Introduction : types of machine learning

① supervised learning

goal: to use a labelled dataset to produce a model that takes feature vector $\underset{\sim}{x}$ as an input and outputs information that allows for deducing of the label

data: a collection of labelled examples $\{(\underset{\sim}{x}_i, y_i)\}_{i=1}^{N}$

$\underset{\sim}{x}_i$ feature vector $\begin{pmatrix} x_1 \\ x_2 \\ \vdots \end{pmatrix}$

how it works: 1) conversion of a real-world entity / phenomenon into a feature vector, and then label it ⇒ creation of machine readable data

eg. contains 'a' → $x_1 = 1$ or 0
contains 'b' → $x_2 = 1$ or 0
⋮
if contains 'a' and 'b', then $y_i = 1$

choice of model form, optimisation methods ↓ applied in code

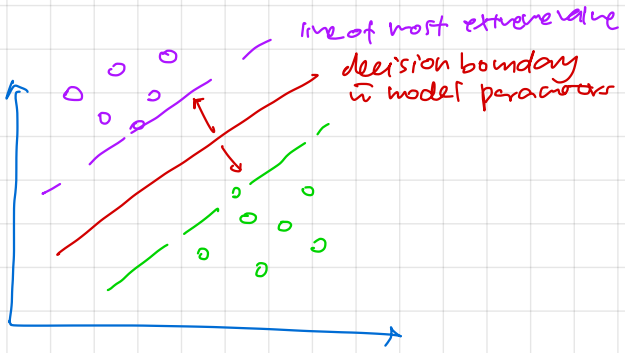
2) choose learning algorithm
3) use data set to calculate model parameters (eg. linear regression coefficients)
4) use model as is to take input and produce output

types of supervised learning

classification
(discrete classes / outcomes / labels)

⇒ put all feature vectors into n-dimensional plane, uses optimisation to find a decision boundary hyperplane with largest margin (separation) between groups of labels

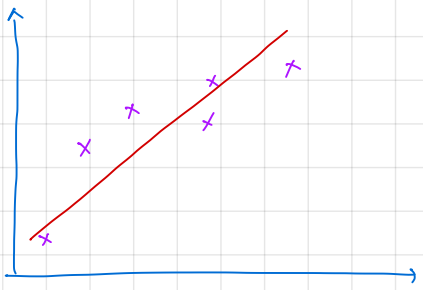
↳ input → which side of boundary is it on? → classification



regression
(continuous labels / values)

⇒ put all feature into n dimensional space, use optimisation to determine model parameters relating feature vectors ie label

↳ input → model relation to label → output value (prediction)



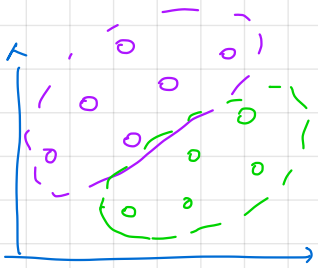
② unsupervised learning

dataset: a group of unlabelled feature vectors

goal: to take an unlabelled set of feature vectors and transform it into another vector or value that can be used → to gain useful insight from data you don't know much about

types of unsupervised learning:

- clustering
- dimensionality reduction
  - feature selection
    removing less relevant
  - feature extraction
    mapping to simpler variables eg. a, b → ab
    mathematically: projecting a space of higher dimensions into fewer dimensions
- outlier detection



eg. K means clustering

③ semi supervised learning

eg. not practical to label all, but have huge amount of data

dataset: a mix of labelled and unlabelled feature vectors. majority are unlabelled.

goal: to find a model's parameters using both data type, where unlabelled examples can help the learning algorithm get better parameters by leveraging on the additional probabilities, distribution and statistics from the unlabelled data.

mechanism: train w labelled data as in supervised learning → use to predict unlabelled data, pseudolabelling unlabelled data

pseudolabelled data w sufficiently high probability of being correct added and used to train model further

④ reinforcement learning

goal: to solve problems involving sequential decision making w long term goals

mechanism: machine can perceive state of environment / game as a vector of features, and can carry out actions to change the game, each with different rewards (defined by the programmer). uses past data to learn a policy (a model) that then takes an input (state of game) and outputs a move (according to the policy) that maximises the expected reward.

eg. tictactoe

⑤ parametric vs. non-parametric learning

select a form of model → learn model parameters/coefficients from calculations w training data

pros: simple, speed, less data

cons: strong assumption, limited complexity

use certain estimation functions to establish relationships → output

eg. kNN

pros: flexibility, feeds off data

cons: more data, slow, risk of overfitting is higher

⑥ inductive vs deductive reasoning

machine learning: to make inferences

types of inferences

inductive

to reach probable decisions / conclusions. Not all information is known, creating uncertainty.

↓

use of probability and statistics

deductive

to reach deterministic logical conclusions: all information that can lead to the optimal decision is available

Basics of data engineering

① Data types

- quantitative
  - discrete: distinct and separate data that can only take on a finite number of values
  - continuous: data that cannot be counted but can be measured. Readings are quantised, so measurements are approximations of infinite values.
- categorical (represent characteristics)
  - nominal: discrete, non-quantitative values that are nothing but labels.
  - ordinal: discrete, non-quantitative values that are ordered, but differences cannot be quantified.
    eg. bad, average, good, great

② Encoding categorical data

- numerical encoding
  ⇒ assigning arbitrary numbers to each class
  → higher values may have greater influence than those w/ smaller values when extreme values / data set sizes are used
- binary encoding
  eg. binary-coded decimal, n-ary gray code
  - one-hot encoding
    assigning a column for each class, each w/ a yes or no.

eg. RGB →
Red [1 0 0]
Green [0 1 0]
Blue [0 0 1]

③ Data wrangling

The process of transforming and mapping data from one "raw" form to another to make it more suitable for down stream analytics. easier to handle, more suited for algorithm (eg. KNN uses Euclidean distance, values being smaller speeds up calculation)

Numerical normalisation: (important because many ML models use Euclidean distance; having data w/ diff. orders of magnitude can skew model parameters in their favour / make calc. slower)

1) min-max scaling data has range of order of magnitudes → normalising it to range of 0 to 1 using min and max of each feature

$$x_{i,scaled} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \quad \text{for each feature}$$

(scale so that their dynamic ranges are similar)

2) Z score standardisation assumes a normal distribution, so scale it so that they are comparable normal distributions

$$x_{i,standardised} = \frac{x_i - E(X)}{\sigma(X)} \quad \text{for each feature}$$

3) Feature clipping removing extreme outliers from dataset

④ Data cleaning

1) missing or censored data

- do not know the mechanism for why the missing
- you know why the missing (eg. measurement failure)

indicated as NA, censored starter should be indicated as True / False

2) Dealing with missing features

- removing
  - rationale: removing as if no data on it's will not affect model.
  - drawback: if not large enough dataset, model may not be robustly trained
- data imputation
  - replacing w average value

$$\hat{x}^j = \frac{1}{N}\sum_{i=1}^{N} x_i^j$$

this way, the value will not significantly affect the classification / prediction

  - replacing w value outside normal range

eg. normal ∈ [0, 1] → put -1

The idea is that the learning algorithm will learn what is best to do when feature has a distant value (scenario)

Probability and statistics

① random variables, probability and distributions

uncertainty exists in data collection ⟶ we consider them as random variables, with distributions

1) axioms of probability

$\sum P(x_i) = 1$

$P(A \cup B) = P(A) + P(B) - P(A \cap B)$

2) independence

Discrete case

if independent, $P(A \cap B) = P(A) \times P(B)$

$P(A|B) = P(A)$ — i.e. B occurring does not give you more info about A

Continuous case

$f_{X,Y}(x,y)$ describes joint probability distribution.

$P(a \le X \le b, c \le Y \le d) = \int_c^d \int_a^b f_{X,Y}\, dx\, dy$

continuous, described by pdf

$E(X) = \int_{-\infty}^{\infty} x_i \cdot P(x_i)\, dx$

$Var(X) = \int (x_i - \mu) P(x_i)\, dx$

$= E(X^2) - [E(X)]^2$

$P(a \le X \le b) = \int_a^b f(x)\, dx$

discrete, described by pmf

$E(X) = \sum x_i \cdot P(x_i)$

$Var(X) = E[(X-\mu)^2]$

$= E(X^2) - [E(X)]^2$

$= \sum (x_i - \mu) P(x_i)$

$P(X_1 \cup X_2 \cdots) = P(X_1) + P(X_2) \cdots$

② maximum likelihood estimation

MLE is an estimator of θ that gives the maximum probability of observing the collected dataset.

population $\sim$ ⟶ sample $(x_1, x_2 \cdots x_n)$

$\theta \sim (\mu, \sigma^2)$

↓

model distribution of random variable X using sample $(x_1 \cdots x_n)$

eg. normal distribution ⟶ $P(x)$ modelled by known function $f(x)$

↓

$L = P($ of getting this dataset, given that $\theta =$ ⓗ (a value) $) = P(X = x_1, X = x_2 \cdots X = x_n \mid \theta =$ ⓗ $)$

$= P(X = x_1) \cdot P(X = x_2) \cdots P(X = x_n)$, assuming sample was random and independent.

$= \prod_{i=1}^{N} P(X = x_i)$ — some function of θ, known as the 'likelihood function'

↳ convert to sum using log ⟶ $l = \lg L$

$\log(ab) = \log a + \log b$

$= \lg P(X = x_1) + \lg P(X = x_2) \cdots$

$= \sum_{i=1}^{N} \lg P(X = x_i)$, a function of θ

(eg. in a bernoulli distribution of $X|\theta$,

$P(X|\theta) = \begin{cases} 1-\theta & x=0 \\ \theta & x=1 \end{cases}$

↓

$\frac{d}{d\theta}(l(\theta)) = 0$

maximising probability of dataset given $\theta =$ ⓗ

↓ make θ subject, solve for θ ⟶ $\hat{\theta}$. maximum likelihood estimator

(3) Bayesian estimation

1) Bayer rule

two random variables, X and θ. Consider a sample of size n where X and θ are sampled. Let number of data points $\in$ $X = x_i$ and $\theta = \theta_j$ be n_{ij}.

Then $P(X = x_i \wedge \theta = \theta_j) = \frac{n_{ij}}{n_{total}}$

$P(X = x_i \text{ only}) = \sum_{j=1}^{N} \frac{n_{ij}}{n_{total}}$

sum of all probabilities where $n = n_{ij}$ (j) — from $j=1$ to $j=N$

so $P(X = x_i \text{ only}) = \sum_{j=1}^{N} P(X = x_i \text{ and } \theta = \theta_j)$

similarly, $P(\theta = \theta_j \text{ only}) = \sum_{i=1}^{M} P(X = x_i \text{ and } \theta = \theta_j)$

$P(\theta = \theta_j \mid X = x_i) = \frac{n_{ij}}{\sum_{j=1}^{N} n_{ij}}$

so $P(X = x_i \text{ and } \theta = \theta_j) = \frac{n_{ij}}{n_{total}} = \frac{n_{ij}}{\sum_{j=1}^{N} n_{ij}} \cdot \frac{\sum_{j=1}^{N} n_{ij}}{n_{total}}$

$= P(\theta = \theta_j \mid X = x_i) \cdot P(X = x_i)$

$P(X \mid \theta) = \frac{P(X \wedge \theta)}{P(\theta)}$ ← so $P(X \wedge \theta) = P(\theta \mid X) \cdot P(X)$

$\frac{P(X \mid \theta) \cdot P(\theta)}{P(X)} = P(\theta \mid X)$

2) Bayes rule for parameter estimation using data

⇒ given model of $P(X \mid \theta)$ (data) and prior distribution of θ ($P(\theta)$), find posterior pdf $P(\theta \mid x)$ — distribution of θ now that X has been observed

prior = prior beliefs about θ, often modelled by $\theta \sim N(m, s^2)$

$$P(\theta \mid X) = \frac{P(X \mid \theta) \cdot P(\theta)}{P(X)}$$

- $P(\theta \mid X)$: updated distribution of θ
- $P(\theta)$: prior distribution of θ
- $P(X \mid \theta)$: joint distribution given last known distribution of θ
- $P(X)$: distribution from data, known

allows for inclusion of incorporating prior with new info and finding estimator using cost functions

$\hat{\theta}$, estimator of θ = value of θ that minimises $E(c(\hat{\theta} - \theta))$

$= \arg\min_{\hat{\theta}} (E(c(\hat{\theta} - \theta)))$

- $E(c(\hat{\theta} - \theta))$: expected cost
- $\hat{\theta} - \theta$: estimation error
- c: cost function → eg. MAP

3) maximum a posteriori (MAP) estimation

cost function : $c(\hat{\theta} - \theta) = \begin{cases} 1 & \hat{\theta} \neq \theta \\ 0 & \hat{\theta} = \theta \end{cases}$ function returns a value of 1 when estimator is correct

$\hat{\theta}$, estimator of θ = value of θ that minimises $E(c(\hat{\theta} - \theta))$

$= \arg\min_{\hat{\theta}} (E(c(\hat{\theta} - \theta)))$

$\hat{\theta}$, max a posteriori = $\arg\max_{\theta} \lg P(\theta = \hat{\theta} | X = x)$

value of $\hat{\theta}$ that maximises probability of $\theta = \hat{\theta}$ given that you have observed your data set X

Notes

1. Do consider causation vs correlation → but the score can be quantified by r, r^2, cov, but must have some sort of causal link - Also remember common factor and confounding.

linear equations, functions and calculus

① matrix operations

1) dot product : $\underset{\sim}{x} \cdot \underset{\sim}{y} = \underset{\sim}{x}^T * \underset{\sim}{y} = |\underset{\sim}{x}||\underset{\sim}{y}| \cos\theta$

2) matrix multiplication : $m_1 \times n_1 * m_2 \times n_2 \rightarrow m_1 \times n_2$

↳ properties — $A(BC) = (AB)C$

$A(B_1 + B_2) = AB_1 + AB_2$

$(C_1 + C_2)A = C_1A + C_2A$

3) transpose : $m \times n \rightarrow n \times m$

↳ properties — $A = A^T$ if symmetric

$(A^T)^T = A$

$(A+B)^T = A^T + B^T$

$kA^T = (kA)^T$

$(AB)^T = B^T A^T$

Consider a 3×3 matrix

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

The minor $\mathbf{A}_{11} = \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix}$

Its cofactor matrix is

$$\mathbf{C} = \begin{pmatrix} +\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & -\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} & +\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \\ -\begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} & +\begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & -\begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} \\ +\begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} & -\begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} & +\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{pmatrix}.$$

4) determinant : $\det(A)$ = alternating sum of minors until $2\times2 \rightarrow ad - bc$

5) matrix inverse : $A^{-1} = \frac{1}{\det(A)} \cdot \text{adj}(A)$ — transpose of cofactor

6) linear independence : full rank, $\det \neq 0$, $AX = 0 \rightarrow X$ is trivial

② systems of linear equations

1) even determined systems : no. of unknowns = no. of equations → exact solution at intercept

2) over determined system : no. of equations > no. of unknowns → no exact solution that can fulfil all equations

but if $\text{rank}(X) = \text{rank}(X|Y) \Rightarrow$ got solution, just remove entire dependent row

$\underset{\sim}{X}\underset{\sim}{w} = \underset{\sim}{y}$ → minimise euclidean error

approximate using least squares solution

alternatively, can also use projection theorem: minimal distance when orthogonal

$\min \|y - Xw\| = \min (y - Xw)^T (y - Xw)$

$= \min \; y^Ty - y^TXw - w^TX^Ty + w^TX^TXw$

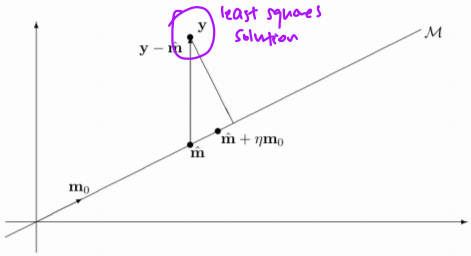
$\downarrow$

$\frac{d}{d\underset{\sim}{w}} = 0$

$-(y^TX^T) - (X^Ty) + 2X^TXw = 0$

$w = (M^TM)^{-1}M^Ty$

pseudo inverse



3) underdetermined system : no of equations < no. of unknowns → infinitely many solutions

choose smallest solution, so that no error will be 'amplified'

approximate using least norm solution

constraint $Xw = y$, minimise w

↳ use lagrange multipliers

↳ $\|w\|^2 + \lambda^T(y - wX) \rightarrow \frac{d}{dw} = 0 \rightarrow$

$2w - X^T\lambda = 0$

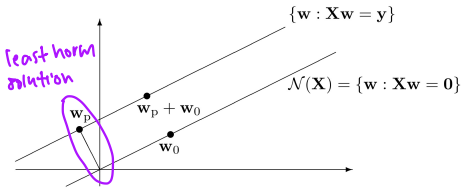
$2Xw - XX^T\lambda = 0$

$2y = XX^T\lambda$

$\lambda = 2(XX^T)^{-1}y$

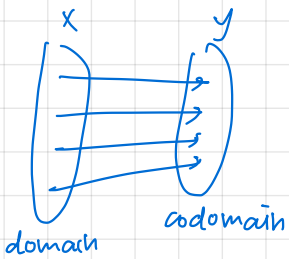
$w = X^T(XX^T)^{-1}y$

pseudo inverse



(3) Functions

1) meaning : a relation that associates the argument (from a domain) to a single element of another set (the codomain)



notation : $\mathbb{R}^d \rightarrow \mathbb{R}$, takes in vector of d dimensions and outputs scalar

eg. $f(\underset{\sim}{x}) = f(x_1, x_2, \ldots)$

2) scalar function : can take in vector or scalar argument, but outputs scalar

vector function : can take in a vector or scalar argument, but outputs vector

3) linear and affine functions

satisfies

homogeneity: $f(\alpha \underset{\sim}{x}) = \alpha f(\underset{\sim}{x})$

additivity: $f(\underset{\sim}{x} + \underset{\sim}{y}) = f(\underset{\sim}{x}) + f(\underset{\sim}{y})$

superposition property, extends to linear combinations of vectors

'linear' and can be expressed in an offset, $f(\underset{\sim}{x}) = a^T\underset{\sim}{x} + b$

normally expressed as w_0, and first column of $\underset{\sim}{x}$ is '1'

④ Matrix derivative

1) intuition

- w.r.t. to a scalar $\frac{dA}{d\alpha} = \begin{bmatrix} \frac{\partial a_{11}}{\partial \alpha} & \frac{\partial a_{12}}{\partial \alpha} & \cdots \\ \vdots & & \end{bmatrix}$

  intuition: a matrix of partial derivatives wrt to constant α

- w.r.t. another matrix $\frac{dA}{dZ} = \begin{bmatrix} \frac{\partial a_{11}}{\partial z_{11}} & \frac{\partial a_{12}}{\partial z_{21}} & \cdots \\ \frac{\partial a_{21}}{\partial z_{11}} & \vdots & \\ \vdots & & \end{bmatrix}$

  intuition: a matrix of partial derivatives wrt to corresponding element in Z

  following matrix multiplication order

  → change in A (each element of A) w.r.t. to elements in matrix it will be multiplied by

2) chain rule applies: $\frac{\partial}{\partial Z} F(X) = F'(X) \frac{\partial X}{\partial Z}$

3) rules

$$\frac{\partial}{\partial X} AX = A, \quad \frac{\partial}{\partial Z} AX = A \frac{\partial X}{\partial Z}$$

$$\frac{\partial}{\partial X} y^T A X = y^T A, \quad \frac{\partial}{\partial y} y^T A X = X^T A^T$$

$$\frac{\partial}{\partial X} X^T A X = X^T (A + A^T)$$

$$= 2X^T A \text{ if } A \text{ is a symmetric matrix}$$

$$\frac{\partial}{\partial Z} y^T x = X^T \frac{\partial y}{\partial Z} + y^T \frac{\partial X}{\partial Z}$$

$$\frac{\partial}{\partial Z} X^T X = 2X^T \frac{\partial X}{\partial Z}$$

Notes

1. when using transpose to change eqn: eg. $w^T X = y^T \rightarrow X^T w = y$

   if they ask for w^T, make sure to transpose back!

Learning model building blocks

① Anatomy of a learning algorithm

a loss function → an optimisation criterion → an optimisation routine

- loss function: penalty for mis prediction of example i
- optimisation criterion: objective: what you want to minimise
- optimisation routine: how you minimise, e.g. gradient descent
- model parameters

$$\hat{w} = \underset{w}{\operatorname{argmin}}\ c(w) = \underset{w}{\operatorname{argmin}} \sum_{i=}^{n} L\left(f(x_i, w) + \lambda R(w)\right)$$

- $c(w)$: objective/cost function; overall optimisation criterion function to minimise
- L: loss function; penalty for wrong prediction
- $f(x_i, w)$: learning model; reflects our belief about the relationship between x and y
- $R(w)$: regularisation function; managing bias variance tradeoff

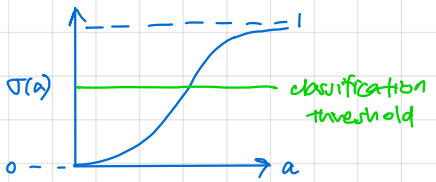
② Different learning models

1) polynomial, linear $f(x) = wx$ or wp

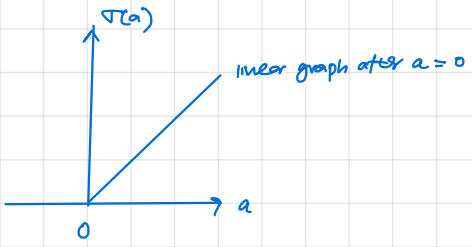
combining linear and non-linear function

2) non linear functions: $a = wx$ or wp

- sigmoid function
- rectified linear unit (ReLU)
- exponential



$$\sigma(a) = \frac{1}{1+e^{-a}}$$



$$\sigma(a) = \max(0, a)$$

- not strictly differentiable, but can be described by piecewise function that is

$$\sigma(a) = \begin{cases} 0 & a < 0 \\ a & a \geq 0 \end{cases}$$

$$\sigma(a) = e^{-a}$$

$$\nabla C(w) = \nabla \sum (f(x_i; w) - y_i)^2$$

$$f(x_i; w) = \sigma(x_i; w)$$

$$\nabla C(w) = \sum 2(\sigma(x_i; w) - y_i)\, \beta\, \sigma(x_i; w)(1 - \sigma(x_i; w))\, x_i$$

$$\nabla C(w) = \sum 2(f - y_i)\, -e^{-x_i w}\, x$$

$$\nabla C(w) = \sum 2(x_i w - y_i)\, x_i \quad \text{, if } x_i w > 0$$

$$\frac{df}{dw} = \frac{df}{da} \cdot \frac{da}{dw}$$

③ Different loss functions

1) ℓ_2 loss

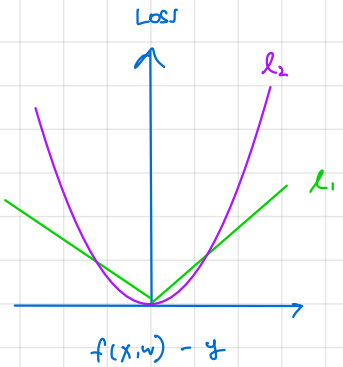
$$\text{Loss}(\hat{y}, y) = (\hat{y}_i - y_i)^2$$

- minimizes deviation of all points

2) ℓ_1 loss

$$\text{Loss}(\hat{y}, y) = |\hat{y}_i - y_i|$$

- enforces robustness to outliers since error is not squared (less heavy compared to ℓ_2 loss)



3) 0-1 / binary loss

$$\text{Loss}(\hat{y}, y) = \begin{cases} 0 & \text{if } f(x,w) = y_i \\ 1 & \text{if } f(x,w) \neq y_i \end{cases}$$

- function is non-convex and non-differentiable, difficult to optimize

4) binary loss in practice

$$\text{Loss}(\hat{y}, y) = \begin{cases} 0 & \text{if } f(x,w) > 0 \\ 1 & \text{if } f(x,w) < 0 \end{cases}$$

- in practice, hard to regress $f(x,w)$ to equal -1 or 1 (classes), so we consider it sufficient if they have the same sign (set threshold @ 0)

non-differentiable → use of similar shape continuous curves

5) exponential loss

$$\text{Loss}(y, \hat{y}) = e^{-y\hat{y}}$$

- application to FS97 classification algorithm

6) hinge loss

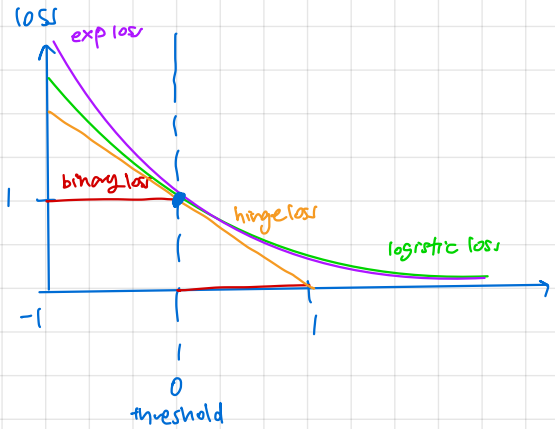
$$\text{Loss}(\hat{y}, y) = \max(0, (1 - f(x,w)))$$

- used in SVM classification algorithms

7) logistic loss

$$\text{loss}(y, \hat{y}) = \lg(1 + e^{-y\hat{y}})$$

- used in logistic regression, which has a probabilistic interpretation



$f(x,w) \cdot y$ → product

↓

same sign, consider as same class

⇓

if positive, correct, loss = 0

④ optimisation routine : gradient descent

gradient descent is an iterative optimisation algorithm for finding the minimum of a function.
↳ only able to find a local minimum

1) motivation

↳ not all learning algorithms have closed form solutions (w in linear regression) → estimate minimum iteratively

2) workings

$\hat{w} = \arg\min C(w)$ → gradient of $C(w) = \nabla C(w) = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \end{pmatrix}$

intuition : the direction in n dimensional space where C is increasing

to find min $\hat{w}$ iteratively, we need convergence criteria

↓

initialise w and learning rate η

↓

algorithm :

while true :

$w = w - \eta \nabla C(w)$ ← hyper parameter determining step size

- too small → slow convergence
- too large → may overshoot and 'oscillate', never converge

if converge : return w

else : ∇

eg. $y = x^2$

$x = x - (\eta 2x)$ even though you technically $- f'(x)$, a 'y' value, what you care about is that it is a function of x and η → magnitude moderated by η

if has the direction of decrease (signof $f'(x)$)

- k loops
- if ∇w or $\nabla \cdot \nabla w$ below a certain threshold (absolute or percentage)
- $\nabla C = 0$

Bias and variance

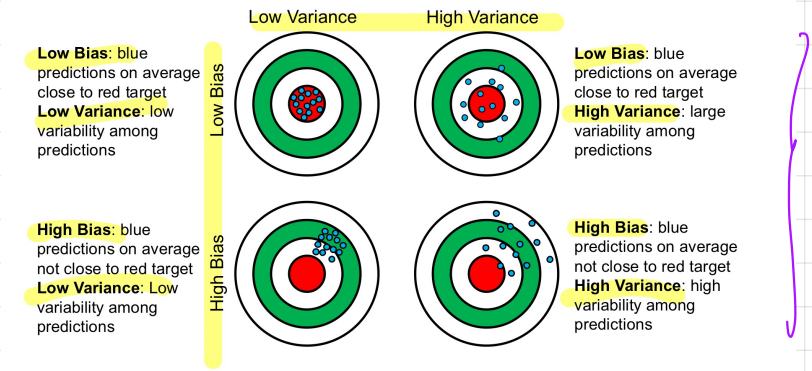
① Bias and variance

1) bias : the systematic deviation of a prediction from the true value

↳ if bias of model is low, then accuracy is high on training data

2) variance - the 'spread' of predictions given different training data; error of the model due to sensitivity to fluctuations in the training set

↳ if variance of model is high, then coefficients (and predictions) vary greatly w different train and test sets



applied to prediction of test set, given different training sets

② Bias - variance trade-off

1) quantifying test error : using mean squared error

2) proof: true population relationship, $y(x) = f(x) + \varepsilon$, $\varepsilon \sim (0, \sigma^2)$

↓ random sample D

learning from D, $\hat{y}(x) = \hat{f}_D(x)$ → expected prediction of a model that has learnt from D : $E(\hat{f}_D(x))$

expected MSE of prediction of a model that has learnt from D $= E\left(\left[y(x) - \hat{f}_D(x)\right]^2\right)$ (bias = E(prediction − true value))

$$= \left[E\left(\hat{f}_D(x) - y(x)\right)\right]^2 + \text{Var}\left(\hat{f}_D(x)\right) + \text{Var}\left(y(x)\right)$$

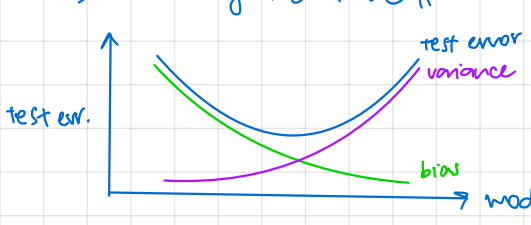
$$= (\text{prediction bias})^2 + (\text{prediction variance}) + \sigma^2$$

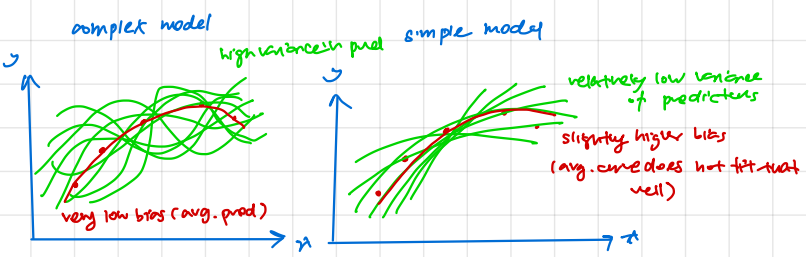
intuition: model learnt from random sample D will have an expected MSE (average across all possible samples D of same size) of its expected bias^2 (constant), prediction variance (constant) and random variance due to noise (constant) summed.

↳ MSE(test error) = bias^2 (in prediction of tests) + variance (in predictors) + noise

(bias: quantifies error due to simplifying assumptions (ie. model itself); noise: quantifies measurement noise; variance: how much estimator fluctuates about its mean)

3) visualising the trade-off

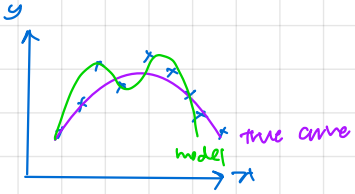




③ overfitting and underfitting

1) overfitting : occurs when model predicts training data well but predicts new data poorly

1. causes : — not enough data to estimate coefficients well, too many features
(model is too complex and fits noise of training data



problem of high variance

low bias ≠ correct predictions. Just means across all, average is true value. But high var means indiv. values are often far off true value.

by being sensitive to small fluctuations (like noise) in the training set, if training data were different, model would change significantly.

↓

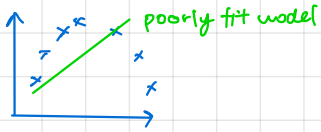
poor performance on test set since train and test set are independent samples

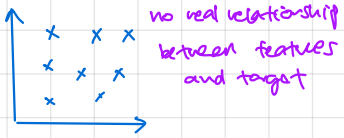
2. solutions : — more data
(try a simpler model
(use regularisation
(reduce dimensionality of dataset

2) underfitting : the inability of the trained model to predict targets in the test and training sets

problem of high bias

1. causes : — model is too simple to capture relationships in the data
(features are not informative enough





2. solution : — try a more complex model
(re-engineer features to learn from

④ regularisation : solution to overfitting

regularisation : an umbrella term referring to methods forcing learning algorithms to build less complex models

1) motivation : — solving ill-posed problems (insufficient data, underdetermined systems)
(reduce overfitting

2) processes

quantifying data loss (fitting error to train set

optimisation criterion : $\underset{w}{\arg\min}\ (\text{data loss}(w) + \lambda_1 \text{regularisation}_1 + \lambda_2 \text{regularisation}_2 \dots)$

λ : hyperparameter to weigh relative importance

'cost' function that penalises complexity

cost function

3) regularisation algorithms

L1 regularisation

$\lambda \|w\|$, magnitude of coefficient vector

- gives sparse model i.e most parameters equal to 0
- makes feature selection by deciding which features are essential for prediction, improving model explainability

L2 regularisation (ridge)

$\lambda \|w^2\|$, square of coefficients

- often gives better results
- differentiable, so gradient descent can be used

4) application: effect of regularisation is more significant when there is large noise in the dataset since it stabilises the solution by reducing sensitivity to fluctuations

Fundamental supervised learning algorithms

① Anatomy of a learning algorithm

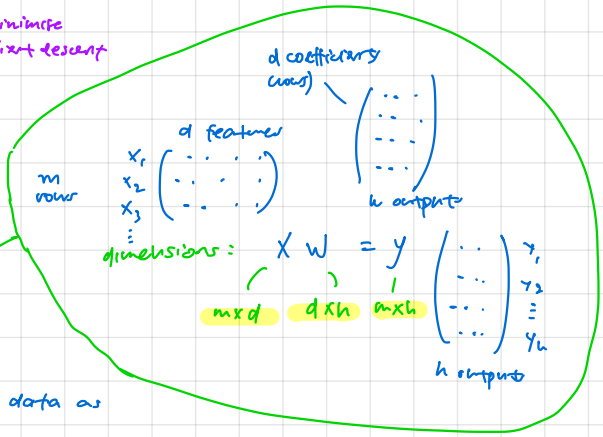
a loss function → an optimisation criterion → an optimisation routine

- loss function: penalty for mis prediction of example i
- optimisation criterion: objective = what you want to minimise
- optimisation routine: how you minimise e.g. gradient descent
- model parameters

② linear regression

1) problem statement: collection of labelled examples $\underset{\sim}{X}$, $\underset{\sim}{y}$

2) model: $\hat{y}_i = \hat{w}_0 + \hat{w}_i x_i \Rightarrow \underset{\sim}{y} = \underset{\sim}{X}\,\underset{\sim}{w}$



3) regression: to predict a continuous y value ⇒ model as close to data as possible

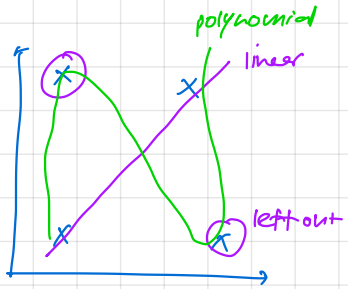
4) cost function: $\frac{1}{N}\sum (f_{w,b}(\underset{\sim}{x}_i) - \underset{\sim}{y}_i)^2$ → mean squared error

5) optimisation criterion: minimisation

6) optimisation routine: solving for $\hat{w}, \hat{b}$ using matrix calculus → least norm solution
- least squares approximation
- exact solution

③ polynomial regression

1) problem: when linear regression is not sufficient to describe the data eg.



2) model: $\hat{y}_i = w_0 + w_i x_i + \sum_{i=1}^{d}\sum_{j=i}^{d} w_{ij} x_i x_j + \sum_{i}^{d}\sum_{j}^{d}\sum_{k}^{d} w_{ijk} x_i x_j x_k \ldots$

- 2^{nd} degree combinations
- 3^{rd} degree combinations

Weierstrass Approximation Theorem. Suppose f is a continuous real-valued function defined on the real interval $[a, b]$. For every $\varepsilon > 0$, there exists a polynomial p such that for all x in $[a, b]$, we have $|f(x) - p(x)| < \varepsilon$.

3) regression: same as linear regression, but adding products of terms to make it non-linear

$X w = y \rightarrow P w = y \rightarrow$ solve for w

expand feature vector to include product terms

eg. $m \left| \begin{pmatrix} x_{1,1} & \cdots & x_{1,d} \\ x_{2,1} & \cdots & x_{2,d} \end{pmatrix} \right|_d \left| \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \right|_h = \left| \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \right|_h$ (X: $m \times d$)

P: $m \times$

$$\begin{pmatrix} 1 & x_1 & x_2 & x_1x_2 & x_1^2 & x_2^2 \\ \vdots & & & & & \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_{12} \\ w_{11} \\ w_{22} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

④ Ridge regression

1) problem statement: 1. in linear regression, we cannot guarantee pseudo inverses exists (that X^TX or XX^T are invertible) → no solution

2. if X^TX or XX^T are almost singular, then inverse of almost-zero matrix will be very large.
⇒ $\hat{w}$ will be huge causing unstable predictions ⇒ high variance

consider restricting the weight of $\underline{w}$ using some budget C.

use 2-norm, L-2 regularisation

decrease variance at the tradeoff of increasing bias (accuracy)

3. some learning applications have large d-dimensional features (eg. DNA sequencing); least norm solution provides only low-fidelity control over bias-variance tradeoff for underdetermined systems

2) model, cost function: unchanged

3) optimisation criterion: $\min_{\underline{w}} \sum \left(f_w(x_i) - y_i\right)^2 + \lambda \|w^2\|$

(regularisation parameter; 2 norm; least square error)

6) optimisation routine: matrix calculus

$$\frac{\partial}{\partial w}(Xw-y)^T(Xw-y^T) + \lambda w^T w = 0$$

$$X^TXw - X^Ty + \lambda w = 0$$

$$(X^TX + \lambda I)w = X^Ty$$

$$w = (X^TX + \lambda I)^{-1}X^Ty$$

primal form, for overdetermined systems

$w = X^Ta$

$\lambda a = y - XX^Ta$

$(XX^T + \lambda I)a = y$

$a = (XX^T + \lambda I)^{-1}y$

$a = (X^TX - \lambda I)^{-1}y$ (equivalent)

$w = X^T(XX^T + \lambda I)^{-1}y$

dual form, for underdetermined systems

notice that solving for w here is only in n dimensions ($XX^T \to \lambda I$ guarantees solution to exist)

↓

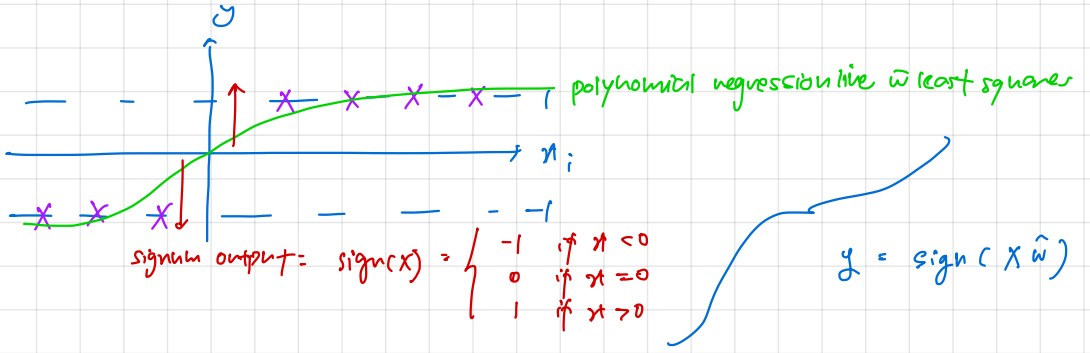
regardless of dimension of feature d

equivalent

⑤ Classification

1) binary classification

model : any regression model + signum function + encoding labels as -1 / +1



$y = \text{sign}(X\hat{w})$ → binary output of -1 or 1.

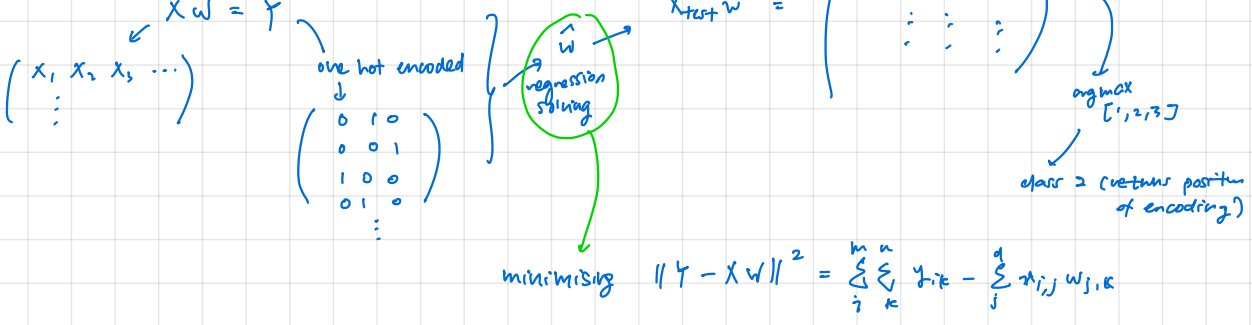
if 0, then cannot classify

2) multi-category classification

model : any regression model + one-hot encoding + argmax operator

(minimal normalise or standardise this to get probability)

(domain = position)



minimising $\|Y - XW\|^2 = \sum_i^m \sum_k^n y_{ik} - \sum_j^d x_{i,j} w_{j,k}$

(6) Dimensions and computation of LSE for linear regression

1) For an even determined system:

$$\begin{pmatrix} x_{1,1} & \cdots & x_{1,d} \\ x_{2,1} & \cdots & \vdots \\ x_{3,1} & \cdots & \vdots \\ \vdots & \cdots & \vdots \\ x_{m,1} & \cdots & x_{m,d} \end{pmatrix} \begin{pmatrix} w_{1,1} & \cdots & w_{1,n} \\ w_{2,1} & \cdots & \vdots \\ \vdots & \cdots & \vdots \\ w_{d,1} & \cdots & w_{d,n} \end{pmatrix} = \begin{pmatrix} y_{1,1} & \cdots & y_{1,n} \\ y_{2,1} & \cdots & \vdots \\ \vdots & \cdots & \vdots \\ y_{m,1} & \cdots & y_{m,n} \end{pmatrix}$$

$m \times d$ $\quad d \times n$ $\quad m \times n$

cost function to find $\hat{W}$, $j(W) = \|(XW - Y)\|^2 = (XW - Y)^T (XW - Y)$ ($\|E\|^2$, E^T, E)

$$= \begin{pmatrix} e_1^T \cdots \\ e_2^T \cdots \\ \vdots \\ e_m^T \cdots \end{pmatrix} \cdot (e_1 \; e_2 \; \cdots \; e_m)$$

you want to minimize this, but what you really need to minimize is the individual square errors $e_k \cdot e_k$, it is more efficient

take trace of the cost function

trace $(E^T \cdot E)$

use that for minimization

$$= \text{trace}\left(\begin{bmatrix} \mathbf{e}_1^T\mathbf{e}_1 & \mathbf{e}_1^T\mathbf{e}_2 & \cdots & \mathbf{e}_1^T\mathbf{e}_h \\ \mathbf{e}_2^T\mathbf{e}_1 & \mathbf{e}_2^T\mathbf{e}_2 & \cdots & \mathbf{e}_2^T\mathbf{e}_h \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{e}_h^T\mathbf{e}_1 & \mathbf{e}_h^T\mathbf{e}_2 & \cdots & \mathbf{e}_h^T\mathbf{e}_h \end{bmatrix}\right) = \sum_{k=1}^{h} \mathbf{e}_k^T\mathbf{e}_k$$

2) under- and over determined systems

$Y = XW$

$m \times n$ $\quad m \times d$ $\quad d \times n$

over $\qquad$ under

$$W = (X^T X + \gamma I)^{-1} X^T y \qquad W = X^T (XX^T + \lambda I)^{-1} y$$

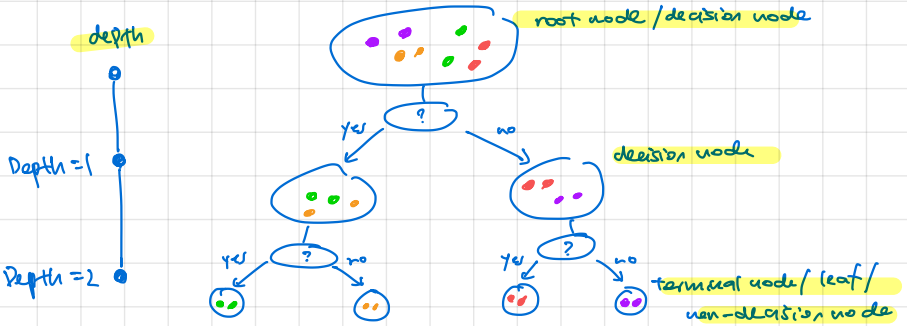
$d \times n$ $\quad d \times m$ $\quad m \times d$ $\quad m \times m$ $\quad d \times m$ $\quad m \times n$

$d \times n$ $\quad d \times d$ $\quad m \times m$ $\quad d \times m$ $\quad m \times n$

⑦ Decision trees

1) basic terminology and working principle

graphically



principle

1. start at root node. Quantify error/impurity in some way. Find feature and threshold that splits dataset in a way that reduces error quantity by the most.
2. Repeat at sub nodes. Continue until end-conditions reached or minimal entropy (ie. only 1 datum)

Finding smallest possible tree is computationally difficult (since iterative, not analytical), so a greedy algorithm is used to maximize chance of getting, but does not guarantee.

↓

Need way to quantify node impurity (error)

2) binary classification trees : node impurity

(how 'pure' the node is in classification. If only 1 class, pure. purity is desirable since prediction is perfect.

↳ Node impurity measures:

$$p_i = \frac{\text{no. of datum from class } i}{\text{total no. of datum}}$$

Gini impurity

$$Q_{node} = 1 - \sum_{i=1}^{n} p_i^2$$

(n = no. of classes)

entropy

$$Q_{node} = -\sum_{i=1}^{n} p_i \log_2 p_i$$

misclassification rate

$$Q_{node} = 1 - \max(p_i)$$

↳ algorithm:

root = all training samples
for d to max_depth:
    for each leaf @ depth d-1:
        find best feature and best threshold, so splitting node @ depth d will reduce the most impurity
    split according to best feature and threshold
return tree

3) regression trees → continuous variables

↳ minimise MSE / other error-cost functions

total MSE = $\sum_{m}^{M} \frac{J_m}{N} MSE_m$

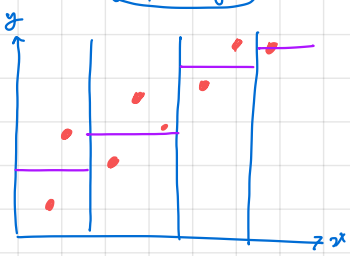
(J_m: no. of training samples @ leaf node m; MSE_m: MSE @ leaf node m; N: total no. of training samples)

↳ prediction @ terminal node = mean target value of samples @ node

algorithm

↳ same as classification. Difference is in cost function and prediction function.

graphically



4) Advantages and disadvantages of decision trees

- easy to visualise and understand, can trace path to investigate mechanisms
- can work w continuous and discrete data
- makes less assumptions between y and $\underset{\sim}{X}$
- less data cleaning

- trees can become highly complex and overfit
- trees can be unstable (small changes in training data result in very different trees)

5) reducing overfitting and tree instability → random forest algorithm

- set max depth
- set min no. of samples for splitting a leaf node
- set minimum % or absolute change in impurity / error to split leaf node
- instead of looking at all features, look at first T_m features randomly

⑧ random forest

1) motivation: to reduce tree instability

↳ small perturbations result in very different trees — low bias, high variance

2) mechanism

↳ we can perturb training set to generate m perturbed training sets, and we train M trees. prediction = average prediction of trees / most freq prediction

↳ individual tree is overfit to training data and likely to have large error → random forest (w/ bootstrapping) is based on principle that a suitably large no. of uncorrelated error average out to zero → since each tree learns from different data, they are fairly uncorrelated w each other.

3) perturbing the data : bootstrapping

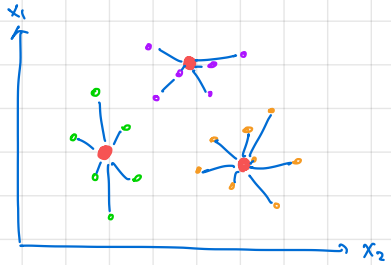
- draw randomly from original dataset (w replacement)
- bootstrapped set is same size, may contain repeated samples, might not contain some samples from original dataset

Fundamental unsupervised learning algorithms

① naive (hard) k-means clustering

1) intuition : to identify the number of 'groups' underlying the data, by testing the spread of k identified-to-be-clusters clusters

2) algorithm
1. randomly assign initial centroids
2. compute euclidean distance from each centroid, assign each datum to nearest centroid
3. find new centroid by calculating mean
4. repeat until no more changes in centroid



3) optimisation (objective) function

$$J = \sum_{i=0}^{N} \sum_{k=0}^{K} w_{ik} \| \underline{x}_i - \underline{c}_k \|^2$$

data point, centroid, centroid label, centroid, datum

⇒ by always assigning label to nearest centroid, you minimise J, since class label will be 0 for all other centroids

4) initialisation

↳ initial position of centroids can influence the final positions ⇒ need to choose based on context; no strict method known to be optimal

random partition — fuzzy method

↳ randomly assigns each point to a cluster; immediately proceeds to update step

↳ randomly chooses k observations and uses them as initial means

② fuzzy (soft) clustering

1) intuition : same as naive k-means, except that you consider using fuzzy logic, giving you an added dimension of assessing likelihood of cluster existence

2) objective function :

$$J = \sum_{i=0}^{N} \sum_{k=0}^{K} (w_{ik})^r \| \underline{x}_i - \underline{c}_k \|^2$$

data point, centroid, centroid label, fuzzifier, determines level of cluster fuzziness, centroid, datum

$$w_{ik} = \frac{1}{\sum_{j=1}^{} \left(\frac{x_i - c_k}{x_i - c_j} \right)^{\frac{2}{r-1}}}$$

weight value that x_i belongs to centroid k, by evaluating euclidean distance to centroid k compared to euclidean distance to overall centroid

3) algorithm

1. initialise
2. calculate euclidean distances to all centroids, calculate fuzzy weight vector w for all datum
3. assign new centroids by finding mean of datum in argmax as class k
4. iterate until no change in centroids

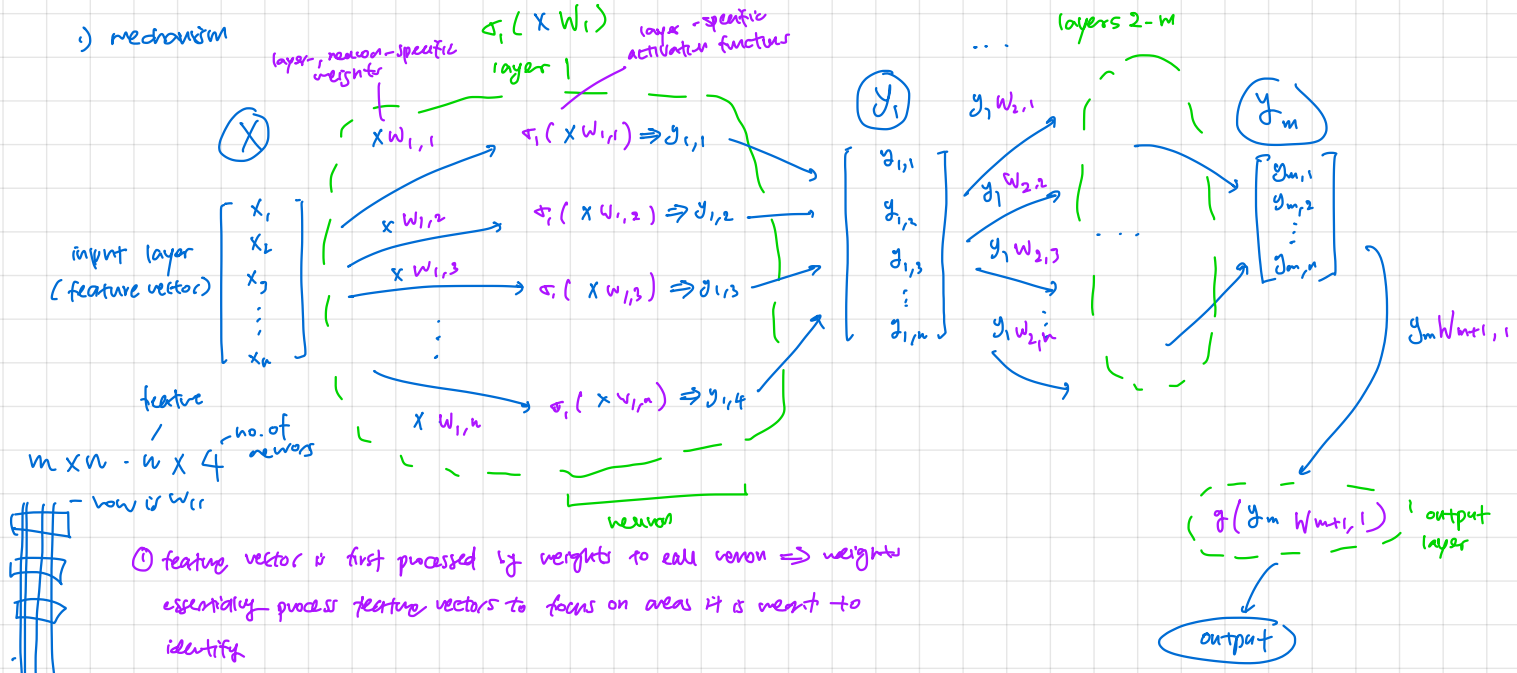
Neural networks

↳ to inject non-linearity into a model

forward function $\sigma\left(\begin{bmatrix} w \\ w \\ \vdots \\ w \end{bmatrix}\begin{bmatrix} x \end{bmatrix}\right)$

① intuition and anatomy of vanilla neural networks

i) mechanism



① feature vector is first processed by weights to each neuron ⇒ weights essentially process feature vectors to focus on areas it is meant to identify

② if things it's looking out for is found ⇒ activation function g outputs an activation signal

③ activation signals become an input vector for the next layer

④ after m layers, output from mth layer is used as an input for the output function ↳ gives output in required dimensions (eg. classification)

ii) mathematical intuition

↳ essentially a nested function

$$\underset{\text{output}}{Y} = \underset{\text{input}}{F(X)} = \sigma_3\left(\sigma_2\left(\sigma_1(XW_1)\cdot W_2\right)W_3\right)\dots$$

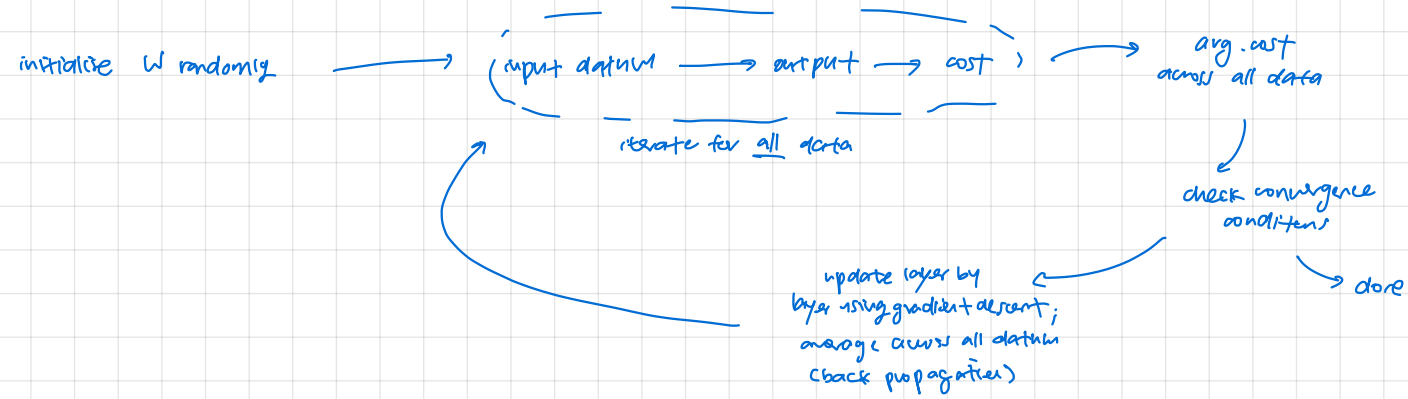
- σ_3: alignment based on matrix dimensions
- σ_2: layer-specific activation function
- W_2, W_3: multidimensional matrices of weight vectors of each neuron in each layer ⇒ layer- and neuron-specific

iii) intuition: increase resolution of model by introducing different functions to model different parts of a single phenomena

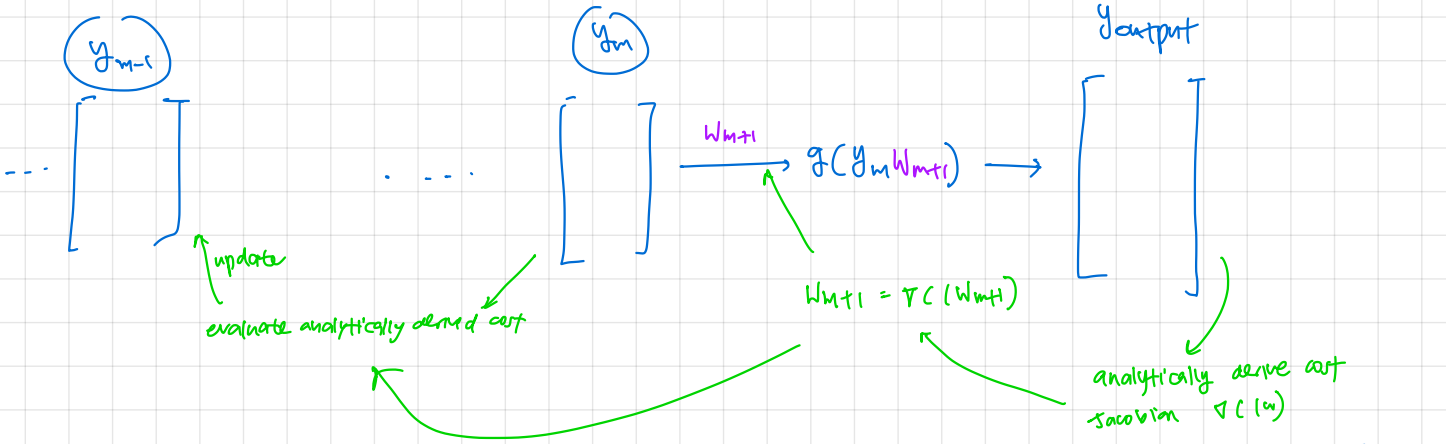
↳ eg. number recognition

② training a neural network : back propagation

1) intuition of training



2) intuition of back propagation



$$\nabla C_{layer}(W_{layer}) = \frac{\partial C_{layer}}{\partial W_{layer}} = \frac{\partial y_{layer-1}}{\partial W_{layer}} \cdot \frac{\partial y_{layer}}{\partial y_{layer-1}} \cdot \frac{\partial C_{layer}}{\partial y_{layer}}$$

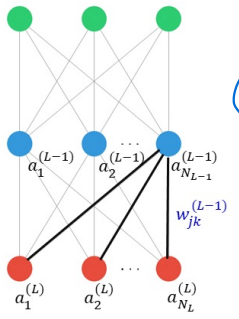
(input from previous layer)

update all W,
store changes for each datum.
⇒ avg. across all datum is the change for 1 epoch (so that you don't overfit to 1 datum)

↳ for each layer, there is an analytical expression of this.

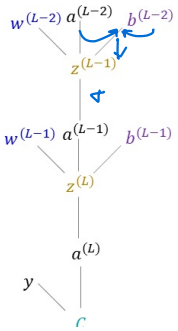
↳ use values in each layer to evaluate, use to update W_{layer}

3) the math



$$\frac{\partial C}{\partial w_{jk}^{(L-1)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C}{\partial a_j^{(L)}}$$

$$\frac{\partial C}{\partial a_k^{(L-1)}} = \sum_{j=1}^{N_L} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C}{\partial a_j^{(L)}}$$



$$\frac{\partial C}{\partial a^{(L)}} = 2(a^{(L)} - y)$$
$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$
$$\frac{\partial z^{(L)}}{\partial a^{(L-1)}} = w^{(L)}$$

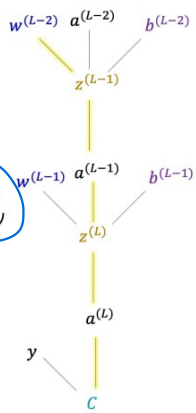
$$\frac{\partial C}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}} = w^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

What we want: $\frac{\partial C}{\partial w^{(L-2)}}$

Chain rule

$$\frac{\partial C}{\partial w^{(L-2)}} = \frac{\partial z^{(L-1)}}{\partial w^{(L-2)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}}$$

$$\frac{\partial C}{\partial w^{(L-2)}} = \frac{\partial z^{(L-1)}}{\partial w^{(L-2)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial C}{\partial a^{(L-1)}}$$



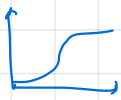
$$\frac{\partial C}{\partial a^{(L)}} = 2(a^{(L)} - y)$$
$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$
$$\frac{\partial z^{(L)}}{\partial w^{(L-1)}} = a^{(L-1)}$$

$$\frac{\partial C}{\partial w^{(L-1)}} = \frac{\partial z^{(L)}}{\partial w^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

③ common activation functions

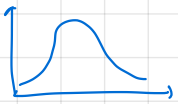
1) sigmoid function

$$\sigma(a) = \frac{1}{1+e^{-\beta a}}$$



2) Gaussian function

$$\sigma(a) = \frac{1}{\sqrt{2\pi}\rho} e^{-\frac{\rho}{2}\left(\frac{a-\mu}{\rho}\right)^2}$$

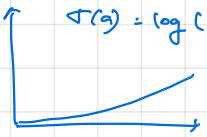


3) ReLu

$$\sigma(a) = \max(0, a)$$

4) softplus activation function

$$\sigma(a) = \log(1+e^{a})$$

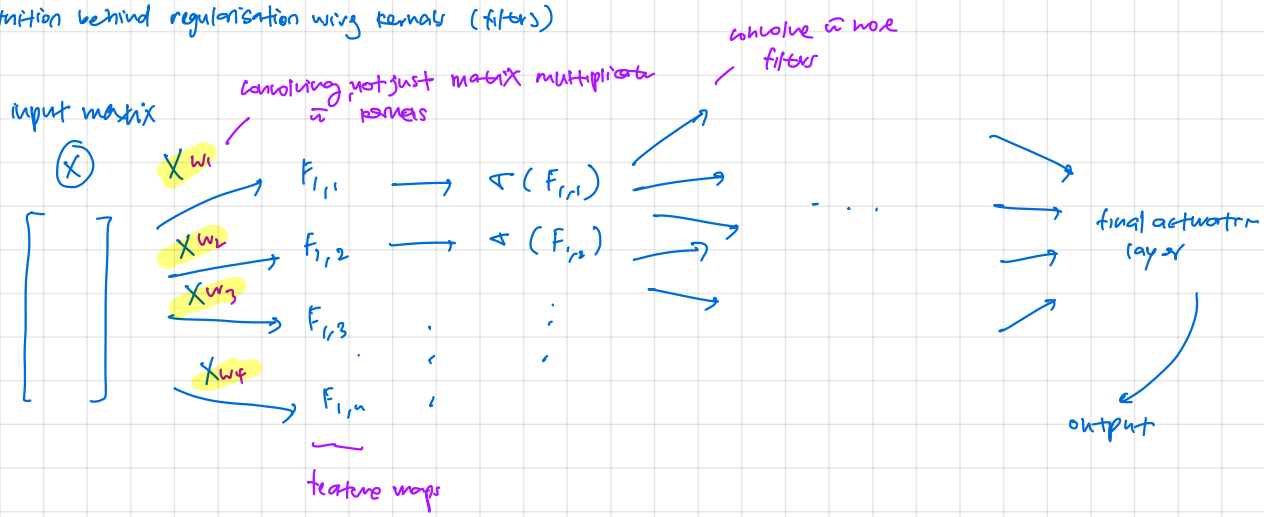


④ convolutional neural networks

↳ a special kind of neural network that reduces the number of parameters in a deep neural networks without losing too much quality. Good for image classification.

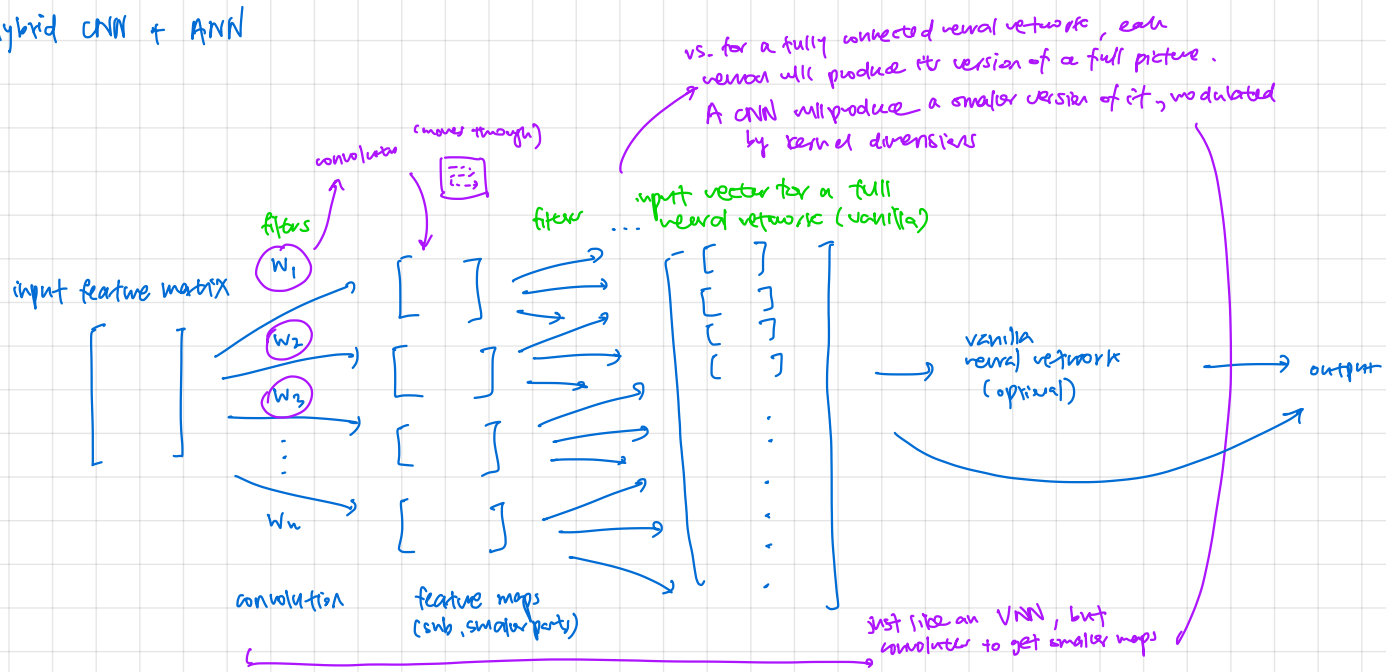
⇒ essentially a regularised vanilla neural network

1) intuition behind regularisation using kernels (filters)



↳ in principle, it works the same way as a vanilla neural network. only difference is that instead of $w_1 \times [x] \Rightarrow$ you convolve using a smaller $w \Rightarrow$ less w to learn

2) Hybrid CNN + ANN



① filters identify specific parts of the original input to look at, just as a normal vanilla network would → except that it always reduces the dimension of its output ⇒ fully connected neural net work does not

↳ regularisation parameter is kernel (filter) size

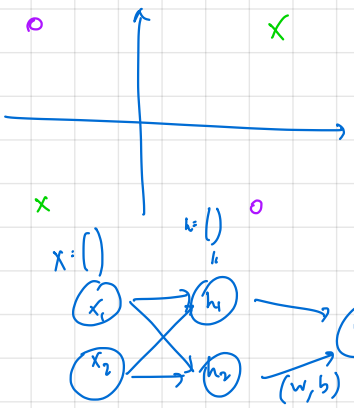
↳ by using filters, we identify features and subfeatures before putting them into a fully connected network to learn deeply about those

↳ regularise by reducing size (blending ⇒ avoid overfitting to specific patterns

② 'half' of work done by filters, then ANN. overall, less weights to learn during back propagation because of the dimensionality reduction

Example of a forward pass

XOR example



$f_1 = \sigma(W, c, X)$ $\quad f_2 = (h, w, b) = w^T h + b$

$= \sigma(W^T X + c)$

$f(x) = w^T \cdot \sigma(W^T x + c) + b$

$H = \sigma(XW + c)$

$$\sigma\left(\begin{bmatrix} -1 & -1 \\ -1 & -1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix}\right)$$

- relu → negative to 0

$$\sigma\left(\begin{bmatrix} -1 & -3 \\ 1 & -1 \\ 1 & 1 \\ 3 & 1 \end{bmatrix}\right)$$

non-linearity introduced by NN allows for better results

output of 1st activation $\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 3 & 1 \end{bmatrix}$

W → to output

$y = w^T \sigma(H) + b$

$$= \begin{bmatrix} 2 & -6 \end{bmatrix}\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 3 & 1 \end{bmatrix} + \begin{pmatrix} -1 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 \\ 2 \\ 2 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ +1 \\ +1 \\ -1 \end{pmatrix}$$

Training and evaluation

① workflow

1) training set → validation set → test set

- training set: learning model parameters
- validation set: pre-test test to help you tune hyperparameters
- test set: independent set to test model ability to generalise

avoid overfitting

2) hyperparameter tuning using validation set

↳ how to choose best model parameters ?

grid search

↳ using possible values of each parameter in lg scale

↳ train all possible combinations of those values

↳ choose the one w best metric

computationally expensive

random search

↳ randomly sample from grid of possible combinations of parameters, with a statistical distribution set to each parameter to sample from

↳ choose the one w best metric

Bayesian hyperparameter optimization

↳ use past evaluation results to choose next value to evaluate.

↳ limits number of expensive optimizations by choosing hyperparameters based on those that have done well

3) insufficient data for training and validation → using cross validation

↳ with limited data, preference is to use more for training ⇒ how to validate and carry out hyperparameter tuning ?

choose hyperparameter values to test for tuning → split training set into n subsets.

use n-1 of them for training, 1 as a validation set.

n-fold cross validation

→ train and validate

→ repeat for all combinations of train and validation

→ this is necessary to get a good representation of all the training data you have into the model (and its performance) while being able to test its performance for tuning in an unbiased way

avg. metric is ← value of metric that represents this set of hyperparameter values

4) leave-one-out cross validation

② evaluation metrics for regression

1) MSE = $\frac{\sum (y_i - \hat{y})^2}{n}$ — the square is such that extreme deviations will have larger values ⇒ use RMSE to evaluate when large errors are particularly undesirable

2) MAE = $\frac{\sum |y_i - \hat{y}|}{n}$ — when all deviations have 'equal weight'

3) MBE = $\frac{\sum y_i - \hat{y}}{n}$ — usually intended to measure model bias → but should be interpreted w variance of residuals, since positive and negative values cancel and we will not see the spread of errors.

③ evaluation metrics for classification

1) confusion matrix and rates

	positive $\hat{P}$	negative $\hat{N}$
positive	TP	FN
negative	FP	TN

recall = $\frac{TP}{TP+FN}$ — model's ability to predict positive for the actual positive.

precision = $\frac{TP}{TP+FP}$ — model's precision in terms of its accuracy rate for positive predictions

accuracy = $\frac{TP+TN}{TP+FP+FN+TN}$ — accuracy in terms of correct predictions out of all predictions

$TPR = \frac{TP}{TP+FN} = \frac{TP}{\text{total } P}$

$FNR = \frac{FN}{TP+FN} = \frac{FN}{\text{total } P}$

$TPR + FNR = 1$

$TNR = \frac{TN}{FP+TN} = \frac{TN}{\text{total } N}$

$FPR = \frac{FP}{FP+TN} = \frac{FP}{\text{total } N}$

$TNR + FPR = 1$

2) cost-sensitive rates

↳ assign weights to rates depending on context

cost matrix

	$\hat{P}$	$\hat{N}$
positive	$C_{pp}\ TP$	$C_{pn}\ FN$
negative	$C_{np}\ FP$	$C_{nn}\ TN$

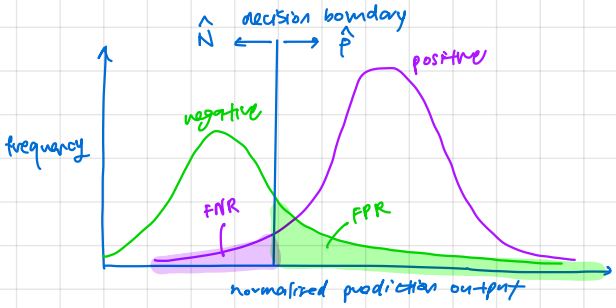
cost sensitive accuracy

$$\text{accuracy} = \frac{C_{pp}\,TP + C_{nn}\,TN}{C_{pp}\,TP + C_{nn}\,TN + C_{np}\,FP + C_{pn}\,FN}$$

3) error types and managing the tradeoff

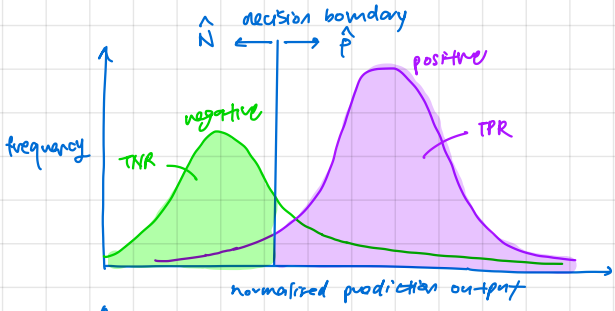
↳ H_0 : y = negative , H_1 : y = positive

- type I error: wrongly rejecting H_0 when H_0 is true → FNR
- failing to reject H_0 when H_0 is false → FPR

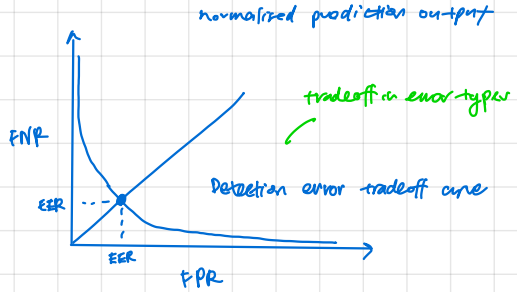


FNR & FPR → shifting decision boundary manages tradeoff between type I and type II error

- equal-error rate (EER): where decision boundary is such that FPR = FNR
- ↓ depends on application which to minimize
- ⇒ e.g. FN more severe than FP for covid testing, use cost-sensitive metrics

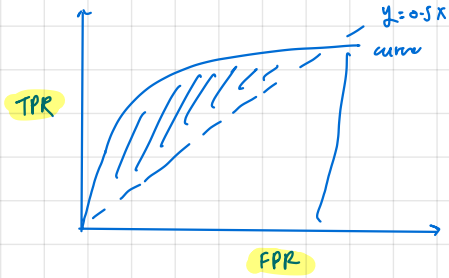


⇒



4) ROC curves only for classifiers that return a confidence / probability score

↳ modeling model's performance by looking at its ability to predict positives

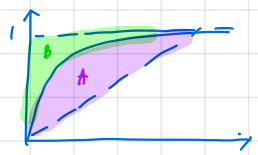


intuition: different curves for different model- and hyper-parameters due to different performance.

↳ if model is perfect, then TPR = 1

↳ if model is as if random, then TPR = FPR = 0.5

Gini coefficient



$$\text{Gini coefficient} = \frac{A}{A+B}$$

⇒ if G = 1, then A is much better than a random classifier since TPR = 1

⇒ if G = 0, then classifier performs along linear line — like a random classifier, AUC = 0.5

Area under ROC curve

⇒ if area under ROC curve = 0, then TPR = 0, 100% wrong prediction.

⇒ if AUC = 1, then TPR = 1, 100% correct positive prediction

↓ mathematically:

$$e_{ij} = f(X_i^+) - f(X_j^-)$$

error term e_{ij} is the difference between prediction of positive datum X_i^+ and prediction of negative datum X_j^-

$$u(e) = \begin{cases} 1 & \text{if } e > 0 \\ 0.5 & \text{if } e = 0 \\ 0 & \text{if } e < 0 \end{cases} \quad \text{(0.5: random)}$$

$$AUC = \frac{1}{m^+ m^-} \sum_{i}^{m^+} \sum_{j}^{m^-} u(e_{ij})$$

AUC = 1

if all e_{ij} are positive → $u(e_{ij}) = 1$

↳ all classifications are correct since threshold separates all positive from negative samples