ML 1

Topics:

1. Bayes Decision Theory
2. Parameter Estimation
3. Fisher Discriminant
4. PCA
5. SVM
6. Learning Theory & Kernels
7. model selection
8. NN 1
9. NN 2
10. Decision Trees & Random Forests
11. Boosting
12. Clustering
13. KRR
14. XAI
15. Applications

Lecture 1: Bayesian Decision Theory

What is Machine Learning?
Designing algorithms (machines) that efficiently convert finite sets of data (observations) into models with predictive capability.

Why Machine Learning?
Autonomous Decision Making: Substitute/support human decision making for achieving gains in efficiency in practical applications.
Knowledge Discovery: Finding laws that explain empirical phenomena.

Posterior: What is the prob. of being a certain class after observing x?

Bayes Decision Theory

A theoretical framework for building decision models that:
-Yields optimal classifiers (under some favourable conditions).
Helps us to understand how:
-properties of the data distribution (e.g. Gaussian-distributed),
-prior probabilities of classes,
-the criterion to optimize (e.g. maximum classification accuracy),
affect the classifier qualitatively and quantitatively.

features: x_1 length, x_2 lightness

Notation:

- $\omega_1, \omega_2, \ldots$ set of classes (e.g. salmon, sea bass, ...),
- $x \in \mathbb{R}^d$ vector of observations (e.g. x_1 is the length and x_2 is the lightness).

classes: salmon ω_1, sea bass ω_2

We are also given the probability laws:

- $P(\omega_j)$: probability of being of class j, prior
- $p(x \mid \omega_j)$: density function of measurements for each class,
- $p(x)$: density function of measurements (marginalized),

$p(\omega_j)$: salmon probability
$p(x|\omega_j)$: size, length of salmon class
$p(x)$: lengths of salmons
$p(\omega_j|x)$: now I have some lengths, what is the prob. that these are in salmon class?

but what we are truly interested in is:

- $P(\omega_j \mid x)$: probability of being of a certain class after observing x. posterior

Bayes Theorem:

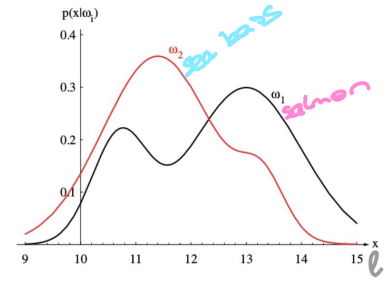
posterior = likelihood · prior / evidence

$$P(\omega_j \mid x) = \frac{p(x \mid \omega_j) \cdot P(\omega_j)}{p(x)}$$

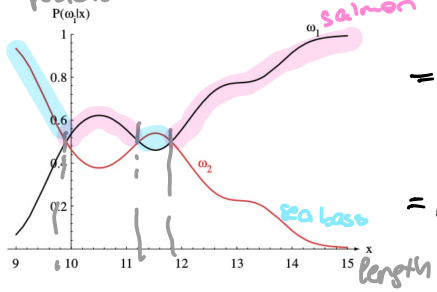
Optimally Accurate Classifier

optimal decision function: $\arg\max_j P(\omega_j \mid x)$

likelihood



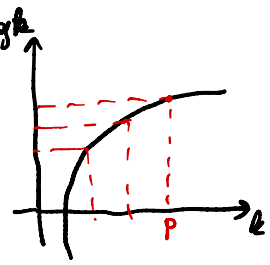
posterior



Alternate Formulas of the decision:

$$= \arg\max_j \frac{P(x|\omega_j) \cdot p(\omega_j)}{p(x)}$$

$$= \arg\max_j p(x|\omega_j) \cdot p(\omega_j)$$



$$= \arg\max_j \log\left[p(x|\omega_j) \cdot p(\omega_j)\right]$$

$$= \arg\max_j \log P(x|\omega_j) + \log P(\omega_j)$$

ℓ salmon : 11,5-ish (avg.) ... < ℓ < 9,75 → sea bass

ℓ sea bass : 13,5-ish (avg.) 9,5 < ℓ < 11 → salmon

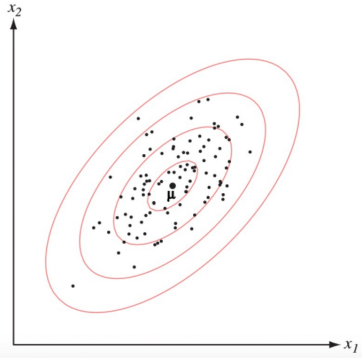
11 < ℓ < 12 → sea bass

12 < ℓ < ... → salmon

Multivariate Normal Distributions

integrates to 1

$$p(x|\omega_j) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma_j)}} \exp\left(-\frac{1}{2}(x-\mu_j)^\top \Sigma_j^{-1}(x-\mu_j)\right)$$



- μ_j is the mean (center of the data distribution) location
- Σ_j is the covariance (elongation of the data distribution and correlation between dimensions).

Image source: Duda et al. (2000)

Classifier for Gaussians ($\Sigma_1 = \Sigma_2$)

Recall: The optimal classifier is $\arg\max_j[\log p(x|\omega_j) + \log P(\omega_j)]$, and we have the data distributions:

$$p(x|\omega_j) \overset{\log}{=} \log\left(\frac{1}{\sqrt{(2\pi)^d \det(\Sigma_j)}}\right) + \log\exp\left(-\frac{1}{2}(x-\mu_j)^\top \Sigma_j^{-1}(x-\mu_j)\right)$$

cst, does not depend on j

because $\Sigma_1 = \Sigma_2$, there we have a single matrix

$$= \arg\max_j \left[-\frac{1}{2}(x-\mu_j)^T \Sigma^{-1}(x-\mu_j) + \log P(\omega_j)\right]$$

$$= \arg\max_j \left[\underbrace{-\frac{1}{2}x^T\Sigma^{-1}x}_{\text{cst, does not depend on } j} + x^T\underbrace{\Sigma^{-1}\mu_j}_{w_j} \underbrace{-\frac{1}{2}\mu_j^T\Sigma^{-1}\mu_j + \log P(\omega_j)}_{+b_j}\right]$$

log of a product = sum of the log

log of exponential = identity

adding something constant to argmax (cms function), don't change anything

Classifier for Gaussians ($\Sigma_1 = \Sigma_2$) → db is linear

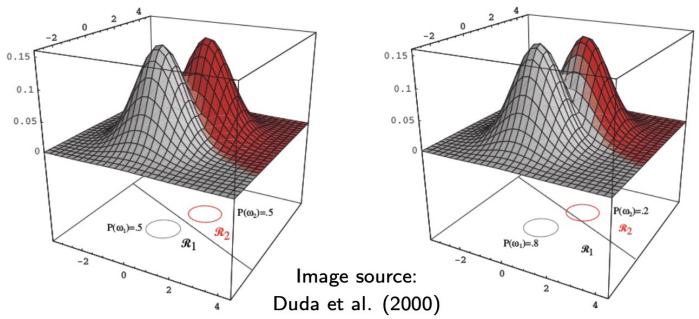
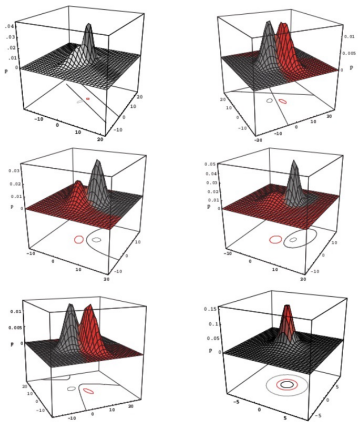


Image source: Duda et al. (2000)

- Decision boundary is linear and oriented by mean and covariance.
- Offset is controlled by class prior probabilities.

Classifier for Gaussians ($\Sigma_1 \neq \Sigma_2$) → then the db is quadratic



- When covariances Σ_1 and Σ_2 are not the same, the decision boundary is quadric instead of linear. Quadrics include circle, ellipse, parabola, hyperbola, and degenerate forms.

Image source: Duda et al. (2000)

Classifying Binary Data

- Assume that our data is binary, i.e. $x \in \{0,1\}^d$, with each dimension generated *independently* according to some Bernoulli distribution:

$$P(x_i = 0 \mid \omega_j) = 1 - q_{ij}$$
$$P(x_i = 1 \mid \omega_j) = q_{ij}$$

where q_{ij} are the parameters.

- The probability of the whole multivariate observation can be written as:

$$P(x|\omega_{ij}) = \prod_{i=1}^{d} [q_{ij}x_i + (1 - q_{ij}) \cdot (1 - x_i)]$$

- **Question:** How to express the optimal decision boundary

$$\arg\max_j P(\omega_j|x)$$

Minimum Cost Decisions

Example: Buying a Car

- Suppose you would like to purchase a second-hand car. After observing the car (collecting a vector of measurements x), you assess that it has a defect with probability

$$P(\text{defect} \mid x) = 0.1 \qquad P(\text{no defect} \mid x) = 0.9$$

- Concretely, the decision you need to take is *not* to *classify* the whether the car has a defect or not, but whether to *buy* the car or not.
- For this, we need to evaluate the cost of each scenario, e.g.

$$\text{cost}(\text{buy} \mid \text{defect}) = 100.0$$
$$\text{cost}(\text{buy} \mid \text{no defect}) = -20.0$$
$$\text{cost}(\text{not buy} \mid \text{defect}) = 0.0$$
$$\text{cost}(\text{not buy} \mid \text{no defect}) = 0.0$$

and take the action with the lowest expected cost.

Classification Accuracy Special Case

Recall: $\lambda(\alpha_k|x) = \sum_{j=1}^{C} \lambda(\alpha_k|\omega_j)P(\omega_j|x)$

Show that the problem of maximum accuracy classification is a special instance of expected cost minimization with a particular set of actions $(\alpha_k)_k$ and a particular cost function $\lambda(\alpha_k|\omega_j)$.

α_k: classify as ω_k

$\lambda(\alpha_k \mid \omega_j) = -\delta_{jk}$

$\operatorname{argmin}_k \sum_j -\delta_{jk} \cdot P(\omega_j|x)$

$= \operatorname{argmin}_k \{-P(\omega_k|x)\} = \operatorname{argmax}_k P(\omega_k|x)$

$\operatorname{argmin}(-) = \operatorname{argmax}$

Classifying Binary Data

Recall: The optimal classifier is $\arg\max_j[\log p(x|\omega_j) + \log P(\omega_j)]$, and we have the data distributions:

$$P(x|\omega_j) = \prod_{i=1}^{d} [q_{ij}x_i + (1 - q_{ij}) \cdot (1 - x_i)]$$

$x_i = 0 \to 1 - q_{ij}$, $x_i = 1 \to q_{ij}$

$q_{ij}^{x_i} \cdot (1-q_{ij})^{(1-x_i)}$

$$= \operatorname*{argmax}_j \left[\sum_i \log\left(q_{ij}^{x_i}(1-q_{ij})^{(1-x_i)}\right) + \log P(\omega_j)\right]$$

$$= \operatorname*{argmax}_j \left[\sum_i x_i \underbrace{\log q_{ij}}_{a_{ij}} + (1-x_i)\underbrace{\log(1-q_{ij})}_{b_{ij}} + \log P(\omega_j)\right]$$

$$= \operatorname*{argmax}_j \left(x^T a_j + (1-x)^T b_j + \log P(\omega_j)\right)$$

$$= \operatorname*{argmax}_j \left\{x^T(a_j - b_j) + 1^T b_j + \log P(\omega_j)\right\}$$

Minimum Cost Decisions

General problem formulation:

- Let $(\alpha_k)_k$ be the set of actions. The expected cost "λ" of taking a certain action is given by

$$\lambda(\alpha_k|x) = \sum_{j=1}^{C} \lambda(\alpha_k|\omega_j)P(\omega_j|x)$$

where $\lambda(\alpha_k|\omega_j)$ is the cost of taking action α_k given the class ω_j.

- The optimal action to take is therefore: $\arg\min_k \lambda(\alpha_k|x)$

$\lambda(\text{buy}|x) = 100.0 \cdot 0.1 + (-20) \cdot 0.9 = -8$

$\lambda(\text{not buy}|x) = 0.0 \cdot 1 + 0.0 \cdot 0.9 = 0$

$\operatorname{argmin}\{\text{buy}: -8, \text{not buy}: 0\} = \text{buy}$

Measuring Classification Error

- So far, we have studied what the decision boundary should be in order to predict optimally.
- However, in certain cases, it is also important to determine what is the *expected error* of the classifier (e.g. to determine whether the classifier is good enough for practical use).
- The expected error is the probability that the data is of a different class than the one predicted, e.g. for a binary classifier:

$$P(\text{Err} \mid x) = \begin{cases} P(\omega_1 \mid x) & \text{if "decide } \omega_2\text{"} \\ P(\omega_2 \mid x) & \text{if "decide } \omega_1\text{"} \end{cases}$$

- For the Bayes optimal classifier, this reduces to

$$P(\text{Err} \mid x) = \min\{P(\omega_1 \mid x), P(\omega_2 \mid x)\}$$

- The expected error of this maximally accurate classifier is computed as the integral of its error probability over the distribution $p(x)$.

$$P(\text{Err}) = \int_x P(\text{Err} \mid x) p(x) dx$$
$$= \int_x \min\{P(\omega_1 \mid x), P(\omega_2 \mid x)\} p(x) dx$$

This is also known as the *Bayes error rate*.

- Generally, this integral cannot be solved analytically, because of the min function. Error must instead be evaluated numerically/empirically, or it can also be **bounded** analytically.

Bounding the Error of the Classifier

Very basic bound

- Observe for binary classification that $P(\omega_2 \mid x) = 1 - P(\omega_1 \mid x)$.
- The error of an optimal binary classifier can be bounded as:

$$P(\text{Err}) = \int_x \min\{P(\omega_1 \mid x), P(\omega_2 \mid x)\} p(x) dx$$
$$= \int_x \min\{P(\omega_1 \mid x), 1 - P(\omega_1 \mid x)\} p(x) dx$$
$$\leq \int_x 0.5\, p(x) dx$$
$$= 0.5$$

i.e. the classifier predicts the correct class at least 50% of the time.

- Note that, unlike an empirical evaluation, this result is general and independent on the data distributions.

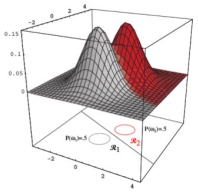
→ If we have model knowledge (posterior, prior and likelihood) then Bayes will have the lowest error rate.

Recap: Bayes Optimal Classifier

Recap:

- Assume our data is generated for each class ω_j according to the multivariate Gaussian distribution $p(\mathbf{x}|\omega_j) = \mathcal{N}(\boldsymbol{\mu}_j, \Sigma)$ and with class priors $P(\omega_j)$. The Bayes optimal classifier is derived as

$$\arg\max_j \{P(\omega_j|\mathbf{x})\}$$
$$= \arg\max_j \{\log p(\mathbf{x}|\omega_j) + \log P(\omega_j)\}$$
$$= \arg\max_j \left\{\mathbf{x}^\top \Sigma^{-1} \boldsymbol{\mu}_j^\top - \frac{1}{2} \boldsymbol{\mu}_j \Sigma^{-1} \boldsymbol{\mu}_j + \log P(\omega_j)\right\}$$



- Given our generative assumptions, there is no better classifier than the one above.
- However, in practice, we don't know these distributions and only have the data.

Bounding the Error of the Classifier

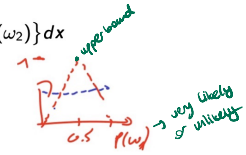
Another simple bound

$$P(\text{Err}) = \int_x \min\{P(\omega_1 \mid x), P(\omega_2 \mid x)\} p(x) dx$$
$$= \int_x \min\{P(\omega_1 \mid x) p(x), P(\omega_2 \mid x) p(x)\} dx$$

Recall: $P(\omega_j \mid x) p(x) = p(x \mid \omega_j) P(\omega_j)$

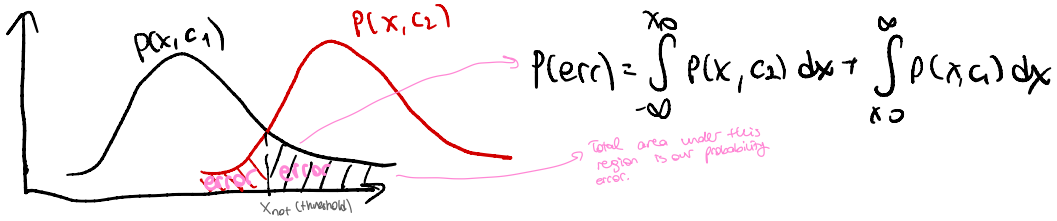
$$= \int_x \min\{p(x \mid \omega_1) P(\omega_1), p(x \mid \omega_2) P(\omega_2)\} dx$$
$$\leq \int_x \min\{\textstyle\sum_j \{p(x \mid \omega_j)\} P(\omega_1), \sum_j \{p(x \mid \omega_j)\} P(\omega_2)\} dx$$
$$= \left(\textstyle\sum_j \int_x \{p(x \mid \omega_j)\} dx\right) \cdot \min\{P(\omega_1), P(\omega_2)\}$$
$$= 2 \cdot \min\{P(\omega_1), P(\omega_2)\}$$

sum of 2 classes; integrates to 1; sum if 2 classes; so it depends on the class priors



Additional insight: The optimal classifier improves its accuracy when one class prior probability is strongly dominant over to the other class.

- best classifier if I know the data properties, when we know everything about datas generation
- conditions: data Gaussian distributed & both classes same cov → then Bayes best classifier
- Limitations: real life we don't know everything about data, (prior, density function...)
  - Σ^{-1} very hard to calculate



- Bayes error:
  - → lowest possible error rate that any classifier could achieve on a given problem.
  - → Bayes error all possible classifiers: linear or nonlinear
  - → It's about the theoretical minimum error rate achievable by any classifier, given perfect knowledge of the data distribution.

---

iid ≙ independent and identically distributed

→ Independent: $P(xy) = P(x) \cdot P(y)$

→ Identically: all the random variables in the set have the same distribution same mean, variance, PDF (prob. density function)

$$\sum p(x) = 1$$

Harmonic mean: $\min(a,b) \leq \frac{2}{\frac{1}{a} + \frac{1}{b}} = \frac{2ab}{a+b}$

Lecture 2: Parameter Estimation

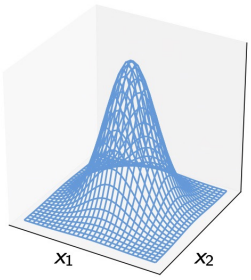
Learning $p(x \mid \omega_j)$: Model-Based

we know its Gaussian, but we don't know the parameters (μ, Σ)

Idea:

- Assume that $p(x \mid \omega_j)$ is a known parametric form, e.g. $\mathcal{N}(\mu_j, \Sigma_j)$ where μ_j, Σ_j are the parameters of the distribution.
- Estimate the parameters of the distribution that best fit the few observations we have. that explains the generated data the best

parameters = # dimensions



Advantage:

- With the model-based approach, we need to estimate a finite and typically small number of model parameters and not the whole data distribution. much smaller and not exponential

Maximum Likelihood Estimation

Goal: Let $\{p(x \mid \theta), \theta \in \Theta\}$ be a set of density functions (e.g. Gaussian), where θ denotes a parameter vector (e.g. mean / covariance). We would like to find the parameter θ that is the most likely with respect to the data.

Maximum Likelihood (ML) Approach:

- Assume that we have a dataset $\mathcal{D} = (x_1, \ldots, x_N)$.
- Assume that each example $x_k \in \mathbb{R}^d$ in the dataset has been is generated independently and from the same density function $p(x \mid \theta)$.
- In that case, the joint density function can be written as:

$$p(\mathcal{D} \mid \theta) = \prod_{k=1}^{N} p(x_k \mid \theta)$$

product of all the densities

We also call this quantity the *likelihood* of θ w.r.t. the dataset $\mathcal{D}$.

- The best parameter is then given by $\hat{\theta} = \arg\max_{\theta} p(\mathcal{D} \mid \theta)$.

Max Likelihood, Simple Gaussian Case

we're looking for θ that best represent our dataset $\mathcal{D}$.

Assume the data density is modeled as a univariate Gaussian with unit variance and unknown mean θ. For a given data point x_k, the density function can be written as:

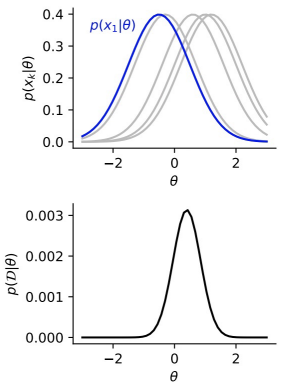
$$p(x_k \mid \theta) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}(x_k - \theta)^2\right]$$

Using the independence assumption, the joint density becomes:

objective function:

$$p(\mathcal{D} \mid \theta) = \prod_{k=1}^{N} \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}(x_k - \theta)^2\right]$$

Question: How to find θ that maximizes the function $p(\mathcal{D} \mid \theta)$.



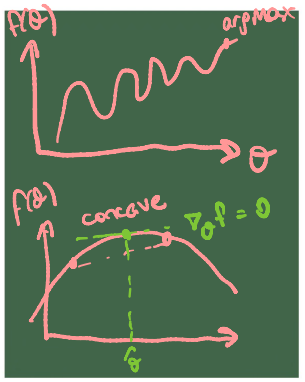
Finding the Maximum of a Function

For some function f of interest (e.g. the data likelihood) we would like to compute:

$$\hat{\theta} = \arg\max_{\theta} f(\theta)$$

When the function to optimize is continuously differentiable and concave, the maximum is found at the point where the gradient is zero.

$$\nabla_\theta f(\theta) = \begin{bmatrix} \partial f/\partial\theta_1 \\ \partial f/\partial\theta_2 \\ \vdots \\ \partial f/\partial\theta_h \end{bmatrix} = \mathbf{0}$$



concave check: 2nd derivate $\stackrel{!}{=} 0$ → $\frac{\partial^2}{\partial\theta^2} p(\mathcal{D}|\theta) \stackrel{!}{=} 0$

Max Likelihood, Simple Gaussian Case

Observation: The function $p(\mathcal{D} \mid \theta)$ is not concave with θ.

Idea: Transform the function in a way that
(i) doesn't change its argmax and
(ii) makes it concave.

Applying the logarithm ensures the two properties above:

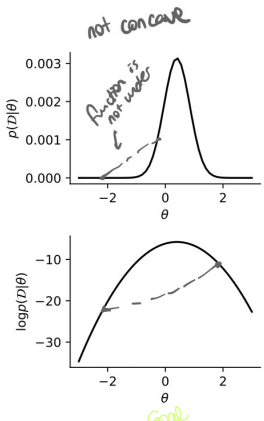
$$\log p(\mathcal{D} \mid \theta) = \log \prod_{k=1}^{N} \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}(x_k - \theta)^2\right]$$

$$= \sum_{k=1}^{N} \left[-\frac{1}{2}\log(2\pi) - \frac{1}{2}(x_k - \theta)^2\right]$$

cst. concave

not concave

Function is not under



Goal

Max Likelihood, Simple Gaussian Case

Having found the log-likelihood w.r.t. $\mathcal{D}$ to be

cst. → 0

$$\log p(\mathcal{D} \mid \theta) = \sum_{k=1}^{N} \left[-\frac{1}{2}\log(2\pi) - \frac{1}{2}(x_k - \theta)^2\right],$$

the best parameter $\hat{\theta}$ can then be found by solving $\nabla_\theta \log P(\mathcal{D} \mid \theta) = 0$.

$$\nabla_\theta \log P(\mathcal{D}|\theta) = \sum_{k=1}^{N} -\frac{1}{2} \cdot 2\,(x_k - \theta)\cdot(-1) = \sum_{k=1}^{N} x_k - \theta = 0$$

we don't have θ here, so only multiplication by N

$$\Rightarrow \sum_{k=1}^{N} x_k = \sum_{k=1}^{N} \theta = N\cdot\theta$$

$$\hat{\theta} = \frac{1}{N}\sum_{k=1}^{N} x_k$$

empirical mean

choose θ that best explains the data

Empirical mean

$\cdot$ set $\theta = \bar{x}$

Max Likelihood, Multivariate Case

extention of gaussian to multivariate

Σ^{-1} is PSD → $-\Sigma^{-1}$ neg. def. → concave function

The log-likelihood of a *multivariate* Gaussian distribution w.r.t. $\mathcal{D}$ is given by

$$\log p(\mathcal{D}|\mu) = \sum_{k=1}^{N} -\frac{1}{2}\log\left[(2\pi)^d \det(\Sigma)\right] - \frac{1}{2}(x_k - \mu)^\top \Sigma^{-1}(x_k - \mu)$$

does not depend on μ

Question: Assuming Σ is fixed, what is the optimal parameter vector μ?

$$\nabla_\mu \log P(D|\mu) = \sum_{k=1}^{N} 0 + \Sigma^{-1}(x_k - \mu) = 0$$

$$\sum_{k=1}^{N} \Sigma^{-1} x_k = N\Sigma^{-1}\mu \rightarrow \hat{\mu} = \frac{1}{N}\sum_{k=1}^{N} x_k$$

emp. mean of data

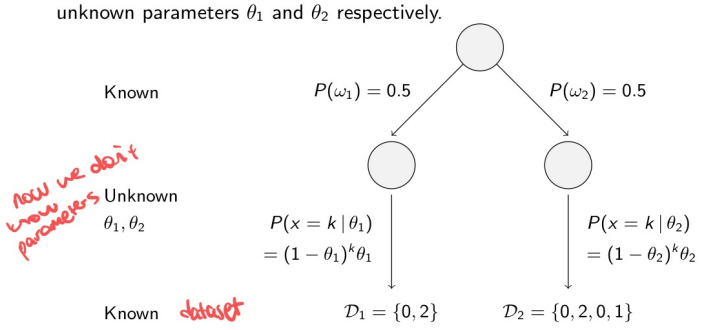
$$\left(-\frac{1}{2}\cdot\Sigma^{-1}(x_k-\mu)^2\right)' = -\Sigma^{-1}(x_k-\mu)\cdot(-1)$$

shift μ to $\bar{x}$ to maximally explain the data

Building a Classifier with ML

① learn θ_1, θ_2 given the available data ② use θ_1 and θ_2 in Bayes to build the max. accurate classifier

Consider a labeled dataset containing two examples for the first class, and four examples for the second class. Points are in $\mathbb{N}_0$ and we assume they are generated from classes ω_1, ω_2 following geometric probability distributions of unknown parameters θ_1 and θ_2 respectively.



now we don't know parameters

from dataset we'll try to learn the parameters of distribution

Building a Classifier with ML

Recall:

$P(\omega_1) = 0.5$
$P(\omega_2) = 0.5$
$P(x = k \mid \theta_1) = (1-\theta_1)^k\theta_1$
$P(x = k \mid \theta_2) = (1-\theta_2)^k\theta_2$
$\mathcal{D}_1 = \{0, 2\}$
$\mathcal{D}_2 = \{0, 2, 0, 1\}$

formula for dataset → π

$$P(D_1|\theta_1) = (1-\theta_1)^k\theta_1 = (1-\theta_1)^0\cdot\theta_1 = \theta_1\cdot(1-\theta_1)^2\cdot\theta_1$$

$$P(x=0|\theta_1)\,P(x=2|\theta_1) = \theta_1^2(1-\theta_1)^2$$

not convex, apply log

convex ✓

$$\log P(D_1|\theta_1) = 2\log\theta_1 + 2\log(1-\theta_1)$$

gradient: $\nabla_\theta(\dots) = \frac{2}{\theta_1} + \frac{2}{(1-\theta_1)}\cdot(-1) \overset{!}{=} 0$

$\rightarrow \theta_1 = 1/2$

our ML-estimator for the parameters of first class (ω_1).

Building a Classifier with ML

Recall:

$P(\omega_1) = 0.5$
$P(\omega_2) = 0.5$
$P(x = k \mid \theta_1) = (1-\theta_1)^k\theta_1$
$P(x = k \mid \theta_2) = (1-\theta_2)^k\theta_2$
$\mathcal{D}_1 = \{0, 2\}$
$\mathcal{D}_2 = \{0, 2, 0, 1\}$

θ_2 θ_2

$$P(D_2|\theta_2) = \theta_2\cdot(1-\theta_2)^2\theta_2\cdot\theta_2\cdot(1-\theta_2)\theta_2$$

$D_2 = \{0, 2, 0, 1\}$

$$= \theta_2^4(1-\theta_2)^3$$

$$\log P(D_2|\theta_2) = 4\log\theta_2 + 3\log(1-\theta_2)$$

$$\rightarrow 4\theta_2 - 3(1-\theta_2) \overset{!}{=} 0$$

$$4\theta_2 + 3\theta_2 = 3$$

$$7\theta_2 = 3$$

$$\theta_2 = 3/7$$

Bayes decision theory + ML example

Building a Classifier with ML

after finding the full model, apply Bayes

what do you want to classify? Assume a new datapoint $x=1$ is coming.

Recall:

prior $P(\omega_1) = 0.5$
prior $P(\omega_2) = 0.5$
$P(x = k \mid \theta_1) = (1-\theta_1)^k\theta_1$
$P(x = k \mid \theta_2) = (1-\theta_2)^k\theta_2$
$\mathcal{D}_1 = \{0, 2\}$
$\mathcal{D}_2 = \{0, 2, 0, 1\}$

$$\text{argmax}_j\, P(\omega_j|x) = \text{argmax}_j\, P(x|\omega_j)\,P(\omega_j)$$

ω_1: $P(x=1|\omega_1)\cdot P(\omega_1) = (1-\theta_1)^1\theta_1\cdot P(\omega_1)$

$$= \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2} = 0{,}125$$

ω_2: $P(x=1|\omega_2)\cdot P(\omega_2) = (1-\theta_2)^1\cdot\theta_2\cdot P(\omega_2)$

$$= (1-\theta_2)^1\theta_2\cdot P(\omega_2)$$

$$= 3/7\cdot 4/7\cdot 0{,}5 = 0{,}124$$

outcome of classifier ⇒ decide (argmax) ω_1

Bayes Parameter Estimation

$p(\theta)$: so in bayes param. est. user needs to specify what user believes θ is, before observing any data

Assuming some dataset $\mathcal{D} = (x_1, \dots, x_N)$. We would like to model this dataset using some probability function $p(\mathcal{D}|\theta)$ with θ some unknown parameter that needs to be learned.

ML parameter estimation: Chose the parameter θ that maximises the data likelihood:

$$\hat{\theta} = \arg\max_\theta p(\mathcal{D}|\theta)$$

Bayes parameter estimation: Instead of learning a fixed estimate, build a posterior distribution over this parameter using the Bayes theorem:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}$$

prior distribution over new parameter θ

find a distribution over our parameters

$$= \frac{p(\mathcal{D}|\theta)p(\theta)}{\int_\theta p(\mathcal{D}|\theta)p(\theta)d\theta}$$

This approach requires to define a *prior* distribution $p(\theta)$ which models our initial belief about the parameter θ before observing the data.

From ML to Bayes Classifiers

How to make a classifier from $P(\theta|D)$?

ML classifier: Class posteriors are given by:

$$P(\omega_i \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid \omega_i; \hat{\theta}_i)\, P(\omega_i)}{\sum_i p(\boldsymbol{x} \mid \omega_i; \hat{\theta}_i)\, P(\omega_i)}$$

where $\hat{\theta}_i$ is the maximum likelihood parameter for the distribution of class ω_i.

Bayes classifier: Class posteriors are computed by bypassing the intermediate computation of the parameters $\hat{\theta}_i$, and instead conditioning directly on the data:

$$P(\omega_i \mid \boldsymbol{x}, \mathcal{D}) = \frac{p(\boldsymbol{x} \mid \omega_i, \mathcal{D})\, P(\omega_i \mid \mathcal{D})}{\sum_i p(\boldsymbol{x} \mid \omega_i, \mathcal{D})\, P(\omega_i \mid \mathcal{D})} = \frac{p(\boldsymbol{x} \mid \omega_i, \mathcal{D}_i)\, P(\omega_i)}{\sum_i p(\boldsymbol{x} \mid \omega_i, \mathcal{D}_i)\, P(\omega_i)}$$

condition the prob. to dataset

class priors are known

Goal: connect this posterior to the posterior of data, $P(\theta|D)$.

Bayes Classifiers (cont.)

The terms of the class posterior:

$$P(\omega_i \mid \boldsymbol{x}, \mathcal{D}) = \frac{p(\boldsymbol{x} \mid \omega_i, \mathcal{D}_i)\, P(\omega_i)}{\sum_i p(\boldsymbol{x} \mid \omega_i, \mathcal{D}_i)\, P(\omega_i)}$$

can be expressed with model parameters as:

$$p(\boldsymbol{x} \mid \omega_i, \mathcal{D}_i) = \int p(\boldsymbol{x} \mid \theta_i, \omega_i, \mathcal{D}_i)\, p(\theta_i \mid \omega_i, \mathcal{D}_i) d\theta_i = \int p(\boldsymbol{x} \mid \theta_i)\, p(\theta_i \mid \mathcal{D}_i) d\theta_i$$

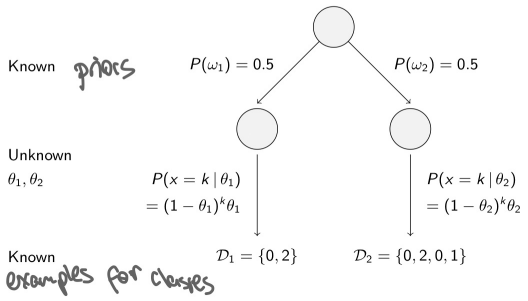
and

$$p(\theta_i \mid \mathcal{D}_i) = \frac{p(\mathcal{D}_i \mid \theta_i)\, p(\theta_i)}{\int p(\mathcal{D}_i \mid \theta_i)\, p(\theta_i) d\theta_i}.$$

are our Bayes parameter estimates.

Building a Classifier with Bayes

Recall: we consider the following data generating process



Building a Classifier with Bayes

Recall:

$P(\omega_1) = 0.5$
$P(\omega_2) = 0.5$
$P(x = k \mid \theta_1) = (1-\theta_1)^k \theta_1$
$P(x = k \mid \theta_2) = (1-\theta_2)^k \theta_2$
$\mathcal{D}_1 = \{0, 2\}$
$\mathcal{D}_2 = \{0, 2, 0, 1\}$

... and we further set:

uniform distribution

$p(\theta_1) \sim \mathcal{U}(0, 1)$

$p(\theta_2) \sim \mathcal{U}(0, 1)$

1) Estimate a θ_i:

Bayes:

$$P(\theta_1 \mid D_1) = \frac{P(D_1 \mid \theta_1) \cdot p(\theta_1)}{\int p(D_1 \mid \theta_1) \cdot p(\theta_1) d\theta_1}$$

posterior

$$= \frac{\theta_1^2 (1-\theta_1)^2 \cdot 1}{\int_0^1 \theta_1^2 (1-\theta_1)^2 \cdot 1 \cdot d\theta} = 30\, \theta_1^2 (1-\theta_1)^2$$

due to the effect of observing some data, estimate of θ_1 gets shrinked, centered around 0.5.

Building a Classifier with Bayes

Recall:

$P(\omega_1) = 0.5$
$P(\omega_2) = 0.5$
$P(x = k \mid \theta_1) = (1-\theta_1)^k \theta_1$
$P(x = k \mid \theta_2) = (1-\theta_2)^k \theta_2$
$\mathcal{D}_1 = \{0, 2\}$
$\mathcal{D}_2 = \{0, 2, 0, 1\}$

... and we further set:

prior distribution

uniform!

$p(\theta_1) \sim \mathcal{U}(0, 1)$ $P(\theta_1)=1$

$p(\theta_2) \sim \mathcal{U}(0, 1)$ $P(\theta_2)=2$

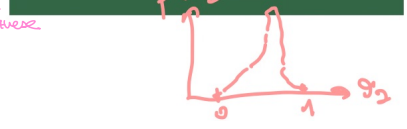
we chose these

Bayes:

$$P(\theta_2 \mid D_2) = \frac{P(D_2 \mid \theta_2) \cdot P(\theta_2)}{\int_0^1 P(D_2 \mid \theta_2) \cdot P(\theta_2) d\theta_2}$$

$$= \frac{\theta_2^4 (1-\theta_2)^3 \cdot 1}{\int_0^1 \theta_2^4 (1-\theta_2)^3 \cdot 1 \cdot d\theta} = 280 \cdot \theta_2^4 (1-\theta_2)^3$$

$p(\theta_2 \mid D_2)$



Building a Classifier with Bayes

1) estimate post. distr. of each class with parameters

2) express the prob. of new datapoint coming from each class

focus on case $X=1$

Recall:

$P(\omega_1) = 0.5$
$P(\omega_2) = 0.5$
$P(x = k \mid \theta_1) = (1-\theta_1)^k \theta_1$
$P(x = k \mid \theta_2) = (1-\theta_2)^k \theta_2$
$\mathcal{D}_1 = \{0, 2\}$
$\mathcal{D}_2 = \{0, 2, 0, 1\}$

... and we further set:

$p(\theta_1) \sim \mathcal{U}(0, 1)$

$p(\theta_2) \sim \mathcal{U}(0, 1)$

$$P(x \mid w_1, D_1) = \int_0^1 p(x \mid \theta_1) \cdot p(\theta_1 \mid D_1) d\theta_1$$

$$= \int_0^1 (1-\theta_1) \cdot \theta_1 \cdot 30 \cdot \theta_1^2 (1-\theta_1)^2 d\theta_1$$

$$= \int_0^1 (1-\theta_1)^3 \cdot \theta_1^3 \cdot 30\, d\theta_1 = 3/14 = 0{,}214$$

$$P(x \mid w_2, D_2) = \int_0^1 P(x \mid \theta_2) \cdot P(\theta_2 \mid D_2) d\theta_2$$

$$= \int_0^1 (1-\theta)^4 \theta^3 \cdot 280\, d\theta_2 = \frac{2}{9} = 0{,}222$$

Building a Classifier with Bayes (3) the decision

Recall:

$P(\omega_1) = 0.5$
$P(\omega_2) = 0.5$
$P(x = k \mid \theta_1) = (1-\theta_1)^k \theta_1$
$P(x = k \mid \theta_2) = (1-\theta_2)^k \theta_2$
$\mathcal{D}_1 = \{0, 2\}$
$\mathcal{D}_2 = \{0, 2, 0, 1\}$

... and we further set:

$$p(\theta_1) \sim \mathcal{U}(0,1)$$
$$p(\theta_2) \sim \mathcal{U}(0,1)$$

argmax $P(x|\omega_j, \mathcal{D}_j) \cdot p(\omega_j)$
j

ω_1: 0,216 . 0,5

ω_2: 0,222 . 0,5

decide for ω_2

ML said decide for ω_1, but Bayes said decide for ω_2. Why?

(1) Because in Bayes we introduced a new modelling set: $P(\theta_1) \sim \mathcal{U}(0,1)$
$P(\theta_2) \sim \mathcal{U}(0,1)$

With this we influenced our initial estimate of parameters

(2) More observed data means more shrinkage the distribution will have.
Dataset 2 had more data, so its estimate is better.
→ more confident as a classifier.

ML vs. Bayes:

Observations:

- ML and Bayes classifiers do not always produce the same decisions
- Bayes classifiers are influenced by the prior distribution and are consequently less sensitive to the data.
- Bayes classifiers will tend to favor the outcome that is supported by a larger amount of data.

- Bayes posterior is influenced by the prior, if little data is available.
- Bayes also provides a confidence estimate, ML just gives a number, without any interval of confidence.

- In practice, parameters of the class-conditioned distributions are not known and they must be inferred from some finite dataset.
- Two main approaches for learning these parameters: (1) maximum likelihood estimation and (2) Bayesian inference.
- Bayesian inference is often more difficult than maximum likelihood estimation (both analytically and computationally), because it requires integration.
- Bayesian inference readily incorporates interesting functionalities (inclusion of priors, construction of confidence estimates).

ML-Estimation Steps:

① Objective Function

② transform objective function to concave with log, so argmax doesn't change

log Π = Σ log + log ...

$$① \; p(\mathcal{D}|\theta) = \prod_{k=1}^{N} \frac{1}{\sqrt{2\pi}} exp[-\frac{1}{2}(x_k - \theta)^2]$$

$$② \Rightarrow \log p(\mathcal{D}|\theta) = \log \prod_{k=1}^{N} \frac{1}{\sqrt{2\pi}} exp[-\frac{1}{2}(x_k - \theta)^2]$$

$$= \sum_{k=1}^{N} [log(\frac{1}{\sqrt{2\pi}}) + log(exp[-\frac{1}{2}(x_k - \theta)^2])]$$

$$= \sum_{k=1}^{N} [-\frac{1}{2} log(2\pi) - \frac{1}{2}(x_k - \theta)^2]$$

$$\frac{\partial}{\partial\theta} \log p(\mathcal{D}|\theta) = \frac{\partial}{\partial\theta} \sum_{k=1}^{N} [-\frac{1}{2} log(2\pi) - \frac{1}{2}(x_k - \theta)^2] \stackrel{!}{=} 0$$

$$= \sum_{k=1}^{N} -\frac{1}{2} \cdot 2(x_k - \theta) \cdot (-1) = 0$$

$$\Leftrightarrow \sum_{k=1}^{N} x_k = N\theta$$

$$\Rightarrow \hat{\theta} = \frac{1}{N} \sum_{k=1}^{N} x_k$$

optimum θ of our ML is the mean of our data for multivariate Gaussians as well!

ML vs. Bayes: Gaussian Case

var is fix

Consider the simple data density $p(x \mid \mu) = \mathcal{N}(\mu, \sigma^2)$ with unknown parameter μ. We would like to compare the ML and Bayes approaches to estimate the parameter μ from some dataset $\mathcal{D} = \{x_1, \ldots, x_n\}$.

ML approach:

- The maximum likelihood estimate is given by $\hat{\mu} = \frac{1}{n} \sum_{k=1}^{n} x_k$, i.e. the empirical mean (cf. previous slides).

emp. mean

Bayes approach:

- Assuming some prior distribution $p(\mu) = \mathcal{N}(\mu_0, \sigma_0^2)$, the posterior distribution can be computed as:

>i.i.d!

$$p(\mu \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \mu) p(\mu)}{p(\mathcal{D})} = \alpha \prod_{k=1}^{n} p(x_k|\mu) p(\mu)$$

where α is a normalizing factor.

→ more data, more our Bayes will be closer to Max. Likelihood

- our goal is to find suitable parameters in real world

→ few datapoints: μ closer to μ_0 (prior)

- Bayes is more expensive, but it also gives confident estimates.

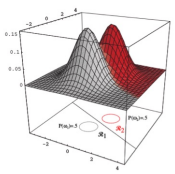
Recap: Parameter Estimation

Example of estimator:

- Maximum likelihood estimator:

$$\mu = \frac{1}{N} \sum_{k=1}^{N} \mathbf{x}_k$$

$$\hat{\Sigma} = \frac{1}{N} \sum_{k=1}^{N} (\mathbf{x}_k - \mu)(\mathbf{x}_k - \mu)^\top$$



Problem:

- The covariance matrix (and its inverse) may be difficult to estimate.
- We make an assumption about the data (e.g. Gaussian-distributed) which may not correspond to reality.

How to turn ML to Bayes?

1. calculate $P(D|\theta)$ from ML : 1. $P(D|\theta)$ Likelihood Fct.
   2. find $\hat{\theta}$: function → log → derivative

2. $P(\theta|D) = \frac{P(D|\theta) \cdot P(\theta)}{\int_{\theta} P(D|\theta) \cdot P(\theta) d\theta}$ $\quad \theta \in [0,1]$ take the integral

---

What is the likelihood of θ? $P(D|\theta) = \prod_{k=1}^{N} p(x_k|\theta)$ likelihood of θ

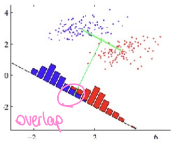
where D contains N samples, $x_1, \dots, x_n$

Lecture 3: Fisher Discriminant (LDA) Σ^{-1}

Derivation of Mean Separation

- The constrained optimization problem (subject to the constraint $\|\mathbf{w}\| = 1$) can be developed as:

$$\arg\max_{\mathbf{w}} |\mu_2(\mathbf{w}) - \mu_1(\mathbf{w})|$$
$$= \arg\max_{\mathbf{w}} \left| \frac{1}{N_2}\sum_{k\in\mathcal{C}_2} z_k(\mathbf{w}) - \frac{1}{N_1}\sum_{k\in\mathcal{C}_1} z_k(\mathbf{w}) \right|$$
$$= \arg\max_{\mathbf{w}} \left| \frac{1}{N_2}\sum_{k\in\mathcal{C}_2} \mathbf{w}^\top \mathbf{x}_k - \frac{1}{N_1}\sum_{k\in\mathcal{C}_1} \mathbf{w}^\top \mathbf{x}_k \right|$$
$$= \arg\max_{\mathbf{w}} \left| \mathbf{w}^\top \Big(\frac{1}{N_2}\sum_{k\in\mathcal{C}_2} \mathbf{x}_k\Big) - \mathbf{w}^\top \Big(\frac{1}{N_1}\sum_{k\in\mathcal{C}_1} \mathbf{x}_k\Big) \right|$$
$$= \arg\max_{\mathbf{w}} |\mathbf{w}^\top(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)|$$



where $\boldsymbol{\mu}_2$ and $\boldsymbol{\mu}_1$ are the means in *input space*.

- The best vector $\mathbf{w}$ is the one that aligns with $(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$, i.e. $\mathbf{w} = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)/\|\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1\|$.

Limitations of mean separation:

- class overlap in projected space
- if we rotate the projection a few degrees, then we can achieve a better classifier
- Making means distant may not be enough to induce class separability in projected space

Fisher Discriminant

R.A. Fisher (1890 - 1962)

Idea:

- In addition to maximizing the separation between class means in projected space, also consider to reduce the within-class variance.

data spread

$$\mu_1 = \frac{1}{|\mathcal{C}_1|}\sum_{k\in\mathcal{C}_1} z_k \qquad \mu_2 = \frac{1}{|\mathcal{C}_2|}\sum_{k\in\mathcal{C}_2} z_k$$

deviation from μ in projected space

$$s_1 = \sum_{k\in\mathcal{C}_1}(z_k - \mu_1)^2 \qquad s_2 = \sum_{k\in\mathcal{C}_2}(z_k - \mu_2)^2$$

- Maximizing distance between means while minimizing within-class variance can be formulated as:

$$\arg\max_{w} \frac{(\mu_2(\mathbf{w}) - \mu_1(\mathbf{w}))^2}{s_1(\mathbf{w}) + s_2(\mathbf{w})}$$

(between means max / within variance min)

Linear Discriminant Analysis

Goal: Find a (normal vector of a linear decision boundary) $\mathbf{w} \in \mathbb{R}^D$ that

Maximizes mean class difference

$$(\mathbf{w}^\top \mathbf{w}_o - \mathbf{w}^\top \mathbf{w}_\Delta)^2 = \mathbf{w}^\top \underbrace{(\mathbf{w}_o - \mathbf{w}_\Delta)(\mathbf{w}_o - \mathbf{w}_\Delta)^\top}_{S_B - \text{"between class scatter"}} \mathbf{w} \qquad (2)$$

cov-matrix

Minimizes variance in each class

$$\frac{1}{N_o}\sum_{i=1}^{N_o}\left(\mathbf{w}^\top(\mathbf{x}_{oi} - \mathbf{w}_o)\right)^2 + \frac{1}{N_\Delta}\sum_{j=1}^{N_\Delta}\left(\mathbf{w}^\top(\mathbf{x}_{\Delta j} - \mathbf{w}_\Delta)\right)^2$$
$$= \mathbf{w}^\top \underbrace{\left(\frac{1}{N_o}\sum_{i=1}^{N_o}(\mathbf{x}_{oi} - \mathbf{w}_o)(\mathbf{x}_{oi} - \mathbf{w}_o)^\top + \frac{1}{N_\Delta}\sum_{j=1}^{N_\Delta}(\mathbf{x}_{\Delta j} - \mathbf{w}_\Delta)(\mathbf{x}_{\Delta j} - \mathbf{w}_\Delta)^\top\right)}_{S_W - \text{"within class scatter"}} \mathbf{w}$$

$$\arg\max_{w} \frac{w^T S_B w}{w^T S_W w}$$

How to optimize Fisher criterion:

$$\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}$$

$$\frac{(w^T S_W w) S_B w - (w^T S_B w) S_W w}{(w^T S_W w)^2} = 0 \quad \text{cst.}$$

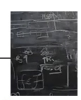
$$(w^T S_W w) S_B w = (w^T S_B w) S_W w$$
$$S_W w = S_B w \underbrace{\frac{w^T S_W w}{w^T S_B w}}_{\lambda}$$

$$S_W w = S_B w \lambda$$
$$w \propto S_W^{-1}(w_o - w_\Delta)$$

↳ inversed covariance

Fisher LDA

Algorithm



Computes: Normal vector w of decision hyperplane for binary classification

Input: Data $\{(x_1, y_1), \ldots, (x_N, y_N)\}, x_i \in \mathbb{R}^D, \; y_i \in \{-1, +1\}$,

1. Compute class mean vectors
   $w_{-1} = \sum_{i\in\mathcal{Y}_{-1}} x_i$
   $w_{+1} = \sum_{j\in\mathcal{Y}_{+1}} x_j$
2. Compute *between-class* covariance matrix
   $S_B = (w_{-1} - w_{+1})(w_{-1} - w_{+1})^\top$
3. Compute *within-class* covariance matrices
   $S_W = \sum_{i\in\mathcal{Y}_{-1}}(x_i - w_{-1})(x_i - w_{-1})^\top + \sum_{j\in\mathcal{Y}_{+1}}(x_j - w_{+1})(x_j - w_{+1})^\top$
4. Compute eigenvalue decomposition
   $S_B w = S_W w \lambda$ → generalized EV

Output: w

within class variance:

$$S_j(w) = \sum (z_k - \mu_j)^2 = \sum (w^T x_k - w^T \mu_j)^2$$
$$= \sum (w^T x_k - w^T \mu_j)(w^T x_k - w^T \mu_j)$$
$$= \sum (w^T x_k w^T x_k - w^T x_k w^T \mu_j - w^T \mu_j w^T x_k + w^T \mu_j w^T \mu_j)$$
$$= w^T(\sum x_k x_k^T) w - w^T(\sum x_k \mu_j^T) w - \sum w^T \mu_j w^T x_k + \sum w^T \mu_j \mu_j^T w$$
$$= w^T \underbrace{\sum (x_k - \mu_j)(x_k - \mu_j)^T}_{= S_j} w$$

Fisher Discriminant: $J(w) = \dfrac{(\mu_2(w) - \mu_1(w))^2}{S_1(w) + S_2(w)} \Rightarrow J(w) = \dfrac{w^T S_B w}{w^T S_W w}$

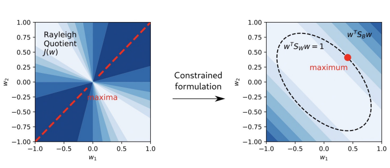
$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T$ Between class scatter

$S_W = S_1 + S_2$ Within class

"Rayleigh Coefficient"

- A solution that maximizes the Rayleigh quotient $J(\mathbf{w})$ can be obtained by first observing that $\forall_{\alpha \neq 0}: \; J(\alpha\mathbf{w}) = J(\mathbf{w})$ and searching for the particular solution for which the denominator is exactly one. This can be stated as the constrained optimization problem

$$\arg\max_{w} \mathbf{w}^\top S_B \mathbf{w} \quad \text{s.t.} \quad \mathbf{w}^\top S_W \mathbf{w} = 1$$



optimization problem: $\arg\max_{w} w^T S_B w$ s.t. $w^T S_W w = 1$ → $w^T S_W w - 1 = 0$ → $\lambda(w^T S_W w) - 1$

Step 1: Lagrange $\mathcal{L}(w) = w^T S_B w + \lambda(1 - w^T S_W w)$

Step 2: $\nabla \mathcal{L}(w, \lambda) = 0 \rightarrow S_B w = \lambda S_W w$ generalized EV-Problem

Further steps:

- If $\mathbf{S}_W$ is invertible, it can be restated as the standard eigenvalue problem

$$(\mathbf{S}_W^{-1}\mathbf{S}_B)\,\mathbf{w} = \lambda\mathbf{w}$$

- Expanding the term $\mathbf{S}_B$, we get:

$$\mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)\underbrace{(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \mathbf{w}}_{\text{scalar!}} = \lambda\mathbf{w}$$

- Therefore, one possible solution for $\mathbf{w}$ is given by

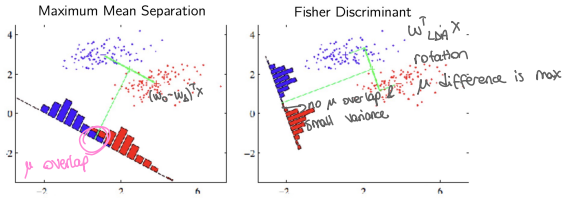
$$\boxed{\mathbf{w} = \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)}$$

(Reminder, in our original Rayleigh quotient formulation, $J(\alpha\mathbf{w}) = J(\mathbf{w})$, i.e. the optimum is defined up to a scaling factor.)

Mean Separation vs. Fisher Discriminant

(Perceptron) (LDA)

better separation



- Fisher Discriminant leads (in general) to better class separability, and therefore, better classification accuracy. → within class variance
- Fisher Discriminant requires inversion of a covariance matrix (only tractable for low-dimensional data). EV-Problem Σ^{-1} very hard to find

NCC

Bayes

- if data Gaussian distributed: μ_1, μ_2 & same cov Σ
- Bias also given by the analytics
- Discriminant:

$$w = \Sigma^{-1}(\mu_2 - \mu_1)$$

LDA

Gaussian with a cov structure

Fisher

- No generative assumption
- Classifier only derived from a criterion on mean and dispersion of data in projected space.
- Bias not provided in analysis but can be fitted in a second step.
- no cov-matrix per class

Discriminant:

$$w = S_w^{-1}(\mu_2 - \mu_1)$$

more general

Fisher Discriminant (Strengths/Limitations)

very hard to estimate cov-matrices

Strengths:

- Accurate when the means/covariances describe well the data, and close to optimal when the data is Gaussian of fixed covariance.
- The Fisher discriminant is given in closed form and is fast to compute.

Limitations:

- Although applicable to non-Gaussian distributions, the resulting decision boundary can become in that case strongly suboptimal.
- In particular, like principal component analysis, Fisher Discriminant is not robust to outliers.

Idea: ↳ because LDA based on cov-matrices ⇒ classification problem

- To overcome these limitations, we will discuss other learning algorithms that more specifically focus on modeling the decision boundary between the two classes.

Fisher LDA

- supervised, not robust to outliers because of cov-matrices
- finds a vector that maximizes the seperation between classes of the projected data

max: distance of projected means S_B

min: projected within class variance S_W

- correlations between features can affect classification accuracy
- DO LDA if we have labels → supervised (classification)
  DO PCA if we don't have labels → unsupervised (no labels)

it will turn the db a lot when we have an outlier because it'll think its Gaussian

Perceptron

- Perfectly seperates training data if linear seperable
- training simple & cheap iterations
- will not converge if we have overlap + not robust

Optimization of Perceptron

- Perceptron as the minimization of error function:

$$\mathcal{E}(\mathbf{w}, b) = \frac{1}{N}\sum_{k=1}^{N} \max(0, -z_k t_k)$$

$\mathcal{E}_k(\mathbf{w}, b)$

- The perceptron can be seen as the minimization of the error function

$$\mathcal{E}(\mathbf{w}, b) = \frac{1}{N}\sum_{k=1}^{N} \underbrace{\max(0, -z_k t_k)}_{\mathcal{E}_k(\mathbf{w}, b)}$$

→ Objective function of perceptron:
- measuring for each wrongly classified data point how far the data point is from the db, and penalizing accordingly
- → To optimize: · gradient descent on $\mathcal{E}(w,b)$ directly
  · adding momentum to gradient descent
- → Optimization also works for linearly non-seperable data

- *Proof:* Computing the gradient of $\mathcal{E}_k$ gives

$$\begin{aligned}\nabla_w \mathcal{E}_k(\mathbf{w}, b) &= 1_{-z_k t_k > 0} \cdot (-\mathbf{x}_k t_k) \\ &= 1_{y_k \neq t_k} \cdot (-\mathbf{x}_k t_k) \\ &= \begin{cases} 0 & y_k = t_k \\ -\mathbf{x}_k t_k & y_k \neq t_k \end{cases}\end{aligned}$$

And we observe that the update rule $\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \nabla_w \mathcal{E}_k(\mathbf{w}, b)$ is equivalent to that of Eq. (2)

- Similar result can be obtained for the bias.

The Perceptron Algorithm

- Consider our linear model

$$z_k = \mathbf{w}^\top \mathbf{x}_k + b \qquad y_k = \text{sign}(z_k)$$

and let t_k be 1 and -1 when the true class of $\mathbf{x}_k$ is ω_1 and ω_2 respectively.

Algorithm

- Iterate over all examples $k = 1 \ldots, N$ (multiple times).
  - If example $\mathbf{x}_k$ is correctly classified ($y_k = t_k$), continue.
  - If example $\mathbf{x}_k$ is wrongly classified ($y_k \neq t_k$), apply:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \cdot \mathbf{x}_k t_k \quad (2)$$

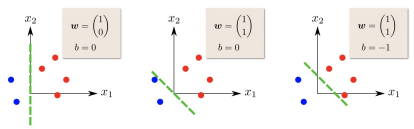
(new ← old)

$$b \leftarrow b + \eta \cdot t_k \quad (3)$$

where η is a learning rate.

- The algorithm stops once all examples are correctly classified.

Problem with Perceptron

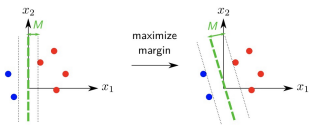


- All these solutions have an error $\mathcal{E}(\mathbf{w}, b) = 0$.
- Some solutions are obviously better than other, e.g. those where the decision boundary is separated from the data with a large margin.

hepsinde error = 0

ama bazılarında data large marginle ayrılmış → iyi bişey eğer data pointler cok yakınsa hyperplane bulmak uzun sürer

Large Margin Classifiers



Idea: Induce large margin M by redefining the optimization problem as:

$$\underbrace{\arg\min_{\mathbf{w},b,M}}_{(1)} \Big[\underbrace{\frac{1}{M}}_{(2)} + \frac{1}{N}\sum_{k=1}^{N} \underbrace{\max(0, M - (\mathbf{w}^\top \mathbf{x}_k + b)t_k)}_{(3)}\Big] \quad \text{s.t.} \quad \underbrace{\|\mathbf{w}\| = 1}_{(4)} \quad (4)$$

- (1) actively optimize the margin
- (2) apply a penalty if the margin is too small
- (3) apply a penalty if some points violate the margin and
- (4) constrain the projection so that M is interpretable as a margin.

- An equivalent (and more standard) formulation of the optimization problem is given by:

$$\boxed{\arg\min_{\mathbf{w},b} \frac{1}{N}\sum_{k=1}^{N} \max(0, 1 - (\mathbf{w}^\top \mathbf{x}_k + b)t_k) + \|\mathbf{w}\|^2} \quad (5)$$

(proof in next slide).

- Large Margin Classifiers are typically solved using (stochastic) gradient descent or quadratic programming.

$$\underset{w,b,\mu}{\operatorname{argmin}} \frac{1}{N}\sum_{k=1}^{N} \max(0, \mu - (w^T x_k + b)t_k) + \frac{1}{\mu} \quad \text{s.t.} \quad \|w\| = 1$$

$$= \underset{w,b,\mu}{\operatorname{argmin}} \frac{1}{N}\sum_{k=1}^{N} \max\left(0, \mu - (w^T x_k + b)t_k\right) + \frac{\|w\|^2}{\mu} \quad \text{s.t.} \quad \|w\| = 1$$

Dividing μ doesn't change argmin or constraint

$$= \underset{w,b,\mu}{\operatorname{argmin}} \frac{1}{N}\sum_{k=1}^{N} \max\left(0, 1 - \left(\frac{w^T}{\mu} x_k + \frac{b}{\mu}\right)t_k\right) + \frac{\|w\|^2}{\mu} \quad \text{s.t.} \quad \frac{\|w\|}{\mu} = \frac{1}{\mu}$$

$w \leftarrow w/\mu$, $b \leftarrow b/\mu$

$$= \underset{w,b,\mu}{\operatorname{argmin}} \frac{1}{N}\sum_{k=1}^{N} \max(0, 1 - (w^T x_k + b)t_k) + \|w\|^2 \quad \text{s.t.} \quad \|w\| = \frac{1}{\mu}$$

$$= \boxed{\underset{w,b}{\operatorname{argmin}} \frac{1}{N}\sum_{k=1}^{N} \max(0, 1 - (w^T x_k + b)t_k) + \|w\|^2}$$

Summary

→ Practice, train a classifier directly rather than learning the distributions

→ Max mean seperation and Fisher Discriminant: we only need to estimate the mean & cov of data without estimating full distributions.

→ Perceptron (& its large-margin extension) focuses on the decision boundary, which leads to higher classification accuracy on general tasks.

Recap: The Perceptron Algorithm

Consider the linear model $y = \boldsymbol{w}^\top \boldsymbol{x} + b$, where sign($y$) gives the classification decision. Let $t \in \{-1, +1\}$ be the binary class label associated to $\boldsymbol{x}$.

Algorithm

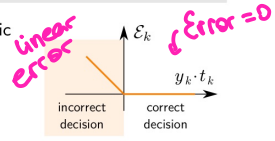
- Iterate over all data points $\{(\boldsymbol{x}_k, t_k)\,;\, k = 1\ldots, N\}$ (multiple times).
  - Compute $y_k = \boldsymbol{w}^\top \boldsymbol{x}_k + b$
  - If $\boldsymbol{x}_k$ is correctly classified (i.e. sign(y_k) = t_k), continue.
  - If $\boldsymbol{x}_k$ is wrongly classified (i.e. sign(y_k) ≠ t_k), apply:

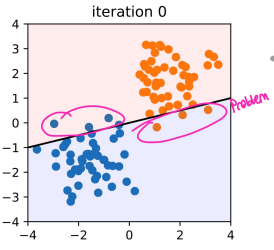
$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \gamma \cdot \boldsymbol{x}_k t_k \quad \text{and} \quad b \leftarrow b + \gamma \cdot t_k$$

- Stop once all examples are correctly classified.

The perceptron can also be seen as a stochastic gradient descent of the error function

$$\mathcal{E}(\boldsymbol{w}, b) = \frac{1}{N}\sum_{k=1}^{N} \underbrace{\max(0, -y_k t_k)}_{\mathcal{E}_k(\boldsymbol{w}, b)}$$





→ hyperplane will rotate until all data are correctly classified

the term $\gamma y_k t_k$ changes

→ change the hyperplane by rotating if data on the wrong side

→ adapting in high dimensions with γ

→ $y_k t_k$ will be negative if data misclassified
$t = -1$ if data wrong classified
$-y_k t_k = 1$ → if data misclassified

→ Error fct. takes the max, so the error deviation will be big.

→ If data correct classified: $t = 1$
$-y_k t_k = -1$
our $\max(0, -y_k t_k)$ will choose 0 so our Error function won't get anything.

Margin
- if margin between the data is large, then the hyperplane doesn't need to be that precise
- small margin: perceptron runs long time
- thinner the margin, longer the iterations

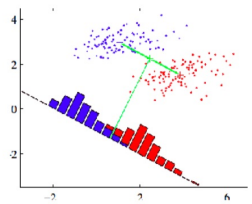
Recap: Mean Separation Criterion

take your data x_k apply projection on w (constraint as: $\|w\| = 1$ to only rotate and not scale the data goals: nice linear seperation

- We want to learn a projection of the data $z_k = \mathbf{w}^\top \mathbf{x}_k$ with $\|\mathbf{w}\| = 1$ such that the means of classes in projected space are as distant as possible.
- First, we compute the means in projected space for the two classes

$$\mu_1 = \frac{1}{N_1}\sum_{k \in \mathcal{C}_1} z_k \qquad \mu_2 = \frac{1}{N_2}\sum_{k \in \mathcal{C}_2} z_k$$

project data on hyperplane, build mean



- Then we find $\mathbf{w}$ that maximizes the difference of means, i.e. we express the means as a function of $\mathbf{w}$ and pose the optimization problem:

$$\arg\max_{\mathbf{w}} |\mu_2(\mathbf{w}) - \mu_1(\mathbf{w})| \quad \text{with} \quad \|\mathbf{w}\| = 1$$

closed form: classifier easy to produce

means as far away as possible
max the seperation
μ_k is a function of w (mean objectives on w functions)
μ_k also function of w

→ Perceptron may not work if:
① data not linearly seperable
② margin is very thin, takes a lot of time when fine tuning

→ If Perceptron converges, then data 100% linearly seperable

→ if there's a solution the perceptron will find it in a finite number of steps.

Fishes LDA: → finds a linear combination of features that seperates two or more classes

Fisher objective function: $J(w) = \frac{w^T S_B w}{w^T S_W w}$

max
S_B: Between class variance
S_W: within class variance

→ The max criterion assures that the classes are as seperated as possible in the projected space, to get easier and more accurate classification.

- PCA seeks directions that are efficient for representation
- Fishes LDA seeks directions that are efficient for discrimination.

- Goal of LDA: move the db around and try to find a line that the projected samples are well seperated.

Weight of Fisher LDA: $w = S_W^{-1}(m_1 - m_2)$

When is the Fisher LDA minimized? $\max_w \frac{w^T S_B w}{w^T S_W w}$

and its min if: $w^T S_B w \overset{!}{=} 0$

$S_B = (m_2 - m_1)(m_2 - m_1)^T \rightarrow w^T (m_2 - m_1)(m_2 - m_1)^T w \overset{!}{=} 0$ → so $w \perp (m_2 - m_1)$

$\langle w, (m_2 - m_1) \rangle \overset{!}{=} 0 \rightarrow [w_1 \; w_2] \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} \overset{!}{=} 0 \rightarrow w_1 m_1 + w_2 m_2 \overset{!}{=} 0$

Lecture 4: PCA

→ Curse of dimensionality
→ PCA
→ Lagrange

Goal: Reduce the dimension to the leading variance elements.
- min noise & max. variance
- Lagrange multipliers to find PCA solutions
- PCA solutions ≙ EV's of the data covariance

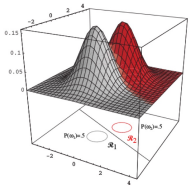
→ Dimensionality Reduction : Curse of Dimensionality

- curse of dimensionality increases the models complexity in high-dimensional spaces, making it possibly overfitting.

→ if $d > N$: ill-defined Many dimensions but not enough training data
→ if $d \leq N$: ill-conditioned / numerically unstable

Example of ill-conditioning: Bayes Classifier with estimated Cov

- Assume our data is generated for each class ω_j according to the multivariate Gaussian distribution $p(\mathbf{x}|\omega_j) = \mathcal{N}(\boldsymbol{\mu}_j, \Sigma)$ and with class priors $P(\omega_j)$. The Bayes optimal classifier is derived as



$$\begin{aligned}
&\arg\max_j \{P(\omega_j|\mathbf{x})\} \\
&= \arg\max_j \{\log p(\mathbf{x}|\omega_j) + \log P(\omega_j)\} \\
&= \arg\max_j \{\mathbf{x}^\top \Sigma^{-1} \boldsymbol{\mu}_j - \frac{1}{2}\boldsymbol{\mu}_j \Sigma^{-1} \boldsymbol{\mu}_j + \log P(\omega_j)\}
\end{aligned}$$

Example of estimator:

- Maximum likelihood estimator:

$$\hat{\Sigma} = \frac{1}{N}\bar{X}\bar{X}^\top$$

where $\bar{X}$ is a matrix of size $d \times N$ containing the centered data.

Problem:

- The matrix $\hat{\Sigma}^{-1}$ used in the classifier only exists when $d \leq N$ and the matrix $\bar{X}$ has full rank.
- The matrix $\hat{\Sigma}^{-1}$ used in the classifier is typically accurate only when $d \ll N$. need to have lot of data

Solutions:

- Reduce the dimensionality of the data **(today)**
- Regularize the model (later)

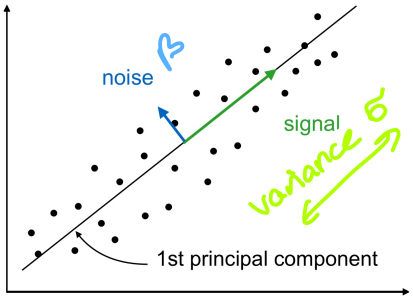
→ Because we don't know true Σ, we estimate $\hat{\Sigma}$
  Goal: a good $\hat{\Sigma}$ estimator
→ inverted covarince Σ^{-1} is expensive to compute
→ in practice: we don't know μ or Σ. We just know the datapoints
→ Reduce d and project data there

if: Dimension ↑ + space volume ↑ available data becomes sparse

- The amount of data needed for a reliable result (e.g. no overfitting) often grows exponentially with the dimensionality.
- Here again, this can be addressed by reducing the dimensionality of the data **(today)** or regularizing (later).

Principal Component Analysis

- PCA maps the data from a high dimension to a low dimension



Which line fits data best?
The line **w** that minimizes the noise and maximizes the signal

min β
max σ

PCA as noise minimization

- Given a dataset $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$, find a one-dimensional subspace, so that the data projected on that subspace has **minimum distance** to the original data.
- The reconstruction model is given by:

$$\hat{\mathbf{x}} = \mathbf{w}\mathbf{w}^\top \mathbf{x}$$

estimate from subspace

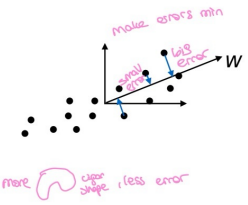
with $\|\mathbf{w}\| = 1$.

- Minimizing the noise (i.e. reconstruction error) can be written as:

$$\arg\min_{\mathbf{w}} \left[\frac{1}{N}\sum_{k=1}^{N} \|\mathbf{x} - \underbrace{\mathbf{w}\mathbf{w}^\top \mathbf{x}}_{\hat{\mathbf{x}}}\|^2\right]$$

error distances

original x − subspace x distance



0 loss if: entire d is projected
1 reconstruction error if: d−1 is projected

PCA as signal maximisation

- Given a dataset $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$. Find a one-dimensional subspace, so that the data projected has **maximum variance**.
- The projection model is given by:

$$h = \mathbf{w}^\top \mathbf{x}$$

with $\|\mathbf{w}\| = 1$.

- Maximizing the signal (i.e. variance of projected data) can be written as:

$$\arg\max_{\mathbf{w}} \left[\frac{1}{N}\sum_{k=1}^{N} (\mathbf{w}^\top \mathbf{x}_k - \mathbb{E}[\mathbf{w}^\top \mathbf{x}])^2\right]$$

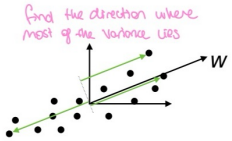
and assuming centered data, i.e. $\mathbb{E}[\mathbf{x}] = \mathbf{0}$, it simplifies to:

Variance

$$\arg\max_{\mathbf{w}} \left[\frac{1}{N}\sum_{k=1}^{N} \underbrace{(\mathbf{w}^\top \mathbf{x}_k)^2}_{\text{scalar}}\right]$$

how far we are apart?

find the direction where most of the variance lies



$ww^\top$ gives the noise and its norm is 1. $\rightarrow \|w\| = 1$

From Min Noise to Max Signal

$$\arg\min_{\mathbf{w}} \left[\frac{1}{N}\sum_{k=1}^{N} \|\mathbf{x}_k - \mathbf{w}\mathbf{w}^\top \mathbf{x}_k\|^2\right]$$

$$= \arg\min_{\mathbf{w}} \left[\frac{1}{N}\sum_{k=1}^{N} (\mathbf{x}_k - \mathbf{w}\mathbf{w}^\top \mathbf{x}_k)^\top (\mathbf{x}_k - \mathbf{w}\mathbf{w}^\top \mathbf{x}_k)\right]$$

$$= \arg\min_{\mathbf{w}} \left[\frac{1}{N}\sum_{k=1}^{N} \underbrace{\mathbf{x}_k^\top \mathbf{x}_k}_{\text{cst.}} - 2\mathbf{x}_k^\top \mathbf{w}\mathbf{w}^\top \mathbf{x}_k + (\mathbf{w}\mathbf{w}^\top \mathbf{x}_k)^\top (\mathbf{w}\mathbf{w}^\top \mathbf{x}_k)\right]$$

cst. cuz data's norm stays the same

$$= \arg\min_{\mathbf{w}} \left[\frac{1}{N}\sum_{k=1}^{N} -2(\mathbf{x}_k^\top \mathbf{w})^2 + \mathbf{x}_k^\top \mathbf{w}\underbrace{\mathbf{w}^\top \mathbf{w}}_{\text{norm}=1}\mathbf{w}^\top \mathbf{x}_k\right]$$

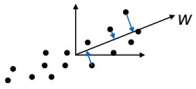
projection of every data

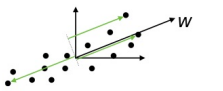
$$= \arg\min_{\mathbf{w}} \left[\frac{1}{N}\sum_{k=1}^{N} -2(\mathbf{x}_k^\top \mathbf{w})^2 + (\mathbf{x}_k^\top \mathbf{w})^2\right]$$

$\operatorname{argmin}(\ominus) = \operatorname{argmax}(\oplus)$

$$= \arg\max_{\mathbf{w}} \left[\frac{1}{N}\sum_{k=1}^{N} (\mathbf{x}_k^\top \mathbf{w})^2\right]$$

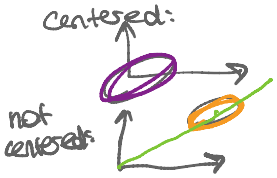
max variance





⚠ Data needs to be centered

→ Projection of PCA won't be correct if we don't center:



- Gaussian
- Linear
- Centered

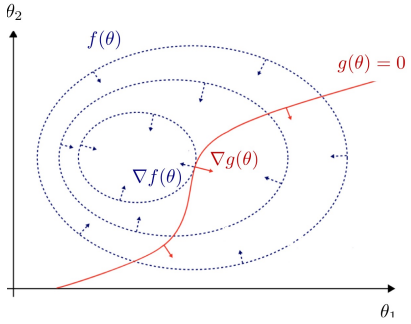
→ PCA is not robust to outliers so remove outliers before doing PCA.

How to compute Principal Components? Lagrange

Lagrange general framework:

① $\mathcal{L}(\theta, \lambda) = f(\theta) + \lambda g(\theta)$

② $\nabla \mathcal{L}(\theta, \lambda) = 0$



Lagrange 2D Example:

Consider the constrained optimization problem

$$\boxed{\arg\max_{\theta}[1-(\theta_1^2+\theta_2^2)] \quad \text{s.t.} \quad \theta_1+\theta_2=1}$$

- **Step 1:** Build the Lagrangian

$$\mathcal{L}(\theta,\lambda)=1-(\theta_1^2+\theta_2^2)+\lambda\cdot(\theta_1+\theta_2-1)$$

- **Step 2:** Set gradient of Lagrangian to zero:

$$\begin{aligned}\nabla_{\theta_1}\mathcal{L}(\theta,\lambda)=0 &\quad\Rightarrow\quad 2\theta_1=\lambda\\ \nabla_{\theta_2}\mathcal{L}(\theta,\lambda)=0 &\quad\Rightarrow\quad 2\theta_2=\lambda\\ \nabla_{\lambda}\mathcal{L}(\theta,\lambda)=0 &\quad\Rightarrow\quad \theta_1+\theta_2=1\end{aligned}$$

which gives us the solution:

$$\theta=(0.5,0.5)$$

→ The 1st PC is the direction that the data's projection has the highest variance.

Solving EV-Problem: $\Sigma u=\lambda u$ gives the highest EV λ will give the direction.

Magnitude of λ represents the variance of the data projected onto the corresponding EV u so choosing the highest λ maximizes this variance.

Solution PCA

projection problem → $\arg\max_{w}\left[\frac{1}{N}\sum_{k=1}^{N}(x_k^\top w)^2\right]$ s.t. $\|w\|=1$

can be rewritten as:

$$=\arg\max_{w}\left[\frac{1}{N}\sum_{k=1}^{N}(w^\top x_k)(x_k^\top w)\right] \quad \text{s.t.} \quad \|w\|^2=1$$

$$=\arg\max_{w}\Big[w^\top\underbrace{\Big(\frac{1}{N}\sum_{k=1}^{N}x_k x_k^\top\Big)}_{\hat{\Sigma}}w\Big] \quad \text{s.t.} \quad \|w\|^2=1$$

$w^\top\Sigma w$

where $\hat{\Sigma}$ is the covariance matrix (recall: the data is assumed to be centered). The covariance matrix does not depend on $\mathbf{w}$ and can be precomputed.

Starting from our rewritten PCA optimization problem

$$\arg\max_{w}[w^\top\hat{\Sigma}w] \quad \text{s.t.} \quad \|w\|^2=1$$

we look for a solution by applying the method of Lagrange multipliers.

- **Step 1:** Build the Lagrangian

$$\mathcal{L}(\mathbf{w},\lambda)=\mathbf{w}^\top\hat{\Sigma}\mathbf{w}+\lambda\cdot(1-\|\mathbf{w}\|^2)$$

- **Step 2:** Set gradient of Lagrangian to zero:

$$\begin{aligned}\nabla_{w}\mathcal{L}(\mathbf{w},\lambda)=\mathbf{0} &\quad\Rightarrow\quad \hat{\Sigma}\mathbf{w}=\lambda\mathbf{w}\\ \nabla_{\lambda}\mathcal{L}(\mathbf{w},\lambda)=0 &\quad\Rightarrow\quad \|\mathbf{w}\|^2=1\end{aligned}$$

EV-Problem

! PCA solution is an eigenvector of $\hat{\Sigma}$

To find PC's:
- estimate Cov matrices
- EV decomposition of cov-matrix
- 1st EV = PCA solution

if you want more than 1 component:

1) Compute the 1st component
2) remove from data
↻ recompute

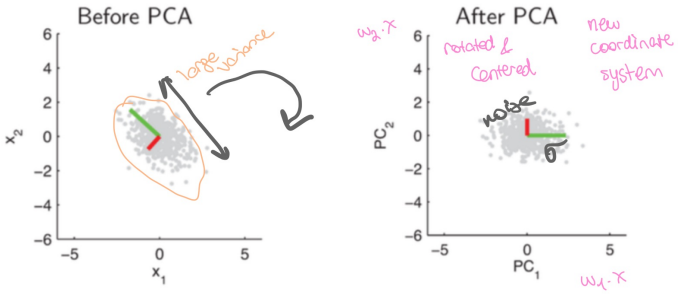
Finding multiple PCA components

for $i=1$ to h do
  $\mathbf{w}_i \leftarrow \mathrm{PC}_1(\bar{X})$ 1)
  $\bar{X}\leftarrow\bar{X}-\mathbf{w}_i\mathbf{w}_i^\top\bar{X}$ 2)
end for

→ Resulting components $w_1 \dots w_k$ are the eigenvectors of $\hat{\Sigma}$

→ it's enough to only find the leading EV's of $\hat{\Sigma}$.

→ Cov Matrix ≙ Size of EV's



- PCA rotates data into new coordinate system with the directions of largest variance being the new coordinate axes.

Stable PCA Computation via SVD

Singular Value Decomposition (SVD)

A singular value decomposition factorizes a matrix M as $U\Lambda V = M$ where

- U contains the eigenvectors of $MM^\top$
- V contains the eigenvectors of $M^\top M$
- The square roots of the eigenvalues of $MM^\top$ are on the diagonal of Λ.

Observation:

- Setting $M = \frac{1}{\sqrt{N}}\bar{X}$ and computing SVD results in
  - a matrix $U = (\mathbf{w}_1|\ldots|\mathbf{w}_d)$ that contains the eigenvectors of $\widehat{\Sigma}$, i.e. the principal components, and
  - a matrix $\Lambda = \text{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_d})$ (but applying the square root does not change the ordering of eigenvalues).
- SVD is computationally faster and more stable than covariance computation + eigenvalue decomposition. → Vorteil

PCA Computation via SVD, limitations

- The SVD has computational complexity of $\mathcal{O}(\min(N^2 d, d^2 N))$.
- When $d \approx N$, the complexity is roughly as bad as the that of computing the eigenvectors of $\widehat{\Sigma}$, that is, $\mathcal{O}(d^3)$.
- This makes the computation of principal components using SVD prohibitive for large datasets.

Note:

- In practice, we often need **only the first few principal components**, whereas SVD computes all of them.

Question:

- Can we design a faster procedure that only extracts the first few principal components? Power iteration

Power Iteration

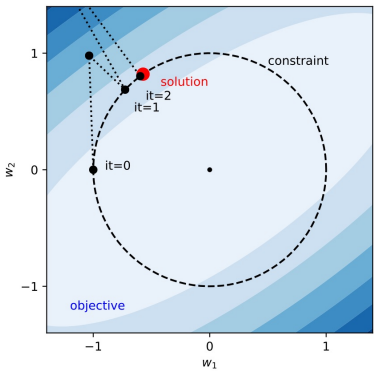
Find the first component by applying the *iteration*

$$\mathbf{v}^{(t)} \leftarrow \widehat{\Sigma}\mathbf{w}^{(t-1)}$$
$$\mathbf{w}^{(t)} \leftarrow \frac{\mathbf{v}^{(t)}}{\|\mathbf{v}^{(t)}\|}$$

T times, starting from a random (or any) unit-norm vector $\mathbf{w}^{(0)}$.

Questions:

- Does it always converge? **yes**
- How fast it converges? **exponentially fast**



→ POWIT only gives us the leading EV
→ If we want more PCA components, we need to compute them iteratively

Naive approach (cf. slide 17): Recompute covariance at each step

for $i = 1$ **to** h **do**
  $\widehat{\Sigma} = \frac{1}{N}\bar{X}\bar{X}^\top$
  $\mathbf{w}_i \leftarrow \text{POWIT}(\widehat{\Sigma})$
  $\bar{X} \leftarrow \bar{X} - \mathbf{w}_i\mathbf{w}_i^\top\bar{X}$
end for

Better approach: Work directly in covariance space more stable

$\widehat{\Sigma} = \frac{1}{N}\bar{X}\bar{X}^\top$
for $i = 1$ **to** h **do**
  $\mathbf{w}_i \leftarrow \text{POWIT}(\widehat{\Sigma})$
  $\widehat{\Sigma} \leftarrow \widehat{\Sigma} - \mathbf{w}_i\mathbf{w}_i^\top\widehat{\Sigma}$
end for

- starts with a random vector $w^{(0)} \in \mathbb{R}^d$ and iteratively applies the parameter update:

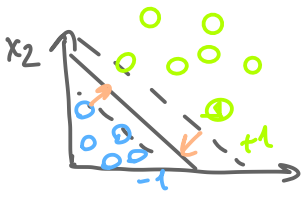
$$w^{(t+1)} = \frac{S_w^{(t)}}{\|S_w^{(t)}\|}$$

until some convergence is met

Lecture 5: SVM (Support Vector Machines)

Supervised classification

- hyperplane is d-1 dimensions where n is our features $(x_1, x_2 \dots, x_d)$ → $\langle w, x \rangle + b = 0$
- support vectors are the closest ones to our hyperplane
- if we remove a SV, our hyperplane may shift
- margin: the distance our hyperplane is to the closest SV
- choose the hyperplane that: - seperates data the most
  - largest margin



- distance between hyperplane and margin: $\frac{|w^T x + b|}{\|w\|} = \frac{|\pm 1|}{\|w\|} = \frac{1}{\|w\|}$
- max total margin to get good data seperation
  to max margin you need to min $\|w\|$
- SVMs can be used for linearly seperable and non-linearly seperable data.
- With appropriate nonlinear mapping $\phi(\cdot)$ to a sufficiently high dimension, data can always be seperated by a hyperplane.

→ Linear Soft Margin SVM

primal: $\min_{w,b,\zeta} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \zeta_i$ s.t. $\forall_{i=1}^{N}: y_i \cdot (w^T \phi(x_i) + b) \geq 1 - \zeta_i$ and $\zeta_i \geq 0$

$\|\cdot\|$: eudidian norm
ϕ: feature map

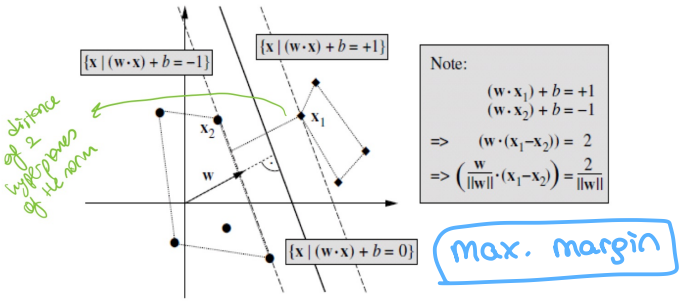
$w \in \mathbb{R}^d$
$b \in \mathbb{R}$

$x_i \in \mathbb{R}^d$
$y_i \in \{-1, 1\}$: labeled data points

Once the soft margin SVM is learned, prediction of any data point $x \in \mathbb{R}^d$ is given by the function:

$$f(x) = \text{sign}(w^T \phi(x) + b)$$

→ Linear Hyperplane classifier



max. margin

- hyperplane $y = \text{sgn}(w \cdot x + b)$ in canonical form if $\min_{x_i \in X} |(w \cdot x_i) + b| = 1$., i.e. scaling freedom removed.
- larger margin $\sim 1/\|w\|$ is giving better generalization → LMC!

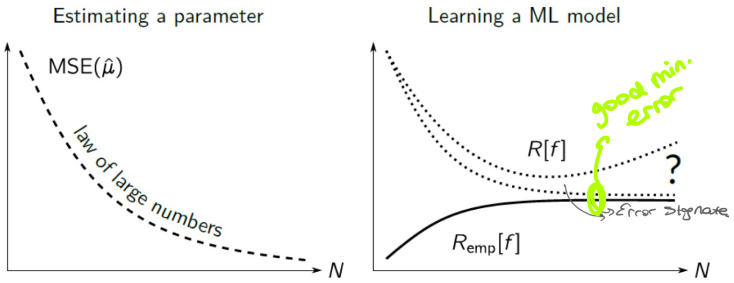
$\max \|w\|^2 = \min \frac{1}{\|w\|^2}$

Because we want new points to be correctly classified.

⇒ larger margin, smaller the norm, better generalisation

Optimal seperating hyperplane is the one that has the largest margin ≙ smallest norm $\|w\|$

- the first datapoints we hit has norm 1
- margin for canonical hyperplanes is: $\frac{1}{\|w\|}$
- If we manage to seperate training data with a large margin, then we'll do well on test set.
- make the norm $\|w\|$ small to make the margin large



Questions:

- Given a dataset of N points, how well will my model generalize?
- How quickly $R[f]$ and $R_{emp}[f]$ converge with N? $1/N$ OR $1/\sqrt{N}$
- Does it actually converge? there's a limit

- we don't want to overparametrize because it might lead to overfit
- we don't know the prob. distribution of our data, so we don't have the true risk R[f] instead, we have an empirical risk minimization
- we want a small generalisation (test) error

$R[f] \leq R_{emp} + \text{cst.}$ upper bound R[f]

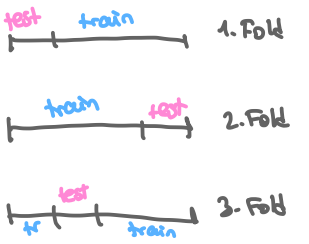
if we have our ERM, then we add the VC constant and say that our true risk must be $\leq$ to that.

→ To predict R[f]

- Cross validation

dataset into train / test

k-Fold CV:



Test error
0,38
0,43
0,42
N_{test}
$\approx R[f]$

- Error Bound upper bound to R[f]

Law of large Numbers: $R_{emp} \to R_F$ as $N \to \infty$

Empirical Risk Minimisation (ERM)

Empirical risk will converge to true risk, but it won't be the same, we need uniform convergence

super complex function class: can fit anything, emp. risk will be zero, because the complexity of the function is enough to predict it with 0 error. VC-Dimension however is very large

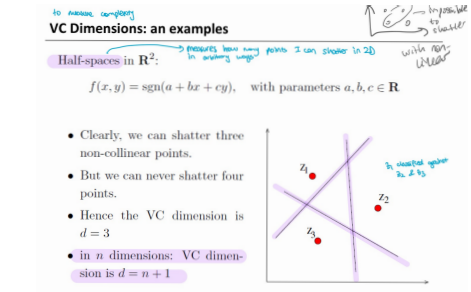
- d = VC dimension for class complexity
- try to minimize the VC-Dimension to get low complexity

VC-Dimension:

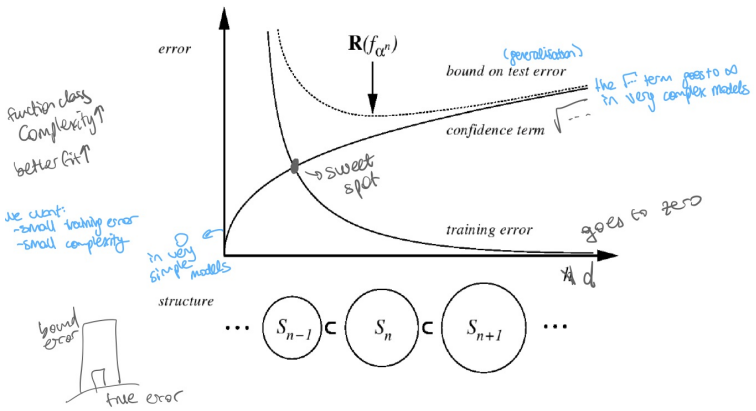
$$R[f] \leq R_{emp}[f] + \sqrt{\frac{d\left(\log\frac{2N}{d}+1\right) - \log(\eta/4)}{N}}.$$

→ if the function is too simple, VC dimension will be very low but Remp will be very large → bad upperbound

→ Use VC to measure complexity



Structural Risk Minimization: the picture



Learning f requires small training error *and* small complexity of the set $\{f_\alpha\}$.

Dimension ↑ :- Training error ↓
- Bound on test error ↑

find the sweet spot

VC Theory

VC Theory applied to hyperplane classifiers

- **Theorem (Vapnik 95)**: For hyperplanes in canonical form VC–dimension satisfying

$$d \leq \min\{[R^2\|\mathbf{w}\|^2] + 1, n + 1\}.$$

(worst case)

Here, R is the radius of the smallest sphere containing data. Use d in SRM bound

$$R[f] \leq R_{emp}[f] + \sqrt{\frac{d\left(\log\frac{2N}{d} + 1\right) - \log(\eta/4)}{N}}.$$

- maximal margin = minimum $\|\mathbf{w}\|^2$ → good generalization, i.e. low risk, i.e. optimize $\min \|\mathbf{w}\|^2$
- independent of the dimensionality of the space!

→ The complexity of a linear classifier is directly related to the dimensionality of the input space. Adding more features (increasing dimensionality) without sufficient data may however lead to overfitting.

· VC Dimension of a large margin ≤ " small margin

→ If you can find $\Phi(\cdot)$ in higher dimensions that has:
- small training error
- low complexity,

then you can guarantee good generalization

complexity ↗ dimensionality

Three scenarios: regression, classification & density estimation. Learn f from examples

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \in \mathbf{R}^N \times \mathbf{R}^M \text{ or } \{\pm 1\}, \quad \text{generated from } P(\mathbf{x}, y),$$

such that expected number of errors on test set (drawn from $P(\mathbf{x}, y)$),

$$R[f] = \int \frac{1}{2}|f(\mathbf{x}) - y)|^2 \, dP(\mathbf{x}, y),$$

is minimal *(Risk Minimization (RM))*.

Problem: P is unknown. ⟶ need an *induction principle*.

Empirical risk minimization (ERM): replace the average over $P(\mathbf{x}, y)$ by an average over the training sample, i.e. minimize the training error

$$R_{emp}[f] = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}|f(\mathbf{x}_i) - y_i|^2$$

→ With SRM we can upperbound the test error by the empirical error

→ If hypothesis very complex, then we'll overfit.

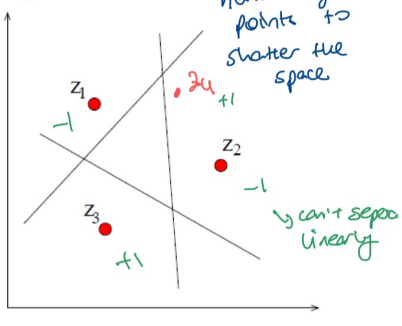
→ SVM's measure the capacity of the algorithm by their VC-Dimensions

VC dimension: example

Half-spaces in $\mathbf{R}^2$: (classify 3 points correctly if VC=3 in 2D; for 2 class problems: VC dim = 3 = n+1)

$$f(x, y) = \text{sgn}(a + bx + cy), \quad \text{with parameters } a, b, c \in \mathbf{R}$$

- Clearly, we can shatter three non-collinear points.
- But we can never shatter four points.
- Hence the VC dimension is $d = 3$
- in n dimensions: VC dimension is $d = n + 1$ (linear)



→ Support Vector Approach

preprocesses the data to a higher feature space

$$\begin{aligned} \Phi : \mathbf{R}^N &\to F \\ \mathbf{x} &\mapsto \Phi(\mathbf{x}) \\ \text{where } N &\ll \dim(F). \end{aligned}$$

to get data $(\Phi(\mathbf{x}_1), y_1), \ldots, (\Phi(\mathbf{x}_N), y_N) \in F \times \mathbf{R}^M$ or $\{\pm 1\}$.

Learn $\tilde{f}$ to construct $f = \tilde{f} \circ \Phi$

- SV-Learning: (in some cases) **simpler:** find $\tilde{f}$'s that make it simpler

If Φ is chosen such that $\{\tilde{f}\}$ allows small training error (= emp. error) *and* has low complexity, then we can guarantee good generalization.

⇒ The *complexity* matters, not the *dimensionality* of the space.

→ Learning Theory

minimal Risk Minimization: (RM)

to get a low generalisation error

Three scenarios: regression, classification & density estimation.
Learn f from examples

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \in \mathbb{R}^n \times \mathbb{R}^m$ or $\{\pm 1\}$, generated from $P(\mathbf{x}, y)$,

such that expected number of errors on test set (drawn from $P(\mathbf{x}, y)$),

Risk: $$R[f] = \int \frac{1}{2} |f(\mathbf{x}) - y)|^2 \, dP(\mathbf{x}, y),$$

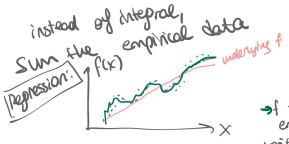
(vector; univariate … ; label; regression …)

is minimal *(Risk Minimization (RM))*.

→ Problem: we don't know the prob. distribution so we look at the training error instead of generalisation error.

empirical Risk Minimisation (ERM)

Empirical risk minimization (ERM): replace the average over $P(\mathbf{x}, y)$ by an average over the training sample, i.e. minimize the training error



$$R_{emp}[f] = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} |f(\mathbf{x}_i) - y_i|^2$$

→ if that min the emp. risk won't consider the noise too ⇒ OVERFIT

There's an unknown process:
$y = f(x) + \underbrace{\xi}_{\text{noise}}$

depending on the noise, we may change the norm

SVM Primal

The classifier with largest margin between the positive and negative data points can be obtained by solving the minimization problem:

$$\min_{\boldsymbol{w},b} \ \frac{1}{2}\|\boldsymbol{w}\|^2$$

→ determines how flat the decision function/spaced is

subject to the constraints

$$\forall_{i=1}^{N}: \ y_i \cdot (\boldsymbol{w}^\top \phi(\boldsymbol{x}_i) + b) \geq 1.$$

hard margin SVM

Once the parameters have been optimized, the decision function is given by

$$\text{sign}(\boldsymbol{w}^\top \phi(\boldsymbol{x}) + b)$$

and it can be applied to classify new data.

How to optimize the decision? ⇒ Slater's condition

Lagrange had (=), but now we have (≥)

Slater's Condition

data linearly seperable

Consider the optimization problem

$$\min_{\theta} f(\theta)$$

with f convex, subject to the inequality constraints

$$\forall_{i=1}^{m}: \ g_i(\theta) \leq 0$$

with g_i also convex. If there exists a point θ in the input domain that satisfies all constraints with *strict* inequalities, then the solution of the optimization problem is also given by its *Lagrange dual* formulation, the constrained optimization problem:

$$\max_{\boldsymbol{\alpha}} \left\{ \min_{\theta} \left\{ f(\theta) + \sum_{i=1}^{m} \alpha_i g_i(\theta) \right\} \right\} \quad \forall_{i=1}^{m} \ \alpha_i \geq 0, \qquad (1)$$

where the inner minimization problem is typically solved analytically.

→ 1st min the Lagrange, then max over Lagrange variable α where $\alpha_i \geq 0$

→ when g_i positive: constraint is not satisfied $g_i(\theta) \leq 0$

inner optimization

so if $g_i(\theta) < 0$ make very large with α and make it go to ∞

→ Unlike Lagrange, this is not a closed form solution, but a on α based dual variable solution.

How do we apply Slater to SVM?

SVM: from the Primal to the Dual

hard-margin

Slater's condition for SVM:

$$\exists \ \boldsymbol{w}, b \ \text{ s.t. } \ \forall_{i=1}^{N}: \ y_i \cdot (\boldsymbol{w}^\top \phi(\boldsymbol{x}_i) + b) > 1$$

In other words, the two classes must be *linearly separable*. When ϕ can be computed explicitly, this can be tested by running e.g. the perceptron algorithm.

→ if the perceptron of the kernel converges to 0, then classes linearly seperable

→ or use soft margin because it doesn't request linear seperability.

If the Slater's conditions are verified, we inject the SVM objective in Eq. (1):

$$\max_{\boldsymbol{\alpha}} \left\{ \min_{\boldsymbol{w},b} \left\{ \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \alpha_i \cdot (1 - y_i \cdot (\boldsymbol{w}^\top \phi(\boldsymbol{x}_i) + b)) \right\} \right\} \ \text{ s.t. } \ \boldsymbol{a} \succeq 0,$$

$f(w,b) = f(\theta)$; $g_i(w,b) \leq 0$

Roadmap for finding the dual in hard margin SVM:
- you'll find an explicit solution for w, but not for α.
- for α, you'll get a constraint that relates to dual variables.
- reinject w, b and get α

Predicting with the Dual Solution

Recall: we predict based on output of:

$$\boldsymbol{w}^\top \phi(\boldsymbol{x}) + b$$

Question: how do we find the bias b ?

- ANSWER: We recall that because the model has maximized the margin between the two classes, the nearest point to the decision boundary for each class is exactly on the margin, i.e.

$$\min_{i} \{y_i \cdot (\boldsymbol{w}^\top \phi(\boldsymbol{x}_i) + b)\} = 1$$

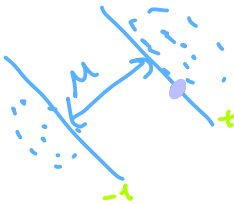
After some manipulations, we get: $b = 1 - \min_{i|y_i=1} \{\boldsymbol{w}^\top \phi(\boldsymbol{x}_i) + b\}$.

- you get w by directly minimizing the inner optimisation problem:

$$\max_{\alpha} \left\{ \min_{w,b} \left\{ \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \alpha_i \cdot (1 - y_i \cdot (\boldsymbol{w}^\top \phi(\boldsymbol{x}_i) + b)) \right\} \right\} \quad \text{s.t.} \quad \boldsymbol{a} \succeq 0,$$

$f(w,b)$ $= f(\theta)$; $g_i(w,b) \leq 0$

$w = \alpha_i y_i \phi(x_i)$



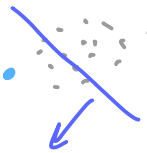
- SVM will try to find the max margin between these 2 classes
- $w^T \phi(x) + b$ is ± 1 → linear model
- if we take +1 as point we can say that the smallest output must be = 1.

$$\min_{i} \{y_i \cdot (\boldsymbol{w}^\top \phi(\boldsymbol{x}_i) + b)\} = 1$$ $y_i = 1$

→ min of my output is 1.

To find bias: 1 − min ↑

Difference hard- soft margin SVM:



- there might be outliers in my class and the margin will not consider with lowest output $w^T \phi(x) + b$, but it will be sth. else
- hard margin wants the output to be 1, but not the case in soft margin

Question: how do we find the bias b for the soft-margin SVM?

- ANSWER: Bias can be recovered from the dual using another set of conditions, the KKT conditions.

KKT Conditions (simplified)

Consider the optimization problem $\min f(\theta)$. Subject to the inequality constraints $\forall_{i=1}^m : g_i(\theta) \leq 0$. For this problem, the Karush-Kuhn-Tucker (KKT) conditions are:

$\nabla f(\theta) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\theta) = 0$	(stationarity) → gradient of Lagrangian
$\forall_{i=1}^m : g_i(\theta) \leq 0$	(primal feasibility) } already given
$\forall_{i=1}^m : \lambda_i \geq 0$	(dual feasibility) }
$\lambda_i g_i(\theta) = 0$	(complementary slackness) → useful to find bias

- If **Slater's conditions** are satisfied, then the **KKT conditions** are satisfied at the optimum.
- The bias of the primal can be recovered from the 'complementary slackness' equation.

$\lambda_{\text{Lagrange}} \cdot \text{constraint} \overset{!}{=} 0$ soft SVM

$w^T \phi(x) + b - 1$ is our constraint

$\alpha_i \cdot (1 - w^T \phi(x) - b) = 0$

one of these is 0

if $\alpha_i = 0$: the $(1 - w^T \phi(x) - b)$ can be anything

if $\alpha_i \neq 0$: this is zero

Lecture 6: Kernels

Exercise Sheet 6

A kernel function $k\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ must satisfy the *Mercer's condition*, which verifies that for any sequence of data points $x_1, \ldots, x_n \in \mathbb{R}^d$ and coefficients $c_1, \ldots, c_n \in \mathbb{R}$ the inequality

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i\, c_j\, k(x_i, x_j) \geq 0$$

is satisfied. If it is the case, the kernel is called a *Mercer kernel*.

Conversely, the *representer theorem* states that if k is a Mercer kernel on $\mathbb{R}^d$, then there exists a Hilbert space (i.e., a finite or infinite dimensional $\mathbb{R}$-vector space with norm and scalar product) $\mathcal{F}$, the so-called feature space, and a continuous map $\varphi : \mathbb{R}^d \to \mathcal{F}$, such that

$$k(x, x') = \langle\, \varphi(x)\,,\, \varphi(x')\,\rangle_{\mathcal{F}} \quad \text{for all } x, x' \in \mathbb{R}^d.$$

Kernel Trick:
→ Take a polynomial basis to increase the dimension in order to map the data to a higher dimension so that you get a simpler model.

→ Kernel Trick

2 polynomials

$$\begin{aligned}(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})) &= (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)(y_1^2, \sqrt{2}\, y_1 y_2, y_2^2)^\top \\ &= (\mathbf{x} \cdot \mathbf{y})^2 \quad \text{rewrite as a scalar-product in the original space} \\ &=: k(\mathbf{x}, \mathbf{y}) \quad \text{kernel}\end{aligned}$$

- Scalar product in (**high dimensional**) feature space can be computed in $\mathbf{R}^2$!
- works only for Mercer Kernels $k(\mathbf{x}, \mathbf{y})$

Scalar product to not compute in hi dimension space

$\vec{w} \cdot \vec{x}$ Scalar product based linear learning model

- take 2 polynomials
- compute their scalar product & map into a function
- Map data to higher dimension to solve with a simpler model.

of kernel ≈ regularizer : If you want your function to have a certain property, then map it itself to a kernel & use as a SV.

→ Mercer Kernels

[Mercer] If k is a continuous kernel of a positive integral operator on $L_2(\mathcal{D})$ (where $\mathcal{D}$ is some compact space),

if this is a positive operator, then we can always use kernel trick

$$\int f(\mathbf{x}) k(\mathbf{x}, \mathbf{y}) f(\mathbf{y})\, d\mathbf{x}\, d\mathbf{y} \geq 0, \quad \text{for} \quad f \neq 0$$

it can be expanded as

high-dim scalar product

$$k(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N_F} \lambda_i \psi_i(\mathbf{x}) \psi_i(\mathbf{y})$$

with $\lambda_i > 0$, and $N_F \in \mathbf{N}$ or $N_F = \infty$. In that case

λ : EV

$$\Phi(\mathbf{x}) := \begin{pmatrix} \sqrt{\lambda_1}\psi_1(\mathbf{x}) \\ \sqrt{\lambda_2}\psi_2(\mathbf{x}) \\ \vdots \end{pmatrix}$$

satisfies $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})) = k(\mathbf{x}, \mathbf{y})$.

---

1. Stat. learning theory
2. risk minimisation: hard to find
3. ERM: min the training error
   → not a good idea: overfit
4. SRM: bounds the gen. error from above
   - small errors on training set
   - simple
   ≈ Occams razor
5. Large margin classifier
   - map to high-dim. in non-linear: optimal
   - non-linear mapping: optimal generalising

The Kernel Trick in Practice

$\|\text{function} - \mu\|$

Example:

We have a dataset $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ and would like to compute the (squared) distance to the mean in feature space, i.e.

$$f(\boldsymbol{x}) = \Big\| \underbrace{\phi(\boldsymbol{x})}_{\langle a|0\rangle} - \underbrace{\frac{1}{N}\sum_{i=1}^{N}\phi(\boldsymbol{x}_i)}_{\langle b|0\rangle} \Big\|^2$$

→ we need to take scalarproduct of each function − μ terms

→ however: do this to make it simpler

Applying the identity

$$\|\boldsymbol{a} - \boldsymbol{b}\|^2 = \langle \boldsymbol{a}, \boldsymbol{a}\rangle - 2\langle \boldsymbol{a}, \boldsymbol{b}\rangle + \langle \boldsymbol{b}, \boldsymbol{b}\rangle$$

we arrive at a formulation

$$f(\boldsymbol{x}) = \underbrace{\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x})\rangle}_{k(\boldsymbol{x},\boldsymbol{x})} - \frac{2}{N}\sum_{i=1}^{N}\underbrace{\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}_i)\rangle}_{k(\boldsymbol{x},\boldsymbol{x}_i)} + \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\underbrace{\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j)\rangle}_{k(\boldsymbol{x}_i,\boldsymbol{x}_j)}$$

where the function (initially defined in feature space) can be expressed solely in terms of the kernel function.

Scalar product in high-dim. is gone with this trick

Kernel to avoid expensive computations.

Lagrangian $L(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i (y_i \langle x_i, w\rangle + b - 1)$ $\quad \alpha_i \geq 0$

- Lagrangian must be: -maximized w.r.t. α_i
  -minimized w.r.t. w and b
- Derivatives w.r.t. primal variables must vanish: $\frac{\partial}{\partial b} L(w,b,\alpha) = 0$, $\frac{\partial}{\partial w} L(w,b,\alpha) = 0$

  which leads to: $\sum_{i=1} \alpha_i y_i = 0$ and $w = \sum_{i=1} \alpha_i y_i x_i$
- According to KKT, only the Lagrange multipliers $\alpha_i > 0$ at the saddle point correspond to constraints.
  $\alpha_i [y_i \langle x_i, w\rangle + b - 1] = 0$

→ The patterns x_i for which $\alpha_i > 0$ are Support Vectors and lie exactly on the margin.

Gaussian Gram matrix (K) each element K_{ij} represents the Gaussian kernel between observations x_i and x_j.

$$K_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

σ: bandwidth of Gaussian kernel.

Properties: Gram matrix is: - symmetric $K_{ij} = K_{ji}$

- PSD $u \in \mathbb{R}^N$, $u^T K u \geq 0$

Mercer kernels (most common)

Polynomial: $k(x,y) = (x y + c)^d$

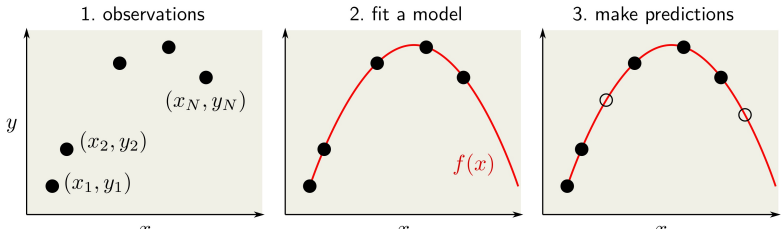
RBF: $k(x,y) = \exp(-\|x-y\|^2 / 2\sigma^2)$

Inverse quadratic: $k(x,y) = \frac{1}{\sqrt{\|x-y\|^2 + c^2}}$

Lecture 7: Model Selection

→ we want to learn a model f(x) that not only predicts well, but also predicts future unseen data well.

Occams Razor
Popper's prediction strength
Cross-Validation
Bias-Variance Analysis



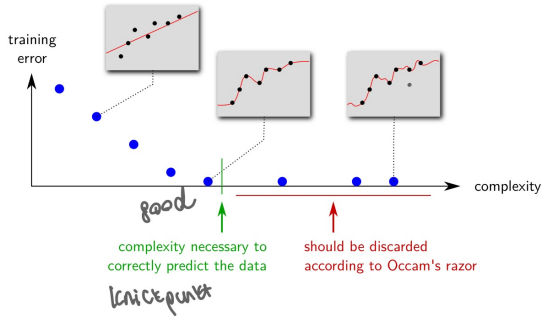
the model needs to understand the „essence"

- perfect fit / too simple / too complex is not always the best. → over/or underfit problems

→ Occams Razor

"Model complexity must not be increased beyond what is necessary to correctly predict the data."

Occam's Razor for Model Selection



find the sweet spot for complexity to reduce over-/underfit

Choose the model that is the least complex one, it will: predict new (unseen) data better & simplicity is better

Because: generalization error bound is tighter.

Quantifying Complexity: → counting model parameters
smoothness
→ SRM
Bayesian information criterion (BIC)

Approach 1: Counting the Parameters

Idea:

- If several models predict the data sufficiently well, prefer the one with the fewest parameters.

Examples of models:

complexity ↓

Constant classifier	$g(\mathbf{x}) = \underbrace{c}_{1}$ 1 parameter
Means difference	$g(\mathbf{x}) = \mathbf{x}^T \underbrace{(\mathbf{m}_1 - \mathbf{m}_2)}_{2d} + \underbrace{c}_{1}$ close to NCC
PCA + Fisher	$g(\mathbf{x}) = \underbrace{\mathrm{PCA}(\mathbf{x})^T}_{k \cdot d} \underbrace{S_W^{-1}}_{k^2} \underbrace{(\mathbf{m}_1 - \mathbf{m}_2)}_{2k} + \underbrace{c}_{1}$

k = #PC's that we project down

SRM

if we have the nesting, then we can start doing some generalisation bounds

SRM Idea:

- Structure the space of solutions into a nesting of increasingly large regions. Benefit of nesting: Bound on generalization errors

nested set of function classes

$$\cdots \; S_{n-1} \subset S_n \subset S_{n+1} \; \cdots$$

- If two solutions fit the data, prefer the solution that also belongs to the smaller regions.

Particular choices of $\cdots \subseteq S_{n-1} \subseteq S_n \subseteq S_{n+1} \subseteq \ldots$ lead to upper-bounds on the generalization error of a model (cf. next lecture).

→ the model with lowest generalization error is preferable.

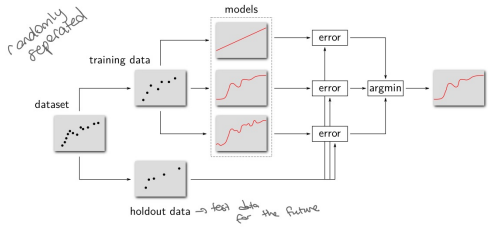
generalisation error ≈ training data error

All the points are coming from some unknown distributions. How can we know that our constructed model is good on unseen data?

The Holdout Selection Procedure

"mimique" the unknown

Idea: Predict out-of-sample error by splitting the data randomly in two parts (one for training, and one for estimating the error of the model).



- holdout data: data we haven't touched before
- we split the data to 2 parts: test & holdout
- we measure the error and take the argmin
- Problem: the more we take from the dataset to holdout, the less accurate our estimation will be.
  easiest case: lots of data

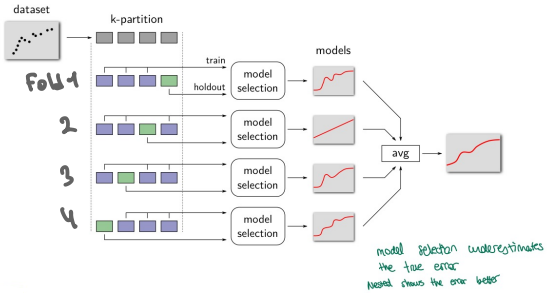
Problem: The more data we use to accurately estimate the prediction error of each model, the less data is available for training an accurate model in the first place.

Cross-Validation (k-Fold Procedure)

model selection is very hard for few data points

splitting data to k-parts ← training hold out

Idea: Retain more data for training, and make up for the lower amount of holdout data by repeating the model selection procedure over multiple data splits (and averaging the selected models):



model selection underestimates the true error
Nested shows the error better

1. fold data to k-parts
2. separate data to: test & train
3. measure your errors
4. take the average

- gives a good selection, but doesn't explain well what the model is capable of.

Holdout / Cross-Validation

Clever Hans effect:

model gives the right classification for incorrect reasons: it understands the copyright on the classifier, not the horse itself.

Datasets may include nonsensical associations.

Advantages: mimique the unseen

- The model can now be selected directly based on simulated future observations (implements Popper's principle for model selection).

Limitations:

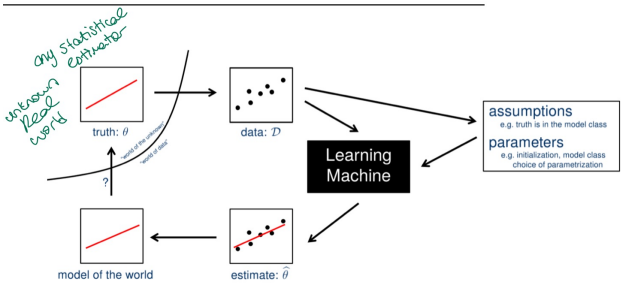
- For a small number of folds k, the training data is reduced significantly, which may lead to less accurate models. For k large, the procedure becomes computationally costly.
- This technique assumes that the available data is representative of the future observations (not always true!).

→ sample selection bias: data does not properly represent the underlying distribution

To solve: Heckman estimator

Heckman: he only did the survey on the phone
→ Sample selection bias

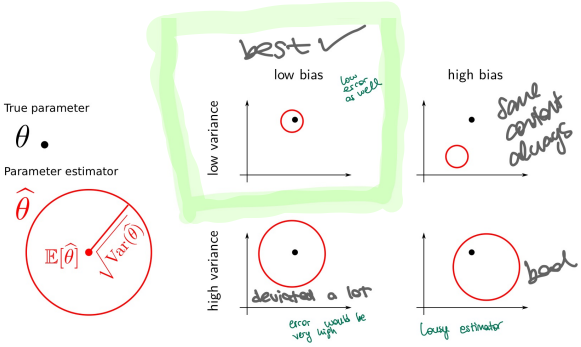
→ Bias - Variance Analysis



A good learning machine is one that produces an estimator $\hat{\theta}$ close to the truth θ. Closeness to the truth can be measured by some error function, e.g. the square error:

$$\text{Error}(\hat{\theta}) = (\theta - \hat{\theta})^2.$$

truth, in, estimator



Parametric estimation:

- θ is a value in $\mathbb{R}^h$
- $\hat{\theta}$ is a function of the data $\mathcal{D} = \{X_1, \ldots, X_N\}$, where X_i are random variables producing the data points.

Statistics of the estimator:

$$\text{Bias}(\hat{\theta}) = \mathbb{E}[\hat{\theta} - \theta]$$ (measures expected deviation of the mean)

How far are we from the truth? (on average: because dataset is sampled) average deviation

$$\text{Var}(\hat{\theta}) = \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2]$$ (measures scatter around estimator of mean)

How much does it deviate from the mean of overall estimates?

$$\text{MSE}(\hat{\theta}) = \mathbb{E}[(\hat{\theta} - \theta)^2]$$ (measures prediction error)

How far is our estimator from the truth?

Note: for $\theta \in \mathbb{R}^h$, we use the notation $\theta^2 = \theta^\top \theta$.

product notation

How far is our model from the ground truth? -Bias -Variance will tell.

if we put too much complexity we overfit

a lot of data!
⇒low Bias
⇒high Variance

Bias-Variance Decomposition

MSE is Bias + Variance

$$\text{Bias}(\hat{\theta}) = \mathbb{E}[\hat{\theta} - \theta], \quad \text{Var}(\hat{\theta}) = \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2], \quad \text{MSE}(\hat{\theta}) = \mathbb{E}[(\hat{\theta} - \theta)^2]$$

Exercise: Show that $\boxed{\text{MSE}(\hat{\theta}) = \text{Bias}(\hat{\theta})^2 + \text{Var}(\hat{\theta})}$.

MSE is our generalization error

$$\text{MSE}(\hat{\theta}) = \mathbb{E}[(\hat{\theta} - \theta)^2]$$

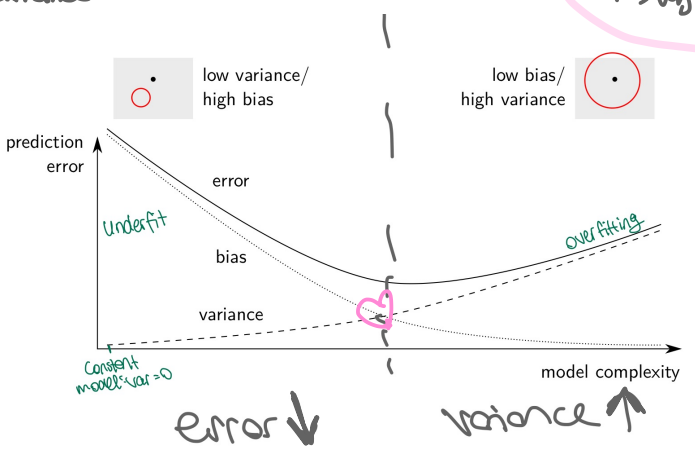
→ Trick: add 0 by adding ± 𝔼[θ̂]

$$= \mathbb{E}[(\underbrace{\hat{\theta} - \mathbb{E}[\hat{\theta}]}_{a} + \underbrace{\mathbb{E}[\hat{\theta}] - \theta}_{b})^2]$$

$$= \mathbb{E}[\underbrace{(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2}_{a^2} + \underbrace{(\mathbb{E}[\hat{\theta}] - \theta)^2}_{b^2} + \underbrace{2(\hat{\theta} - \mathbb{E}[\hat{\theta}]) \cdot (\mathbb{E}[\hat{\theta}] - \theta)}_{+2ab}]$$

$$= \underbrace{\mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2]}_{\text{Var}(\hat{\theta})} + \underbrace{(\mathbb{E}[\hat{\theta} - \theta])^2}_{\text{Bias}(\theta)^2} + \underbrace{2\,\mathbb{E}[\hat{\theta} - \mathbb{E}[\hat{\theta}]] \cdot (\mathbb{E}[\hat{\theta}] - \theta)}_{0}$$

expectation of − expect. = 0



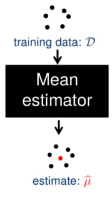
→ Parameters of a Gaussian:

parametric estimation:

θ is a value in $\mathbb{C}^n$ (e.g. $\theta = (\mu, \Sigma)$ for Gaussians)

$\hat{\theta}$ is function in the data $\mathcal{D} = \{X_1, \ldots, X_N\}$

(X_i are random variables giving back data points)



e.g. mean estimator $\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} X_i$

covariance estimator $\hat{\Sigma} = \frac{1}{N-1}(X_i - \hat{\mu})(X_i - \hat{\mu})^\top$

↳ how the data varies around the mean?

insert $\hat{\mu}$

Exercise: Show that the bias and variance of the mean estimator $\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} X_i$ are given by

NT Estimator ↑

$$\boxed{\text{Bias}(\hat{\mu}) = 0} \quad \text{and} \quad \boxed{\text{Var}(\hat{\mu}) = \sigma^2/N}.$$

our estimator:

$$\text{Bias}(\hat{\mu}) = \mathbb{E}[\hat{\mu} - \mu]$$
$$= \mathbb{E}[(\tfrac{1}{N}\textstyle\sum_{i=1}^{N} X_i) - \mu]$$
Definition of mean ←
$$= (\tfrac{1}{N}\textstyle\sum_{i=1}^{N} \mathbb{E}[X_i]) - \mu$$
$$= (\tfrac{1}{N}\textstyle\sum_{i=1}^{N} \mu) - \mu$$
$$= 0$$
↳ unbiased estimator

$$\text{Var}(\hat{\mu}) = \text{Var}[\tfrac{1}{N}\textstyle\sum_{i=1}^{N} X_i]$$
$$= \tfrac{1}{N^2}\text{Var}[\textstyle\sum_{i=1}^{N} X_i]$$
$$= \tfrac{1}{N^2}\textstyle\sum_{i=1}^{N} \text{Var}[X_i]$$
$$= \tfrac{1}{N^2}\textstyle\sum_{i=1}^{N} \sigma^2$$
$$= \tfrac{1}{N^2} N\sigma^2$$
$$= \sigma^2/N$$

- unbiased estimator is good, because if we add more data it'll estimate more correctly.

The James-Stein Estimator

"better than the mean" but not unbiased anymore

$$\widehat{\mu} = \frac{1}{N}\sum_{i=1}^{N} X_i$$

"natural" estimator

$\text{Bias}(\widehat{\mu}) = 0$

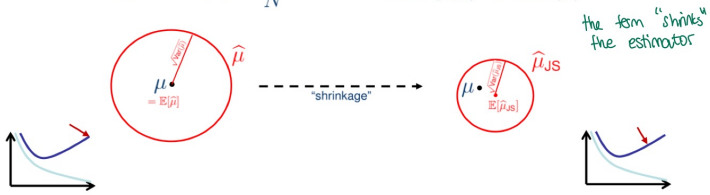
$\text{MSE}(\widehat{\mu}) = \text{Var}(\widehat{\mu}) = \frac{\sigma^2}{N}$

$$\widehat{\mu}_{\text{JS}} = \widehat{\mu} - \frac{(n-2)\sigma^2}{\widehat{\mu}^2}\widehat{\mu}$$

James-Stein estimator

$\text{Bias}(\widehat{\mu}_{\text{JS}}) > 0$

$\text{MSE}(\widehat{\mu}_{\text{JS}}) < \text{MSE}(\widehat{\mu})$



the term "shrinks" the estimator

variance is much smaller

add sth. to the diagonal also adds bit bias, but variance goes small

- **Occam's Razor:** Given two models that have sufficiently low training error, the simpler one should be preferred.
  - **Measuring Complexity:** Counting the parameters can be misleading. Structured risk minimization (SRM) is more reliable in practice.
- **Popper's View:** How to make sure that a model predicts well? By testing it on out-of-sample data.
  - **Holdout and Cross-Validation:** Common practical procedures to simulate out-of-sample prediction behavior. Has some limitations (e.g. assume that the current dataset is representative of the true distribution).
- **Bias-Variance Decomposition:** The error of a predictive model can be decomposed into bias and variance. Best models can often be interpreted as finding a good tradeoff between the two terms.

the simpler the better

for few data points: leave-one out CV is a good strategy

simple models: ↓ low variance high bias ↑
complex models: ↑ high variance low bias ↓

Biased Estimator

to make the estimator less affected by the sampling of data.

- A biased estimator can reduce the variance of the estimate at cost of introducing some bias.
- Introduce some bias to the model to reduce variance to get more generalized results. → Regularization: Prevent overfitting by penalizing overly complex models

- In some models, bias is deliberately introduced to achieve lower variance to get better generalisation on unseen data.
- Introducing bias to the model makes the model less sensitive (because variance is decreasing) by reducing the impact of random fluctuations in the sample.

Overfitted estimator has: high variance - low bias

High variance models are sensitive to small fluctuations in the training set, leading to overfitting.
They perform well on training data but do poor generalisation.

Curse of dimensionality: Number of features >> observations

Introduce a bias (f.e. with regularization) to reduce estimation error

$$MSE = bias^2 + var$$

Biased Estimator

to make the estimator less affected by the sampling of data.

- A biased estimator can reduce the variance of the estimate at cost of introducing some bias.
- Introduce some bias to the model to reduce variance to get more generalized results. → **Regularization**: Prevent overfitting by penalizing overly complex models.
- In some models, bias is deliberately introduced to achieve lower variance to get better **generalisation** on unseen data.
- Introducing bias to the model makes the model less sensitive (because variance is decreasing) by reducing the impact of random fluctuations in the sample.

Overfitted estimator has: high variance - low bias

High variance models are sensitive to small fluctuations in the training set, leading to overfitting.
They perform well on training data but do poor generalisation.

Curse of dimensionality: Number of features » observations

Introduce a bias (f.e. with regularization) to reduce estimation error

$$MSE = \text{bias}^2 + \text{var}$$

Lecture 8: Neural Networks 1

Bayes Decision Theory → optimal classifiers
we know the data properties
no better classifier

What if we don't know the data?

→ parameter estimation

→ Mean Separation / Fischer → Perceptron
focus on db seperation of 2 classes
→ NN
Linear + Non-Linear

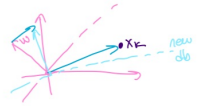
→ Define classification based directly on data rather than learning data distribution

Model Selection → Occams Razor: build a reduced set of function to feed the data; less overfit chance ex: constant classifier
Simplicity and Generalization evaluation
cross validation (training bunda done / test (holdout) data bunu ne kadar accurate diye bak sonra
(i) amount of data, can we still build a model that generalize?

Perceptron (check Lecture 2)

The Perceptron Algorithm

bias to adjust the location of the db



- Consider our linear model

$$z_k = \mathbf{w}^\top \mathbf{x}_k + b \qquad y_k = \text{sign}(z_k)$$

and let t_k be 1 and -1 when the true class of $\mathbf{x}_k$ is ω_1 and ω_2 respectively.

Algorithm

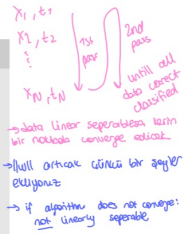
- Iterate over $k = 1 \ldots, N$ (multiple times).
  - If $\mathbf{x}_k$ is correctly classified($y_k = t_k$), continue.
  - If $\mathbf{x}_k$ is wrongly classified ($y_k \neq t_k$), apply:

update rules: ← w' yeni hali

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \cdot \mathbf{x}_k t_k$$
$$b \leftarrow b + \eta \cdot t_k$$

where η is a learning rate.

- Stop once all examples are correctly classified.



The Perceptron: Optimization View

The perceptron can be seen as a gradient descent of the error function

$$\mathcal{E}(\mathbf{w}, b) = \frac{1}{N} \sum_{k=1}^{N} \underbrace{\max(0, -z_k t_k)}_{\mathcal{E}_k(\mathbf{w}, b)}$$

no change / this change

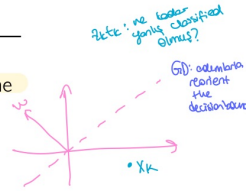
Gradient:

$$\frac{\partial \mathcal{E}_k}{\partial w} = \underbrace{\mathbb{1}_{\{-z_k t_k \geq 0\}}}_{\text{indicator not correctly classified}} \cdot (-x_k \cdot t_k)$$

$-t_k \cdot$ internal derivative of z_k

$$w = w - \mu \frac{\partial \mathcal{E}_k}{\partial w} = w + \mu \cdot x_k t_k$$

where I have incorrectly classified the data



Non-linear Classification

(first time talking about non-linearity in this course)

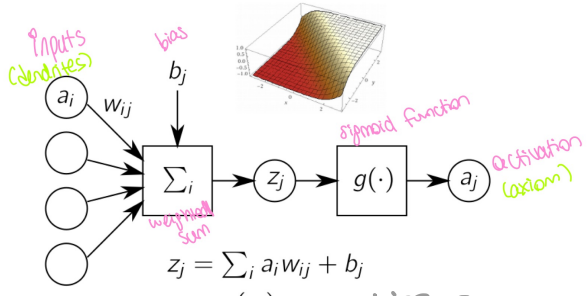
Key Idea: → Kernel Trick

- Transform the data nonlinearly through some function Φ before applying the linear decision function.

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

- *Example:* $\Phi(\mathbf{x}) = [x_1, x_2, x_1^2, x_2^2, x_1 x_2]$ and $\mathbf{w} \in \mathbb{R}^5$.

artificial neuron:



$$z_j = \sum_i a_i w_{ij} + b_j$$
$$a_j = g(z_j) \rightarrow \text{nonlinear}$$

→ simple multivariate, nonlinear, differentiable
→ complex nonlinear representations with neurons
→ ability to learn from the data

→ universal approx. theorem

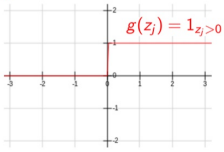
Examples of Nonlinear Functions

activation functions

Universal approximator

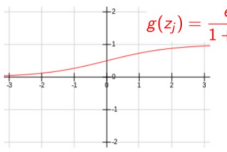
threshold function

$g(z_j) = 1_{z_j > 0}$



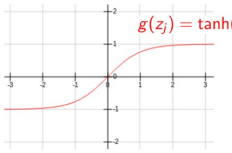
logistic sigmoid → has gradient → differentiable

$g(z_j) = \frac{e^{z_j}}{1 + e^{z_j}}$



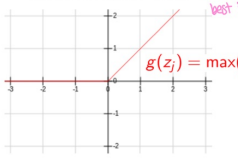
hyperbolic tangent

$g(z_j) = \tanh(z_j)$

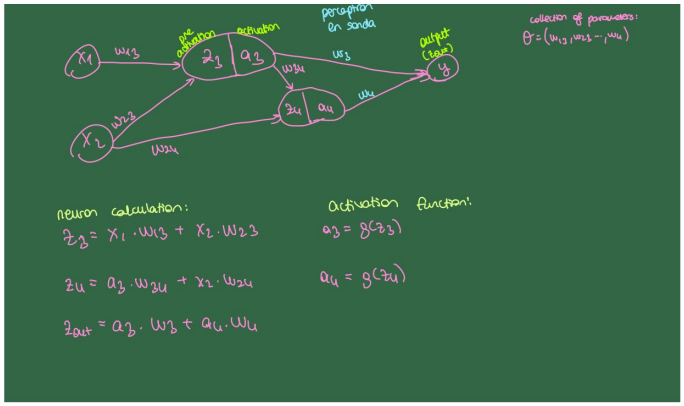


rectified linear unit — ReLU best in practice

$g(z_j) = \max(0, z_j)$



The Forward Pass

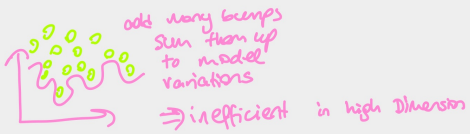


Universal Approx. Theorem

„With sufficiently many neurons, NN can approximate any nonlinear function."

add the bumps together, each iteration has a hidden layer

as many bumps as I need to encode every local variation of our function



approximation with infinite Fourier:

$$y(x_1, x_2) \approx \sum_s A_s(x_1) \cos(s x_2)$$

Step function:

→ move hyperplane by the bias

→ scale it with w

→ then add the step functions of neurons

Training a NN

- Use the same error function as the perceptron, but replace the perceptron output z by the neural network output z_{out}:

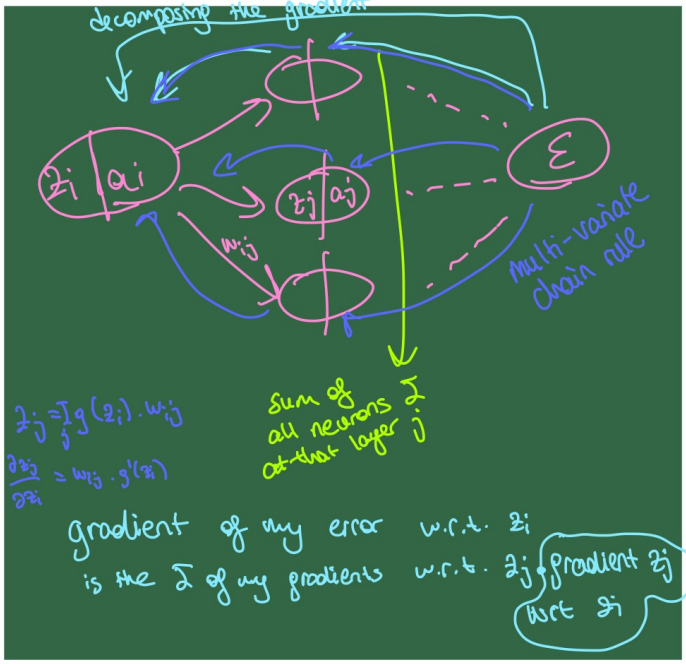
$$\mathcal{E}(\theta) = \frac{1}{N} \sum_{k=1}^{N} \underbrace{\max(0, -z_{\text{out}}^{(k)} t^{(k)})}_{\mathcal{E}^{(k)}(\theta)}$$

and compute the gradient of the error function w.r.t. the parameters θ of the neural network.

Error Backpropagation

chain rule

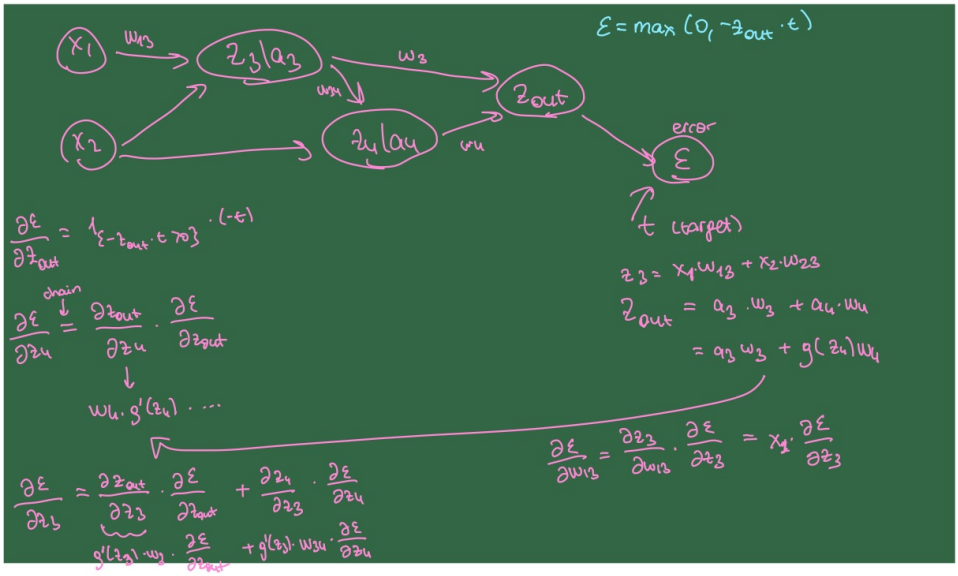
to find Gradients



Gradient with chain rule: $\frac{\partial \mathcal{E}}{\partial z_i} = \sum_j \frac{\partial z_j}{\partial z_i} \frac{\partial \mathcal{E}}{\partial z_j}$ for layers

for parameters: $\frac{\partial \mathcal{E}}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \cdot \frac{\partial \mathcal{E}}{\partial z_j}$

Error Backpropagation



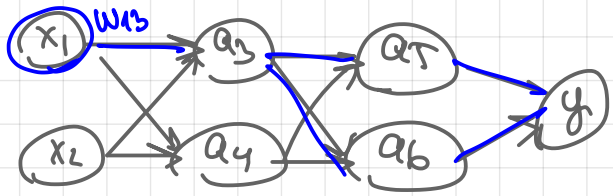
Summary

- The **perceptron** and the **neural network** enable training classifiers on more complex distributions by focusing on what is critical for classification, i.e. the boundary between classes.
- The **neural network** enables learning **nonlinear** decision boundaries. This is useful when the problem is complex (most practical problems are nonlinear).
- The gradient of a **neural network** required for learning can be easily and quickly computed using the method of **error backpropagation**.
- The perceptron and the neural network do not have closed form solutions but can be trained iteratively using **gradient descent**.

▶ NN can learn any function, even though not the best solution
→ the from-NN-learned feature space can have multi-scale properties:
it doesn't have to be a reproducing kernel H. space.
→ 1-1 mapping of NN & SVM possible.
choose kernel
↳ representation
↳ NN learns the representation

How to do backpropagation?

It will ask you $\frac{\partial \text{output}}{\partial \text{weight of one input neuron}}$.



Go to that input neuron and follow its arrows to following neurons.

Example: We need to find.. $\frac{\partial y}{\partial w_{13}}$ w_{13} goes to all the blue paths

so seperate into 2 cases: $\frac{\partial y}{\partial w_{13}} = \frac{\partial y}{\partial a_6} + \frac{\partial y}{\partial a_5}$ } then calculate 2 paths chain rule

$\frac{\partial y}{\partial a_6} = \frac{\partial y}{\partial a_6} \cdot \frac{\partial a_6}{\partial z_6} \cdot \frac{\partial z_6}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_{13}}$

each activation function must be derivated from their zs.

$\frac{\partial y}{\partial a_5} = \frac{\partial y}{\partial a_5} \cdot \frac{\partial a_5}{\partial z_5} \cdot \frac{\partial z_5}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_{13}}$

How to find bias and w of a NN?

- give points : 1 for class A 1 for class B

$a_j = g(x \cdot w_{ij} + b_j)$ → needs to be 0 in critical points

Lecture 9: Neural Networks 2

→ Optimizing NN: training efficiently & effectively — Hessian & condition number
→ Regularizing NN: learned model is good & generalizes well — margin losses & perturbations

Optimizing the NN

① Initialization

weights should be initialized randomly

- Weights should be initialized *randomly* (it breaks symmetries in parameter space and reduces the risk of landing in bad local minima or getting stuck on plateaus).
- If necessary, train the network *multiple times* using different random seeds, and retain the network with the *lowest error*.

He-et-al[1] random initialization heuristic:

For neural networks with ReLU neurons, it is recommended to use: $w_{ij} \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = \sqrt{2/\#\text{ input connections}}$ and where $\mathcal{N}$ is the Gaussian distribution.

Example: For a two-layer neural network with 50 input features, 200 hidden neurons, and 1 output, weights in the first layer should be initialized using $\sigma = \sqrt{2/50}$, and in the second layer using $\sigma = \sqrt{2/200}$.

③ Hessian

2nd derivative
to understand the curvature of the error function
→ results of Hessian = EV (λ)

Condition number: highest EV / lowest EV ratio

condition number ↑ (slower) the convergence ↓

⇒ lower condition number, the better.

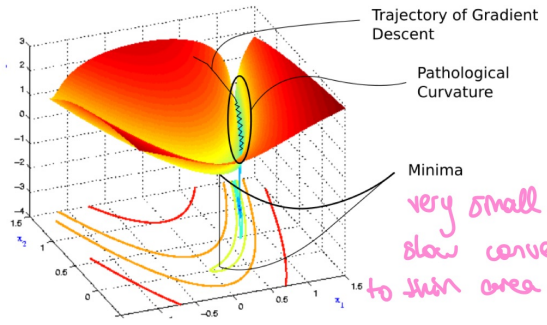
- Hessian describes how the minimum looks like
- Rounder & low conditioning number minimum is the best.
- Condition number can be reduced by centering the data before training.

② Neural Networks: Pathological Curvature

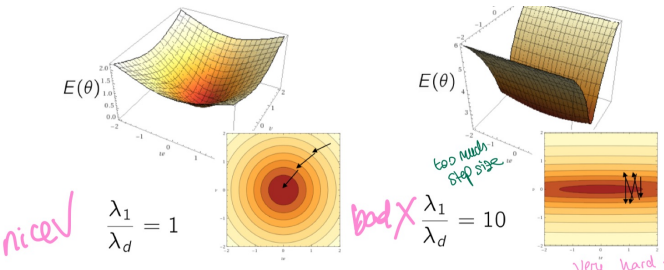
- How hard is it to optimize a neural network? pathological curvature ideal
- Are we guaranteed to converge to a good local minimum?
- How quickly do we converge to this local minimum?

low conditioning numbers

Another problem: The neural network may converge to some good local optimum but slowly due to *pathological curvature* of the error function.



very small step
slow convergence due to this area



nice ✓ bad ✗

Very hard to optimize, therefore condition number very important

The lower the condition number, the better.

4

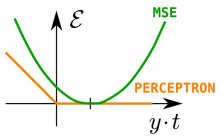
Hessian Analysis (Linear Case, MSE)

↳ How the minimum looks like

minimum the best if: ① low conditioning number ② round: 😊 no taco

We center the data to get a low conditioning number:

Consider the simplest case of a homogeneous linear model $y = \boldsymbol{w}^\top \boldsymbol{x}$. Consider $\mathcal{E}(\boldsymbol{w}) = \mathbb{E}[0.5 \cdot (1 - y \cdot t)^2]$, the mean square error (MSE), which is easier to analyze than the perceptron loss.



The Hessian of the error function is given by:

$$H = \frac{\partial^2 \mathcal{E}}{\partial \boldsymbol{w}^2} = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]$$

data covariance

i.e. the data uncentered covariance.

Condition Number: Assuming $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 I)$, the Hessian reduces to $H = \sigma^2 I + \boldsymbol{\mu}\boldsymbol{\mu}^\top$, and the condition number is

$$\frac{\lambda_1}{\lambda_d} = 1 + \frac{\|\boldsymbol{\mu}\|^2}{\sigma^2}.$$

Importance of centering the data: • $\mu \neq 0$ ⇒ so data not centered • therefore condition number very high • if data is centered: $\mu = 0$ and the condition number $\frac{\lambda_1}{\lambda_d} = 1$

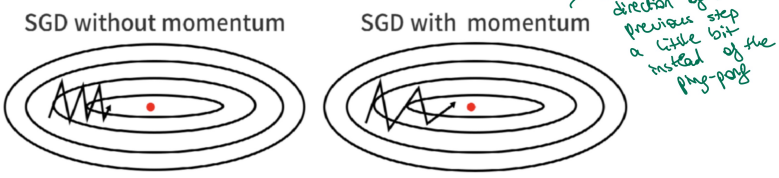
(show this in the homework). In other words, the condition number can be reduced by **centering the data** before training.

so that optimization is more smoother

19/3

→ While centering it is also ok to have a function that: $g(0) = 0$ (durch die Ursprung)
suitable activation functions for that are:
ReLu: $\max(0, z)$, $\tanh(z)$, centered softplus

→ Centering reduces the pathological curvature, but does not completely eliminate
To make it faster: Gradient Descent with Momentum:



5

Improving Gradient Descent with Momentum:

(Stochastic) Gradient Descent:

new weight = old weight − step size · gradient

$$\theta = \theta - \gamma \cdot \frac{\partial \mathcal{E}_k}{\partial \theta} \qquad \text{with} \quad k \sim \{1, \ldots, N\}$$

where γ is the learning rate.

(Stochastic) Gradient Descent + Momentum: ≈ filtering

momentum + corrected value

$$\Delta_{\text{new}} = \mu \Delta_{\text{old}} + \frac{\partial \mathcal{E}_k}{\partial \theta} \qquad \text{with} \quad k \sim \{1, \ldots, N\}$$

$$\theta = \theta - \gamma \Delta_{\text{new}}$$

→ Backpropagation gives us this as a linear time algorithm

where γ is the learning rate and μ is the momentum.

How to compute Hessian? Take 2nd derivative of the objective function: $H = \frac{\partial}{\partial w}\left(\frac{\partial J(w)}{\partial w}\right)$

∇_0 $Cov(X) = E(XX^T) - E(X)E(X)^T$

How to find the condition number? $\frac{\lambda_1}{\lambda_d} = 1 + \frac{\|\mu\|^2}{\sigma^2}$

1. Take 1st EV (highest):

$\lambda_1 = \max_{\|v\|=1} v^T H v$ | plug in H

| $v^T v = 1$

$v \perp \mu$

IF our $H = \sigma^2 I + \mu\mu^T$: our Eigenvalues are σ^2 in the diagonal

Regularizing the NN that generalizes well to new data

→ Hinge Loss adding a bit more to margin to penalize

So far:

- We have used the perceptron loss:

$$\mathcal{E}_k(\theta) = \max(0, -y_k t_k)$$

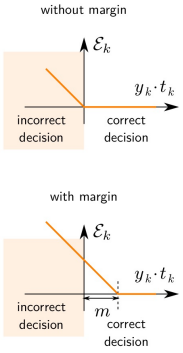
which becomes zero as soon as the current data point is being correctly classified.

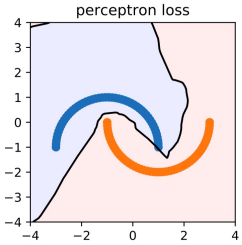
Idea:

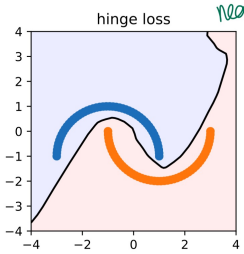
- Impose a penalty to points that lie too close to the decision boundary (i.e. add a margin), aka. the Hinge Loss:

$$\mathcal{E}_k(\theta) = \max(0, 1 - y_k t_k)$$

This is similar to the slack variables penalties in the soft-margin SVM formulation.







neater, decision surface is more away from the data.

- Using the hinge loss, the decision function of the neural network moves away from the data.
- This leads to a better generalization performance.

→ How to regularize NN?

① weight decay add a term $\lambda \|w\|^2$ to the objective, to make w smaller and closer to a linear model.

weight ↑ non-linearity ↑
try to stop early if weight to big, to remain in a linear space.

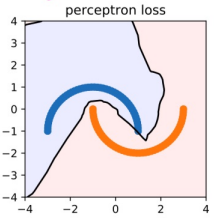
② Data perturbation (anxiety) random perturbations to inputs & labels
Gaussian noise and random flips

③ Representation perturbation Dropout method
randomly turning the neurons on/off while training

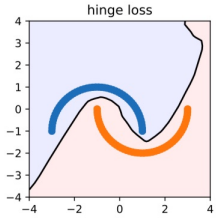
④ Training noise stochastic Gradient descent:
-to regularize choose some random data (=stochastic)
-then take a step according to these summed up gradients from random chosen bag of data

Data Perturbation at Work

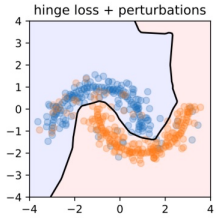
small margin



mid-margin



smoother



Observation:

- Data perturbations pull the decision boundary further away from the class manifolds, and typically further improves generalization.

SVMs & kernels:
→ easy to optimize & regularize
→ bad at large datasets
→ unique minimum
→ not efficient to regularize because they are a squared algorithm

NNs:
→ can be trained on very large distributions (GPU)
→ hard to optimize and regularize, we care more about generalization here.

Lecture 10: Decision Trees and Random Forests

Tree-based methods:
- random forests segment the body to different parts and we make decisions (X-box)
- recommender systems
- banking
- medical ...

Given: Features: $X_1, X_2, \ldots, X_p$
Target: Y

Key idea:

(1) Partition feature space into a set of rectangles.
(2) Fit a simple model (e.g. a constant) in each one.

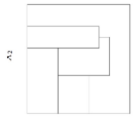
General Partition
(can not be obtained)

divide the space into some segments and have the very simple model predict every outcome of the rectangle

Problem: Dividing a large p-dimensional space to dimensions is hard.

For each partition: model Y with a different constant.

very hard computation
better to use divide & conquer



General Partition
(can not be obtained)

Although each partitioning line has a simple description, e.g. X_1=c, some regions are complicated to describe.

Simplification: recursive binary partitions

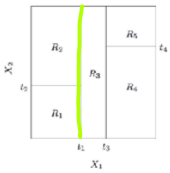
Recursive
Binary Splitting

Algorithm (divide & conquer):

repeat (1) Split space into two regions.
(2) Model response by mean of Y in each region.

What is the best partition for x_i to have a certain classification?

Concept of Decision Trees



Recursive
Binary Splitting

Example:

① everything < t_1 / everything > t_1

- Split at $X_1 = t_1$
- Split the region $X_1 <= t_1$ at t_2
- etc.

② then do the same to t_2
③ then to t_3
⋮
compute the mean of the label for every box ≈ Ensemble (bagging)

Result: Partitioning into R_1-R_5

Model: $\hat{f}(X) = \sum_{m=1}^{5} c_m I\{(X_1, X_2) \in R_m\}$

Splitting is hard, can't draw the boxes in high dimensions, but we can draw a decision tree.

Decision Trees – algorithm

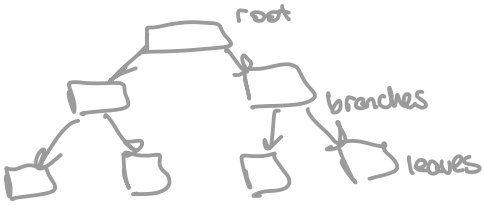
Given: N observations, P features
(x_i, y_i) for $i = 1, 2, \ldots, N$, with $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$.

Wanted: splitting variables j, split points s
(to partition space into M regions)

Minimization criterion: e.g. $\hat{c}_m = \text{ave}(y_i | x_i \in R_m)$ (sum of squares)

use in each region to find variance

Finding best binary partition is computationally infeasible (NP hard): **use greedy approach**



- split the area such that it gives good regression info.
- then compute the average

→ splitting p-dimension to S splits is too expensive
so better to go step-by-step: greedy approach (not optimal!)

Splitting

- which variable should we split?
inner minimisation

Given: N observations, P features
(x_i, y_i) for $i = 1, 2, \ldots, N$, with $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$.

Consider a splitting variable j and split point s and define a pair of half planes:
$R_1(j,s) = \{X | X_j \leq s\}$ and $R_2(j,s) = \{X | X_j > s\}$

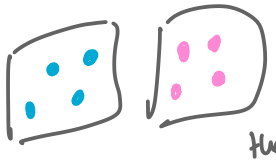
We seek j and s to solve:
$$\min_{j,s}\left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2\right]$$

Splitting:
Inner minimization solved by: $\hat{c}_1 = \text{ave}(y_i | x_i \in R_1(j,s))$ and $\hat{c}_2 = \text{ave}(y_i | x_i \in R_2(j,s))$
I.e. scan through all P features and determine optimal (j,s)

→ Splitting metrics

To measure our splitting

entropy: measure of disorder



do the split such that the boxes are the purest.

I_{pi}: prob. in child nodes

Info Gain = parent entropy − children weighted sum entropy

Information gain

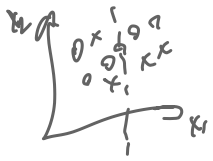
Based on the concept of entropy from information theory.

$$H(T) = I_E(p_1, p_2, \ldots, p_J) = -\sum_{i=1}^{J} p_i \log_2 p_i$$

were $p_1, p_2, \ldots$ represent the probability of each class present in the child node that results form splitting the tree.

$$\overbrace{IG(T,a)}^{\text{Information Gain}} = \overbrace{H(T)}^{\text{Entropy(parent)}} - \overbrace{H(T|a)}^{\text{Weighted Sum of Entropy(Children)}}$$

Choose the split that results in the purest daughter nodes.



put some threshold
measure frequencies on both sides
take the sum
⇒ gini impurity

Gini impurtiy

Measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset

$$GI(\mathcal{X}) = \sum_{c=1}^{C} p(c)(1 - p(c)) = \sum_{c \neq \tilde{c}} p(c)p(\tilde{c})$$

Similar to entropy, zero if all samples belong to same class.

→ for continious variables
→ mainly for regression
→ if we take sth. close to a constant, then the variance of the box very low.
→ too much variance in the box: split it up

Split test < true / false > is then my threshold

Variance Reduction (often used for regression)

Variance reduction of a node is defined as the total reduction of the variance of the target variable due to the split at this node.

$$I_V(N) = \frac{1}{|S|^2} \sum_{i \in S} \sum_{j \in S} \frac{1}{2}(x_i - x_j)^2 - \left(\frac{1}{|S_t|^2} \sum_{i \in S_t} \sum_{j \in S_t} \frac{1}{2}(x_i - x_j)^2 + \frac{1}{|S_f|^2} \sum_{i \in S_f} \sum_{j \in S_f} \frac{1}{2}(x_i - x_j)^2 \right)$$

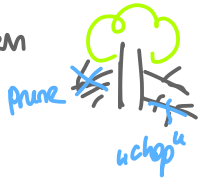
were S, S_t, and S_f are the set of pre-split sample indices for which the split test is true and false, respectively.

When to stop the split?

Pruning

a lot of splits → very large tree ⇒ overfit
too small ⇒ not all important things are captured

⇒ PRUNE it! cut branches or collapse them



small box size = small N → MSE = 0
→ our cost = 0

but our tree is huge → |T| very large
→ α to balance the complexity

→ idea: randomly remove nodes in the tree with splitting tests so that it's always true

How large to grow the tree?

- too large: overfitting
- too small: not all important structure is captured

Solution: grow large tree, then prune using *cost-complexity criterion*
(collapse any number of internal (non-leaf) nodes)

$N_m = \#\{x_i \in R_m\}$, (training data in region)

$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$, (prediction in region) = mean of regression

$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$, (squared loss) mean — how good are we fitting?

$|T|$ (number of terminal leaves) — how deep is the tree?

$\alpha \geq 0$ (regularization parameter)

Find subtree that minimizes:

$$C_\alpha(T) = \sum_{m=1}^{|T|} \underbrace{N_m Q_m(T)}_{\text{cost}} + \underbrace{\alpha |T|}_{\text{complexity}}$$

Find subtree that minimizes:

$$C_\alpha(T) = \sum_{m=1}^{|T|} \underbrace{N_m Q_m(T)}_{\text{cost}} + \underbrace{\alpha |T|}_{\text{complexity}}$$

training error

simplicity

- complexity will decrease if we start to collapse the nodes → T will decrease
- if we have low layers, we won't get a low training error.
  too many layers → complexity high, cost low

Weakest Link Pruning:

1. Starting with the full tree, successively collapse the internal node that produces the smallest per node increase in $\sum_m N_m Q_m(T)$

   Result: Sequence of trees $T_0, T_1, T_2, \ldots, T_m$ (were T_m is a single-leaf tree)

2. Choose optimal α (the optimal tree T_i) via cross-validation.

Decision Trees- advantages

- Interpretability
- Generalizes to higher dimensions
  ↳ no curse of dimensionality
- Can handle mixed predictors: quantitative & qualitative
- Easily ignores redundant variables
- Handles missing data elegantly

BUT: decision trees are prone to overfitting (high variance)

Bounding Generalization Error

[Haussler'88]: When the hypothesis space H is finite, then, the generalization error of a consistent (i.e. correctly classifying) hypothesis h can be bounded as:

$$P(\text{error}_{\mathcal{X}}(h) > \varepsilon) \leq |H| e^{-m\varepsilon}$$

... where $|H|$ is the size of the hypothesis space, and m is the number of iid samples in the training set.

Observation: generalization error grows with the number of hypotheses and decreases with the number of data points.

Bounding Generalization Error

Limitation of Haussler's bound:

- It applies only to set of hypotheses that perfectly classify the data. What if we use a simple model (e.g. shallow decision tree or linear classifier?)

Another bound (derived from the Chernoff inequality) for the generalization error is given by:

$$\text{error}_{\mathcal{X}}(h) \leq \underbrace{\text{error}_{\mathcal{D}}(h)}_{\text{training error}} + \underbrace{\sqrt{\frac{\log |H| + \log \frac{1}{\delta}}{2m}}}_{\text{error due to model complexity}}$$

training error ~ **bias**

error due to model complexity ~ **variance**

→ to bound the decision tree
· variance here is the size of the hypothesis space of a decision tree

Complexity of Decision Trees

- **Example 1:** General decision tree with n binary features (i.e. expanded until it perfectly classifies data):

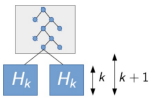
  $|H| = 2^{2^n}$ i.e. all possible lookup tables over n binary features.

- **Example 2:** Decision tree of depth 0:

  $|H| = 2$ H = {always labeling "0", always labeling "1"}

Solution for trees of depth k: Use induction

- Initial condition: $|H_0| = 2$



- Recursion: $|H_{k+1}| = n \times |H_k| \times |H_k|$

  choice of root attribute | left tree | right tree

k: # tree depth
n: # features

- Result $\log_2 |H_k| = (2^k - 1)(1 + \log_2 n) + 1$

Wrap-up

How large is the hypothesis space of decision trees?

$$\text{error}_{\mathcal{X}}(h) \leq \underbrace{\text{error}_{\mathcal{D}}(h)}_{\sim \text{ bias}} + \underbrace{\sqrt{\frac{\log |H| + \log \frac{1}{\delta}}{2m}}}_{\sim \text{ variance}}$$

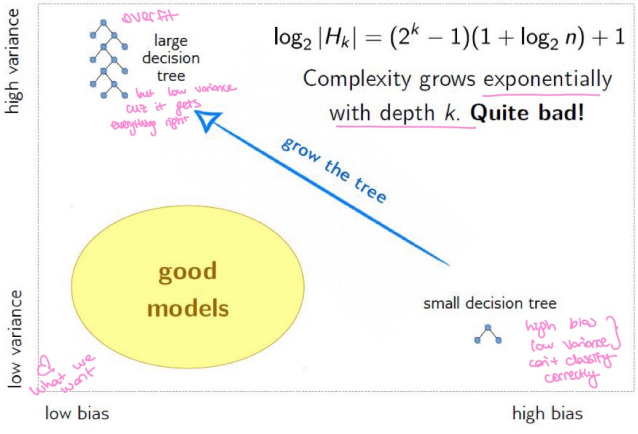
Decision tree:

- Depth 0 (constant labeling) $\log_2 |H_0| = 1$ — very high bias, low variance, don't looking the data at all
- Depth k $\log_2 |H_k| = (2^k - 1)(1 + \log_2 n) + 1$
- Unrestricted $\log_2 |H| = 2^n$ — very high depths, no bias

high bias / low variance

low bias / high variance

Decision Trees and Bias Variance



→ **We need better capacity control.**

Idea: combine decision trees with averaging, boosting. (= bagging)

→ Decision Tree regularization via model averaging (ensembles)

Generalization error: $\mathrm{E}\left[(y-\hat{f}(x))^2\right] = \mathrm{Bias}\left[\hat{f}(x)\right]^2 + \mathrm{Var}\left[\hat{f}(x)\right] + \sigma^2$

bias: low (if sufficiently deep)
+ **variance:** high (sensitive to choice of splits)
+ **residual error**

Consequence: decision trees often produce noisy or weak classifiers!

idea: build decision trees with same random setup, such that Var goes, but Bias stays low.

it needs to have
- low training error
- high variance needs to go away because of average

Idea: Combine the predictions of several randomized trees into single model.

Bagging (Bootstrap Aggregating)

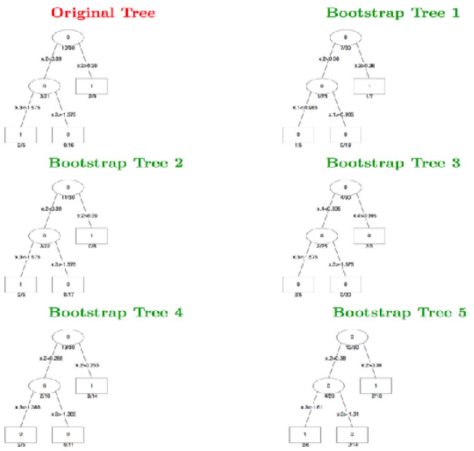
Classifier $C(\mathcal{S}, x)$
training data S
↓ subsample

Train B models on different dataset:

Draw $\mathcal{S}^{*1}, \ldots \mathcal{S}^{*B}$
samples sets of length *N*
(bootstrap samples: duplicates allowed!)

Prediction:

$\hat{C}_{bag}(x) = \text{Majority Vote } \{C(\mathcal{S}^{*b}, x)\}_{b=1}^{B}$
(in classification; regression: avergage)

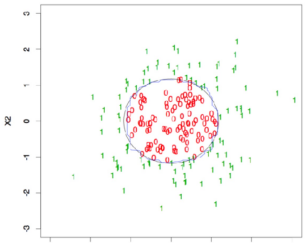


first:

↳ draw different data from the bag and build your model

→ then: do a majority vote over all the bootstrap trees

→ Variance is now reduced:



Improved prediction via reduced variance

Bagging averages trees for smoother decision boundaries.

→ limitations of bagging:

1) Datasets highly overlap:
predictions of different trees highly correlated
↳ Because var-reduction works only well if data not correlated.

2) Variance is not reduced as much as wanted ☹

Random Forests

Random Subspace Method (Tin Kam Ho, 1998):
Randomly sample features $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ without replacement

➡ De-correlates estimators and decreases variance of the aggregate.

→ we take a subspace and sample different subspaces and build models for randomly sampled subspaces and add them up.

Random Forest (Breiman, 1999):
Combination of Bagging + Random Subspace Method applied to decision trees

- Complexity control through "out-of-bag" control
  (estimate generalization error using untrained samples)
- Random feature selection either at tree or split level

→ some samples are not used for training and we use that as the generalisation error
→ to add randomness for variance reduction

Random Forrests - algorithm

在 Bagging 的基础上引入了 **随机特征选择（Random Feature Selection）**：
在构建每棵决策树时，除了对样本进行 Bootstrap 采样，还对特征子集进行随机采样（即每次分裂时只从部分特征中选择最佳划分）。
这样可以进一步增加模型的多样性，减少相关性，提高泛化能力。

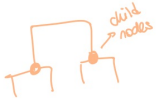
For $b = 1$ to B:

(a) Draw a bootstrap sample (random sampling with replacement) of size N from the training data.
(b) Grow a tree T_b to the bootstrapped data until minimum node size reached:
   i. Select m features at random.
   ii. Pick best variable/split-point among the m.
   iii. Split node into two daughter nodes.

→ ensemble of B trees where we did random sampling

→ algorithm simple but powerful

Result: ensemble of B trees.



Prediction: $\hat{f}_{\mathrm{rf}}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$ (regression)

$\hat{C}_{\mathrm{rf}}^{B}(x) = majority\ vote\ \{\hat{C}_b(x)\}_1^B$ (classification)

Random Forest vs. Single tree:

- Train each tree on bootstrap resample of data.
- For each split: consider only a subset of m randomly selected features.
  (typically $m = \sqrt{p}$ or $\log_2 p$ were p is the total number of features)
- No pruning.
- Fit B trees this way and aggregate predictions.

Single Trees

+ yield insight into decision rules
+ rather fast
+ easy to tune parameters

- predictions with high variance
  don't generalize well ∴

Random Forests

+ smaller prediction variance: better performance
+ easy to tune parameters

- slower
- "black box"

Boosting

Idea: given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote.

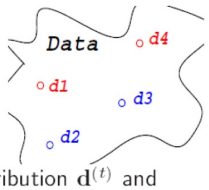
- prior distribution when drawing the data is changed → essence of boosting
- harder to learn data will be evaluated better

On each iteration:

- Like in bagging, draw a sample of the observations from data (with replacement).
- Unlike in bagging, observations are not sampled randomly
  Higher weight observations are more likely chosen.
- Weight each training example by how incorrectly it was classified.
  Weight is higher for examples that are harder to classify.

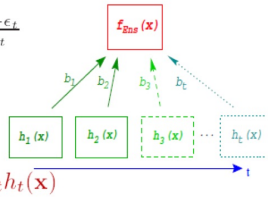
The Adaboost Algorithm

Input: N examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$
Initialize: $d_i^{(1)} = 1/N$ for all $i = 1 \ldots N$
Do for $t = 1, \ldots, T$,



1. Train base learner according to example distribution $\mathbf{d}^{(t)}$ and obtain hypothesis $h_t : \mathbf{x} \mapsto \{\pm 1\}$.
2. compute weighted error $\epsilon_t = \sum_{i=1}^{N} d_i^{(t)} \mathrm{I}(y_i \neq h_t(\mathbf{x}_i))$
3. compute hypothesis weight $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
4. update example distribution

$$d_i^{(t+1)} = d_i^{(t)} \exp\left(-\alpha_t y_i h_t(\mathbf{x}_i)\right) / Z_t$$



Output: final hypothesis $f_{\mathrm{Ens}}(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$

Gradient Tree Boosting - conceptually

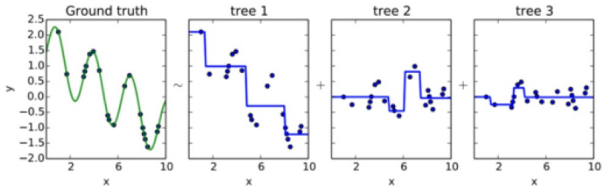
- all the trees are added up

Idea: Iteratively improve a weak learner by correcting it: $F_{m+1}(x) = F_m(x) + h(x)$

A perfect correction h(x) implies: $F_{m+1}(x) = F_m(x) + h(x) = y$
i.e. $h(x) = y - F_m(x)$

Hence we fit a new tree to the residual: $y - F_m(x)$ → $\frac{1}{2}(y - F(x))^2$
(negative gradient of quadratic loss function)

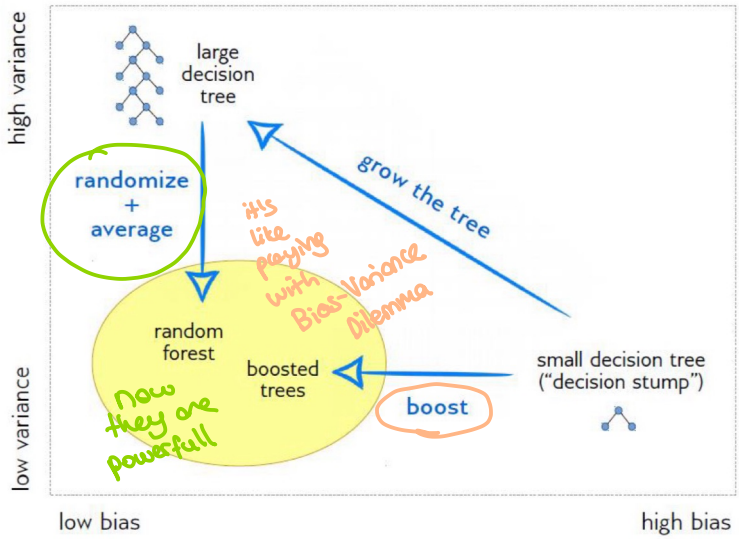
Residual fitting:



Gradient boosting is a gradient descent algorithm and can be trivially generalized with other loss functions and their gradients.

Decision Trees and Bias Variance

before: decision trees couldn't generalize well
now: 2 approaches for low bias & low variance
- random forest
- boosted trees



Lecture 11: Boosting & Ensemble Learning

main goal: find α

Ensemble Learning and Classification

- Ensemble for binary classification consists of
  - Hypotheses (basis functions) $\{h_t(\mathbf{x}) : t = 1, \ldots, T\}$
    - of some hypothesis ("concept") set $\mathcal{H} = \{h \mid h(\mathbf{x}) \mapsto \{\pm 1\}\}$ sign fct.
  - Weights $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_T]$ → find this
    - satisfying $\alpha_t \geq 0$
- Classification Output: weighted majority of the votes
  - $f_{\text{Ens}}(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$ weighted sum
- How to find the hypotheses and their weights?
  - Bagging (Breiman, 1996): $\alpha_t = 1/T$ T: # weak classifier
  - AdaBoost (Freund & Schapire, 1994)

- we wanna do binary classification
- we have T basis functions (hypotheses) (non-linear)
- for each of the T function we have a weight that is positive

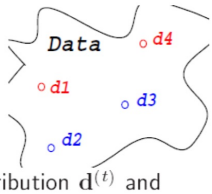
→ Ensemble: weighted sum over all the hypotheses ≙ majority vote of the classification

Bagging: • take all the functions h_t and think of them as small classifiers
• sum them up with equal weight: $1/t$
∴ sum up things and get the majority decision

AdaBoost: • reweight sum of the hypotheses in a way that you trust some more than others

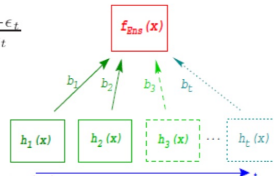
→ AdaBoost Algorithm

Input: N examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$
Initialize: $d_i^{(1)} = 1/N$ for all $i = 1 \ldots N$
Do for $t = 1, \ldots, T$,



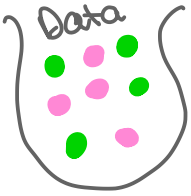
1. Train base learner according to example distribution $\mathbf{d}^{(t)}$ and obtain hypothesis $h_t : \mathbf{x} \mapsto \{\pm 1\}$.
2. compute weighted error $\epsilon_t = \sum_{i=1}^{N} d_i^{(t)} \mathrm{I}(y_i \neq h_t(\mathbf{x}_i))$
3. compute hypothesis weight $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
4. update example distribution

$$d_i^{(t+1)} = d_i^{(t)} \exp\left(-\alpha_t y_i h_t(\mathbf{x}_i)\right) / Z_t$$



Output: final hypothesis $f_{\text{Ens}}(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$

- In each round the prob. of drawing that data is changed
→ start with equal prob. for all data



draw with equal prob. then build your model

→ only in the 1st iteration: $d_i^{(1)} = \frac{1}{N}$ (prob. of drawing) and after it changes

- at 2: we see how good h_t was. If $h_t = -1$ when $y_t = +1$: then misclassification. ⇒ Error
- I counts how often the hypothesis h_t is wrong
- weight the error with the prob. of drawing x_i

finally put: $d \cdot y \cdot h$ to your exp and then normalize with Z to make a probability that sums to 1.

→ start with an even distribution
→ train the model
→ see where the model is wrong
→ sum up the error
→ depending on error amount, weight it
→ look at each datapoint and look how strong your h_t is

Exercise 2: AdaBoost as an Optimization Problem (25 + 25 P)

Consider AdaBoost for binary classification applied to some dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$. The algorithm starts with uniform weighting ($\forall_{i=1}^N : \; p_i^{(1)} = 1/N$) and performs the following iteration:

for $t = 1 \ldots T$:

Step 1: $\mathcal{D}, p^{(t)} \mapsto h_t$ (learn tth weak classifier using weighting $p^{(t)}$)

Step 2: $\epsilon_t = \mathbb{E}_{p^{(t)}}[1_{(h_t(x) \neq y)}]$ (compute the weighted error of the classifier)

Step 3: $\alpha_t = \frac{1}{2} \log\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$ (set its contribution to the boosted classifier)

Step 4: $\forall_{i=1}^N : \; p_i^{(t+1)} = Z_t^{-1} p_i^{(t)} \exp(-\alpha_t y_i h_t(x_i))$ (set a new weighting for the data)

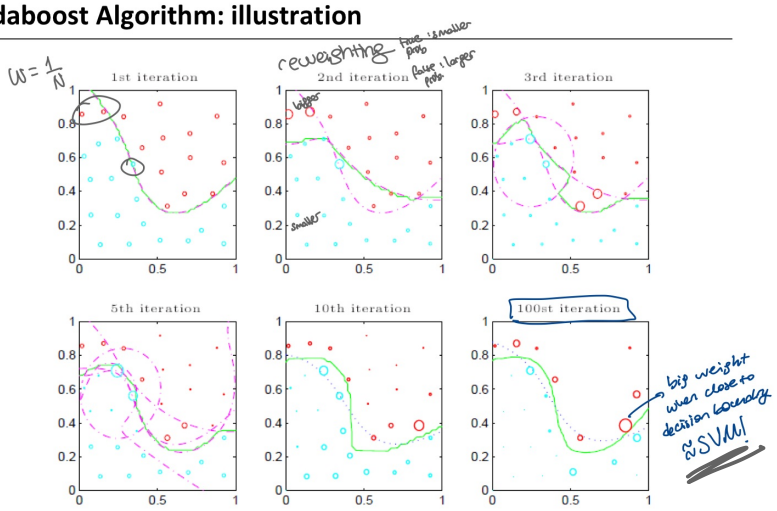
The term $\mathbb{E}_{p^{(t)}}[\cdot]$ denotes the expectation under the data weighting $p^{(t)}$, and Z_t is a normalization term. An interesting property of AdaBoost is that it can be shown to minimize some objective function

$$\mathcal{G}(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \exp(-y_i f_{\boldsymbol{\alpha},t}(x_i))$$

where $f_{\boldsymbol{\alpha},t}(x) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(x)$ is the output score of the boosted classifier after t iterations.

- wrong classified data gets higher prob. of drawing until it's correctly classified.

Adaboost Algorithm: illustration



- function gets complexer in each iteration
- wrong classified points are bigger
- most of the data becomes irrelevant we only care about datapoints close to the db / misclassified

outliers & SVMs are important

→ The distribution changes on each iteration and we focus more and more on hard-to-get data.

Error Function of Adaboost

- AdaBoost stepwise minimizes a function of $y_i f_{\boldsymbol{\alpha}}(x_i) = y_i \sum_t \alpha_t h_t(\mathbf{x}_i)$

$$\mathcal{G}(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \exp\{-y_i f_{\boldsymbol{\alpha}}(\mathbf{x}_i)\}$$

- The gradient of $\mathcal{G}(\boldsymbol{\alpha}^{(t)})$ gives exactly the example weights used for AdaBoost:

$$\frac{\partial \mathcal{G}(\boldsymbol{\alpha}^{(t)})}{\partial f(\mathbf{x}_i)} \sim \exp\{-y_i f_{\boldsymbol{\alpha}}(\mathbf{x}_i)\} \sim d_i^{(t+1)}$$

- The hypothesis coefficient α_t is chosen, such that $\mathcal{G}(\boldsymbol{\alpha}^{(t)})$ is minimized:

$$\alpha_t = \underset{\alpha_t \geq 0}{\operatorname{argmin}}\, \mathcal{G}(\boldsymbol{\alpha}^{(t)}) = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

- **AdaBoost is a gradient descent method to minimizes $\mathcal{G}(\boldsymbol{\alpha})$.**
  $\Longrightarrow$ Bregman Divergences (Entropy Projections, ...)
  $\Longrightarrow$ Coordinate Descent Methods & Column Generation

- Boosting does step-wise minimisation
→ we want that y_i and $\alpha_t h_t(x_i)$ has the same sign
- $G(\alpha)$ is an exponential margin function.

PAC Boosting – exponential convergence

upperbound on the training error:

Theorem 1 (Schapire et al. 1997) *Suppose AdaBoost generates hypotheses with weighted training errors $\epsilon_1, \ldots, \epsilon_T$. Then we have*

better than random guessing

$$\sum_{i=1}^{N} \mathrm{I}(y_i \neq \mathrm{sign}(f_{Ens}(\mathbf{x}_i))) \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)}$$

error of ensemble

→ this bound decrease expo- fast

If $\epsilon_t < \frac{1}{2} - \frac{1}{2}\gamma$ (for all $t = 1, \ldots, T$), then the training error will decrease exponentially fast, i.e. will be **zero** after only

binary guess error: $\frac{1}{2}$ ✓

use it for 2 classes only

$$\frac{2\log(N)}{\gamma^2} = \mathcal{O}(\log(N))$$

iterations.

- take an ensemble
- our error will be max 1/2 → yes/no
  our error is then smaller as:

$$\epsilon_t < \frac{1}{2} - \frac{1}{2}\gamma$$

γ is smarter than chance

- if we do that our training error will decrease exponentially fast to zero

weak learner: you have to be a bit smarter than luck

PAC Boosting – VC dimension of combined Hypothesis

Let d be the VC dimension of the base hypothesis class $\mathcal{H}$.
Then the VC dimension of the class of combined functions is

$$d_{Ens}(N, \gamma) = \mathcal{O}\Big(d \underbrace{\frac{\log(N)}{\gamma^2}}_{\sim T} \log\Big(\frac{\log(N)}{\gamma^2}\Big)\Big) = \mathcal{O}\left(d \log(N) \log^2(N)\right).$$

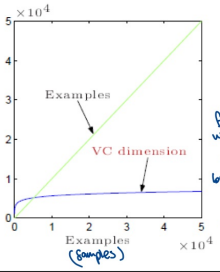
VC-dimensions of ensemble

→ to limit overfit

- the complexity doesn't go up even though we increase our ensemble because we are already at 0 training error.

An Example

- VC dimension $d = 2$ (e.g. decision stumps)
- $\epsilon_t \leq 0.4 = \frac{1}{2} - \frac{1}{2}\gamma$
  $\Rightarrow \gamma \geq 0.2$



we increase the VC-dimensions

fct. doesn't get complex with increasing samples

(VC) but it will converge, grows very little

→ samples
→ VC-dimen.

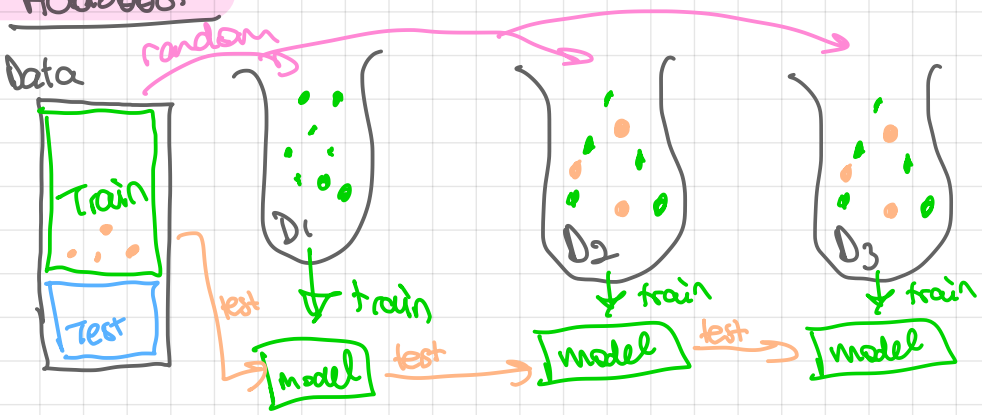
VC describes the overall complexity by shattering.

PAC Boosting – Digestion

- properties of weak learner imply exponential convergence to a consistent hypothesis
- Fast convergence ensures small VC dimension of the combined hypothesis → because you don't need many ensembles
- small VC implies small deviation from the empirical risk → because each learner can be very simple
- for any $\varepsilon > 0$ and $\delta > 0$ exists a sample size N, such that with probability $1 - \delta$ the expected risk is smaller than ε
- Any weak learner can be boosted to achieve an arbitrary high accuracy! ($\rightsquigarrow$ strong learner)

boost to achieve arbitrary high accuracy

Adaboost



1. select randomly from training data
2. take all training data and use it to test the model.
3. some points will be not well predicted → error
4. for D2: choose again randomly, but each instance is weighted according to the error of D1.
   So points with significant error are more likely to be chosen

Bagging: choosing some subset of the data at random with replacement and create each bag in the same way.

Boosting: add-on to bagging, where in subsequent bags we choose those data that has been modeled poorly in the overall system before.

→ both methods build on ensemble of models

Bagging:
- builds them parallel
- less training time
- classifiers are equally weighted

Boosting:
- sequentially
- models are weighted in the ensemble based on their training performance

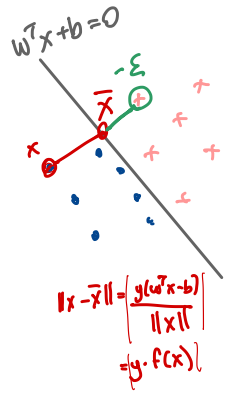
→ both reduce the variance
but: bagging does not have bias reduction

- Boosting very likely to overfit

→ Margin Distributions

Show for any datapoint that the nearest point on margin: $x - \bar{x}$ is basically $\frac{y \cdot w^T x + b}{\|w\|}$

do it for every datapoint as a signed distance



- Function set used in boosting: Convex Hull of $\mathcal{H}$

$$S := \left\{ f : \mathbf{x} \mapsto \sum_{h \in \mathcal{H}} \alpha_h h(\mathbf{x}) \;\middle|\; \alpha_h \geq 0, \sum_{h \in \mathcal{H}} \alpha_h = 1 \right\}$$

- the α's are the parameters
- Find a hyperplane in the Feature Space spanned by the hypotheses set $\mathcal{H} = \{h_1, h_2, \ldots\}$
- Margin ρ for an example $(\mathbf{x}_i, y_i)$ by — proportional to the margin ⇒ SVM

$$\rho_i(\boldsymbol{\alpha}) := y_i f_{\text{Ens}}(\mathbf{x}_i) = y_i \sum_{t=1}^{T} \frac{\alpha_t}{\sum_t \alpha_t} h_t(\mathbf{x}_i)$$

(proportional; prediction of ensemble; → normalized α; rewriting of hypothesis)

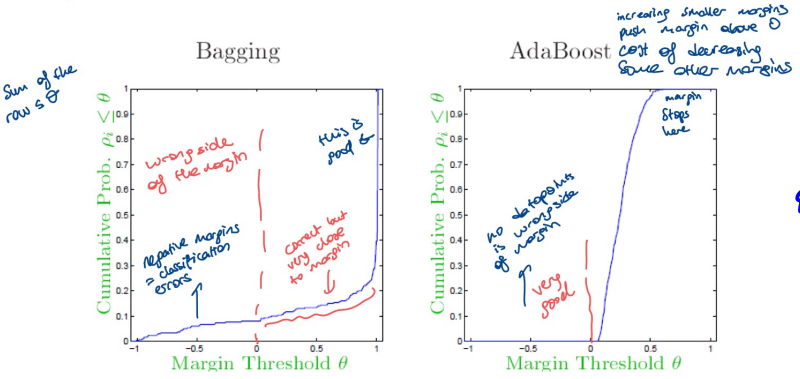
- Margin ϱ for a function f_{Ens} by $\varrho(\boldsymbol{\alpha}) := \min_{i=1,\ldots,N} \rho_i(\boldsymbol{\alpha})$

SVM: look at all datapoints find smallest distance to dh ⇒ margin

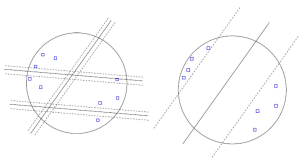
→ if the margin ρ_i is very large, then the model classified correctly
→ margin of the whole Ensemble function is the min one

Margin Distributions - illustration

AdaBoost tends to increase small margins, while decreasing large margins



small margin → many ways of shattering data
large margin → not so many ways



Lower Bounding the margin

Theorem 2 *Suppose the base learning algorithm generates hypothesis with weighted training errors $\epsilon_1, \ldots, \epsilon_T$. Then we have for any θ* (can also be bounded by: margin is below threshold θ)

$$P_{\mathbf{Z}}(y f_{Ens}(\mathbf{x}) \leq \theta) \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t{}^{1-\theta}(1-\epsilon_t)^{1+\theta}}$$

(prob. that margin below threshold)

Corollary 3 *If the base learning algorithm always achives $\epsilon_t \leq \frac{1}{2} - \frac{1}{2}\gamma$ then AdaBoost will generate a combined hyperplane with margin at least $\frac{1}{2}\gamma$.*

→ prob. of finding a θ that is smaller than our margin is exp. small

if each bayes error is better than guessing ($\frac{1}{2}$)
⇒ fct. of our error gives a bound on the margin → use it to bound test error

SVM vs. Boosting

- SVMs

$$R[f] \leq R_{emp}[f] + \mathcal{O}\left(\sqrt{\frac{\log\left(N\theta^2\right)}{\theta^2 N} + \frac{\log(1/\eta)}{N}}\right).$$

- Boosting

$$R[f] \leq R^{\theta}_{emp}[f] + \mathcal{O}\left(\sqrt{\frac{d\log^2\left(\frac{N}{d}\right)}{\theta^2 N} + \frac{\log(1/\delta)}{N}}\right)$$

depends on VC dimension of Bayes

- independent of the dimensionality of the space!

margin θ is part of emp. Risk

→ both of them in terms of margins are the same:

large margin : small term at $+\mathcal{O}(\sqrt{\ldots})$

small VC-Dimension of the hypothesis : small $+\mathcal{O}(\sqrt{\ldots})$

Boosting the Limit

AdaBoost is a coordinate gradient descent method which minimizes $\mathcal{G}(\alpha)$ stepwise.

What happens in the long run?

- Explicit expression for $d_i^{(t+1)}$:

$$d_i^{(t+1)} = \frac{\exp\left\{-\rho_i(\boldsymbol{\alpha}^{(t)})\right\}^{\|\boldsymbol{\alpha}^{(t)}\|}}{\sum_{j=1}^{N}\exp\left\{-\rho_j(\boldsymbol{\alpha}^{(t)})\right\}^{\|\boldsymbol{\alpha}^{(t)}\|}}$$

→ with every iteration, d will grow since vector gets longer: the 1-norm will increase monotonically.

$\rightsquigarrow$ **Soft-Max Function** with parameter $\|\boldsymbol{\alpha}^{(t)}\|_1$

- $\|\boldsymbol{\alpha}\|_1$ will increase monotonicaly ($\sim$ linear)

$\rightsquigarrow$ the d's concentrate on a **few** difficult patterns

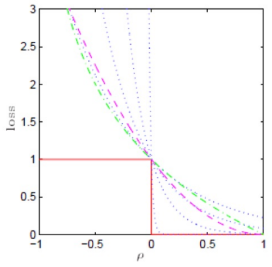
→ **Support Patters**

$\rightsquigarrow$ Annealing Process:

$$\mathcal{G}(\boldsymbol{\alpha}) = \sum \exp\left\{-\rho_i(\boldsymbol{\alpha})\right\}^{\|\boldsymbol{\alpha}\|_1}$$

→ $0/\infty$-Loss approximated asymptoticaly

→ **Barrier Optimization**



SVM's vs. Boosting

→ Adaboost is very similar to SVMs: they both focus on hard to learn data.

Mathematical Program Formulation- SVMs

The SVM minimization of

$$\begin{aligned} \min_{\mathbf{w}\in\mathcal{F}_\Phi} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i\langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle \geq 1, \quad i = 1,\ldots,N. \end{aligned} \tag{1}$$

reformulate as maximization of the margin ρ

$$\begin{aligned} \max_{\mathbf{w}\in\mathcal{F}_\Phi,\rho\in\mathbf{R}_+} \quad & \rho \\ \text{subject to} \quad & y_i\sum_{j=1}^{D} w_j\Phi_j(\mathbf{x}_i) \geq \rho \quad \text{for} \quad i = 1,\ldots,N \\ & \|\mathbf{w}\|_2 = 1, \end{aligned} \tag{2}$$

where $D = \dim(\mathcal{F})$ and Φ_j is the j-th component of Φ in feature space:

$$\Phi_j = P_j[\Phi]$$

SVM as a margin maximisation of some projected components

we'd like to have all margins $\geq \rho$
push every individual norm to max
normalize $\|w\|_2$ as 1.

quadratic: $\|w\|_2 = 1$

- Boosting also as a max margin problem
- tries to solve a linear program because: $\|w\|_1 = 1$

Boosting as a Mathematical Program

- master hypothesis

$$f(\mathbf{x}) = \sum_{t=1}^{T}\frac{w_t}{\|\mathbf{w}\|_1}h_t(\mathbf{x})$$

- base hypotheses h_t produced by the base learning algorithm.

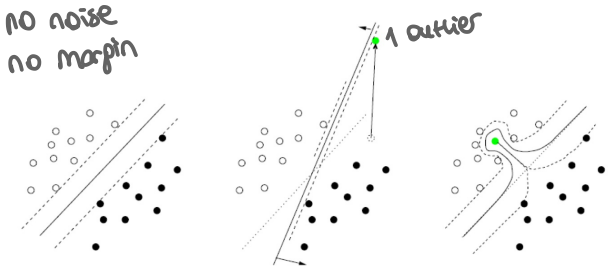
Arc-GV solution is asymptotically the same as linear program solution, maximizing smallest margin ρ:

$$\begin{aligned} \max_{\mathbf{w}\in\mathbf{R}^J,\rho\in\mathbf{R}_+} \quad & \rho \\ \text{subject to} \quad & y_i\sum_{j=1}^{J} w_j h_j(\mathbf{x}_i) \geq \rho \quad \text{for} \quad i = 1,\ldots,N \\ & \|\mathbf{w}\|_1 = 1, \end{aligned} \tag{3}$$

where J is the number of hypotheses in $\mathcal{H}$.

→ Soft Margins

Hard Margin Classification



- The problem of finding a maximum margin "hyper-plane" on reliable data (left), data with outlier (middle) and a mislabeled pattern (right). The hard margin implies **noise sensitivity**.

- hard margin does not have slack variables (exceptions), so it's very sensitive to outliers

→ so we use soft margin SVM to make it easier.

Adaboost with Soft Margins

- Define a Soft Margin

$$\tilde{\rho}_n(\boldsymbol{\alpha}) = \rho_n(\boldsymbol{\alpha}) + \zeta_n,$$

  - where ζ_n is the amount of uncertainty in example $(\mathbf{x}_n, y_n)$

- Once we have defined the uncertainity measure ζ_n, we can easily get a new regularized Boosting algorithm.

  $\Rightarrow$ Improve the Error Function by plugging-in the Soft Margin

$$\tilde{G}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \exp\left\{-\|\boldsymbol{\alpha}\|_1 \tilde{\rho}_n(\boldsymbol{\alpha})\right\}$$

$$d_n^{t+1} = \frac{\partial \tilde{G}(\boldsymbol{\alpha})}{\partial f_{\boldsymbol{\alpha}}(\mathbf{x}_n))}$$

$$\alpha_t = \underset{\alpha_t \geq 0}{\operatorname{argmin}}\, \tilde{G}(\boldsymbol{\alpha})$$

Regularizing Adaboost

Positive:

- first algorithm that addresses overfitting in Boostin
- much improved results

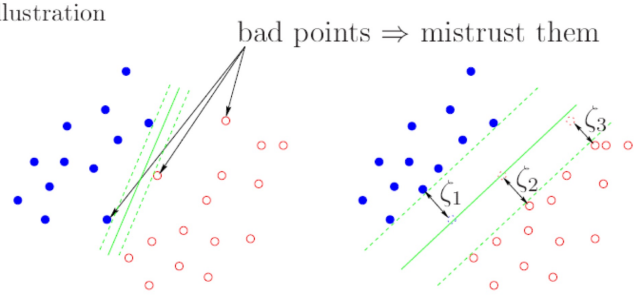
Negative:

- Modification on algorithmic level
- hard to analyze
  - which optimization problem is solved
  - no generalization results

Idea:

- go back to beginning and redesign optimization problem
- use convergence results for leveraging
- apply margin bounds

Illustration

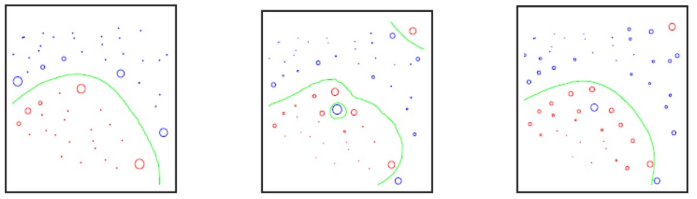


Regularizing Adaboost – Reducing the *Influence*

- How can we know which patterns are unreliable?

  AdaBoost focuses on difficult-to-learn patterns by assigning high pattern weights d_n that we can exploit. Hence, we define the Influence of a pattern

$$\mu_n^t = \sum_{r=1}^{t} \frac{\alpha_r}{\|\boldsymbol{\alpha}\|_1} d_n^r \qquad \zeta_n = C(\mu_n^t)^2$$



AdaBoost without "Noise" | AdaBoost with "Noise" | AdaBoost$_{Reg}$ with "Noise"

- Boosting algorithms
  - AdaBoost
  - PAC Motivation
  - Boosting with Large Margins
  - Strategies for Dealing with High Dimensional Spaces
  - Relations to Mathematical Programming & SVMs

Lecture 12: Clustering

→ k-means clustering
→ Gaussian Mixture Model
→ Expectation Maximisation:
- ML-bound for later variables
- lower bound on log-likelihood

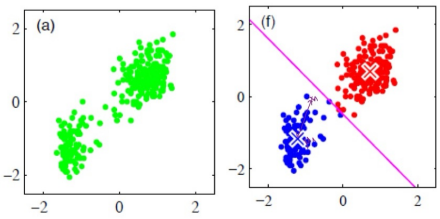
Clustering: unsupervised
no labels
to understand the underlying data structure
would be nice to find a Gaussian distribution
no labels, but we try to seperate the data

Given N d-dimensional datapoints $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ (no labels),

Data comes from gaussian like clusters

→ How good is the cluster?
→ How do we measure similarity?
- assume data is uniform, homogenous

find the structure of bunch of data points in high dimensional space



partition data into K disjoint sets $\mathcal{S}_k$ based on similarity.

$$\forall k \forall l \; \mathcal{S}_k \cap \mathcal{S}_l = \emptyset, \quad \bigcup_{k=1}^{K} \mathcal{S}_k = X$$

sets shouldn't overlap

union of our sets is the whole data

K-Means

μ: center of the cluster

Define clusters by minimum Euclidean distance to cluster mean.

→ Find $\theta = \{\mathcal{S}_1, \ldots, \mathcal{S}_K\}$ that minimize

squared distance of data point - respective cluster mean

$$J(\theta) = \sum_{k=1}^{K} \sum_{\mathbf{x}_n \in \mathcal{S}_k} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2, \text{ where } \boldsymbol{\mu}_k = \frac{1}{|\mathcal{S}_k|} \sum_{\mathbf{x}_n \in \mathcal{S}_k} \mathbf{x}_n$$

How much is the scatter around the mean?

normalizing

▸ How do we measure similarity?
euclidian distance

▸ How do we judge similarity?
distance all data points of that cluster x_n to their cluster center μ_k

- little variance around each cluster mean

Algorithm ("Expectation Maximization")

1. Choose K random points as initial cluster centers $\boldsymbol{\mu}_1^{(0)}, \ldots, \boldsymbol{\mu}_K^{(0)}$

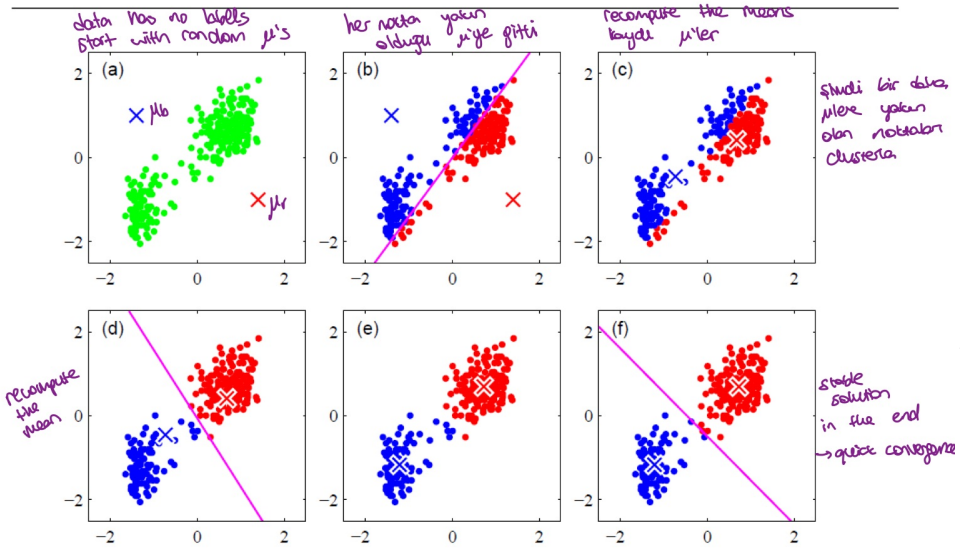
1) choose random μ's
2) expectation step: everything closest to the cener goes to that center
3) assign the data points now to the chosen cener & compare the μ's again → update mean
4) do it again to find local min

2. **Assignment (E):** closest center
$$\mathcal{S}_k^{(t)} = \left\{\mathbf{x}_n : \left\|\mathbf{x}_n - \boldsymbol{\mu}_k^{(t)}\right\|^2 \leq \left\|\mathbf{x}_n - \boldsymbol{\mu}_l^{(t)}\right\|^2 \; \forall l, 1 \leq l \leq K\right\}$$
3. **Update (M):** $\boldsymbol{\mu}_k^{(t+1)} = \frac{1}{|\mathcal{S}_k^{(t)}|} \sum_{\mathbf{x}_n \in \mathcal{S}_k^{(t)}} \mathbf{x}_n$
4. Iterate 2. and 3. until convergence to local minimum

you can't get global min from k-means

Ex: 100 datapoints, 5 clusters → very hard to solve with enumeration 10^{67}
instead: do expectation maximization
- choose random μ's → choose K points
- find the closest center to that μ → Assignment (E)
- compute the mean again → Update (M)
- do the assignment step again and again until local minimum is reached

K-Means



assignment ↑
recompute the mean
until convergence

→ very fast, if data not large.

How do you choose the number of clusters? Open problem no labels hard to model selection

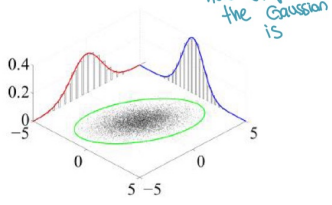
How to choose the 1st μ? Random, no right way

Density Estimation

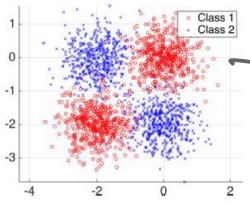
Multivariate Gaussian:

$$p(\mathbf{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$$
$$\sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

how ellipsoid the Gaussian is



bunch of Gaussians



→ 1 Gaussian is not enough

Idea: take bunch of Gaussians and weight them.

scaling

Does not always model data (e.g. class-conditional densities) well.

Gaussian Mixture Model (GMM)

weight of Gaussians

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \tau_k p_k(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k)$$

$$p_k(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k) \sim \mathcal{N}(\boldsymbol{\mu}_k, \Sigma_k)$$

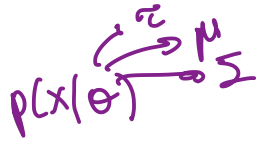
τ_k : scaling or "prior" of $p(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k)$

$$\sum_{k=1}^{K} \tau_k = 1$$ normalize

parameter vector θ : $\boldsymbol{\theta} = \{\underbrace{\tau_1, \ldots, \tau_K}_{\text{how they are scaled}}, \underbrace{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K}_{\text{means}}, \underbrace{\Sigma_1, \ldots, \Sigma_K}_{\text{covs of all Gaussians}}\}$

τ_k is our scaling factor

- we use GMM to do complex probability distributions
- GMM contains a parameter vector that has the means and covariances of all the Gaussians + how they are superimposed (τ_k)

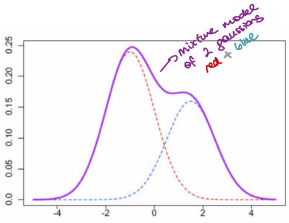


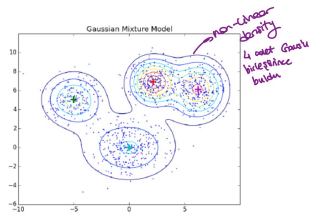
Clustering is essentially finding & describing bunch of blobs and finding their structure.

Gaussian Mixture models are universal density approximators.

→ ∞ many $P(x|\theta)$ can model anything, because it is summed up from smaller Gaussians.

→ As a byproduct they provide a clustering solution.





Fitting a GMM using EM

GMM is probabilistic & generative model

Trick: introduce auxiliary variables indicating the membership of each sample to a Gaussian

$$\mathbf{z}_1, \ldots, \mathbf{z}_N \in \mathbb{R}^K \sim \textit{Categorical}\,(\boldsymbol{\tau})$$

if the nth data point belongs to the kth cluster

$$p(z_{nk} = 1) = \tau_k$$

$$\forall n \; \exists! k \quad z_{nk} = 1, \; z_{nj, j \neq k} = 0$$

$$\text{e.g. } \mathbf{z}_n = (0, 0, 0, \underset{\text{nth data point, kth cluster}}{1}, 0, \ldots, 0)^\top$$

tells the cluster

→ with this we don't need to sample from the huge distribution but we can choose the k-th Gaussian we want

Note: this is also how to sample from a GMM

1. Sample $\mathbf{z} \sim \textit{Categorical}\,(\boldsymbol{\tau})$ from which of the gaussians?
2. Sample $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_k, \Sigma_k)$, where $z_k = 1$ → sampling from the respective distribution

Fitting a GMM using EM: algorithm

1. Initialize t=0, $\boldsymbol{\theta}^{(0)} = \{\tau_1, \ldots, \tau_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \Sigma_1, \ldots, \Sigma_K\}$
   (e.g., $\tau_k^{(0)} = 1/K, \Sigma_k^{(0)} = I, \boldsymbol{\mu}_k^{(0)} = \text{rand}$)
   how many times the algorithm runs
2. **Expectation:** compute membership probabilities given $\boldsymbol{\theta}^{(t)}$
   2. notation: n points & clusters

$$q^{(t)}(z_{nk}) := p(z_{nk} = 1 | \mathbf{x}_n, \boldsymbol{\theta}^{(t)}) \overset{\text{Bayes}}{=} \frac{p(\mathbf{x}_n | z_{nk}, \boldsymbol{\theta}^{(t)}) p(z_{nk}, \boldsymbol{\theta}^{(t)})}{p(\mathbf{x}_n | \boldsymbol{\theta}^{(t)})}$$

$$= \frac{\tau_k^{(t)} p_k(\mathbf{x}_n | \boldsymbol{\mu}_k^{(t)}, \Sigma_k^{(t)})}{\sum_{l=1}^K \tau_l^{(t)} p_l(\mathbf{x}_n | \boldsymbol{\mu}_l^{(t)}, \Sigma_l^{(t)})}$$

datapoint n belongs to cluster k given data

3. **Maximization:** update $\boldsymbol{\theta}$ given (soft) cluster assignments

expectation value

$$\tau_k^{(t+1)} = \frac{1}{N} \sum_{n=1}^N q^{(t)}(z_{nk}) \qquad \boldsymbol{\mu}_k^{(t+1)} = 1/N\tau_k^{(t+1)} \sum_{n=1}^N q^{(t)}(z_{nk}) \mathbf{x}_n$$

new means

weighted sum for the means

$$\Sigma_k^{(t+1)} = 1/N\tau_k^{(t+1)} \sum_{n=1}^N q^{(t)}(z_{nk}) \left(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)}\right) \left(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)}\right)^\top$$

lump parameters: τ, μ, I

variance $\Sigma_k = I$: unit (no ellipsoids allowed)

μ_k = random

- prob. of sampling from a certain Gaussian is $\tau_k = 1/K$ → uniform distribution

Fitting a GMM using EM: algorithm

Note: also possible to use hard cluster assignments.

1. **Expectation:**

hard assignments: Expectation is the max of all possible things

max over all possible things

$$q^{(t)}(z_{nk}) = p(z_{nk} = 1 | \mathbf{x}_n, \boldsymbol{\theta}^{(t)})$$

$$z_{nk}^{(t)} = \begin{cases} 1 & \text{if } q^{(t)}(z_{nk}) = \max_l q^{(t)}(z_{nl}) \\ 0 & \text{otherwise} \end{cases}$$

2. **Maximization:** update $\boldsymbol{\theta}$ given hard cluster assignments

just summing

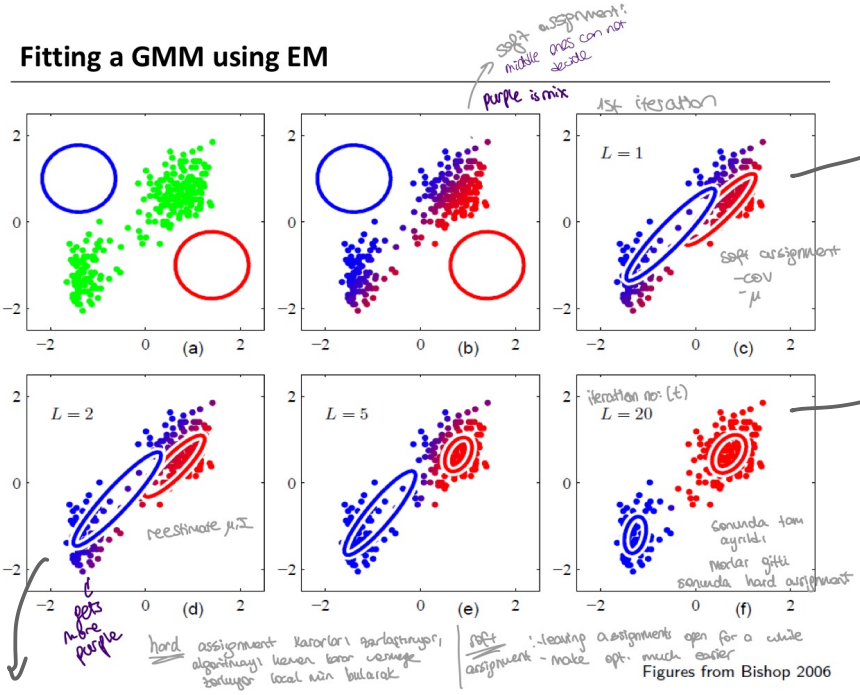
$$\tau_k^{(t+1)} = \frac{1}{N} \sum_{n=1}^N z_{nk}^{(t)} \qquad \boldsymbol{\mu}_k^{(t+1)} = 1/N\tau_k^{(t+1)} \sum_{n=1}^N z_{nk}^{(t)} \mathbf{x}_n$$

often you hard sum'da

$$\Sigma_k^{(t+1)} = 1/N\tau_k^{(t+1)} \sum_{n=1}^N z_{nk}^{(t)} \left(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)}\right) \left(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)}\right)^\top$$

limit of soft assignment is hard assignment

Fitting a GMM using EM



Figures from Bishop 2006

- signs are not that hard
  - compute cov and μ's
  - L: # iterations (t)
- final clusters in full color
  - solid cov-structure
  - hard assignments

not hard blue

- classes aren't that easy to seperate. Having a hard assignment will make the algorithm push to a local minimum. But if we have a soft assignment we're leaving assignments open for a while and that makes the optimisation numerically easier.

→ K-means assumes that classes have similar size. If the class sizes are not so similar, we'll get suboptimal solutions.

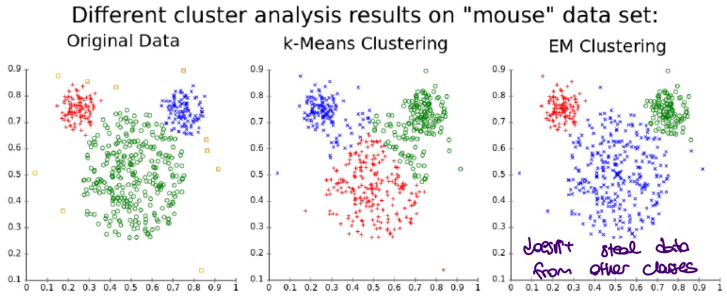
Assignment (latent variables fix) → Estimation parameters (μ, σ^2)

latent part: where does the data belong to is unknown
first optimize randomly
then estimate latent variables classes
then given that the assignment was right, assign parameters to get μ's and σ²'s.
fixing the μ's and σ's we can do again an assignment
again do μ's & σ's...

GMM vs. K-means clustering

In contrast to K-means, GMM allows for

- Unequal cluster variances
- Unequal cluster probabilities
- Non-spherical clusters
- Soft cluster assignment



→ GMM is more okay to different class sizes
→ K-means not ok for different class size

→ Fitting a GMM with Max-Likelihood

- Change θ such that likelihood of generating x is maximum $\Rightarrow p(x|\theta)$

General EM Algorithm

EM = Expectation Maximisation

- universal tool to apply for latent variables
- z (latent variables) tell us the underlying structure of data

Maximum Likelihood for latent variable models

z latent (unobserved variables)
X observed data
θ model parameters

We want to maximize the likelihood of the observed data (= incomplete-data likelihood), $L(\theta|X) = p(X|\theta)$:

$$\hat{\theta} = \arg\max_\theta \log\left[p(X|\theta)\right] = \arg\max_\theta \log\left[\sum_{z\in\mathcal{Z}} p(X,z|\theta)\right].$$

- max log-likelihood w.r.t. model variables
- we are transparent that there's latent variables

Maximizing this directly is difficult because of $\log\sum\ldots$. headache

On the other hand, it is often easy to optimize the complete-data likelihood, $L(\theta|X,z) = \log p(X,z|\theta)$.

Trick: get rid of log Σ:

Example: GMM

Incomplete-data log-likelihood

$$\log\left[p(X|\theta)\right] = \sum_{n=1}^{N} \log\left[\sum_{k=1}^{K} \tau_k p_k(\mathbf{x}_n|\boldsymbol{\mu}_k, \Sigma_k)\right]$$

z_{nk}: nth datapoint is in the kth cluster

Complete-data log-likelihood

we don't know z — latent variable — delta function = 1 if nth datapoint is in kth cluster

$$\log\left[p(X,z|\theta)\right] = \sum_{n=1}^{N} \log\left[\sum_{k=1}^{K} \delta_{z_{nk}=1} \tau_k p_k(\mathbf{x}_n|\boldsymbol{\mu}_k, \Sigma_k)\right]$$

→ Analytic ML estimate for each $\theta_k = (\tau_k, \boldsymbol{\mu}_k, \Sigma_k)$

Problem: We don't know z.

EM Algorithm

z is unknown, so estimate jointly with θ

Expectation Maximization algorithm:

- Iterate between updates of hidden variables and parameters
- **Theory:** updates are defined in a way such that $p(X|\theta)$ increases in each step

→ Guaranteed to find local maximum of $p(X|\theta)$
(hard to find global maximum if $p(X|\theta)$ is non-concave)

- **Technically:** optimize a lower bound on $p(X|\theta)$ and subsequently improve bound

Split the optimization to:
- 1 part is taking care of unknown latent variables
- other part " rest of parameters (τ, μ, Σ)

- update the Max-Likelihood in a way that log-likelihood increases

→ Jensens Inequality

Instead of optimising (max.) the function itself, maximize the lower bound to the function

→ optimize a lower bound on $p(X|\theta)$ and subsequently improve bound

Jensen's Inequality

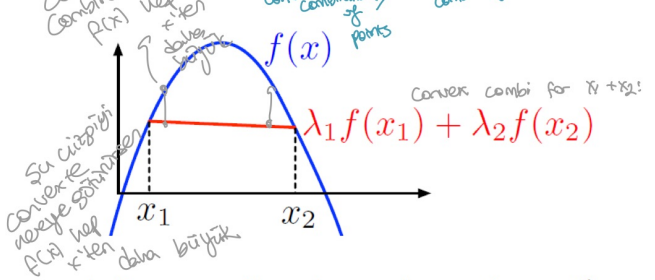
For any convex combination $\lambda_1, \ldots, \lambda_I, \quad \lambda_i \geq 0, \quad \sum_{i=1}^{I} \lambda_i = 1$

and any concave function f:

$$f\left(\sum_{i=1}^{I} \lambda_i x_i\right) \geq \sum_{i=1}^{I} \lambda_i f(x_i) \; .$$

convex combination of points

convex comb. of function values



(conversely for convex f, analogous for continuous f)

A lower bound on the log-likelihood

θ: a parameter setting

$q(z)$: a probability mass function of choice on z

$$\begin{aligned}
\log p(X|\theta) &= \log \sum_z p(X, z|\theta) \\
&= \log \sum_z q(z) \left[\frac{p(X, z|\theta)}{q(z)}\right] \\
\text{Jensen's inequality} \quad &\downarrow \quad \text{(remember that } \sum_z q(z) = 1 \text{, log concave)} \\
&\geq \sum_z \underbrace{q(z)}_{\lambda_i} \underbrace{\log}_{f(.)} \underbrace{\left[\frac{p(X, z|\theta)}{q(z)}\right]}_{x_i} \\
&=: F(q(z), \theta)
\end{aligned}$$

This lower bound is much easier to optimize ($\log \sum$ vs. $\sum \log$)

Expectation Maximization

True objective: maximize the data log likelihood w.r.t. θ

$$\hat{\theta} = \arg\max_{\theta} \log p(X|\theta) \; .$$

This is difficult.

EM objective: maximize, w.r.t. q and θ, the lower bound

$$\hat{q}, \hat{\theta} = \arg\max_{q,\theta} F(q(z), \theta) \; .$$

Maximization of lower bound

$$F(q(z), \theta) = \sum_z q(z) \log \left[\frac{p(X, z|\theta)}{q(z)}\right]$$

There are two ways to improve the lower bound:

1. **Expectation: improve** $q(z)$ for given θ
2. Maximization: improve θ for given $q(z)$

1. How to select the $q(z)$?

Take the difference between what we actually want to optimize and its lower bound. How large is this difference? Can we minimize that?

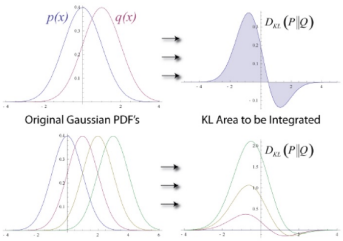
we wanna actually optimize: its lower bound!

$$\log p(X|\theta) - \sum_z q(z) \log \left[\frac{p(X,z|\theta)}{q(z)}\right] = KL(q(z) \,||\, p(z|x,\theta))$$

data log-likelihood − lower bound

KL-Divergence

$$KL(P||Q) = \sum_x P(x) \log \frac{Q(x)}{P(x)} = \mathbb{E}_x \left[\log \frac{Q(x)}{P(x)}\right] \geq 0$$



- Measures the distance between two distributions P and Q
- Not a true metric (not symmetric, no triangle inequality)
- Important quantity in information theory

- a way of comparing prob. distributions
- positive quantity
- difference metric
- ideally we want the KL-divergence between data log likelihood and lower bound 0 so that they are the same.

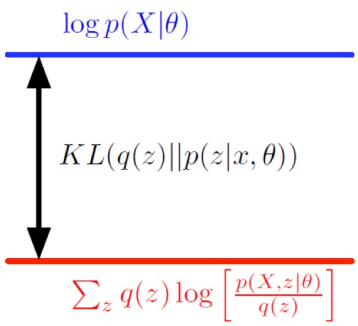
Expectation: improving q(z)

improve q (yikes!)

$$KL(q(z)||p(z|x,\theta)) = 0 \iff \underbrace{q(z)}_{\text{assignment}} = p(z|X,\theta)$$

$\Rightarrow$ Lower bound is strict if $q(z) = p(z|X,\theta)$.

Expectation step: set $q^{(t)}(z) = p(z|X,\theta^{(t)})$ → here optimal blue and red lines will be together



2. Now improve θ:

Goal: maximize $F(q(z),\theta)$ w.r.t. θ.

$$\theta^* = \arg\max_\theta \sum_z q(z) \log \left[\frac{p(X,z|\theta)}{q(z)}\right]$$

$$= \arg\max_\theta \sum_z q(z) \log p(X,z|\theta) \underbrace{- \sum_z q(z) \log q(z)}_{\text{"Entropy" } H(q)}$$

q is fix

Some constant, doesn't depend on θ.

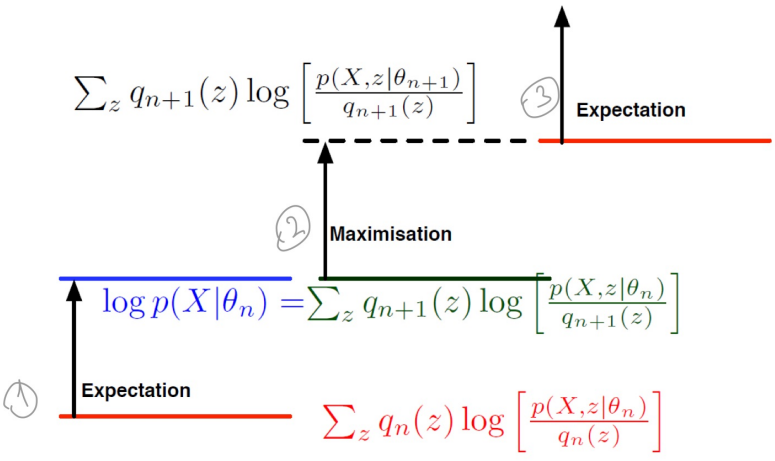
$$= \arg\max_\theta \sum_z q(z) \log p(X,z|\theta)$$

Approach: set gradient to zero . . .

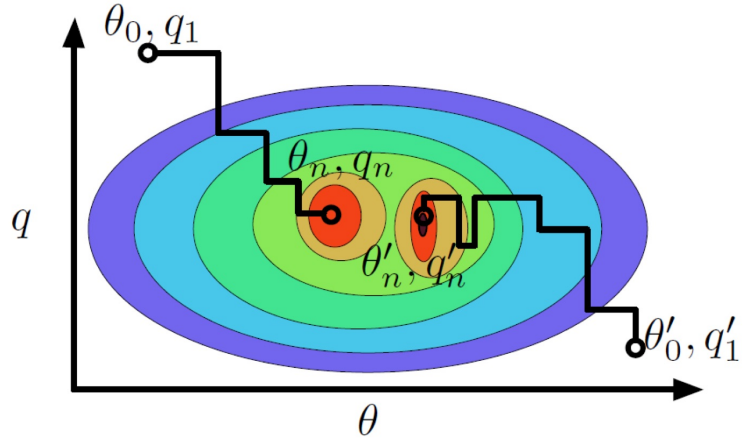
Typically, easy (analytic) solution, due to $\sum \log$ rather than $\log \sum$.

→ every little step brings us closer to the maximum

Iterative Optimization



Convergence to local maximum



EM summary

z latent (unobserved variables)
X observed data
θ model parameters

1. Initialize $\theta^{(0)} = \text{rand}$
2. Expectation: $q^{(t)}(z) = p(z|X, \theta^{(t)})$ assignment
3. Maximization: $\theta^{(t+1)} = \arg\max_\theta \sum_z q^{(t)}(z) \log p(X, z|\theta)$
   ↳ reestimating μ and Σ
4. Iterate until convergence local min als Ergebnis

Summary

- EM is a "meta-algorithm" for obtaining local ML estimates
- Also applicable to maximum a-posteriori (MAP) estimation
- Particularly useful in models with latent variables z, where optimizing the incomplete-data likelihood directly is hard, but optimizing the complete-data likelihood $p(X, z|\theta)$ is easy.

→ Alternate between estimating z and θ

- Can be *applied to* to a GMM, but EM is not equal to a GMM
- Other applications:
  - Hidden Markov Models (Baum-Welch algorithm)
  - Missing/incomplete data
  - Only summary data observed

- Clustering basically measures similarity and judges the solution
- Optimisation of how to find the clustering solution?
  iterative, otherwise it would be expensive because combinatorial
- k-means or Gmm
- k-means is like a meta-algo because we optimise our error function and find the local max/min
- EM is a one big algorithm that we can apply to many models (Gmm included)

Properties of EM

Pro

- No stepsize/learning rate
- Each iteration improves likelihood

Con

- "Only" local minima found
- Solution dependent on initialization
- Can be slow

Note: sometimes possible to use generic solvers (e.g. Newton)

But:

- Complicated gradients, update rules
- No improvement guarantee (e.g., Jensen requires densities) KDL

Lecture 13: Kernel Ridge Regression

Classification

map vectors to output $\{-1, 1\}$
predict classes
data generated from 2 classes
learn a decision fct. to map observations
Goal: predict class labels
LDA, SVM

Regression

different values
map datapoint x to real value
price prediction, weather...
above / below a certain threshold
Goal: predict continious function value

→ Linear Regression

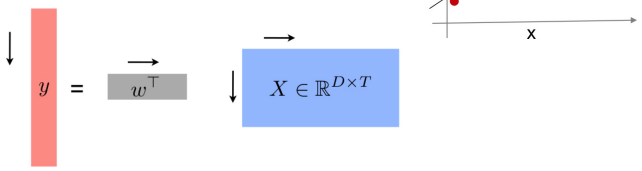
Linear Regression

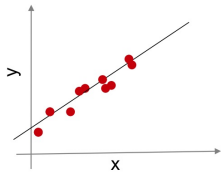
Problem:

Given a set of samples $(x_1, y_1), \ldots, (x_T, y_T)$ predict the outcome y for a new input x

Model:

$$\hat{y} = \mathbf{w}^\top \mathbf{x}$$





For optimal w:
minimize the loss between true output y and prediction:

$$\min E(w), \quad E_{lsq}(w) = \sum_{t=1}^{T} (y_t - w^\top x_t)^2$$

Least-Square-Error

How to minimize LSE function?

$$E_{lsq}(w) = (y - w^\top X)^2 = y^\top y - 2 y^\top w^\top X + w^\top X X^\top w$$

→ take derivative w.r.t. w:

$$\frac{\partial E_{lsq}(w)}{\partial w} = -2Xy + 2XX^\top w$$

set derivative to zero:

$$2XX^\top w = 2Xy$$

$$w = \underbrace{(XX^\top)^{-1}}_{\text{Cov-matrix}} \cdot Xy$$

→ However, Linear Regression is not robust to outliers

→ Ridge Regression

- has a regularizer (squared Euclidian norm of w) with λ in loss function to control the complexity of the function.

→ Whole idea comes from SRM! If we have a different set of solution and if each of them correctly fit the data, then it is preferable to chose the smallest one because the smallest one is more likely to generalize well.

- RR adds a penalty term that shrinks the regression coefficients towards zero, reducing the model's complexity and thus reducing the risk of overfitting.

How to find the optimal w in RR?

$$E_{RR}(w) = (y - w^T X)^2 + \lambda \|w\|_2^2$$

→ Regularization term in RR

$$\frac{\partial E_{RR}(w)}{\partial w} = -2Xy + 2XX^T w + \lambda 2w \stackrel{!}{=} 0$$

$$2w(\lambda + XX^T) = 2Xy$$

$$\boxed{w = (XX^T + \lambda . I)^{-1} . Xy}$$

Parameter λ:

→ determines the influence of the regularizer:

- if λ big, then the values of w will go towards zero (shrinkage)
- $\lambda = 0$: we recover the linear model

higher $\lambda \uparrow$, flatter the predicted function

High values of λ is good for noisy high-dimensional data.

- if λ large: no small EVs.

→ Kernel Ridge Regression

nonlinear

- map data to a higher dimension and do the regression there if the relationship between input and output non linear: $X \rightarrow \phi(X)$ mapping

Non-linear Ridge Regression

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

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

How to find the optimal ϕ? → Kernel Trick:

Optimal solution has to lie in the space spanned by the data.

New predictions of arbitrarily complex categorizations require **only comparisons with previously learned examples**

the Kernel Trick

The prediction $\hat{y}$ given a new data point x can be expressed as linear combination of data points x_i

$$\hat{y} = f(\mathbf{x}) = \sum_{i=1}^{N} k_\phi(\mathbf{x}, \mathbf{x}_i)\, \alpha_i$$

where $i = 1,\ldots,N$ are the indices of previously seen data points.

The **kernel function** k corresponds to an inner product in feature space an measures the *similarity between two data points*

$$k_\phi(\mathbf{x}, \mathbf{x}_i) = \langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle_\phi = \phi(\mathbf{x})^\top \phi(\mathbf{x}_i)$$

unseen seen

Rewrite nonlinear ridge regression model again **in more detail**

$$\begin{aligned}
\hat{y} &= \phi(\mathbf{x})^\top \mathbf{w} = \phi(\mathbf{x})^\top (\phi(\mathbf{X})\phi(\mathbf{X})^\top + \lambda I)^{-1} \phi(\mathbf{X})\mathbf{y} && (1)\\
&= \phi(\mathbf{x})^\top (\phi(\mathbf{X})\phi(\mathbf{X})^\top + \lambda I)^{-1} \phi(\mathbf{X})(\mathbf{K} + \lambda I)(\mathbf{K} + \lambda I)^{-1}\mathbf{y}\\
&= \phi(\mathbf{x})^\top (\phi(\mathbf{X})\phi(\mathbf{X})^\top + \lambda I)^{-1} \phi(\mathbf{X})(\phi(\mathbf{X})^\top \phi(\mathbf{X}) + \lambda I)(\mathbf{K} + \lambda I)^{-1}\mathbf{y}\\
&= \phi(\mathbf{x})^\top \underbrace{(\phi(\mathbf{X})\phi(\mathbf{X})^\top + \lambda I)^{-1}(\phi(\mathbf{X})\phi(\mathbf{X})^\top + \lambda I)}_{I} \phi(\mathbf{X})(\mathbf{K} + \lambda I)^{-1}\mathbf{y}\\
&= \underbrace{\phi(\mathbf{x})^\top \phi(\mathbf{X})}_{\mathbf{k}^*} \underbrace{(\phi(\mathbf{X})^\top \phi(\mathbf{X}) + \lambda I)^{-1}}_{\mathbf{K}} \mathbf{y} && (2)
\end{aligned}$$

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

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

1 new datapoint

matrix of all data

k^* K

apply Kernel Trick: $\hat{y} = k^* (K + \lambda I)^{-1} y = \sum_{i=1}^{N} k_\phi(x, x_i)\, \alpha_i$

α

k is now a scalarproduct of kernel (x, x_i)

Results in the kernel ridge regression model

$$\hat{y} = \sum_{i=1}^{N} k_\phi(\mathbf{x}, \mathbf{x}_i)\, \alpha_i$$

$$\text{with} \quad \alpha = (\mathbf{K} + \lambda I)^{-1}\mathbf{y} \quad \text{and } \mathbf{K}_{i,j} = k_\phi(\mathbf{x}_i, \mathbf{x}_j)$$

The matrix K is called **kernel matrix**; $K_{i,j}$ denotes the similarity between data points x_i and x_j in the training data set.

→ Bayesian Linear Model

Apply Bayesian inference to model **supervised learning** problem:
given samples $(x_1,y_1),\ldots,(x_T,y_T)$ make prediction for new input x^*

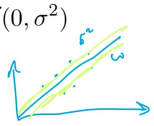
1. Assume that the observation y arises from a **parametric model** f plus and independent, identically distributed noise term ε:

$$y = f_{\mathbf{w}}(\mathbf{x}) + \varepsilon \quad \text{with} \quad f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}; \quad p(\varepsilon) = \mathcal{N}(0, \sigma^2)$$

2. Calculate **likelihood** of observed data:

this likelihood is gaussian

mean: $y = w \cdot x$



$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = \prod_{i=1}^{n} p(y_i \mid \mathbf{x}_i, \mathbf{w}) = \mathcal{N}(\mathbf{X}^\top \mathbf{w}, \sigma^2)$$

data iid: no correlation

3. Define **prior** $p(w)$ and calculate **posterior** $p(w|X,y)$ using Bayes rule:

$$p(\mathbf{w}) = \mathcal{N}(0, \Sigma)$$

$$p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{w}) p(\mathbf{y} \mid \mathbf{X}, \mathbf{w})}{p(\mathbf{y} \mid \mathbf{X})} = \frac{\text{prior} \times \text{likelihood}}{\text{marginal likelihood}}$$

$$\propto \mathcal{N}(\sigma^{-2} \mathbf{A}^{-1} \mathbf{X}\mathbf{y}, \mathbf{A}^{-1})$$

$$\text{with} \quad \mathbf{A} = \sigma^{-2} \mathbf{X}\mathbf{X}^\top + \Sigma^{-1}$$

---

4. Make **prediction** for new input x^*:

$$p(f^* \mid \mathbf{x}^*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\sigma^{-2} \mathbf{x}^{*\top} \mathbf{A}^{-1} \mathbf{X}\mathbf{y}, \mathbf{x}^{*\top} \mathbf{A}^{-1} \mathbf{x})$$

$$\text{with} \quad \mathbf{A} = \sigma^{-2} \mathbf{X}\mathbf{X}^\top + \Sigma^{-1}$$

Note: In contrast to previous models the Bayesian prediction is not a single number but a **predictive distribution.**

The **mean** often serves as predicted value; the **variance** as an estimate of confidence.

→ nonliner Bayesian Model

To make the Bayesian approach nonlinear we reconsider the predicted mean

$$\bar{f}^* = \sigma^{-2} \mathbf{x}^{*\top} \mathbf{A}^{-1} \mathbf{X}\mathbf{y} \quad \text{with} \quad \mathbf{A} = \sigma^{-2} \mathbf{X}\mathbf{X}^\top + \Sigma^{-1}$$

1. Project data into feature space via function Φ

$$\bar{f}^* = \sigma^{-2} \phi(\mathbf{x}^*)^\top \mathbf{A}^{-1} \phi(\mathbf{X})\mathbf{y}$$
$$\text{with} \quad \mathbf{A} = \sigma^{-2} \phi(\mathbf{X}) \phi(\mathbf{X})^\top + \Sigma^{-1}$$

2. Apply kernel trick

$$\bar{f}^* = \sigma^{-2} \phi(\mathbf{x}^*)^\top \mathbf{A}^{-1} \phi(\mathbf{X})\mathbf{y}$$

$$= \underbrace{\phi(\mathbf{x}^*)^\top \Sigma \phi(\mathbf{X})}_{\mathbf{k}^*} \underbrace{(\phi(\mathbf{X})^\top \Sigma \phi(\mathbf{X})}_{\mathbf{K}} + \sigma^2 I)^{-1} \mathbf{y}$$

$$= \mathbf{k}^* (\mathbf{K} + \sigma^2 I)^{-1} \mathbf{y}$$

Bayesian & Frequencies → always the same results

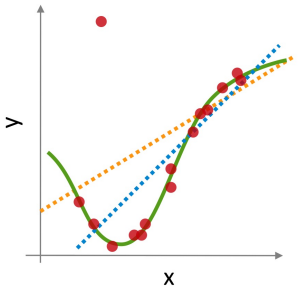
3. Resulting model is called **Gaussian Process**

$$\bar{f}^* = \mathbf{k}^* (\mathbf{K} + \sigma^2 I)^{-1} \mathbf{y}$$
$$\text{var}(f^*) = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^* (\mathbf{K} + \sigma^2 I)^{-1} \mathbf{k}^{*\top}$$

Again a **kernel function** k measures the similarity between samples

$$\text{here} \quad k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \Sigma \, \phi(\mathbf{x}_j)$$
$$\mathbf{k}^* = k(\mathbf{x}^*, \mathbf{X}); \quad \mathbf{K} = k(\mathbf{X}, \mathbf{X})$$

Overview:



Linear Regression
ignores the outlier with regularizing
Ridge Regression
nonlinear
Kernel Ridge Regression closest
Training Data

Method	biased	robust	nonlinear
Linear Regression	no	no	no
Ridge Regression (Shrinkage estimator: ||w|| makes it robust)	yes	yes	no
Kernel Ridge Regression	yes	yes	yes

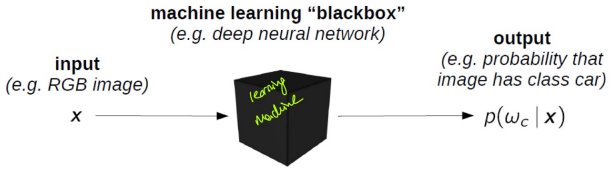
Bayesian Regression Methods

- Bayesian methods **predict distribution** of outputs
- **Bayesian Linear Model**
  - Bayesian counterpart of linear regression
- **Gaussian Process**
  - Bayesian method for nonlinear regression
  - Can be viewed as inference in space of functions
  - Predictive mean equals the kernel ridge regression prediction, if a squared exponential covariance function is assumed

Lecture 16: XAI

- Explaining non-linear or single decisions is difficult.
- single decisions are based on heat maps, model thinks its real because of the ship

Interpreting with class prototypes



Goal: Understand what the model considers as a "typical car".

Activation maximization framework:

given the NN can we max x such that it max the possibility of the output?

$$\max_{x \in \mathcal{X}} \log p(\omega_c \mid x) - \lambda \|x\|^2$$

↳ class prototype

prob. model scan X space find the answer that max. the prob.

Building more natural prototypes

$$\max_{x \in \mathcal{X}} \log p(\omega_c \mid x) - \lambda \|x\|^2$$ ---------- find input pattern that produces the strongest class activation

replace l2-norm by a data-dependent regularizer ↓ we only allow density of our observation

$$\max_{x \in \mathcal{X}} \log p(\omega_c \mid x) + \log p(x)$$

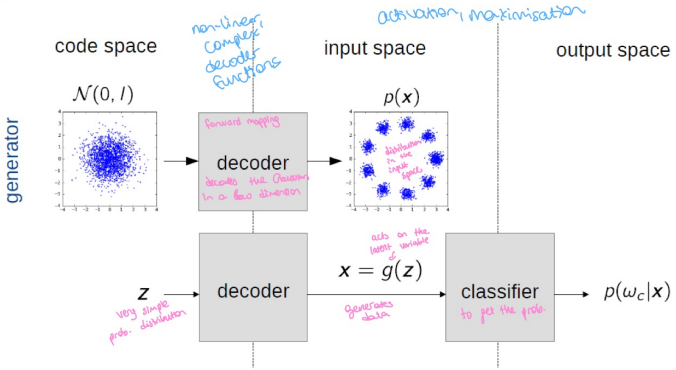
↓ apply the Bayes theorem

$$\max_{x \in \mathcal{X}} \log p(x \mid \omega_c)$$ ---------- find most likely input pattern for a given class

Insight: a good prototype must also depend on the data distribution.

don

Building Prototypes using a generator



Activation maximization in code space: not in very high-dim. X space anymore

$$\max_{z \in \mathcal{Z}} \log p(\omega_c \mid g(z)) - \lambda \|z\|^2$$

was x before — regularizing in code space

NN: classifier
take a latent space and generate X-space
latent space very low dim.
So its very easy to find its max
↳ Class prototypes in latent space

Types of Interpretation

Global Interpretation:

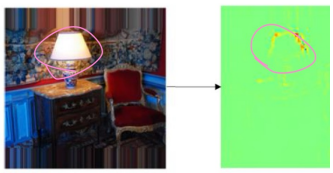
Understanding how a lamp typically looks like.

looking at class prototypes

model's prototypical lamp

Local Interpretation:

Understanding why this image of a lamp contains a lamp.



some image of a lamp — why it is cla as a lan

Approaches to interpretability

by the book interpretable

Ante-hoc interpretability:

Choose a model that is readily interpretable and train it.

Example:

$$f(x) = \sum_{i=1}^{d} g_i(x_i)$$ (contribution of ith variable)

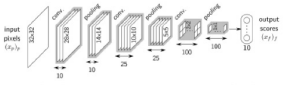
Is the model expressive enough to predict the data?

ex: decision tree — super secure interpretable — only 1st layer ≈ 2nd layer ≈ gets complicated

Post-hoc interpretability:

Choose a model that works well in practice, and develop a special technique to interpret it.

Example:



$$f(x) = \text{DeepNet}(x)$$

lets take the best model

How to determine the contribution each input variable?

Explaining NN predictions

Explanation: *"Which pixels contribute how much to the classification"*
(what makes this image to be classified as a car)

$$f(x) = \sum_p h_p$$

Sensitivity / Saliency: *"Which pixels lead to increase/decrease of prediction score when changed"*
(what makes this image to be classified more/less as a car)

$$h_p = \left|\left| \frac{\partial}{\partial x_p} f(x) \right|\right|_\infty$$

∂ of the nonlinear f w.r.t. one point x

Deconvolution: "Matching input pattern for the classified object in the image"
↳ activity maximisation (relation to f(x) not specified)

Each method solves a **different** problem!!!

change increase/decrease

classification → this way in a NN

$$x_j = \sigma(\sum_i x_i w_{ij} + b_j)$$

explanation ← this way in NN

- model fully trained
- don't do backpropagation
- only to understand the model

$r_j = f(x)$

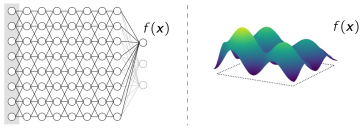
Deep Taylor Decomposition

$$r_i = x_i \sum_j \frac{w_{ij} r_j}{\sum_i x_i w_{ij}} = x_i c_i$$

r_i depends on the activations **and** the weights

→ Activation Maximisation

- **Step 1:** Think of the ML model as a function of the input



- **Step 2:** Explain the function f by generating a maximally activating input pattern:

$$x^\star = \arg\max_x f(x, \theta)$$

Problem: In most cases $f(x)$ does not have single point corresponding to the maximum.

> *E.g. in linear models, $f(x) = w^\top x + b$, we can keep moving the point x further along the direction w, and the output continues to grow).*

Therefore, we would like to apply a preference for 'regular' regions of the input domain, i.e.

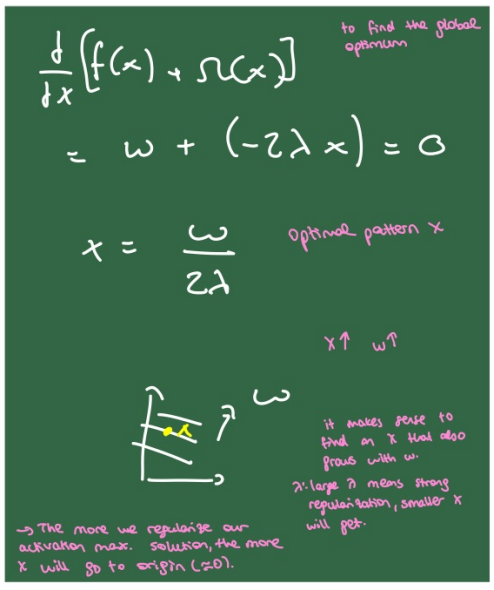
$$x^\star = \arg\max_x \{f(x) + \Omega(x)\}$$

In practice, the preference can be for data points with small norm (i.e. we set $\Omega(x) = -\lambda\|x\|^2$ so that points with large norm are penalized.)

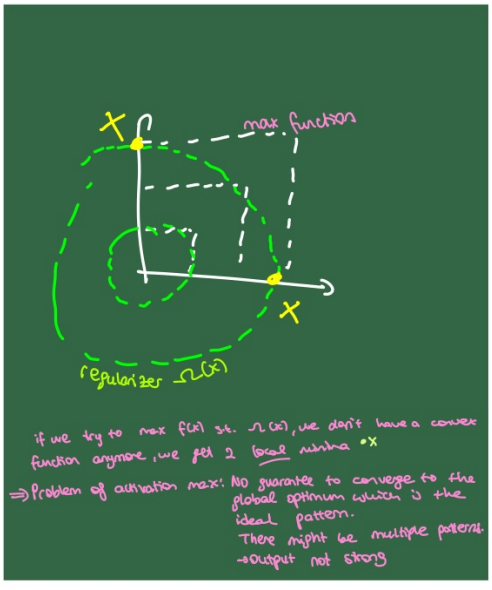
Activation Maximization: Examples

f(x)+Ω(x) is concave → so there's a global optimum

$f(x) = w^\top x + b$ and $\Omega(x) = -\lambda\|x\|^2$



$f(x) = \max(x_1, x_2)$ and $\Omega(x) = -\lambda\|x\|^2$



Activation Maximization: Probability View

Assume the model produces a log-probability for class ω_c:

$$f(\boldsymbol{x}) = \log p(\omega_c|\boldsymbol{x})$$

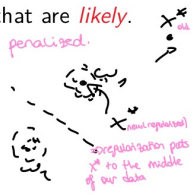
The input $\boldsymbol{x}^\star$ that maximizes this function can be interpreted as the point where the classifier is the most *sure* about class ω_c.

Choose the regularizer $\Omega(\boldsymbol{x}) = \log p(\boldsymbol{x})$, i.e. favor points that are *likely*.

↳ everything unlikely gets penalized.

The optimization problem becomes:

$$\begin{aligned}\boldsymbol{x}^\star &= \arg\max_{\boldsymbol{x}} \ \log p(\omega_c|\boldsymbol{x}) + \log p(\boldsymbol{x}) \\ &= \arg\max_{\boldsymbol{x}} \ \log p(\boldsymbol{x}|\omega_c)\end{aligned}$$

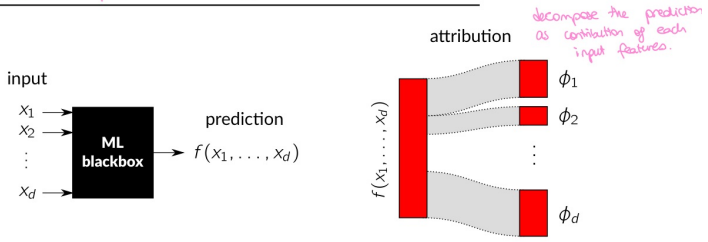


where $\boldsymbol{x}^\star$ can now be interpreted as the most *typical* input for class ω_c.

Attribution

Attribution of a Prediction to Input Features

To understand the prediction for a given datapoint



decompose the prediction as contribution of each input features.

1. The data $\boldsymbol{x} \in \mathbb{R}^d$ is fed to the ML black-box and we get a prediction $f(\boldsymbol{x}) \in \mathbb{R}$.
2. We explain the prediction by determining the contribution of each input feature.

Key property of an explanation: **conservation** ($\sum_{i=1}^{d} \phi_i = f(\boldsymbol{x})$).

Sum of scores is f(x)
f(x) is a chunck of different input features

contribution of each input feature:

- Shapley Values
- Taylor Expansion
- LRP Method

→ Shapley Values

The Shapley values $\phi_1, \ldots, \phi_d$ measuring the contribution of each feature are:

$$\phi_i = \sum_{\mathcal{S}:\, i \notin \mathcal{S}} \frac{|\mathcal{S}|!(d-|\mathcal{S}|-1)!}{d!} \left[f(\boldsymbol{x}_{\mathcal{S}\cup\{i\}}) - f(\boldsymbol{x}_{\mathcal{S}})\right]$$

where $(\boldsymbol{x}_{\mathcal{S}})_{\mathcal{S}}$ are all possible subsets of features contained in the input $\boldsymbol{x}$.

each input variable is a player and function output is the profit realized by the cooperating players

ϕ_i : each features contribution needs to be calculated

take the feature list as:

1. $\emptyset$
2. other features alone
3. other features combined

α_S: $\frac{|S|!\,(d-|S|-1)!}{d!}$ — S: size of condition — $S_\emptyset = 0$ — $S_{\{2\}} = 1$ — $S_{\{2,3\}} = 2$ — d: # features

Attribution: Shapley Values

Sum of all combos that is not on it

for ϕ_1: $S = 0, 1, 2$

Recall: $\phi_i = \sum_{\mathcal{S}:\, i \notin \mathcal{S}} \underbrace{\frac{|\mathcal{S}|!(d-|\mathcal{S}|-1)!}{d!}}_{\alpha_S} \underbrace{\left[f(\boldsymbol{x}_{\mathcal{S}\cup\{i\}}) - f(\boldsymbol{x}_{\mathcal{S}})\right]}_{\Delta_S}$

input with feature — input without feature

S: size of condition, d: 3

for ϕ_1: $f(x) = 1 \cdot (0+2)$; for 2: $f(x) = 1 \cdot (1+0)$; for 2,3: $f(x) = 1 \cdot (1+1) = 2$

Worked-through example: Consider the function $f(\boldsymbol{x}) = x_1 \cdot (x_2 + x_3)$. Calculate the contribution of each feature to the prediction $f(\mathbf{1}) = 1 \cdot (1+1) = 2$.

$S! = 0! = 1$ for $\emptyset$; d: # features: 3; for ϕ_1: $S = 0$, $d = 3$; $|S|! = 1$; $(d-S-1)! = 2!$; $\frac{2}{6} = \frac{1}{3}$

size of condition

Feature 1

look at all possible feature sets that does not contain it

→ effect of feature 1

→ scores on ϕ_1 is zero depends on the size of condition

ϕ_1	α_S	Δ_S	ϕ_2	α_S	Δ_S	ϕ_3 ...
$\emptyset$ empty set $S=0$	$1/3$	0	$\emptyset$	$1/3$	0	Shapley value satisfies conservation axiom: sum of all contributions = output of network
$\{2\}$ feature $S=1$	$1/6$	1	$\{1\}$	$1/6$	1	$\sum \phi_i = 2$
$\{3\}$ $S=1$	$1/6$	1	$\{3\}$	$1/6$	0	$\phi_1 = 1$, $\phi_2 = 1/2$ → then $\phi_3 = 1/2$
$\{2,3\}$ $S=2$	$1/3$	2	$\{1,3\}$	$1/3$	1	also: symmetry $x_2 + x_3$ does have the same effect: $\phi_2 = \phi_3$
$\sum \phi_1 = 1$			$\phi_2 = 1/2$			$\phi_3 = 1/2$

→ Taylor Expansions

- take a complex nonlinear function into smaller chunks and observe them as simple linear local function
- then try to approx. via Taylor:
  ϕ_i (first order terms) will be enough to explain the function.

- The function can be approximated locally by some Taylor expansion:

$$f(\boldsymbol{x}) = f(\tilde{\boldsymbol{x}}) + \sum_{i=1}^{d} \underbrace{[\nabla f(\tilde{\boldsymbol{x}})]_i \cdot (x_i - \tilde{x}_i)}_{\phi_i} + \dots$$

- First-order terms ϕ_i of the expansion can serve as an explanation.
- The explanation $(\phi_i)_i$ depends on the choice of root point $\tilde{\boldsymbol{x}}$.

Example: Attribute the prediction $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ with $\boldsymbol{x} \in \mathbb{R}^d$ on the d input features.

$\tilde{x} = 0$

$\phi_i = [\nabla f(\tilde{x})]_i \cdot (x_i - \tilde{x}_i)$

$= w_i \cdot (x_i - 0)$

$= w_i x_i$

- root point $\tilde{x} = 0$
- take derivative of the function
- $(x_i -$ root point$)$ · derivative

→ then you have your first order terms as explanation

Because gradient is the same at the entire function.

Problem: Gradient information too localized
it can't explain discontinuities in a function.

Solution: decompose the problem in multiple parts ⇒ LRP Method

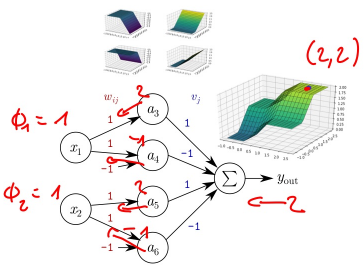
→ LRP Method

- to identify the contributions of each feature.

X → a → R

if weight: -1 → a.w = 0 ignore
weight: +1 → take it ✓

Example: Consider y_{out} to be the quantity to explain.



- **Step 1:** Propagate on the hidden layer

$$\forall_{j=3}^{6} : R_j \leftarrow \frac{a_j v_j}{\sum_{j=3}^{6} a_j v_j} y_{\text{out}}$$

- **Step 2:** Propagate on the first layer

$$\forall_{i=1}^{2} : R_i \leftarrow \sum_{j=3}^{6} \frac{x_i w_{ij}}{\sum_{i=1}^{2} x_i w_{ij}} R_j$$

Note: Other propagation rules can be engineered, and choosing appropriate propagation rules is important to ensure LRP works well for practical neural network architectures.

→ Taylor Expansions

- take a complex nonlinear function into smaller chunks and observe them as simple linear local function
- then try to approx. via Taylor:
  ϕ_i (first order terms) will be enough to explain the function.

- The function can be approximated locally by some Taylor expansion:

$$f(\boldsymbol{x}) = f(\tilde{\boldsymbol{x}}) + \sum_{i=1}^{d} \underbrace{[\nabla f(\tilde{\boldsymbol{x}})]_i \cdot (x_i - \tilde{x}_i)}_{\phi_i} + \dots$$

- First-order terms ϕ_i of the expansion can serve as an explanation.
- The explanation $(\phi_i)_i$ depends on the choice of root point $\tilde{\boldsymbol{x}}$.

Example: Attribute the prediction $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ with $\boldsymbol{x} \in \mathbb{R}^d$ on the d input features.

$\tilde{x} = 0$

$\phi_i = [\nabla f(\tilde{x})]_i \cdot (x_i - \tilde{x}_i)$

$= w_i \cdot (x_i - 0)$

$= w_i x_i$

- root point $\tilde{x} = 0$
- take derivative of the function
- $(x_i -$ root point$)$ · derivative

→ then you have your first order terms as explanation

Because gradient is the same at the entire function.

Problem: Gradient information too localized
it can't explain discontinuities in a function.

Solution: decompose the problem in multiple parts ⇒ LRP Method

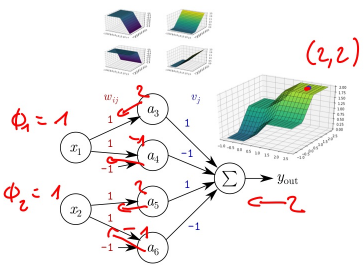
→ LRP Method

- to identify the contributions of each feature.

X → a → R

if weight: -1 → a.w = 0 ignore
weight: +1 → take it ✓

Example: Consider y_{out} to be the quantity to explain.



- **Step 1:** Propagate on the hidden layer

$$\forall_{j=3}^{6} : R_j \leftarrow \frac{a_j v_j}{\sum_{j=3}^{6} a_j v_j} y_{\text{out}}$$

- **Step 2:** Propagate on the first layer

$$\forall_{i=1}^{2} : R_i \leftarrow \sum_{j=3}^{6} \frac{x_i w_{ij}}{\sum_{i=1}^{2} x_i w_{ij}} R_j$$

Note: Other propagation rules can be engineered, and choosing appropriate propagation rules is important to ensure LRP works well for practical neural network architectures.