the lowest $Gini_A$ = purest groups
reduction = $Gini(D) - Gini_A(D)$
- highest reduction = same pick as lowest $Gini_A$

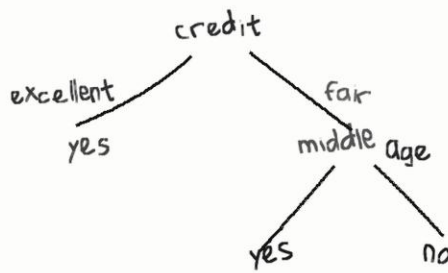
worked

- $Gini(D) = 1 - (4/7)^2 - (3/7)^2 = 0.49$
- age split 1 $\rightarrow$ young vs middle and senior
 $(2/7)(0.5) + (5/7)(1 - (3/5)^2 - (2/5)^2)$
 $= 0.143 + 0.343 = 0.486$
- age split 2 $\rightarrow$ senior vs young and middle
 $(3/7)(0.444) + (4/7)(0.375) = 0.405$
- age split 3 $\rightarrow$ middle vs young and senior
 $(2/7)(0) + (5/7)(0.48) = 0.343$
- best age split = middle vs others at 0.343
reduction = $0.49 - 0.343 = 0.15$
- CAREFUL slide writes $Gini(D) = 0.51$ and young vs others = 0.62 $\rightarrow$ the right values are 0.49 and 0.486
- its 0.17 reduction comes from the wrong 0.51
- student $\rightarrow (6/7)(0.5) = 0.429$
- credit $\rightarrow (4/7)(0.375) = 0.214 \rightarrow$ lowest $\rightarrow$ credit wins with gini too
- gini vs entropy $\rightarrow$ nearly always pick the same split $\rightarrow$ gini is faster cuz there is no log

the finished tree

- root = credit
- excellent $\rightarrow$ rows 1 3 4 $\rightarrow$ all buy $\rightarrow$ leaf yes
- fair $\rightarrow$ rows 2 5 6 7 $\rightarrow$ 1 buy and 3 no $\rightarrow$ split again

- age middle $\rightarrow$ row 7 $\rightarrow$ yes and the others $\rightarrow$ rows 2 5 6 $\rightarrow$ no $\rightarrow$ both pure $\rightarrow$ stop



- customer 8 = middle age and student and excellent $\rightarrow$ credit excellent $\rightarrow$ buy yes
- the slide tree has age as root $\rightarrow$ that tree comes from a bigger textbook dataset and not from these 7 rows

numbers as attributes

- not on the slides
- numeric column $\rightarrow$ sort the values and try a threshold between each pair of neighbours
- keep the threshold with the best gain or gini
 - Iris $\rightarrow$ petal length ≤ 2.45 splits off every setosa in one test
- this is why trees do not need scaling $\rightarrow$ only the order of values matters and scaling keeps the order

tree pruning

- a full tree also learns the noise and outliers in train $\rightarrow$ branches that fit one odd row
- pruning = removing the least reliable branches $\rightarrow$ smaller and simpler tree $\rightarrow$ easier to read and better on new rows

pre pruning

- stop growing early $\rightarrow$ when a test says the split is not worth it the node becomes a leaf
- the leaf holds the majority class or the class probabilities of its rows
 - leaf with 8 yes and 2 no $\rightarrow$ predicts yes or says $P(\text{yes}) = 0.8$
- sklearn knobs $\rightarrow$ max_depth and min_samples_split and min_samples_leaf and min_impurity_decrease
- cheap but can stop too early $\rightarrow$ a split that looks weak may lead to a strong one below it

post pruning

- grow the full tree first then cut subtrees off from the bottom
- a cut subtree is replaced by a leaf labelled with the majority class of the rows under it

- keep the cut if accuracy on held out data does not drop
- sklearn $\rightarrow$ ccp_alpha = cost complexity $\rightarrow$ bigger alpha = more cutting
- slower but usually better than pre pruning cuz it sees the whole tree before deciding
- like week 2 regularization $\rightarrow$ both trade some train fit for simpler models that generalize

why trees are good

- the learning objectives list benefits but the slides never do
- readable $\rightarrow$ the path is an if then rule you can explain to a doctor
- no scaling needed and handles numbers and categories
- fast to predict $\rightarrow$ only depth many tests
- finds the important features on its own $\rightarrow$ feature_importances_
- weak points
 - overfits easily without pruning
 - unstable $\rightarrow$ change a few rows and the whole tree can change
 - splits are straight cuts along one feature $\rightarrow$ a diagonal border needs many steps
 - greedy so not the best possible tree

random forest and adaboost

- these are in the lab $\rightarrow$ the week 6 ensemble slides are not up yet
- ensemble = many models vote and the group beats one model

random forest

- builds many trees $\rightarrow$ 100 by default
- each tree trains on a bootstrap sample = N rows drawn with replacement from the N training rows
- at each split a tree only looks at a random few features
- final answer = majority vote of all trees
- why it works $\rightarrow$ one deep tree has high variance = unstable $\rightarrow$ trees trained on different rows make different mistakes $\rightarrow$ the vote cancels them out
 - like asking 100 people instead of 1 $\rightarrow$ individual mistakes average away $\rightarrow$ analogy
- = bagging + random features

adaboost

- trains small trees one after another $\rightarrow$ stumps = trees with 1 split by default
- after each tree the rows it got wrong get more weight
- the next tree is forced to focus on those hard rows

- final answer = weighted vote → more accurate trees get a bigger say

why it works → each weak tree is barely better than guessing but each fixes what the last missed

- bagging lowers variance and boosting lowers bias

	random forest	adaboost
trees	parallel	in sequence
each tree	deep	stump
fixes	variance	bias

- lab orthopedic → tree 0.72 and forest 0.82 and adaboost 0.78

exam check

- confusion matrix given → find TP FP FN TN first then plug into the formulas
- check the layout → which axis is actual
- scenario → which error is worse
 - disease or fraud → FN worse → recall
 - spam or ads → FP worse → precision
- accuracy high but useless → rare class → look at recall or F1
- compute entropy or info gain → write the class counts per group first then use the lookup
- gain ratio question → it is the fix for attributes with many values
- gini question → binary splits so list every 2 group option
- build the tree → best attribute at the root then repeat on each impure branch
- overfit or underfit → train vs test accuracy
- pre vs post pruning → stop early vs cut after
- read a classification report → support = row count and macro vs weighted
- log loss → lower better → punishes confident mistakes
- mix ups
 - precision = column of predicted positive and recall = row of actual positive
 - specificity = recall of the negative class
 - gain → pick highest and gini → pick lowest
 - ID3 = gain and C4.5 = gain ratio and CART = gini
 - FP = false alarm and FN = missed case
 - bagging = parallel vote and boosting = sequence that fixes mistakes

lab code

- Iris (150 flowers and 4 measurements and 3 species) then 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 → nicer statistical plots on top of matplotlib

load

```
url = "https://raw.githubusercontent.com/
elflin/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 would only help memorise rows
- 3 head → shows the 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 and Species is object = text
- 2 describe → count and mean and std and min and quartiles and max per numeric column

3 countplot → bar of how many rows per class → Iris is 50 50 50 so balanced

4 title and show → label the plot and draw it

- why check the class counts → an unbalanced target makes accuracy misleading

Orthopedic → Spondylolisthesis 150 and Normal 100 and Hernia 60 → not balanced

```
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=[15,16], diagonal='hist')  
sns.pairplot(data=data, hue='Species')
```

- 1 color_list → one colour per row picked from its species
- 2 data.loc[:, data.columns != 'Species'] → every row and every column except Species
- 3 scatter_matrix → a grid of scatter plots of every feature against every other and histograms on the diagonal
- 4 pairplot → the same grid in seaborn coloured by hue
 - what to read → petal length and petal width split setosa off clearly → 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 = the Species column
- 2 MinMaxScaler → a tool that 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 are lost and 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 → 30% test → Iris 105 train and 45 test and 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 → defaults criterion gini and no depth limit → so no pruning

2 fit → grows the tree on the training rows using the split search above
3 predict → runs each test row down the tree → returns a species per row

4 format → puts the array into the printed text

5 score → accuracy on the test rows = predict then compare with y_{test}

- lab Iris → 0.978 → 44 of 45

- lab Orthopedic → 0.72

plot the tree

```
plt.figure(figsize=(15,15))
```

```
tree.plot_tree(dtrees)
```

```
plt.show()
```

```
import graphviz
```

```
dot_data = tree.export_graphviz(dtrees, 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 → a 15 by 15 inch canvas so the tree fits

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 instead of saving a file

4 feature_names → real column names since X_scaled lost them

5 class_names → names for the classes in the boxes

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

7 graphviz.Source → renders the DOT text as a picture

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

- CAREFUL class_names=y.unique() lists classes in order of first

appearance but sklearn orders them alphabetically → dtrees.classes_

Iris → both orders are the same by luck

- Orthopedic → unique gives Hernia Spondylolisthesis Normal and sklearn uses Hernia Normal Spondylolisthesis → Normal and Spondylolisthesis are swapped in the picture

- fix → class_names=dtrees.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 → counts with rows = actual and columns = predicted → classes in alphabetical order

2 heatmap → colours the matrix and annot=True writes the numbers in the cells

3 classification_report → precision and recall and F1 and support per class plus accuracy and macro avg and weighted avg

- lab Iris → only 1 mistake → one virginica called versicolor

- lab orthopedic treematrix → 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

```
from sklearn.ensemble import RandomForestClassifier
```

```
randforest = RandomForestClassifier()
```

```
randforest.fit(X_train, y_train)
```

```
prediction = randforest.predict(X_test)
```

```
randforest.score(X_test, y_test)
```

```
tree.export_graphviz(randforest.estimators_[1], ...)
```

1 RandomForestClassifier → 100 trees by default each on a bootstrap sample

2 fit and predict and score → same calls as the tree → predict is the majority vote

3 estimators → the list of fitted trees → [1] picks the second one to draw cuz all 100 cannot be shown

- lab → Iris 0.978 and Orthopedic 0.82

adaboost

```
from sklearn.ensemble import AdaBoostClassifier
```

```
adaBoost = AdaBoostClassifier(n_estimators=100)
```

```
adaBoost.fit(X_train, y_train)
```

```
prediction = adaBoost.predict(X_test)
```

```
adaBoost.score(X_test, y_test)
```

1 AdaBoostClassifier → boosted stumps → n_estimators=100 = up to 100 rounds → default 50

- 2 fit → trains stumps one by one reweighting the missed rows
- 3 predict → weighted vote of the stumps
- lab → Iris 1.00 and Orthopedic 0.78 → default 50 rounds there

exercise 2

- same calls on the Orthopedic file with the target column `class_status`
- the notebook keeps the outputs but the code cells are empty → fill them by copying exercise 1 and changing `url` and `Species` → `class_status`
- CAREFUL the tree and forest have no `random_state` → each run gives a slightly different score
Iris tree on the same split gave 1.00 or 0.978 across reruns
- CAREFUL scaling does nothing for trees → same split order → over 60 splits the Orthopedic tree averages 0.788 with or without `MinMaxScaler`
- CAREFUL the scaler is fit on all rows before the split → test rows leak into the min and max → harmless for trees but wrong for scaled models
- CAREFUL the 0.72 is one split → over 60 random splits the Orthopedic tree ranges 0.70 to 0.88
- CAREFUL `countplot` with `palette` and no `hue` gives a `FutureWarning` → write `sns.countplot(x="Species", hue="Species", data=data)`

doing it properly

- not on the slides
- split with `stratify=y` so each class keeps its share in train and test
`train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)`
- set `random_state` on the model too
`tree.DecisionTreeClassifier(random_state=0)`
- skip the scaler for trees and forests
- prune by searching the knobs with 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
- forest 0.86 and adaboost 0.80 by the same CV → the forest wins
- compare models by CV mean and not by one test split
- unbalanced classes → report macro F1 and not only accuracy
- `class_weight='balanced'` → makes missed small classes like Hernia count more