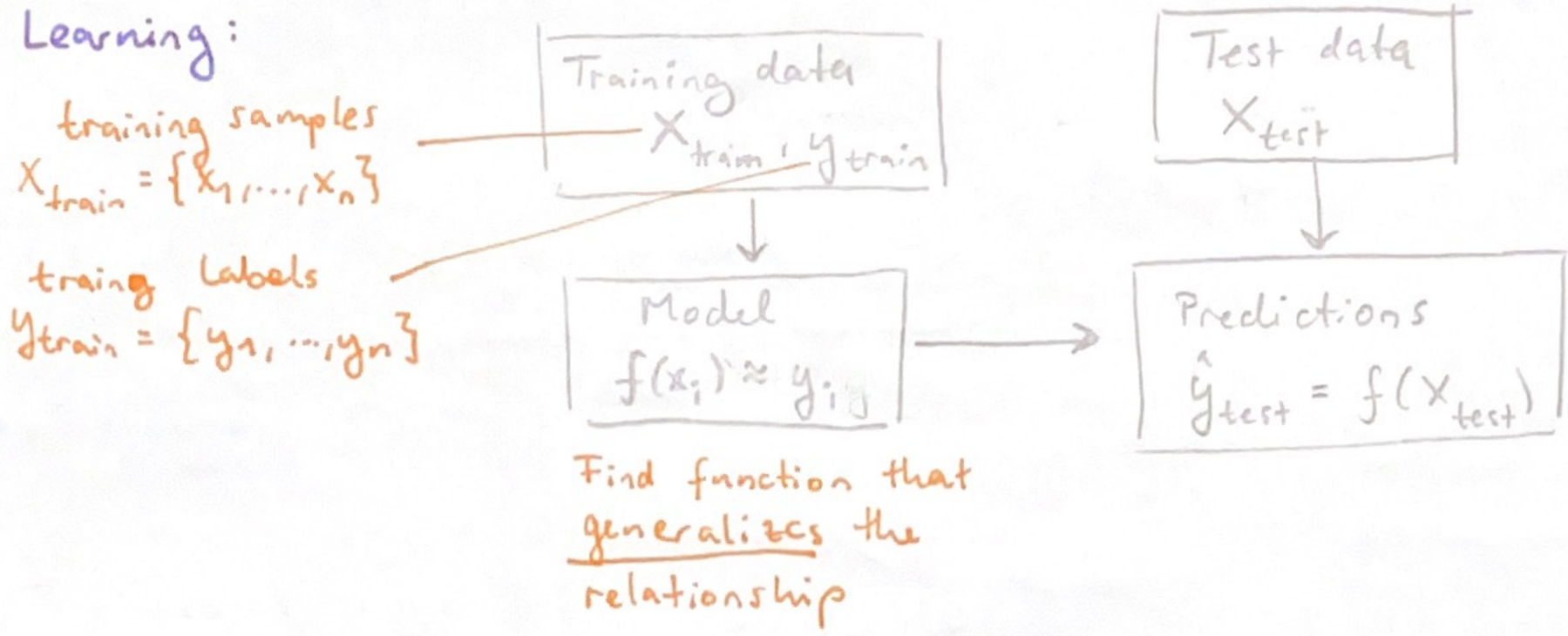
ML 01 : Intro

- Supervised Learning:



- Classification: If target y_i represents categories.
- Regression: If the targets y_i represent continuous numbers.
- Unsupervised Learning: Finding a structure in unlabelled data.
  - ↳ Clustering
  - ↳ Dimensionality Reduction

ML 02 : KNN

- 1-NN-Algorithm: Given a dataset $D = \{(\vec{x}_i, y_i)\}_{i=1}^{N}$

  → to classify new observations
  1. Define distance measure
  2. Compute NN
  3. Label with the NN's label

- k-NN-Algorithm: Look at multiple NNs and pick the majority label.

prob. of belonging in class c

$$p(y=c \mid \vec{x}, k) = \frac{1}{k} \sum_{i \in N_k(x)} \mathbb{I}(y_i = c) \quad \text{with} \quad \mathbb{I}(e) = \begin{cases} 1, & \text{if true} \\ 0, & \text{else} \end{cases}$$

$\vec{x}$ and k given

k nearest Neighboors of x

Classification

$$\Rightarrow \hat{y} = \operatorname{argmax}_c p(y=c \mid \vec{x}, k)$$

Complexity per prediction: $\mathcal{O}(n)$

⇒ more robust against errors in the training set (generalization)

- Wheighted K-NN: Use $p(y=c \mid \vec{x}, k) = \frac{1}{z} \sum_{i \in N_k(x)} \frac{1}{d(x_i, x)} \mathbb{I}(y_i = c)$ with $z = \sum_{x \in N_k(x)} \frac{1}{d(x_i, x)}$

distance measure

Normalization wheight → sum to 1

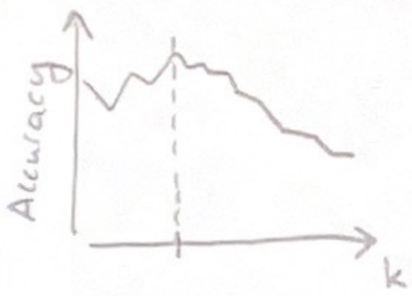
- k-NN-regression:

$$\hat{y} = \frac{1}{z} \sum_{i \in N_k(x)} \frac{1}{d(x, x_i)} y_i$$

"weighted mean"

- How to choose k?

Training set D_{tr}	Validation set D_v	Test set D_t
→ learn model	→ evaluate performance and pick k	→ final performance



Sampling → Random
↘ Stratified (proportional representations of classes)

- How to measure classification performance?

→ Confusion table

Predicted	True label $y=1$	True label $y=0$
$y=1$	TP	FP
$y=0$	FN	TN

True positives & True negatives

=> Accuracy: $acc = \frac{TP+TN}{TP+TN+FP+FN}$

Trade-off:
- Precision: $prec = \frac{TP}{TP+FP}$
- Sensitivity / Recall: $rec = \frac{TP}{TP+FN}$

...

F1 Score: $f1 = \frac{2 \cdot prec \cdot rec}{prec + rec}$ (tries to max. prec & rec)

- Distance measures:

* Euclidean L_2: $\sqrt{\sum_i (u_i - v_i)^2}$
* L_1 norm: $\sum_i |u_i - v_i|$
* L_∞ norm: $\max_i |u_i - v_i|$
* Angle: $\cos \alpha = \frac{u^T v}{\|u\| \cdot \|v\|}$

(is equal to Euclidean distance on Normalized data)

* Mahalanobis distance: $\sqrt{(u-v)^T \Sigma^{-1} (u-v)}$ (Σ: Covariance matrix)

- How to circumvent scaling issues?

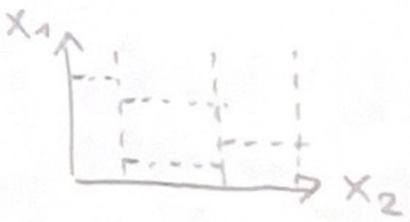
→ Data standardization: $x_{i,std} = \frac{x_i - \mu_i}{\sigma_i}$ (Scale each feature to zero mean and unit variance)

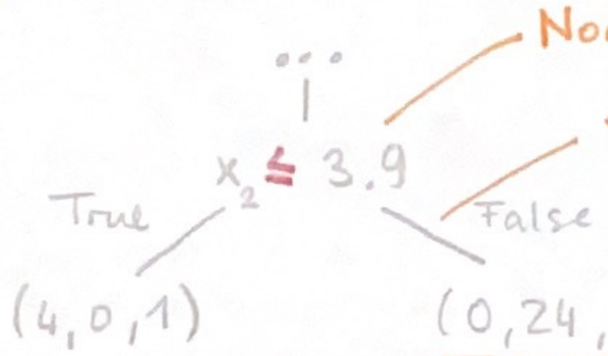
- Curse of dimensionality: With an increase in dims the space becomes more empty.
(in kNN the NN is far away → N would have to grow exponentially with the num of features)

ML 02: Decision Trees (DT)

- Decision tree:

* partition the input space into cuboid regions.





Node ≙ feature test (binary split)

Branch ≙ different outcome of the feature test

Leaf ≙ final nodes and the represented region in the input space.

Inference:
Predicting a new sample.

- Probability for classifying:

$$p(y=c|R) = \frac{n_{c,R}}{\sum_{c_i \in C} n_{c_i,R}}$$

$p(y=c|R)$: Probability that a point should be in a class, given the region of the input space.

$n_{c,R}$: Number of points in a cluster and region

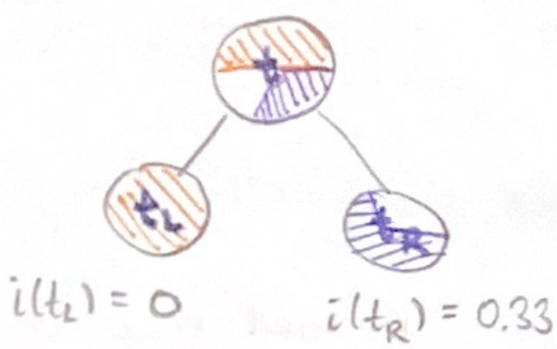
$\sum_{c_i \in C} n_{c_i,R}$: points of all classes in this region

Inference

$$\Rightarrow \hat{y} = \underset{c}{\operatorname{argmax}}\, p(y=c|R) = \underset{c}{\operatorname{argmax}}\, n_{c,R}$$

- How are decision trees built?

→ Greedy top-down approach (choosing the best split at each node)



⇒ Split the node if it improves the misclassification rate

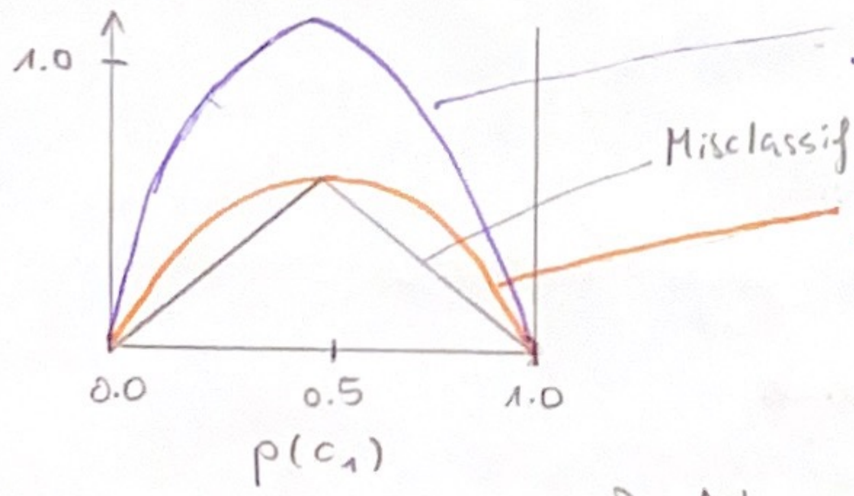
$$i_E(t) = 1 - \max_c p(y=c|t)$$

⇒ Improvement of split: $\Delta i_E(s,t) = i_E(t) - p_L \cdot i_E(t_L) - p_R \cdot i_E(t_R)$

(split, node, proportions)

⇒ Repeat the heuristic (split where $\Delta i(s,t)$ is largest)

- Impurity measures: ("criterion") (for categorical features)



For $C=\{c_1,c_2\}$

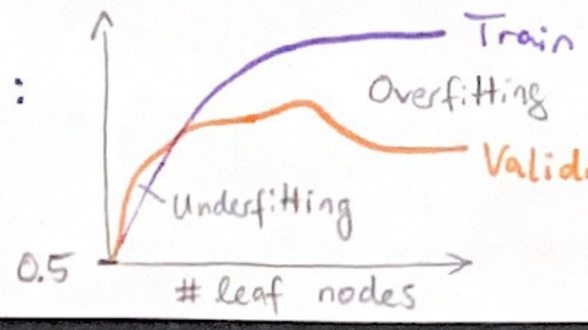
Entropy: $i_H(t) = -\sum_{c_i \in C} \pi_{c_i} \log_2 \pi_{c_i}$

Gini Index: $i_G(t) = \sum_{c_i \in C} \pi_{c_i}(1-\pi_{c_i}) = 1 - \sum_{c_i \in C} \pi_{c_i}^2$

with $\pi_c = p(y=c|t)$

⇒ Advantage: The error $i(t)$ for unpure nodes is exponentially greater than for pure nodes.
(Gini Index is a bit faster than Entropy)

- Overfitting:



→ tells how well the model generalizes

- K-fold Cross-Validation:

  * Split data into K folds (e.g., 10 ⇒ train 10 Models)
    ↳ Use K-1 for training and 1 for eval.
  ⇒ Average over all folds to estimate → best hyperparams
                                       → model

  [Extreme Case: leave-one-out-cross-validation (LOOCV)
  = N-fold CV if N = # of samples]

- When to stop growing a decision tree?

pre-pruning:
→ Stopping criteria:
  * distribution in branch is pure: $i(t) = 0$ (→ overfitting)
  * max. depth
  * # of samples in node below threshold
  * benefit of splitting below threshold $\Delta i(s,t) < t_\Delta$
  * accuracy on the validation set

→ Alternative: Grow fully + (Post-) pruning

- Reduced-Error Pruning:

1) Get the entire tree's error estimate on the valid. set: $err_{D_v}(T)$
2) For each node, calculate: $err_{D_v}(T/T_t)$ — pruned tree at node t
3) Prune at node with highest error reduction
4) Repeat until $\forall t : err_{D_v}(T_i) < err_{D_v}(T/T_t)$

- Regression in DTs: * Compute mean at the leafs
  * MSE as splitting heuristic

- What's an ensemble? → Aggregate prediction from multiple models.
  ↳ Reduces Variance

  * Bagging (bootstrap aggregating):
    → Train separate models on sampled sub-data-sets
    → combine predictions
    e.g. Random Forests ↳ also use a sampled subset of features per tree

  * Boosting:
    → Incrementally train weak classifiers that correct previous mistakes ("a chain")
    → Give higher weight to misclassified examples

Rule of thumb:

	# of features
Classif.	$\log \sqrt{d}$
Regr.	$\log_2(d)$

ML 03: Probabilistic Inference

- Maximum Likelihood Estimation:

Example:
Random experiment with 3 coin flips. With $p(T) = \theta$.
The probability of a sequence of flips given by the parameters of some model:

$p(\underbrace{HTT}_{\text{observed data, D}} | \theta_1, \theta_2, \theta_3)$ = The likelihood of seeing $\underbrace{H,T,T}_{\text{the observations}}$

$\Rightarrow$ Find the θ_i's such that $p(HTT|\theta_1,\theta_2,\theta_3)$ is as large as possible.

> Max. Likelihood: Find the parameters that maximise the likelihood of our observation.

Assumption 1: The flips are (the) identical distribution.

$\rightarrow p_i(F_i = f_i | \theta_i) = p(F_i = f_i | \theta)$

(F_i: random variable; f_i: outcome)

Assumption 2: The flips are independent.

$\Rightarrow p(HTT|\theta) = p(H|\theta) \cdot p(T|\theta) \cdot p(T|\theta)$

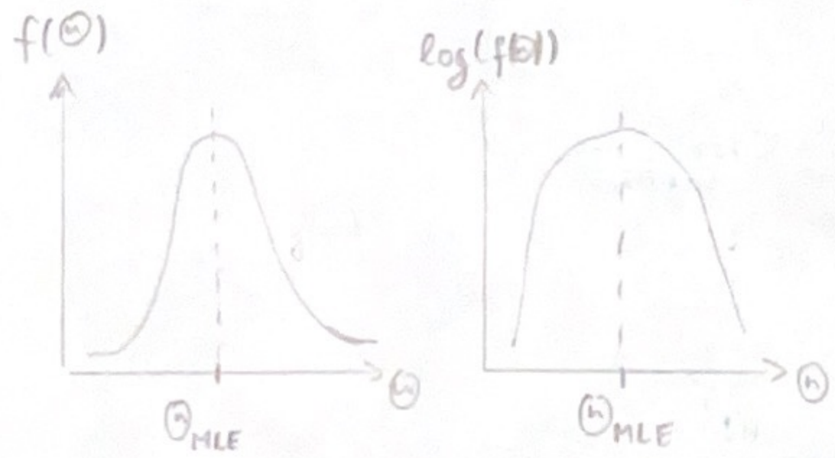
$= \prod_{i=1}^{3} p(F_i = f_i | \theta)$ #

Total assumption: The flips are independent and identically distributed (i.i.d.).

Note: This is not a probability distribution (does not integrate to 1)

$\Rightarrow$ Interpret $p(D|\theta)$ as a function $f(\theta)$:

Find the maximum of this function:



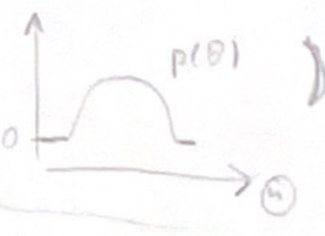
Monotonic functions preserve critical points

$\theta_{MLE} = \underset{\theta \in [0,1]}{\operatorname{argmax}} f(\theta) = \underset{\theta \in [0,1]}{\operatorname{argmax}} \log(f(\theta))$

$\dots \downarrow = \frac{|T|}{|H|+|T|}$ for the given Bernoulli coin example

- Bayesian inference:

$\rightarrow$ A prior distribution $p(\theta)$ represents our beliefs about θ before we observe any data.

(= basically a distribution/likelihood of values for θ; e.g. 

)

constraints:
1) Does not depend on data
2) $p(\theta) \geq 0 \quad \forall \theta$
3) $\int p(\theta) d\theta = 1$

→ The posterior distribution encodes our beliefs in Θ after observing the data D.

$$p(\Theta \mid D) = \frac{p(D \mid \Theta) \cdot p(\Theta)}{p(D)}$$

($p(D \mid \Theta)$: likelihood of D given Θ; $p(\Theta)$: prior distribution; $p(D)$: Evidence distribution / marginal likelihood)

$$\text{posterior} \propto \text{likelihood} \cdot \text{prior}$$

* The likelihood dominates the posterior with more data.

- Maximum a posteriori estimation (MAP): [MLE ignores prior beliefs and performs poorly if little data is available.]

$$\Rightarrow \Theta_{MAP} = \arg\max_{\Theta} p(\Theta \mid D)$$

$$= \arg\max_{\Theta} \frac{p(D \mid \Theta)\, p(\Theta)}{p(D)} = \arg\max_{\Theta} p(D \mid \Theta)\, p(\Theta)$$

($p(D)$: the constant does not matter)

The prior $p(\Theta)$ is chosen to be a Beta distribution (for simpler calculations)

$$\text{Beta}(\Theta \mid a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \Theta^{a-1}(1-\Theta)^{b-1}, \quad \Theta \in [0,1]$$

(a, b: distribution parameters, $a, b > 0$; Γ: Gamma function, $\Gamma(n) = (n-1)!$ for $n \in \mathbb{N}$)

$$\Rightarrow \Theta_{MAP} = \arg\max_{\Theta} p(D \mid \Theta) \cdot \underbrace{\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}}_{\text{constant}} \Theta^{a-1}(1-\Theta)^{b-1}$$

$$= \arg\max_{\Theta} \underbrace{p(D \mid \Theta)}_{\text{In the Bernoulli-Experiment}} \cdot \Theta^{a-1}(1-\Theta)^{b-1}$$

$$= \Theta^{|T|}(1-\Theta)^{|H|}$$

($|T|$: # of tails, $|H|$: # of heads)

$$= \arg\max_{\Theta} \Theta^{|T|+a-1}(1-\Theta)^{|H|+b-1}$$

$$= \arg\max_{\Theta} (|T|+a-1)\log\Theta + (|H|+b-1)\log(1-\Theta)$$

$$\vdots$$

$$= \frac{|T|+a-1}{|H|+|T|+a+b-2}$$

(solution for the given Bernoulli coin example)

The data dominates this for a lot of data

- How to estimate the posterior distribution:

→ One wants to know the entire posterior distribution $p(\Theta|D)$:

↳ How certain are we in our estimate θ_{MAP}?

→ What is the probability that θ is in some interval?

$$\left[p(\theta|D) \propto \theta^{|T|+a-1} (1-\theta)^{|H|+b-1} \right]$$

⇒ We need the normalization constant, such that

$$\int p(\theta|D)\, d\theta \overset{!}{=} 1$$

Method 1: Compute Integral and divide by it

Method 2: Compare the unnormalized posterior to a known PDF (e.g., Beta distribution)

$$\Rightarrow \underline{\underline{p(\theta|D) = \text{Beta}(\theta \mid a+|T|, b+|H|)}}$$

Solution for the given coin example

* With more data the posterior becomes more peaky
→ more certain of estimate for θ

- Posterior predictive distribution:

With the outcome of the next flip $f \in \{0,1\}$

$$p(F=f|D,a,b) = p(f|D,a,b)$$

Sum rule of probability:
$$= \int_0^1 p(f,\theta|D,a,b)\, d\theta$$

Conditional independence:
$$= \int_0^1 p(f|\theta,D,a,b)\, p(\theta|D,a,b)\, d\theta$$

?
$$= \int_0^1 p(f|\theta)\, p(\theta|D,a,b)\, d\theta$$

...
$$= \frac{(|T|+a)^f (|H|+b)^{(1-f)}}{|T|+a+|H|+b}$$

$p(f|\theta) = \text{Ber}(f|\theta)$
$= \theta^f (1-\theta)^{1-f}$

$$= \text{Ber}\left(f \,\middle|\, \frac{|T|+a}{|T|+a+|H|+b}\right)$$

The approach that computes the full Bayes posterior distribution

„Fully Bayesian"

⇒ With lots of data, the differences in predictions between MLE, MAP and fully Bayesian become less noticeable.

ML 04: Linear Regression

- What's the linear regression problem?

Given: → observations $X = \{x_1, \dots, x_N\}$

targets $y = \{y_1, \dots, y_N\}$

Find: Mapping from Inputs to targets: $y_i \approx f(x_i)$

→ which generalizes to new x

- What's the lin. model?

bias weights

$$f_w(\vec{x}_i) = w_0 + \vec{w}^T \vec{x}_i$$

$$= (w_0, w_1, \dots, w_D) \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix} = \tilde{\vec{w}}^T \tilde{\vec{x}}$$

Notation: Assume the bias term is always absorbed and write $\vec{w}$ and $\vec{x}$.

- How is the ~~solver~~ optimal lin. model found?

1) Loss function, e.g. least squares: $E_{LS}(\vec{w}) = \frac{1}{2} \sum_{i=1}^{N} (\vec{w}^T \vec{x}_i - \vec{y}_i)$

↳ Measures the misfit between the model and observations

2) Objective: $w^* = \arg\min_w E_{LS}(w)$

$= \arg\min_w \frac{1}{2} (X\vec{w} - \vec{y})^T (X\vec{w} - \vec{y})$

multiple observations $\vec{x}_i^T$ stacked into a matrix

3) Optimal solution: $\nabla_w E_{LS}(\vec{w}) \overset{\text{Slide 10}}{=} X^T X \vec{w} - X^T \vec{y} \overset{!}{=} 0$

$\rightarrow w^* = (X^T X)^{-1} X^T \vec{y}$

- What if the dependency between y and $\vec{x}$ is not linear?

→ Choose a universal function approximator.

f is still lin. in $\vec{w}$ but not lin. in x

$$f_{\vec{w}}(\vec{x}) = w_0 + \sum_{j=1}^{M} w_j \phi_j(\vec{x})$$

Absorb bias $\phi_0 = 1$

$$= \vec{w}^T \vec{\phi}(\vec{x})$$

with $\phi_j(x)$ as some non-linear function, e.g. $\phi_1(x) = \sin(x)$

"transform the basis"

→ Typical basis functions:

* Polynomials $\phi_j(x) = x^j$
* Gaussian $\phi_j(x) = e^{\frac{-(x-\mu_j)^2}{2s^2}}$
* Logistic Sigmoid $\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$ with $\sigma(a) = \frac{1}{1+e^{-a}}$

Design matrix:

$$\Phi = \begin{pmatrix} \overbrace{\phi_0(x_1)}^{=1} & \phi_1(x_1) & \cdots & \phi_M(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_M(x_N) \end{pmatrix}$$

$\in \mathbb{R}^{N \times (M+1)}$

$\frac{1}{2}\sum(\vec{w}^T\vec{\phi}(\vec{x}_i) - y_i)^2$

=D 2) Objective: $E_{LS}(\vec{w}) = \frac{1}{2}(\Phi\vec{w} - \vec{y})^T(\Phi\vec{w} - \vec{y})$

Design matrix

3) Solution: $w^* = (\Phi^T\Phi)^{-1}\Phi^T\vec{y}$

How is the degree of the polynomial chosen?

M → too small => underfitting
M → too large => overfitting

} =D Use training & validation sets to optimize performance

Observation: Overfitting occurs when the coefficients $\vec{w}$ become too large.

→ Controlling overfitting with regularization:

Regularization with
.. L_2 leads to overall smaller $\vec{w}$
.. L_1 leads to sparse $\vec{w}$ (lots of entries =0)

L2-Regularization/ Ridge Regression

regularization strength

$$E_{ridge}(\vec{w}) = \frac{1}{2}\sum^{N}(\vec{w}^T\vec{\phi}(\vec{x}_i) - \vec{y}_i) + \frac{\lambda}{2}\|\vec{w}\|_2^2$$

squared L2 norm

→ Larger regularization strength λ leads to smaller weights $\vec{w}$

- What's the tradeoff between bias and variance?

total error { Bias: expected error due to model mismatch
Variance: variation due to high randomness in the data

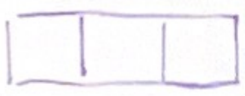
("Sensitivity of model towards noise in the data")

("Preference for certain types of func.")

→ Model is too rigid to fit the data distribution
↳ model is misspecified or the regularization strength is too large

Each dot is the solution of a trained model (on a different subset of the training data)

→ model is too flexible and thus captures noise in the data
↳ this is overfitting
↳ If the model has a high capacity or λ is too low



→ Tradeoff: 1) Select a model with high capacity
2) keep the variance low by choosing a good λ

ML 04: Probabilistic linear regression

* Given data is sampled from a function plus noise

$$\Rightarrow y_i = f_{\vec{w}}(\vec{x}_i) + \underbrace{\varepsilon_i}_{\text{noise}}, \quad \varepsilon_i \sim \mathcal{N}(0, \beta^{-1}) \quad \text{with } \beta = \frac{1}{\sigma^2}$$

↑ "fixed precision"

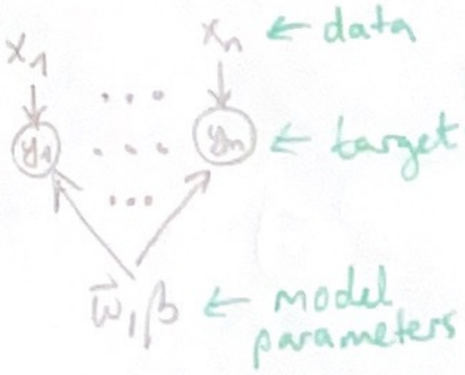
$$\Rightarrow y_i \sim \mathcal{N}(f_{\vec{w}}(\vec{x}_i), \beta^{-1})$$

↳ Gaussian distribution

- With MLE:

Likelihood

Likelihood of a single sample: $p(y_i \mid f_{\vec{w}}(\vec{x}_i), \beta) = \mathcal{N}(y_i \mid f_{\vec{w}}(\vec{x}_i), \beta^{-1})$



$\Rightarrow$ Likelihood of entire dataset: $p(\vec{y} \mid X, \vec{w}, \beta) = \prod_{i=1}^{N} p(y_i \mid f_{\vec{w}}(\vec{x}_i), \beta)$

↑ independent

Maximum Likelihood

$$\Rightarrow \vec{w}_{ML}, \beta_{ML} = \underset{\vec{w},\beta}{\operatorname{argmax}}\ p(\vec{y} \mid X, \vec{w}, \beta)$$

ln is monotonic ↘

$$= \underset{\vec{w},\beta}{\operatorname{argmin}}\ \underbrace{-\ln\left(p(\vec{y} \mid X, \vec{w}, \beta)\right)}_{= E_{ML}(\vec{w}, \beta)}$$

maximum likelihood error function

$$\Rightarrow E_{ML}(\vec{w}, \beta) = -\ln\left[\prod_{i=1}^{N} \underbrace{\sqrt{\frac{\beta}{2\pi}} \exp\left(-\frac{\beta}{2}(\vec{w}^T \vec{\phi}(\vec{x}_i) - y_i)^2\right)}_{\text{Gaussian}}\right]$$

likelihood of all data

expand + log rules ↘

$$= \frac{\beta}{2} \sum_{i=1}^{N} \left[(\vec{w}^T \vec{\phi}(\vec{x}_i) - y_i)^2\right] - \frac{N}{2} \ln\beta + \frac{N}{2} \ln 2\pi$$

Optimize the likelihood w.r.t. $\vec{w}$

(with respect to)

$$\Rightarrow \vec{w}_{ML} = \underset{\vec{w}}{\operatorname{argmin}}\ E_{ML}(\vec{w}, \beta)$$

$$= \dots = \underset{\vec{w}}{\operatorname{argmin}}\ \frac{1}{2} \sum_{i=1}^{N} \underbrace{(\vec{w}^T \vec{\phi}(\vec{x}_i) - y_i)^2}_{\text{This is the least squares error fn!}} = \underset{\vec{w}}{\operatorname{argmin}}\ E_{LS}(\vec{w})$$

$\Rightarrow$ Maximizing the likelihood is equivalent to minimizing the least squares error function.

Optimize the likelihood w.r.t. β

$$\beta_{ML} = \underset{\beta}{\operatorname{argmin}}\ E_{ML}(\underbrace{\vec{w}_{ML}}_{\text{already minimizes for } \vec{w}}, \beta)$$

Derivative

$$\Rightarrow \frac{\partial}{\partial\beta} E_{ML}(\vec{w}_{ML}, \beta) = \dots \overset{!}{=} 0$$

Solve for β

$$\Rightarrow \frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{i=1}^{N} (\vec{w}_{ML}^T \vec{\phi}(\vec{x}_i) - y_i)^2$$

This ist the mean residual/error

$\Rightarrow$ Advantage compared to "regular" lin. regression:
Besides the prediction we get a confidence value for it.

- With MAP:

Connection to the coin flip example:

	train data	likelihood	prior	posterior
Coin	$D = X$	$p(D\|\vec{\theta})$	$p(\theta\|a,b)$	$p(\theta\|D)$
Regr.	$D = \{X, \vec{y}\}$	$p(\vec{y}\|X,\vec{w},\beta)$	$p(\vec{w}\|\cdot)$	$p(\vec{w}\|X,y,\beta,\cdot)$

RECAP :D

$$\underbrace{p(\vec{w}|X,\vec{y},\beta)}_{\text{posterior}} = \frac{\overbrace{p(\vec{y}|X,\vec{w},\beta)}^{\text{likelihood}} \cdot \overbrace{p(\vec{w}|\cdot)}^{\text{prior}}}{\underbrace{p(\vec{y}|X,\beta,\cdot)}_{\text{Normalization constant}}} \propto \text{likelihood} \cdot \text{prior}$$

Pick prior $p(\vec{w}|\cdot)$

$$p(\vec{w}|a) = \mathcal{N}(\vec{w}|0, \alpha^{-1}I) = \left(\frac{\alpha}{2\pi}\right)^{\frac{M}{2}} \exp\left(-\frac{\alpha}{2}\vec{w}^T\vec{w}\right)$$

(M – # of elements in $\vec{w}$; α – precision)

Zero mean → small values of w → reduces overfitting

optimize the posterior w.r.t. $\vec{w}$

(This ex. assumes that β is given)

$$\vec{w}_{MAP} = \underset{\vec{w}}{\operatorname{argmax}}\ p(\vec{w}|X,\vec{y},\beta)$$

plug in + log rules + remove constant

$$= \dots = \underset{\vec{w}}{\operatorname{argmin}} \underbrace{-\ln[p(\vec{y}|X,\vec{w},\beta)] - \ln[p(\vec{w}|a)]}_{= E_{MAP}(\vec{w})}$$

$$E_{MAP} = \dots \propto E_{ridge}(\vec{w}) + \text{const.}$$

If the prior is Gaussian

- With fully Bayesian approach:

→ Computes the full posterior distribution

$$p(\vec{w}|D) \propto p(\vec{y}|X,\vec{w},\beta) \cdot p(\vec{w}|\alpha)$$

Conjugate prior: Since the prior and likelihood are Gaussians, the posterior → $p(\vec{w}|D) = \mathcal{N}(\vec{w}|\vec{\mu},\Sigma)$ is also a Gaussian! (in the given example)

with $\mu = \beta\Sigma\Phi^T y$

$\Sigma^{-1} = \alpha I + \beta\Phi^T\Phi$

- How to predict new data?

Plug in the computed parameters

With MLE: $p(\hat{y}_{new}|\vec{x}_{new},\vec{w}_{ML},\beta_{ML}) = \mathcal{N}(\hat{y}_{new}|\vec{w}_{ML}^T\vec{\phi}(\vec{x}_{new}), \beta_{ML}^{-1})$

With MAP: $p(\hat{y}_{new}|\vec{x}_{new},\vec{w}_{MAP},\beta) = \mathcal{N}(\hat{y}_{new}|\vec{w}_{MAP}^T\vec{\phi}(\vec{x}_{new}), \beta^{-1})$

predictive distributions

Use the full posterior distribution to comp. the posterior predictive distribution: → also shows variance

$$p(\hat{y}_{new}|\vec{x}_{new},D) = \int p(\hat{y}_{new},\vec{w}|\vec{x}_{new},D)\,d\vec{w}$$

$$= \dots$$

ML 05: Linear Classification

- What's the classification problem?

Given: Observations $X = \{x_1, x_2, \dots, x_N\}$, $x_i \in \mathbb{R}^D$

Possible classes $C = \{1, \dots, C\}$

Labels $y = \{y_1, y_2, \dots, y_N\}$, $y_i \in C$

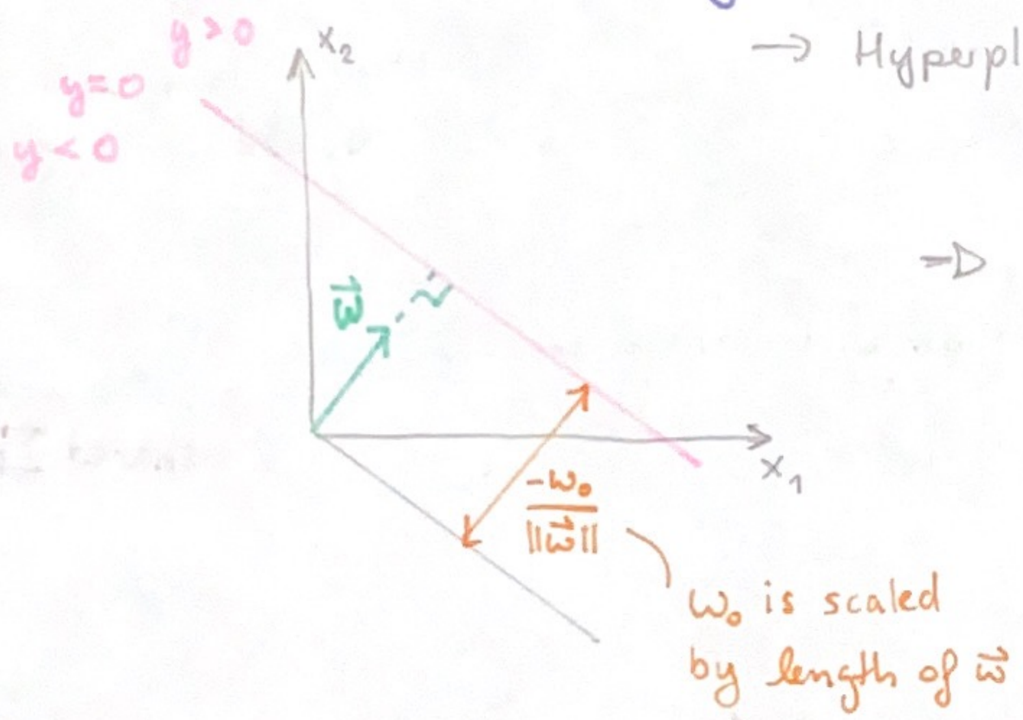
Find: function $f: \mathbb{R}^D \to C$ that maps x_i to y_i

- Zero-One-Loss: → denotes the number of misclassified examples

$$l_{01}(\vec{y}, \vec{y}) = \sum_{i=1}^{N} \mathbb{I}(\hat{y}_i \neq y_i)$$

$\mathbb{I}$ (Indicator) $= \begin{cases} 1, & \text{if condition is true} \\ 0, & \text{else} \end{cases}$

- Hyperplane as a decision boundary:



→ Hyperplane is defined by normal $\vec{\omega}$ and offset ω_0

⇒ $\vec{\omega}^T\vec{x} + \omega_0 \begin{cases} = 0, & \text{if } \vec{x} \text{ on plane} \\ > 0, & \text{if } \vec{x} \text{ on normal's side} \\ < 0, & \text{else} \end{cases}$

If a hyperplane can perfectly seperate all points into two classes, the data set is lin. separable

- Perceptron for binary classification:

$$\hat{y} = f(\vec{\omega}^T\vec{x} + \omega_0) \quad \text{with} \quad f(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{else} \end{cases}$$

Training: Init. $\vec{\omega}, \omega_0 \leftarrow \vec{0}$

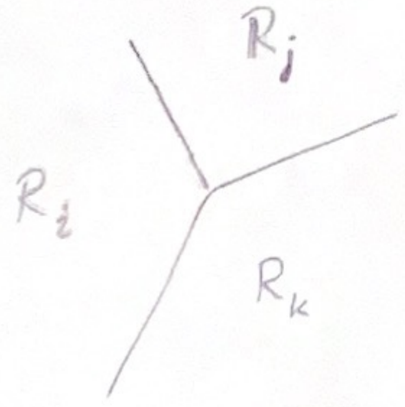
iterate until convergence:

For each misclassified sample update

$$\vec{\omega} \leftarrow \begin{cases} \vec{\omega} + \vec{x}_i & \text{if } y_i = 1 \\ \vec{\omega} - \vec{x}_i & \text{if } y_i = 0 \end{cases}$$

$$\omega_0 \leftarrow \begin{cases} \omega_0 + 1 & \text{if } y_i = 1 \\ \omega_0 - 1 & \text{if } y_i = 0 \end{cases}$$

- Multiclass discriminant:



$$\hat{y} = \arg\max_c f_c(\vec{x}) \quad \text{with} \quad f_c(\vec{x}) = \vec{\omega}_c^T\vec{x} + \omega_{0c}$$

Assign $\vec{x}$ to the class which produces the largest $f_c(\vec{x})$

Each class has a normal vector

→ need basis func. ϕ that maps data to a space where it is lin. seperable

⇒ doesn't work if the data is not lin. seperable

- Probabilistic models for classification:

1) Generative models
2) Discriminative models

$$p(y=c \mid \vec{x}) \overset{\text{Bayes}}{=} \frac{p(\vec{x} \mid y=c) \cdot p(y=c)}{p(\vec{x})}$$

- Probabilistic generative models for lin. classification:

Idea: Obtain posterior: $p(y=c \mid \vec{x}) \propto \underbrace{p(\vec{x} \mid y=c)}_{\text{class conditional}} \cdot \underbrace{p(y=c)}_{\text{class prior}}$

→ class conditional: distribution of samples in class c

→ class prior: probability of a point belonging to class c

1.) Choose parametric models $p(y=c \mid \vec{\Theta})$ and $p(\vec{x} \mid y=c, \vec{\Psi})$

Choose prior: $y \sim \text{Categorical}(\vec{\theta})$

$$\Rightarrow p(y) = \prod_{c=1}^{C} \vec{\theta}_c^{\,\mathbb{I}(y=c)} \quad \text{so that} \quad p(y=c) = \theta_c$$

$$\dots \Rightarrow \theta_c^{MLE} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(y_i = c)$$ "# of samples in class divided by # of all data"

Choose class conditionals: $\vec{x}$ is continuous

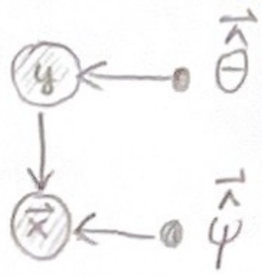
Multivariate normal for each class

$$\Rightarrow p(\vec{x} \mid y=c) = \mathcal{N}(\vec{x} \mid \vec{\mu}_c, \Sigma)$$ separate means but shared Σ

$$= \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\vec{x}-\vec{\mu}_c)^T \Sigma^{-1} (x-\mu_c)\right\}$$

2.) Estimate parameters $\{\vec{\Psi}, \vec{\Theta}\}$ from data (e.g. with MLE)

(3.) Possibility to generate new data:



1. Sample $y_{new} \sim p(y \mid \vec{\hat{\Theta}})$
2. Sample $x_{new} \sim p(\vec{x} \mid y = y_{new}, \vec{\hat{\Psi}})$

4.) Inference with LDA

Calculate $p(y=c \mid \vec{x})$:

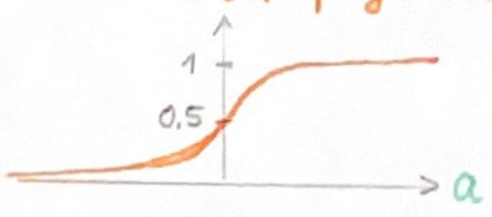
Assume two classes $C = \{0, 1\}$

$$p(y=1 \mid \vec{x}) = \frac{p(\vec{x} \mid y=1)\, p(y=1)}{p(\vec{x} \mid y=1)\, p(y=1) + p(\vec{x} \mid y=0)\, p(y=0)}$$

$$a = \log \frac{p(\vec{x} \mid y=1) p(y=1)}{p(\vec{x} \mid y=0) p(y=0)} \rightarrow = \frac{1}{1 + \exp(-a)} := \sigma(a)$$

$\Rightarrow y \mid \vec{x} \sim \text{Bernoulli}(\sigma(\vec{\omega}^T \vec{x} + \omega_0))$

$\sigma(a) = p(y=1 \mid \vec{x})$



Linear Discriminant Analysis (LDA) with

$$a = -\frac{1}{2}(\vec{x}-\vec{\mu}_1)^T \Sigma^{-1} (\vec{x}-\vec{\mu}_1) + \log p(y=1) + \frac{1}{2}(\vec{x}-\vec{\mu}_0)^T \Sigma^{-1} (\vec{x}-\vec{\mu}_0) \quad \text{(} - \log p(y=0) \text{)}$$

$$= \vec{\omega}^T \vec{x} + \omega_0 \overset{!}{=} 0 \Rightarrow \text{Decision Boundary}$$

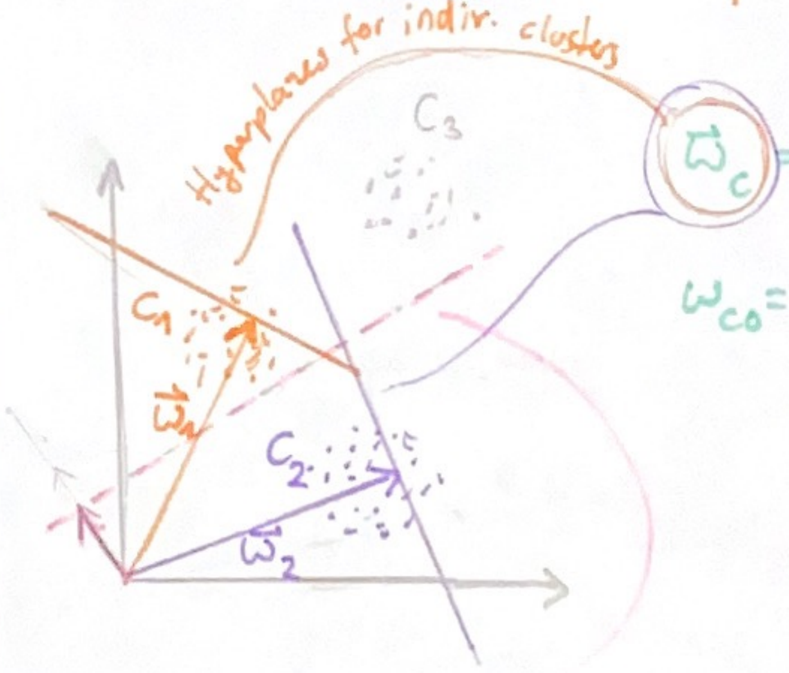
$$\vec{\omega} = \Sigma^{-1}(\vec{\mu}_1 - \vec{\mu}_0)$$

$$\omega_0 = -\frac{1}{2}\vec{\mu}_1^T \Sigma^{-1} \vec{\mu}_1 + \frac{1}{2}\vec{\mu}_0^T \Sigma^{-1} \vec{\mu}_0 + \log \frac{p(y=1)}{p(y=0)}$$

LDA for $C=2$: Posterior distr. is **sigmoid of lin. func. of $\vec{x}$**

LDA for $C>2$:

$$p(y=c|\vec{x}) = \frac{p(\vec{x}|y=c)\,p(y=c)}{\sum_{c'=1}^{C} p(\vec{x}|y=c')\,p(y=c')}$$

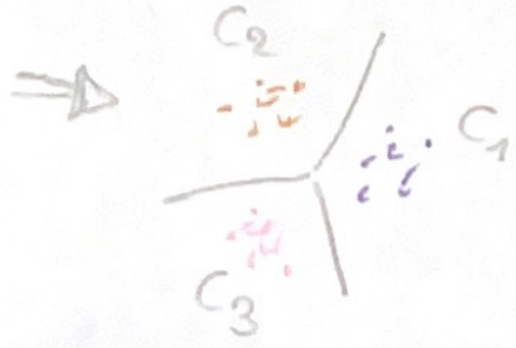


$$\vec{\omega}_c = \Sigma^{-1}\vec{\mu}_c$$

$$\omega_{c0} = -\frac{1}{2}\vec{\mu}_c^T\Sigma^{-1}\vec{\mu}_c + \log p(y=c)$$

$$= \frac{\exp(\vec{\omega}_c^T\vec{x} + \omega_{c0})}{\sum_{c'=1}^{C}\exp(\vec{\omega}_{c'}^T\vec{x} + \omega_{c'0})}$$

each class has its own function



is Softmax

$$\sigma(\vec{x})_i = \frac{\exp(x_i)}{\sum_{k=1}^{K}\exp(x_k)}$$

$$\sigma: \mathbb{R}^K \to \Delta^{K-1}$$

Probability Simplex (Space of non-negative vectors that sum up to one)

Decision boundary for 1 & 2

$$\vec{\omega}_1^T\vec{x} + \omega_{10} = \vec{\omega}_2^T\vec{x} + \omega_{20}$$

$$\Rightarrow \underbrace{(\vec{\omega}_1 - \vec{\omega}_2)^T\vec{x} + \omega_{10} - \omega_{20}}_{\text{new Hyperplane}} = 0$$

In practice $\Rightarrow$

$$p(y|\vec{x}) = \sigma(\underbrace{W\vec{x} + \vec{\omega}_0}_{\text{"Logits"}})$$

SOFTMAX; $W \in \mathbb{R}^{C\times D}$, $\vec{x} \in \mathbb{R}^{D\times 1}$, $\vec{\omega}_0 \in \mathbb{R}^{C\times 1}$

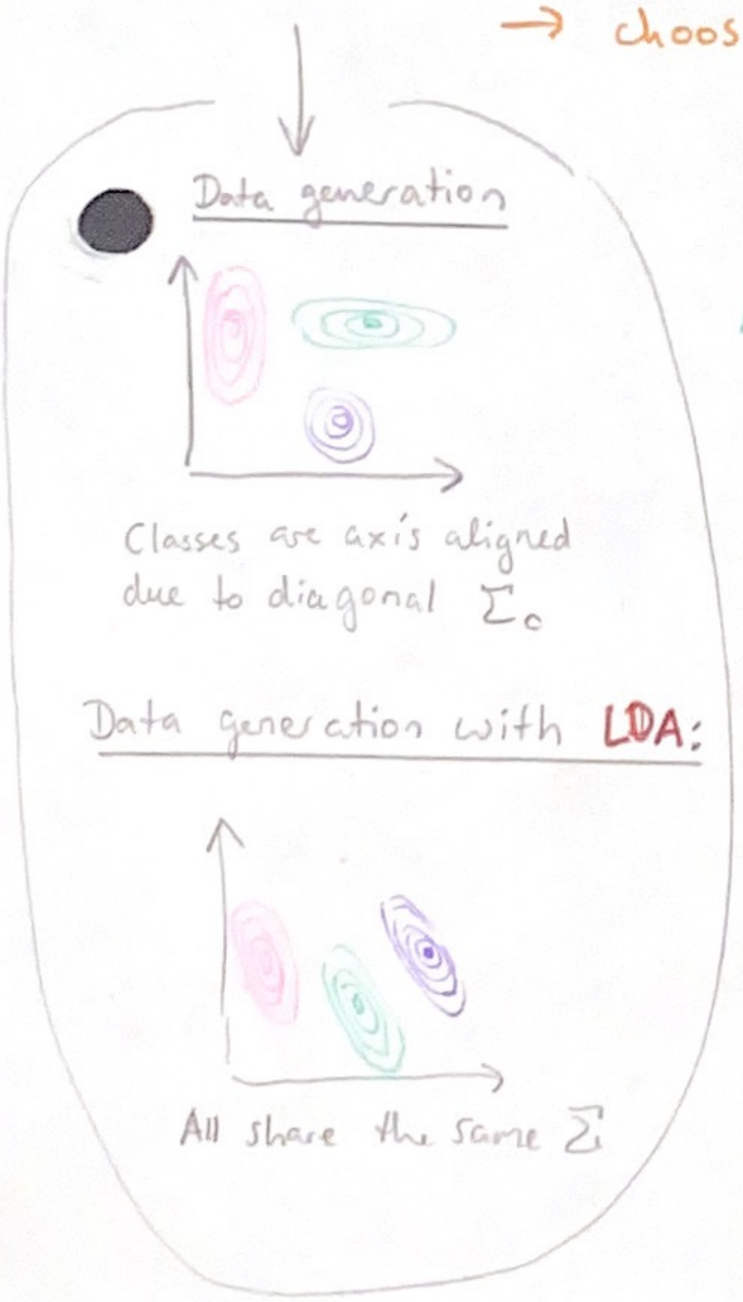
"Logits" $\rightarrow$ probabilities

- Naive Bayes (Variant II):

$\rightarrow$ chooses a different class conditional

$$\Rightarrow p(\vec{x}|y=c) = \mathcal{N}(\vec{x}|\vec{\mu}_c, \Sigma_c)$$

Covariance matrices are individual but diagonal



Assumption: d features of a sample $\vec{x}$ are conditionally independent

$$p(x_1, \dots, x_d|y=c) = \prod_{i=1}^{d} p(x_i|y=c)$$

Example for two classes $C = \{0,1\}$ (Slide 28)

$$\Rightarrow a = \log\frac{p(\vec{x}|y=1)\,p(y=1)}{p(\vec{x}|y=0)\,p(y=0)} = \dots = \vec{x}^T W_2\vec{x} + \vec{\omega}_1^T\vec{x} + \omega_0$$

$\downarrow$ quadratic term

$\rightarrow$ quadratic decision boundary

(LDA leads to lin. decision boundaries)

ADVANTAGE OF Naive Bayes:

$\rightarrow$ due to the cond. independence of features

$\rightarrow$ choose a (univariate) distribution for each feature

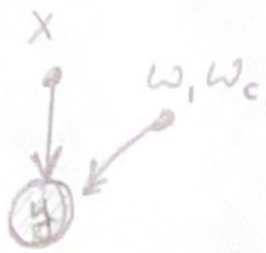
x_1 - Gaussian

x_2 - Categorical

...

- Probabilistic discriminative model:

→ Instead of $\vec{w}, w_0$ depending on μ_0, μ_1, Σ, they are chosen directly



Example: logistic regression (binary)

$$y \mid \vec{x} \sim \text{Bernoulli}(\sigma(\vec{w}^T\vec{x} + w_0)) \quad \text{with} \quad \sigma(a) = \frac{1}{1+e^{-a}}$$

$$\Rightarrow p(\vec{y} \mid \vec{w}, X) \overset{\text{i.i.d.}}{=} \prod_{i=1}^{N} p(y_i \mid \vec{x}_i, \vec{w}) \qquad \text{absorbs } w_0$$

$$= \prod_{i=1}^{N} \sigma(\vec{w}^T\vec{x}_i)^{y_i} \left(1 - \sigma(\vec{w}^T\vec{x}_i)\right)^{(1-y_i)}$$

Error func

$$\Rightarrow E = -\log p(\vec{y} \mid X, \vec{w})$$

$$= -\sum_{i=1}^{N} y_i \log \sigma(\vec{w}^T\vec{x}_i) + (1-y_i) \log\left(1 - \sigma(\vec{w}^T\vec{x}_i)\right)$$

:= Binary cross entropy loss

equivalent to MLE

$$\Rightarrow \vec{w}^* = \operatorname{argmin}_{\vec{w}} E(\vec{w})$$

→ No closed form (use optimization)

With weight regularization: $E(\vec{w}) = -\log p(\vec{y} \mid \vec{w}, X) + \lambda \|\vec{w}\|_q^q$

Example: Multiclass logistic regression

$$p(Y \mid \vec{w}, X) \overset{\text{i.i.d.}}{=} \prod_{i=1}^{N} p(\vec{y}_i \mid \vec{x}_i, \vec{w}) = \prod_{i=1}^{N} \prod_{c=1}^{C} p(y_i = c \mid \vec{x}_i, \vec{w}_c)^{y_{ic}}$$

Binary matrix / $Y \in \{0,1\}^{N \times C}$

→ One-hot-encoding

$$\Rightarrow E(\vec{w}) = -\log p(Y \mid \vec{w}, X) = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log p(y_i = c \mid \vec{x}_i, \vec{w}_c)$$

$$= -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \frac{\exp(\vec{w}_c^T\vec{x})}{\sum_{c'} \exp(\vec{w}_{c'}^T\vec{x})}$$

Softmax

:= Cross Entropy loss w/ one-hot-encoding

↳ $y_{ic} = \begin{cases} 1, & \text{if sample } i \text{ in class } c \\ 0, & \text{else} \end{cases}$

ML 06 : Optimization

↳ General task: Find solution $\vec{\theta}^*$ (variables / parameters) minimizing function

$$f: \vec{\theta}^* = \underset{\vec{\theta} \in X}{\operatorname{argmin}} f(\vec{\theta})$$

X: domain of $\vec{\theta}$ (constraints on parameters); $f(\vec{\theta})$: objective func.

- **Convexity:**

⇒ Set X is convex, if $\forall x, y \in X \; ; \; \lambda x + (1-\lambda) y \in X$ for $\lambda \in [0,1]$

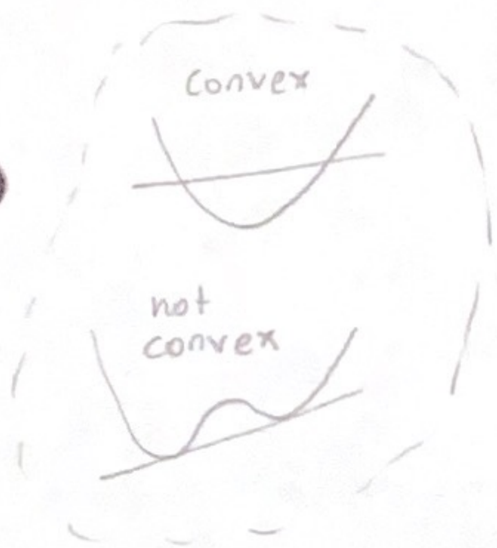
⇒ 

Not convex : 

Region above a convex function is a convex set.

→ $f(x)$ is a convex function on convex set X iff

$$\left[\forall x, y \in X : \lambda f(\vec{x}) + (1-\lambda) f(\vec{y}) \geq f(\lambda \vec{x} + (1-\lambda)\vec{x})\right] \quad \text{for } \lambda \in [0,1]$$



→ For a convex func: Each local minimum is a global minimum

↳ $f(\vec{y}) - f(\vec{x}) \geq \underbrace{\frac{f((1-t)\vec{x} + t\vec{x}) - f(\vec{x})}{t}}$

for $t \to 0$: $\frac{[f(\vec{x}+\epsilon) - f(\vec{x})]\,(y-x)}{\epsilon \cdot (y-x)}$

Iff this condition holds for all $x, y \in X$, then f is convex on X.

$$f(\vec{y}) - f(\vec{x}) \geq (\vec{y} - \vec{x})^T \nabla f(\vec{x})$$

⊖ **Verifying convexity:**

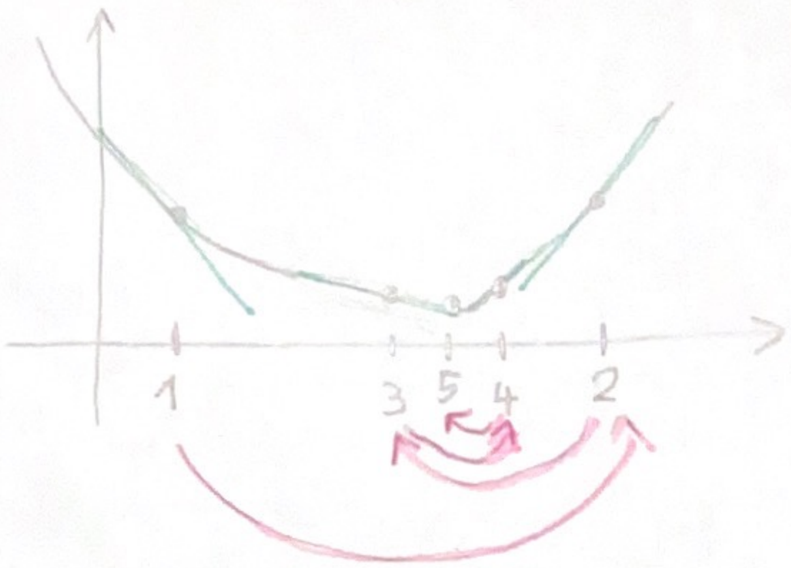
of func. {
① A twice diff. func. is convex iff its Hessian is positive semi-definite.
② Prove condition above.
③ Show that the func. is obtained from simple convex func. that preserve convexity
}

Convexity-preserving Operations:

- $h(\vec{x}) = f_1(\vec{x}) + f_2(\vec{x})$ (f_1: convex func)
- $h(\vec{x}) = \max(f_1(\vec{x}), f_2(\vec{x}))$
- $h(\vec{x}) = c \cdot f_1(\vec{x})$ if $c > 0$
- $h(\vec{x}) = c \cdot g(\vec{x})$ if $c < 0$ (g: concave func)
- $h(\vec{x}) = f_1(A\vec{x} + \vec{b})$
- $h(\vec{x}) = m(f_1(\vec{x}))$ (m: convex & non-decreasing)

of sets {
① Prove definition
② Apply intersection rule
→ If A and B are convex sets, $A \cap B$ is convex
}

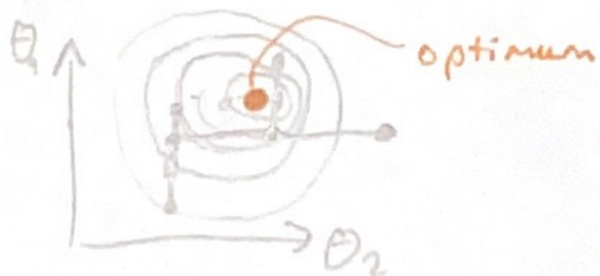
- Optimization in a One-dim. problem:



Goal: For diff. f search for θ with $\nabla f(\theta) = 0$

→ Do interval bisection

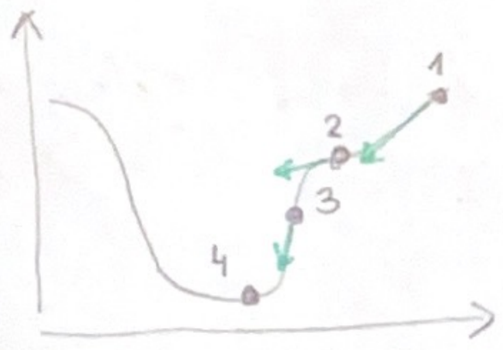
(In a multivariate problem do interval bisection in one dim., then in the next one, ...)



- Optimization in Multi-dim. ploroblems: Gradient descent (GD)

→ Idea: Gradient points in the direction of steepest ascent

Approximates the objective function locally



→ works on non-convex functions

GD with Line Search

given a starting point $\vec{\theta} \in Dom(f)$

repeat

1. $\Delta\vec{\theta} = -\nabla f(\vec{\theta})$ Get direction of (locally) steepest descent.
2. Line Search. $t^* = \underset{t>0}{\operatorname{argmin}} f(\vec{\theta} + t\Delta\vec{\theta})$ Solve one-dim. Line-Search problem
3. Update. $\vec{\theta} = \vec{\theta} + t^* \cdot \Delta\vec{\theta}$

until stopping criterion is met

Residual error $\rho = f(\vec{\theta}^{(k)}) - p^* \leq \varepsilon$ after at most $\frac{\log((f(\vec{\theta}^{(0)}) - p^*)\varepsilon)}{\log(1/c)}$ steps

→ ~~Fast~~ Parallel / distributed implementation

Often problems are in the form $f(\vec{\theta}) = \sum_i L_i(\vec{\theta}) + g(\vec{\theta})$ (iterates over data)

→ decompose ~~distribute~~ gradient by using the sum rule

→ Faster algorithm (actually the standard one)

→ Set a learning rate τ → too small: slow convergence; too high: alg. oscillates w/o converg.

→ use learning rate schedule

→ Momentum GD (incorporates history of previous gradients)

$$\vec{m}_t \leftarrow \tau \nabla f(\vec{\theta}_t) + \gamma \vec{m}_{t-1}$$

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \vec{m}_t$$

(accelerates the search in directions where many previous gradients point in the same direction)

- AdaGrad:
  - Different learning rate per parameter

- Adam: $\vec{m}_t = \beta_1 \vec{m}_{t-1} + (1-\beta_1)\nabla f(\vec{\theta}_t)$ (1. Moment)

$$\vec{v}_t = \beta_2 \vec{v}_{t-1} + (1-\beta_2)(\nabla f(\vec{\theta}_t))^2 \quad \text{(2. Moment)}$$

$$\Rightarrow \vec{\theta}_{t+1} = \vec{\theta}_t - \frac{\tau}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t \quad \text{with} \quad \hat{m}_t = \frac{\vec{m}_t}{1-\beta_1^t}, \; \hat{v}_t = \frac{\vec{v}_t}{1-\beta_2^t}$$

GD = First Order Method

- Newton-Method (second-Order-Method)

Given: - convex function f
- Nonneg. second derivative: $\nabla^2 f(\vec{\theta}) \geq 0$
  := Hessian is positive semi-definite

→ Taylor Expansion at point $\vec{\theta}_t$

$$f(\vec{\theta}_t + \vec{\delta}) = \underbrace{f(\vec{\theta}_t) + \vec{\delta}^T \nabla f(\vec{\theta}_t) + \frac{1}{2}\vec{\delta}^T \nabla^2 f(\vec{\theta}) \vec{\delta} + O(\vec{\delta}^3)}_{:= g(\vec{\delta})}$$

Minimize ⇒ $\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t - \underbrace{[\nabla^2 f(\vec{\theta}_t)]^{-1}}_{\tau := \text{learning rate}} \nabla f(\vec{\theta}_t)$

$\Rightarrow \nabla g(\vec{\delta}) \overset{!}{=} 0$

→ for low-dimensional methods
- ⊕ good convergence rate
- ⊕ Gradient requires $O(d)$ data (memory)
- ⊖ Hessian requires $\& O(d^2)$ data
- ⊖ Update step is $O(d^3)$ & tricky to parallelize (↖ inverse)

- Stochastic Optimization: (minimizes ~~costs with~~ computational costs)

→ Stochastic Gradient Descent

SGE computes noisy estimate of gradients on subset (minibatch)

$$\frac{1}{n}\left(\sum_{i=1}^{n} \nabla L_i(\vec{\theta})\right) = \underset{i \sim \{1,..,n\}}{E}\left[\nabla L_i(\vec{\theta})\right] \approx \frac{1}{|S|}\sum_{j \in S}\left(\nabla L_j(\vec{\theta})\right) \quad \text{with } S \subseteq \{1,..,n\}$$

write Loss as empirical risk minimization

Approximate estimation on subset S of the data

Scale Loss on subset such that it matches the loss on all data

$$\Rightarrow \sum_{i=1}^{n} \nabla L_i(\vec{\theta}) \approx \frac{n}{|S|}\sum_{j \in S} \nabla L_j(\vec{\theta})$$

Example: Perceptron (binary lin. classifier)

$$\delta(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w}^T\vec{x} + b > 0 \\ -1 & \text{else} \end{cases}$$

$$\Rightarrow \vec{w}^*, b^* = \underset{\vec{w},b}{\operatorname{argmin}} \sum_i L(y_i, \vec{w}^T\vec{x}_i + b)$$

$$\downarrow$$

Hinge-Loss $L(u, \overbrace{v}^{\text{prediction}}) = \max(0, \varepsilon - uv)$

ground truth

$$= \begin{cases} \varepsilon - uv, & \text{if } uv < \varepsilon \text{ incorrect prediction} \\ 0, & \text{else correct prediction} \end{cases}$$

$$\Rightarrow \nabla_w L(y_i, \vec{w}^T\vec{x} + b) = \begin{cases} -y_i x_i, & \text{if } uv < \varepsilon \\ 0, & \text{else} \end{cases}$$

$$\nabla_b L(y_i, \vec{w}^T\vec{x} + b) = \begin{cases} -y_i, & \text{if } uv < \varepsilon \\ 0, & \text{else} \end{cases}$$

SGD → with minibatch of size 1

initialize $\vec{w} = 0$ and $b = 0$

repeat

if $y_i \cdot (\vec{w}^T\vec{x}_i + b) < \varepsilon$ then (MISCLASSIFIED)

$\vec{w} \leftarrow \vec{w} + \tau \cdot n \cdot y_i \cdot \vec{x}_i$ and $b \leftarrow b + \tau \cdot n \cdot y_i$

(τ: learning rate, n: normalization)

end if

until all classified correctly

SGD: Expectation of residual error decreases with speed

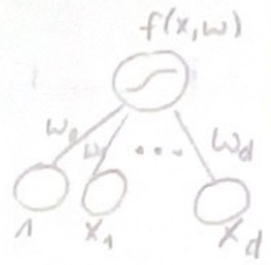
# of iterations $t \sim E[\rho]^{-1}$ (residual)

GD: speed has $t \sim \log \rho^{-1}$ } GD has faster convergence speed, but each iteration takes longer

- Distributed Learning:
  - → Data Parallelism (multiple workers operate on the same model and update parameters)
  - → Model parallelism (Use inherent model parallelisms ~~and~~ share on multiple workes (e.g. matrix ops)

And share data

ML 07 : Deep Learning I



- How to handle ~~basis functions~~ non-linearity ?

→ Regular logistic regression only for lin. seperable data $y|\vec{x} \sim \text{Bernoulli}(\sigma(\vec{w}^T\vec{x}))$

→ Use basis functions: $f(x,y) = \sigma\left(w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\vec{x})\right) = \sigma(\vec{w}^T\vec{\phi}(\vec{x}))$

↳ maps samples into a space where they're lin. separable

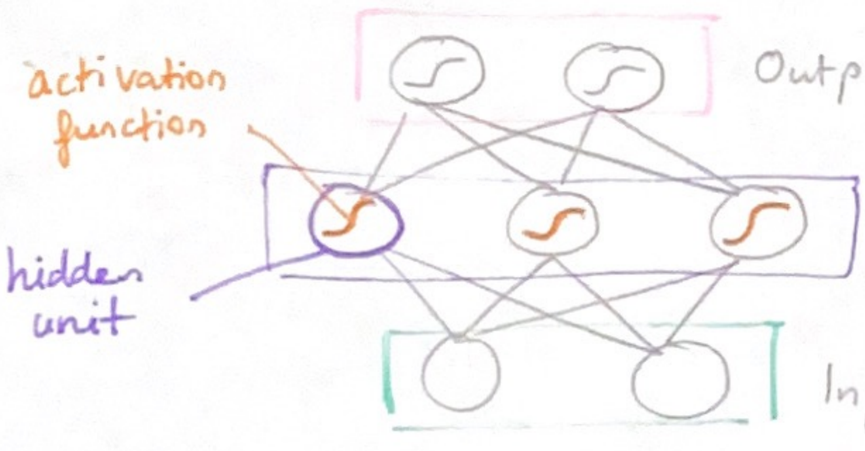
basis function

=> Then find the best weights with the binary cross entropy loss

$$W^* = \underset{W}{\operatorname{argmin}} \sum_{n=1}^{N} -\left(y_n \log f(x_n, W) + (1-y_n)\log(1-f(x_n,W))\right)$$

=> Is feed-forward NN with 1 hidden layer

- Multi-layered perceptron: (MLP) (feed-forward; fully-connected)

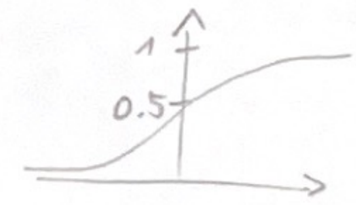


$$f(x,W) = \sigma_2(W_2\sigma_1(W_1\sigma_0(W_0 x)))$$

w/ 2 hidden layers

Activation func.:

Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$



ReLU: $\max(0,x)$

...

- Why use nonlinear activation funcs?

→ Lin. layers result in lin. transf.: $f(x,W) = W_L(W_{L-1}(\dots(W_0\vec{x})\dots)) = W'\vec{x}$

- Universal approximation theorem:

→ With enough hidden units an MLP with a lin. output layer and one hidden layer can approximate any continuous function (over a closed subset of $\mathbb{R}^D$)

- Why add more layers?

→ Use more layers with fewer hidden units instead of one wide layer

=> fewer parameters

→ In practice: Deep networks are easier to train

- What needs to be changed in a NN for different tasks?

→ the activation func in the final layer and the loss func

Prediction target	$p(\vec{y}\mid\vec{x})$	Final layer	Loss function
Binary	Bernoulli	Sigmoid	Binary cross entropy
Discrete	Categorical	Softmax	Cross entropy
Continuous	Gaussian	Identity	Squared error

The loss func should allow to do gradient based training

- How to compute the gradient for GD?

1) Work out manually (by hand)

2) Numerically: $\frac{\partial E_n}{\partial w_{ij}} = \frac{E_n(w_{ij}+\varepsilon) - E(w_{ij})}{\varepsilon} + \mathcal{O}(\varepsilon)$

→ expensive and maybe not stable

Computing this means evaluating the NN → $\mathcal{O}(|W|)$ for every parameter → $\mathcal{O}(|W|^2)$

3) Symbolic differentiation: Automates deriving if by hand.

↔ "Writing down" expression is expensive

4) Automatic differentiation / Backpropagation

→ Eval. $\nabla_W E(W)$ at current W

→ $\mathcal{O}(|W|)$

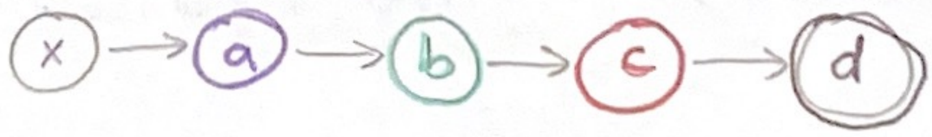
- Backprop:

Example:

1. Write func as composition of modules

$$f(x) = \frac{2}{\sin(\exp(-x))} = d(c(b(a(x))))$$

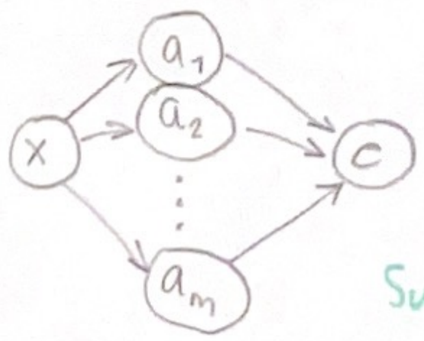
Comp. graph:



2. Do a forward pass and cache intermediate values for given input $\vec{x}$
3. Compute local derivatives for $\vec{x}$
4. Compute global derivative

Derivative: $\frac{\partial f}{\partial x} = \frac{\partial d}{\partial c}\cdot\frac{\partial c}{\partial b}\cdot\frac{\partial b}{\partial a}\cdot\frac{\partial a}{\partial x}$

- Multivariate chain rule:



$$\frac{\partial c}{\partial x} = \sum_{i=1}^{m} \frac{\partial c}{\partial a_i}\frac{\partial a_i}{\partial x}$$

Sum of the derivatives along all paths.

- Jacobian & Gradient:

• Jacobian: $\vec{a}: \mathbb{R}^n \to \mathbb{R}^m$

$$\frac{\partial \vec{a}}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial a_1}{\partial x_1} & \cdots & \frac{\partial a_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_m}{\partial x_1} & \cdots & \frac{\partial a_m}{\partial x_n} \end{bmatrix} = (\nabla_{\vec{x}} a_1)^T$$

($\frac{\partial \vec{a}}{\partial \vec{x}} \in \mathbb{R}^n$)

→ "the gradient is the transpose of the Jacobian"

$\nabla_{\vec{x}} a$ has the same shape as $\vec{x}$

• Gradient: $a: \mathbb{R}^m \to \mathbb{R}$

$$\nabla_{\vec{x}} a = \left(\frac{\partial a}{\partial \vec{x}}\right)^T = \begin{bmatrix} \frac{\partial a}{\partial x_1} & \cdots & \frac{\partial a}{\partial x_m} \end{bmatrix}^T$$

($\nabla_{\vec{x}} a \in \mathbb{R}^m$)

- Chain rule in matrix form:

$$\frac{\partial c}{\partial x_j} = \sum_{i=1}^{m} \frac{\partial c}{\partial a_i} \frac{\partial a_i}{\partial x_j} \quad \text{In matrix form} \Rightarrow \quad \frac{\partial c}{\partial \vec{x}} = \frac{\partial c}{\partial \vec{a}} \frac{\partial \vec{a}}{\partial \vec{x}} \quad \text{with} \quad \left[\frac{\partial \vec{a}}{\partial \vec{x}}\right]_{ij} = \frac{\partial a_i}{\partial x_j}$$

$$= \nabla_{\vec{x}} c \qquad = \left(\frac{\partial \vec{a}}{\partial \vec{x}}\right)^T \nabla_{\vec{a}} c$$

→ Matrix calculus:

$f: \bullet \to$	scalar	vector	matrix
scalar	sc	vec	matr
vector	vec	matr	3-way-tensor
matrix	matr	3-way-tensor	4-way-tensor

(output is a ... / Input is a ...)

$\left(\frac{\partial \vec{a}}{\partial W}\right)_{ijk} = \frac{\partial a_i}{\partial W_{jk}}$ e.g. shape of the derivative of the func

- Accumulating the gradient (Example):

Given feed-forward NN:

$$\begin{cases} \vec{a} = W\vec{x} + \vec{b} & \leftarrow \text{affine layer} \\ \vec{h} = \sigma(\vec{a}) \\ \hat{y} = V\vec{h} + c \\ E = (\hat{y} - y)^2 \end{cases}$$

→ In the affine layer calculate:

forward $(W, \vec{x}, \vec{b})$: compute $W\vec{x} + \vec{b}$

backward $\left(\frac{\partial E}{\partial \vec{a}}\right)$: Compute grads w.r.t. the output of the affine layer

would be 3-way-tensor

Derived on sl. 43

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial \vec{a}} \frac{\partial \vec{a}}{\partial W} = \vec{x} \frac{\partial E}{\partial \vec{a}}$$

We can comp. the gradient w/o comp. the Jacobian!

$$\frac{\partial E}{\partial \vec{x}} = \frac{\partial E}{\partial \vec{a}} \frac{\partial \vec{a}}{\partial \vec{x}} = \frac{\partial E}{\partial \vec{a}} W$$

$$\frac{\partial E}{\partial \vec{b}} = \frac{\partial E}{\partial \vec{a}} \frac{\partial \vec{a}}{\partial \vec{b}} = \frac{\partial E}{\partial \vec{a}}$$

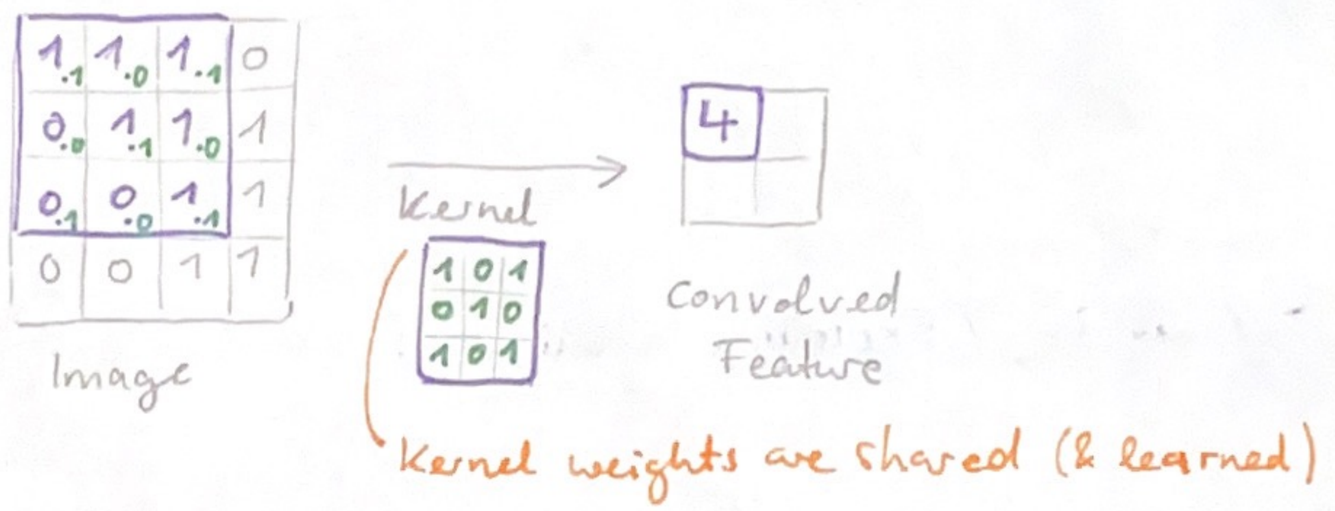
ML 08: Deep Learning II:

- Convolution:
  - continuous definition: $(x * k)(t) = \int_{\tau=-\infty}^{\infty} x(\tau)\, k(t-\tau)\, d\tau$

    ↑ "weighted avg of input x with weights / kernel / filter k at each point in time t"
  - discrete definition: $(x * k)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)\, k(t-\tau)$
  - 2-D "convolution" in CNNs: $\hat{x}(i,j) = \sum_{l=1}^{L} \sum_{m=1}^{M} x(i+l, j+m)\, k(l,m)$

    $\hat{x}(i,j)$: output of convolution; ↑ is actually cross-correlation; $x(i+l, j+m)$: eval kernel over corresponding input

Convolutions often act on multiple channels:
→ $L \times M \times C_{in} \times C_{out}$ parameters



* **Padding** schemes: → VALID: No padding. Reduces output size to

$$D_{l+1} = (D_l - K) + 1$$

(D_l: input size along a dim; K: kernel width)

→ SAME: Add padding to preserve the input size

$$P = \lfloor K/2 \rfloor$$

→ FULL: Add K-1 values on each side → increase output size

* **Strides**: (the distance between positions the kernel is applied)

$$\Rightarrow D_{l+1} = \left\lfloor \frac{D_l + 2P - K}{S} \right\rfloor + 1$$

→ Strides $S > 1$ downsample the signal

* **Pooling**: (calculate summary statistics in sliding window → downsampling)

→ Max / mean / Lp-norm pooling

- Weight initialization:

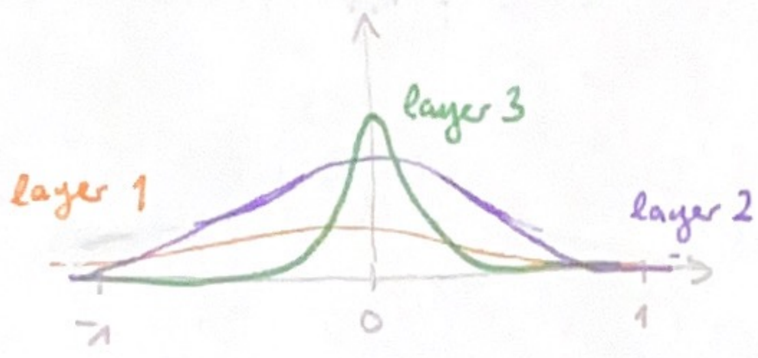
important aspects:

* **Weight symmetry**: Hidden units with the same weights & biases will always get the exact same gradients
→ never learn different features
→ break symmetry w/ small random values

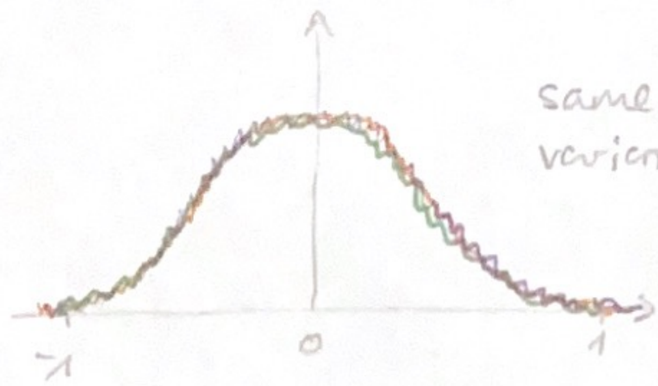
* **Weight scale**: Activation func may saturate for hidden unit w/ large fan-in
→ vanishing or exploding gradients w/ wrong weight scales
→ init. weights w/ good mean & variance

- Xavier-Glorot initialization : → idea: preserve mean & variance of an incoming i.i.d. signal in the forward & backward pass

⇒ By controlling the mean and variance of the weight distribution



Xavier-Glorot →



same mean & variance in all scores

Weight matrices : $\begin{cases} \mu = 0 \\ \text{Var}(W) = \dfrac{2}{\text{fan-in} + \text{fan-out}} \end{cases}$

→ Normal with

> In regression tasks:
> For initial weights of last layer, consider the target scale

- Vanishing / Exploding gradients :

Example : Say, a NN multiplies the input t times by W

⇒ $W^t = (V \, \text{diag}(D) \, V^{-1})^t = V \, \text{diag}(W)^t \, V^{-1}$

⇒ $\begin{cases} \text{if } D_{ii} < 1 \text{, then } D_{ii}^t \text{ is small} \longrightarrow \textbf{vanishing} \text{ gradients} \\ \text{if } D_{ii} > 1 \text{, then } D_{ii}^t \text{ explodes} \longrightarrow \textbf{exploding} \text{ gradients} \end{cases}$

* Saturated activation funcs also cause **exploding / vanishing** gradients.

→ Solutions: - Use batch-norm, other activations, etc.
- gradient clipping

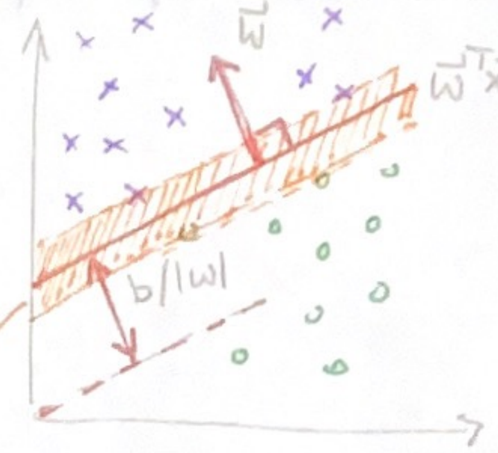
- Regularization : → prevents overfitting

* L_2-norm : enforces small weights
* L_1-norm : enforces sparsity
* Data augmentation, injecting noise, parameter sharing, dropout
  ↓
  randomly disable neurons during a forward / backward pass
  → sampling from different architectures with shared weights

Machine Learning 09: SVM and Kernels

- Linear classifier:

$$h(\vec{x}) = \text{sign}(\vec{w}^T\vec{x} + b) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$



a wide margin around the dividing hyperplane makes it more likely to correctly classify new samples
⇒ Find hyperplane with maximum margin

- Linear classifier w/ margin:

* Require $\vec{w}^T\vec{x} + (b-s) > 0$ for all points from class +1

and $\vec{w}^T\vec{x} + (b+s) < 0$ for all points from class −1

⇒ Size of margin $m = \underbrace{-\frac{b-s}{\|w\|}}_{\text{signed distance from the origin to the purple hyperplane}} - \underbrace{\left(-\frac{b+s}{\|w\|}\right)}_{\text{-"- green -"-}} = \underbrace{\frac{2s}{\|w\|}}_{\text{m only depends on this ratio.}}$

w.l.o.g. we set $s=1$

⇒ scale $\|w\|$ and b

$s=1$

⇒ Require the constraints:

$$\vec{w}^T\vec{x}_i + b \begin{cases} \geq +1 & \text{for } y_i = +1 \\ \leq -1 & \text{for } y_i = -1 \end{cases}$$

⇐⇒ $y_i(\vec{w}^T\vec{x}_i + b) \geq 1$ for all i

with the optimization problem $\underset{\vec{w},b}{\text{argmax}}\,(m)$

- SVM's optimization problem:

constrained convex optimization problem

$$\vec{w}^*, b^* = \underset{\vec{w},b}{\text{argmax}} \underbrace{\frac{2}{\sqrt{w^Tw}}}_{=\frac{2}{\|w\|}} \overset{\text{squaring is a monotonic func}}{\downarrow} = \underset{\vec{w},b}{\text{argmin}} \underbrace{\frac{w^Tw}{2}}_{\frac{1}{2}\|\vec{w}\|^2 := f_0(\vec{w},b)}$$

constraint: $f_i(\vec{w},b) = y_i(\vec{w}^T\vec{x}_i + b) - 1 \geq 0$ for $i = 1,\dots,N$

- Optimization w/ inequality constraints:

Given $f_0 : \mathbb{R}^d \to \mathbb{R}$ and $f_i : \mathbb{R}^d \to \mathbb{R}$

minimize $f_0(\vec{\theta})$ subject to $f_i(\theta) \leq 0 \quad \forall\, i$

* A point $\vec{\theta}$ is **feasible** if is satisfies the constraints
* Optimal value is the **minimum** p^* and the point where it's obtained it the **minimizer** $\vec{\theta}^*$

$$\Rightarrow p^* = f_0(\vec{\theta}^*)$$

Lagrange multiplier

Lagrangian: $$L(\vec{\theta}, \vec{\alpha}) = f_0(\vec{\theta}) + \sum_{i=1}^{M} \alpha_i f_i(\vec{\theta})$$

Lagrange dual function: $$g(\vec{\alpha}) = \min_{\vec{\theta} \in \mathbb{R}^d} L(\vec{\theta}, \vec{\alpha}) = \min_{\vec{\theta} \in \mathbb{R}^d} \left(f_0(\vec{\theta}) + \sum_{i=1}^{M} \alpha_i f_i(\vec{\theta}) \right)$$

is concave in α

($\vec{\theta}$ is unconstrained in this eq., since f_i is contained)

p^*
$g(\alpha)$

=> For every $\vec{\alpha}$, the unconstrained $g(\vec{\alpha})$ is a lower bound on the optimal value of the constrained problem:

$$\min_{\substack{\vec{\theta} \in \mathbb{R}^d \\ f_i(\vec{\theta}) \leq 0}} f_0(\vec{\theta}) = f_0(\vec{\theta}^*) \geq f_0(\vec{\theta}^*) + \underbrace{\sum_{i=1}^{M} \alpha_i f_i(\vec{\theta}^*)}_{\leq 0} = L(\vec{\theta}^*, \vec{\alpha}) \geq \min_{\vec{\theta} \in \mathbb{R}^d} L(\vec{\theta}, \vec{\alpha}) = g(\alpha)$$

≥ 0, by def. α must be feasible

≤ 0, since $\vec{\theta}^*$ is must be feasible

$$\Rightarrow \forall \vec{\alpha} \quad f_0(\vec{\theta}^*) \geq g(\vec{\alpha})$$

=> the maximum d^* of the Lagrangian dual problem is the best possible lower bound on p^*

* Weak duality: $g(\vec{\alpha}) \leq p^* \quad \forall \quad \vec{\alpha} \geq \vec{0}$

=> duality gap: $p^* - d^* \geq 0$

Strong duality: $g(\vec{\alpha}^*) = p^* = f_0(\vec{\theta}^*)$

($g(\vec{\alpha}^*)$: max of Lagrange dual problem; $f_0(\vec{\theta}^*)$: minimum of original problem)

Lagrange dual problem:

$$\max_{\vec{\alpha}} g(\vec{\alpha}) \quad \text{subject to} \quad \alpha_i \geq 0 \quad \forall\, i$$

- Constraint optimization problem of SVM:

1. Lagrangian: $L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}\vec{w}^T\vec{w} - \sum_{i=1}^{N} \alpha_i [y_i(\vec{w}^T\vec{x}_i + b) - 1]$

2. Minimize w.r.t. $\vec{w}$ and b

$$\nabla_{\vec{w}} L(\vec{w}, b, \vec{\alpha}) = \vec{w} - \sum_{i=1}^{N} \alpha_i y_i \vec{x}_i \overset{!}{=} 0 \quad \Rightarrow \quad \vec{w} = \sum_{i=1}^{N} \alpha_i y_i \vec{x}_i \quad (1)$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{N} \alpha_i y_i \overset{!}{=} 0$$

$$\Rightarrow g(\vec{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \vec{x}_i^T \vec{x}_j$$

$$\text{subject to} \quad \sum_{i=1}^{N} \alpha_i y_i = 0 \quad , \quad \alpha_i \geq 0 \quad \forall i$$

3. Solve $\underset{\vec{\alpha}}{\operatorname{argmax}}\, g(\vec{\alpha})$.

$\Rightarrow$ Find α^* via a QP-solver

$\Rightarrow$ Get $\vec{w}$ from (1)

$\Rightarrow$ Get b from:

$$f_0(\vec{\theta}^*) + \underbrace{\sum_i \alpha_i f_i(\vec{\theta}^*)}_{=0} \overset{\text{strong duality}}{=} f_0(\vec{\theta}^*)$$

Complementary slackness condition: holds for strong dualities: $\alpha_i^* f_i(\vec{\theta}^*) = 0$

for an $\vec{x}_i$ for which $\alpha_i \neq 0$, the corresponding constraint $f_i(\vec{w}, b)$ must be zero $\Rightarrow \vec{w}^T\vec{x}_i + b = y_i \quad \Rightarrow b = y_i - \vec{w}^T\vec{x}_i$

4. From $\alpha_i^* f_i(\vec{\theta}^*) = 0 \quad \Rightarrow \alpha_i [y_i(\vec{w}^T\vec{x}_i + b) - 1] = 0 \quad \forall i$

(a training sample only contributes to $\vec{w}$ if $\alpha_i \neq 0$, such that $y_i(\vec{w}^T\vec{x}_i + b) = 1$ (it lies on the margin!))

$\Rightarrow$ Then, sample x_i with $\alpha_i \neq 0$ is called **support vector**

5. $\Rightarrow$ Classification:

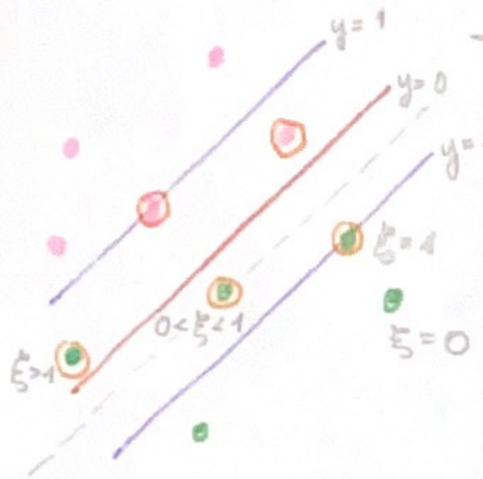
$$h(\vec{x}) = \operatorname{sign}\left(\sum_{i=1}^{N} \alpha_i y_i \vec{x}_i^T \vec{x} + b \right)$$

Since most α_i are zero, we only ~~need~~ store the $\vec{x}_i$ where $\alpha_i \neq 0$.

("solution is sparse")

- Soft margin suppor vector machines: "can handle noisy data / outliers"

Idea: Relax constraints, but punish relaxation



⇒ slack variable $\xi_i \geq 0$ for every sample $\vec{x}_i$

(the distance by which the constraints is violated in units of $\|\vec{w}\|$.

$$\Rightarrow \vec{w}^T\vec{x}_i + b \begin{cases} \geq +1 - \xi_i & \text{for } y_i = +1 \\ \leq -1 + \xi_i & \text{for } y_i = -1 \end{cases} \quad = y_i(\vec{w}^T\vec{x}_i + b) - 1 + \xi_i \geq 0$$

$y=-0.3$, since $\vec{w}^T\vec{x} + b = y$

⇒ Cost function becomes: (determines how heavily a violation is punished; $C \to \infty$ is hard-margin SVM)

$$f_0(\vec{w}, b, \vec{\xi}) = \frac{1}{2}\vec{w}^T\vec{w} + C\sum_{i=1}^{N}\xi_i$$

(Constrained)

∨

Optimization problem: minimize $f_0(\vec{w}, b, \vec{\xi}) = \frac{1}{2}\vec{w}^T\vec{w} + C\sum_{i=1}^{N}\xi_i$

$$\text{subject to} \cdot\ y_i(\vec{w}^T\vec{x}_i + b) - 1 + \xi_i \geq 0 \quad \forall\, i$$
$$\cdot\ \xi_i \geq 0$$

⇒ Lagrangian: ① $L(\vec{w}, b, \vec{\xi}, \vec{\alpha}, \vec{\mu}) = \underbrace{\frac{1}{2}\vec{w}^T\vec{w} + C\sum_{i=1}^{N}\xi_i}_{f_0(\vec{\theta})} - \underbrace{\sum_{i=1}^{N}\alpha_i\left(y_i(\vec{w}^T\vec{x}_i + b) - 1 + \xi_i\right)}_{f_i} - \underbrace{\sum_{i=1}^{N}\mu_i\xi_i}_{f_i}$

two sets of constraints

② Min. $L(\vec{w}, b, \vec{\xi}, \vec{\alpha}, \vec{\mu})$ w.r.t. $\vec{w}, b, \vec{\xi}$

$$\nabla_{\vec{w}} L = \vec{w} - \sum_{i=1}^{N}\alpha_i y_i \vec{x}_i \stackrel{!}{=} 0 \ , \quad \frac{\partial L}{\partial b} = \sum_{i=1}^{N}\alpha_i y_i \stackrel{!}{=} 0$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \mu_i \stackrel{!}{=} 0 \rightarrow \alpha_i = C - \mu_i$$

Dual feasibility: $\mu_i, \alpha_i \geq 0 \longrightarrow 0 \leq \alpha_i \leq C$

③ Dual problem: maximize $g(\vec{\alpha}) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \vec{x}_i^T\vec{x}_i$

$$\text{subject to } \sum_{i=1}^{N}\alpha_i y_i = 0 \ , \ 0 \leq \alpha_i \leq C \quad \forall\, i$$

→ Hinge loss formulation:

From the constrained optimization problem: ④ →

$$\xi_i = \begin{cases} 1 - y_i(\vec{w}^T\vec{x}_i + b) & \text{, if } y_i(\vec{w}^T\vec{x}_i + b) \leq 1 \\ 0 & \text{, else} \end{cases}$$
$$= \max\left(0,\ 1 - y_i(\vec{w}^T\vec{x}_i + b)\right)$$

"SVMs are L2-regularized perceptron with a Hinge-loss function"

⇒ Rewrite as unconstrained problem: $\min_{\vec{w}, b} \frac{1}{2}\vec{w}^T\vec{w} + C\sum_{i=1}^{N}\max\left(0, 1 - y_i(\vec{w}^T\vec{x}_i + b)\right)$

→ optimize with GD

- **Feature space:**

  * We use basis functions $\vec{\phi}(\cdot)$ to make models nonlinear and classify non-lin. seperable data

$$\phi: \mathbb{R}^D \to \mathbb{R}^M \quad , \vec{x}_i \to \phi(\vec{x}_i)$$

- **Kernels:**

Rewrite the Lagrange dual function as

$$g(\vec{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j k(\vec{x}_i, \vec{x}_j) \quad \text{with} \quad k(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i)^T \phi(\vec{x}_j).$$

referred to as the kernel trick

↑ Advantage of kernels: 1) Encode similarity between arbitrary data (numerical, objects, graphs, ...)

2) Easy to evaluate...

Mercer's theorem

A kernel is **valid**, if it gives rise to a symmetric, PSD kernel matrix K for some input data X.

$$\text{Kernel matrix} = \text{Gram matrix} = K = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_N) \\ \vdots & \ddots & \vdots \\ k(x_N, x_1) & \cdots & k(x_N, x_N) \end{pmatrix}$$

or, a kernel is valid, if it can be represented as a dot product in some feature space.

$\Rightarrow$ if the kernel is not valid, our optimization problem may become not convex

* Kernel preserving functions:

- $k(\vec{x}_1, \vec{x}_2) = k_1(\vec{x}_1, \vec{x}_2) + k_2(\vec{x}_1, \vec{x}_2)$
- $k(\vec{x}_1, \vec{x}_2) = c \cdot k_1(\vec{x}_1, \vec{x}_2)$ with $c > 0$
- $k(\vec{x}_1, \vec{x}_2) = k_1(\vec{x}_1, \vec{x}_2) \cdot k_2(\vec{x}_1, \vec{x}_2)$
- $k(\vec{x}_1, \vec{x}_2) = k_3(\phi(\vec{x}_1), \phi(\vec{x}_2))$ with k_3 on $X' \subseteq \mathbb{R}^M$ and $\phi: X \to X'$
- $k(\vec{x}_1, \vec{x}_2) = \vec{x}_1^T A \vec{x}_2$ with $A \in \mathbb{R}^{N \times N}$ symmetric and PSD

- Examples of kernels:

  - Polynomial: $k(\vec{a},\vec{b}) = (\vec{a}^T\vec{b})^p$ or $(\vec{a}^T\vec{b}+1)^p$
  - Gaussian: $k(\vec{a},\vec{b}) = \exp\left(-\frac{\|\vec{a}-\vec{b}\|^2}{2\sigma^2}\right)$ — hyperparameters
  - Sigmoid: $k(\vec{a},\vec{b}) = \tanh(\kappa\cdot\vec{a}^T\vec{b}-\delta)$ for $\kappa,\delta>0$

    ↳ is not PSD, but works well in practice

- Classifying new points w/ a kernelized SVM:

$$\underbrace{\alpha^* f_i(\vec{\theta}^*)}_{\text{complementary slackness condition}} \overset{!}{=} 0 \iff y_i\left(\sum_{\{j\,|\,\vec{x}_j\in S\}} \alpha_j y_j k(\vec{x}_i,\vec{x}_j)+b\right) = 1$$

$\{j\,|\,\vec{x}_j\in S\}$: set of support vectors with $\xi_i = 0$ (lie on the margin)

$$\Rightarrow \quad b = y_i - \left(\sum_{\{j\,|\,\vec{x}_j\in S\}} \alpha_j y_j k(\vec{x}_i,\vec{x}_j)\right)$$

$$\Rightarrow \quad h(\vec{x}) = \operatorname{sign}\left(\sum_{\{j\,|\,\vec{x}_j\in S\}} \alpha_j y_j k(\vec{x}_j,\vec{x}) + b\right)$$

- Approaches for multiclass data:

1) One-vs-rest: C SVM classifiers for C classes
   → Winner is the class where the dist from the hyperplane is max
2) One-vs-one: $\binom{C}{2}$ classifiers and evaluate all. The winner is the class with the majority vote (weighted by the distance from the margin)

   ↳ all possible pairings

ML 10 : Dimensionality reduction and matrix factorization

- Unsupervised learning: → Find hidden/latent structure is the data
  - ↳ Can be viewed as compression (compress data point from a highe dim. to a lower dim. latent space, e.g. its cluster label)
  - → Used for: * Visualization (understanding data)
    * preprocessing (Dim. reduction)
    * Unsupervised pretraining (Autoencoders)
  - * High-dim. data is problematic: 1) Curse of dim.
    2) Costly computation
    3) Highly correlated features cause problems w/ some algorithms

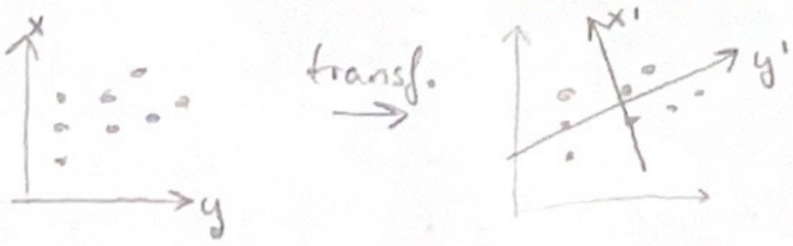
- Dim. Reduction via lin. transformations:
  - → want to capture intrinsic value of data in any coordinate system
    - → Use orthogonal basis transformation (and then potentially discard dims)
  - => Transform data vector into new coordinate system

$$(x')^T = x^T \cdot F \quad \text{(F: orthonormal transformation matrix } \in \mathbb{R}^{d \times k})$$

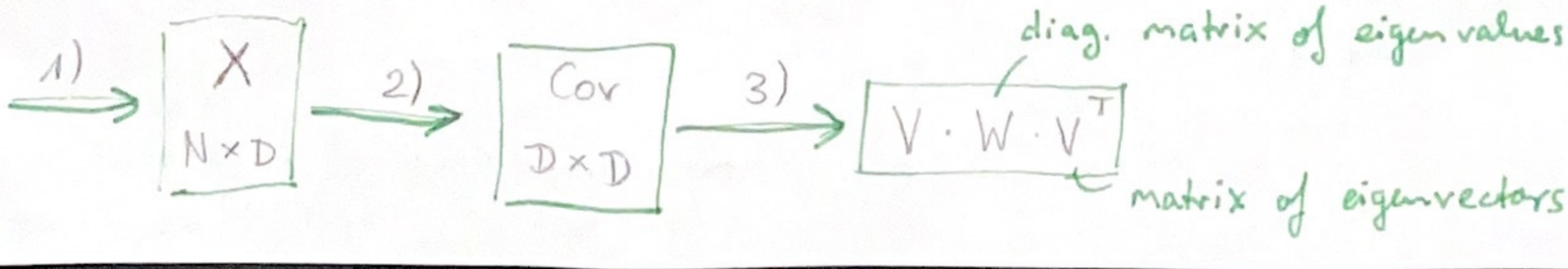
$$\overset{N\times k}{X'} = \overset{N\times d}{X} \cdot \overset{d\times k}{F} \quad \text{for all points } x_i$$

  - → After the transf. discard dimensions w/ low variance.

- Principal Component Analysis: (PCA)
  - Goal:
    - → Find a transf., such that features in the new coordinate system are lin. uncorrelated (→ the covariance between the new dim. is 0)
    - ↳ then discard dims. w/ lowest variance

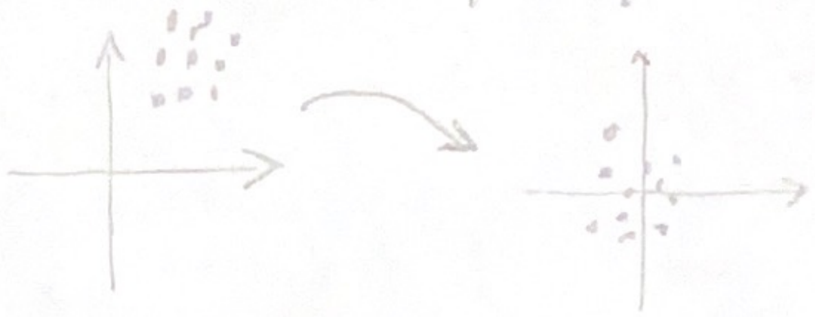


Approach:
1) Center data (zero mean)
2) Comp. covariance matrix
3) Use the eigenvector decomposition to transf. the coord. sys.



In detail: ① Center data: $\tilde{x}_i = x_i - \bar{x}$ with $\bar{x} = \begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_3 \\ \vdots \\ \bar{x}_d \end{pmatrix} = \frac{1}{N} X^T \cdot \vec{1}_N$

$\mathcal{O}(N \cdot d^2)$



$\mathcal{O}(d^3)$ ② Compute covariance matrix:

Variance in the jth dim. of X: $Var(X_j) = \frac{1}{N} \sum_{i=1}^{N} (x_{ij} - \bar{x}_j)^2 = \frac{1}{N} \cdot X_j^T X_j - \bar{x}_j^2$

Covariance between dim. j_1 and j_2: $Cov(X_{j_1}, X_{j_2}) = \frac{1}{N} \sum_{i=1}^{N} (x_{ij_1} - \bar{x}_{j_1})^T (x_{ij_2} - \bar{x}_{j_2}) = \frac{1}{N} X_{j_1}^T X_{j_2} - \bar{x}_{j_1} \bar{x}_{j_2}$

"Sigma" $\Rightarrow \Sigma_x = \begin{bmatrix} Var(X_1) & Cov(X_1, X_2) & \cdots & Cov(X_1, X_d) \\ Cov(X_2, X_1) & Var(X_2) & & \vdots \\ \vdots & & \ddots & \vdots \\ Cov(X_d, X_1) & \cdots & \cdots & Var(X_d) \end{bmatrix}$

Since the mean of $\tilde{X}$ is zero: $\Sigma_{\tilde{X}} = \frac{1}{N} X^T X - \underbrace{\bar{x}\bar{x}^T}_{=0}$ (sometimes omitted)

③ Transf. the coord. sys. such that the covariances between the new axes are zero:

$\mathcal{O}(N \cdot d \cdot k)$

$Cov' = \begin{bmatrix} Var(1)' & 0 & \cdots & 0 \\ 0 & Var(2)' & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Var(d)' \end{bmatrix}$

⇒ Eigendecomposition:

$\Sigma_{\tilde{X}} = \Gamma \cdot \Lambda \cdot \Gamma^T$

"gamma"; orthonormal matrix w/ eigenvectors (normalized) of $\Sigma_{\tilde{X}}$ as columns

"lambda"; diagonal matrix of eigenvalues

⇒ New coordinate sys. is defined by eigenvectors

Eigenvector – Eigenvalue: $A\gamma = \lambda\gamma$

Covariance after transf: $\Sigma_{\tilde{X}'} = \Gamma^T \Sigma_{\tilde{X}} \Gamma = \Gamma^T \Gamma \Lambda \Gamma^T \Gamma = \Lambda$ has the desired form

$\Gamma = \begin{bmatrix} | & | & | & | \\ | & | & | & | \end{bmatrix}$ (truncated), $\Lambda = \begin{bmatrix} 10 & & & \\ & 7 & & \\ & & 3 & \\ & & & 0.1 \end{bmatrix}$ ($k=3$)

⇒ Reduce dimensionality by keeping only the columns in Γ corresponding to the k largest eigenvalues (because the eigenvalue says how much the vector is scaled!)

k is often 90% of the energy

$\rightarrow \sum_{i=1}^{k} \lambda_i \geq 0.9 \sum_{i=1}^{d} \lambda_i$

⇒ $Y_{reduced} = \tilde{X} \cdot \Gamma_{reduced}$

Eigenvectors = Principal components

→ There are alternative views of PCA that treat it e.g. as an optimization problem

- Singular Value Decomposition: (SVD) → Finds best low-rank approximation to given matrix

* Data often lies in a low-dim manifold embedded in a higher-dim. space
  → measure this dim. through the rank of the data matrix

* The data is noisy
  → Find best low-rank approximation

Frobenius-Norm $\|A-B\|_F^2 = \sum_{i=1}^{N}\sum_{j=1}^{D}(a_{ij}-b_{ij})^2$

Euclidean dist. of flattened matrices

⇒ Constraint optimization problem $\min_B \|A-B\|_F^2$

constraint: $\text{rank}(B) = k$

→ Definition:

$\min\left(\mathcal{O}(nd^2), \mathcal{O}(n^2d)\right)$

$A^{n\times d} = U \cdot \Sigma \cdot V^T \quad \forall \; A \in \mathbb{R}^{n\times d}$

$V \in \mathbb{R}^{d\times r}$ right singular vectors
$U \in \mathbb{R}^{n\times r}$ left singular vectors
} column orthonormal

$\Sigma \in \mathbb{R}^{r\times r}$ singular values } diagonal w/ positive values sorted in decreasing order

$$= \sum_{i=1}^{r} \underbrace{\sigma_i \cdot \vec{u}_i^{\,n\times 1} \circ \vec{v}_i^{T\,1\times d}}_{n\times d \text{ rank } 1}$$

rank r

⇒ Represent the data as a linear combination of these latent concepts!

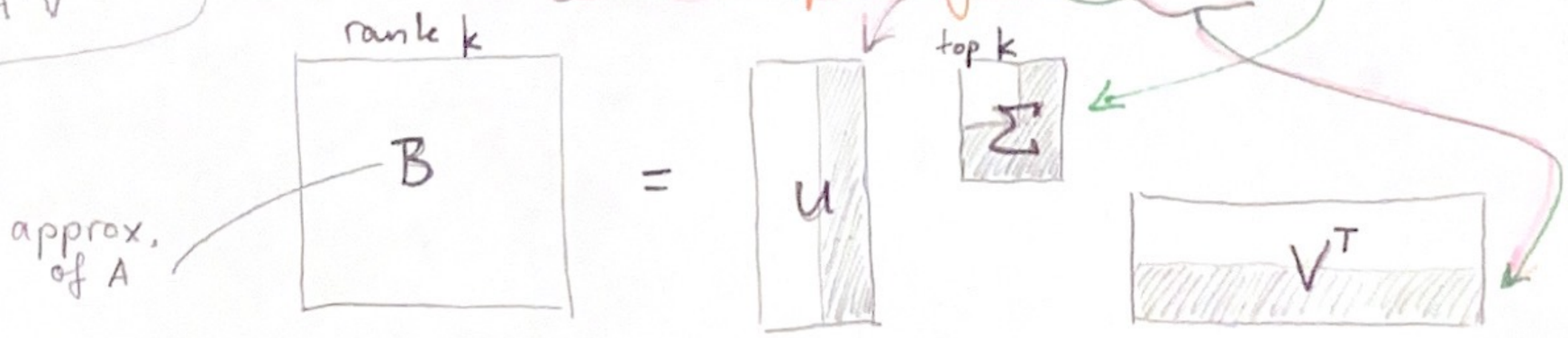
U and V are "concepts"
a.k.a. latent dimensions
a.k.a. latent factors

* The "strength" values σ in Σ weight the respective concept

* Interpretation: V^T: rows are the new coordinate axes (basis vectors)
  σ: the "spread"/variance in the direction of a coordinate axis
  $U\cdot\Sigma$: The coordinates of the points in the new coordinate system

The reduced data is
$P = U\Sigma = A\cdot V$

→ Set smallest singular values σ to zero and ignore the corresponding dimensions



- Comparison:

→ Given data X (centered)

SVD: $X = U\Sigma V^T$ and new (projected) data is $XV = U\Sigma$ (or truncated V)

PCA: Computes eigendecomposition of cov. matrix

→ $\Sigma = X^T X$

→ Eigendecomposition: $X^T X = \Gamma \Lambda \Gamma^T$

Projected data is $X \cdot \Gamma$ (or truncated Γ)

→ PCA and SVD are equivalent:

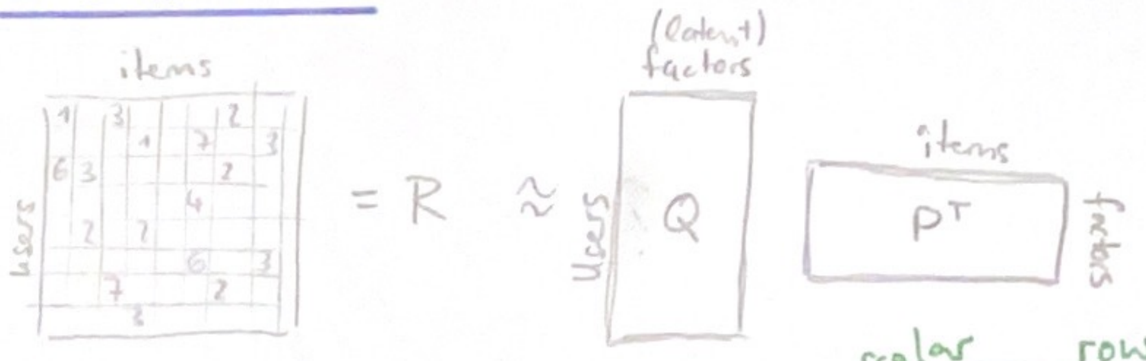
$$X^T X = (U\Sigma V^T)^T U\Sigma V^T = V\Sigma^T U^T U\Sigma V^T = V\Sigma^2 V^T$$

$$\hat{=} \Gamma \Lambda$$

$$\Rightarrow \quad V = \Gamma$$
$$\Sigma^2 = \Lambda$$

ML 11: Dim. Reduction & Matrix Factorization II

- Latent Factor Models: Given:



Netflix recommendation challenge

Goal: Find $Q \in \mathbb{R}^{n \times k}$ and $P \in \mathbb{R}^{d \times k}$: $\min_{P,Q} \sum_{u,i \in S} (r_{ui} - \vec{q}_u \vec{p}_i^T)^2$

(r_{ui}: scalar; $\vec{q}_u$: row u of Q; $\vec{p}_i^T$: row i of P)

Differences to PCA & SVD:
- → Only sum over existing entries: $S = \{(u,i) \mid r_{ui} \neq \text{missing}\}$
- → Don't require colums of P and Q to be orthogonal & unit length

- Alternating optimization: (a.k.a. blocke coordinate optimization)

Pseudo Code:
1. Init $P^{(0)}, Q^{(0)}, t=0$
2. $P^{(t+1)} = \operatorname{argmin}_P f(P, Q^{(t)})$
3. $Q^{(t+1)} = \operatorname{argmin}_Q f(P^{(t)}, Q)$
4. $t = t+1$
5. go to 2 until convergence

(2. and 3. alternatingly keep one variable fixed and solve for the other)

- Initialization:
  - With mean / random / ... values
  - Use result from SVD where empty entries are replaced with 0

$$Q = U\Sigma \text{ and } P^T = V^T$$

- Solve: $P^{(t+1)} = \operatorname{argmin}_P f(P, Q^{(t)}) = \operatorname{argmin}_P \sum_{(u,i) \in S} (r_{ui} - \vec{q}_u \vec{p}_i^T)$

$$= \sum_{i=1\dots d} \underbrace{\operatorname{argmin}_{p_i} \sum_{u \in S_{*,i}} (r_{ui} - \vec{q}_u \vec{p}_i^T)^2}$$

Since Q is fixed, we can solve for each $\vec{p}_i$ independently

ordinary least squares regression problem

$S_{*,i} = \{u \mid (u,i) \in S\}$

equivalently for $q^{(t+1)}$

$$\min_w \sum_{j=1}^n (y_j - \vec{w}^T \vec{x}_j)^2 \longrightarrow w^* = \left(\frac{1}{n}\sum_{j=1}^n \vec{x}_j \vec{x}_j^T\right)^{-1} \cdot \frac{1}{n}\sum_{j=1}^n \vec{x}_j y_j$$

$$\Rightarrow \vec{p}_i^T = \left(\frac{1}{|S_{*,i}|} \sum_{u \in S_{*,i}} \vec{q}_u^T \vec{q}_u\right)^{-1} \cdot \frac{1}{|S_{*,i}|} \sum_{u \in S_{*,i}} \vec{q}_u^T r_{ui}$$

equivalently for $\vec{q}_u$

* Advantages & Drawbacks:

+ Solution to difficult optimization problem
+ simple to implement & efficient on sparse data
− solution is only approximation
− solution is highly dependant on initial solution

- With SGD : objective : $\mathcal{L} = \sum_{(u,i) \in S} (r_{ui} - \vec{q}_u \vec{p}_i^T)$

1) Pick random user u and item i with rating r_{ui}

2) Compute gradients w.r.t. parameters = $\frac{\partial \mathcal{L}}{\partial \vec{q}_u}$ and $\frac{\partial \mathcal{L}}{\partial \vec{p}_i}$

3) Update parameters : $\vec{q}_u \leftarrow \vec{q}_u - \eta \frac{\partial \mathcal{L}}{\partial \vec{q}_u}$, $\vec{p}_i \leftarrow \vec{p}_i - \eta \frac{\partial \mathcal{L}}{\partial \vec{p}_i}$

(η = learning rate)

- Rating prediction :

Estimate missing rating for user u and item i : $\hat{r}_{ui} = \vec{q}_u \cdot \vec{p}_i^T$

($\vec{q}_u$ = row u of Q, $\vec{p}_i$ = row i of P)

- User & item biases :

$$\min_{P,Q,b_u,b_i,b} \sum_{(u,i) \in S} (r_{ui} - (\vec{q}_u \vec{p}_i^T + b_u + b_i + b))^2$$

(b_u = user bias (e.g. this user gives overly neg. reviews), b_i = item bias, b = overall bias)

- Challenge : Overfitting :

# of regression parameters = number of latent factors = k

# of data points = cardinality of $S_{*,i}$ / $S_{u,*}$

} overfit if large k and less data

⇒ Use Regularization

$$\min_{P,Q} \sum_{(u,i) \in S} (r_{ui} - \vec{q}_u \vec{p}_i^T)^2 + \left[\lambda_1 \sum_u \|\vec{q}_u\|^2 + \lambda_2 \sum_i \|\vec{p}_i\|^2 \right]$$

optim. { → Closed form solution from ridge regression
→ SGD

L2 vs L1-Regularization :

- L2 : tries to "shrink" all values equally.
  → large values are highly penalized
  → it is unlikely that values are exactly 0
- L1 : enforces sparsity

* We want sparsity → Less data to store
→ More intuitive, that sparse input data comes from sparse signal

- Further Matrix Factorization stuff :

→ Used for dim. reduction + Visualization (compute and plot 2-D or 3D (data analysis) latent factors)

* Often the data contains only non-negative values
→ SVD may lead to factor that have neg. values
→ difficult to interpret + this may predict neg. values
⇒ Use Non-neg. matrix factorization

- Neighbor graph methods:

  - Disadvantage of matrix factorization methods (PCA, SVD): preserve global structure, but may loose local structure

  - Neighbor graph methods try to preserve the neighborhood of each point (local structure)

Procedure:
1. Construct neighborhood graph of high-dimensional data points
2. (Randomly) initialize points in a low-dim. space
3. Optimize the coordinates of points in low-dim. space s.t. similarities align.
   (between the random points in low-dim data and the orig. points in high dim. data)

"adjacency matrix of the graph"
"pairwise similarity matrix"
"pairwise distance matrix"

- t-SNE: t-distributed stochastic neighbor embedding

* High-dim similarities for input $\vec{x}_i$: $p_{j|i} = \frac{\exp\left(-\frac{\|\vec{x}_i - \vec{x}_j\|}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|\vec{x}_i - \vec{x}_k\|}{2\sigma_i^2}\right)}$

$p_{j|i}$: the probability of choosing point j when choosing according to the similarity to point i

$\|\vec{x}_i - \vec{x}_j\|$: dist. between $\vec{x}_i$ and $\vec{x}_j$

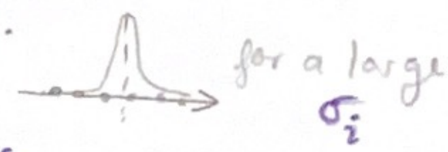
convert to distribution

Similarity between $\vec{x}_i$ and $\vec{x}_j$

example for 1-D data, (if the dist is smaller, the similarity will be larger)

→ we could use other ways to measure the similarity

→ by setting σ_i we get effective neighbors, e.g.
↳ set σ_i for each data point s.t. it has a fixed # of neighbors (fixed perplexity)



for a large σ_i

only (very close)

$p_{ii} = 0$, $p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$

* Low-dim similarities for parameters $\vec{y}_i$: $q_{ij} = \frac{(1 + \|\vec{y}_i - \vec{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\vec{y}_i - \vec{y}_l\|)^{-1}}$, $q_{ii} = 0$

t-distribution

=> Make probabilities p_{ij} match q_{ij} by changing $\vec{y}_i$:

$$\min_{\vec{y}_i} KL(P\|Q) = \sum_i \sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

→ KL-Divergence:
- $KL(P\|Q) \geq 0 \quad \forall P, Q$ and $KL(P\|Q) = 0$ iff $P = Q$
- $KL(P\|Q) \neq KL(Q\|P)$

→ t-SNE is SotA for visualizing high-dim. data
↳ be careful when interpreting cluster sizes and distances
→ sensitive to hyperparams
→ Not good for more than 3 dimensions (→ not used for dim. reduction)

- Autoencoders (Non-lin. dim. reduction):

* PCA / SVD can only capture lin. structure : $Y = \tilde{X} \cdot \Gamma$

→ But data may lie in a non-lin. low-dim. manifold

→ Autoencoder finds low-dim. representation by learning to reconstruct the input

$$f(\vec{x}, W) := \hat{x} \overset{!}{=} \vec{x}$$

min. reconstruction error between $\hat{x}$ and $\vec{x}$ $\left\{ \min_W \frac{1}{N} \sum_{i=1}^{N} \| f(\vec{x}_i, W) - \vec{x}_i \|^2 \right.$

→ Find latent representation $\vec{z} \in \mathbb{R}^L$ which is a compact representation of $\vec{x} \in \mathbb{R}^D$

encoder: $f_{enc}(\vec{x}) = \vec{z}$

decoder: $f_{dec}(\vec{z}) \approx \vec{x}$

* Linear Autoencoders are PCA

$$\left.\begin{aligned} f_{enc}(\vec{x}, W_1) &= \vec{x} W_1 \\ f_{dec}(\vec{x}, W_2) &= \vec{x} W_2 \end{aligned}\right\} \; f_{dec}(f_{enc}(\vec{x})) = \vec{x} W_1 W_2 = \vec{x} W$$

→ $\min_W \frac{1}{N} \sum_{i=1}^{N} \| \vec{x}_i W - \vec{x}_i \|^2 = W^*$ s.t. the rank of W^* is L

↓

optimal solution trough PCA : $W^* = \Gamma$

Summary:

Dim. Red. ~~Matrix fact. Methods~~

- require less storage
- more efficient data processing
- PCA & SVD give optimal low-rank approximation

Matrix fact. methods:

- allows to handle missing values

Autoencoders:

- Works on a lot of data (images, ...)

ML 12: Clustering

→ group objects $\vec{x}_i$ into K clusters based on their similarity

* is unsupervised learning

* Used for gene expression analysis, data compression, visualization, ...

→ Given data $X = \{x_1, \dots, x_N\}$

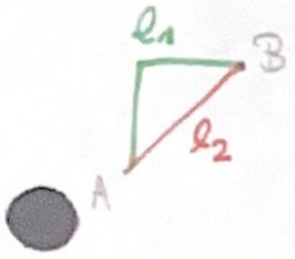
for each object x_i find the assignment to a cluster $z_i \in \{1, \dots, K\}$ s.t.:

- objects within clusters are similar to each other
- objects between clusters are dissimilar to each other

- K-means algorithm:

→ distance-based: → maximize similarity or minimize dissimilarity (distance)

* Manhattan-distance: $d(\vec{x}_i, \vec{x}_j) = \sum_d |x_{id} - x_{jd}| = \|\vec{x}_i - \vec{x}_j\|_1$



* Euclidean distance: $d(\vec{x}_i, \vec{x}_j) = \sqrt{\sum_d (x_{id} - x_{jd})^2} = \|\vec{x}_i - \vec{x}_j\|_2$

* Mahalanobis distance: $d(\vec{x}_i, \vec{x}_j) = \sqrt{(\vec{x}_i - \vec{x}_j)^T \Sigma^{-1} (\vec{x}_i - \vec{x}_j)}$ if $\Sigma^{-1} = I$ this is euclidean dist.

Model:

- Each cluster is defined by its centroid $\vec{\mu}_k \in \mathbb{R}^D$
- Cluster indicator $\underbrace{\vec{z}_i \in \{0,1\}^K}_{\text{one-hot encoding}}$ with $z_{ik} = 1 \Leftrightarrow \vec{x}_i$ belongs to cluster i

- K-means-objective (distortion measure)

$$J(X, Z, \mu) = \sum_{i=1}^{N} \sum_{k=1}^{K} z_{ik} \|\vec{x}_i - \vec{\mu}_k\|_2^2 \quad \hat{=} \text{ dist. of every point to it's cluster's centroid}$$

→ Goal: find best centroids and cluster assignments

$$Z^*, \mu^* = \underset{Z, \mu}{\operatorname{argmin}}\, J(X, Z, \mu)$$

Solution: - Solving the joint optim. problem is difficult → solve two sub problems

→ **Lloyd's algorithm:**

1) Initialize the centroids $\mu = \{\mu_1, \dots, \mu_K\}$

2) Update cluster indicators (solve $\min_Z J(X, Z, \mu)$) } assign every point to its nearest cluster centroid

$$z_{ik} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\vec{x}_i - \vec{\mu}_j\|_2^2 \\ 0 & \text{else} \end{cases}$$

3) Update centroids (solve $\min_\mu J(X, Z, \mu)$) } set centroids to mean of all points in the cluster

$$\vec{\mu}_k = \frac{1}{N_k} \sum_{i=1}^{N} z_{ik} \vec{x}_i \quad \text{where} \quad N_k = \sum_{i=1}^{N} z_{ik}$$

4) Return to 2) until convergence

→ K-means++ algorithm: (Used to initialise good centroids)

1) Choose the first centroid $\vec{\mu}_1$ uniformly at random among the data points

2) For each point $\vec{x}_i$ compute $D_i^2 = \| \vec{x}_i - \vec{\mu}_1 \|_2^2$

3) Sample the next centroid $\vec{\mu}_k$ from $\{\vec{x}_i\}$ w/ probability proportional to D

4) Recompute $D_i^2 = \min \{ \underbrace{\| x_i - \mu_1 \|_2^2, \dots, \| x_i - \mu_k \|_2^2}_{\text{dist for point } x_i \text{ to every centroid}} \}$

5) Return to 3) until all K initial centroids are chosen

→ Advantages & Disadvantages:

\+ cheap to compute w/ Lloyd's alg.: $O(nK)$

model issues:
- Quality depends on distance func a lot
- Can't detect clusters w/ overlapping hulls
- Sensitivity to outliers

algorithm issues:
- Extreme sensitivity to initialization

- Gaussian Mixture Model: (GMM)

general prob. view of clustering:

Probabilistic Model: $p(X|\theta) = \sum_z p(X, z|\theta) = \sum_z p(X|z,\theta)\, p(z|\theta)$

since z are never observed → "latent variables"

and optimize w.r.t. θ

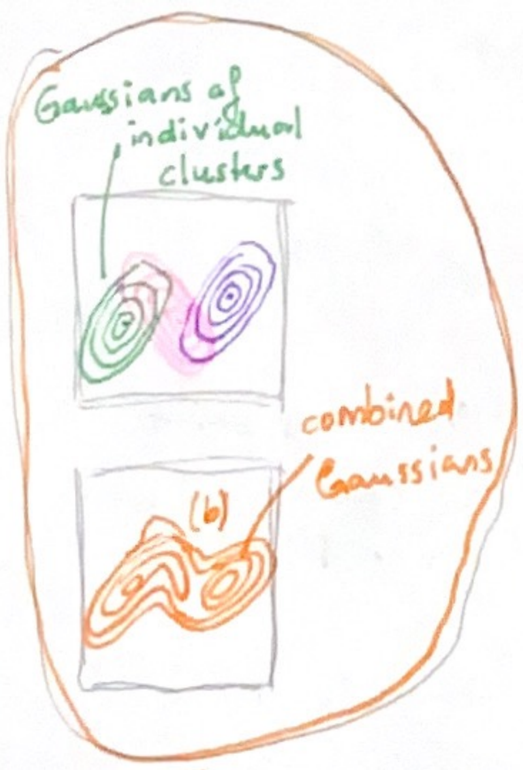
$$\theta^* = \arg\max_\theta p(X|\theta)$$

Gaussian mixture model: $p(\vec{x}, \vec{z}|\theta) = p(\vec{x}|\vec{z},\theta) \cdot p(\vec{z}|\theta)$

with $p(\vec{z}|\theta) = \text{Cat}(\pi)$ Cluster prior is Categorical

$p(\vec{x}|z_k=1,\theta) = \mathcal{N}(\vec{x}|\vec{\mu}_k, \Sigma_k)$ Each cluster has its own multivariate normal distribution

$\Rightarrow \theta = \{\pi, \mu, \Sigma\}$



→ Generative process of GMMs: 1) Draw one-hot cluster indicator $\vec{z} \sim \text{Cat}(\pi)$

2) Draw sample $\vec{x} \sim \mathcal{N}(\mu_k, \Sigma_k)$ if $z_k = 1$

* GMM Likelihood: $p(\vec{x}|\vec{\pi}, \mu, \Sigma) \overset{\text{marginalization over } z}{=} \sum_{k=1}^{K} p(z_k=1|\vec{\pi})\, p(\vec{x}|z_k=1, \vec{\mu}_k, \Sigma_k)$

$$= \sum_{k=1}^{K} \pi_k \mathcal{N}(\vec{x}|\vec{\mu}_k, \Sigma_k)$$

"The likelihood is a mixture of k different Gaussians"

(b)

i.i.d. data $\Rightarrow p(X|\vec{\pi}, \mu, \Sigma) = \sum_{i=1}^{N} p(\vec{x}_i|\vec{\pi}, \mu, \Sigma)$

→ Inference in GMM:

Given parameters $\{\vec{\pi}, \mu, \Sigma\}$, we want to assign points to clusters

⇒ compute the posterior distr. of cluster indicators $p(z|X, \pi, \mu, \Sigma)$

Bayes' rule

$$\Rightarrow p(z_{ik}=1 \mid \vec{x}_i, \underbrace{\pi, \mu, \Sigma}_{\theta}) = \frac{p(z_{ik}=1 \mid \vec{\pi})\, p(\vec{x}_i \mid z_{ik}=1, \vec{\mu}_k, \Sigma_k)}{p(\vec{x}_i \mid \vec{\pi}, \mu, \Sigma)}$$

$$= \frac{\pi_k \mathcal{N}(\vec{x}_i \mid \vec{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}_i \mid \vec{\mu}_j, \Sigma_j)}$$

$$:= \gamma(z_{ik}) = \text{the responsibility of component } k \text{ for observation } i$$

"How likely does the datapoint belong to the cluster?"

$$p(z \mid x, \theta) = \frac{p(x \mid z, \theta)\, p(z \mid \theta)}{p(x \mid \theta)}$$

→ Learning in GMM:

We want to determine optimal parameters (find values that max. the log-likelihood)

$$\pi^*, \mu^*, \Sigma^* = \underset{\pi, \mu, \Sigma}{\operatorname{argmax}} \underbrace{\log(p(X \mid \pi, \mu, \Sigma))}_{\sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\vec{x}_i \mid \vec{\mu}_k, \Sigma_k)\right)}$$

The log is over the sum makes it difficult to solve this!

Instead of optimizing $\log p(X \mid \theta)$ (which we can't) we optimize a proxy objective:

$$\mathcal{L}(q_t, \theta) := \underbrace{\mathop{E}_{z \sim q_t(z)} [\log p(X, z \mid \pi, \mu, \Sigma)]}_{\text{is a lower bound of the log-likelihood (proof in lecture)}} = \underbrace{\sum_{N} \sum_{K} \gamma_t(z_{ik}) \log p(x_i, z_{ik} \mid \theta)}_{\text{sum over log which we can solve}}$$

Appl. of the EM-Alg. in general:

Given a problem: $\max_\theta \log p(x \mid \theta) = \log \sum_z p(X, z \mid \theta)$

E-step: Eval the posterior $\gamma_t = p(z \mid x, \theta^{(t)})$

M-step: max. the expected joint log-likelihood $\log p(X, z \mid \theta^{(t+1)})$ w.r.t. θ under the current beliefs $\gamma_t(z)$

→ Expectation Maximization (EM) algorithm: (is alternating optim.)

1) Initialize model parameters: $\{\pi^{(0)}, \mu_1^{(0)}, \dots, \mu_K^{(0)}, \Sigma_1^{(0)}, \dots, \Sigma_K^{(0)}\}$

2) E Step — Evaluate the responsibilities

$$\gamma_t(z_{ik}) = p(z_{ik}=1 \mid \vec{x}_i, \pi_k^{(t)}, \mu_k^{(t)}, \Sigma_k^{(t)})$$ } same as Inference

3) M Step (max. the proxy objective w.r.t. the parameters)

$$\mu_k^{(t+1)} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_t(z_{ik})\, x_i$$

$$\Sigma_k^{(t+1)} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_t(z_{ik}) (\vec{x}_i - \vec{\mu}_k^{(t+1)})(\vec{x}_i - \vec{\mu}_k^{(t+1)})^T$$

$$\pi_k^{(t+1)} = \frac{N_k}{N}$$

} with $N_k = \sum_{i=1}^{N} \gamma_t(z_{ik})$

4) Return to 2) until the proxy objective converges

- Summary of EM:

* It is not possible to optimize the log-likelihood due to latent variables Z

$$\log(X|\theta) = \log\left(\sum_z p(X|z,\theta)\, p(z|\theta)\right)$$

* It is "easy" to optimize the joint probability

$$\log(X,z|\theta) = \log p(z|\theta) + \log p(X|z,\theta)$$

=> Main idea of EM:

1) Use our current beliefs $p(z|X,\theta^{(t)})$ to "pretend" that we know Z

2) E step: update the responsibilities $\gamma_t(z) = p(z|X,\theta^{(t)})$

3) M step: update parameters $\theta^{(t+1)} = \arg\max_\theta E_{z\sim\gamma_t(z)}\left[\log p(X,z|\theta)\right]$

- Choosing the number of clusters:

→ Heuristic methods: Elbow / Knee heuristic: Plot within-cluster sum of squared distances for varying K

→ optimal k at knee

Gap statistic: Compare within-cluster variation to uniform data. ...

Silhouette:

Per point difference of the within-cluster mean distance for points in this cluster to points in closest cluster. Maximize this for all points.

→ Probabilistic methods:

* Need a generative model that defines the data likelihood $\hat{L} = p(X|\hat{z},\hat{\theta})$ ($\hat{z},\hat{\theta}$: optimal parameters)

Bayesian information criterion: (BIC)

$$BIC = M\log N - 2\log\hat{L}$$

M: num free parameters; N: num samples

Akaike information criterion: (AIC)

$$AIC = 2M - 2\log\hat{L}$$

- Hierarchical clustering:

→ Agglomerative: Bottom up

→ Divisive: Top down

ML 13 - Privacy

- What do we have to consider beyond accuracy:
  - * Privacy (models with large capacity can memorize data)
  - * Security (fool classifier with adversarial input)
  - * Fairness
  - * Explainability
  - * Accountability (can an outside editor ensure the system works as intended?)

- Differential privacy: (DP)

→ Setup: Input space X with symmetric neighbouring relation $\simeq$

Output space Y

Function of interest f and privacy parameter $\varepsilon \geq 0$

→ Randomized mechanism:

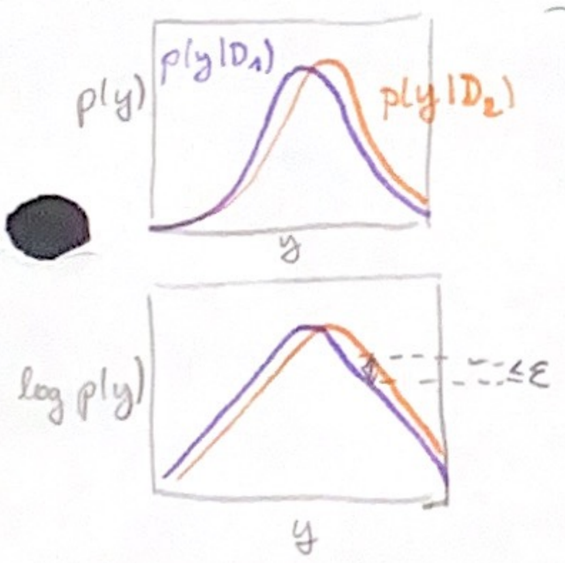
→ computes the function f and adds noise

* $M_f: X \to Y$ is ε-differentially private if for all neighboring inputs $X \simeq X'$ ("a single instance is changed") and for all set of outputs $Y \subseteq Y$ we have:

$$e^{-\varepsilon} \leq \frac{\mathbb{P}[M_f(X) \in Y]}{\mathbb{P}[M_f(X') \in Y]} \leq e^{\varepsilon} \quad \Rightarrow \left| \log(\cdot) - \log(\cdot) \right| \leq \varepsilon$$

$\mathbb{P}[M_f(X) \in Y]$: distribution over noise z for every single data

$\mathbb{P}[M_f(X') \in Y]$: prob. of output y for changed input is very similar



the two distributions w/ changed data must be similar

- Laplace - Mechanism:

→ global sensitivity ~~function~~: of $f: X \to \mathbb{R}^d$:

$$\Delta_p = \sup_{X \simeq X'} \| f(X) - f(X') \|_p$$

$X \simeq X'$: captures what is protected

Δ_p: measures by the magnitude by which a single instance can change the output of the func in the worst case.

Output perturbation w/ Laplace noise:

1) A curator holds data $X = (x_1, \dots, x_n) \in \mathcal{X}$
2) The curator computes $f(X)$
3) They sample i.id. Laplace noise $Z \sim Lap(0, \frac{\Delta_1}{\varepsilon})^d$
4) They reveal the noisy data $f(X) + Z$

DP for Machine Learning:

Goal: Train a model on dataset D and release the optimal weights (from the private data) in public.

Setup:
- dataset D
- parameters θ
- $f(D) = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(D, \theta)$ are the optimal weights
- $D \simeq D'$ when they differ in one instance
  ("D neighbors D'")

→ Techniques:

* Perturb input (add noise to dataset)
* Perturb weights (add noise to weights)
* Perturb objective (Optimize $\mathcal{L}(D, \theta) + \theta^T Z$) — Z: noise
* Perturb gradients

→ Fundamental properties of DP.

* Robustness to post-processing (operations after applying the DP-mechanism preserve DP)
* Composition (Apply multiple DP-mechanisms)
* Group privacy (If $\mathcal{M}$ is ε-DP w.r.t. $X \simeq X'$, then applying $\mathcal{M}$ to data with t changed instances is $(t\varepsilon)$-DP)

- Federated Learning:

→ learning a model w/o any centralized entity having access to all the data

1) Send weights from server to user
2) User computes loss + gradient locally
3) Send updated weights back (add randomization at user level)

ML 14 - Fairness

- What causes bias?
  - * Tainted (verunreinigt) training data ("If there is already a bias in the data")
  - * Skewed sample (selection bias; initial predictions may influence future observations)
    - ↳ "What is in the data is as important as what is not in the data"
  - * Proxies: Some features are legally protected (e.g. race, etc.). But other features are highly correlated w/ these.
  - * Sample size disparity: Models tend to fit larger groups first and possibly trade off accuracy for smaller groups
  - * Limited features: Features for minority groups may be less informative

- Definitions of fairness:
  - * Group Fairness: Treat all groups equally
  - * Individual Fairness: Treat similar individuals similarly
  - * Counterfactual Fairness: "What would the decision be if the individual was changed s.t. it belongs to a different group"

(group)

- How to achieve fairness:

Setup: $X \in \mathbb{R}^d$ ⟶ features of indiv.

$A \in \{a, b, \dots\}$ ⟶ sensitive features.

$R = r(X, A) \in \{0, 1\}$ ⟶ binary predictor

$Y \in \{0, 1\}$ ⟶ target

⇒ assume X, Y, A are generated from underlying distribution

⇒ Notation $P_a\{R\} = P\{R \mid A = a\}$

→ Fairness through unawareness:

- * Use $R = r(X)$ instead of $R = r(X, A)$
- \+ easy to use and implement
- − does not work due to correlations w/ other features

→ First Fairness Criterion: Independence:

- * R must be independent of A ⇒ $R \perp\!\!\!\perp A$
- * In binary classification: For all groups a, b we require $P_a\{R=1\} = P_b\{R=1\}$ (same percentage of applicants receive loan)

Approx. version ⇒ $\frac{P_a\{R=1\}}{P_b\{R=1\}} \geq 1 - \varepsilon$

* Achieve Independence: 1) Post-processing → decorrelate
2) Training-Time-Constraint
3) Pre-processing / representation learning
↳ Map (X, A) to a low-dim. representation Z which is not correlated to A. Train on Z.

⊖ cannot use information A
→ Optimal predictor R=Y is ruled out, since we need to cover each group equally

→ **Second Fairness criterion: Separation:**

* Require R and A to be conditionally independent given Y
⇒ $R \perp\!\!\!\perp A \mid Y$

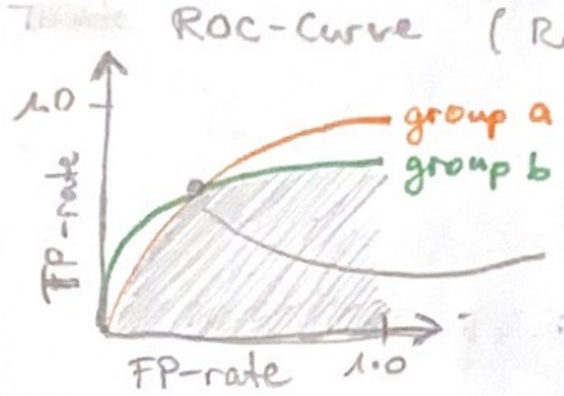
* In binary classification: (Enforce that the true positive and FP rates are equal for all groups)

$P_a(R=1 \mid Y=1) = P_b(R=1 \mid Y=1)$ (TP)

$P_a(R=1 \mid Y=0) = P_b(R=1 \mid Y=0)$ (FP)

* Commonly used relaxation: Only match the TP rate → Equal opportunity

* Achieve Separation: ROC-Curve (Receiver Operating Characteristic)



find thresholds for each group s.t. the TP and FP-rates match

⊕ Optimal predictor not ruled out

⊖ does not help closing gap between two groups

→ **Third Criterion: Sufficiency:**

* Require Y and A to be independent conditional on the target:
⇒ $Y \perp\!\!\!\perp A \mid R$

* In binary classification: (Enforce that the rate of the target is the same given same scores for different groups)

$P_a(Y=1 \mid R=r) = P_b(Y=1 \mid R=r)$

⇒ Each group should be calibrated $\underbrace{P_a(Y=1 \mid R=r) = r}$

the confidence of the classifier should match the probability of the outcome

⊕ Equal chance of success (Y=1) given acceptance (R=1)

⊕ To predict Y we do not need A when we have R

⊖ may not help to close gaps

- **Adversarial Examples / Robustness:**
↳ deliberate perturbations of the data to achieve some malicious goal
↳ w.r.t. worst-case noise