

# CS-E4710 Machine Learning: Supervised Methods

## Lecture 1: Introduction

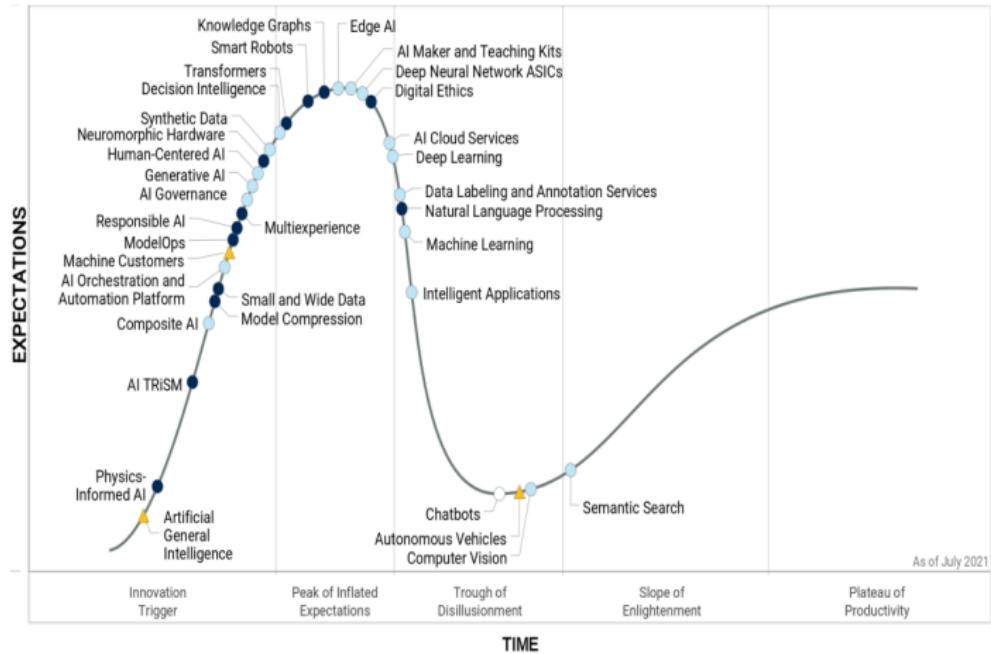
---

Juho Rousu

September 6, 2022

Department of Computer Science  
Aalto University

# Mission: going beyond the hype in machine learning



<https://www.gartner.com/en/articles/>

the-4-trends-that-prevail-on-the-gartner-hype-cycle-for-ai-2021

# Understanding machine learning

For a professional machine learning engineers / data scientists it is useful to go beyond using machine learning tools as black boxes:

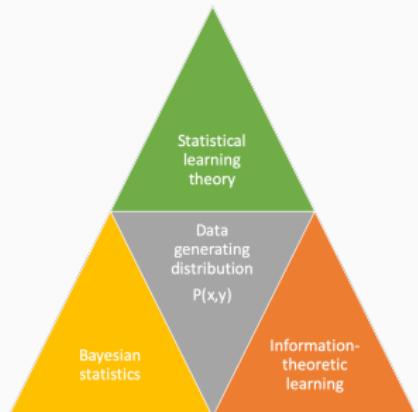
- Machine learning does not always succeed, one needs to be able to understand why and find remedies
- Not possible to follow the scientific advances in the field without understanding the underlying principles
- Competitive advantage: more jobs and better pay for people that have understanding of the algorithms and statistics



# Theoretical paradigms of machine learning

Theoretical paradigms for machine learning differ mainly on what they assume about the process generating the data:

- Statistical learning theory (focus on this course): assumes data is i.i.d from an unknown distribution  $P(x)$ , does not estimate the distribution (directly)
- Bayesian Statistics (focus of course CS-E5710 - Bayesian Data Analysis): assumes prior information on  $P(x)$ , estimates posterior probabilities
- Information theoretic learning (e.g. Minimum Description Length principle, MDL): estimates distributions, but does not assume a prior on  $P(x)$



# Supervised and unsupervised machine learning

- Supervised machine learning (Focus of this course)
  - training data contains inputs and outputs (=supervision)
  - goal is to predict outputs for new inputs
  - typical tasks: classification, regression, ranking
- Unsupervised machine learning (Focus of course CS-E4650 - Methods of Data Mining,  
<https://mycourses.aalto.fi/course/view.php?id=28201>)
  - training data does not contain outputs
  - goal is to describe and interpret the data
  - typical tasks: clustering, association analysis, dimensionality reduction, pattern discovery

# Course topics

- Part I: Theory
  - Introduction
  - Generalization error analysis & PAC learning
  - Rademacher Complexity & VC dimension
  - Model selection
- Part II: Algorithms and models
  - Linear models: perceptron, logistic regression
  - Support vector machines
  - Kernel methods
  - Boosting
  - Neural networks (MLPs)
- Part III: Additional topics
  - Feature learning, selection and sparsity
  - Multi-class classification
  - Preference learning, ranking, multi-output learning

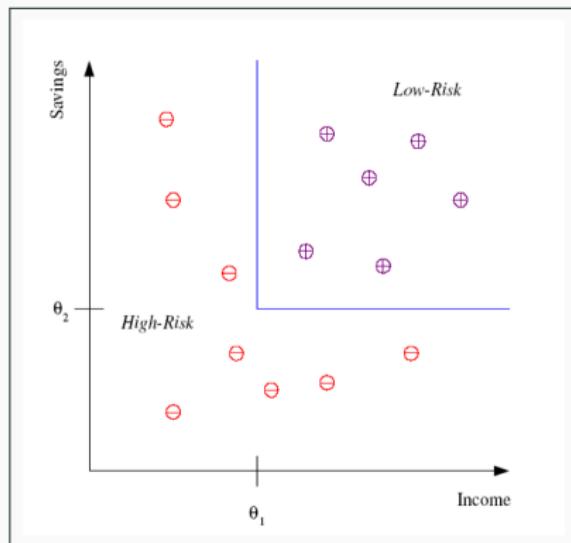
# **Supervised Machine Learning Tasks**

---

# Classification

Classification deals with the problem of partitioning the data into pre-defined classes by a **decision boundary** or **decision surface** (blue line in the figure below)

- Example: In credit scoring task, a bank would like to predict if the customer should be given credit or not
- Decision can be based on available input variables: Income, Savings, Employment, Past financial history, etc.
- Output variable is a class label: low-risk (0) or high-risk (1)
- This is called binary classification since we have two classes



# Multi-class classification

Multi-class classification tackles problems where there are more than two classes

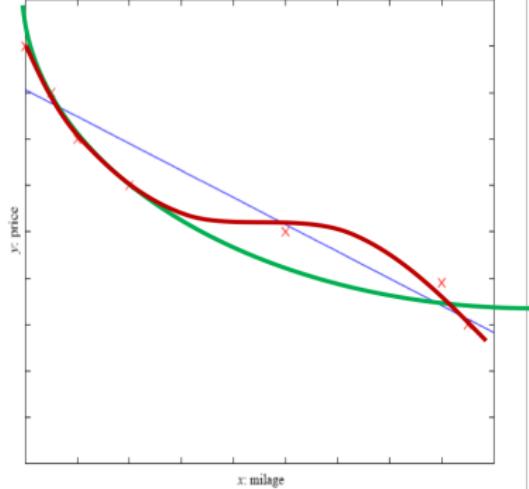
- Example: hand-written character recognition
- Input: images of hand-written characters
- Output: the identity of the character (e.g. Unicode ID)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9

- **Multi-label Classification** An example can belong to multiple classes at the same time
- **Extreme classification** Learning with thousands to hundreds of thousands of classes (Prof. Rohit Babbar @ Aalto)

# Regression

- Regression deals with output variables which are numeric
- Example: predicting the price of the car based on input variables (model, year, engine capacity, mileage)
- Linear regression: our model is a line:  
 $f(x) = wx + w_0$
- Polynomial regression: our model is a polynomial: quadratic  
 $f(x) = w_2x^2 + w_1x + w_0$ , cubic  
 $f(x) = w_3x^3 + w_2x^2 + w_1x + w_0$  or even higher order
- Many other non-linear regression models besides polynomials



# Ranking and preference learning

- Sometimes we do not need to predict exact values but a ordered list of preferred objects is sufficient
- Example: a movie recommendation system
- Input: characteristics of movies the user has liked
- Output: an ranked list of recommended movies for the user
- Training data typically contains pairwise preferences: user  $x$  prefers movie  $y_i$  over movie  $y_j$



# Dimensions of a supervised learning algorithm

1. **Training sample:**  $S = \{(x_i, y_i)\}_{i=1}^m$ , the training examples  $(x, y) \in X \times \mathcal{Y}$  independently drawn from a identical distribution (i.i.d)  $D$  defined on  $X \times \mathcal{Y}$ ,  $X$  is a space of inputs,  $\mathcal{Y}$  is the space of outputs
2. **Model or hypothesis**  $h : X \mapsto \mathcal{Y}$  that we use to predict outputs given the inputs  $x$
3. **Loss function:**  $L : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ ,  $L(\dots) \geq 0$ ,  $L(y, y')$  is the loss incurred when predicting  $y'$  when  $y$  is true
4. Optimization procedure to find the hypothesis  $h$  that minimize the loss on the training sample

- The **input space**  $X$ , also called the **instance space** is generally taken as an arbitrary set
  - Often  $X \subset \mathbb{R}^d$ , then the training inputs are vectors  $\mathbf{x} \in \mathbb{R}^d$  - we use boldface font to indicate vectors
  - But they can also be non-vectorial objects (e.g. sequences, graphs, signals)
  - Often data are mapped to **feature vectors** in preprocessing or during learning.
- The **output space**  $\mathcal{Y}$  containing the possible outputs or **labels** for the model, depends on the task:
  - Binary classification:  $\mathcal{Y} = \{0, 1\}$  or  $\mathcal{Y} = \{-1, +1\}$
  - Multiclass classification:  $\mathcal{Y} = \{1, \dots, K\}$
  - Regression:  $\mathcal{Y} = \mathbb{R}$
  - Multi-task/multi-label learning:  $\mathcal{Y} = \mathbb{R}^d$  or  $\mathcal{Y} = \{-1, +1\}^d$

## Loss functions

- Loss function  $L : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ , measures the discrepancy  $L(y, y')$  between two outputs  $y, y' \in \mathcal{Y}$
- Used to measure an approximation of the error of the model, called the **empirical risk**, by computing the average of the losses on individual instances

$$\hat{R}(h) = \frac{1}{m} \sum_{i=1}^m L(h(x_i), y_i)$$

- Loss functions depend on the task:
  - squared loss is used in regression:  $L_{\text{sq}}(y, y') = (y' - y)^2$ ,  $y, y' \in \mathbb{R}$
  - 0/1 loss is used in classification:  $L_{0/1}(y, y') = \mathbf{1}_{y \neq y'}$
  - Hamming loss is used in multilabel learning:  
$$L(y, y') = \sum_{j=1}^d L_{0/1}(y_j, y'_j), y, y' \in \{-1, +1\}^d$$
- Loss functions taking into account the structure of the output space or the cost of errors can also be defined

# Generalization

- Our aim is to predict as well as possible the outputs of future examples, not only for training sample
- We would like to minimize the **generalization error**, or the (true) risk

$$R(h) = \mathbb{E}_{(x,y) \sim D} [ L(h(x), y) ],$$

- Assuming future examples are independently drawn from the same distribution  $D$  that generated the training examples (i.i.d assumption)
- But we do not know  $D$ !
- What can we say about  $R(h)$  based on training examples and the hypothesis class  $\mathcal{H}$  alone? Two possibilities:
  - Empirical evaluation through testing
  - Statistical learning theory (Lectures 2 and 3)

## Hypothesis classes

There is a huge number of different **hypothesis classes** or **model families** in machine learning, e.g:

- Linear models such as logistic regression and perceptron
- Neural networks: compute non-linear input-output mappings through a network of simple computation units
- Kernel methods: implicitly compute non-linear mappings into high-dimensional feature spaces (e.g. SVMs)
- Ensemble methods: combine simpler models into powerful combined models (e.g. Random Forests)

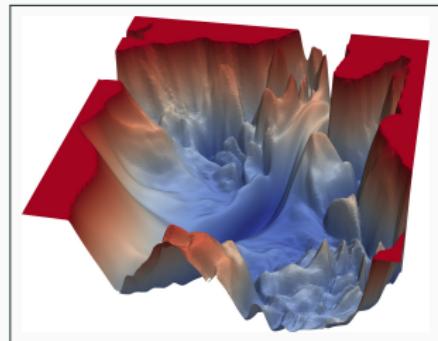
Each have their different pros and cons in different dimensions (accuracy, efficiency, interpretability); No single best hypothesis class exists that would be superior to all others in all circumstances.

# Optimization algorithms

The difficulty of finding the best model depends on the model family and the loss function

We are often faced with

- Non-convex optimization landscapes (e.g. neural networks) ↪ hard to find the global optimum
- NP-hard optimization problems (e.g finding a linear classifier that has the smallest empirical risk) ↪ need to use approximations and heuristics



Optimization landscape of a neural net. Source:

<https://www.cs.umd.edu/~tomg/projects/landscapes/>

"Big Data" with very large training sets (1 million examples and beyond) amplifies these problems

## Poll: Vaccine decision support system

Government wants to develop a decision support system that would help them choose the best COVID-19 vaccine for a population. The system should be able to use the existing data available for the current vaccines (e.g. price, efficacy, storage requirements, side-effects) but the model should be able to work also for future vaccines. Government health officials have given each existing vaccine a goodness score that reflects their opinion of each vaccine.

Which machine learning task would be the best match for the system?

1. Classification
2. Regression
3. Ranking
4. Unsupervised learning

Answer to the poll in Mycourses: Go to Lectures page and scroll down to "Lecture 1 poll":

[https:](https://mycourses.aalto.fi/course/view.php?id=37029&section=2)

//mycourses.aalto.fi/course/view.php?id=37029&section=2

## Linear regression

---

## Example: linear regression

- Training Data:  $\{(x_i, y_i)\}_{i=1}^m$ ,  $(x, y) \in \mathbb{R}^d \times \mathbb{R}$
- Loss function: squared loss  $L_{sq}(y, y') = (y - y')^2$
- Hypothesis class: hyperplanes in  $h(\mathbf{x}) = \sum_{j=1}^d w_j x_j + w_0$
- Model:  $y = h(\mathbf{x}) + \epsilon$ , where  $\epsilon$  is random noise corrupting the output.  
We assume zero-mean normal distributed noise:  $\epsilon \sim \mathcal{N}(0, \sigma^2)$ , with unknown  $\sigma$
- Optimization: essentially, inverting a matrix (low polynomial time complexity)

## Example: linear regression

Optimization problem

$$\begin{aligned} & \text{minimize} \quad \sum_{i=1}^m \left( y_i - \sum_{j=1}^d w_j x_{ij} + w_0 \right)^2 \\ & \text{w.r.t. } w_j, j = 0, \dots, d \end{aligned}$$

Write this in matrix form:

$$\begin{aligned} & \text{minimize } (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) \\ & \text{w.r.t. } \mathbf{w} \in \mathbb{R}^{d+1} \end{aligned}$$

where  $\mathbf{X} = \begin{bmatrix} 1 & \mathbf{x}_1 \\ \vdots & \vdots \\ 1 & \mathbf{x}_i \\ \vdots & \vdots \\ 1 & \mathbf{x}_m \end{bmatrix}$ ,  $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$ ,  $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_m \end{bmatrix}$

## Example: linear regression

Minimum of

$$\text{minimize } (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\text{w.r.t. } \mathbf{w} \in \mathbb{R}^{d+1}$$

is attained when the derivatives w.r.t  $\mathbf{w}$  go to zero

$$\begin{aligned} & \frac{\partial}{\partial \mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) \\ &= \frac{\partial}{\partial \mathbf{w}} \mathbf{y}^T \mathbf{y} - \frac{\partial}{\partial \mathbf{w}} 2(\mathbf{X}\mathbf{w})^T \mathbf{y} + \frac{\partial}{\partial \mathbf{w}} (\mathbf{X}\mathbf{w})^T \mathbf{X}\mathbf{w} \\ &= -2\mathbf{X}^T \mathbf{y} + 2(\mathbf{X}^T \mathbf{X})\mathbf{w} = 0 \end{aligned}$$

This gives us a set of linear equations  $\mathbf{X}^T \mathbf{y} = (\mathbf{X}^T \mathbf{X})\mathbf{w}$  that can be solved by inverting  $\mathbf{X}^T \mathbf{X}$ :

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

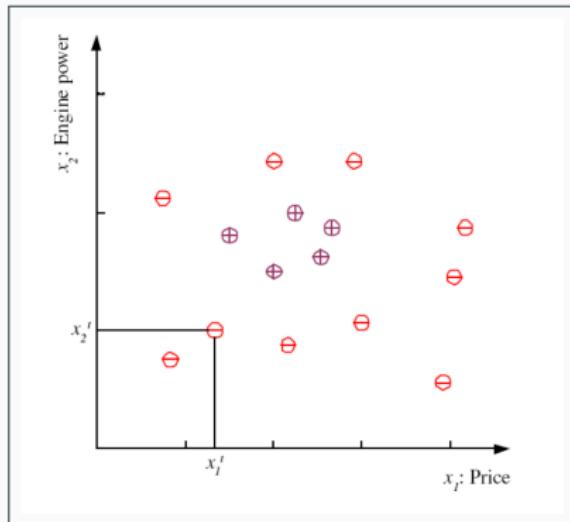
if  $\mathbf{X}^T \mathbf{X}$  invertible, and by computing a pseudo-inverse, otherwise

## Binary classification

---

# Binary classification

- Goal: learn a class  $C$ ,  $C(\mathbf{x}) = 1$  for members of the class,  $C(\mathbf{x}) = 0$  for others
- Example: decide if car is a family car ( $C(\mathbf{x}) = 1$ ) or not ( $C(\mathbf{x}) = 0$ )
- We have a training set of labeled examples
  - positive examples of family cars
  - negative examples of other than family cars
- Assume two relevant input variables have been picked by a human expert: price and engine power



# Data representation

- The inputs are 2D vectors

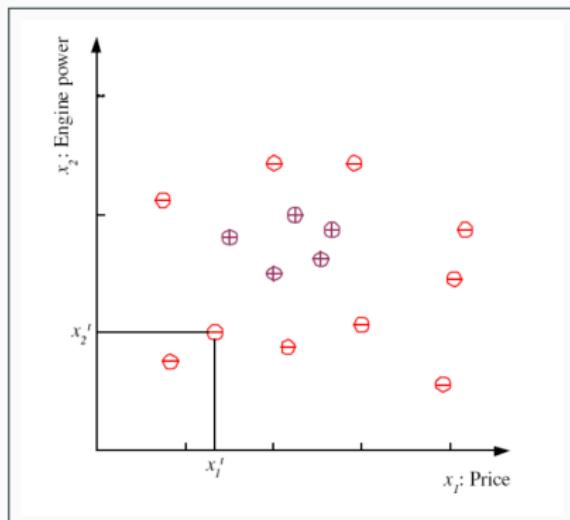
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2, \text{ where } x_1 \text{ is the price}$$

and  $x_2$  is the engine power

- The label is a binary variable

$$y = \begin{cases} 1 & \text{if } \mathbf{x} \text{ is a family car} \\ 0 & \text{if } \mathbf{x} \text{ is not a family car} \end{cases}$$

- Training sample  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$  consists of training examples, pairs  $(\mathbf{x}, y)$
- The labels are assumed to usually satisfy  $y_i = C(\mathbf{x}_i)$ , but may not always do e.g. due to **intrinsic noise** in the data



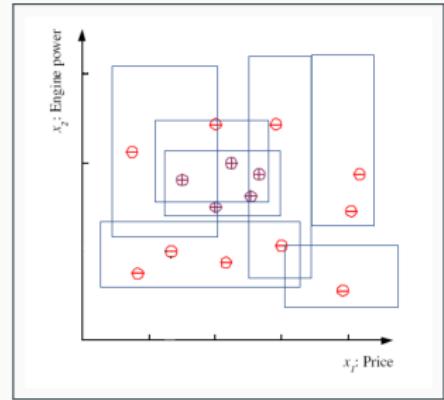
# Hypothesis class

- We choose as the **hypothesis class**  
 $\mathcal{H} = \{h : X \mapsto \{0, 1\}\}$  the set of axis parallel rectangles in  $\mathbb{R}^2$

$$h(\mathbf{x}) = (p_1 \leq x_1 \leq p_2) \text{ AND } (e_1 \leq x_2 \leq e_2)$$

- The classifier will predict a "family car" if both price and engine power are within their respective ranges
- The learning algorithm chooses a  $h \in \mathcal{H}$  by assigning values to the parameters  $(p_1, p_2, e_1, e_2)$  so that  $h$  approximates  $C$  as closely as possible

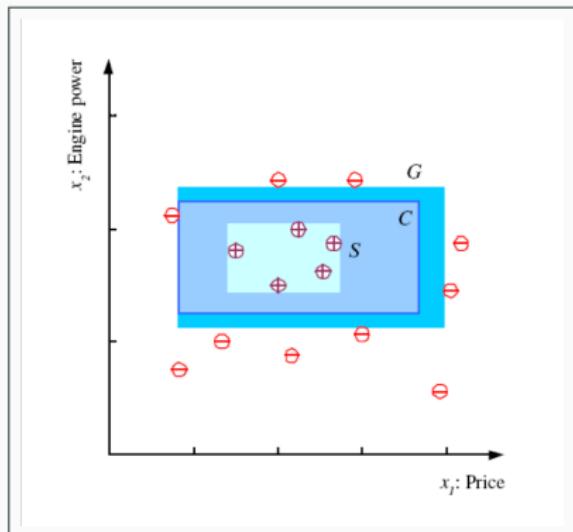
However, we do not know  $C$ , so cannot measure exactly how close  $h$  is to  $C$ !



## Version space

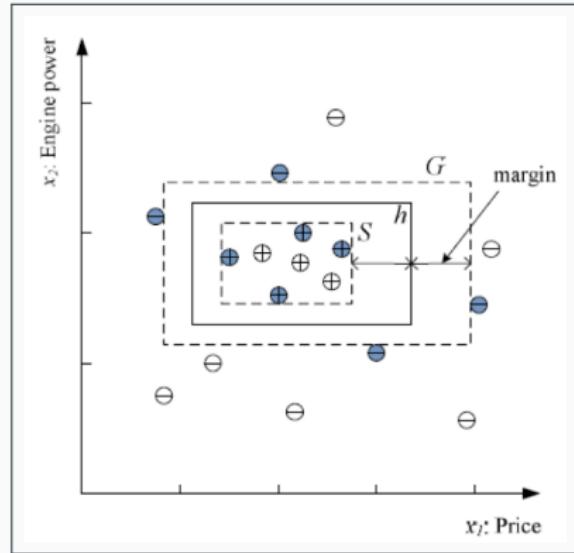
If a hypothesis correctly classifies all training examples we call it a **consistent hypothesis**

- Version space: the set of all consistent hypotheses of the hypothesis class
- Most general hypothesis  $G$ : cannot be expanded without including negative training examples
- Most specific hypothesis  $S$ : cannot be made smaller without excluding positive training points



# Margin

- Intuitively, the "safest" hypothesis to choose from the version space is the one that is furthers from the positive and negative training examples
- Margin is the minimum distance between the decision boundary and a training point



The principle of choosing the hypothesis with a maximum margin is used, e.g. in Support Vector Machines

## Model evaluation

---

## Zero-one loss

---

- The most commonly used loss function for classification is the zero-one loss:  $L_{0/1}(y, y') = \mathbf{1}_{y \neq y'}$  where  $\mathbf{1}_A$  is the indicator function:

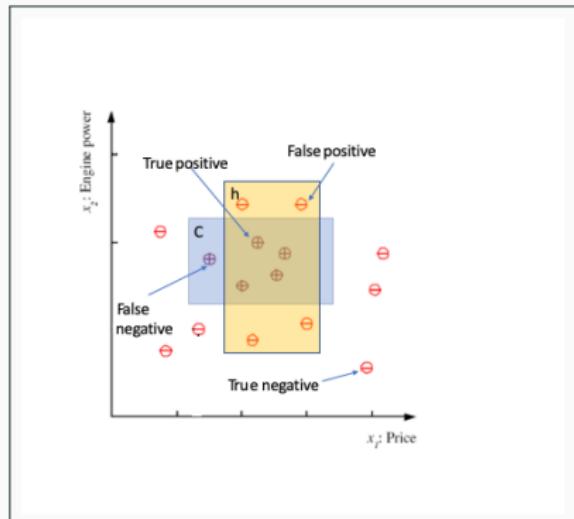
$$\mathbf{1}_A = \begin{cases} 1 & \text{if } A \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

- However, zero-one loss is not a good metric when the class distributions are imbalanced
  - consider a binary problem with 9990 examples in class 0 and 10 examples in class 1
  - if model predicts everything to be class 0, accuracy is  $9990/10000 = 99.9\%$  which is misleading
- Class-dependent misclassification costs are another weakness:
  - consider we are dealing with a rare but fatal disease, the cost of failing to diagnose the disease of a sick person is much higher than the cost of sending a healthy person to more tests

# Confusion matrix

In binary classification, the zero-one loss is composed of

- **False positives**  
 $\{\mathbf{x}_i : h(\mathbf{x}_i) = 1 \text{ and } y_i = 0\}$  and
- **False negatives**  
 $\{\mathbf{x}_i : h(\mathbf{x}_i) = 0 \text{ and } y_i = 1\}$
- Generally
  - making the hypothesis more specific leads to increased false negative rate and decreased false positive rate (here: smaller rectangle)
  - making the hypothesis more general (here: larger rectangle) does the opposite



# Confusion matrix

- Consider all four possible combinations of the predicted label (0 or 1) and the actual label (0 or 1)
- The counts of examples in the four combinations can be compactly represented in a **confusion matrix**

		Actual class	
		Yes	No
Predicted class	Yes	True positive (TP)	False positive (FP)
	No	False negative (FN)	True negative (TN)

- True Positives:  $m_{TP} = |\{x_i : h(x_i) = 1 \text{ and } y_i = 1\}|$
- True Negatives:  $m_{TN} = |\{x_i : h(x_i) = 0 \text{ and } y_i = 0\}|$
- False Positives:  $m_{FP} = |\{x_i : h(x_i) = 1 \text{ and } y_i = 0\}|$
- False Negatives:  $m_{FN} = |\{x_i : h(x_i) = 0 \text{ and } y_i = 1\}|$

		Actual class		count
		Yes	No	
Predicted class	Yes	$m_{TP}$	$m_{FP}$	$m_{TP}+m_{FP}$
	No	$m_{FN}$	$m_{TN}$	$m_{FN}+m_{TN}$
count		$m_{TP}+m_{FN}$	$m_{FP}+m_{TN}$	$m$

# Confusion matrix

From the confusion matrix, many **evaluation metrics** besides can be computed

- Empirical risk (zero-one loss as the loss function):

$$\hat{R}(h) = \frac{1}{m} (m_{FP} + m_{FN})$$

- Precision or Positive Predictive Value

$$: Prec(h) = \frac{m_{TP}}{m_{TP} + m_{FP}}$$

- Recall or Sensitivity:

$$Rec(h) = \frac{m_{TP}}{m_{TP} + m_{FN}}$$

- F1 score:  $F_1(h) = 2 \frac{Prec(h) \cdot Rec(h)}{Prec(h) + Rec(h)} = \frac{2m_{TP}}{2m_{TP} + m_{FP} + m_{FN}}$

		Actual class	
		Yes	No
Predicted class	Yes	True positive (TP)	False positive (FP)
	No	False negative (FN)	True negative (TN)

		Actual class		count
		Yes	No	
Predicted class	Yes	mTP	mFP	mTP+mFP
	No	mFN	mTN	mFN+mTN
count		mTP+mFN	mFP+mTN	m

And many others see e.g.

[https://en.wikipedia.org/wiki/Confusion\\_matrix](https://en.wikipedia.org/wiki/Confusion_matrix)

# Receiver Operating Characteristics(ROC)

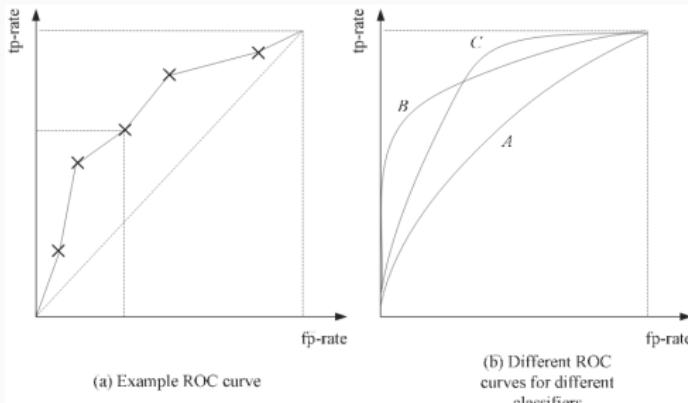
- ROC curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds.
- Consider a system which returns an estimate of the class probability  $\hat{P}(y|x)$  or a any score that correlates with it.
- We may choose a threshold  $\theta$  and make a classification rule:

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } \hat{P}(y|x) \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

- For high values of  $\theta$  prediction will be 0 for large fraction of the data (and there are likely more false negatives), for low values of  $\theta$  prediction will be 1 for a large fraction of data (and there are likely more false positives)

# Receiver Operating Characteristics(ROC)

- Taking into account all possible values  $\theta$  we can plot the resulting **ROC curve**, x-coordinate: False positive rate  $FPR = m_{FP}/m$ , y-coordinate: True positive rate  $TPR = m_{TP}/m$
- The higher the ROC curve goes, the better the algorithm or model (higher TP rate for the same FP rate)
- If two ROC curves cross it means neither model/algorithim is globally better
- The curve is sometimes summarized into a single number, the area under the curve (**AUC** or **AUROC**)



## Model evaluation by testing

- How to estimate the model's ability to generalize on future data
- We can compute an approximation of the true risk by computing the empirical risk on an independent test sample:

$$R_{\text{test}}(h) = \sum_{(\mathbf{x}_i, y_i) \in S_{\text{test}}}^m L(h(\mathbf{x}_i), y_i),$$

- The expectation of  $R_{\text{test}}(h)$  is the true risk  $R(h)$
- Why not just use the training sample?
  - Empirical risk  $\hat{R}(h)$  on the training sample is generally lower than the true risk, thus we would get overly optimistic results
  - The more complex the model the lower empirical risk on training data: we would select overly complex models
  - "Learning  $\neq$  Fitting"

# Summary

- Supervised machine learning is concerned about building models that efficiently and accurately predict output variables from input variables.
- Many different supervised learning tasks: classification, regression, ranking
- Loss functions are used to measure how accurately the model predicts the output
- The ultimate interest in machine learning is **generalization**: the models' ability to generalize to new data

# CS-E4710 Machine Learning: Supervised Methods

Lecture 2: Statistical learning theory

---

Juho Rousu

September 13, 2022

Department of Computer Science  
Aalto University

# Generalization

- Our aim is to predict as well as possible the outputs of future examples, not only for training sample
- We would like to minimize the **generalization error**, or the (true) **risk**

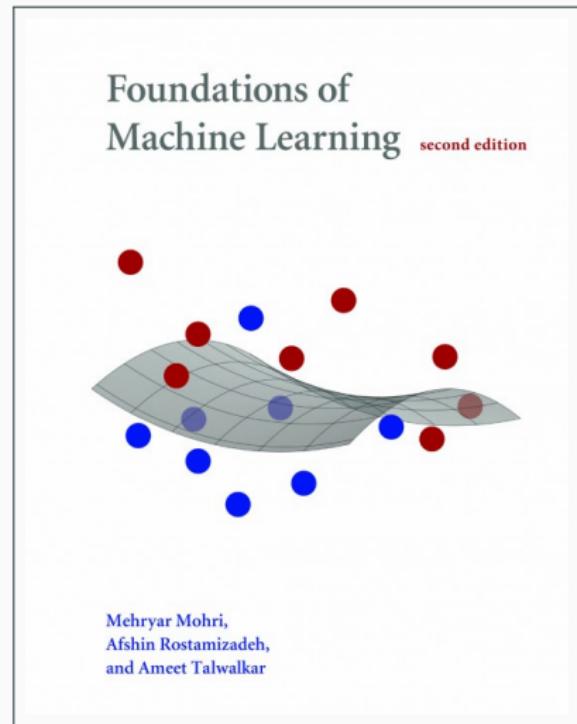
$$R(h) = \mathbb{E}_{(\mathbf{x}, y) \sim D} [L(h(\mathbf{x}), y)],$$

where  $L(y, y')$  is a suitable loss function (e.g. zero-one loss)

- Assuming future examples are independently drawn from the same distribution  $D$  that generated the training examples (i.i.d assumption)
- But we do not know  $D$ !
- What can we say about  $R(h)$  based on training examples and the hypothesis class  $\mathcal{H}$  alone? Two possibilities:
  - Empirical evaluation through testing
  - **Statistical learning theory (Lectures 2 and 3)**

## Additional reading

- Lectures 2-4 are mostly based on Mohri et al book: chapters 2-4
- Available online in Aalto eBookAalto Central:  
<https://ebookcentral.proquest.com/lib/aalto-ebooks/detail.action?pq-origsite=primo&docID=6246520>
- The book goes much deeper in the theory (e.g. proofs of theorems) than what we do in the course



# **Probably approximately correct learning**

---

# Probably Approximate Correct Learning framework

- Probably Approximate Correct (PAC) Learning framework formalizes the notion of generalization in machine learning
- Ingredients:
  - input space  $X$  containing all possible inputs  $x$
  - set of possible labels  $\mathcal{Y}$  (in binary classification  $\mathcal{Y} = \{0, 1\}$  or  $\mathcal{Y} = \{-1, +1\}$ )
  - Concept class  $\mathcal{C}$  containing concepts  $C : X \mapsto \mathcal{Y}$  (to be learned), concept  $C$  gives a label  $C(x)$  for each input  $x$
  - unknown probability distribution  $D$
  - training sample  $S = (x_1, C(x_1)), \dots, (x_m, C(x_m))$  drawn independently from  $D$
  - hypothesis class  $\mathcal{H}$ , in the basic case  $\mathcal{H} = \mathcal{C}$  but this assumption can be relaxed
- The goal in PAC learning is to learn a hypothesis with a low generalization error

$$R(h) = \mathbb{E}_{x \sim D} [L_{0/1}(h(x), C(x))] = \Pr_{x \sim D} (h(x) \neq C(x))$$

## PAC learnability

- A class  $\mathcal{C}$  is **PAC-learnable**, if there exist an algorithm  $\mathcal{A}$  that given a training sample  $S$  outputs a hypothesis  $h_S \in \mathcal{H}$  that has generalization error satisfying

$$\Pr(R(h_S) \leq \epsilon) \geq 1 - \delta$$

- for **any** distribution  $D$ , for arbitrary  $\epsilon, \delta > 0$  and sample size  $m = |S|$  that grows polynomially in  $1/\epsilon, 1/\delta$ 
  - for **any** concept  $C \in \mathcal{C}$
- In addition, if  $\mathcal{A}$  runs in time polynomial in  $m, 1/\epsilon$ , and  $1/\delta$  the class is called **efficiently PAC learnable**

# Interpretation

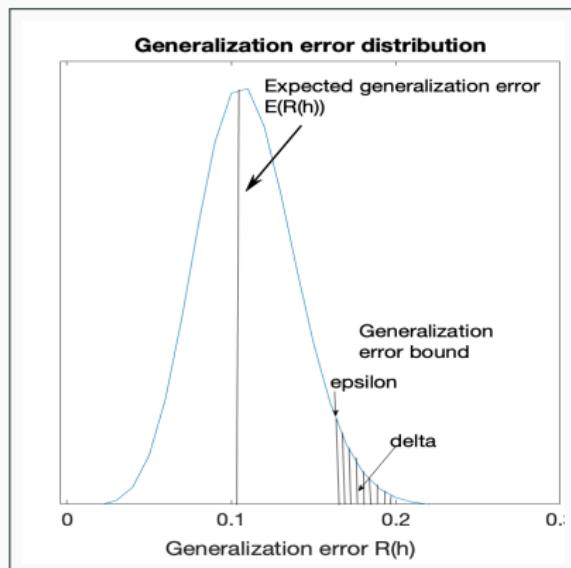
Let us interpret the bound

$$\Pr(R(h_S) \leq \epsilon) \geq 1 - \delta$$

- $\epsilon$  sets the level of generalization error that is of interest to us, say we are content with predicting incorrectly 10% of the new data points:  
 $\epsilon = 0.1$
- $1 - \delta$  sets a level of confidence, if we are content of the training algorithm to fail 5% of the time to provide a good hypothesis:  
 $\delta = 0.05$
- Samples size and running time should **not explode** when we make  $\epsilon$  and  $\delta$  stricter: requirement of polynomial growth
- The event "low generalization error",  $\{R(h_S) \leq \epsilon\}$  is considered as a random variable because we cannot know beforehand which hypothesis  $h_S \in \mathcal{H}$  will be selected by the algorithm

# Generalization error bound vs. test error

- Generalization error bounds concern the tail of the error distribution
  - We wish a high generalization error to be a **rare event**
- Expected generalization error which might be considerably lower
  - Analyzing average behaviour where most data distributions and concepts are "not bad"
- We expect there be a gap between the expected error and the tail
  - The smaller the failure probability  $\delta$ , the larger the gap

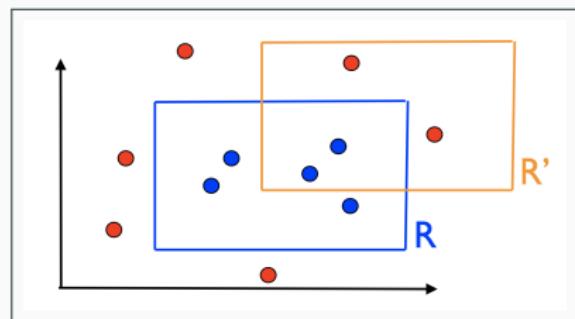


## **Example: learning axis-aligned rectangles**

---

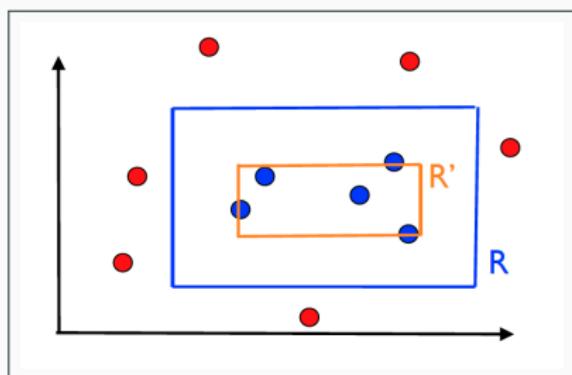
## Learning setup

- The goal is to learn a rectangle  $R$  (representing the true concept) that includes all blue points and excludes all red points
- The hypotheses also will be rectangles (here  $R'$ ), which will in general have both false positive predictions (predicting blue when true label is red) and false negative predictions (predicting red when true label is blue).



## Example: learning axis-aligned rectangles

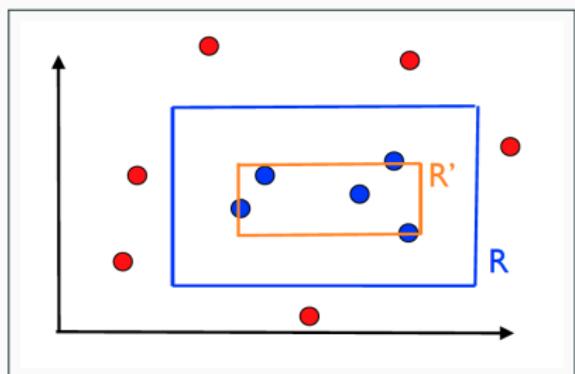
- We will use a simple algorithm: choose the tightest rectangle that contains all blue points
- Note that this will be a consistent hypothesis: no errors on the training data
- The hypothesis  $R'$  will only make false negative errors compared to the true concept  $R$ , no false positive errors (Q: Why is that?)



## Example: learning axis-aligned rectangles

Questions:

- Is the class of axis-aligned rectangles PAC-learnable?
- How much training data will be needed to learn?
- Need to bound the risk of outputting a bad hypothesis  $R(R') > \epsilon$  with high probability  $1 - \delta$
- We can assume  $Pr_D(R) > \epsilon$  (otherwise  $R(R') \leq \epsilon$  trivially)

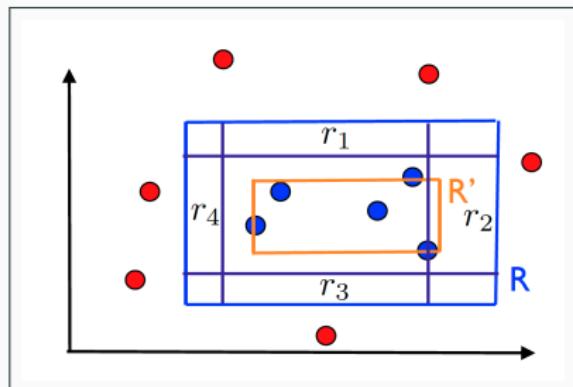


## Example: learning axis-aligned rectangles

Let  $r_1, r_2, r_3, r_4$  be rectangles along the sides of  $R$  such that  
 $\Pr_D(r_i) = \epsilon/4$

- Their union satisfies  
 $\Pr_D(r_1 \cup r_2 \cup r_3 \cup r_4) \leq \epsilon$
- Errors can only occur within  $R - R'$  (false negative predictions)
- If  $R'$  intersects all of the four regions  $r_1, \dots, r_4$  then we know that

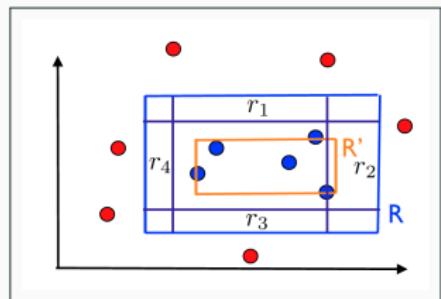
$$R(R') \leq \epsilon$$



Thus, if  $R(R') > \epsilon$  then  $R'$  must miss **at least one** of the four regions

## Example: learning axis-aligned rectangles

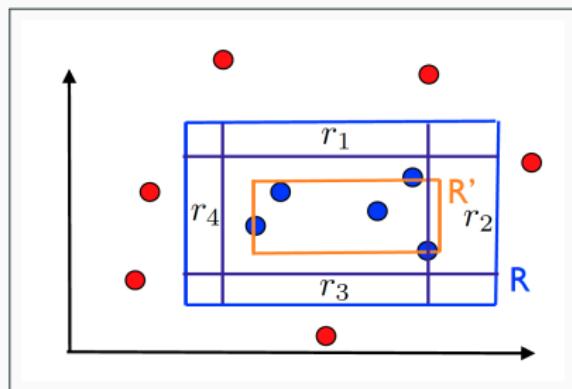
- Events  $A = \{R' \text{ intersects all four rectangles } r_1, \dots, r_4\}$ ,  
 $B = \{R(R') < \epsilon\}$ , satisfy  $A \subseteq B$
- Complement events  $A_C = \{R' \text{ misses at least one rectangle}\}$ ,  
 $B_C = \{R(R') \geq \epsilon\}$  satisfy  $B_C \subseteq A_C$
- $B_C$  is the bad event (high generalization error), we want it to have low probability
- In probability space, we have  
 $Pr(B_C) \leq Pr(A_C)$
- Our task is to upper bound  $Pr(A_C)$



## Example: learning axis-aligned rectangles

- Each  $r_i$  has probability mass  $\epsilon/4$  by our design
- Probability of one example missing one rectangle:  
 $1 - \epsilon/4$
- Probability of  $m$  examples missing one rectangle:  
 $(1 - \epsilon/4)^m$   
( $m$  times repeated trial with replacement)
- Probability of all examples missing at least one of the rectangles:

$$\Pr(A_C) \leq 4(1 - \epsilon/4)^m$$



## Example: learning axis-aligned rectangles

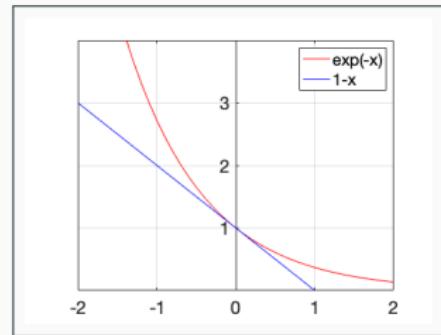
- We can use a general inequality  
 $\forall x : (1 - x) < \exp(-x)$  to obtain:

$$\Pr(R(h) \geq \epsilon) \leq 4(1 - \epsilon/4)^m \leq 4 \exp(-m\epsilon/4)$$

- We want this probability to be small ( $< \delta$ ):

$$4 \exp(-m\epsilon/4) < \delta$$

$$\Leftrightarrow m \geq 4/\epsilon \log 4/\delta$$

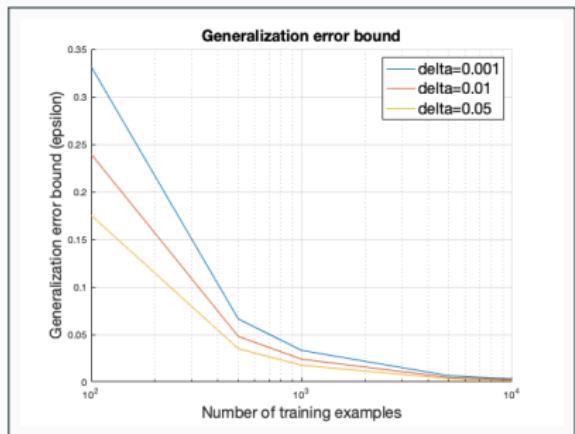
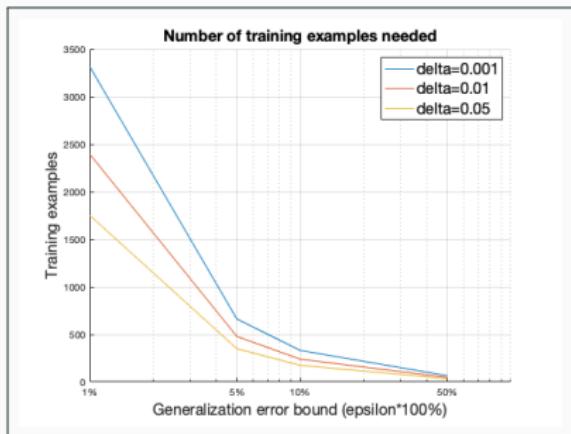


- The last inequality is our first generalization error bound, a **sample complexity** bound to be exact

Note: corresponding to Mohri et al (2018)  $\log$  denotes the natural logarithm:  $\log(\exp(x)) = x$

# Plotting the behaviour of bound

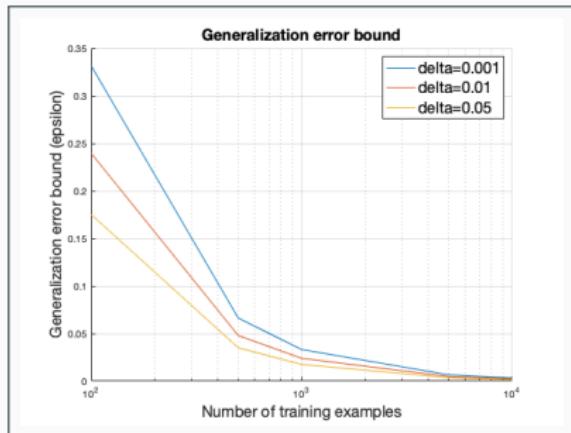
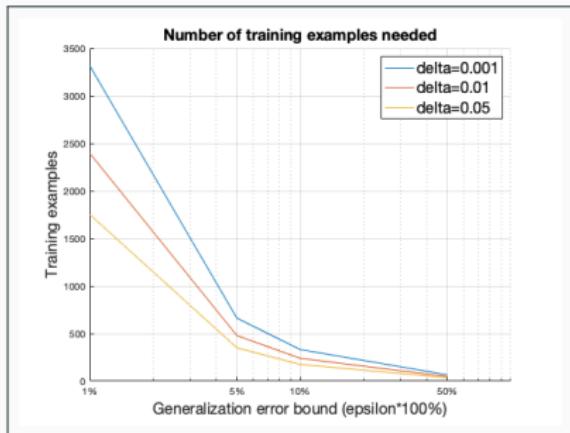
- Left, the sample complexity, the number of examples needed to reach a given generalization error level is shown  $m(\epsilon, \delta) = 4/\epsilon \log 4/\delta$
- Right, the generalization bound is plotted as a function of training sample size  $\epsilon(m, \delta) = 4/m \log 4/\delta$
- Three different confidence levels ( $\delta$ ) are plotted



# Plotting the behaviour of the bound

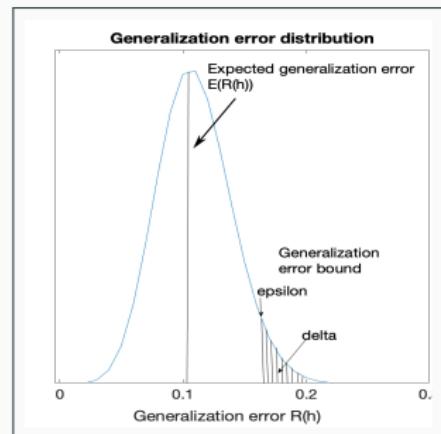
Typical behaviour of ML learning algorithms is revealed:

- increase of sample size decreases generalization error
- extra data gives less and less additional benefit as the sample size grows (law of diminishing returns)
- requiring high level of confidence (small  $\delta$ ) for obtaining low error requires more data for the same level of error



# Generalization error bound vs. expected test error

- The error bounds hold for any concepts from the class
  - including concepts that are harder to learn than "average concept"
- They hold for **any** distribution  $D$  generating the data
  - Including adversarially generated distributions (aiming to make learning harder)
- We bound the probability of being in the high error tail of the distribution (not the convergence to the mean or median generalization error)



For these reasons empirically estimated test errors might be considerably lower than the bounds suggest

## Half-time poll: Personalized email spam filtering system

Company is developing a personalized email spam filtering system. The system is tuned personally for each customer using the customers data on top of the existing training data. The company has a choice of three machine learning algorithms, with different performance characteristics. So far, the company has tested three different algorithms on a small set of test users.

Which algorithm should the company choose?

1. Algorithm 1, which guarantees error rate of less than 10% for 99% of the future customer base
2. Algorithm 2, which guarantees error rate of less than 5% for 90% of the future customer base
3. Algorithm 3, which empirically has error rate of 1% on the current user base

Answer to the poll in Mycourses by 11:15: Go to Lectures page and scroll down to "Lecture 2 poll": <https://mycourses.aalto.fi/course/view.php?id=37029&section=2>

## **Guarantees for finite hypothesis sets**

---

## Finite hypothesis classes

---

- Finite concept classes arise when:
  - Input variables have finite domains or they are converted to such in preprocessing (e.g. discretizing real values), and
  - The representations of the hypotheses have finite size (e.g. the number of times a single variable can appear)
  - Dealing with subclasses of Boolean formulae, expressions binary input variables (literals) combined with logical operators (AND, OR, NOT,...)
- Finite concept classes have been thoroughly analyzed hypothesis classes in statistical learning theory

## Example: Boolean conjunctions

- Aldo likes to do sport only when the weather is suitable
- Also has given examples of suitable and not suitable weather
- Let us build a classifier for Aldo to decide whether to do sports today
- As the classifier we use rules in the form of boolean conjunctions (boolean formulae containing AND, and NOT, but not OR operators): e.g. if (Sky=Sunny) AND NOT(Wind=Strong) then (EnjoySport=1)

t	$x^t$						$r(x^t)$ EnjoySport
	Sky	AirTemp	Humidity	Wind	Water	Forecast	
1	Sunny	Warm	Normal	Strong	Warm	Same	1
2	Sunny	Warm	High	Strong	Warm	Same	1
3	Rainy	Cold	High	Strong	Warm	Change	0
4	Sunny	Warm	High	Strong	Cool	Change	1

Table: Aldo's observed sport experiences in different weather conditions.

## Finite hypothesis class - consistent case

- Sample complexity bound relying on the size of the hypothesis class (Mohri et al, 2018):  $\Pr(R(h_s) \leq \epsilon) \geq 1 - \delta$  if

$$m \geq \frac{1}{\epsilon} (\log(|\mathcal{H}|) + \log(\frac{1}{\delta}))$$

- An equivalent generalization error bound:

$$R(h) \leq \frac{1}{m} (\log(|\mathcal{H}|) + \log(\frac{1}{\delta}))$$

- Holds for any finite hypothesis class assuming there is a consistent hypothesis, one with zero empirical risk
- Extra term compared to the rectangle learning example is the term  $\frac{1}{\epsilon} (\log(|\mathcal{H}|))$
- The more hypotheses there are in  $\mathcal{H}$ , the more training examples are needed

## Example: Boolean conjunctions

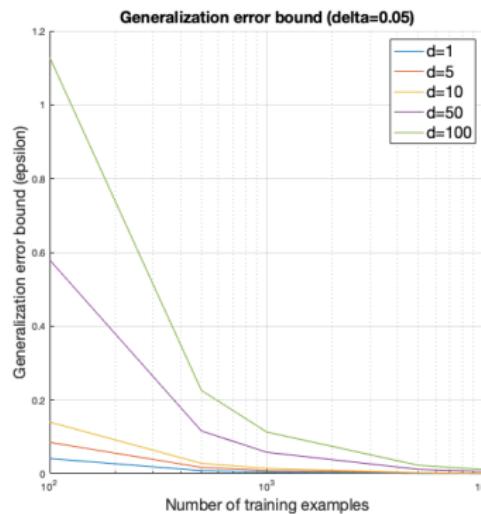
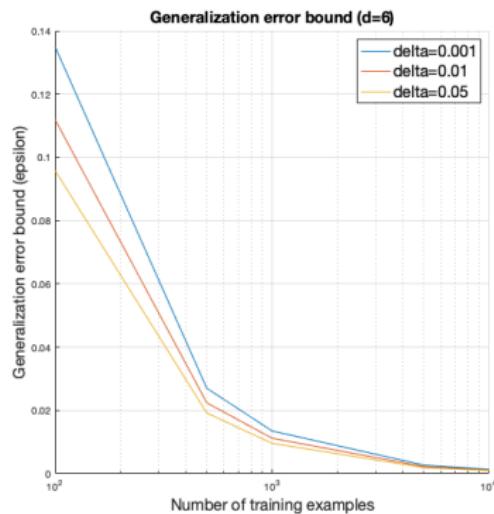
- How many different conjunctions can be built ( $=|\mathcal{H}|$ )
- Each variable can appear with or without "NOT" or can be excluded from the rule = 3 possibilities
- The total number of hypotheses is thus  $3^d$ , where  $d$  is the number of variables
- We have six variables in total, giving us  $|\mathcal{H}| = 3^6 = 729$  different hypotheses

t	$x^t$						$r(x^t)$ <i>EnjoySport</i>
	Sky	AirTemp	Humidity	Wind	Water	Forecast	
1	Sunny	Warm	Normal	Strong	Warm	Same	1
2	Sunny	Warm	High	Strong	Warm	Same	1
3	Rainy	Cold	High	Strong	Warm	Change	0
4	Sunny	Warm	High	Strong	Cool	Change	1

Table: Aldo's observed sport experiences in different weather conditions.

# Plotting the bound for Aldo's problem using boolean conjunctions

- On the left, the generalization bound is shown for different values of  $\delta$ , using  $d = 6$  variables
- On the right, the bound is shown for increasing number of input variables  $d$ , using  $\delta = 0.05$



## Arbitrary boolean formulae

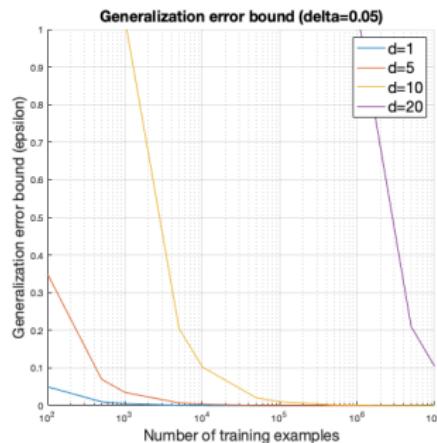
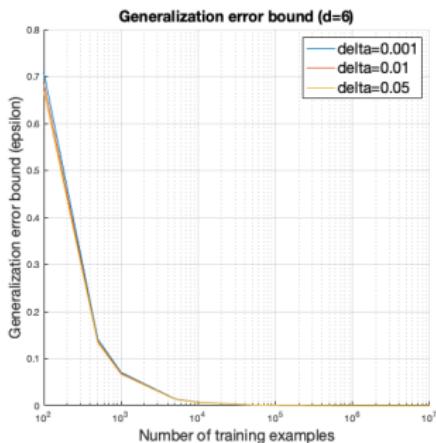
- What about using **arbitrary** boolean formulae?
- How many boolean formulae of  $d$  variables there are?
- There are  $2^d$  possible input vectors, size of the input space is  $|X| = 2^d$
- We can define a boolean formula that outputs 1 for an arbitrary subset of  $S \subset X$  and zero outside that subset:  
$$f_S(\mathbf{x}) = (\mathbf{x} = \mathbf{x}_1) OR (\mathbf{x} = \mathbf{x}_2) OR \cdots OR (\mathbf{x} = \mathbf{x}_{|S|})$$
- We can pick the subset in  $2^{|X|}$  ways (Why?)
- Thus we have  $|\mathcal{H}| = 2^{2^d}$  different boolean formula
- Our generalization bound gives

$$m \geq \frac{1}{\epsilon} (2^d \log 2 + \log(\frac{1}{\delta}))$$

- Thus we need exponential number of examples with respect to the number of variables; the hypothesis class is considered not PAC-learnable!

# Plotting the bound for Arbitrary boolean formulae

- With  $d = 6$  variables we need ca. 500 examples to get bound below 0.07 (left picture)
- Increase of number of variables quickly raises the sample complexity to  $10^6$  and beyond (right picture)



## **Proof outline of the PAC bound for finite hypothesis classes**

---

## Proof outline (Mohri et al., 2018)

---

- Consider any hypothesis  $h \in \mathcal{H}$  with  $R(h) > \epsilon$
- For  $h$  to be consistent  $\hat{R}(h) = 0$ , all training examples need to miss the region where  $h$  is making an error.
- The probability of this event is

$$Pr(\hat{R}(h) = 0 | R(h) > \epsilon) \leq (1 - \epsilon)^m$$

- $m$  times repeated trial with success probability  $\epsilon$
- This is the probability that one consistent hypothesis has high error

## Proof outline

---

- But we do not need which consistent hypothesis  $h$  is selected by our learning algorithm
- Hence our result will need to hold for all consistent hypotheses
  - This is an example of **uniform convergence** bound
- We wish to upper bound the probability that some  $h \in \mathcal{H}$  is consistent  $\hat{R}(h) = 0$  and has a high generalization error  $R(h) > \epsilon$  for a fixed  $\epsilon > 0$ :

$$\Pr(\exists h \in \mathcal{H} | \hat{R}(h) = 0 \wedge R(h) > \epsilon)$$

- Above  $\wedge$  is the logical "and"

## Proof outline

- We can replace  $\exists$  by enumerating all hypotheses in  $\mathcal{H}$  using logical-or ( $\vee$ )

$$\Pr(\exists h \in \mathcal{H} | \hat{R}(h) = 0 \wedge R(h) > \epsilon) =$$

$$\Pr(\{\hat{R}(h_1) = 0 \wedge R(h_1) > \epsilon\} \vee \{\hat{R}(h_2) = 0 \wedge R(h_2) > \epsilon\} \vee \dots)$$

- Using the fact that  $\Pr(A \cup B) \leq \Pr(A) + \Pr(B)$  and  $\Pr(A \cap C) \leq \Pr(A|C)$  for any events  $A, B$  and  $C$  the above is upper bounded by

$$\begin{aligned} &\leq \sum_{h \in \mathcal{H}} \Pr(\hat{R}(h) = 0 \wedge R(h) > \epsilon) \leq \sum_{h \in \mathcal{H}} \Pr(\hat{R}(h) = 0 | R(h) > \epsilon) \\ &\qquad\qquad\qquad \leq |\mathcal{H}|(1 - \epsilon)^m \end{aligned}$$

- Last inequality follows from using the  $\Pr(\hat{R}(h) = 0 | R(h_1) > \epsilon) \leq (1 - \epsilon)^m$  for the  $|\mathcal{H}|$  summands

## Proof outline

---

- We have established

$$\Pr(\exists h \in \mathcal{H} | \hat{R}(h) = 0 \wedge R(h) > \epsilon) \leq |\mathcal{H}|(1 - \epsilon)^m \leq |\mathcal{H}| \exp(-m\epsilon)$$

- Set the right-hand side equal to  $\delta$  and solve for  $m$  to obtain the bound:

$$\delta = |\mathcal{H}| \exp(-m\epsilon)$$

$$\log \delta = \log |\mathcal{H}| - m\epsilon$$

$$m = \frac{1}{\epsilon} (\log(|\mathcal{H}|) + \log(1/\delta))$$

## Finite hypothesis class - inconsistent case

- So far we have assumed that there is a consistent hypothesis  $h \in \mathcal{H}$ , one that achieves zero empirical risk on training sample
- In practise this is often not the case
- However as long as the empirical risk  $\hat{R}(h)$  is small, a low generalization error can still be achieved
- Generalization error bound (Mohri, et al. 2018): Let  $\mathcal{H}$  be a finite hypothesis set. Then for any  $\delta > 0$  with probability at least  $1 - \delta$  we have for all  $h \in \mathcal{H}$ :

$$R(h) \leq \hat{R}(h) + \sqrt{\frac{\log(|\mathcal{H}|) + \log(2/\delta)}{2m}}$$

- We see the dependency from  $\log |\mathcal{H}|/m$  as in the consistent case but now under square root
  - Slower convergence w.r.t number of examples

# Summary

- Probably approximately correct learning is a theoretical framework for analysing the generalization performance of machine learning algorithms
- PAC theory is concerned about upper bounding the probability  $\delta$  of "bad" events, those of high generalization error ( $\epsilon$ )
- In finite hypothesis classes, (the logarithm of) number of hypothesis  $\log |\mathcal{H}|$  in the hypothesis class affects the number of examples needed to obtain a given level of risk with high probability

# CS-E4710 Machine Learning: Supervised Methods

Lecture 3: Learning with infinite hypothesis classes

---

Juho Rousu

September 20, 2022

Department of Computer Science  
Aalto University

## Recall: PAC learnability

- A class  $C$  is PAC-learnable, if there exist an algorithm  $\mathcal{A}$  that given a training sample  $S$  outputs a hypothesis  $h_S$  that has generalization error satisfying

$$\Pr(R(h_S) \leq \epsilon) \geq 1 - \delta$$

- for any distribution  $D$ , for arbitrary  $\epsilon, \delta > 0$  and sample size  $m = |S|$  that grows at polynomially in  $1/\epsilon, 1/\delta$

## Recall: PAC learning of a finite hypothesis class

- Sample complexity bound relying on the size of the hypothesis class (Mohri et al, 2018):  $\Pr(R(h_s) \leq \epsilon) \geq 1 - \delta$  if

$$m \geq \frac{1}{\epsilon} (\log(|\mathcal{H}|) + \log(\frac{1}{\delta}))$$

- An equivalent generalization error bound:

$$R(h) \leq \frac{1}{m} (\log(|\mathcal{H}|) + \log(\frac{1}{\delta}))$$

- Holds for any finite hypothesis class assuming there is a consistent hypothesis, one with zero empirical risk
- Extra term compared to the rectangle learning example is the term  $\frac{1}{\epsilon} (\log(|\mathcal{H}|))$
- The more hypotheses there are in  $\mathcal{H}$ , the more training examples are needed

# Learning with infinite hypothesis classes

- The size of the hypothesis class is a useful measure of complexity for **finite** hypothesis classes (e.g boolean formulae)
- However, most classifiers used in practise rely on infinite hypothesis classes, e.g.
  - $\mathcal{H}$  = axis-aligned rectangles in  $\mathbb{R}^2$  (the example last lecture)
  - $\mathcal{H}$  = hyperplanes in  $\mathbb{R}^d$  (e.g. Support vector machines)
  - $\mathcal{H}$  = neural networks with continuous input variables
- Need better tools to analyze these cases

## Vapnik-Chervonenkis dimension

---

- VC dimension can be understood as measuring the capacity of a hypothesis class to adapt to different concepts
- It can be understood through the following thought experiment:
  - Pick a fixed hypothesis class  $\mathcal{H}$ , e.g. axis-aligned rectangles in  $R^2$
  - Let us enumerate all possible labelings of a training set of size  $m$ :  
 $\mathcal{Y}^m = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{2^m}\}$ , where  $\mathbf{y}_j = (y_{j1}, \dots, y_{jm})$ , and  $y_{ij} \in \{0, 1\}$  is the label of  $i$ 'th example in the  $j$ 'th labeling
  - We are allowed to freely choose a distribution  $D$  generating the inputs and to generate the input data  $x_1, \dots, x_m$
  - $VCdim(\mathcal{H}) = \text{size of the } \mathbf{\text{largest training set}} \text{ that we can find a consistent classifier for } \mathbf{\text{all labelings}} \text{ in } \mathcal{Y}^m$
- Intuitively:
  - low  $VCdim \implies$  easy to learn, low sample complexity
  - high  $VCdim \implies$  hard to learn, high sample complexity
  - infinite  $VCdim \implies$  cannot learn in PAC framework

# Shattering

- The underlying concept in VC dimension is **shattering**
- Given a set of points  $S = \{x_1, \dots, x_m\}$  and a fixed class of functions  $\mathcal{H}$
- $\mathcal{H}$  is said to **shatter**  $S$  if for any possible partition of  $S$  into positive  $S_+$  and negative subset  $S_-$  we can find a hypothesis for which  $h(x) = 1$  if and only if  $x \in S_+$

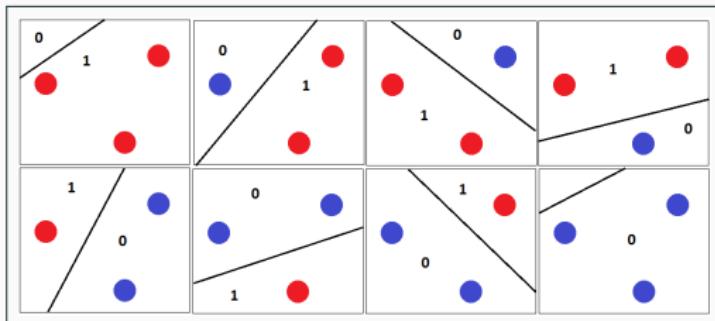


Figure source:

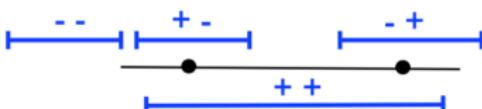
<https://datascience.stackexchange.com>

## How to show that $VCdim(\mathcal{H}) = d$

- How to show that  $VCdim(\mathcal{H}) = d$  for a hypothesis class
- We need to show two facts:
  1. There **exists a set of inputs** of size  $d$  that can be shattered by hypothesis in  $\mathcal{H}$  (i.e. we can pick the set of inputs any way we like):  
 $VCdim(\mathcal{H}) \geq d$
  2. There does not exist **any set of inputs** of size  $d + 1$  that can be shattered (i.e. need to show a general property):  $VCdim(\mathcal{H}) < d + 1$

## Example: intervals on a real line

- Let the hypothesis class be intervals in  $R$
- Each hypothesis is defined by two parameters  $b_h, e_h \in \mathbb{R}$ : the beginning and end of the interval,  $h(x) = \mathbf{1}_{b_h \leq x \leq e_h}$
- We can shatter any set of two points by changing the end points of the interval:



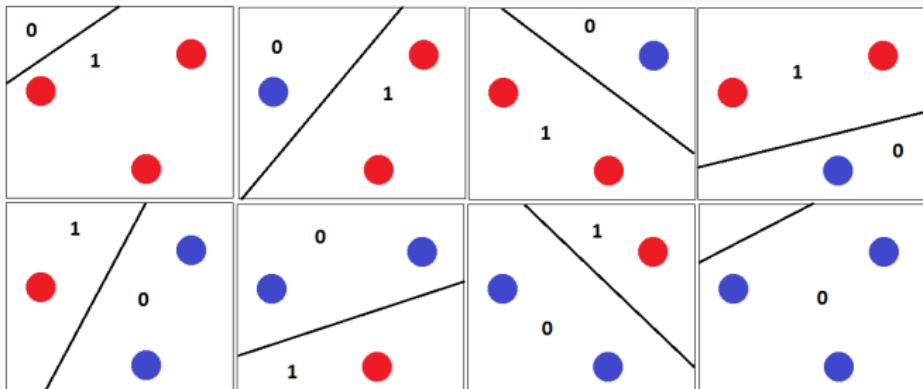
- We cannot shatter a three point set, as the middle point cannot be excluded while the left-hand and right-hand side points are included



We conclude that VC dimension for real intervals = 2

# Lines in $\mathbb{R}^2$

- A hypothesis class of lines  $h(x) = ax + b$  shatters a set of three points  $\mathbb{R}^2$ .

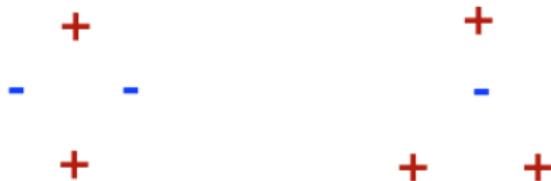


- We conclude that VC dimension is  $\geq 3$

## Lines in $\mathbb{R}^2$

Four points cannot be shattered by lines in  $\mathbb{R}^2$ :

- There are only two possible configurations of four points in  $\mathbb{R}^2$ :
  1. All four points reside on the convex hull
  2. Three points form the convex hull and one is in interior
- In the first case (left), we cannot draw a line separating the top and bottom points from the left-and right-hand side points
- In the second case, we cannot separate the interior point from the points on the convex hull with a line
- The two examples are sufficient to show that  $VCdim = 3$



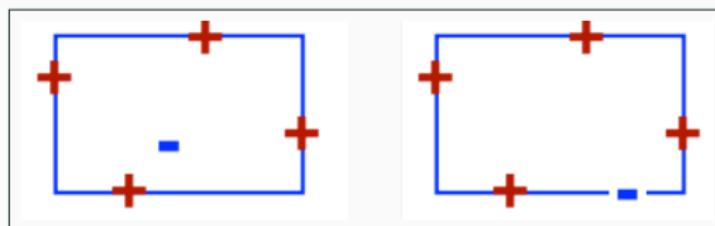
## VC-dimension of axis-aligned rectangles

- With axis aligned rectangles we can shatter a set of four points (picture shows 4 of the 16 configurations)
- This implies  $VCdim(\mathcal{H}) \geq 4$



## VC-dimension of axis-aligned rectangles

- For five distinct points, consider the minimum bounding box of the points
- There are two possible configurations:
  1. There are one or more points in the interior of the box: then one cannot include the points on the boundary and exclude the points in the interior
  2. At least one of the edges contains two points: in this case we can pick either of the two points and verify that this point cannot be excluded while all the other points are included
- Thus by the two examples we have established that  $VCdim(\mathcal{H}) = 4$



## Vapnik-Chervonenkis dimension formally

- Formally  $VCdim(\mathcal{H})$  is defined through the growth function

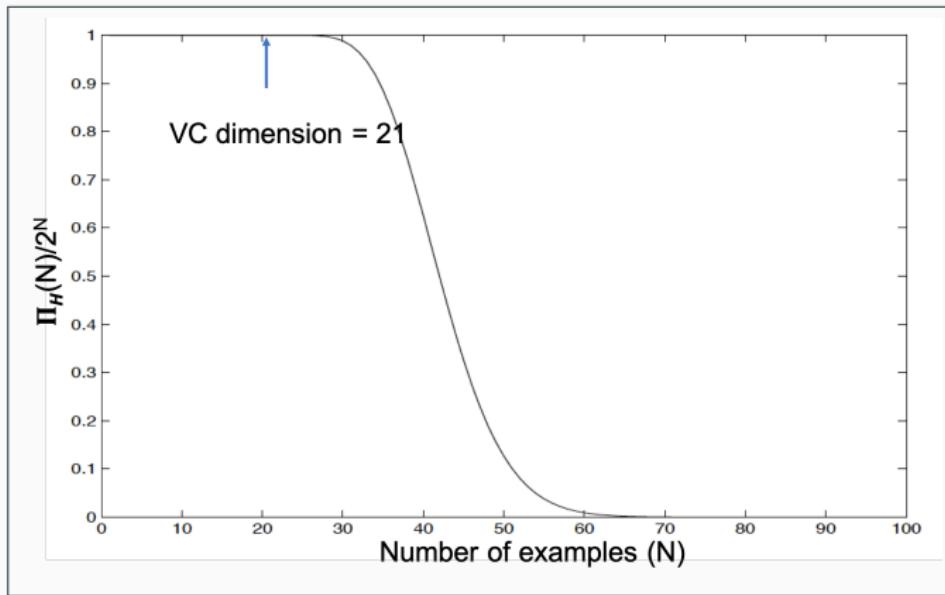
$$\Pi_{\mathcal{H}}(m) = \max_{\{x_1, \dots, x_m\} \subset X} |\{(h(x_1), \dots, h(x_m)) : h \in \mathcal{H}\}|$$

- The growth function gives the maximum number of unique labelings the hypothesis class  $\mathcal{H}$  can provide for an arbitrary set of input points
- The maximum of the growth function is  $2^m$  for a set of  $m$  examples
- Vapnik-Chervonenkis dimension is then

$$VCdim(\mathcal{H}) = \max_m \{ m | \Pi_{\mathcal{H}}(m) = 2^m \}$$

# Visualization

- The ratio of the growth function  $\Pi_{\mathcal{H}}(m)$  to the maximum number of labelings of a set of size  $m$  is shown
- Hypothesis class is 20-dimensional hyperplanes (VC dimension = 21)



## VC dimension of finite hypothesis classes

- A finite hypothesis class have VC dimension  $VCdim(\mathcal{H}) \leq \log_2 |\mathcal{H}|$
- To see this:
  - Consider a set of  $m$  examples  $S = \{x_1, \dots, x_m\}$
  - This set can be labeled  $2^m$  different ways, by choosing the labels  $y_i \in \{0, 1\}$  independently
  - Each hypothesis in  $h \in \mathcal{H}$  fixes one labeling, a length- $m$  binary vector  $\mathbf{y}(h, S) = (h(x_1), \dots, h(x_m))$
  - All hypotheses in  $\mathcal{H}$  together can provide at most  $|\mathcal{H}|$  different labelings in total (different vectors  $\mathbf{y}(h, S)$ ,  $h \in \mathcal{H}$ )
  - If  $|\mathcal{H}| < 2^m$  we cannot shatter  $S \implies$  we cannot shatter a set of size  $m > \log_2 |\mathcal{H}|$

## VC dimension: Further examples

Examples of classes with a finite VC dimension:

- convex  $d$ -polygons in  $\mathbb{R}^2$ :  $VCdim = 2d + 1$  (e.g. for general, not restricted to axis-aligned, rectangles  $VCdim = 5$ )
- hyperplanes in  $\mathbb{R}^d$ :  $VCdim = d + 1$  - (e.g. single neural unit, linear SVM)
- neural networks:  $VCdim = |E| \log |E||$  where  $E$  is the set of edges in the networks (for *sign* activation function)
- boolean monomials of  $d$  variables:  $VCdim = d$
- arbitrary boolean formulae of  $d$  variables:  $VCdim = 2^d$

## Zoom poll: VC dimension of threshold functions in $\mathbb{R}$

Consider a hypothesis class  $\mathcal{H} = \{h_\theta\}$  of threshold functions  
 $h_\theta : \mathbb{R} \mapsto \{0, 1\}, \theta \in \mathbb{R} :$

$$h_\theta(x) = \begin{cases} 1 & \text{if } x > \theta \\ 0 & \text{otherwise} \end{cases}$$

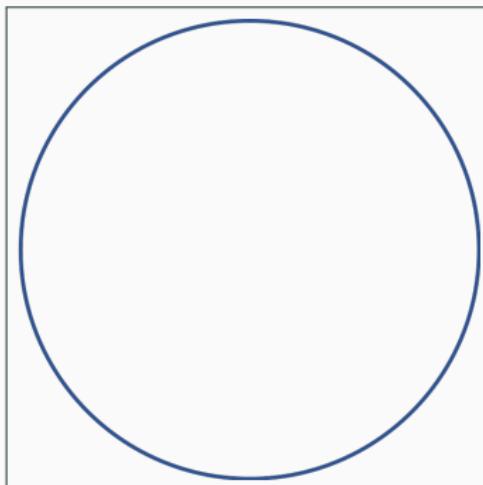
What is the VC dimension of this hypothesis class?

1.  $VCdim = 1$
2.  $VCdim = 2$
3.  $VCdim = \infty$

Answer in the zoom poll. Answers are anonymous and do not affect grading of the course.

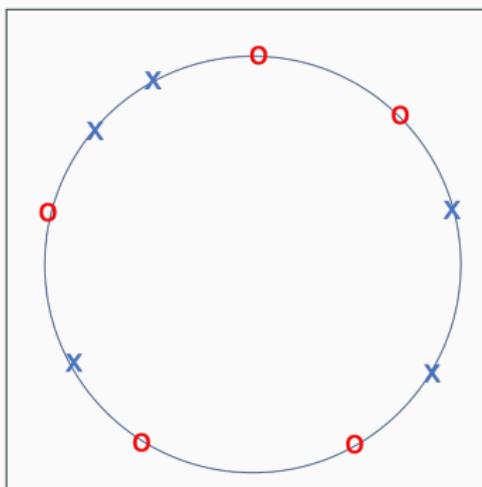
## Convex polygons have VC dimension = $\infty$

- Let our hypothesis class be convex polygons in  $\mathbb{R}^2$  without restriction of number of vertices  $d$
- Let us draw an arbitrary circle on  $\mathbb{R}^2$  - the distribution  $D$  will be concentrated on the circumference of the circle
  - This is a difficult distribution for learning polygons - we choose it on purpose



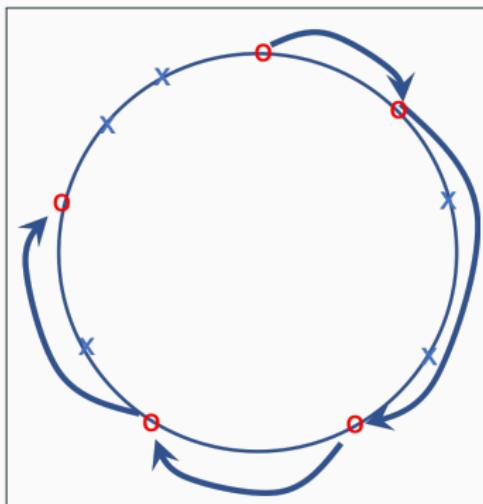
## Convex polygons have VC dimension = $\infty$

- Let us consider a set of  $m$  points with arbitrary binary labels
- For any  $m$ , let us position  $m$  points on the circumference of the circle
  - simulating drawing the inputs from the distribution  $D$



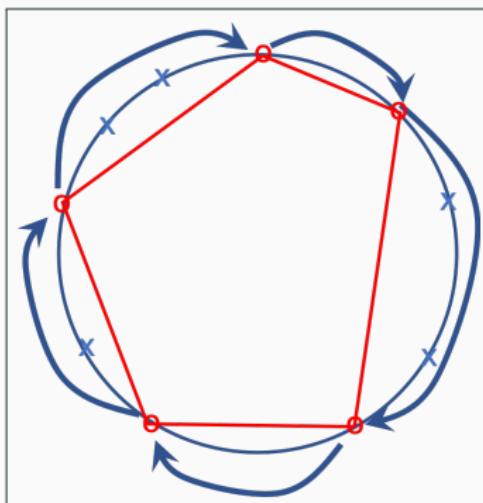
## Convex polygons have VC dimension = $\infty$

- Start from an arbitrary positive point (red circles)
- Traverse the circumference clockwise skipping all negative points and stopping at positive points



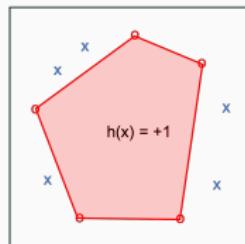
## Convex polygons have VC dimension = $\infty$

- Connect adjacent positive points with an edge
- This forms a  $p$ -polygon inside the circle, where  $p$  is the number of positive data points



## Convex polygons have VC dimension = $\infty$

- Define  $h(x) = +1$  for points inside the polygon and  $h(x) = 0$  outside
- Each of the  $2^m$  labelings of  $m$  examples gives us a  $p$ -polygon that includes the  $p$  positive points in that labeling and excludes the negative points  $\implies$  we can shatter a set of size  $m$ :  $VCdim(\mathcal{H}) \geq m$
- Since  $m$  was arbitrary, we can grow it without limit  $VCdim(\mathcal{H}) = \infty$



## Generalization bound based on the VC-dimension

- (Mohri, 2018) Let  $\mathcal{H}$  be a family of functions taking values in  $\{-1, +1\}$  with VC-dimension  $d$ . Then for any  $\delta > 0$ , with probability at least  $1 - \delta$  the following holds for all  $h \in \mathcal{H}$ :

$$R(h) \leq \hat{R}(h) + \sqrt{\frac{2 \log(em/d)}{m/d}} + \sqrt{\frac{\log(1/\delta)}{2m}}$$

- $e \approx 2.71828$  is the base of the natural logarithm
- The bound reveals that the critical quantity is  $m/d$ , i.e. the number of examples divided by the VC-dimension
- Manifestation of the Occam's razor principle: to justify an increase in the complexity, we need reciprocally more data

## Rademacher complexity

---

## Experiment: how well does your hypothesis class fit noise?

- Consider a set of training examples  $S_0 = \{(x_i, y_i)\}_{i=1}^m$
- Generate  $M$  new datasets  $S_1, \dots, S_M$  from  $S_0$  by randomly drawing a new label  $\sigma \in \mathcal{Y}$  for each training example in  $S_0$

$$S_k = \{(x_i, \sigma_{ik})\}_{i=1}^m$$

- Train a classifier  $h_k$  minimizing the empirical risk on training set  $S_k$ , record its empirical risk

$$\hat{R}(h_k) = \frac{1}{m} \sum_{i=1}^m \mathbf{1}_{h_k(x_i) \neq \sigma_{ik}}$$

- Compute the average empirical risk over all datasets:

$$\bar{\epsilon} = \frac{1}{M} \sum_{k=1}^M \hat{R}(h_k)$$

## Experiment: how well does your hypothesis class fit noise?

- Observe the quantity

$$\hat{\mathcal{R}} = \frac{1}{2} - \bar{\epsilon}$$

- We have  $\hat{\mathcal{R}} = 0$  when  $\bar{\epsilon} = 0.5$ , that is when the predictions correspond to random coin flips (0.5 probability to predict either class)
- We have  $\hat{\mathcal{R}} = 0.5$  when  $\bar{\epsilon} = 0$ , that is when all hypotheses  $h_i, i = 1, \dots, M$  have zero empirical error (perfect fit to noise, not good!)
- Intuitively we would like our hypothesis
  - to be able to separate noise from signal - to have low  $\hat{\mathcal{R}}$
  - have low empirical error on real data - otherwise impossible to obtain low generalization error

## Rademacher complexity

---

- Rademacher complexity defines complexity as the capacity of hypothesis class to fit random noise
- For binary classification with labels  $\mathcal{Y} = \{-1, +1\}$  empirical Rademacher complexity can be defined as

$$\hat{\mathcal{R}}_S(\mathcal{H}) = \frac{1}{2} E_{\sigma} \left( \sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{t=1}^m \sigma^i h(\mathbf{x}_i) \right)$$

- $\sigma_i \in \{-1, +1\}$  are Rademacher random variables, drawn independently from uniform distribution (i.e.  $Pr\{\sigma = 1\} = 0.5$ )
- Expression inside the expectation takes the highest correlation over all hypothesis in  $h \in \mathcal{H}$  between the random true labels  $\sigma_i$  and predicted label  $h(\mathbf{x}_i)$

# Rademacher complexity

$$\hat{\mathcal{R}}_S(\mathcal{H}) = \frac{1}{2} E_{\sigma} \left( \sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m \sigma^i h(\mathbf{x}_i) \right)$$

- Let us rewrite  $\hat{\mathcal{R}}_S(\mathcal{H})$  in terms of empirical error
- Note that with labels  $\mathcal{Y} = \{+1, -1\}$ ,

$$\sigma_i h(\mathbf{x}_i) = \begin{cases} 1 & \text{if } \sigma_t = h(\mathbf{x}_i) \\ -1 & \text{if } \sigma_i \neq h(\mathbf{x}_i) \end{cases}$$

- Thus

$$\begin{aligned} \frac{1}{m} \sum_{i=1}^m \sigma_i h(\mathbf{x}_i) &= \frac{1}{m} \left( \sum_i \mathbf{1}_{\{h(\mathbf{x}_i) = \sigma_i\}} - \sum_i \mathbf{1}_{\{h(\mathbf{x}_i) \neq \sigma_i\}} \right) \\ &= \frac{1}{m} (m - 2 \sum_i \mathbf{1}_{\{h(\mathbf{x}_i) \neq \sigma_i\}}) = 1 - 2\hat{\epsilon}(h) \end{aligned}$$

# Rademacher complexity

- Plug in

$$\begin{aligned}\hat{\mathcal{R}}_S(\mathcal{H}) &= \frac{1}{2} E_{\sigma} \left( \sup_{h \in \mathcal{H}} (1 - 2\hat{\epsilon}(h)) \right) \\ &= \frac{1}{2} (1 - 2E_{\sigma} \inf_{h \in \mathcal{H}} \hat{\epsilon}(h)) = \frac{1}{2} - E_{\sigma} \inf_{h \in \mathcal{H}} \hat{\epsilon}(h)\end{aligned}$$

- Now we have expressed the empirical Rademacher complexity in terms of expected empirical error of classifying randomly labeled data
- But how does the Rademacher complexity help in model selection?
  - We need to relate it to generalization error

## Generalization bound with Rademacher complexity

(Mohri et al. 2018): For any  $\delta > 0$ , with probability at least  $1 - \delta$  over a sample drawn from an unknown distribution  $D$ , for any  $h \in \mathcal{H}$  we have:

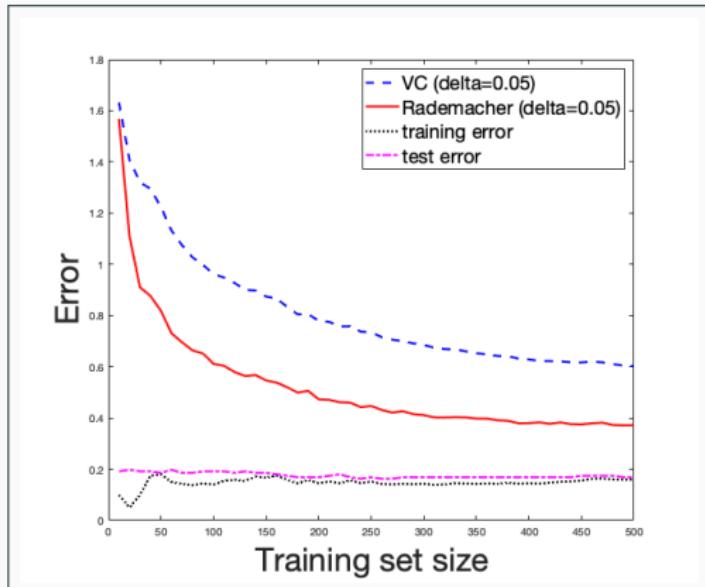
$$R(h) \leq \hat{R}_S(h) + \hat{\mathcal{R}}_S(\mathcal{H}) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}}$$

The bound is composed of the sum of :

- The empirical risk of  $h$  on the training data  $S$  (with the original labels):  $\hat{R}_S(h)$
- The empirical Rademacher complexity:  $\hat{\mathcal{R}}_S(\mathcal{H})$
- A term that tends to zero as a function of size of the training data as  $O(1/\sqrt{m})$  assuming constant  $\delta$ .

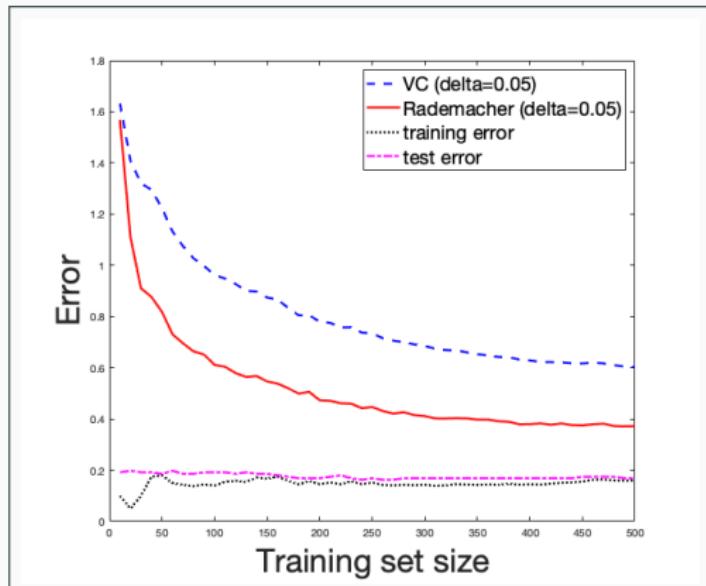
## Example: Rademacher and VC bounds on a real dataset

- Prediction of protein subcellular localization
- 10-500 training examples, 172 test examples
- Comparing Rademacher and VC bounds using  $\delta = 0.05$
- Training and test error also shown



## Example: Rademacher and VC bounds on a real dataset

- Rademacher bound is sharper than the VC bound
- VC bound is not yet informative with 500 examples ( $> 0.5$ ) using ( $\delta = 0.05$ )
- The gap between the mean of the error distribution ( $\approx$  test error) and the 0.05 probability tail (VC and Rademacher bounds) is evident (and expected)



## Rademacher vs. VC

---

Note the differences between Rademacher complexity and VC dimension

- VC dimension is independent of any training sample or distribution generating the data: it measures the worst-case where the data is generated in a bad way for the learner
- Rademacher complexity depends on the training sample thus is dependent on the data generating distribution
- VC dimension focuses the extreme case of realizing all labelings of the data
- Rademacher complexity measures smoothly the ability to realize random labelings

## Rademacher vs. VC

- Generalization bounds based on Rademacher Complexity are applicable to any binary classifiers (SVM, neural network, decision tree)
- It motivates state of the art learning algorithms such as support vector machines
- But computing it might be hard, if we need to train a large number of classifiers
- Vapnik-Chervonenkis dimension (VCdim) is an alternative that is usually easier to derive analytically

## Summary: Statistical learning theory

- Statistical learning theory focuses in analyzing the generalization ability of learning algorithms
- Probably Approximately Correct framework is the most studied theoretical framework, asking for bounding the generaliation error ( $\epsilon$ ) with high probability  $(1 - \delta)$ , with arbitrary level of error  $\epsilon > 0$  and confidence  $\delta > 0$
- Vapnik-Chervonenkis dimension lets us study learnability infinite hypothesis classes through the concept of shattering
- Rademacher complexity is a practical alternative to VC dimension, giving typically sharper bounds (but requires a lot of simulations to be run)

# CS-E4710 Machine Learning: Supervised Methods

## Lecture 4: Model selection

---

Juho Rousu

September 27, 2022

Department of Computer Science  
Aalto University

# Model selection and Occam's Razor

- Occam's razor principle "Entities should not be multiplied unnecessarily" captures the trade-off between generalization error and complexity
- Model selection in machine learning can be seen to implement Occam's razor



William of Ockham  
(1285–1347) "Pluralitas  
non est ponenda sine  
necessitate"

## Stochastic scenario

- The analysis so far assumed that the labels are deterministic functions of the input
- Stochastic scenario relaxes this assumption by assuming the output is a probabilistic function of the input
- The input and output is generated by a joint probability distribution  $D$  or  $X \times \mathcal{Y}$ .
- This setup covers different cases when the same input  $x$  can have different labels  $y$
- In the stochastic scenario, there **may not always exist** a target concept  $f$  that has zero generalization error  $R(f) = 0$

## Sources of stochasticity

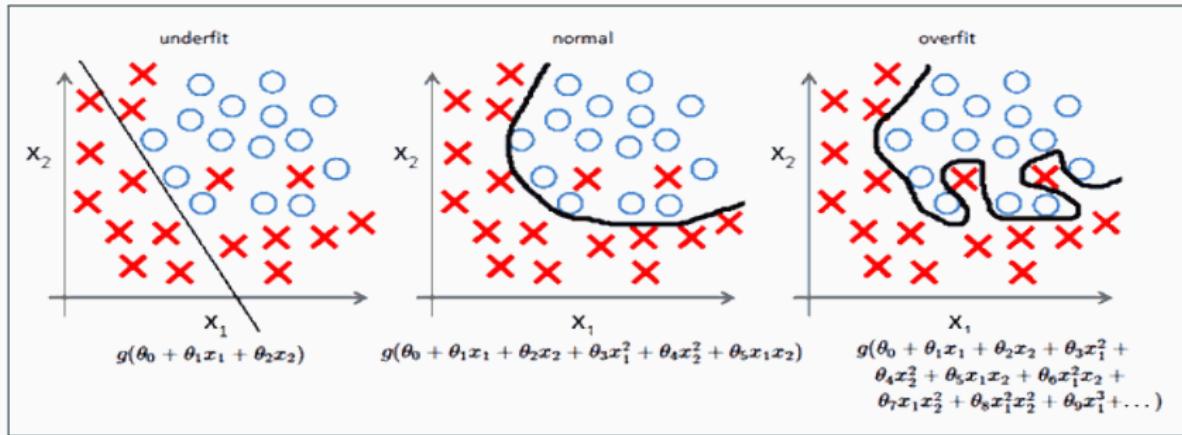
The stochastic dependency between input and output can arise from various sources

- Imprecision in recording the input data (e.g. measurement error), shifting our examples in the input space
- Errors in the labeling of the training data (e.g. human annotation errors), flipping the labels some examples
- There may be additional variables that affect the labels that are not part of our input data

All of these sources could be characterized as adding noise (or hiding signal)

# Noise and complexity

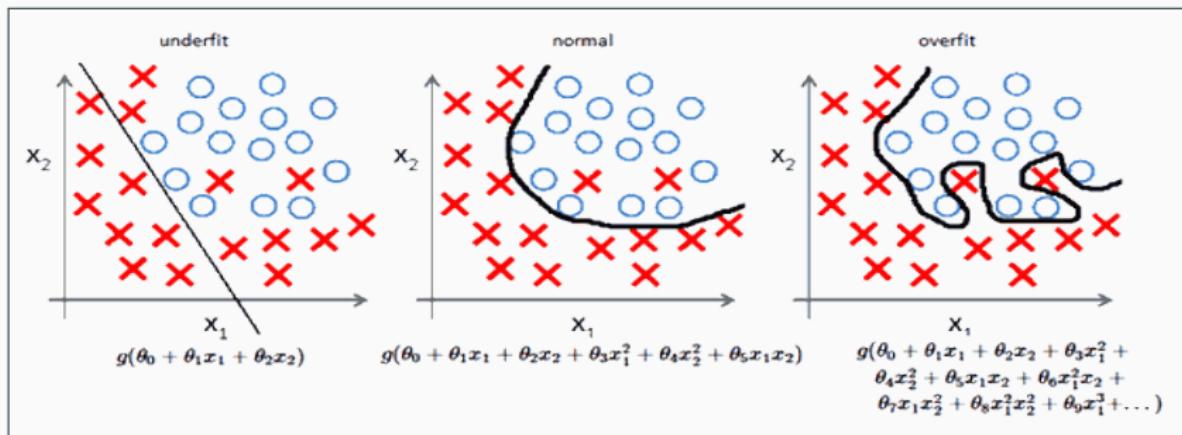
- The effect of noise is typically to make the decision boundary more complex
- To obtain a consistent hypothesis on noisy data, we can use a more complex model e.g. a spline curve instead of a hyperplane
- But this may not give a better generalization error, if we end up merely re-classifying points corrupted by noise



# Noise and complexity

In practice, we need to balance the complexity of the hypothesis and the empirical error carefully

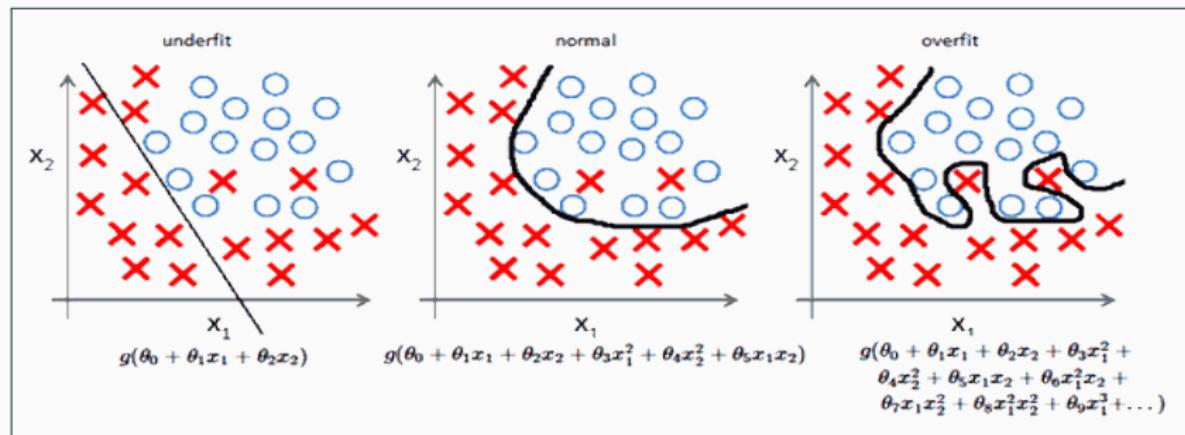
- A too simple model does not allow optimal empirical error to be obtained, this is called underfitting
- A too complex model may obtain zero empirical error, but have worse than optimal generalization error, this is called overfitting



# Controlling complexity

Two general approaches to control the complexity

- Selecting a hypothesis class, e.g. the maximum degree of polynomial to fit the regression model
- Regularization: penalizing the use of too many parameters, e.g. by bounding the norm of the weights (used in SVMs and neural networks)



# Measuring complexity

What is a good measure of complexity of a hypothesis class?

We have already looked at some measures:

- Number of distinct hypotheses  $|\mathcal{H}|$ : works for finite  $\mathcal{H}$  (e.g. models build from binary data), but not for infinite classes (e.g. geometric hypotheses such as polygons, hyperplanes, ellipsoids)
- Vapnik-Chervonenkis dimension (VCdim): the maximum number of examples that can be classified in all possible ways by choosing different hypotheses  $h \in \mathcal{H}$
- Rademacher complexity: measures the capability to classify after randomizing the labels

Lots of other complexity measures and model selection methods exist c.f.  
[https://en.wikipedia.org/wiki/Model\\_selection](https://en.wikipedia.org/wiki/Model_selection) (these are not  
in the scope of this course)

## Bayes error

---

## Bayes error

- In the stochastic scenario, there is a minimal non-zero error for any hypothesis, called the **Bayes error**
- Bayes error is the minimum achievable error, given a distribution  $D$  over  $X \times \mathcal{Y}$ , by measurable functions  $h : X \mapsto \mathcal{Y}$

$$R^* = \inf_{\{h | h \text{ measurable}\}} R(h)$$

- Note that we cannot actually compute  $R^*$ :
  - We cannot compute the generalization error  $R(h)$  exactly (c.f. PAC learning)
  - We cannot evaluate all measurable functions (intuitively: hypothesis class that contains all functions that are mathematically well-behaved enough to allow us to define probabilities on them)
- Bayes error serves us a theoretical measure of best possible performance

## Bayes error and noise

- A hypothesis with  $R(h) = R^*$  is called the **Bayes classifier**
- The Bayes classifier can be defined in terms of conditional probabilities as

$$h_{\text{Bayes}}(x) = \operatorname{argmax}_{y \in \{0,1\}} Pr(y|x)$$

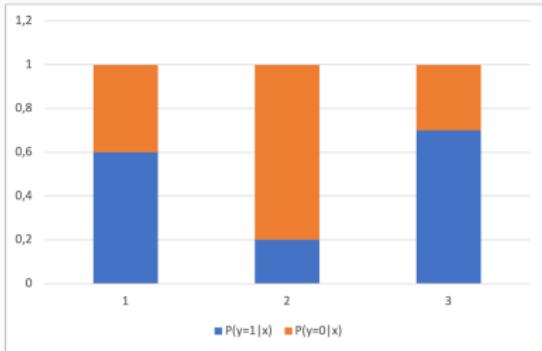
- The average error made by the Bayes classifier at  $x \in X$  is called the **noise**

$$\text{noise}(x) = \min(Pr(1|x), Pr(0|x))$$

- Its expectation  $E(\text{noise}(x)) = R^*$  is the Bayes error
- Similarly to the Bayes error, Bayes classifier is a theoretical tool, not something we can compute in practice

# Bayes error example

- We have a univariate input space:  
 $x \in \{1, 2, 3\}$ , and two classes  
 $y \in \{0, 1\}$ .
- We assume a uniform distribution of data:  $Pr(x) = 1/3$
- The Bayes classifier will predict the most probable class for each possible input value



X	1	2	3
$P(Y = 1 X)$	0.6	0.2	0.7
$P(Y = 0 X)$	0.4	0.8	0.3
$h_{Bayes}(x)$	1	0	1
$noise(x)$	0.4	0.2	0.3

The Bayes error is the expectation of the noise over the input domain

$$R^* = \sum_x P(x) noise(x) = 1/3 (0.4 + 0.2 + 0.3) = 0.3$$

## Decomposing the error of a hypothesis

The **excess error** of a hypothesis compared to the Bayes error  $R^*$  can be decomposed as:

$$R(h) - R^* = \epsilon_{\text{estimation}} + \epsilon_{\text{approximation}}$$

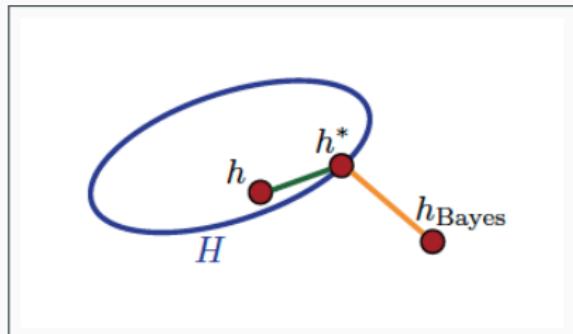
- $\epsilon_{\text{estimation}} = R(h) - R(h^*)$  is the excess generalization error  $h$  has over the optimal hypothesis  $h^* = \operatorname{argmin}_{h' \in \mathcal{H}} R(h')$  in the hypothesis class  $\mathcal{H}$
- $\epsilon_{\text{approximation}} = R(h^*) - R^*$  is the approximation error due to selecting the hypothesis class  $\mathcal{H}$  instead of the best possible hypothesis class (which is generally unknown to us)

Note: The approximation error is sometimes called the **bias** and the estimation error the **variance**, and the decomposition **bias-variance decomposition**

## Decomposing the error of a hypothesis

Figure on the right depicts the concepts:

- $h_{Bayes}$  is the Bayes classifier, with  
 $R(h_{Bayes}) = R^*$
- $h^* = \inf_{h \in \mathcal{H}} R(h)$  is the hypothesis with the lowest generalization error in the hypothesis class  $\mathcal{H}$
- $R(h)$  has both non-zero estimation error  $R(h) - R(h^*)$  and approximation error  $R(h^*) - R(h_{Bayes})$



## Example: Approximation error

- Assume the hypothesis class of univariate threshold functions:

$$\mathcal{H} = \{h_{a,\theta} : X \mapsto \{0, 1\} | a \in \{-1, +1\}, \theta \in \mathbb{R}\},$$

where  $h_{a,\theta}(x) = \mathbf{1}_{ax \geq \theta}$

- The classifier from  $\mathcal{H}$  with the lowest generalization error separates the data with  $x = 3$  from data with  $x < 3$  by, e.g. choosing  $a = 1, \theta = 2.5$ , giving  $h^*(1) = h^*(2) = 0, h^*(3) = 1$

- The generalization error is  
 $R(h^*) =$   
 $1/3(0.6 + 0.2 + 0.3) \approx 0.367$

X	1	2	3
$P(y = 1 x)$	0.6	0.2	0.7
$P(y = 0 x)$	0.4	0.8	0.3
$h^*(x)$	0	0	1

- Approximation error satisfies:

$$\epsilon_{approximation} = R(h^*) - R^* \approx 0.367 - 0.3 \approx 0.067$$

## Example: estimation error

- Consider now a training set on the right consisting of  $m = 13$  examples from the same distribution as before

x	1	2	3
y=1	3	1	3
y=0	1	3	2
$\Sigma$	4	4	5

- The classifier from  $h \in \mathcal{H}$  with the lowest training error  $\hat{R}(h) = 0.385$  separates the data with  $x = 1$  from data with  $x > 1$  by, e.g. choosing  $a = -1, \theta = 1.5$ , giving  $h(1) = 1, \hat{h}(2) = h(3) = 0$
- However, the generalization error of  $h$  is  $R(h) = \frac{1}{3} (0.4 + 0.2 + 0.7) \approx 0.433$
- The estimation error satisfies

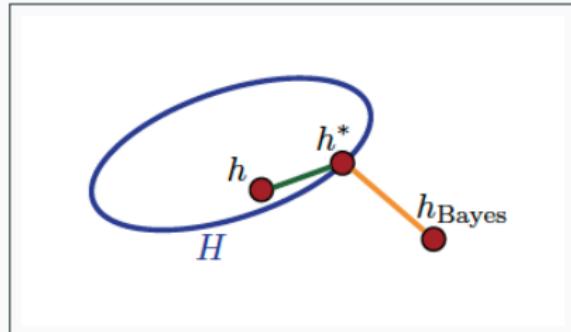
$$\epsilon_{estimation} = R(h) - R(h^*) \approx 0.433 - 0.367 \approx 0.067$$

- The decomposition of the error of  $h$  is given by:

$$R(h) = R^* + \epsilon_{approximation} + \epsilon_{estimation} \approx 0.3 + 0.067 + 0.067 \approx 0.433$$

# Error decomposition and model selection

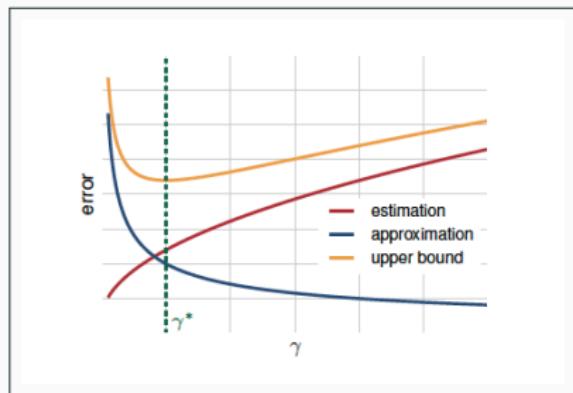
- We can bound the estimation error  
 $\epsilon_{estimation} = R(h) - R(h^*)$  by generalization bounds arising from the PAC theory
- However, we cannot do the same for the approximation error since  $\epsilon_{approximation} = R(h^*) - R^*$  remains unknown to us
- In other words, we do not know how good the hypothesis class is for approximating the label distribution



# Complexity and model selection

In model selection, we have a trade-off: increasing the complexity of the hypothesis class

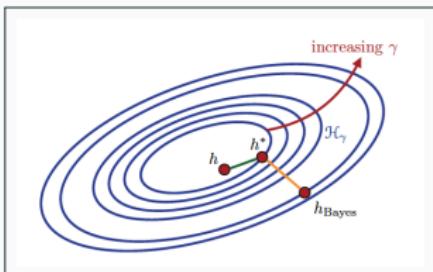
- decreases the approximation error as the class is more **likely to contain** a hypothesis with error close to the Bayes error
- increases the estimation error as **finding the good hypothesis** becomes more hard and the generalization bounds become looser(due to increasing  $\log |\mathcal{H}|$  or the VC dimension)



To minimize the generalization error over all hypothesis classes, we should find a balance between the two terms

# Learning with complex hypothesis classes

- One strategy for model selection to initially choose a very complex hypothesis class with zero or very low empirical risk  $\mathcal{H}$
- Assume in addition the class can be decomposed into a union of increasingly complex hypothesis classes  $\mathcal{H} = \bigcup_{\gamma \in \Gamma} \mathcal{H}_\gamma$ , parametrized by  $\gamma$ , e.g.
  - $\gamma =$  number of variables in a boolean monomial
  - $\gamma =$  degree of a polynomial function
  - $\gamma =$  size of a neural network
  - $\gamma =$  regularization parameter penalizing large weights
- We expect the approximation error to go down and the estimation error to up when  $\gamma$  increases
- Model selection entails choosing  $\gamma^*$  that gives the best trade-off



## Half-time poll: Inequalities on different errors

Let  $R^*$  denote the Bayes error,  $R(h)$  the generalization error and  $\hat{R}_S(h)$  the training error of hypothesis  $h$  on training set  $S$ .

Which of the inequalities always hold true?

1.  $R^* \leq \hat{R}_S(h)$
2.  $\hat{R}_S(h) \leq R(h)$
3.  $R^* \leq R(h)$

Answer to the poll in Mycourses by 11:15: Go to Lectures page and scroll down to "Lecture 4 poll": <https://mycourses.aalto.fi/course/view.php?id=37029&section=2>

Answers are anonymous and do not affect grading of the course.

## **Regularization-based algorithms**

---

# Regularization-based algorithms

- Regularization is technique that penalizes large weights in a model based on weighting input features:
  - linear regression, support vector machines and neural networks are fall under this category
- An important example is the class of linear functions  $\mathbf{x} \mapsto \mathbf{w}^T \mathbf{x}$
- The classes as parametrized by the norm  $\|\mathbf{w}\|$  of the weight vector bounded by  $\gamma$ :  $\mathcal{H}_\gamma = \{\mathbf{x} \mapsto \mathbf{w}^T \mathbf{x} : \|\mathbf{w}\| \leq \gamma\}$
- The norm is typically either
  - $L^2$  norm (Also called Euclidean norm or 2-norm):  
$$\|\mathbf{w}\|_2 = \sqrt{\sum_{j=1}^n w_j^2}$$
: used e.g. in support vector machines and ridge regression
  - $L^1$  norm (Also called Manhattan norm or 1-norm):  
$$\|\mathbf{w}\|_1 = \sum_{j=1}^n |w_j|$$
: used e.g. in LASSO regression

## Regularization-based algorithms

- For the  $L^2$ -norm case, we have an important computational shortcut: the empirical Rademacher complexity of this class can be bounded analytically!
- Let  $S \subset \{\mathbf{x} \mid \|\mathbf{x}\| \leq r\}$  be a sample of size  $m$  and let  $\mathcal{H}_\gamma = \{\mathbf{x} \mapsto \mathbf{w}^T \mathbf{x} : \|\mathbf{w}\|_2 \leq \gamma\}$ . Then

$$\hat{\mathcal{R}}_S(\mathcal{H}_\gamma) \leq \sqrt{\frac{r^2 \gamma^2}{m}} = \frac{r\gamma}{\sqrt{m}}$$

- Thus the Rademacher complexity depends linearly on the upper bound  $\gamma$  norm of the weight vector, as  $r$  and  $m$  are constant for any fixed training set
- We can use  $\|\mathbf{w}\|$  as an efficiently computable upper bound of  $\hat{\mathcal{R}}_m(\mathcal{H}_\gamma)$

## Regularization-based algorithms

- A regularized learning problem is to minimize

$$\operatorname{argmin}_{h \in \mathcal{H}} \hat{R}_S(h) + \lambda \Omega(h)$$

- $\hat{R}_S(h)$  is the empirical error
- $\Omega(h)$  is the regularization term which increases when the complexity of the hypothesis class increases
- $\lambda$  is a regularization parameter, which is usually set by cross-validation
- For the linear functions  $h : \mathbf{x} \mapsto \mathbf{w}^T \mathbf{x}$ , usually  $\Omega(h) = \|\mathbf{w}\|_2^2$  or  $\Omega(h) = \|\mathbf{w}\|_1$
- We will study regularization-based algorithms during the next part of the course

## **Model selection using a validation set**

---

## Model selection by using a validation set

We can use the given dataset for empirical model selection, if the algorithm has input parameters (hyperparameters) that define/affect the model complexity

- Split the data into training, validation and test sets
- For the hyperparameters, use **grid search** to find the parameter combination that gives the best performance on the validation set
- Retrain a final model using the optimal parameter combination, use both the training and validation data for training
- Evaluate the performance of the final model **on the test set**



# Grid search

- Grid search is a technique frequently used to optimize hyperparameters, including those that define the complexity of the models
- In its basic form it goes through all combinations of parameter values, given a set of candidate values for each parameter
- For two parameters, taking of value combinations  $(v, u) \in V \times U$ , where  $V$  and  $U$  are the sets of values for the two parameters, defines a two-dimensional **grid** to be searched
- Even more parameters can be optimized but the exhaustive search becomes computationally hard due to exponentially exploding search space

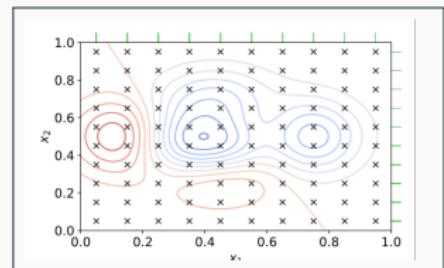
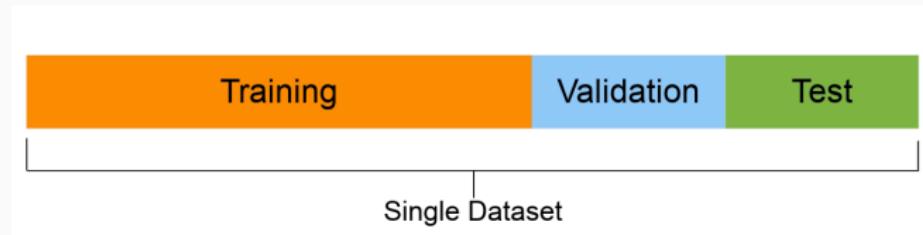


Figure by Alexander Elvers - Own work, CC BY-SA 4.0,

<https://commons.wikimedia.org/w/index.php?curid=842554>

## Model selection by using a validation set

- The need for the validation set comes from the need to avoid overfitting
- If we only use a simple training/test split and selected the hyperparameter values by repeated evaluation on the test set, the performance estimate will be optimistic
- A reliable performance estimate can only be obtained from the test set



## How large should the training set be in comparison of the validation set?

---

- The larger the training set, the better the generalization error will be (e.g. by PAC theory)
- The larger the validation set, the less variance there is in the test error estimate.
- When the dataset is small generally the training set is taken to be as large as possible, typically 90% or more of the total
- When the dataset is large, training set size is often taken as big as the computational resources allow

# Stratification

---

- Class distributions of the training and validation sets should be as similar to each another as possible, otherwise there will be extra unwanted variance
  - when the data contains classes with very low number of examples, random splitting might result in no examples in the class in the validation set
- Stratification is a process that tries to ensure similar class distributions across the different sets
- Simple stratification approach is to divide all classes separately into the training and validation sets and the merge the class-specific training sets into global training set and class-specific validation sets into a global validation set.

## Cross-validation

---

# The need of multiple data splits

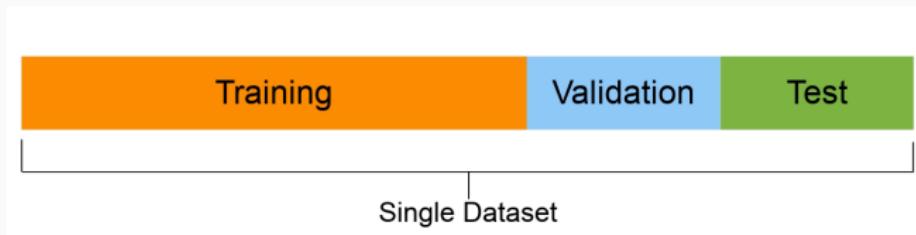
One split of data into training, validation and test sets may not be enough, due to randomness:

- The training and validation sets might be small and contain noise or outliers
- There might be some randomness in the training procedure (e.g. initialization)
- We need to fight the randomness by averaging the evaluation measure over multiple (training, validation) splits
  - The best hyperparameter values are chosen as those that have the best average performance over the  $n$  validation sets.



## Generating multiple data splits

- Let us first consider generating a number of training and validation set pairs, after first setting aside a separate test set
- Given a dataset  $S$ , we would like to generate  $n$  random splits into training and validation set
- Two general approaches:
  - Repeated random splitting
  - $n$ -fold cross-validation



## $n$ -Fold Cross-Validation

- The dataset  $S$  is split randomly into  $n$  equal-sized parts (or folds)
- We keep one of the  $n$  folds as the validation set (light blue in the Figure) and combine the remaining  $n - 1$  folds to form the training set for the split

Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 5	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5

- $n = 5$  or  $n = 10$  are typical numbers used in practice

## Leave-one-out cross-validation (LOO)

---

- Extreme case of cross-validation is leave-one-out (LOO) : given a dataset of  $m$  examples, only one example is left out as the validation set and training uses the  $m - 1$  examples.
- This gives an unbiased estimate of the average generalization error over samples of size  $m - 1$  (Mohri, et al. 2018, Theorem 5.4.)
- However, it is computationally demanding to compute if  $m$  is large

# Nested cross-validation

- $n$ -fold cross-validation gives us a well-founded way for model selection
- However, only using a single test set may result in unwanted variation
- Nested cross-validation solves this problem by using two cross-validation loops



# Nested cross-validation

The dataset is initially divided into  $n$  outer folds ( $n = 5$  in the figure)

- Outer loop uses 1 fold at a time as a test set, and the rest of the data is used in the inner fold
- Inner loop splits the remaining examples into  $k$  folds, 1 fold for validation,  $k - 1$  for training ( $k = 2$  in the figure)



The average performance over the  $n$  test sets is computed as the final performance estimate

# Summary

- Model selection concerns the trade-off between model complexity and empirical error on training data
- Regularization-based methods are based on continuous parametrization the complexity of the hypothesis classes
- Empirical model selection can be achieved by grid search on a validation dataset
- Various cross-validation schemes can be used to tackle the variance of the performance estimates

# CS-E4710 Machine Learning: Supervised Methods

## Lecture 5: Linear classification

---

Juho Rousu

October 4, 2022

Department of Computer Science  
Aalto University

# Course topics

- Part I: Theory
  - Introduction
  - Generalization error analysis & PAC learning
  - Rademacher Complexity & VC dimension
  - Model selection
- Part II: Algorithms and models
  - **Linear models: perceptron, logistic regression**
  - Support vector machines
  - Kernel methods
  - Boosting
  - Neural networks (MLPs)
- Part III: Additional topics
  - Feature learning, selection and sparsity
  - Multi-class classification
  - Preference learning, ranking

## Linear classification

---

# Linear classification

- Input space  $X \subset \mathbb{R}^d$ , each  $\mathbf{x} \in X$  is a  $d$ -dimensional real-valued vector, output space:  $\mathcal{Y} = \{-1, +1\}$
- Target function or concept  $f : X \mapsto \mathcal{Y}$  assigns a (true) label to each example
- Training sample  $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$ , with  $y_i = f(x_i)$  drawn from an unknown distribution  $D$
- Hypothesis class  
$$\mathcal{H} = \{\mathbf{x} \mapsto \text{sgn} \left( \sum_{j=1}^d w_j x_j + w_0 \right) \mid \mathbf{w} \in \mathbb{R}^d, w_0 \in \mathbb{R}\}$$
 consists of functions  $h(\mathbf{x}) = \text{sgn} \left( \sum_{j=1}^d w_j x_j + w_0 \right)$  that map each example in one of the two classes
- $\text{sgn}(a) = \begin{cases} +1, & a \geq 0 \\ -1 & a < 0 \end{cases}$  is the sign function

# Linear classifiers

## Linear classifiers

$$h(\mathbf{x}) = \text{sgn} \left( \sum_{j=1}^d w_j x_j + w_0 \right) = \text{sgn} (\mathbf{w}^T \mathbf{x} + w_0)$$

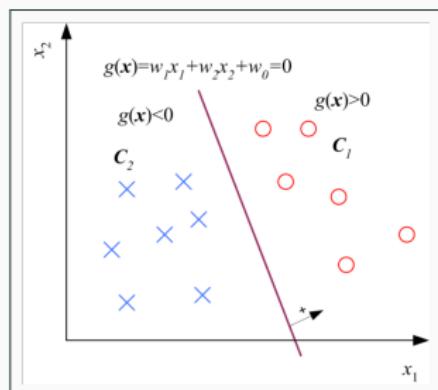
have several attractive properties

- They are fast to evaluate and takes small space to store ( $O(d)$  time and space)
- Easy to understand:  $|w_j|$  shows the importance of variable  $x_j$  and its sign tells if the effect is positive or negative
- Linear models have relatively low complexity (e.g.  $VCdim = d + 1$ ) so they can be reliably estimated from limited data

Good practise is to try a linear model before something more complicated

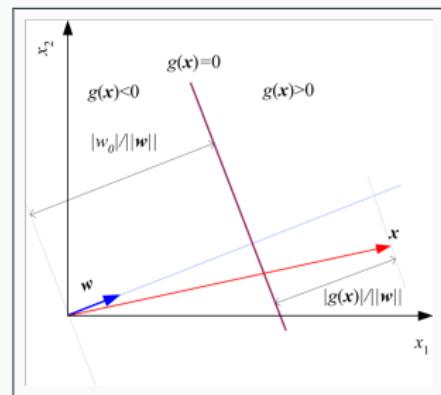
# The geometry of the linear classifier

- The points  $\{\mathbf{x} \in X | g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0\}$  define a hyperplane in  $\mathbb{R}^d$ , where  $d$  is the number of variables in  $\mathbf{x}$
- The hyperplane  $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0$  splits the input space into two half-spaces. The linear classifier predicts  $+1$  for points in the halfspace  $\{\mathbf{x} \in X | g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 \geq 0\}$  and  $-1$  for points in  $\{\mathbf{x} \in X | g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 < 0\}$



# The geometry of the linear classifier

- $\mathbf{w}$  is the **normal vector** of the hyperplane  $\mathbf{w}^T \mathbf{x} + w_0 = 0$
- The distance of the hyperplane from the origin is  $|w_0| / \|\mathbf{w}\|$
- If  $w_0 < 0$  the hyperplane lies in the direction of  $\mathbf{w}$  from origin, otherwise it lies in the direction of  $-\mathbf{w}$
- The distance of a point  $\mathbf{x}$  from the hyperplane is  $|g(\mathbf{x})| / \|\mathbf{w}\|$
- If  $g(\mathbf{x}) > 0$ ,  $\mathbf{x}$  lies in the halfspace that is in the direction of  $\mathbf{w}$  from the hyperplane, otherwise it lies in the direction of  $-\mathbf{w}$  from the hyperplane



## Learning linear classifiers

---

## Change of representation

- Consider learning the parameters of the linear discriminant  
$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$
- For presentation it is convenient to subsume term  $w_0$  into the weight vector

$$\mathbf{w} \Leftarrow \begin{bmatrix} \mathbf{w} \\ w_0 \end{bmatrix}$$

and augment all inputs with a constant 1:

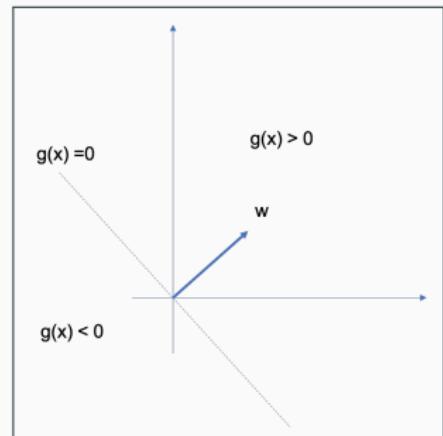
$$\mathbf{x} \Leftarrow \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

- The models have the same value for the discriminant:

$$\begin{bmatrix} \mathbf{w} \\ w_0 \end{bmatrix}^T \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \mathbf{w}^T \mathbf{x} + w_0$$

# Geometric interpretation

- Geometrically, the hyperplane defined by the discriminant goes now through origin
- The positive points have an **acute angle** with  $w$ :  $w^T x > 0$
- The negative points have an **obtuse angle** with  $w$ :  $w^T x \leq 0$



## Checking for prediction errors

- When the labels are  $\mathcal{Y} = \{-1, +1\}$  for a training example  $(\mathbf{x}, y)$  we have for  $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ ,

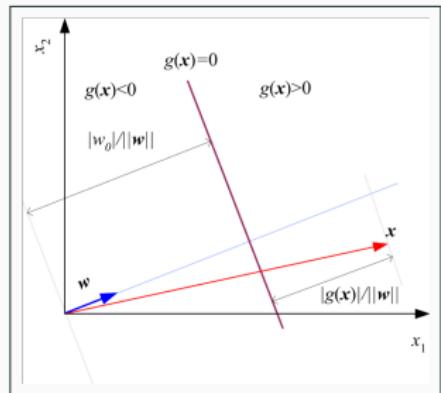
$$\text{sgn}(g(\mathbf{x})) = \begin{cases} y & \text{if } \mathbf{x} \text{ is correctly classified} \\ -y & \text{if } \mathbf{x} \text{ is incorrectly classified} \end{cases}$$

- Alternative we can just multiply with the correct label to check for misclassification:

$$yg(\mathbf{x}) = \begin{cases} \geq 0 & \text{if } \mathbf{x} \text{ is correctly classified} \\ < 0 & \text{if } \mathbf{x} \text{ is incorrectly classified} \end{cases}$$

# Margin

- The geometric margin of an example  $\mathbf{x}$  is given by  $\gamma(\mathbf{x}) = yg(\mathbf{x}) / \|\mathbf{w}\|$
- It takes into account both the distance  $|\mathbf{w}^T \mathbf{x}| / \|\mathbf{w}\|$  from the hyperplane, and whether  $\mathbf{x}$  is on the correct side of the hyperplane
- The unnormalized version of the margin is sometimes called the **functional margin**  $\gamma(\mathbf{x}) = yg(\mathbf{x})$
- Often the term **margin** is used for both variants, assuming the context makes clear which one is meant

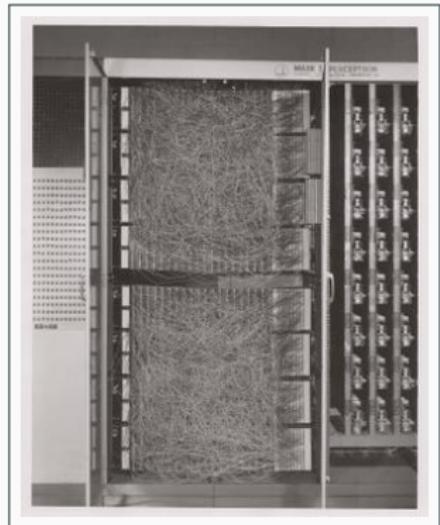


# Perceptron

---

# Perceptron

- Perceptron algorithm by Frank Rosenblatt (1956) is perhaps the first machine learning algorithm
- Its purpose was to learn a linear discriminant between two classes
- It was built in hardware and shown to be capable of performing rudimentary pattern recognition tasks
- New York Times in 1958: "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."  
(Source: Wikipedia)



Mark I perceptron ca. 1958 (Picture: Wikipedia)

# The perceptron algorithm

- The perceptron algorithm learns a hyperplane separating two classes

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

- It processes incrementally a set of training examples
  - At each step, it finds a training example  $\mathbf{x}_i$  that is incorrectly classified by the current model
  - It updates the model by adding the example to the current weight vector together with the label:  $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$
  - This process is continued until incorrectly predicted training examples are not found

# The perceptron algorithm

**Input:** Training set  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m, \mathbf{x} \in \mathbb{R}^d, y \in \{-1, +1\}$

Initialize  $\mathbf{w}^{(1)} \leftarrow (0, \dots, 0)$ ,  $t \leftarrow 1$ ,  $stop \leftarrow FALSE$

**repeat**

**if** exists  $i$ , s.t.  $y_i \mathbf{w}^{(t)} \mathbf{x}_i \leq 0$  **then**

$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$

**else**

$stop \leftarrow TRUE$

**end if**

$t \leftarrow t + 1$

**until**  $stop$

## Understanding the update rule

- Let us examine the update rule

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$$

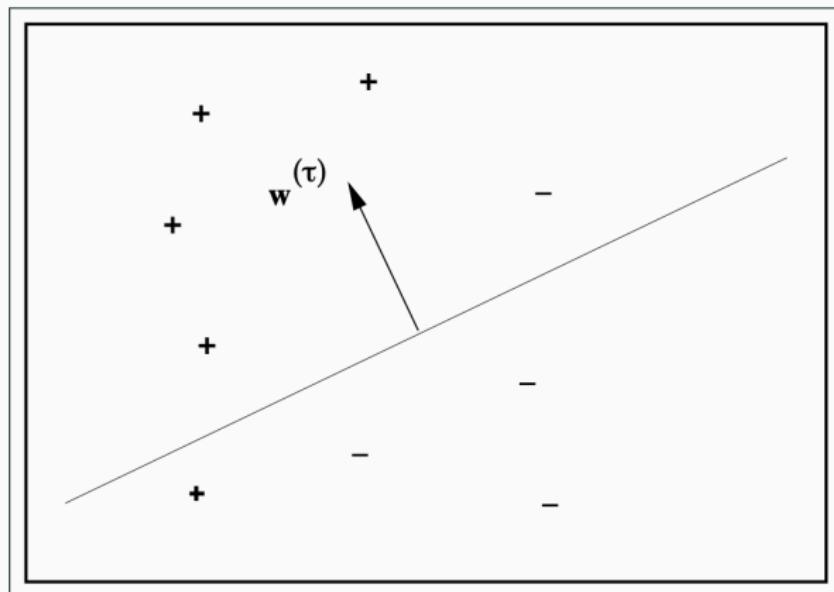
- We can see that the margin of the example  $(\mathbf{x}_i, y_i)$  increases after the update

$$\begin{aligned} y_i g^{(t+1)}(\mathbf{x}_i) &= y_i \mathbf{w}^{(t+1)T} \mathbf{x}_i = y_i (\mathbf{w}^{(t)} + y_i \mathbf{x}_i)^T \mathbf{x}_i \\ &= y_i \mathbf{w}^{(t)T} \mathbf{x}_i + y_i^2 \mathbf{x}_i^T \mathbf{x}_i = y_i g^{(t)}(\mathbf{x}_i) + \|\mathbf{x}_i\|^2 \\ &\geq y_i g^{(t)}(\mathbf{x}_i) \end{aligned}$$

- Note that this does not guarantee that  $y_i g^{(t+1)}(\mathbf{x}_i) > 0$  after the update, further updates may be required to achieve that

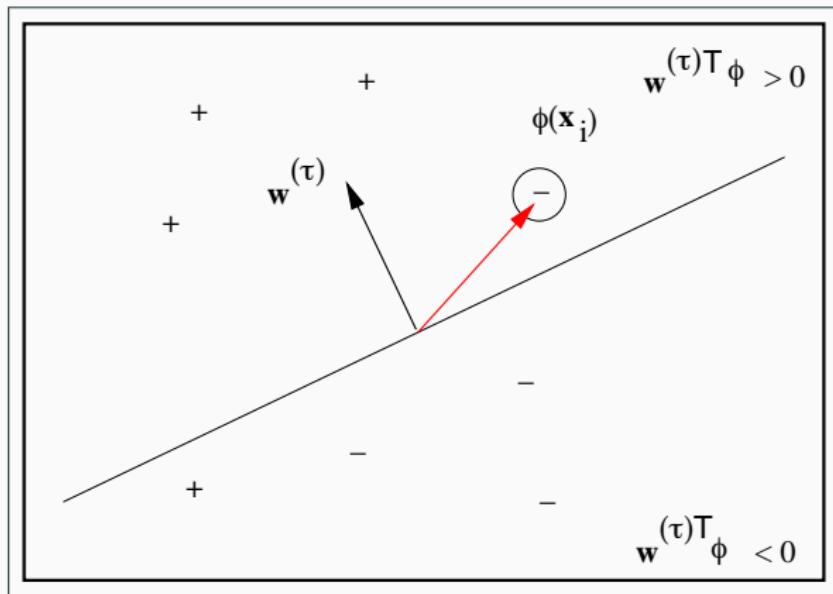
## Perceptron animation

- Assume  $\mathbf{w}^{(t)}$  has been found by running the algorithm for  $t$  steps
- We notice two misclassified examples



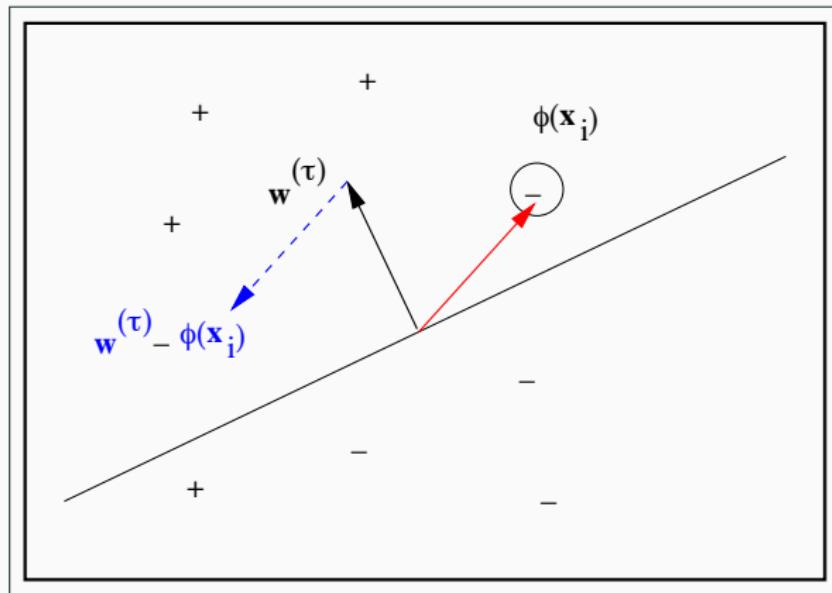
## Perceptron animation

- Select the misclassified example  $(\phi(\mathbf{x}_i), -1)$
- Note:  $\phi(\mathbf{x}_i)$  is here some transformation of  $\mathbf{x}_i$  e.g. with some basis functions but it could be identity  $\phi(\mathbf{x}) = \mathbf{x}$



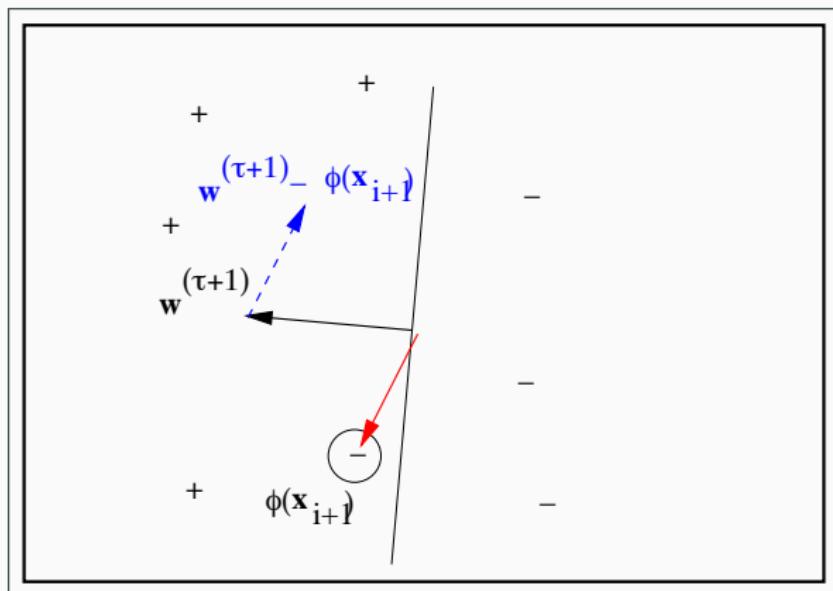
# Perceptron animation

- Update the weight vector:  $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \phi(\mathbf{x}_i)$



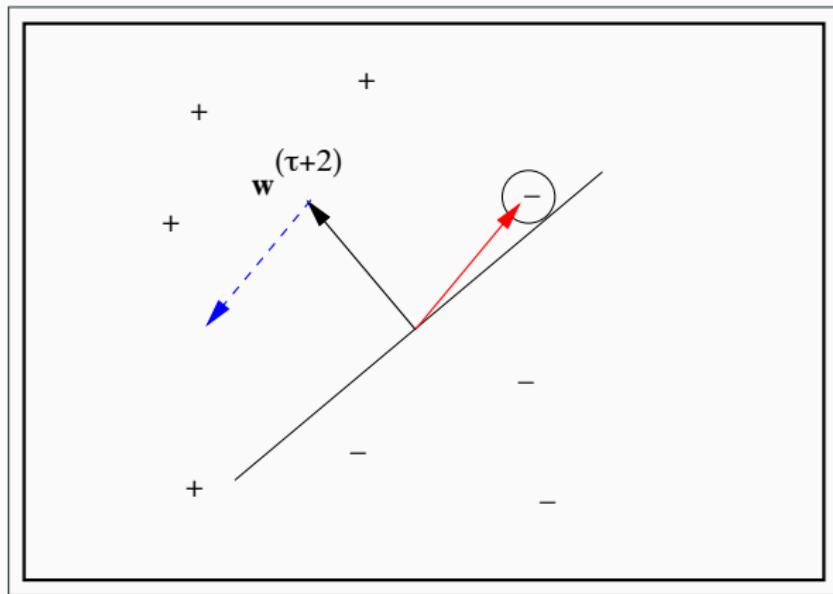
# Perceptron animation

- The update tilts the hyperplane to make the example "more correct", i.e. more negative
- We repeat the process by finding the next misclassified example  $\phi(\mathbf{x}_{i+1})$  and update:  $\mathbf{w}^{(t+2)} = \mathbf{w}^{(t+1)} + y_{i+1}\phi(\mathbf{x}_{i+1})$



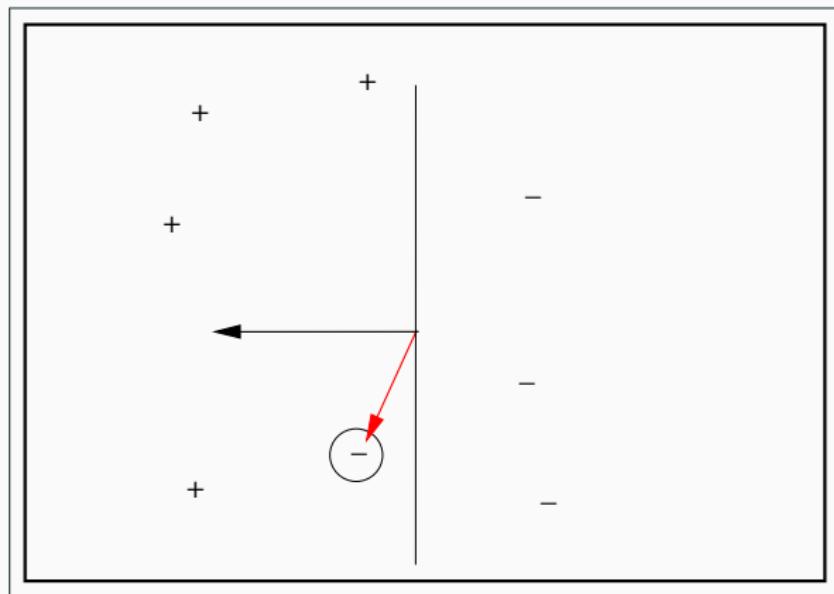
# Perceptron animation

- Next iteration



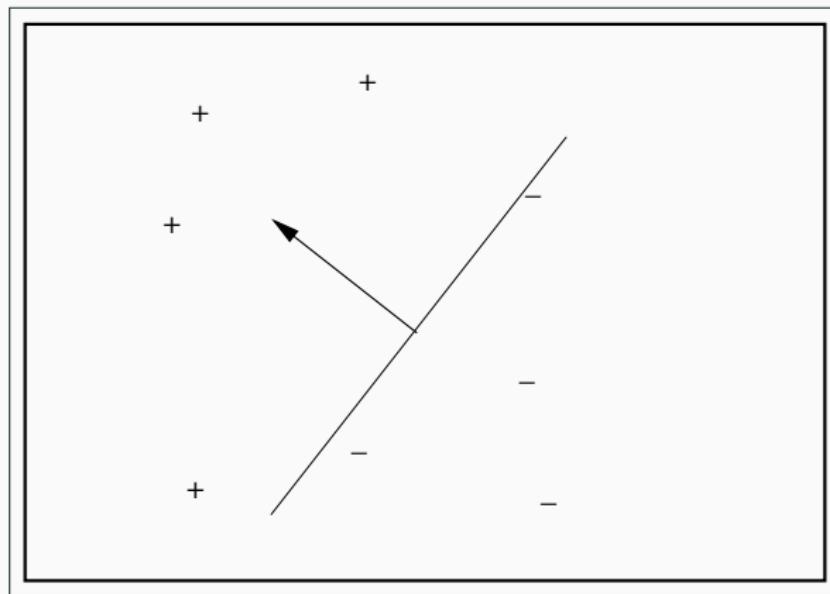
# Perceptron animation

- Next iteration



## Perceptron animation

- Finally we have found a hyperplane that correctly classify the training points
- We can stop the iteration and output the final weight vector



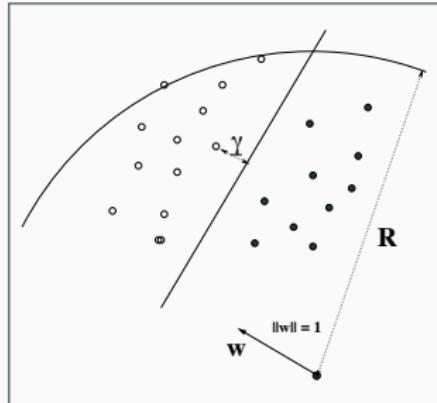
## Convergence of the perceptron algorithm

- The perceptron algorithm can be shown to eventually converge to a consistent hyperplane if the two classes are **linearly separable**, that is, if there exists a hyperplane that separates the two classes
- Theorem (Novikoff):
  - Let  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$  be a linearly separable training set.
  - Let  $R = \max_{\mathbf{x}_i \in S} \|\mathbf{x}_i\|$ .
  - Let there exist a vector  $\mathbf{w}_*$  that satisfies  $\|\mathbf{w}_*\| = 1$  and  $y_i \mathbf{w}_*^T \mathbf{x}_i + b_{opt} \geq \gamma$  for  $i = 1 \dots, m$ .
  - Then the perceptron algorithm will stop after at most  $t \leq (\frac{2R}{\gamma})^2$  iterations and output a weight vector  $\mathbf{w}^{(t)}$  for which  $y_i \mathbf{w}^{(t)} \mathbf{x}_i \geq 0$  for all  $i = 1 \dots, m$

# Convergence of the perceptron algorithm

The number of iterations in the bound  $t \leq (\frac{2R}{\gamma})^2$  depend on:

- $\gamma$ : The largest achievable geometric margin so that all training examples have at least that margin
- $R$ : The smallest radius of the  $d$ -dimensional ball that encloses the training data
- Intuitively: how large the margin in is relative to the distances of the training points



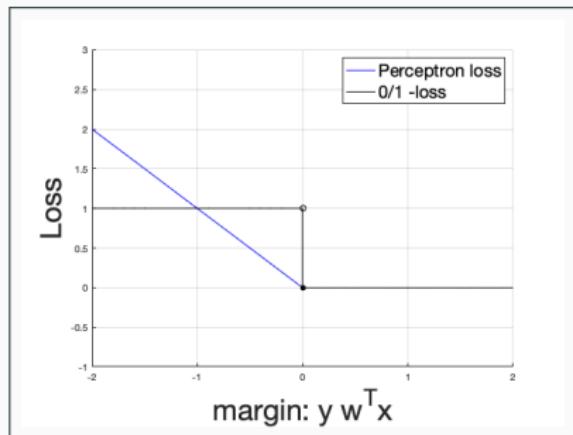
However, Perceptron algorithm does not stop on a non-separable training set, since there will always be a misclassified example that causes an update

# The loss function of the Perceptron algorithm

It can be shown that the Perceptron algorithm is using the following loss:

$$L_{\text{Perceptron}}(y, \mathbf{w}^T \mathbf{x}) = \max(0, -y\mathbf{w}^T \mathbf{x})$$

- $y\mathbf{w}^T \mathbf{x}$  is the margin
- if  $y\mathbf{w}^T \mathbf{x} < 0$ , a loss of  $-y\mathbf{w}^T \mathbf{x}$  is incurred, otherwise no loss is incurred

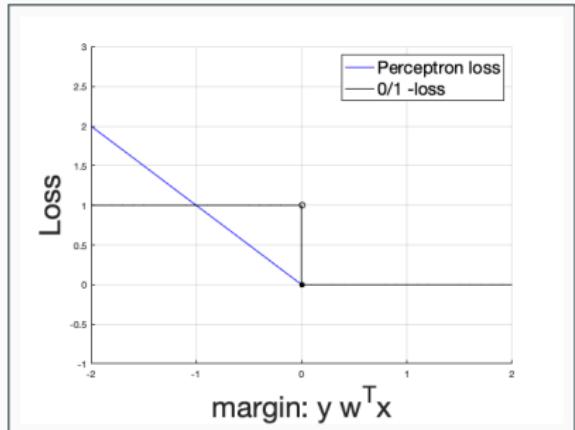


# Convexity of Perceptron loss

A function  $f : \mathbb{R}^n \mapsto \mathbb{R}$  is convex if for all  $x, y$ , and  $0 \leq \theta \leq 1$ , we have

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y).$$

- Geometrical interpretation:  
the graph of a convex  
function lies below the line  
segment from  $(x, f(x))$  to  
 $(y, f(y))$
- It is easy to see that  
Perceptron loss is convex but  
zero-one loss is not convex



## Convexity of Perceptron loss

- The convexity of the Perceptron loss has an important consequence: every local minimum is also the global minimum
- In principle we can minimize it with incremental updates that gradually decrease the loss
- In contrast, finding a hyperplane that minimizes the zero-one loss is computationally hard (NP-hard to minimize training error)
- However, we need better algorithms than the Perceptron, which terminate when we are close to the optimum

## Logistic regression

---

# Logistic regression

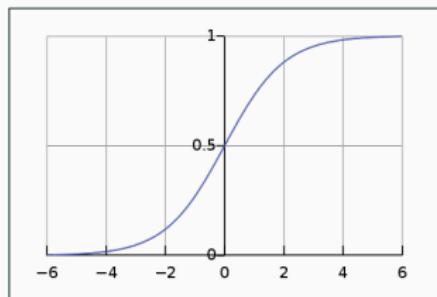
Logistic regression is a classification technique (despite the name)

- it gets its name from the logistic function

$$\phi_{logistic}(z) = \frac{1}{1 + \exp(-z)} = \frac{\exp(z)}{1 + \exp(z)}$$

that maps a real valued input  $z$  onto the interval  $0 < \phi_{logistic}(z) < 1$

- The function is an example of **sigmoid** ("S" shaped) functions



## Logistic function: a probabilistic interpretation

- The logistic function  $\phi_{logistic}(z)$  is the inverse of **logit function**
- The logit function is the logarithm of **odds ratio** of probability  $p$  of an event happening vs. the probability of the event not happening,  $1 - p$ ;

$$z = \text{logit}(p) = \log \frac{p}{1-p} = \log p - \log(1-p)$$

- Thus the logistic function

$$\phi_{logistic}(z) = \text{logit}^{-1}(z) = \frac{1}{1 + \exp(-z)}$$

answer the question "what is the probability  $p$  that gives the log odds ratio of  $z$ "

## Logistic regression

- Logistic regression model assumes a underlying conditional probability:

$$Pr(y|\mathbf{x}) = \frac{\exp(+\frac{1}{2}y\mathbf{w}^T\mathbf{x})}{\exp(+\frac{1}{2}y\mathbf{w}^T\mathbf{x}) + \exp(-\frac{1}{2}y\mathbf{w}^T\mathbf{x})}$$

where the denominator normalizes the right-hand side to be between zero and one.

- Dividing the numerator and denominator by  $\exp(+\frac{1}{2}y\mathbf{w}^T\mathbf{x})$  reveals the logistic function

$$Pr(y|\mathbf{x}) = \phi_{logistic}(y\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp(-y\mathbf{w}^T\mathbf{x})}$$

- The margin  $z = y\mathbf{w}^T\mathbf{x}$  is thus interpreted as the log odds ratio of label  $y$  vs. label  $-y$  given input  $\mathbf{x}$ :

$$y\mathbf{w}^T\mathbf{x} = \log \frac{Pr(y|\mathbf{x})}{Pr(-y|\mathbf{x})}$$

## Logistic loss

- Consider the maximization of the likelihood of the observed input-output in the training data:

$$\mathbf{w}^* = \operatorname{argmax}_{\mathbf{w}} \prod_{i=1}^m P(y_i | \mathbf{x}_i) = \operatorname{argmax}_{\mathbf{w}} \prod_{i=1}^m \frac{1}{1 + \exp(-y_i \mathbf{w}^T \mathbf{x})}$$

- Since the logarithm is monotonically increasing function, we can take the logarithm to obtain an equivalent objective:

$$\sum_{i=1}^m \log Pr(y_i | \mathbf{x}_i) = - \sum_{i=1}^m \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))$$

- The right-hand side is the **logistic loss**:

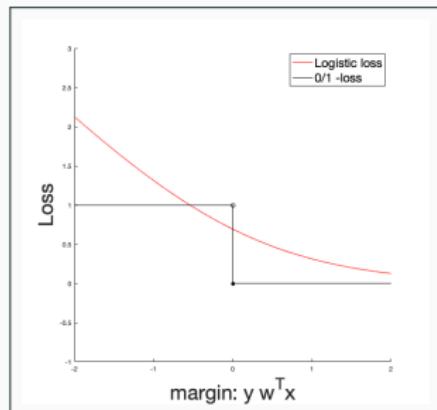
$$L_{\text{logistic}}(y, \mathbf{w}^T \mathbf{x}) = \log(1 + \exp(-y \mathbf{w}^T \mathbf{x}))$$

- Minimizing the logistic loss correspond maximizing the likelihood of the training data

# Geometric interpretation of Logistic loss

$$L_{logistic}(y, \mathbf{w}^T \mathbf{x}) = \log(1 + \exp(-y\mathbf{w}^T \mathbf{x}))$$

- Logistic loss is convex and differentiable
- It is a monotonically decreasing function of the margin  $y\mathbf{w}^T \mathbf{x}$
- The loss changes fast when the margin is highly negative  $\Rightarrow$  penalization of examples far in the incorrect halfspace
- It changes slowly for highly positive margins  $\Rightarrow$  does not give extra bonus for being very far in the correct halfspace



## Logistic regression optimization problem

- To train a logistic regression model, we need to find the  $\mathbf{w}$  that minimizes the average logistic loss  $J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m L_{logistic}(y_i, \mathbf{w}^T \mathbf{x}_i)$  over the training set:

$$\min \quad J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))$$

w.r.t parameters  $\mathbf{w} \in \mathbb{R}^d$

- The function to be minimized is continuous and differentiable
- However, it is a non-linear function so it is not easy to find the optimum directly (e.g. unlike in linear regression)
- We will use **stochastic gradient descent** to incrementally step towards the direction where the objective decreases fastest, the **negative gradient**

# Gradient

---

- The gradient is the vector of partial derivatives of the objective function  $J(\mathbf{w})$  with respect to all parameters  $w_j$

$$\nabla J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m \nabla J_i(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m \left[ \frac{\partial}{\partial w_1} J_i(\mathbf{w}), \dots, \frac{\partial}{\partial w_d} J_i(\mathbf{w}) \right]^T$$

- Compute the gradient by using the regular rules for differentiation.  
For the logistic loss we have

$$\begin{aligned}\frac{\partial}{\partial w_j} J_i(\mathbf{w}) &= \frac{\partial}{\partial w_j} \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)) = \frac{\exp(-y_i \mathbf{w}^T \mathbf{x}_i)}{1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)} \cdot (-y_i x_{ij}) \\ &= -\frac{1}{1 + \exp(y_i \mathbf{w}^T \mathbf{x}_i)} y_i x_{ij} = -\phi_{logistic}(-y_i \mathbf{w}^T \mathbf{x}_i) y_i x_{ij}\end{aligned}$$

## Stochastic gradient descent

- We collect the partial derivatives with respect to a single training example into a vector:

$$\nabla J_i(\mathbf{w}) = \begin{bmatrix} -(\phi_{logistic}(-y_i \mathbf{w}^T \mathbf{x}_i) y_i) \cdot x_{i1} \\ \vdots \\ -(\phi_{logistic}(-y_i \mathbf{w}^T \mathbf{x}_i) y_i) \cdot x_{ij} \\ \vdots \\ -(\phi_{logistic}(-y_i \mathbf{w}^T \mathbf{x}_i) y_i) \cdot x_{id} \end{bmatrix} = -\phi_{logistic}(-y_i \mathbf{w}^T \mathbf{x}_i) y_i \cdot \mathbf{x}_i$$

- The vector  $-\nabla J_i(\mathbf{w})$  gives the update direction that fastest decreases the loss on training example  $(\mathbf{x}_i, y_i)$

# Stochastic gradient descent

- Evaluating the full gradient

$$\nabla J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m \nabla J_i(\mathbf{w}) = -\frac{1}{m} \sum_{i=1}^m \phi_{logistic}(-y_i \mathbf{w}^T \mathbf{x}_i) y_i \cdot \mathbf{x}_i$$

is costly since we need to process all training examples

- Stochastic gradient descent instead uses a series of smaller updates that depend on single randomly drawn training example  $(\mathbf{x}_i, y_i)$  at a time
- The update direction is taken as  $-\nabla J_i(\mathbf{w})$
- Its expectation is the full negative gradient:

$$-\mathbb{E}_{i=1,\dots,m} [\nabla J_i(\mathbf{w})] = -\nabla J(\mathbf{w})$$

- Thus on average, the updates match that of using the full gradient

# Stochastic gradient descent algorithm

---

Initialize  $\mathbf{w} = 0$

**repeat**

    Draw a training example  $(x_i, y_i)$  uniformly at random

    Compute the update direction corresponding to the training example:

$$\Delta \mathbf{w} = -\nabla J_i(\mathbf{w})$$

    Determine a stepsize  $\eta$

$$\text{Update } \mathbf{w} = \mathbf{w} - \eta \nabla J_i(\mathbf{w})$$

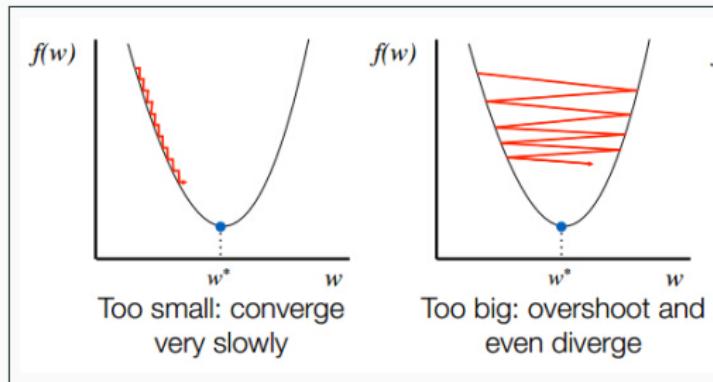
**until** stopping criterion satisfied

Output  $\mathbf{w}$

# Stepsize selection

Consider the SGD update:  $\mathbf{w} = \mathbf{w} - \eta \nabla J_i(\mathbf{w})$

- The stepsize parameter  $\eta$ , also called the **learning rate** is a critical one for convergence to the optimum value
- If one uses small constant stepsize, the initial convergence may be unnecessarily slow
- If too large stepsize may cause the method to continually overshoot the optimum.



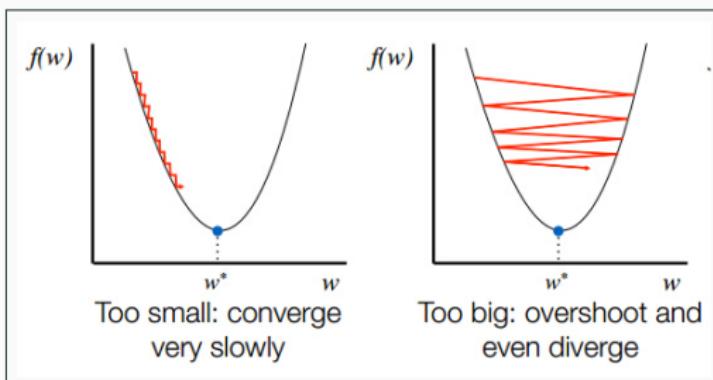
# Diminishing stepsize

- Initially larger but diminishing stepsize is one option:

$$\eta^{(t)} = \frac{1}{\alpha t}$$

for some  $\alpha > 0$ , where  $t$  is the iteration counter

- Caution: In practice, finding a good value for parameter  $\alpha$  requires experimenting with several values



## Stopping criterion

When should we stop the algorithm? Some possible choices:

1. Set a maximum number of iterations, after which the algorithm terminates
  - This needs to be separately calibrated for each dataset to avoid premature termination
2. Gradient of the objective: If we are at a optimum point  $\mathbf{w}^*$  of  $J(\mathbf{w})$ , the gradient vanishes  $\nabla J(\mathbf{w}^*) = 0$ , so we can stop  $\|J(\mathbf{w})\| < \gamma$  where  $\gamma$  is some user-defined parameter
3. It is usually sufficient to train until the **zero-one error** on training data does not change anymore
  - This usually happens before the logistic loss converges

# Summary

- Linear classification model are and important class of machine learning models, they are used as standalone models and appear as building blocks of more complicated, non-liner models
- Perceptron is a simple algorithm to train linear classifiers on linearly separable data
- Logistic regression is a classification method that can be interpreted as maximizing odds ratios of conditional class probabilities
- Stochastic gradient descent is an efficient optimization method for large data that is nowadays very widely used

# CS-E4710 Machine Learning: Supervised Methods

## Lecture 6: Support vector machines

---

Juho Rousu

11. October, 2022

Department of Computer Science  
Aalto University

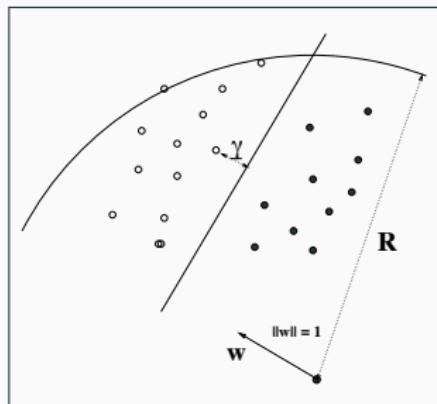
## Finding optimal separating hyperplanes

---

## Recall: Perceptron algorithm on linearly separable data

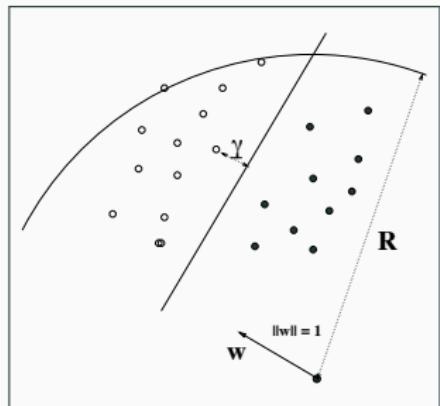
Recall the upper bound  $(\frac{2R}{\gamma})^2$  of iterations of perceptron algorithm on linearly separable data

- $\gamma$ : The largest achievable geometric margin in the training set,  $\frac{y_i \mathbf{w}^T \mathbf{x}_i}{\|\mathbf{w}\|} \geq \gamma$  for all  $i = 1 \dots, m$
- $R = \max_i \|\mathbf{x}_i\|$ : The smallest radius of the  $d$ -dimensional ball that encloses the training data



## Finding optimal separating hyperplanes

- The hyperplane output by the perceptron algorithm is guaranteed to be consistent
- All training data are on the correct side of the hyperplane
- However, typically there are several hyperplanes that are consistent
- Which one is the best?



## Maximum margin hyperplane

One good solution is to choose the hyperplane  $\mathbf{w}^T \mathbf{x} = 0$  that lies furthest away from the training data (maximizing the minimum geometric margin of the training examples):

*Maximize*  $\gamma$

*w.r.t.* variables  $\mathbf{w} \in \mathbb{R}^d$

*Subject to*  $\frac{y_i \mathbf{w}^T \mathbf{x}_i}{\|\mathbf{w}\|} \geq \gamma$ , for all  $i = 1, \dots, m$ ,

The maximum margin hyperplane has good properties:

- Robustness: small change in the training data will not change the classifications too much
- Theoretically a large margin is tied to a low generalization error
- It can be found efficiently through incremental optimization

Support vector machines (SVM) are based on this principle

# How to Maximize the Margin?

- However, the optimization problem

Maximize  $\gamma$

w.r.t. variables  $\mathbf{w} \in \mathbb{R}^d$

Subject to  $\frac{y_i \mathbf{w}^T \mathbf{x}_i}{\|\mathbf{w}\|} \geq \gamma$ , for all  $i = 1, \dots, m$

does not give us a **unique** optimal weight vector  $\mathbf{w}^*$

- This is because if  $\mathbf{w}^*$  is a solution, then so is any vector  $c\mathbf{w}^*$ ,  $c > 0$  since

$$\frac{y_i(c\mathbf{w})^T \mathbf{x}_i}{\|c\mathbf{w}\|} = \frac{cy_i \mathbf{w}^T \mathbf{x}_i}{\sqrt{c^2 \mathbf{w}^T \mathbf{w}}} = \frac{cy_i \mathbf{w}^T \mathbf{x}_i}{c \|\mathbf{w}\|} = \frac{y_i \mathbf{w}^T \mathbf{x}_i}{\|\mathbf{w}\|}$$

- We can make the functional margin  $y_i \mathbf{w}^T \mathbf{x}_i$  arbitrarily high just by scaling the norm of  $\mathbf{w}$

## How to Maximize the Margin?

- We could add a constraint  $\|\mathbf{w}\| = 1$  to the optimization problem to get an unique answer.
- However, optimization would become more difficult to solve
- Instead, let us multiply the constraint on the geometric margin

$$\frac{y_i \mathbf{w}^T \mathbf{x}_i}{\|\mathbf{w}\|} \geq \gamma$$

by  $\|\mathbf{w}\|$  to obtain a an equivalent constraint on the functional margin

$$y_i \mathbf{w}^T \mathbf{x}_i \geq \gamma \|\mathbf{w}\|$$

- Now fix the functional margin to 1:  $\gamma \|\mathbf{w}\| = 1$  which gives  $\gamma = \frac{1}{\|\mathbf{w}\|}$
- To maximize  $\gamma$ , we should minimize  $\|\mathbf{w}\|$  with the constraint of having functional margin of at least 1

## Support vector machine (SVM)

The so called hard margin support-vector machine (SVM, Cortes & Vapnik, 1995) solves the margin maximization as follows:

$$\text{Minimize} \frac{1}{2} \|\mathbf{w}\|^2$$

w.r.t. variables  $\mathbf{w} \in \mathbb{R}^d$

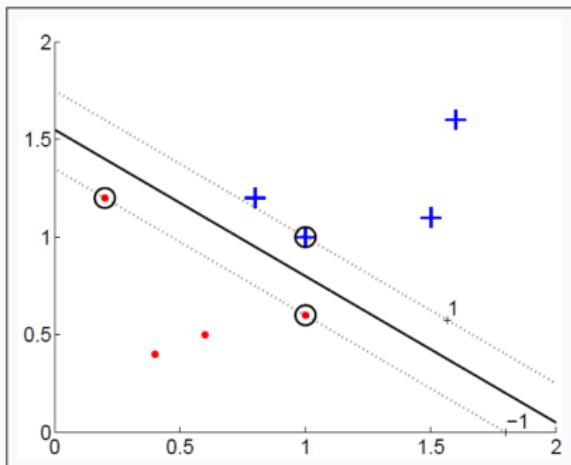
Subject to  $y_i \mathbf{w}^T \mathbf{x}_i \geq 1$ , for all  $i = 1, \dots, m$

- We are minimizing the half of the squared norm of the weight vector, which gives the same answer as minimizing the norm, but easier to optimize
- This is equivalent of finding the maximal geometric margin over the same data

## Geometrical interpretation

The maximum margin hyperplane separates the positive and negative examples with a minimum functional margin of 1

- The points that have exactly margin  $y\mathbf{w}^T \mathbf{x} = 1$  are called the support vectors
- The position of the hyperplane only depends on the support vectors, its position does not change if points with  $y\mathbf{w}^T \mathbf{x} > 0$  are added or removed



## Generalization capability of the maximum margin hyperplane

- The maximum margin hyperplane has significant theoretical backup
- Consider the hypothesis class

$$\mathcal{H} = \{h(\mathbf{x}) = \text{sgn} (\mathbf{w}^T \mathbf{x}) \mid \min_{i=1}^m y_i \mathbf{w}^T \mathbf{x}_i = 1, \|\mathbf{w}\| \leq B, \|\mathbf{x}_i\| \leq R\}$$

- The VC dimension satisfies  $VCdim(\mathcal{H}) \leq B^2 R^2$
- Rademacher complexity satisfies:  $\mathcal{R}(H) \leq \frac{RB}{\sqrt{m}}$
- Thus a small norm ( $\leq B$ ) translates to low complexity of the hypothesis class
- A better generalization error is thus likely if we can find a consistent hyperplane with a small norm (or, equivalently, a large margin)

## Non-separable data

The so called hard margin support-vector machine assumes linearly separable data

$$\begin{aligned} & \text{Minimize} \quad \frac{1}{2} \|\mathbf{w}\|^2 \\ & \text{w.r.t. variables } \mathbf{w} \in \mathbb{R}^d \end{aligned}$$

$$\text{Subject to } y_i \mathbf{w}^T \mathbf{x}_i \geq 1, \text{ for all } i = 1, \dots, m$$

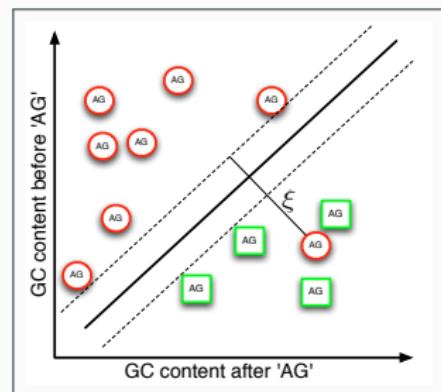
- In the non-separable case, for any hyperplane, there will be an example with a negative margin  $y_i \mathbf{w}^T \mathbf{x}_i < 0$  which violates the constraint  $y_i \mathbf{w}^T \mathbf{x}_i \geq 1$
- Our optimization problem has no feasible solution
- We need to extend our model to allow misclassified training points

# Non-separable data

- To allow non-separable data, we allow the functional margin of some data points to be smaller than 1 by a slack variable  $\xi_i \geq 0$
- The relaxed margin constraint will be expressed as

$$y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i, \xi_i \geq 0$$

- $\xi_i = 0$  corresponds to having large enough margin  $> 1$
- $\xi_i > 1$  corresponds to negative margin, misclassified point
- The set of support vectors includes all  $\mathbf{x}_i$  that have non-zero slack  $\xi_i$  (functional margin  $\leq 1$ )



# Soft-Margin SVM (Cortes & Vapnik, 1995)

The soft-margin SVM allows non-separable data by using the relaxed margin constraints

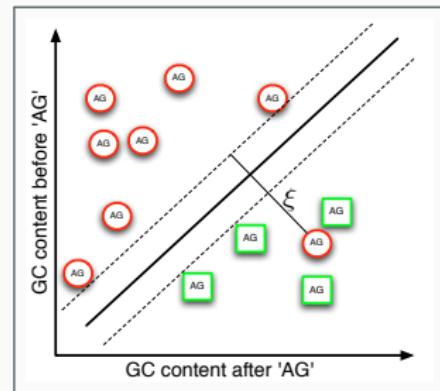
$$\text{Minimize} \frac{1}{2} \|\mathbf{w}\|^2 + \frac{C}{m} \sum_{i=1}^m \xi_i$$

w.r.t variables  $\mathbf{w}, \xi$

$$\text{Subject to } y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i$$

for all  $i = 1, \dots, m$ .

$$\xi_i \geq 0, \text{ for all } i = 1, \dots, m.$$



- The sum (or average) of slack variables appear as a penalty in the objective
- The coefficient  $C > 0$  controls the balance between model complexity (low  $C$ ) and empirical error (high  $C$ )

## The loss function in SVM

- We can interpret the soft-margin SVM in terms of minimization of a loss function
- Observe the relaxed margin constraint:

$$y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i, \xi_i \geq 0$$

- By rearranging, the same can be expressed as

$$\xi_i \geq 1 - y_i \mathbf{w}^T \mathbf{x}_i, \xi_i \geq 0$$

and further

$$\xi_i \geq \max(1 - y_i \mathbf{w}^T \mathbf{x}_i, 0)$$

- The right-hand side is so called Hinge loss:

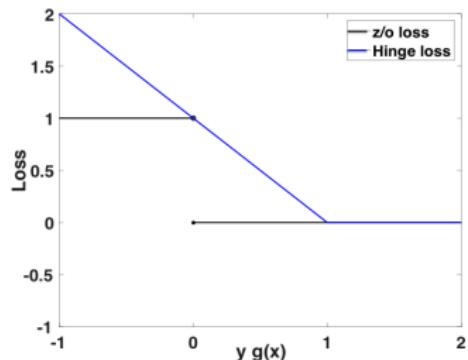
$$L_{\text{Hinge}}(y, \mathbf{w}^T \mathbf{x}) = \max(1 - y \mathbf{w}^T \mathbf{x}, 0)$$

## Loss functions: Hinge loss

Hinge loss can be written for  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  as

$$L_{\text{Hinge}}(y, f(\mathbf{x})) = \max(1 - yf(\mathbf{x}), 0)$$

- Hinge loss is a convex upper bound of zero-one loss
- Hinge loss is zero if margin  $y_i f(\mathbf{x}) \geq 1$
- For a misclassified example, margin is negative and Hinge loss is  $\mathcal{L}_{\text{Hinge}}(f(\mathbf{x}), y_i) > 1$
- The loss grows linearly in the margin violation  $1 - yf(\mathbf{x})$ , for margins  $< 1$



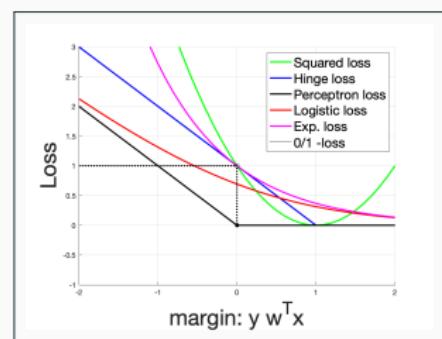
## Interlude: Convex loss functions for classification

We have so far seen three different convex loss functions for classification:

- Perceptron loss
- Hinge loss
- Logistic loss

In addition there are multiple other convex loss functions:

- Squared loss - used for regression, not optimal for classification
- Exponential loss - used in boosting

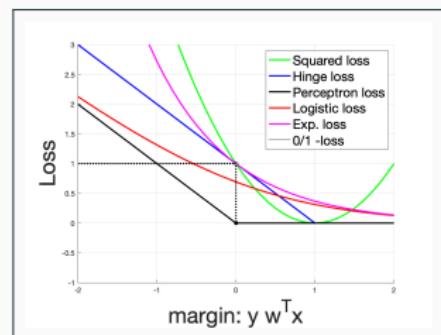


Because of convexity, all of the functions support fast optimization through gradient based approaches, unlike the zero-one loss

## Interlude: Convex loss functions for classification

- Three of the loss functions in the figure are not only convex, but also upper-bounds of zero-one loss: Exponential loss, Hinge loss, Squared loss

- Logistic loss can be scaled by a constant to make it an upper bound of zero-one loss
- Perceptron loss is not an upper bound of zero-one loss



- The benefit of being an upper bound of zero-one loss: if we minimize the upper bound, the zero-one loss will also usually be small
- Because of the convexity, minimization of the upper bound can be done fast, unlike minimization of zero-one loss

# Optimization

---

## Quadratic programming

The soft-margin SVM corresponds to a Quadratic program (QP)

$$\text{Minimize} \frac{1}{2} \|\mathbf{w}\|^2 + \frac{C}{m} \sum_{i=1}^m \xi_i$$

w.r.t variables  $\mathbf{w}, \xi$

Subject to  $y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i$

for all  $i = 1, \dots, m.$

$\xi_i \geq 0$ , for all  $i = 1, \dots, m.$

- A QP is a convex optimization problem (with a unique optimum)
- The QP objective is a quadratic function of the variables
- The QP constraints are linear functions of the variables
- When data is small, QP solvers in optimization libraries can be used to solve the soft-margin SVM problem

## Soft-margin SVM as a regularised learning problem

- We can rewrite the soft-margin SVM problem

$$\begin{aligned} & \text{Minimize} && \frac{1}{2} \|\mathbf{w}\|^2 + \frac{C}{m} \sum_{i=1}^m \xi_i \\ & \text{Subject to} && \xi_i \geq \max(1 - y_i \mathbf{w}^T \mathbf{x}_i, 0) \\ & && \text{for all } i = 1, \dots, N. \\ & && \xi_i \geq 0 \end{aligned}$$

equivalently in terms of Hinge loss as

$$\min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^m \mathcal{L}_{\text{Hinge}}(\mathbf{w}^T \mathbf{x}_i, y_i) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

- This is a so called **regularized learning problem**
  - First term minimizes a loss function on training data
  - Second term, called the **regularizer**, controls the complexity of the model
  - The parameter  $\lambda = \frac{1}{C}$  controls the balance between the two terms

# Optimization on big data

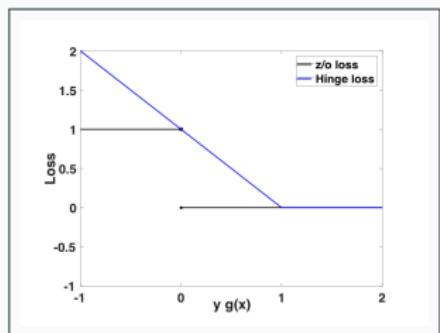
On big data, a stochastic gradient descent procedure is a good option

Rewrite the regularized learning problem as an average:

$$J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m J_i(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m (\mathcal{L}_{\text{Hinge}}(\mathbf{w}^T \mathbf{x}_i, y_i) + \frac{\lambda}{2} \|\mathbf{w}\|^2)$$

where  $J_i(\mathbf{w}) = \mathcal{L}_{\text{Hinge}}(\mathbf{w}^T \mathbf{x}_i, y_i) + \frac{\lambda}{2} \|\mathbf{w}\|^2$

However, Hinge loss is not differentiable at 1 (because of the 'Hinge' at 1), so cannot simply compute the gradient  $\nabla J_i(\mathbf{w})$



## Gradients of the Hinge loss

- We can differentiate the linear pieces of the loss separately

$$L_{\text{Hinge}}(\mathbf{w}^T \mathbf{x}_i, y_i) = \begin{cases} 1 - y_i \mathbf{w}^T \mathbf{x}_i, & \text{if } y_i \mathbf{w}^T \mathbf{x}_i < 1 \\ 0, & \text{if } y_i \mathbf{w}^T \mathbf{x}_i \geq 1 \end{cases}$$

- We get

$$\nabla L_{\text{Hinge}}(\mathbf{w}^T \mathbf{x}_i, y_i) = \begin{cases} -y_i \mathbf{x}_i & \text{if } y_i \mathbf{w}^T \mathbf{x}_i < 1 \\ \mathbf{0} & \text{if } y_i \mathbf{w}^T \mathbf{x}_i \geq 1 \end{cases}$$

- At  $\mathbf{w}^T \mathbf{x}_i = 1$ , the function is not differentiable but we can choose  $\mathbf{0}$  as the value, since the Hinge loss is zero so no update is needed to decrease loss
- (Formally  $\mathbf{0}$  is one of the subgradients of the Hinge loss at 1, so can be justified from optimization theory)

## Stochastic gradient descent algorithm for SVM

To find the update direction we express  $J_i(\mathbf{w})$  as a piecewise differentiable function

$$J_i(\mathbf{w}) = L_{Hinge}(\mathbf{w}^T \mathbf{x}_i, y_i) + \frac{\lambda}{2} \|\mathbf{w}\|^2 = \begin{cases} 1 - y_i \mathbf{w}^T \mathbf{x}_i + \frac{\lambda}{2} \|\mathbf{w}\|^2, & \text{if } y_i \mathbf{w}^T \mathbf{x}_i < 1 \\ 0 + \frac{\lambda}{2} \|\mathbf{w}\|^2 & \text{if } y_i \mathbf{w}^T \mathbf{x}_i \geq 1 \end{cases}$$

Computing the derivatives piecewise gives the gradient:

$$\nabla J_i(\mathbf{w}) = \begin{cases} -y_i \mathbf{x}_i + \lambda \mathbf{w} & \text{if } y_i \mathbf{w}^T \mathbf{x}_i < 1 \\ \mathbf{0} + \lambda \mathbf{w} & \text{if } y_i \mathbf{w}^T \mathbf{x}_i \geq 1 \end{cases}$$

Update direction is the negative gradient  $-\nabla J_i(\mathbf{w})$

# Stochastic gradient descent algorithm for soft-margin SVM

Initialize  $\mathbf{w} = 0$

**repeat**

    Draw a training example  $(x_i, y_i)$  uniformly at random

    Compute the update direction corresponding to the training example:

$$\nabla J_i(\mathbf{w}) = \begin{cases} -y_i \mathbf{x}_i + \lambda \mathbf{w} & \text{if } y_i \mathbf{w}^T \mathbf{x}_i < 1 \\ \lambda \mathbf{w} & \text{if } y_i \mathbf{w}^T \mathbf{x}_i \geq 1 \end{cases}$$

    Determine a stepsize  $\eta$

    Update  $\mathbf{w} = \mathbf{w} - \eta \nabla J_i(\mathbf{w})$

**until** stopping criterion satisfied

Output  $\mathbf{w}$

- For the stepsize, diminishing stepsize of  $\eta = 1/\lambda t$ , has been suggested in the literature
- As the stopping criterion, one can use, e.g. the relative improvement of the objective between two successive iterations – stop iterations once it goes below given threshold

## Interpreting the update

$$\mathbf{w} = \mathbf{w} - \eta \left( \lambda \mathbf{w} + \begin{cases} -y_i \mathbf{x}_i & \text{if } y_i \mathbf{w}^T \mathbf{x}_i < 1 \\ \mathbf{0} & \text{otherwise} \end{cases} \right)$$

- Each update shrinks the weight vector by  $\eta\lambda \implies$  increases the geometric margin and adds regularization
- If the example has positive Hinge loss (functional margin  $< 1$ ), we add  $\eta y_i \mathbf{x}_i$  to the weight vector
- This has an effect of decreasing the Hinge loss on that example, similarly to the perceptron update

## Interpreting the update

$$\mathbf{w} = \mathbf{w} - \eta \left( \lambda \mathbf{w} + \begin{cases} -y_i \mathbf{x}_i & \text{if } y_i \mathbf{w}^T \mathbf{x}_i < 1 \\ \mathbf{0} & \text{otherwise} \end{cases} \right)$$

- Compare to the perceptron update:

$$\mathbf{w} = \mathbf{w} + \begin{cases} y_i \mathbf{x}_i & \text{if } y_i \neq \text{sgn}(\mathbf{w}^T \mathbf{x}_i) \\ \mathbf{0} & \text{otherwise} \end{cases}$$

- Both share the idea of adding to  $\mathbf{w}$  the training example multiplied by the label  $y_i \mathbf{x}_i$ , SVM does this to all examples that have too small margin, not only misclassified ones
- SVM shrinks the weight vector by fraction of  $\lambda$  on all examples, to regularize

## Interpreting the update

- Consider the evolution of the weight vector  $\mathbf{w}^{(t)}$  by the stochastic gradient optimization
- Assume  $\lambda = 0$  and that  $(\mathbf{x}^{(t)}, y^{(t)})$  is the  $t$ 'th training example drawn by the algorithm that has positive Hinge loss, and  $\eta^{(t)}$  is the learning rate
- Then we have

$$\mathbf{w}^{(1)} = \eta^{(1)} y^{(1)} \mathbf{x}^{(1)}$$

$$\mathbf{w}^{(2)} = \eta^{(1)} y^{(1)} \mathbf{x}^{(1)} + \eta^{(2)} y^{(2)} \mathbf{x}^{(2)}$$

$$\mathbf{w}^{(3)} = \eta^{(1)} y^{(1)} \mathbf{x}^{(1)} + \eta^{(2)} y^{(2)} \mathbf{x}^{(2)} + \eta^{(3)} y^{(3)} \mathbf{x}^{(3)}$$

$$\mathbf{w}^{(t)} = \sum_{j=1}^t \eta^{(j)} y^{(j)} \mathbf{x}^{(j)}$$

- Thus the weight vector is a linear combination of the training examples that have been updated on so far

## Dual soft-margin SVM

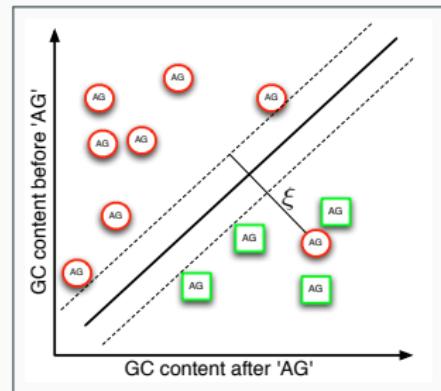
---

# Dual representation of the optimal hyperplane

It can be shown theoretically that the **optimal hyperplane** of the soft-margin SVM has a **dual representation** as the linear combination of the training data

$$\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i$$

- The coefficients, also called the **dual variables** are non-negative  $\alpha_i \geq 0$
- The positive coefficients  $\alpha_i > 0$  appear if and only if  $\mathbf{x}_i$  is a support vector, for other training points we have  $\alpha_i = 0$



## Dual representation of the optimal hyperplane

- Consequently, the functional margin  $y\mathbf{w}^T \mathbf{x}$  also can be expressed using the support vectors:

$$y\mathbf{w}^T \mathbf{x} = y \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i^T \mathbf{x}$$

- The norm of the weight vector can be expressed as

$$\mathbf{w}^T \mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i^T \sum_{j=1}^m \alpha_j y_j \mathbf{x}_j = \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

- Note that the training data appears in pairwise inner products:  $\mathbf{x}_i^T \mathbf{x}_j$

## Dual representations

- We can replace the explicit inner products with a kernel function

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$$

which computes an inner product in the space of the arguments,  
here  $\mathbb{R}^d$

- Plug in:

- Margin:

$$y\mathbf{w}^T \mathbf{x} = y \sum_{i=1}^m \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x})$$

- Squared norm:

$$\|\mathbf{w}\|^2 = \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$$

## Dual Soft-Margin SVM

A dual optimization problem for the soft-margin SVM with kernels is given by

$$\begin{array}{ll} \text{Maximize} & OBJ(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \\ \text{w.r.t} & \text{variables } \alpha \in \mathbb{R}^m \end{array}$$

$$\begin{array}{ll} \text{Subject to} & 0 \leq \alpha_i \leq C/m \\ & \text{for all } i = 1, \dots, m \end{array}$$

- It is a QP with variables  $\alpha_i$ , again with a unique optimum
- At optimum, will have implicitly computed the optimal hyperplane  $\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i$
- The data only appears through the kernel function  $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- Full derivation requires techniques of optimization theory, which we will skip here

## Kernel trick

- We can consider transformations of the input with some basis functions  $\phi : \mathbb{R}^d \mapsto \mathbb{R}^k$
- The optimal hyperplane  $\mathbf{w} \in \mathbb{R}^k$  will satisfy:  $\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \phi(\mathbf{x}_i)$
- Assume  $\kappa_\phi$  computes an inner product in the space  $\kappa_\phi(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$
- Then can compute the hyperplane in the transformed space

$$\mathbf{w}^T \phi(\mathbf{x}) = \sum_{i=1}^m \alpha_i y_i \kappa_\phi(\mathbf{x}_i, \mathbf{x})$$

and the squared norm of the weight vector

$$\|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w} = \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \kappa_\phi(\mathbf{x}_i, \mathbf{x}_j)$$

- We do not need to explicitly refer to the transformed data  $\phi(\mathbf{x})$  or the weight vector  $\mathbf{w}$ , both of which could be high-dimensional
- This is sometimes called the **kernel trick**

# **Stochastic Dual Coordinate Ascent**

---

## Stochastic Dual Coordinate Ascent for dual SVM

- Consider an algorithm updating one randomly selected dual variable (i.e. coordinate, hence the name of the method)  $\alpha_i$  at a time , while keeping the other dual variables fixed
- We take the direction of the positive gradient of the dual SVM objective  $OBJ(\alpha)$

$$\begin{aligned}\Delta\alpha_i &= \frac{\partial}{\partial\alpha_i} OBJ(\alpha) = \frac{\partial}{\partial\alpha_i} \left( \sum_{k=1}^m \alpha_k - \frac{1}{2} \sum_{k=1}^m \sum_{j=1}^m \alpha_k \alpha_j y_k y_j \kappa(\mathbf{x}_k, \mathbf{x}_j) \right) \\ &= 1 - y_i \sum_{j=1}^m \alpha_j y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) = 1 - y_i f(\mathbf{x}_i)\end{aligned}$$

where we used the dual representation

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} = \sum_{j=1}^m \alpha_j y_j \kappa(\mathbf{x}_j, \mathbf{x})$$

## Stochastic Dual Coordinate Ascent for dual SVM

- The update direction thus depends on the margin:

$$\Delta\alpha_i = \begin{cases} < 0 & \text{if } y_i f(\mathbf{x}_i) > 1 \\ 0 & \text{if } y_i f(\mathbf{x}_i) = 1 \\ > 0 & \text{if } y_i f(\mathbf{x}_i) < 1 \end{cases}$$

- If the margin is too small (Hinge loss is positive),  $\alpha_i$  is increased, if there is more than the required margin,  $\alpha_i$  is decreased
- Note the analogy to updating  $\mathbf{w} = \mathbf{w} + \eta y_i \mathbf{x}_i$ , when  $\mathbf{x}_i$  has too small margin –  $\eta$  and  $\alpha_i$  have similar roles

# Stepsize

- We can easily find the optimal update direction and step-size by setting

$$\frac{\partial}{\partial \alpha_i} OBJ(\alpha) = 0,$$

it will give:

$$\alpha_i = \frac{1 - y_i \sum_{j \neq i} \alpha_j y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)}{\kappa(\mathbf{x}_i, \mathbf{x}_i)}$$

- Finally, the bounds  $0 \leq \alpha_i \leq C/m$  need to be adhered  
 $\alpha_i = \min(C/m, \max(\alpha_i, 0))$

# Stochastic Dual Coordinate Ascent for SVM

Initialize  $\alpha = \mathbf{0}$

**repeat**

Select a random training example  $(x_i, y_i)$

Update the dual variable:  $\alpha_i = \frac{1 - y_i \sum_{j \neq i} \alpha_j y_j \kappa(x_i, x_j)}{\kappa(x_i, x_i)}$

Clip to satisfy the constraints:  $\alpha_i = \min(C/m, \max(0, \alpha_i))$

**until** stopping criterion is satisfied

**return**  $\alpha$

# Summary

- Support vector machines are classification methods based on the principle of margin maximization
- SVMs can be efficiently optimized using Stochastic gradient techniques specially developed for piecewise differentiable functions, such as the higge loss
- Dual representation of SVM allows the use of kernel functions (more on kernels next lecture)

# CS-E4710 Machine Learning: Supervised Methods

## Lecture 7: Kernel methods

---

Juho Rousu

25. October, 2022

Department of Computer Science  
Aalto University

# Kernel methods

Key characteristics of kernel methods:

- **Embedding:** Inputs  $\mathbf{x} \in X$  from some input space  $X$  are embedded into a *feature space*  $F$  via a feature map  $\phi : X \mapsto F$ .  $\phi$  may be highly **non-linear** and  $F$  potentially very high-dimensional vector space
- **Linear models:** are built for the patterns in the feature space (typically  $\mathbf{w}^T \phi(\mathbf{x})$ ); efficient to find the optimal model, convex optimization
- **Kernel trick:** Algorithms work with kernels, inner products of feature vectors  $\kappa(\mathbf{x}, \mathbf{z}) = \sum_j \phi_j(\mathbf{x})\phi_j(\mathbf{z})$  rather than the explicit features  $\phi(\mathbf{x})$ ; side-step the efficiency problems of high-dimensionality
- **Regularized learning:** To avoid overfitting, large feature weights are penalized, separation by large margin is favoured

## Data analysis tasks via kernels

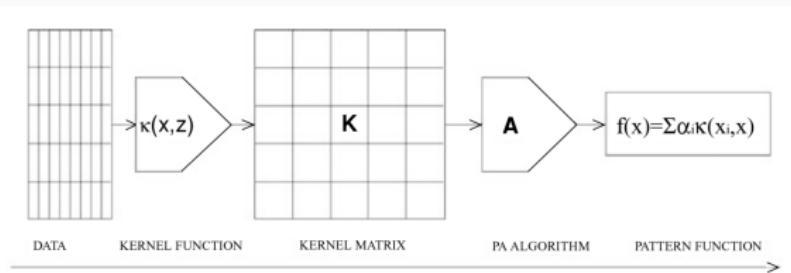
Many data analysis algorithms can be 'kernelized', i.e. transformed to an **equivalent** form by replacing object descriptions (feature vectors) by pairwise similarities (kernels):

- Classification (SVM)
- Regression
- Ranking
- Novelty detection
- Clustering
- Principal component analysis, canonical correlation analysis
- Multi-label/Multi-task/Structured output
- ...

More of the tasks beyond classification will be discussed at the course  
Kernel Methods in Machine Learning (Spring 2023 by Rohit Babbar)

# Modularity of kernel methods

- Algorithms are designed that work with arbitrary inner products (or kernels) between inputs
- The same algorithm will work with any inner product (or kernel)
- This allows theoretical properties of the learning algorithm to be investigated and the results will carry to all application domains
- Kernel will depend on the application domain; prior information is encoded into the kernel



# What is a kernel?

- Informally, a **kernel** is a function that calculates the similarity between two objects, e.g.
  - two proteins
  - two images
  - two documents
  - ...
- $x_i \in X$  and  $x_j \in X$ 
  - $X =$  set of all proteins in the nature (finite set)
  - $X =$  all possible images (infinite set)
  - $X =$  all possible documents (infinite set)
- $\kappa: X \times X \rightarrow \mathbb{R}$

## Data and Feature maps

- We assume inputs  $\mathbf{x}$  to come from an arbitrary set  $X$ :
  - Vectors, matrices, tensors
  - Structured objects: Sequences, hierarchies, graphs
- We further assume the data items can be expressed as objects in some feature space  $F$
- Typically  $F$  is a space of feature vectors,  $F \subseteq \mathbb{R}^N$ , where  $N$  is the dimension of the feature space, or more generally matrices or tensors.
- Inputs  $\mathbf{x}$  are mapped to this space by a feature map  $\phi : X \mapsto F$
- $\phi(\mathbf{x})$  is the image of the data item in the feature space

# What is a kernel?

- Formally: a kernel function is an inner product (scalar product, dot product) in a feature space  $F$ , denoted by  $\langle \cdot, \cdot \rangle_F$ 
  - Often the subscript  $F$  is dropped when it is clear from context
- Linear kernel: If  $\mathbf{x} \in \mathbb{R}^n$  and the feature map  $\phi(\mathbf{x}) = \mathbf{x}$  is the identity, then  $F = \mathbb{R}^n$  and the resulting kernel

$$\kappa_{lin}(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle_F = \langle \mathbf{x}, \mathbf{z} \rangle_{\mathbb{R}^n}$$

is called the linear kernel

- Linear kernel therefore corresponds to the dot product in  $\mathbb{R}^n$

$$\kappa_{lin}(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^n x_j z_j = \mathbf{x}^T \mathbf{z}$$

## Geometric interpretation

- Geometric interpretation of the linear kernel: cosine angle between two feature vectors

$$\cos \beta = \frac{\mathbf{x}^T \mathbf{z}}{\|\mathbf{x}\|_2 \|\mathbf{z}\|_2} = \frac{\kappa_{lin}(\mathbf{x}, \mathbf{z})}{\sqrt{\kappa_{lin}(\mathbf{x}, \mathbf{x})} \sqrt{\kappa_{lin}(\mathbf{z}, \mathbf{z})}},$$

where

$$\|\mathbf{x}\|_2 = \sqrt{\kappa_{lin}(\mathbf{x}, \mathbf{x})} = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = \sqrt{\sum_{j=1}^n x_j^2}$$

is the Euclidean norm.

## Kernel vs. Euclidean distance

- Assume two vectors  $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$  with unit length  $\|\mathbf{x}\|_2 = \|\mathbf{z}\|_2 = 1$
- Kernel:  $\kappa(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$
- Euclidean Distance:  $d(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|_2 = \sqrt{\sum_{k=1}^n (x_k - z_k)^2}$
- Expanding the squares and using unit length of the vectors we get:

$$\begin{aligned}\frac{1}{2}d(\mathbf{x}, \mathbf{z})^2 &= \frac{1}{2}\|\mathbf{x} - \mathbf{z}\|_2^2 = \frac{1}{2}(\mathbf{x} - \mathbf{z})^T(\mathbf{x} - \mathbf{z}) = \\ &= \frac{1}{2} (\|\mathbf{x}\|_2^2 - 2\mathbf{x}^T \mathbf{z} + \|\mathbf{z}\|_2^2) \\ &= 1 - \mathbf{x}^T \mathbf{z} = 1 - \kappa(\mathbf{x}, \mathbf{z})\end{aligned}$$

## Hilbert space\*

Formally the underlying space of a kernel is required to be a Hilbert space

A Hilbert space is a real vector space  $\mathcal{H}$ , with the following additional properties

- Equipped with a **inner product**, a map  $\langle ., . \rangle$ , which satisfies for all objects  $x, x', z \in \mathcal{H}$ 
  - linear:  $\langle ax + bx', z \rangle = a\langle x, z \rangle + b\langle x', z \rangle$
  - symmetric:  $\langle x, x' \rangle = \langle x', x \rangle$
  - positive semi-definite:  $\langle x, x \rangle \geq 0$ ,  $\langle x, x \rangle = 0$  if and only if  $x = 0$
- Complete: every Cauchy sequence  $\{h_n\}_{n \geq 1}$  of elements in  $\mathcal{H}$  converges to an element of  $\mathcal{H}$
- Separable: there is a countable set of elements  $\{h_1, h_2, \dots\}$  in  $\mathcal{H}$  such that for any  $h \in \mathcal{H}$  and every  $\epsilon > 0$   $\|h_i - h\| < \epsilon$ .

On this lecture  $\mathcal{H} = \mathbb{R}^N$ , where the dimension  $N$  is finite or infinite. Both cases are Hilbert spaces.

\* Advanced material; will not be examined

# The kernel matrix

- In kernel methods, a **kernel matrix**, also called the **Gram matrix**, an  $m \times m$  matrix of pairwise similarity values is used:

$$\mathbf{K} = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \kappa(\mathbf{x}_1, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_m) \\ \kappa(\mathbf{x}_2, \mathbf{x}_1) & \kappa(\mathbf{x}_2, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_2, \mathbf{x}_m) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_m, \mathbf{x}_1) & \kappa(\mathbf{x}_m, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix}$$

- Each entry is an inner product between two data points  
 $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ , where  $\phi : X \mapsto \mathcal{F}$  is a feature map
- Since an inner product is symmetric,  $\mathbf{K}$  is a symmetric matrix

## Kernel matrix example

The heatmap on the right illustrates a typical kernel matrix

- Rows and columns index examples in the data
- The colors corresponds to the kernel values (red=high value, blue=low value)
- Notice the dark red diagonal elements: they correspond to the kernel value between an example and itself ( $\kappa(\mathbf{x}_i, \mathbf{x}_i)$ )
- Red/orange diagonal blocks correspond to clusters of similar examples as defined by the kernel values  $\kappa(\mathbf{x}_i, \mathbf{x}_j)$

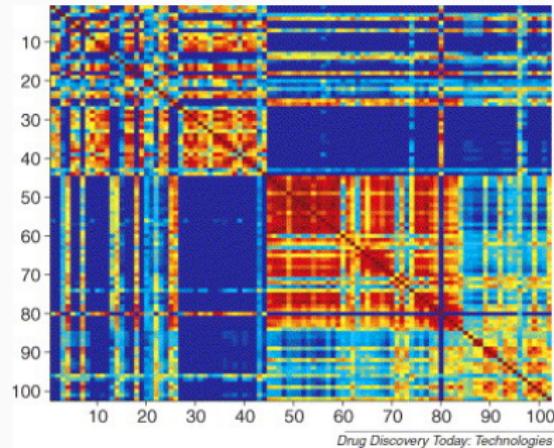


Image source; Mark Girolami et al. 2006. Analysis of complex, multidimensional datasets, Drug Discovery Today: Technologies, 3, 1, pp. 13–19

## Processing the kernel matrix

$$\mathbf{K} = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \kappa(\mathbf{x}_1, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_m) \\ \kappa(\mathbf{x}_2, \mathbf{x}_1) & \kappa(\mathbf{x}_2, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_2, \mathbf{x}_m) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_m, \mathbf{x}_1) & \kappa(\mathbf{x}_m, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix}$$

- Processing the kernel matrix during preprocessing, training and prediction time is the major factor of the time-complexity of kernel methods
- Compare the two matrices assuming  $m$  examples of dimension  $\mathbb{R}^N$ 
  - The kernel matrix has  $m^2$  items, independently from  $N$
  - The data matrix, matrix of feature vectors of the training data has size  $mN$
- Consequently, the kernel matrix scales better than the data matrix when  $N > m$

# The kernel matrix

- A symmetric matrix  $\mathbf{A} \in \mathbb{R}^{m \times m}$  is positive semi-definite (PSD) if for any vector  $\mathbf{v} \in \mathbb{R}^m$ , we have  $\mathbf{v}^T \mathbf{A} \mathbf{v} \geq 0$
- A symmetric PSD matrix has non-negative eigenvalues  $\lambda_1 \geq \dots \geq \lambda_m \geq 0$
- The **kernel matrix** corresponding to the kernel function  $\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$  on a set of data points  $\{\mathbf{x}_i\}_{i=1}^m$  is positive semidefinite:

$$\begin{aligned}\mathbf{v}^T \mathbf{K} \mathbf{v} &= \sum_{i,j=1}^n v_i \mathbf{K}_{ij} v_j = \sum_{i,j=1}^m v_i \langle \phi(x_i), \phi(x_j) \rangle v_j = \\ &= \left\langle \sum_{i=1}^m v_i \phi(x_i), \sum_{j=1}^m v_j \phi(x_j) \right\rangle = \left\| \sum_{i=1}^m v_i \phi(x_i) \right\|^2 \geq 0\end{aligned}$$

## PSD property and optimization

- Consider objective of the dual SVM optimization problem

$$OBJ(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha}$$

where we denoted  $\mathbf{H} = (y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1}^m$

- It is easy to verify that  $\mathbf{H}$  is the Hessian matrix of second derivatives of the objective;

$$\mathbf{H} = \left( \frac{\partial^2 OBJ(\alpha)}{\partial \alpha_i \partial \alpha_j} \right)_{i,j=1}^m$$

- If  $\mathbf{H}$  is PSD  $OBJ(\alpha)$  is concave ( $-OBJ(\alpha)$  is convex), and has no non-optimal local maxima
- However,  $\mathbf{H}$  is PSD if and only if  $\mathbf{K}$  is PSD
- Thus, a PSD kernel matrix  $\mathbf{K}$  ensures that we can find a global optimum by gradient descent approaches

# Rademacher complexity

- Assume a symmetric positive definite kernel  $\kappa : X \times X \mapsto \mathbb{R}$  with associated feature map  $\phi$ , and a sample  $S$  of size  $m$ , with the kernel matrix  $\mathbf{K} = (\kappa(x_i, x_j))_{i,j=1}^m$ , and  $\kappa(\mathbf{x}_i, \mathbf{x}_i) \leq r^2$  for all  $i = 1, \dots, m$
- Empirical Rademacher complexity of the hypothesis class containing support vector machines

$$\mathcal{H} = \{\mathbf{x} \mapsto \langle \mathbf{w}, \phi(\mathbf{x}) \rangle : \|\mathbf{w}\| \leq B\}$$

for some  $B \geq 0$  satisfies (c.f. Mohri book for the proof)

$$\hat{\mathcal{R}}_S(\mathcal{H}) \leq \frac{B \sqrt{\text{trace}(\mathbf{K})}}{m}$$

- The key quantities are
    - the upper bound  $B$  of the norm of weight vector – relates to the margin
    - the trace of the kernel matrix
- $$\text{trace}(\mathbf{K}) = \sum_{i=1}^m \kappa(\mathbf{x}_i, \mathbf{x}_i) = \sum_{i=1}^m \|\phi(\mathbf{x}_i)\|^2 \leq mr^2$$
- relates to the norm of the data points

## Generalization error bound

- We can plug the above to a Rademacher complexity based generalization bound (c.f. Lecture 3)

$$\begin{aligned} R(h) &\leq \hat{R}(h) + \hat{\mathcal{R}}(\mathcal{H}) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}} \\ &\leq \hat{R}(h) + \frac{B\sqrt{\text{trace}(\mathbf{K})}}{m} + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}} \end{aligned}$$

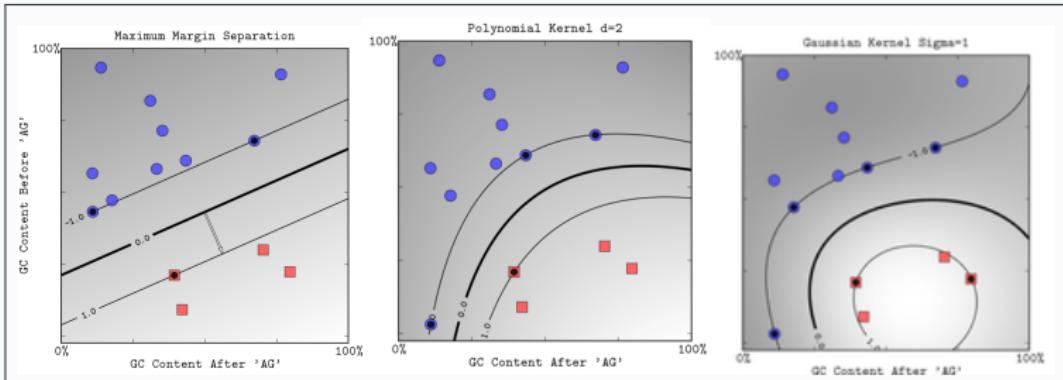
- Evaluating this bound required observing (1) the empirical risk of the hypothesis  $\hat{R}(h)$  on training data, (2) the norm of the weight vector  $B = \|\mathbf{w}\| = \sqrt{\sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)}$ , and (3) the trace of the kernel matrix
- Note that we do not need to run simulations with random labelings to use this bound!

## Non-linear kernels

---

# Non-linear kernels

- By defining kernels that are non-linear functions of the original feature vectors, a linear models (e.g. SVM classifier) can be turned into a non-linear model
- However, the learning algorithm does not need to be changed, apart from plugging in the new kernel matrix
- The most commonly used non-linear kernels:
  - Polynomial kernel:  $\kappa_{pol}(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z} + c)^q$
  - Gaussian (or radial basis function, RBF) kernel:  
$$\kappa_{RBF}(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2 / (2\sigma^2))$$



## Non-linear kernels: Polynomial kernel

- Given inputs  $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$ , the **polynomial kernel** is given by

$$\kappa_{pol}(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z} + c)^q$$

- Integer  $q > 0$  gives the **degree** of the polynomial kernel
- Real value  $c \geq 0$  is a weighting factor for lower order polynomial terms
- The underlying features are **non-linear**: monomial combinations  $x_1 \cdot x_2 \cdots x_k$  of degree  $k \leq q$  of the original features  $x_j$

## Example: Polynomial kernel on 2D inputs

- Consider two-dimensional inputs  $\mathbf{x} = [x_1, x_2]^T \in \mathbb{R}^2$
- The second degree polynomial kernel is given by  
$$\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^2$$
- We can write it as a inner product in  $\mathbb{R}^6$ :

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{x}') &= (\mathbf{x}^T \mathbf{x}' + c)^2 = (x_1 x'_1 + x_2 x'_2 + c)^2 = \\ &= x_1 x'_1 x_1 x'_1 + x_2 x'_2 x_2 x'_2 + c^2 + \\ &\quad + 2x_1 x'_1 x_2 x'_2 + 2cx_1 x'_1 + 2cx_2 x'_2 \\ &= \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1 x_2 \\ \sqrt{2}cx_1 \\ \sqrt{2}cx_2 \\ c \end{bmatrix}^T \begin{bmatrix} x'_1^2 \\ x'_2^2 \\ \sqrt{2}x'_1 x'_2 \\ \sqrt{2}cx'_1 \\ \sqrt{2}cx'_2 \\ c \end{bmatrix} \\ &= \phi(\mathbf{x})^T \phi(\mathbf{x}'),\end{aligned}$$

where  $\phi(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1 x_2, \sqrt{2}cx_1, \sqrt{2}cx_2, c]^T$

## Non-linear kernels: Polynomial kernel

$$\kappa_{pol}(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + c)^q$$

A linear model in the polynomial feature space corresponds to a **non-linear model** in the original feature space

- In the previous example, the model

$$\mathbf{w}^T \phi(\mathbf{x}) = w_1 x_1^2 + w_2 x_2^2 + w_3 \sqrt{2} x_1 x_2 + w_4 \sqrt{2c} x_1 + w_5 \sqrt{2c} x_2 + w_6 c = 0$$

is a second degree polynomial in the original inputs space, but a hyperplane in the new 6-dimensional feature space

- Using the dual representation  $\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \phi(\mathbf{x}_i)$ , the polynomial kernel allows non-linear classification in the input space by

$$\mathbf{w}^T \phi(\mathbf{x}) = \sum_i \alpha_i y_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle = \sum_i \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x})$$

## Example: XOR with polynomial kernel

Consider the following simple example:

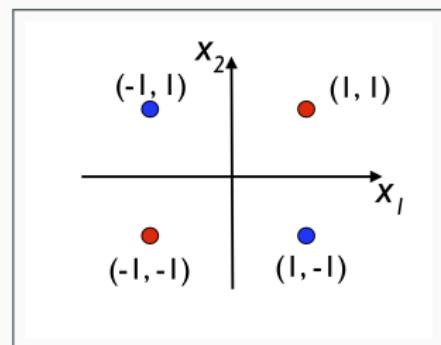
- Input data points

$$\{(-1, -1), (-1, 1), (1, -1), (1, 1)\}$$

and the label (red = 1, blue = -1)  
given by a XOR type function

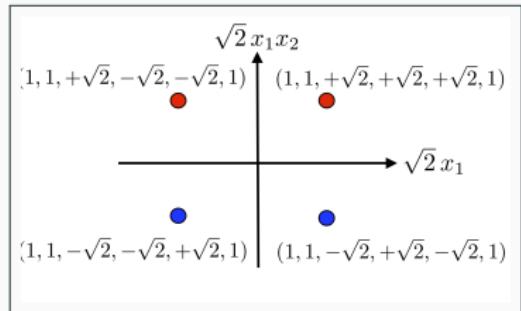
$$y = x_1 x_2 = \begin{cases} +1 & \text{if } x_1 = x_2 \\ -1 & \text{if } x_1 \neq x_2 \end{cases}$$

- The classes are not linearly separable:  
there is no consistent line that  
separates the two classes



## Example: XOR with polynomial kernel

- However, map the data using the feature map  $\phi(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}cx_1, \sqrt{2}cx_2, c]^T$  underlying the polynomial kernel function



- Now, the example data is linearly separable in the feature space, for example, choose  $\alpha_i = 1/(4\sqrt{2})$ , for all  $i$ :

$$\begin{aligned}\mathbf{w} &= \sum_i \alpha_i y_i \phi(\mathbf{x}_i) = \\ &= (\phi([-1, -1]^T) - \phi([-1, 1]^T) - \phi([1, -1]^T) + \phi([1, 1]^T)) / (4\sqrt{2}) \\ &= [0, 0, 1, 0, 0, 0]^T\end{aligned}$$

- We can consistently classify the example data by using the kernel function  $\kappa(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$ :  $h(\mathbf{x}) = \text{sgn} \left( \sum_{i=1}^m \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) \right)$

## Non-linear kernels: Polynomial kernel

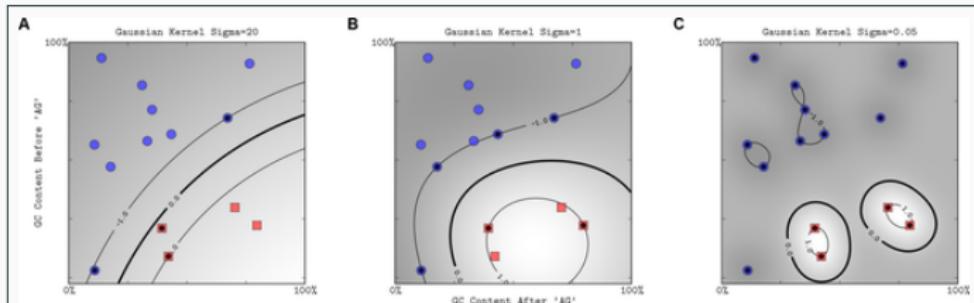
- The dimension of the polynomial feature space is  $\binom{d+q}{q} = O((d+q)^q)$  where  $d$  is the dimension of the input space  $X$  and  $q$  is the degree of the polynomial.
- Explicitly maintaining the feature map  $\phi(\mathbf{x})$  and the weight vector  $\mathbf{w}$ , and evaluating the model  $\mathbf{w}^T \phi(\mathbf{x})$  takes  $O(d^q)$  time and space
- However, the polynomial kernel  $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^q$  can be computed in time  $O(d)$  in preprocessing, and evaluated in constant time
- Evaluating the model using the dual representation  $\sum_{i=1}^m \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x})$  takes  $O(m)$  time
- Trade-off: No computational overhead from working in the high-dimensional feature space, but linear dependency in the size of training data

# Non-linear kernels: Gaussian kernel (Radial basis function kernel, RBF)

Gaussian kernel between two inputs  $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$  with bandwidth parameter  $\sigma > 0$ :

$$\kappa_{RBF}(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2/(2\sigma^2))$$

- Large values for  $\sigma$  give a smoother kernel, slower decay of kernel values as a function of the squared euclidean distance  $\|\mathbf{x} - \mathbf{z}\|^2$ , and a more linear decision boundary
- Small values for  $\sigma$  give a less smooth kernel, faster decay of kernel values as a function of the squared euclidean distance  $\|\mathbf{x} - \mathbf{z}\|^2$ , and a more non-linear decision boundary

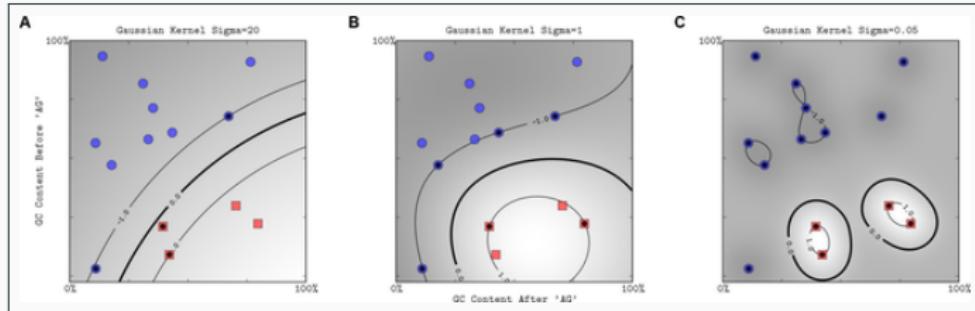


# Non-linear kernels: Gaussian kernel (Radial basis function kernel, RBF)

Gaussian kernel can be seen to correspond to an **infinite dimensional** polynomial kernel:

$$\kappa_{RBF}(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2/(2\sigma^2)) = \sum_{n=0}^{\infty} \frac{(\mathbf{x}^T \mathbf{z})^n}{\sigma^{2n} n!}$$

- It can be shown through the power series expansion of  $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$
- Each individual term is an degree- $n$  polynomial kernel  $(\mathbf{x}^T \mathbf{z})^n$ , exponentially down-weighted by  $\frac{1}{\sigma^{2n} n!}$



## Designing kernels

---

## Several ways to get to a kernel

---

Approach I. Construct a feature map  $\phi$  and think about efficient ways to compute the inner product  $\langle \phi(x), \phi(x) \rangle$

- If  $\phi(x)$  is very high-dimensional, computing the inner product element by element is slow, we don't want to do that
- For several cases, there are efficient algorithms to compute the kernel in low polynomial time, even with exponential or infinite dimension of  $\phi$

## Several ways to get to a kernel

---

Approach II. Construct similarity measure and show that it qualifies as a kernel:

- Show that for any set of examples the matrix  $K = (\kappa(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1}^m$  is positive semi-definite (PSD).
- In that case, there always is an underlying feature representation, for which the kernel represents the inner product
- Example: if you can show the matrix is a covariance matrix for some variates, you will know the matrix will be PSD.

## Several ways to get to a kernel

Approach III. Convert a distance or a similarity into a kernel

- Take any distance  $d(\mathbf{x}, \mathbf{z})$  or a similarity measure  $s(\mathbf{x}, \mathbf{z})$  (that do not need to be a kernel)
- In addition a set of data points  $Z = \{\mathbf{z}_j\}_{j=1}^m$  from the same domain is required (e.g. training data)
- Construct feature vector from distances (similarly for  $s$ ):  
$$\phi(\mathbf{x}) = (d(\mathbf{x}, \mathbf{z}_1), d(\mathbf{x}, \mathbf{z}_2), \dots, d(\mathbf{x}, \mathbf{z}_m))$$
- Compute linear kernel, also known as the **empirical kernel map**:  
$$\kappa(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$
- This will always work technically, but requires that the data  $Z$  captures the essential patterns in the input space  $\implies$  need enough data

# Several ways to get to a kernel

## Approach IV. Making kernels from kernels

- Examples of elementary operations that give valid kernels when applied to kernels  $\kappa_n, n = 1, 2, \dots$ 
  1. Sum:  $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$
  2. Scaling with a positive scalar:  $\kappa(\mathbf{x}, \mathbf{z}) = a\kappa_1(\mathbf{x}, \mathbf{z}), a > 0$
  3. Itemwise product:  $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z})\kappa_2(\mathbf{x}, \mathbf{z})$
  4. Normalization:  $\kappa(\mathbf{x}, \mathbf{z}) = \frac{\kappa_1(\mathbf{x}, \mathbf{z})}{\sqrt{\kappa_1(\mathbf{x}, \mathbf{x})\kappa_1(\mathbf{z}, \mathbf{z})}} = \left\langle \frac{\phi(\mathbf{x})}{\|\phi(\mathbf{x})\|}, \frac{\phi(\mathbf{z})}{\|\phi(\mathbf{z})\|} \right\rangle$
  5. Pointwise limit:  $\kappa(\mathbf{x}, \mathbf{z}) = \lim_{n \rightarrow \infty} \kappa_n(\mathbf{x}, \mathbf{z})$
  6. Composition with a power series of radius of convergence  $\rho$ :  
$$\kappa(\mathbf{x}, \mathbf{z}) = \sum_{n=0}^{\infty} a_n \kappa(\mathbf{x}, \mathbf{z})^n, \text{ with } a_n \geq 0 \text{ for all } n, \text{ and } |\kappa(\mathbf{x}, \mathbf{z})| < \rho$$
- The operations can be combined to construct arbitrarily complex kernels, e.g. polynomial kernels and Gaussian kernels can be derived this way (see details in the Mohri book ch. 6)

## Kernels for structured data

---

## Kernels for structured data

- In many applications the data does not come in the form of numerical vectors or data matrices, but from a general set of objects  $X$ , for example:
  - Sequential data - text analysis kernels, string kernels
  - Molecular data - graph kernels
  - Structured documents, context-free grammars, classification taxonomies, ... - tree kernels
- To compute kernels we could define a feature map  $\phi : X \mapsto F$  and compute feature vectors  $\phi(x)$  for our data, and the kernel as the inner product:  $\kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_F$
- Alternatively, we can use an algorithm that directly computes the kernel values  $\kappa(x_i, x_j)$  for any pair of objects  $x_i, x_j$
- The latter approach can be very effective if the feature space  $F$  has a high dimension

# Kernels for structured data

---

- The commonly seen form for kernels for two structured objects  $x_i$  and  $x_j$

$$\kappa(x_i, x_j) = \sum_{s \in S} \phi_s(x_i) \phi_s(x_j),$$

where  $S$  is the set of substructures of interest and  $\phi_s(x)$  is either

- An indicator function:  $\phi_s(x) = 1$  if and only if  $x$  contains  $s$
- A count:  $x$  contains  $\phi_s(x)$  instances of  $s$
- In many cases, the set  $S$  has exponential size in the size of the objects and the feature vectors  $\phi(x)$  are high dimensional and sparse
- Efficient algorithms exist for computing the kernels  $\kappa(x_i, x_j)$  directly from the structured data, skipping writing down the feature vectors  $\phi(x)$

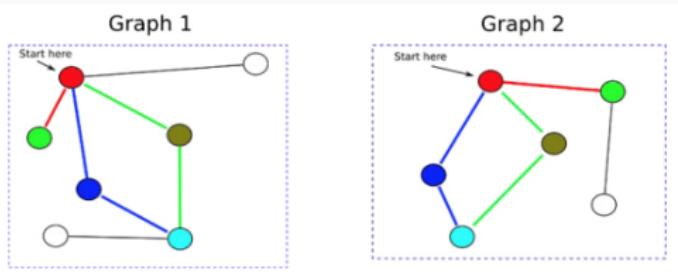
## Example: String kernels

- String kernels is a family of kernels between sequences based on "counting" common subsequences two sequences have
- Underlying feature map, also called subsequence spectrum, contains a feature for each possible substring
- The feature spaces induced by subsequences are generally exponential in the length of the subsequences
- However, low polynomial time algorithms (linear to quadratic time) exist to compute string kernels
- For more information see Lodhi et al. "Text classification using string kernels." Journal of Machine Learning Research 2.Feb (2002): 419-444.

$x$  AAACAAATAAGTAACTAATCTTTACGAAGAACGTTCAACCATTGAG  
 $x'$  TACCTAATTATGAAATTAAATTTCAGTGTGCTGATGGAAACGGAGAAGTC

## Example: Graph kernels

- Basic idea: count common substructures in two graphs
- Challenging problem in general due to the underlying NP-hard subgraph isomorphism problem: given two graphs  $G$  and  $S$ , does a graph  $S$  appear in  $G$  as a subgraph
- Polynomial-time kernel computation possible for restricted substructures:
  - Random walks
  - Tree-shaped subgraphs
  - Small general subgraphs
- For more information see Vishwanathan et al. "Graph kernels." Journal of Machine Learning Research 11 (2010): 1201-1242.



# Summary

- Kernel methods are a broad class of data analysis methods
- Kernels allow efficient non-linear learning in high-dimensional feature spaces
- Special kernels can be designed for different data types such as sequential or graph data
- Time-complexity of kernel methods generally scale quadratically in the number of training points (due to the kernel matrix), which can be a limitation when huge datasets are processed
- More on kernel methods on the course CS-E4830 Kernel methods in machine learning (Spring 2023)

# CS-E4710 Machine Learning: Supervised Methods

## Lecture 8: Neural networks

---

Juho Rousu

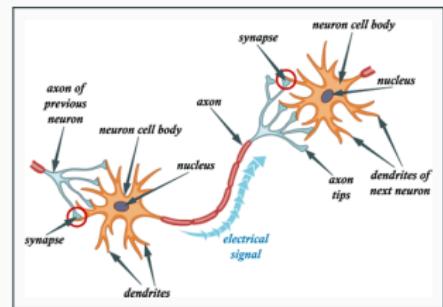
1. November, 2022

Department of Computer Science  
Aalto University

# Neural networks as models of the brain

Neural networks take inspiration from the human brain, with artificial neurons as computation units and edges between the units model the synapses

- However, compared to artificial neural networks, human brain has huge number of neurons ( $10^{11}$ ) and synapses ( $10^5$ )
- Each neuron is believed to operate a 'clock speed' of only 100Hz whereas computers work at clock speeds of a few Gigaherz.



Source:

<https://pulpbits.net/>

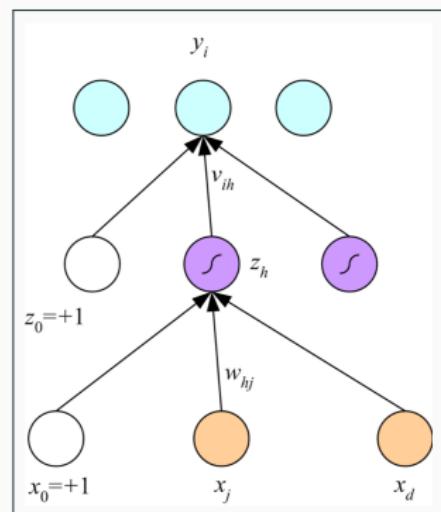
## Multi-layer perceptrons

---

# Multi-layer perceptrons

Multi-layer perceptron is a neural network that combines several perceptrons to achieve non-linear modelling

- Multi-layer perceptrons implement a layered network structure:
  - input layer
  - one or more hidden layers
  - output layer
- Nodes in adjacent layers are connected through weighted edges
- The output of each node is fed through activation functions that is typically non-linear

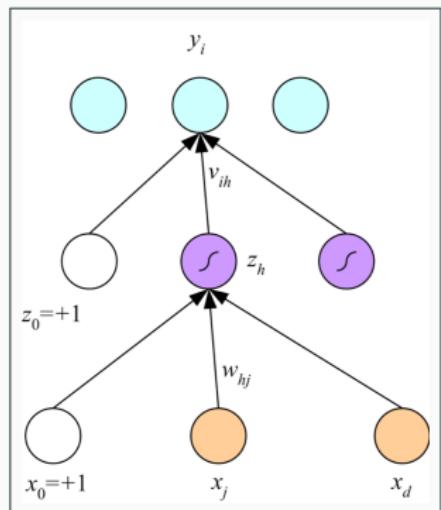


# Multilayer perceptron

The output of a two-layer MLP is computed as follows

- Input  $\mathbf{x}$ , augmented by the constant  $x_0 = 1$  is fed to the input layer
- The input values are fed to a perceptron unit  $h$  in the hidden layer, which computes the linear model  $\mathbf{w}_h^T \mathbf{x}$
- An activation function  $\sigma_h$  (e.g. the logistic function) is then applied to obtain the activation level of the hidden unit

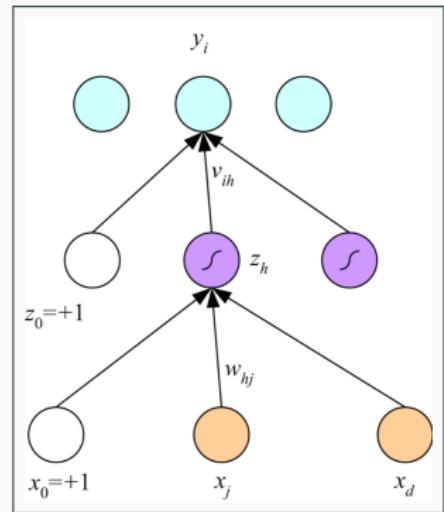
$$z_h = \sigma_h(\mathbf{w}_h^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}_h^T \mathbf{x})}$$



# Multilayer perceptron

- The values  $z_h$  from the hidden units are fed to the output layer through another linear model:  $\mathbf{v}_i^T \mathbf{z}$
- A activation function is again computed  $y_i = \sigma_i(\mathbf{v}_i^T \mathbf{z})$
- Thus, as a whole, the output  $y_i$  is the value of the function

$$y_i = \sigma_i(\mathbf{v}_i^T (\sigma_h(\mathbf{w}_h^T \mathbf{x}))_{h=1}^H)$$



## Activation functions

Each neuron  $v_h$  computes an activation function  $\sigma$ , which can be for example:

- Linear function (used in the output layer for regression):

$$\sigma(\mathbf{w}^T \mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

- The sign function:  $\sigma(\mathbf{w}^T \mathbf{x}) = \text{sgn}(\mathbf{w}^T \mathbf{x})$

- A threshold function (also called the rectified linear unit, ReLU):

$$\sigma(\mathbf{w}^T \mathbf{x}) = \begin{cases} \mathbf{w}^T \mathbf{x} & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Logistic function:  $\sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})} \in [0, 1]$

- Hyperbolic tangent (another sigmoid function that outputs values between -1 and +1):  $\sigma(\mathbf{w}^T \mathbf{x}) = \tanh \mathbf{w}^T \mathbf{x} = \frac{e^{\mathbf{w}^T \mathbf{x}} - e^{-\mathbf{w}^T \mathbf{x}}}{e^{\mathbf{w}^T \mathbf{x}} + e^{-\mathbf{w}^T \mathbf{x}}} \in [-1, +1]$

## Why do we need non-linear activation functions?

- Consider having two layer network with first layer computing  $z_h = \sum_j w_{hj}x_j$  and the second layer computing  $y_i = \sum_j v_{ih}z_h$
- The total function is thus:

$$y_i = \sum_h v_{ih} \sum_j w_{hj}x_j = \sum_j \sum_h v_{ih}w_{hj}x_j$$

- We can compute the same with a linear function:

$$y_i = \sum_j u_{ij}x_j$$

where  $u_{ij} = \sum_h v_{ih}w_{hj}$

- Thus there is no real non-linearity in the model and our model reduces to learning a linear hyperplane

To make the network structure useful, we need non-linear activation functions

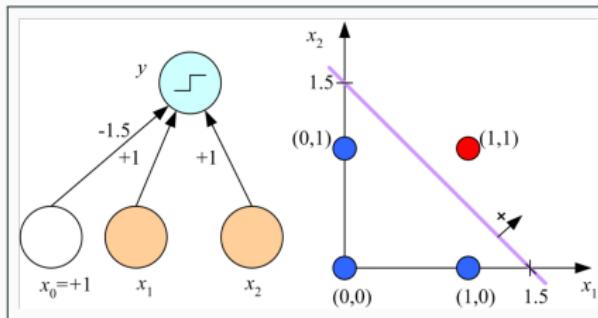
## **Expressive power of neural networks**

---

# Computing Boolean AND with the perceptron

- Perceptron can compute the Boolean AND function as follows
- Set the bias  $w_0 = -1.5$  and the weights  $w_1 = w_2 = 1$
- Now the function  $w_1x_1 + w_2x_2 + w_0 > 0$  if and only if  $x_1 = x_2 = 1$
- The function is a hyperplane (line) that linearly separates the point  $(1, 1)$  from the other three possible input combinations

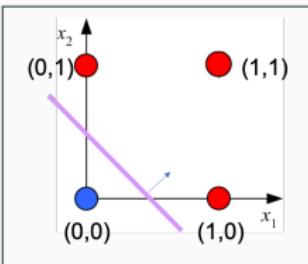
$x_1$	$x_2$	$r$
0	0	0
0	1	0
1	0	0
1	1	1



## Computing Boolean OR with the perceptron

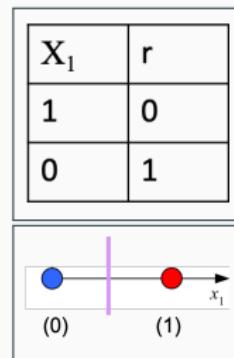
- Boolean OR function can be computed similarly
- Set the bias  $w_0 = -0.5$  and the weights  $w_1 = w_2 = 1$
- Now the function  $w_1x_1 + w_2x_2 + w_0 > 0$  if and only if  $x_1 = 1$  or  $x_2 = 1$
- The function is a hyperplane separating the point  $(0, 0)$  from the other input combinations

X <sub>1</sub>	X <sub>2</sub>	r
0	0	0
0	1	1
1	0	1
1	1	1



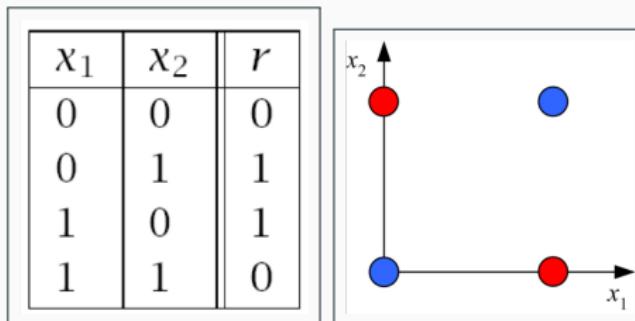
# Computing Boolean NOT with the perceptron

- Boolean NOT function is simple to compute with a neuron with only one input
- Set the bias  $w_0 = 0.5$  and the weight to  $w_1 = -1$
- Now the function  $w_1x_1 + w_0 > 0$  if and only if  $x_1 = 0$
- The function linearly separates 0 from 1



## XOR with perceptron

- The exclusive or, or XOR operator cannot be represented by the perceptron
- This is because the output XOR function is not linearly separable: there is no hyperplane that can separate  $(0, 0), (1, 1)$  from  $(0, 1), (1, 0)$

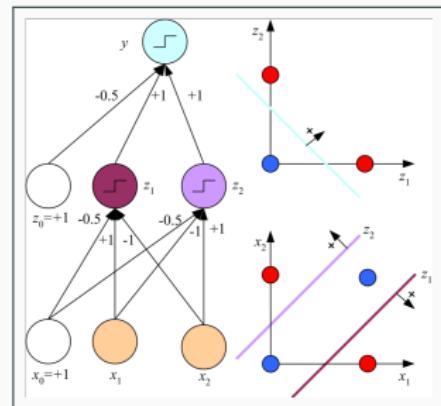


# XOR with MLP

XOR can be computed by a simple neural network consisting of three neurons

$$XOR(x_1, x_2) = (x_1 \text{ AND NOT}(x_2)) \text{ OR } (\text{NOT}(x_1) \text{ AND } x_2)$$

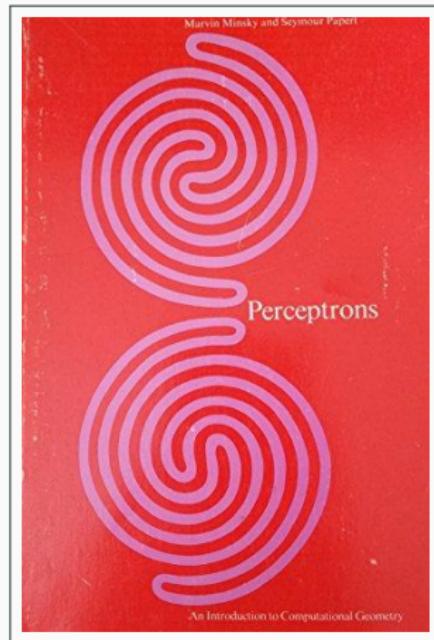
- The first layer computes two hyperplanes:
  - $z_1 = x_1 - x_2 - 0.5 > 0$  if and only if  $(x_1 \text{ AND NOT}(x_2))$
  - $z_2 = -x_1 + x_2 - 0.5 > 0$  if and only if  $(\text{NOT}(x_1) \text{ AND } x_2)$
- The second layer computes a single hyperplane implementing the OR  $z_1 + z_2 - 0.5 > 0$  if and only if  $z_1 \text{ OR } z_2$  is true



## A historical note: XOR with perceptron

A historical note:

- The inability of perceptron to compute the XOR was highlighted by Marvin Minsky and Seymour Papert in their book on Perceptrons published in 1969
- This finding contributed to the research on neural networks going out of fashion in the 1970's
- At the time, the representation power of MLPs was not widely understood
- Also, good algorithms to train MLPs were not known, so they were dismissed by the research community at the time



## Representing arbitrary boolean functions with neural nets

- Perceptron can represent all three basic logical operators AND, OR and NOT
- All Boolean functions can be represented by combinations of these basic operations
- Thus, MLPs can in principle represent arbitrary Boolean functions
- However, **learning** arbitrary Boolean functions may still require prohibitive amount of data and time (e.g. the VC dimension of arbitrary Boolean functions of  $d$  variables is  $2^d$ )

## Representing arbitrary boolean functions with neural nets

- In fact already a MLP with a single hidden layer can represent all Boolean functions
- Construction of the network is simple: there is a hidden unit  $h_i$ , for each  $\mathbf{x}_i$  for which  $f(\mathbf{x}_i) = 1$ , that will output 1 if the input equals  $\mathbf{x}_i$  (the unit computes an AND over all input variables)
- The output layer outputs +1 if any of the hidden units outputs 1 (OR over the hidden units)

# Representing arbitrary boolean functions with neural nets

- The network described before is fully memorizing the Boolean function, no learning or generalization is happening
- This network has exponential size in the number of variables
- Exponential size is not an artifact: one can prove that any network structure that allows representing any Boolean function must have exponential size in the input dimension (Shalev-Shwartz and Ben-David, 2014<sup>1</sup>)

---

<sup>1</sup>Shalev-Shwartz, S. and Ben-David, S., 2014. Understanding machine learning: From theory to algorithms. Cambridge university press.

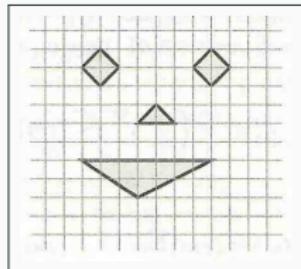
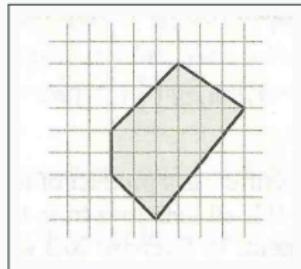
## MLPs as universal approximators

- Besides the ability of represent arbitrary Boolean functions MLPs can also approximate real valued functions that have bounded gradients (called Lipschitz functions) with arbitrary precision
- Given a function  $f(\mathbf{x})$  the network will output value between  $f(\mathbf{x}) - \epsilon$  and  $f(\mathbf{x}) + \epsilon$ , where  $\epsilon > 0$  is the desired precision.
- However, again the price to pay is the size of the network: it will necessarily be of exponential size in the input dimension  
(Shalev-Shwartz and Ben-David, 2014)

In summary, neural networks can fit any function one encounters in practice, but this may require impractically large networks.

# Geometric intuition

- A two layer network can represent convex polytopes, through an intersection of half-spaces defined by hyperplanes (top picture)
  - Each face of the polytope is defined by a single neuron in the hidden layer, the output layer computes an AND of the hidden layer activations
- A three layer network can represent disjunctions of convex polytopes: the final layer computes an OR of the second hidden layer outputs (bottom picture)



(Source:  
Shalev-Shwartz  
and Ben-David,  
2014)

## VC dimension of neural networks

The complexity of learning neural networks depends on the number of weights in the network, which equals the number of edges  $|E|$  (Shalev-Schwartz and Ben-David, 2014):

- VC-dimension of neural networks using the sign activation function is  $O(|E| \log |E|)$
- VC-dimension of neural networks using the sigmoid activation function is upper bounded by  $O(|V|^2|E|^2)$  in general, and  $O(|E|)$  if each weight is constrained to have representation of a small constant number of bits

Thus the above hypothesis classes are learnable in the PAC framework: given enough data, the generalization error can be bounded (e.g. by the bound shown in Lecture 3)

## Learning Multi-Layer Perceptrons

---

## The bad news: hardness of training MLPs

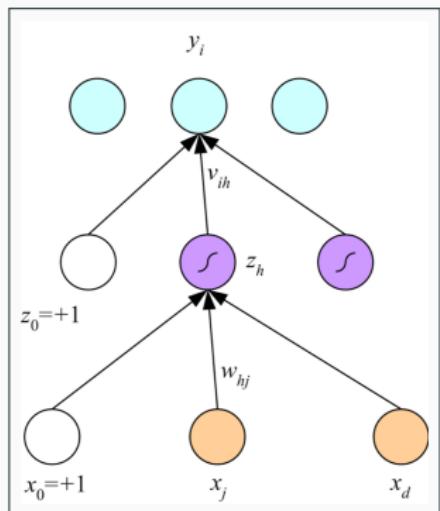
Learning optimal weights for MLPs and other neural networks is computationally hard (Shalev-Shwartz and Ben-David, 2014):

- It is NP-hard to find the parameters that minimizes the empirical error, for a network with a single hidden layer that contains 4 neurons or more
- Even close-to-minimal error is NP-hard to achieve
- Changing the structure of the network is not likely to make learning easier, since any function class that can represent intersections of halfspaces is NP-hard under some cryptographic assumptions

Thus in practice we need to resort in heuristic optimization approaches with no theoretical guarantees of optimality

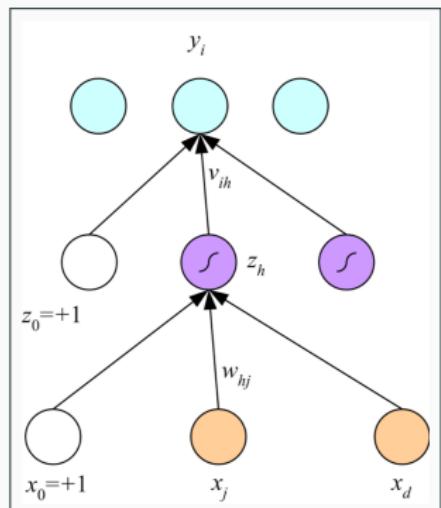
# Stochastic Gradient Descent for MLPs

- Most training algorithms for MLPs are variants of stochastic gradient decent (SGD)
- Unlike with logistic regression and SVM problems, MLP optimization is a non-convex optimization problem
  - SGD generally converges to a local optimum
  - No theoretical guarantees how close to the global optimum we are
- In practise, SGD needs to be run many times with different initializations to find a good local optimum



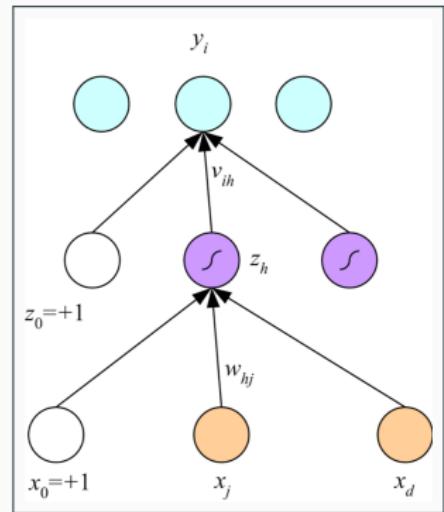
# Stochastic Gradient Descent for MLPs

- SGD requires us to compute the gradient of the loss function with respect to a training example
- Unlike Logistic regression or SVM, there is no analytical expression for the gradient
- The expression for the gradient will be in general a expression involving nested sums and products
- The computation of the gradient and the update of the weights needs to be incrementally, layer by layer



# Stochastic Gradient Descent for MLPs

- Let us study a two-layer MLP for regression
  - There is one output neuron  $i$  that has a linear activation function  $y_i = \mathbf{v}^T \mathbf{z}$   
(The figure has other outputs which we ignore here)
  - There are  $H$  hidden neurons with a logistic activation function
- $$z_h = \frac{1}{1 + \exp(-\mathbf{w}_h^T \mathbf{x})}$$
- Squared loss is used as the loss function:  $L(y_i, r_i) = \frac{1}{2}(y_i - r_i)^2$ , where  $r_i$  denotes the true output value and  $y_i$  denotes the predicted output value



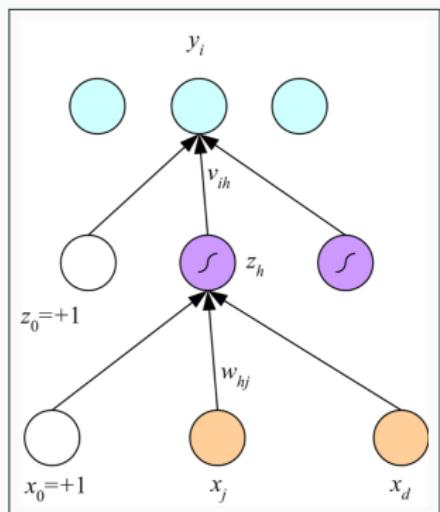
# Stochastic Gradient Descent for MLPs

We traverse the network backwards from the output layer, first taking the hidden layer as fixed, considering  $z_k$  of the hidden units as inputs

- The gradient of the loss function with respect to the output layer weights is

$$\begin{aligned}\frac{\partial}{\partial v_{ih}} L(r_i, y_i) &= \frac{\partial}{\partial v_{ih}} \frac{1}{2} (r_i - \sum_{k=0}^H v_{ik} z_k)^2 \\ &= (r_i - \sum_{k=0}^H v_{ik} z_k)(-z_h)\end{aligned}$$

- The SGD update to the weight  $v_{ih}$  is a step along the negative gradient



$$\Delta v_{ih} = \eta(r_i - y_i)z_h$$

# Stochastic Gradient Descent for MLPs

- The update for the hidden layer weights  $w_{hj}$  is not as simple, as we do not have a "desired output" for hidden layer neurons and thus no loss function either
- We can use the chain rule of differentiation

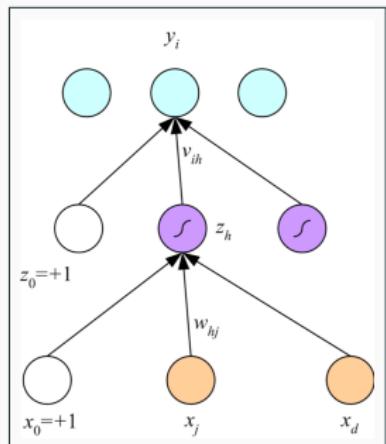
$$\frac{\partial L(r_i, y_i)}{\partial w_{hj}} = \frac{\partial L(r_i, y_i)}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}$$

The derivatives of the three factors are given by:

$$\frac{\partial L(r_i, y_i)}{\partial y_i} = \frac{\partial}{\partial y_i} \frac{1}{2} (r_i - y_i)^2 = -(r_i - y_i)$$

$$\frac{\partial y_i}{\partial z_h} = \frac{\partial}{\partial z_h} \sum_{k=0}^H v_{ik} z_k = v_{ih}$$

$$\frac{\partial z_h}{\partial w_{hj}} = \frac{\partial}{\partial w_{hj}} \sigma_h \left( \sum_{k=0}^d w_{hk} x_k \right)$$

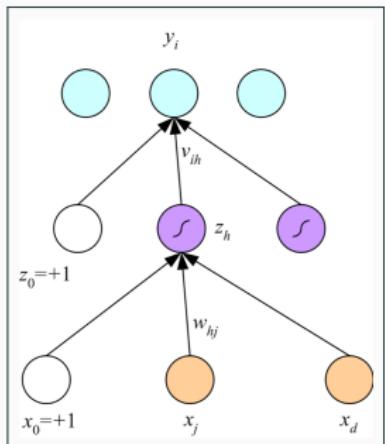


# Stochastic Gradient Descent for MLPs

Using the logistic function as the activation function for the hidden layer

$$z_h = \sigma_h\left(\sum_{k=0}^d w_{hk}x_k\right) = \frac{1}{1+\exp(-\sum_{k=0}^d w_{hk}x_k)}$$
 we get:

$$\begin{aligned}\frac{\partial z_h}{\partial w_{hj}} &= \frac{\partial}{\partial w_{hj}} \sigma_h\left(\sum_{k=0}^d w_{hk}x_k\right) \\ &= \sigma_h\left(\sum_{k=0}^d w_{hk}x_k\right)\left(1 - \sigma_h\left(\sum_{k=0}^d w_{hk}x_k\right)\right)x_j \\ &= z_h(1 - z_h)x_j\end{aligned}$$



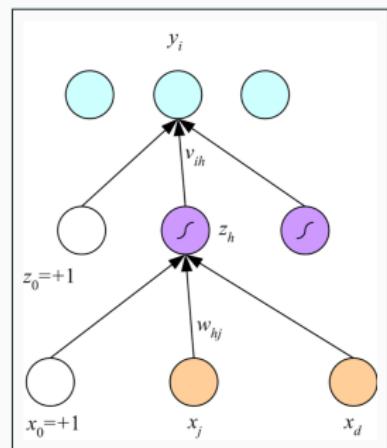
# Stochastic Gradient Descent for MLPs

Now we have all the factors of the derivative of the loss function:

$$\frac{\partial L(r_i, y_i)}{\partial w_{hj}} = \frac{\partial L(r_i, y_i)}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}} = -(r_i - y_i) v_{ih} z_h (1 - z_h) x_j$$

## Interpretation

- $(r_i - y_i)v_h$  can be seen as an error term of hidden unit  $h$
- This error is **backpropagated** from the output layer to the hidden unit
- The larger the weight  $v_h$ , the larger "responsibility" of the error is given to unit  $h$



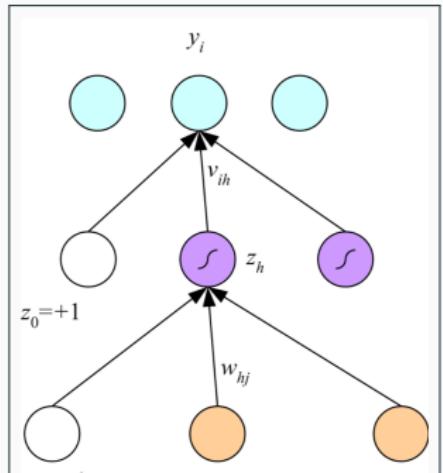
# Stochastic Gradient Descent for MLPs

$$\frac{\partial L(r_i, y_i)}{\partial w_{hj}} = \frac{\partial L(r_i, y_i)}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}} = -(r_i - y_i) v_{ih} z_h (1 - z_h) x_j$$

The update for the weight is a step along the negative gradient

$$\Delta w_{hj} = -\eta \frac{\partial L(r_i, y_i)}{\partial w_{hj}} = \eta (r_i - y_i) v_{ih} z_h (1 - z_h) x_j$$

- Note the update of the hidden layer weight refers to the output layer weight  $v_{ih}$
- We should first update  $w_{hj}$  the hidden layer weights using the old values of  $v_{ih}$ , then update the output layer weights



# Backpropagation training algorithm for two-layer MLP for regression

Initialize all  $w_{hj}, v_{ih}$  randomly to range

$[-0.01, 0.01]$

**repeat**

Draw a training example  $(\mathbf{x}, r)$  at random

**Forward propagation of activation:**

Set  $z_h = \sigma_h(\mathbf{w}_h^T \mathbf{x})$  for  $h = 1, \dots, H$

$$y = \mathbf{v}^T \mathbf{z}$$

**Backpropagation of error:**

$$\Delta \mathbf{v} = \eta(r - y)\mathbf{z}$$

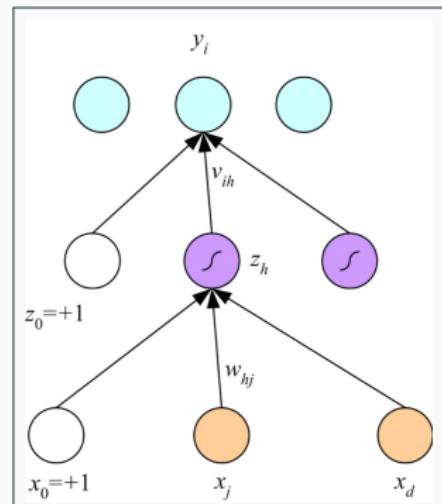
$$\Delta \mathbf{w}_h = \eta(r - y)v_h z_h(1 - z_h)\mathbf{x}, \text{ for } h = 1, \dots, H$$

**Update weights:**

$$\mathbf{v} = \mathbf{v} + \Delta \mathbf{v}$$

$$\mathbf{w}_h = \mathbf{w}_h + \Delta \mathbf{w}_h, \text{ for } h = 1, \dots, H$$

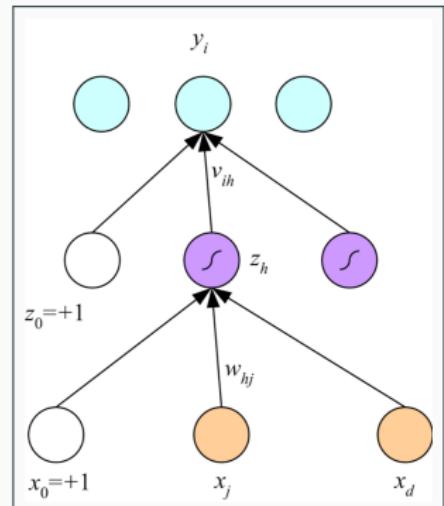
**until** stopping criterion is satisfied



# Backpropagation algorithm for classification tasks

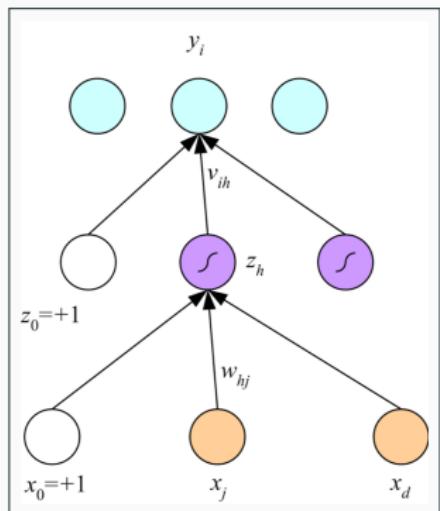
The backpropagation algorithm described can be adapted for classification tasks:

- For binary classification task we change the output activation function to sigmoid function, either logistic (with 0/1 labels) or tanh ( $-1/+1$  labels)
- Multiclass classification can be implemented by using  $K$  output units and applying a softmax-function
$$y_i = \frac{\exp(\mathbf{v}_i^T \mathbf{z})}{\sum_k \exp(\mathbf{v}_k^T \mathbf{z})}$$



# Multiple hidden layers

- Adding hidden layers to the network means that both forward propagation of activation and the backward propagation of error needs to be iterated for more layers
- The error backpropagation then involves a chain-rule over all hidden layers



# Improving convergence

A few simple tricks can be used to speed up convergence:

- Momentum: The SGD update may cause oscillation; subsequent update directions may be very different to each other. This can be helped by computing a running average of the current negative gradient direction and the previous update direction

$$\Delta \mathbf{w}^{(t)} = -\eta \frac{\partial L(r_t, y_t)}{\partial \mathbf{w}} + \alpha \Delta \mathbf{w}^{(t-1)}$$

- Adaptive learning rate: the stepsize  $\eta$  can be changed based on whether error on the training set has been decreasing during the last few passes over the training data (epochs):

$$\Delta \eta = \begin{cases} +a & \text{if } \hat{R}^{(T)} < \frac{1}{p} \sum_{k=1}^p \hat{R}^{(T-k)}, \\ -b\eta & \text{otherwise} \end{cases},$$

where  $\hat{R}^{(t)}$  denotes the average loss over the training data on epoch  $t$

## Using GPUs

---

- The use of Graphical Processing Units (GPU) is widely spread in neural network research
- GPUs can process especially matrix operations (esp. matrix products) very efficiently
- The operations in the backpropagation algorithm can be written so that the majority of computation is in the form of matrix products

# Avoiding overfitting

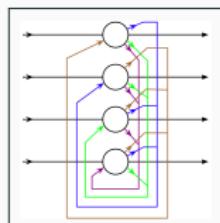
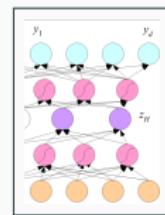
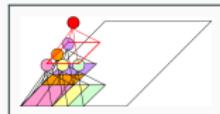
Due to their flexibility neural networks are prone to overfitting. This can be alleviated by certain techniques

- Early stopping: the weights in the network tend to increase during training and gradually overfitting becomes more likely. Stopping training prior convergence can help.
- Dropout: during training, randomly fixing some weights during an update stops the network adjusting to the noise too well. This technique is widely used in current deep learning algorithms

# Other neural network architectures

Particular architectures of neural networks can be used for specific purposes

- **Convolutional Neural Networks** are used e.g. for image input. Instead of fully connected layers, a local neighborhood is cross-connected, but the neighborhoods can overlap
- **Autoencoder networks** have an "hourglass" structure, where a middle hidden layer is much narrower than the input and output layers. This is used for learning new representations for data.
- **Recurrent networks** are used for data that has variable length e.g. speech and natural language



# Summary

- Neural networks are a model family inspired by the human brain
- Multi-layer perceptrons can represent and approximate remarkably complex functions
- Large training data is generally needed to avoid overfitting
- Finding optimal weights for a neural network is generally NP-hard problem
- Variants of stochastic gradient descent are generally used to train neural networks

The Course CS-E4890 - Deep Learning (Spring 2023) is recommended to those who wish to learn more about neural networks

# CS-E4710 Machine Learning: Supervised Methods

## Lecture 9: Ensemble learning

---

Juho Rousu

November 8, 2022

Department of Computer Science  
Aalto University

# Ensemble learning

- Ensemble learning encompasses a wide range of machine learning frameworks which aim to construct high-performance predictive models by combining simpler base models
- Ensemble model generally predicts by computing an average or a majority vote over the base models, possibly weighted in some way
- Many approaches:
  - Boosting (classification task): incrementally add base models which focus on the mistakes made on the previous models by reweighing training examples
  - Bootstrap aggregation aka Bagging (classification or regression tasks): draw random subsamples with replacement from the original training data, train base classifier with each subsample, and combine by voting or averaging.
  - Gradient boosting (regression task): incrementally add base models that focus on the residuals of the prediction error

## Why can model combination work? A thought experiment

Assume the following setup:

- A target concept  $C : X \mapsto \{+1, -1\}$  to be learned
- To predict  $C$ , we have trained a collection of base hypotheses, binary classifiers  $h_1, h_2, \dots, h_L, h_j : X \mapsto \{-1, +1\}$
- We use a majority vote of the hypotheses as the prediction of the ensemble:  $f_{maj}(\mathbf{x}) = \operatorname{argmax}_{y \in \mathcal{Y}} \sum_{j=1}^L \mathbf{1}_{h_j(\mathbf{x})=y}$ , i.e. predict the label that agrees with the majority of the base hypotheses
- When  $L$  is even, both classes may have same number of votes, in that case tie breaking is needed, e.g. by predicting a random label
- This is a so called **majority voting ensemble**

## Why can model combination work? A thought experiment

- Assume further that the base hypotheses have the same true risk  $\epsilon$  but that the probabilities  $P(h_j(\mathbf{x}) \neq C(\mathbf{x}))$  are independent between hypotheses for all  $h_i, h_j$ :

$$\begin{aligned} P(\{h_i(\mathbf{x}) \neq C(\mathbf{x})\} \text{ AND } \{h_j(\mathbf{x}) \neq C(\mathbf{x})\}) \\ = P(h_i(\mathbf{x}) \neq C(\mathbf{x})) \cdot P(h_j(\mathbf{x}) \neq C(\mathbf{x})) = \epsilon^2 \end{aligned}$$

- Not a practical assumption, used here for illustrating a phenomenon
- Now the probability of the majority vote  $h(\mathbf{x})$  to be incorrect, i.e. the true risk  $R(h)$  equals the probability of  $k > L/2$  base hypotheses being incorrect, when  $L$  is odd (no tie-breaking needed)
- This probability is given by the tail of the cumulative binomial probability distribution:

$$R(h) = P(h(\mathbf{x}) \neq C(\mathbf{x})) = \sum_{k=\lceil L/2 \rceil}^L \binom{L}{k} \epsilon^k (1-\epsilon)^{L-k}$$

## Why can model combination work? A thought experiment

Table shows the behaviour of the true risk of the ensemble with different values of  $L$  and  $\epsilon$

$$R(h) = P(h(\mathbf{x}) \neq C(\mathbf{x})) = \sum_{k=\lceil L/2 \rceil}^L \binom{L}{k} \epsilon^k (1-\epsilon)^{L-k}$$

- With low values of  $\epsilon$  the risk goes down quickly to zero
- Even with  $\epsilon = 0.45$  the risk is low with large enough  $L$
- With  $\epsilon = 0.5$  (i.e. random guessing) the risk remains at 0.5 independent of  $L$

$L$	$\epsilon = 0.1$	0.33	0.45	0.5
5	0.0086	0.2050	0.4069	0.5000
11	0.0003	0.1171	0.3669	0.5000
101	0.0000	0.0002	0.1562	0.5000
501	0.0000	0.0000	0.0124	0.5000

## Why can model combination work? A thought experiment

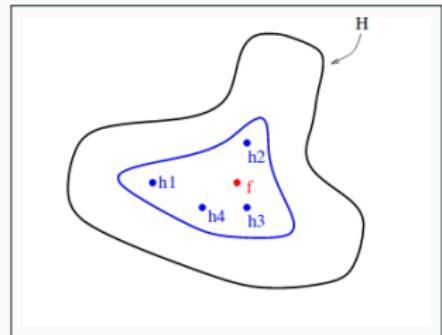
- Even if the base classifiers are **weak**, the combined classifier can be made accurate
- However, the assumption of independent errors is not a realistic one; in practice different classifiers trained on the same or similar data, tend to have correlated errors
- In practice, similar kind of effect can be realized as long as the errors of the classifiers are not perfectly correlated
- A key question in **ensemble learning** is how to obtain **diverse** base classifiers, those that have different error patterns

L	$\epsilon = 0.1$	0.33	0.45	0.5
5	0.0086	0.2050	0.4069	0.5000
11	0.0003	0.1171	0.3669	0.5000
101	0.0000	0.0002	0.1562	0.5000
501	0.0000	0.0000	0.0124	0.5000

# Why can an ensemble work in practice?

Statistical reason (Dietterich, 2000):

- Consider learning as an optimization problem whose goal is to find the best  $h \in \mathcal{H}$  in space of hypotheses  $\mathcal{H}$ .
- When the available data are not sufficient to identify the proper hypothesis, there might be several hypotheses with comparable accuracy.
- Averaging over all these hypotheses can limit the effect of selecting a suboptimal hypothesis.



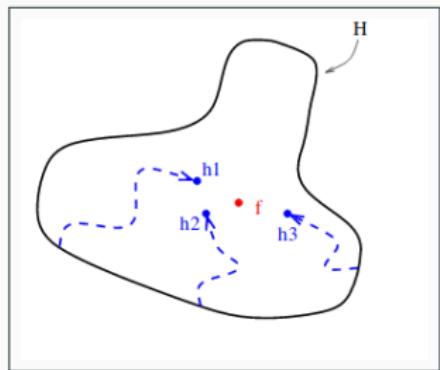
By combining models, we aim to "average out" some of the errors, as in our thought experiment

Dietterich, Thomas G. "Ensemble methods in machine learning." International workshop on multiple classifier systems. Springer, Berlin, Heidelberg, 2000.

# Why can an ensemble work in practice?

Computational reason (Dietterich, 2000):

- The objective function might have several local minima to which the algorithm may converge for computational reasons (e.g. different starting point)
- The different locally optimal hypothesis would often make errors on different examples
- Combining the predictions of such suboptimal learners, can be beneficial to reduce the effect of getting stuck in a single local minimum

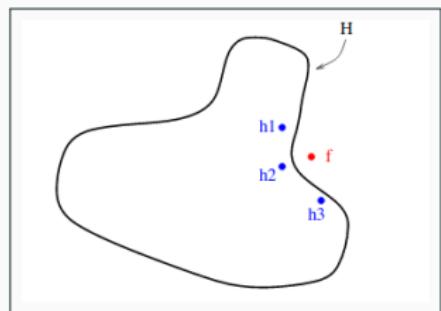


Dietterich, Thomas G. "Ensemble methods in machine learning." International workshop on multiple classifier systems. Springer, Berlin, Heidelberg, 2000.

# Why can an ensemble work in practice?

Representational reason (Diettrich, 2000):

- The representational capacity of a hypothesis space can be extended beyond the search space by averaging over several hypotheses.
- For example: combination of hyperplanes let us represent sets of convex polygons, which is more general than sets of half-spaces, represented by single hyperplanes
- Thus an ensemble can learn patterns outside the original hypothesis class  $\mathcal{H}$ .



Dietterich, Thomas G. "Ensemble methods in machine learning." International workshop on multiple classifier systems. Springer, Berlin, Heidelberg, 2000.

## Diverse learners

- To have an effective ensemble, it should consist of base hypotheses that are **diverse** in the sense that they make errors on different training examples
- Diversity among the hypothesis can arise from different sources:
  - Algorithms: One can combine models of different types (e.g., neural networks, SVM's).
  - Hyperparameters:
    - different numbers of hidden units or layers in a multilayer perceptron
    - different kernel or regularization parameters in support vector machines
  - Input Data:
    - data fusion, where the data from different sources or measurement techniques are integrated (e.g. integrating natural light, infrared and X-ray images in astronomical data)
    - subsampling training data, by choosing random subsets from the examples, or using different input features

# Diversity and model averaging

- Examine and ensemble for a regression task, generated by averaging the predictions of the base models

$$f_{avg}(\mathbf{x}) = \frac{1}{T} \sum_{t=1}^T h_t(\mathbf{x}), h_t(\mathbf{x}) \in \mathbb{R}, t = 1, \dots, T$$

- We focus on its squared loss on a single example  $\mathbf{x}$  that has true label  $y \in \mathbb{R}$ :  $L_{sq}(f_{avg}(\mathbf{x}), y) = (f_{avg}(\mathbf{x}) - y)^2$
- We compare its performance to the average squared loss of the base models

$$\bar{L}_{sq}(\mathbf{x}, y) = \frac{1}{T} \sum_{t=1}^T L_{sq}(h_t(\mathbf{x}), y) = \frac{1}{T} \sum_{t=1}^T (h_t(\mathbf{x}) - y)^2$$

Brown, G. and Kuncheva, L.I., 2010, April. "Good" and "bad" diversity in majority vote ensembles. In International workshop on multiple classifier systems (pp. 124-133). Springer, Berlin, Heidelberg.

# Diversity and model averaging

- The squared loss on the ensemble  $f_{avg}(\mathbf{x})$  on a single example  $\mathbf{x}$  with true label  $y$  satisfies (Brown and Kuncheva, 2010):

$$L_{sq}(f_{avg}(\mathbf{x}), y) = \bar{L}_{sq}(\mathbf{x}, y) - \frac{1}{T} \sum_{t=1}^T (h_t(\mathbf{x}) - f_{avg}(\mathbf{x}))^2$$

- The first term on the right-hand side is the average loss of the base models
- The second term represents the diversity of the base hypotheses in terms of their variance around the ensemble
- As the variance is always non-negative, the ensemble error is always as good as the average error of the base models

Conclusion: for the averaging model using squared loss, diversity among the base models always helps

Brown, G. and Kuncheva, L.I., 2010, April. "Good" and "bad" diversity in majority vote ensembles. In International workshop on multiple classifier systems (pp. 124-133). Springer, Berlin, Heidelberg.

# Diversity and majority voting

- Examine now a classification problems using a majority vote:

$$f_{maj}(\mathbf{x}) = \operatorname{argmax}_y \sum_{t=1}^T \mathbf{1}_{h_t(\mathbf{x})=y}$$

- We focus on the zero-one loss of the ensemble on  $\mathbf{x}$  with true label  $y \in \{-1, +1\}$ :

$$L_{0/1}(f_{maj}(\mathbf{x}), y) = \mathbf{1}_{f_{maj}(\mathbf{x}) \neq y}$$

- We compare its loss to the average zero-one loss of the base models:

$$\bar{L}_{0/1}(\mathbf{x}, y) = \frac{1}{T} \sum_{t=1}^T L_{0/1}(f_t(\mathbf{x}), y)$$

Brown, G. and Kuncheva, L.I., 2010, April. "Good" and "bad" diversity in majority vote ensembles. In International workshop on multiple classifier systems (pp. 124-133). Springer, Berlin, Heidelberg.

## Diversity and majority voting

- Brown and Kuncheva (2010) showed that the loss of the majority voting ensemble can be expressed as:

$$L_{0/1}(f_{maj}(x), y) = \bar{L}_{0/1}(\mathbf{x}) - y f_{maj}(\mathbf{x}) \frac{1}{T} \sum_{t=1}^T \mathbf{1}_{h_t(\mathbf{x}) \neq f_{maj}(\mathbf{x})}$$

- The first term is the average zero-one error of the base models
- The second term accounts for the disagreements between the ensemble and the base models:
  - When the ensemble is correct ( $y = f_{maj}(\mathbf{x})$ ) disagreements reduce the majority vote error
  - When the ensemble is incorrect ( $y \neq f_{maj}(\mathbf{x})$ ) disagreements increase the majority vote error

Conclusion: For majority voting ensembles, diversity among the base models has context-dependent effects on the accuracy

## Boosting

---

# Boosting

- Boosting originally was proposed as an answer to a question raised in Probably Approximately Correct (PAC) theory: Can **weak learners**, whose accuracy is only slightly better than random guessing, can be combined into a strong PAC learner?
- This question was answered positively by Rob Shapire in 1990, a theoretical finding that led to the development of a very practical AdaBoost (Adaptive Boosting) algorithm in 1996, and to the prestigious Gödel prize to be awarded for Schapire and Yoav Freund in 2003.
- AdaBoost creates diversity by re-weighting the training examples during the algorithm, and conducts a weighted majority vote to predict.

## Weak learning

A concept class  $C$  is **weakly PAC-learnable** if there exists an algorithm  $A$ ,  $\gamma > 0$ , such that

- for all  $\delta > 0$  for all  $c \in C$  and all distributions  $D$ ,

$$P_{S \sim D}(R(h_S) \leq \frac{1}{2} - \gamma) \geq 1 - \delta$$

- for a sample  $S$  of size  $m$  which is polynomial in  $1/\delta$

Note: The definition differs from the PAC learnability by only requiring the true risk to be slightly less than random, with high confidence

## Boosting theorem

Boosting Theorem (Schapire, 1990): A concept class  $C$  is weakly PAC-learnable if and only if it is (strongly) PAC-learnable.

- This surprising result implies that learning is an all or nothing phenomenon: if we can find an algorithm that achieves a low level of accuracy in learning  $C$ , then there exists an algorithm that can do the same with a high level of accuracy.
- AdaBoost algorithm described next is a practical algorithm that achieves a (strong) PAC learner by linear combinations of weak learners

Schapire, R.E., 1990. The strength of weak learnability. Machine learning, 5(2), pp.197-227.

# AdaBoost

AdaBoost algorithm iteratively combines weak learners to arrive at a strong classifier in the PAC sense

Inputs:

- A labeled training sample  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ ,  $\mathbf{x}_i \in X$ ,  $y_i \in \{-1, +1\}$
- A hypothesis class  $H$  where base hypotheses  $h_t$ ,  $t = 1, \dots, T$  are drawn
- A distribution  $D_t$ ,  $t = 1, \dots, T$ , that assigns a weight  $D_t(i)$  for the  $i$ 'th training example for the  $t$ 'th weak learner  $h_t$

Output: combined model as a non-negative ( $\alpha_t \geq 0$ ) combination of the weak learners

$$f_T(\mathbf{x}) = \sum_{t=1}^T \alpha_t h_t(\mathbf{x})$$

Note the loose terminology: a weak learner is actually the algorithm outputting the base model, but in boosting literature the base models  $h_t$  are also called weak learners

# AdaBoost

- In each round  $t$ , the algorithm adds a new weak learner  $h_t$ , one minimizes the empirical error on a sample drawn from distribution  $D_t$  (same as empirical errors weighted by  $D_t$ ):

$$\epsilon_t = \min_{h \in H} P_{i \sim D_t}(h(\mathbf{x}_i) \neq y_i) = \min_{h \in H} \sum_{i=1}^m D_t(i) \mathbf{1}_{h(\mathbf{x}_i) \neq y_i}$$

- The weak learner  $h_t$  is weighted by

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

which can be seen as log-odds probability of the weak learner being correct. This value comes from minimization of a bound for the empirical risk.

- If  $\epsilon_t < 1/2$  then  $\frac{1 - \epsilon_t}{\epsilon_t} > 1$  and  $\alpha_t > 0$  thus weak learners that are more accurate on training data than random guessing receive positive weights.

For the round  $t + 1$ , the weights on the training examples are re-weighted:

$$D_{t+1}(i) = D_t(i) \cdot \frac{e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{Z_t}$$

- The term  $y_i \alpha_t h_t(\mathbf{x})$  can be seen as a margin of example (weighted by  $\alpha_t$ )
- Feeding the margin through an negative exponential causes an exponential up-weighting of misclassified examples (those with negative margin) and down-weighting of correctly classified examples (positive margin)
- The factor:  $Z_t = \sum_{i=1}^m D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)} = 2\sqrt{\epsilon_t(1 - \epsilon_t)}$  is a normalization factor ensuring that  $D_t$  sums up to 1.

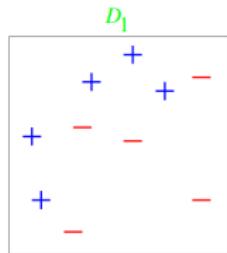
The sequence of the re-weighting is that the weak learners for subsequent rounds will focus on the examples that were misclassified  $\Rightarrow$  increases diversity

## AdaBoost pseudo-code

```
ADABoost( $S = ((x_1, y_1), \dots, (x_m, y_m))$ )
1  for  $i \leftarrow 1$  to  $m$  do
2       $\mathcal{D}_1(i) \leftarrow \frac{1}{m}$ 
3  for  $t \leftarrow 1$  to  $T$  do
4       $h_t \leftarrow$  base classifier in  $\mathcal{H}$  with small error  $\epsilon_t = \mathbb{P}_{i \sim \mathcal{D}_t} [h_t(x_i) \neq y_i]$ 
5       $\alpha_t \leftarrow \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$ 
6       $Z_t \leftarrow 2[\epsilon_t(1 - \epsilon_t)]^{\frac{1}{2}}$      $\triangleright$  normalization factor
7      for  $i \leftarrow 1$  to  $m$  do
8           $\mathcal{D}_{t+1}(i) \leftarrow \frac{\mathcal{D}_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ 
9   $f \leftarrow \sum_{t=1}^T \alpha_t h_t$ 
10 return  $f$ 
```

# AdaBoost example

## Toy Example

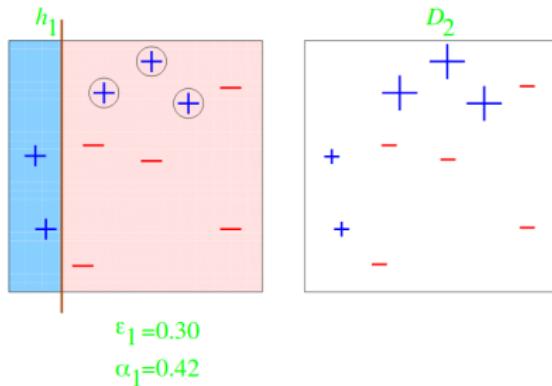


weak classifiers = vertical or horizontal half-planes

source : <https://www.csie.ntu.edu.tw/~mhyang/course/u0030/papers/schapire.pdf>

# AdaBoost example

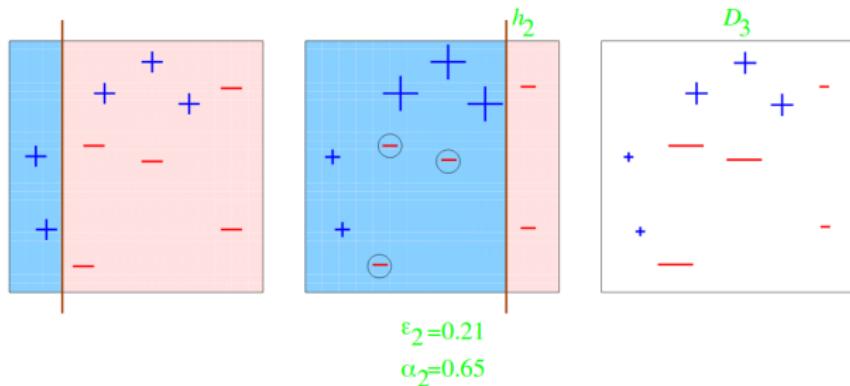
## Round 1



source: <https://www.csie.ntu.edu.tw/~mhyang/course/u0030/papers/schapire.pdf>

# AdaBoost example

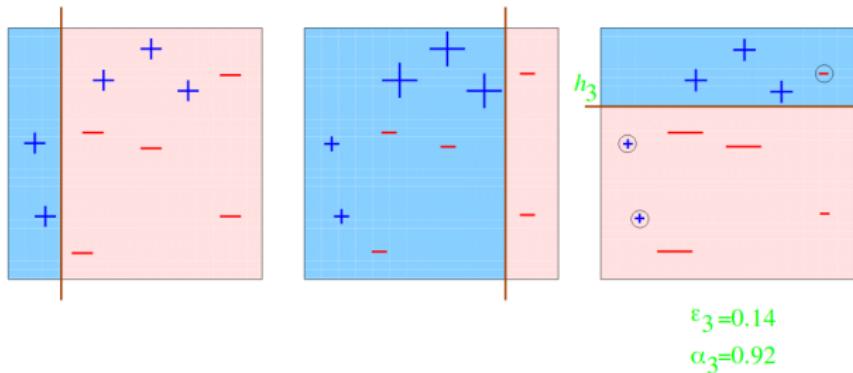
## Round 2



source: <https://www.csie.ntu.edu.tw/~mhyang/course/u0030/papers/schapire.pdf>

# AdaBoost example

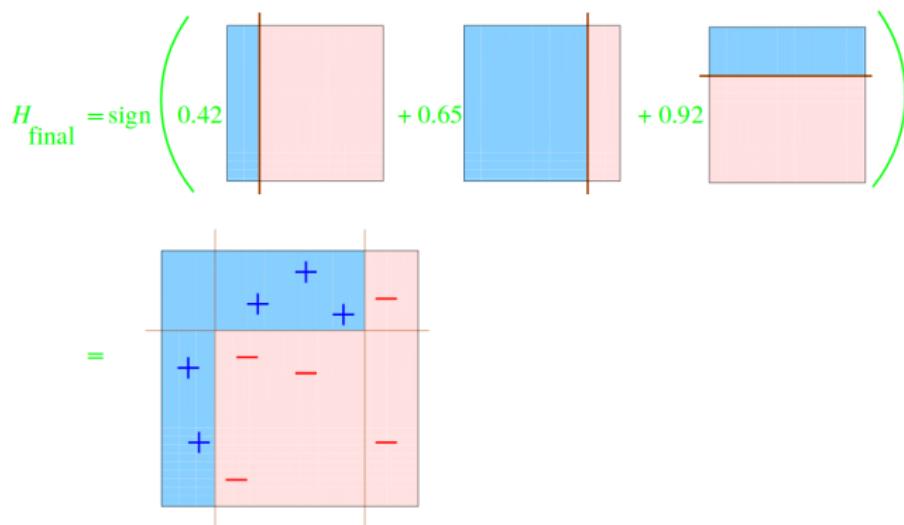
## Round 3



source: <https://www.csie.ntu.edu.tw/~mhyang/course/u0030/papers/schapire.pdf>

# AdaBoost example

## Final Classifier



source: <https://www.csie.ntu.edu.tw/~mhyang/course/u0030/papers/schapire.pdf>

## Empirical error of AdaBoost

Theorem: The empirical error of the classifier  $f$  returned by AdaBoost verifies:

$$\hat{R}_S(f) \leq \exp\left(-2 \sum_{t=1}^T \left(\frac{1}{2} - \epsilon_t\right)^2\right)$$

Furthermore, if for all  $t$ ,  $\gamma \leq (\frac{1}{2} - \epsilon_t)$ , then

$$\hat{R}_S(f) \leq \exp(-2\gamma^2 T)$$

Proof: See Mohri book

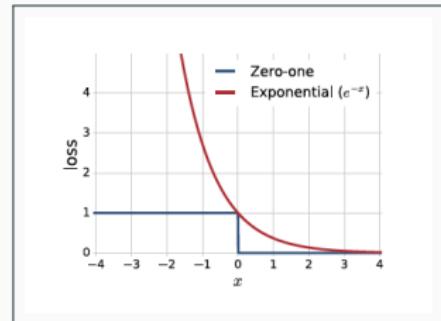
- The empirical error goes down exponentially fast in  $T$ : given enough weak learners, empirical error can be pushed arbitrarily low
- Above  $\gamma$  is the "edge", the amount by which the weak learners are more accurate than random guessing: larger  $\gamma$  results in tighter bound for the empirical error

# The loss function optimized by AdaBoost

It can be shown that AdaBoost is minimizing an upper bound on the zero-one loss, given by the exponential function

$$F_{\text{exp}}(\alpha) = \sum_{i=1}^m e^{-y_i f_T(x_i)} = \sum_{i=1}^m e^{-y_i \sum_{t=1}^T \alpha_t h_t(x_i)}$$

- Here the variables to be optimized are the weights  $\alpha = (\alpha_t)$ , and the collection of base learners are taken as fixed
- Exponential loss penalizes misclassified examples heavily, which reveals a potential weakness of AdaBoost: if labels are noisy, AdaBoost will give high weight on noisy labels with the risk of overfitting



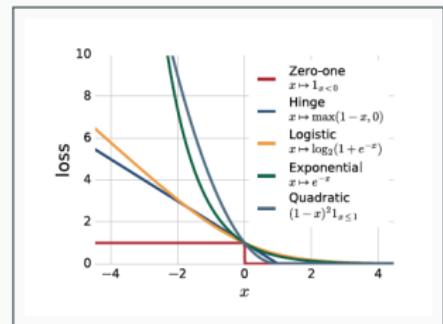
# The loss function optimized by AdaBoost

The viewpoint of loss function optimization suggest alternative boosting algorithms by changing the loss function

- For example, LogitBoost (Friedman et al. 2000) minimizes the logistic loss:

$$F_{\text{Logit}}(\alpha) = \sum_{i=1}^m \log(1 + e^{-y_i \sum_{t=1}^T \alpha_t h_t(x_i)})$$

- LogitBoost is closely related logistic regression (LR): if we consider the weak learners  $h_t$  as the input features and  $\alpha_t$  as the feature weights, LogitBoost is essentially learning a logistic regression model (except for some constants that differ from LR).



Friedman, J., Hastie, T., Tibshirani, R. (2000). "Additive logistic regression: a statistical view of boosting". Annals of Statistics. 28 (2):

## VC-dimension of AdaBoost

- The hypothesis class of AdaBoost after  $T$  rounds is given by

$$\mathcal{F}_T = \left\{ \operatorname{sgn}\left(\sum_{t=1}^T \alpha_t h_t\right) : \alpha_t > 0, h_t \in H \right\}$$

- The VC-dimension of  $\mathcal{F}_T$  can be bounded in terms of the VC-dimension  $d$  of the weak learners:

$$VCdim(\mathcal{F}_T) \leq 2(d+1)(T+1)\log_2((T+1)e)$$

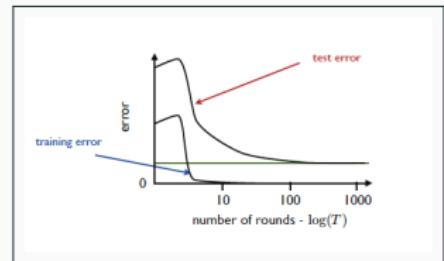
which grows as  $O(dT \log T)$

- This suggests that AdaBoost could overfit when  $T$  grows large, however empirically it has been observed not to be the case

# AdaBoost and generalization

Empirically AdaBoost has been observed not to overfit with large  $T$

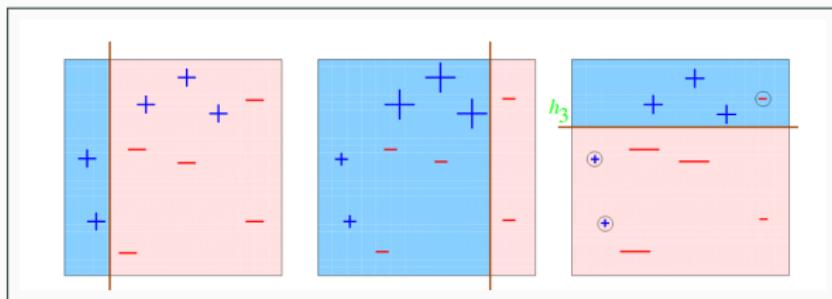
- Instead, the test error continues to go down with increasing  $T$  even after the empirical error reaches zero
- This phenomenon has been explained through margin-based analysis of AdaBoost:



- AdaBoost can be shown to increase the margins of training examples as  $T$  increases, which leads to better generalization
- However, AdaBoost is not strictly speaking optimizing the margins, but can achieve substantial fraction of the maximum margin in suitable conditions
- Boosting algorithms that aim to maximize margins have been proposed, however, they do not empirically outperform AdaBoost

## Boosting in practice

- Boosting requires having an access to a weak learner, it is hard to guarantee formally in real world situations
- However, empirically "reasonable" classifiers generally outperform random guessing on a real world data
- In general a weak learner that is unstable, that is, the chosen hypothesis changes upon small modification of data, works the best (due to introducing diversity to the ensemble)
- In practice, axis-paraller hyperplanes, also called "decision stumps" work well



## **Other ensemble learning schemes**

---

# Bootstrap Aggregating (Bagging)

---

- Bootstrap aggregating (Bagging) is an ensemble method in which the weak learners are built using  $T$  different bootstrap samples of the original training data.
- Bootstrap sampling : drawing randomly with replacement from the original training set (resampling with replacement)  $m$  instances, where  $m$  is the size of the training data.
- The bootstrap samples are different from each other, which creates diversity
- For classification, the ensemble prediction is given by majority voting
- **Random forest** is an effective ensemble learning methods that combines randomized decision trees with bagging

# Gradient boosting

- Gradient boosting (GB) (Friedman, 2011) is an ensemble method based on iteratively growing the ensemble according to the gradient of the loss function

$$f_t(\mathbf{x}) = f_{t-1}(\mathbf{x}) - \eta_t \sum_{i=1}^m \nabla_{f_{t-1}} L(y_i, f_{t-1}(\mathbf{x}_i))$$

- The partial derivatives composing the negative gradient  $-\nabla_{f_{t-1}}$  are called the pseudo-residuals

$$r_{it} = -\frac{\partial L(y_i, f_{t-1}(\mathbf{x}))}{\partial f_{t-1}}$$

- In each iteration, a weak learner  $h_t$  is trained to predict the pseudo-residuals  $r_{it}$  and added to the ensemble: with a step-size  $\eta_t$  that is optimized for maximal descent

Friedman, J. (2001). Greedy boosting approximation: a gradient boosting machine. Ann. Stat. 29, 1189–1232. doi: 10.1214/aos/1013203451

# Summary

- Ensemble methods are based on the idea of generating strong predictive models by combining simpler, potentially weaker models (weak learners)
- Diversity of the weak learners is the key for obtaining an accurate ensemble
- Several mechanisms exist for obtaining diversity
- Boosting is a successful ensemble method that is based on adaptively reweighting training examples

# CS-E4710 Machine Learning: Supervised Methods

Lecture 10: Feature engineering and selection

---

Juho Rousu

November 15, 2022

Department of Computer Science  
Aalto University

# The importance of input representations

- So far in this course, we have (mostly) assumed the representation of the input data as feature vectors in  $\mathbb{R}^d$
- But how do we come up with such representation?
- Is the given representation the best possible?
- It can be argued that good data representation is actually more important than the choice of the learning algorithm
- The importance of good data representations is exemplified by the fact there is nowadays a international conference on the topic (<https://iclr.cc>) with thousands of participants.

# Feature engineering techniques

Variety of techniques:

- Feature transformation - convert the features in a form that allows learning better
- Feature selection - aim to reduce the number of input variables that are used by the predictor
- Feature generation - build new features by combining the original ones either manually (using prior knowledge) or by learning representations by optimizing some objective

Note: we used the term "variable" and "feature" interchangeably

## Feature transformations

---

## Feature transformations

- Simple transformations of the variables may have a large effect on the models
- One can obtain better error rates or faster optimization by transforming the variables
- Feature transformations are generally performed to make the distribution of the input variables to better represent the domain knowledge or to make the data more suitable to the learning algorithm
- However, feature transformation is largely art rather than science - it is hard to give any guarantees of the success of transforming the data

## Examples of feature transformations

Let  $\mathbf{f} = (f_1, \dots, f_m)$  denote the values of a single variable in the dataset

- Centering:  $f'_i = f_i - \bar{f}$  where  $\bar{f} = \frac{1}{m} \sum_i f_i$  is the mean of variable - Rationale: learning algorithms generally learn from the variance of the data, not the mean. Centering makes the variance more obvious.
- Standardization:  $f'_i = \frac{f_i - \bar{f}}{\sqrt{\text{var}(f)}}$ , where  $\text{var}(f) = \frac{1}{m} \sum_i (f_i - \bar{f})^2$  is the variance of variable. Rationale: making all variables have zero mean and unit variance may help if the raw variables have very different scales
- Unit range:  $f' = \frac{f_i - f_{\min}}{f_{\max} - f_{\min}}$ , where  $f_{\min}$  and  $f_{\max}$  are the minimum and maximum values for the variable. Rationale: useful if the variable's relative position in the observed range is important
- Clipping:  $f'_i = \text{sgn}(f_i) \min(b, |f_i|)$ , for some threshold  $b > 0$ . Rationale: if certain large values are known to be non-informative, clipping may make learning easier (clipping small values: use max instead of min)

## Examples of feature transformations

- Logarithmic transformation:  $f'_i = \log(b + f_i)$ , for some user-defined constant  $b > 0$ . Rationale: if small differences between small values (e.g. 0 vs. 1) are more important than small differences between large values (1000 vs. 1001), log-transform can be used to emphasize the former.
- Sigmoid transformation:  $f'_i = \frac{1}{1+\exp(bf_i)}$ . Compresses the high absolute values heavily, "soft version" of clipping.
- Normalization of feature vectors:  $\mathbf{x}'_i = \frac{\mathbf{x}_i}{\|\mathbf{x}_i\|}$  where  $\mathbf{x}_i$  is a single training example. Rationale: useful if the relative values of the variables for a single example are important rather than the absolute values, e.g. if large object produces large average values for all features but the class of the object does not depend on its size

In general, several transformations are used to establish a desired effect,  
e.g. log-transform + normalization

## Feature selection

---

# Potential benefits of feature selection

- Facilitating data visualization and interpretation: a model with fewer features is easier to explain
- Reducing time and space requirements of training models: less data to store and compute over
- Improving the predictive performance: a small subset of variables is less likely to overfit

Guyon, I. and Elisseeff, A., 2003. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar), pp.1157-1182.

## Feature selection by exhaustive search

Given a set of  $d$  features, an idealized approach to feature selection would be to

- Exhaustively go through all  $2^d - 1$  feature subsets
- Train a model using each subset
- Select the feature subset that gives the best predictive accuracy (e.g. by cross-validation)

While this approach would give us the optimal feature subset, it is prohibitively expensive unless  $d$  is very small

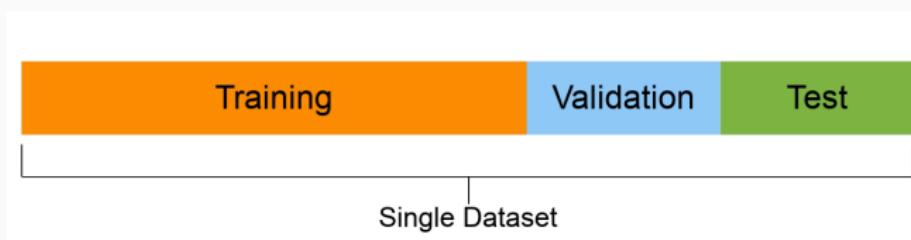
## Feature selection in practice

In general, feature selection approach aim to avoid the exponential complexity of checking each feature subset

- Variable ranking: assess the usefulness of each input feature **individually** in a preprocessing step prior to learning, select a subset of most useful features
- Variable subset selection: generate several different **feature subsets** generally by some greedy search strategy, train a model with each subset, select the subset with the best predictive performance
- Embedded methods: the learning algorithm performs variable selection

# Feature selection and overfitting

- Feature selection heavily affects the underlying hypothesis class (e.g. VC dimension)
  - It can be seen as a form of model selection (c.f. Lecture 4)
- To avoid the risk of overfitting, any feature selection that relies on the output labels, should be performed using validation data separated from the training set
- If an estimate of the final model's accuracy after feature selection is needed (in addition to the selected features), a separate test set is needed for that purpose.



## Variable ranking

---

# Variable ranking approach

A generic variable ranking procedure:

- Compute a score  $s_j$  for each input variable  $j$  using some **scoring criterion**
- Sort the variables in descending order of  $s_j$
- Select a subset of the most highly ranking variables, e.g. by top  $k$  variables or all variables exceeding a score threshold  $\theta$  ( $k$  or  $\theta$  generally decided by the user)

This approach is often called the "filtering" approach, due to the non-selected variables being "filtered out"



Image By Lucien Mousin - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=37776286>

## Regression-based scoring criterion : Pearson correlation

- Assume a dataset  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ , where  $\mathbf{x}_i = (x_{i1}, \dots, x_{id})^T$ , and  $y_i \in \mathbb{R}$
- (Empirical) Pearson correlation of the  $j$ 'th feature and the output is given by

$$r_j = \frac{\sum_{i=1}^m (x_{ij} - \bar{x}_j)(y_i - \bar{y})}{\sqrt{\sum_i (x_{ij} - \bar{x}_j)^2} \sqrt{\sum_i y_i - \bar{y}}^2}$$

- Above,  $\bar{x}_j = \frac{1}{m} \sum_{i=1}^m x_{ij}$  denotes the mean of the  $j$ 'th feature in the dataset and  $\bar{y} = \frac{1}{m} \sum_{i=1}^m y_i$  denotes the mean of the output variable
- $r_j$  ranges from  $+1$  (perfect correlation) to  $-1$  (perfect anti-correlation)
- For feature scoring, we use  $s_j = r_j^2$  to allow selection of both anti-correlated and correlated features

## Regression-based scoring criterion : Pearson correlation

- Pearson correlation has a natural interpretation in terms of linear regression
- $r_j$  is the optimal regression co-efficient in a univariate linear model

$$y = rx + b$$

that has the smallest mean squared error on the data

$$r_j = \operatorname{argmin}_{r,b \in \mathbb{R}} \frac{1}{m} \sum_{i=1}^m (rx_{ij} + b - y_i)^2$$

- $r_j^2$  is the fraction of the variance of the output variable explained by the linear model
- $r_j^2 = 0$  means that the  $j$ 'th feature **alone** does not explain any of the variance of the output

## Classification-based scoring criteria

- Alternatively, one can use univariate classification based scoring criteria
- Given  $y \in \{-1, +1\}$ , a simple approach of to consider a model

$$y = \text{sgn}(ax + \theta)$$

where  $a \in \{-1, +1\}$  and  $\theta \in \mathbb{R}$  is set to optimize the empirical error

$$[a_j, \theta_j] = \operatorname{argmin}_{a \in \{-1, +1\}, \theta \in \mathbb{R}} \sum_{i=1}^m \mathbf{1}_{\text{sgn}(ax_{ij} + \theta) \neq y_i}$$

- The above is essentially the "decision stump" we used in the AdaBoost example (Lecture 9)
- For feature scoring, we can use the accuracy:  
$$r_j = 1 - \frac{1}{m} \sum_{i=1}^m \mathbf{1}_{\text{sgn}(a_j x_{ij} + \theta_j) \neq y_i}$$
- Also can use other evaluation metrics for classification (Lecture 1)

## Pros and cons of the filtering approach

- The filtering approach is reasonably efficient:
  - computation of the feature scores can be done in  $O(md)$  for correlation and  $O(dm \log_2 m)$  time for the decision stump (the dominating cost is sorting the values of the feature  $j$  in  $O(m \log_2 m)$  time).
  - The ranking of the features given the scores also requires sorting the features in  $O(d \log d)$  time.
- However, the method is limited by two issues:
  - Features that are not correlating with the output may still be useful when combined by other variables
  - As the filtering is made independently of the predictive model that will use the selected features, the selected subset might not be optimal

## Example 1: Useless variables may be useful in combination of other variables

In the figure, we have toy dataset of two input variables and binary output (open and closed circles)

- The data lies in four clusters in a "XOR-like" layout
- The marginal distribution of neither input variable (top left and bottom right) give any separation of the classes
- Combination of the two variables can perfectly separate the classes (e.g. by intersection of two hyperplanes)

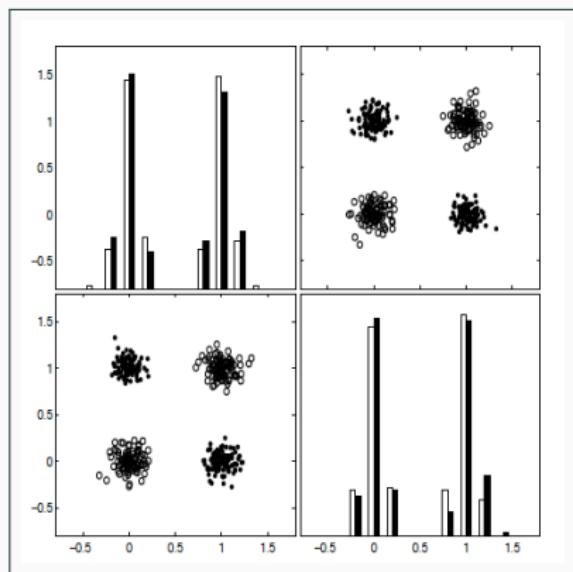


Figure : Guyon and Elisseeff, 2003. Top let and bottom right pane show the distribution of the data along the ranges two features, respectively. The top right and bottom left panes show the same data, with the coordinate axes swapped

## Example 2: Useless variables may be useful in combination of other variables

In the second example the two classes have high within class covariance in the two input variables, which gives the elongated clusters

- One of the variables alone does not give any separation of the classes (top left)
- The other variable gives partial separation, but with some overlap (bottom right)
- However, using both variables gives perfect separation in 2D plane (e.g. by a single hyperplane)

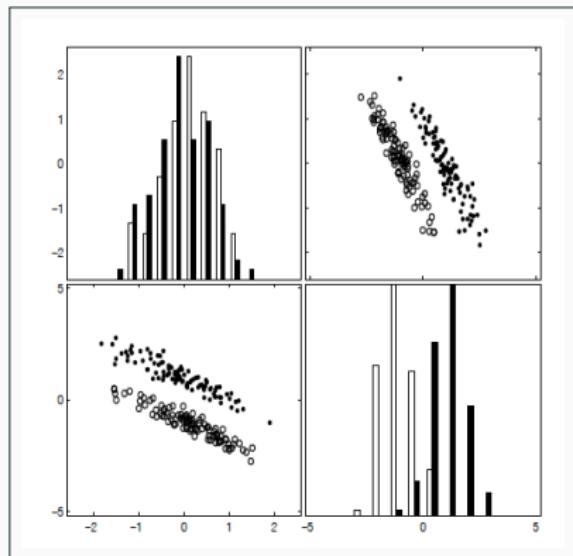


Figure : Guyon and Elisseeff, 2003. Top left and bottom right pane show the distribution of the data along the ranges two features, respectively. The top right and bottom left panes show the same data, with the coordinate axes swapped

## **Variable subset selection**

---

# Wrapper approach

- The wrapper approach to variable selection uses the same learning algorithm that is used for the final model to evaluate variable subsets
- It iterates two steps:
  - Generate a variable subset (by some fixed procedure)
  - Evaluate the subset by training a model with the subset

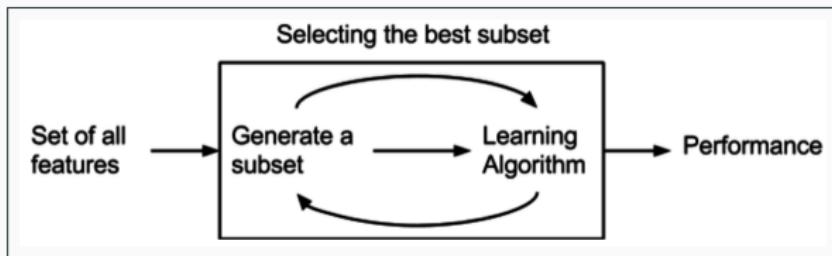


Image By Lastdreamer7591 - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=37208688>

# Wrapper approach

- Two generic procedures for generating a subset:
  - Forward selection - grow the set of selected variables by iteratively
  - Backward elimination - start from set of all variables and iteratively eliminate variables until a given stopping criterion is fulfilled
  - Also more thorough search strategies can be used (best-first, branch-and-bound, etc.)

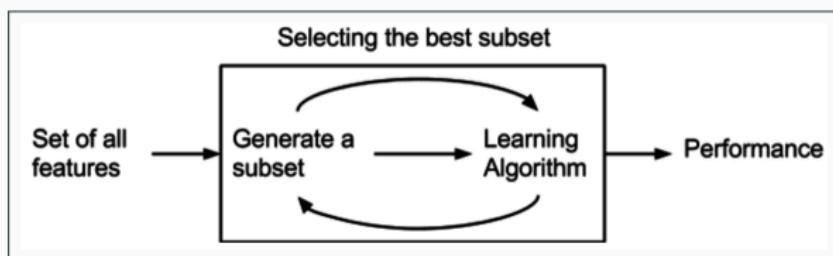


Image By Lastdreamer7591 - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=37208688>

# Variable scoring in wrapper approach

- In a wrapper algorithm it is natural to use the risk of the hypothesis (either on training or validation data) as the variable scoring criterion
- This involves training and testing a hypothesis for each variable subset considered
  - Computationally heavier than the filter approach
  - But can find better variable subsets than the filter approach

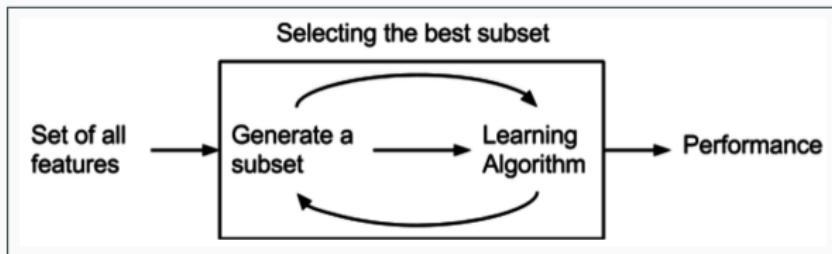


Image By Lastdreamer7591 - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=37208688>

## Greedy forward selection

- In greedy forward selection, we add variables iteratively, choosing in each step a variable that gives the maximum improvement
- The variables that have been chosen to the model are not removed, even if they become redundant (no back-tracking)
- For each iteration, models are trained for every candidate variable to be added
- In total  $O(Kd)$  models are trained where  $K$  is the upper bound for the number of variables selected

## Greedy forward selection pseudo-code

**Input:** Dataset  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$

**Input:** Maximum number of variables  $K$

**Output:** A subset of selected variables  $J \subset \{1, \dots, d\}$

Initialize  $J \leftarrow \emptyset, s_J \leftarrow 0; k \leftarrow 0$

**repeat**

**for**  $j \in \{1, \dots, d\} \setminus J$  **do**

        Train a model  $f_{J \cup j}$  with input variables  $J \cup j$

$s_{J \cup j} \leftarrow$  accuracy of the model  $f_{J \cup j}$  (training set or validation set)

**end for**

    Select the best variable:  $j^* \leftarrow \operatorname{argmax}_{j \in \{1, \dots, d\} \setminus J} s_{J \cup j}$

**if**  $s_{J \cup j^*} > s_J$  **then**

$J \leftarrow J \cup j^*; k \leftarrow k + 1$

**else**

$stop \leftarrow \text{TRUE}$

**end if**

**until**  $k = K$  or  $stop$

## Backward elimination

---

- In backward elimination, one starts from the full set of input variables
- It is argued that backward elimination is more effective in finding good variable subsets than forward selection
- In each stage, the input variable whose elimination has the best effect on the model is eliminated
  - Largest increase or smallest decrease of accuracy on validation set
- Accuracy on validation set generally used, since it will generally have an optimum for some number of selected variables
- In total  $O(d^2)$  models are trained

## Backward elimination pseudo-code

**Input:** Training set  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ , validation set  $V = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$

**Output:** A subset of selected variables  $J_{max} \subset \{1, \dots, d\}$

Initialize  $J \leftarrow \{1, \dots, d\}$ ;  $s_{max} = 0$ ;  $J_{max} = J$

**repeat**

**for**  $j \in J$  **do**

    Train a model  $f_{J \setminus j}$  with input variables excluding  $j$

$s_{J \setminus j} \leftarrow$  accuracy of the model  $f_{J \setminus j}$  on the validation set

**end for**

Variable whose elimination gives the best model:

$j^* \leftarrow \operatorname{argmax}_{j \in J} s_{J \setminus j}$

$J \leftarrow J \setminus j^*$

Keep track of the best model:

**if**  $s_J > s_{max}$  **then**

$s_{max} = s_J$ ;  $J_{max} = J$

**end if**

**until**  $J = \emptyset$

# The case for backward selection

A simple example demonstrates a case where forward selection may miss the best subset

- Variable 3 separates the two classes best when used as a single variable: forward selection would likely choose Variable 3
- However, variables 1 and 2 separate the classes perfectly as a pair: backward selection is likely to remove variable 3, since its redundant when variables 1 and 2 are in the subset.

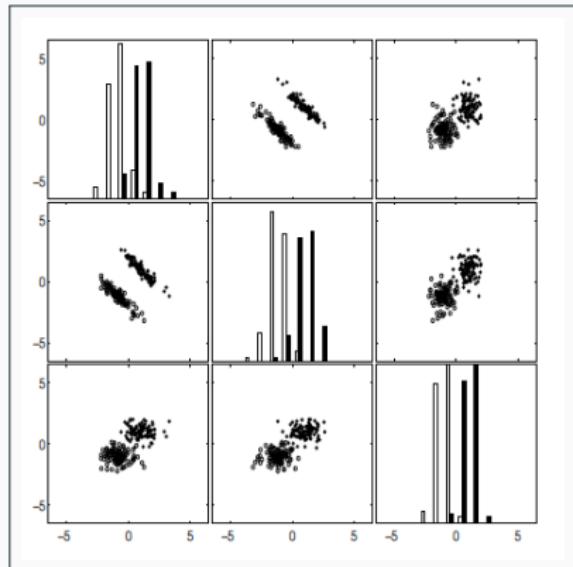
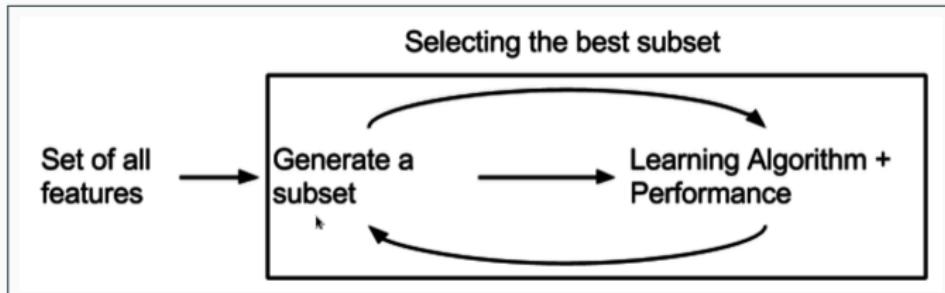


Figure : Guyon and Elisseeff, 2003. The diagonal panes show the histograms of the class distributions for each variable. The off diagonal blocks show the scatter plots of the data using each pair of the variables

# Embedded feature selection methods

Embedded feature selection algorithms refer to cases where the learning algorithm itself is selecting features as part of searching for the best model

- Models based on logical tests on single variable values e.g. decision trees
- Boosting algorithms using base hypotheses on single variables
  - AdaBoost on decision stumps can be seen as an embedded feature selection method
- Sparse modelling methods: focused on penalizing the weights with **sparsity inducing norms**



## Sparse modelling

---

## Sparsity-inducing norms: : $\ell_0$

- Consider the problem of minimizing the empirical risk of a linear model subject to a budget of  $K$  variables

$$\min_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^m L(\mathbf{w}^T \mathbf{x}_i, y_i) \text{ s.t. } |\{j | w_j \neq 0\}| \leq K$$

where  $\mathbf{w}^T \mathbf{x}$  is a model with the weight vector  $\mathbf{w}$

- The size of the set  $\{j | w_j \neq 0\}$  corresponds to the  $\ell_0$  norm of  $\mathbf{w}$ :

$$\|\mathbf{w}\|_0 = |\{j | w_j \neq 0\}|$$

- With small  $K$ , the optimal solutions to the above problem are sparse (with at most  $K$  non-zero weights)

## Sparsity-inducing norms: $\ell_0$

---

- Alternatively, the optimization problem can be written as a regularized learning problem

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^m L(\mathbf{w}^T \mathbf{x}_i, y_i) + \lambda \|\mathbf{w}\|_0$$

- if  $\lambda > 0$  is sufficiently large, the optimal solution will be a sparse containing a small number of non-zero weights
- However, both the constraint version (previous slide) and the above penalization version are computationally hard to solve (non-convex, NP-hard)

## Sparsity-inducing norms: $\ell_1$

---

- Replacing the non-convex  $\|\mathbf{w}\|_0$  by the  $\ell_1$  norm

$$\|\mathbf{w}\|_1 = \sum_{j=1}^d |w_j|$$

gives a convex objective (assuming the loss  $L$  is also convex):

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^m L(\mathbf{w}^T \mathbf{x}_i, y_i) + \lambda \|\mathbf{w}\|_1$$

- It turns out that the above  $\ell_1$  regularized problem often returns sparse solutions
- But what is so special about  $\ell_1$  norm?

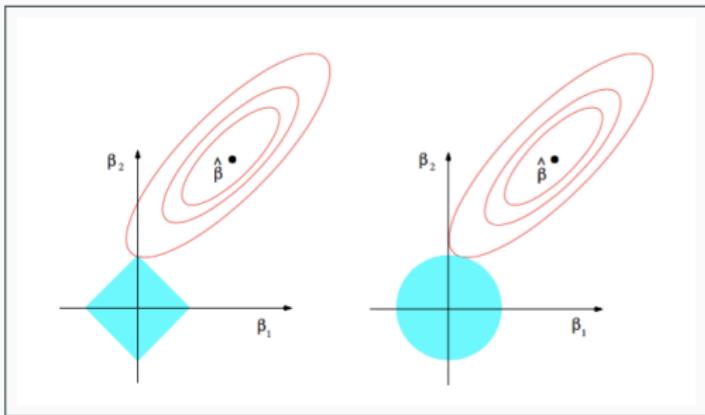
## Sparsity-inducing norms: $\ell_1$

Consider a simple example of minimizing mean squared error (MSE) on two input variables

$$\hat{\beta} = \operatorname{argmin}_{\beta_1, \beta_2} \frac{1}{m} \sum_{i=1}^m (\beta_1 x_{1i} + \beta_2 x_{2i} - y_i)^2 + \lambda \|\beta\|_p$$

where  $p = 1, 2$

- The turquoise diamond ( $p = 1$ ) and ball ( $p = 2$ ) denote the constraint regions  $\|\beta\|_p \leq 1$
- The co-centric ellipses denote the level sets of equal MSE



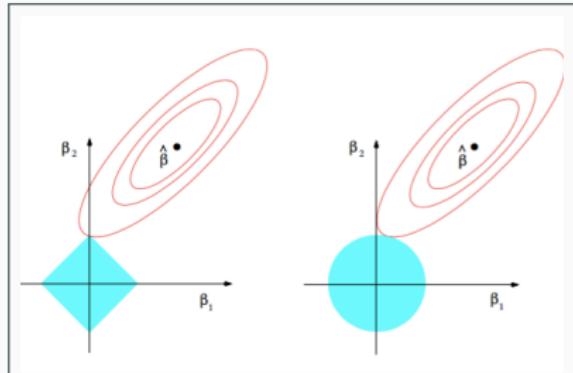
## Sparsity-inducing norms: $\ell_1$

Consider a simple example of minimizing mean squared error (MSE) on two input variables

$$\hat{\beta} = \operatorname{argmin}_{\beta_1, \beta_2} \frac{1}{m} \sum_{i=1}^m (\beta_1 x_{1i} + \beta_2 x_{2i} - y_i)^2 + \lambda \|\beta\|_p$$

where  $p = 1, 2$

- The optimal model is at the intersection of the smallest ellipse
- With  $p = 1$  this intersection is often at the corners of the diamond
- At each corner, one of the variables will have zero weight  $\Rightarrow$  sparsity
- With  $p = 2$  the solutions are more likely to have non-zero weights for all variables

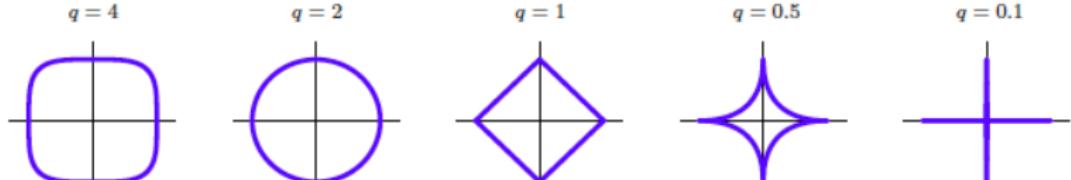


## Sparsity-inducing norms: $\ell_p$

- But is some other  $p$  also possible?
- For general  $p$ , the  $\ell_p$  norm is given by

$$\|\mathbf{w}\|_p = \left( \sum_{j=1}^d |w_j|^p \right)^{1/p}$$

- For  $p < 1$  the constraint region  $\|\mathbf{w}\|_p \leq 1$  becomes non-convex with "spikes" extending towards the corners: will give sparsity but hard to optimize
- For  $p > 1$  the constraint region is more and more "ball-like" and "box-like" and will give less and less sparsity
- $p = 1$  gives the most sparse solutions while keeping optimization problem convex



## Sparse learning problems

Combining  $\ell_1$  regularisation with different loss functions gives sparse variants of well-known models

- Hinge loss  $\Rightarrow$  Sparse SVM:  $\min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^m L_{Hinge}(\mathbf{w}^T \mathbf{x}_i, y_i) + \lambda \|\mathbf{w}\|_1$
- Logistic loss  $\Rightarrow$  Sparse logistic regression:  
$$\min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^m L_{Logistic}(\mathbf{w}^T \mathbf{x}_i, y_i) + \lambda \|\mathbf{w}\|_1$$
- Squared loss  $\Rightarrow$  Sparse regression, LASSO:  
$$\min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^m L_{sq}(\mathbf{w}^T \mathbf{x}_i, y_i) + \lambda \|\mathbf{w}\|_1$$

Sparse modelling can also be used in applications that go beyond classification. See e.g. the book Hastie, T., Tibshirani, R. and Wainwright, M., 2015. Statistical learning with sparsity: the lasso and generalizations. CRC press.

## **Stability of feature selection**

---

## Stability of feature selection

A recognized problem with variable subset selection and sparse modelling approaches is their sensitivity to small perturbations

- upon removal or addition of a few variables or examples
- addition of noise
- initial conditions of the algorithms

The lack of stability may sometimes be a problem

- It may be a symptom of a "bad" model, one that will not generalize well
- The results are not reproducible
- One variable subset fails to capture the "whole picture"

## Stability of feature selection

- A general approach to tackle the lack of stability is to use bootstrapping
- One runs the variable selection or sparse modelling algorithm on  $T$  sub-samples, drawn with replacement from the original training data
- The final variable subset may be selected as
  - The union of all selected features in the  $T$  models, or
  - Counting how many models include a given variable  $j$  and selecting the variables that occur at least given fraction of the  $T$  models

# Summary

- Feature transformations can be used to improve the input data prior to learning
- Feature selection is used to improve the interpretability, resource consumption and predictive performance of machine learning
- Hard computational problem: many heuristics to speed it up, including greedy forward selection and backward elimination
- Sparsity inducing norms can be used to learn a model with small number of features

# CS-E4710 Machine Learning: Supervised Methods

## Lecture 11: Multi-class Classification

---

Juho Rousu

November 22, 2022

Department of Computer Science  
Aalto University

## Multi-class classification

- Given a training data set  $\{(\mathbf{x}_i, y_i)\}_{i=1}^m, (\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$
- Outputs belongs to a set of possible classes or labels:  
 $y_i \in \mathcal{Y} = \{1, 2, \dots, k\}$
- In **multi-class classification**, one of the labels is considered to be the correct label, the other ones incorrect<sup>1</sup>
- In **multi-label classification**, several of the labels can be correct for a given input  $x_i$ , the output space will be  $\mathcal{Y} = \{-1, +1\}^k$ , each output  $\mathbf{y}_i$  is a  $k$ -dimensional vector
- In both cases, we aim learning a function

$$f : X \mapsto \mathcal{Y},$$

for predicting the outputs

---

<sup>1</sup>Mohri et al. book calls this case **mono-label** multi-class classification, but that is not standard vocabulary

# Multi-class classification

Two basic strategies to solve the problem:

1. Aggregated methods using multiple binary classifiers:
  - One-versus-all approach : Separate each class from all the others
  - One-versus-one or all-pairs approach: Separate each class pair from each other
  - Error-correcting output code approach: Represent each class with a binary code vector and predict the bits of the vector
2. Standalone models: learning to predict multiple classes directly
  - Multiclass SVM
  - Multiclass Boosting

## One-versus-All Classification

---

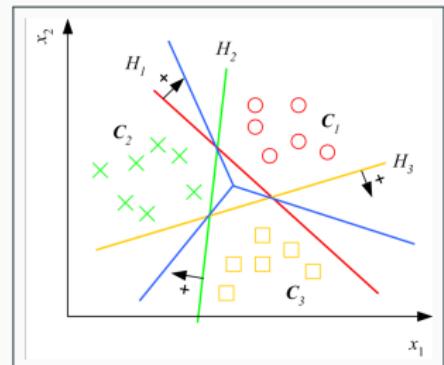
## One-versus-All Classification

- Given a training data set  $\{(\mathbf{x}_i, y_i)\}_{i=1}^m, (\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$
- If we have  $k > 2$  classes we will train  $k$  binary hypotheses  $h_1, \dots, h_k$ ,  
 $h_\ell : X \mapsto \{+1, -1\}$
- For training the  $\ell$ th hypothesis, new binary labels, called the **surrogate labels** are computed  $\tilde{y}_i^{(\ell)} = \begin{cases} +1, & \text{if } y_i = \ell, \\ -1, & \text{if } y_i \neq \ell \end{cases}$
- A binary classifier is trained to predict the surrogate labels
- The hypothesis class for the binary classifiers is not restricted: we can use any that is deemed suitable

## Geometry of the linear OVA model: separable case

An example with three classes in two-dimensional space (green crosses, red circles and yellow boxes)

- Linear classifiers  $\mathbf{w}_\ell^T \mathbf{x} + w_{\ell 0}$  are used as the predictors (Note that the bias terms  $w_{\ell 0}$  are written out explicitly)
- In the linearly separable case, there is a hyperplane  $H_\ell : \mathbf{w}_\ell^T \mathbf{x} + w_{\ell 0} = 0$  so that all  $\mathbf{x} \in C_i$  lie in the positive halfspace and all other points lie in the negative halfspace
- $h_\ell(\mathbf{x}) = \text{sgn} (\mathbf{w}_\ell^T \mathbf{x} + w_{\ell 0}) = +1$  for a single class  $\ell$



## OVA prediction

- In general, there may be more than one class  $\ell$  for which  $h_\ell(\mathbf{x}) = +1$
- Some arbitrary tie-breaking could be used, e.g. predict the class with the smallest index  $\ell$
- Better results can be obtained if the hypotheses also provide some real-valued score  $f_\ell(\mathbf{x}) \in \mathbb{R}$  (confidence, margin, etc.) for the label to be  $\ell$ .
- In that case, we can choose the label with the highest score

$$h(\mathbf{x}) = \operatorname{argmax}_\ell f_\ell(\mathbf{x})$$

- With linear models  $h_\ell(\mathbf{x}) = \operatorname{sgn}(\mathbf{w}_\ell^T \mathbf{x})$  using the margin of the example is a natural choice  $f_\ell(\mathbf{x}) = \mathbf{w}_\ell^T \mathbf{x}$

## OVA training pseudo-code

**Input:** Dataset  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m, \mathbf{x}_i \in X, y_i \in \mathcal{Y} = \{1, \dots, k\}$

**Output:** Multiclass hypothesis  $h : X \mapsto \mathcal{Y}$

**for**  $\ell \in \{1, \dots, k\}$  **do**

    Generate training dataset with surrogate labels:  $\{(\mathbf{x}_i, \tilde{y}_i^{(\ell)})\}_{i=1}^m$

    Train a binary hypothesis  $h_\ell : X \mapsto \{-1, +1\}$

    Let  $f_\ell(\mathbf{x})$  be the score for  $\mathbf{x}$  given by the model  $h_\ell$

**end for**

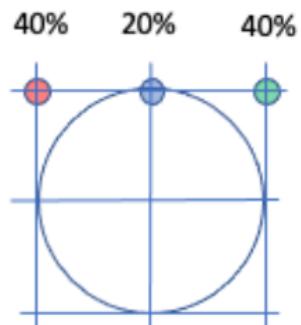
$h(\mathbf{x}) = \operatorname{argmax}_\ell f_\ell(\mathbf{x})$

## Pros and cons of the OVA approach

- OVA classification is simple to implement and therefore popular
- Training is relatively efficient with  $O(kt)$  time where  $t$  is the time to train a single binary classifier, if  $k$  is not too large
- The method may suffer from the **class imbalance** of the training sets for a given class  $\ell$ : there may be a low number of positive examples and a high number of negative examples per class
- In general OVA approach suffers from a **calibration problem**: the scores  $f_\ell(\mathbf{x})$  returned by the individual classifiers may not be comparable
- It does not always produce the optimal empirical error rate for the dataset (example below)

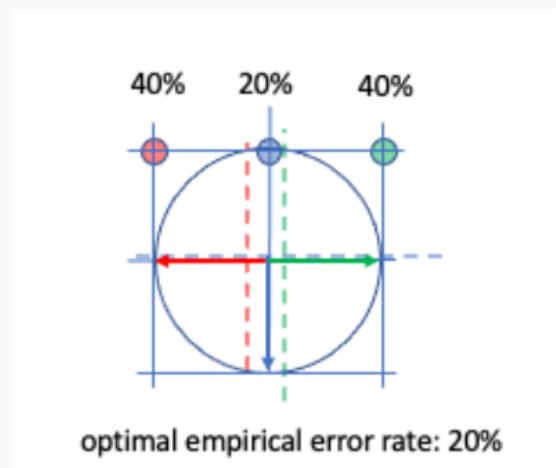
## Example: sub-optimality of OVA classification

- Consider a dataset with 3 classes (red, blue, green), with class frequencies 40%, 20%, 40%, respectively
- The classes are concentrated in distinct clusters centered at  $(-1, 1)$ ,  $(0, 1)$ , and  $(1, 1)$ , respectively



## Example: sub-optimality of OVA classification

- Optimal linear classifiers can separate red and green classes from the other two
- However, the optimal linear classifier for the blue class classifies all data as negative
- Combination of the three classifiers will predict (incorrectly) blue cluster to be either red or green, depending on tie breaking
- Empirical error rate is 20%

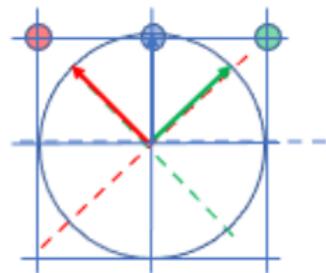


## Example: sub-optimality of OVA classification

- However, the three classes are separable by three hyperplanes  
 $f_\ell(\mathbf{x}) = \mathbf{w}_\ell^T \mathbf{x}$ ,  
 $\mathbf{w}_{red} = (-1/\sqrt{2}, 1/\sqrt{2})$ ,  $\mathbf{w}_{blue} = (0, 1)$   
and  $\mathbf{w}_{green} = (1/\sqrt{2}, 1/\sqrt{2})$  using the rule  $h(\mathbf{x}) = \text{argmax}_\ell f_\ell(\mathbf{x})$

- Note that the hyperplane  $\mathbf{w}_{blue}$  is not a good classifier as an independent model, its empirical error rate is 80%!

- Thus we see that independent training of the binary hypotheses loses information and may result in sub-optimal error rates.



optimal empirical error rate: 0%

## **One-versus-One Classification**

---

## One-versus-one approach

- An alternative is one-versus-one (OVO) or all-pairs approach
- In OVO classification, we divide a multiclass problem into a set of  $k(k - 1)/2$  binary classification problems, one for each pair of classes  $(\ell, \ell')$ ,  $1 \leq \ell < \ell' \leq k$
- This entails generating a new training set consisting of examples of the pair of classes  $(\ell, \ell')$  and generating a surrogate label

$$\tilde{y}^{\ell, \ell'} = \begin{cases} +1 & \text{if } y = \ell \\ -1 & \text{if } y = \ell' \end{cases}$$

- For each class pair, a binary hypothesis  $h_{\ell, \ell'}(\mathbf{x}) : X \mapsto \{-1, +1\}$  is trained using the generated training set

## OVO prediction

- In predicting, for each class  $\ell$  we have  $k - 1$  pairwise hypotheses, one for each class containing  $\ell$  ( $h_{\ell,\ell'}$  and  $h_{\ell',\ell}$ , for all  $\ell' \neq \ell$ )
- In the ideal case, all of the  $k - 1$  hypotheses involving class  $\ell$  would predict class  $\ell$
- In practice this may not happen, we might have for some classes  $\ell', \ell''$ 
  - $h_{\ell,\ell'}(\mathbf{x}) = +1$  - predicting class  $\ell$  for  $\mathbf{x}$
  - $h_{\ell,\ell''}(\mathbf{x}) = -1$  - predicting class  $\ell''$  for  $\mathbf{x}$
- We need to resolve these discrepancies

## OVO prediction

A voting approach can be used:

- We count for each input  $\mathbf{x}$ , how many pairwise hypotheses predict class  $\ell$  (the votes)

$$h(\mathbf{x}) = \operatorname{argmax}_{\ell} \sum_{\ell < \ell'} \mathbf{1}_{\{h_{\ell\ell'}(\mathbf{x})=+1\}} + \sum_{\ell > \ell'} \mathbf{1}_{\{h_{\ell'\ell}(\mathbf{x})=-1\}}$$

- Ties can occur with several classes receiving the same number of votes, we can break them arbitrarily (e.g. predicting the smallest index  $\ell$ )

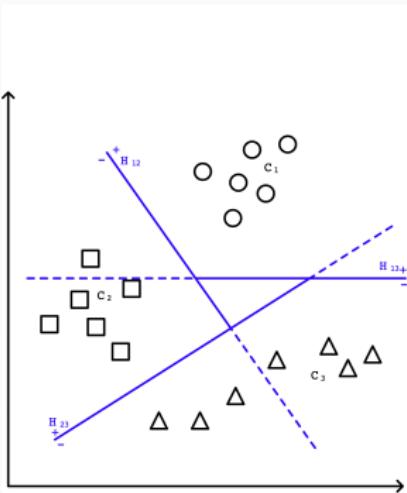
## Geometry of linear OVO classifier

An example with three classes and linear predictors  $\mathbf{w}_{\ell\ell'}^T \mathbf{x} + b_{\ell\ell'}$  for each class pair (Again the bias terms  $b_{\ell\ell'}$  written out explicitly)

- A class  $\ell$  is predicted within a region of the feature space where the number of votes for the class equal the maximum
- Geometrically, the region is defined by intersection of half-spaces

$$H_{\ell,\ell'} = \{\mathbf{x} | \mathbf{w}_{\ell\ell'}^T \mathbf{x} + b_{\ell\ell'} > 0\}, \text{ for all } \ell < \ell'$$

$$H_{\ell',\ell} = \{\mathbf{x} | \mathbf{w}_{\ell'\ell}^T \mathbf{x} + b_{\ell'\ell} < 0\}, \text{ for all } \ell > \ell'$$



- The triangle in the middle represents the region where all classes have one vote

## Pros and cons of the OVO model

- Compared to OVA, we are training many more binary classifiers:  $O(k^2)$  compared to  $O(k)$
- However, the training sets are smaller since they only contain examples of two classes at a time:
  - Faster to train
  - Increased chance of overfitting
- The OVO training sets are less likely to be imbalanced than in OVA
- Better theoretical justification through the voting approach

## Generalization performance of OVO models

- OVO model has some theoretical justification through viewing it as a kind of **majority voting ensemble**
- Assume that the pairwise hypotheses have generalization error of at most  $r$
- Now if an example  $\mathbf{x}$  with true class  $\ell'$  is misclassified by the OVO model, there must be at least one pairwise hypothesis  $h_{\ell\ell'}$  or  $h_{\ell'\ell}$  that makes an error on  $\mathbf{x}$
- The probability of this event is at most

$$\sum_{\ell < \ell'} P("h_{\ell\ell'} \text{ makes an error}") + \sum_{\ell' < \ell} P("h_{\ell'\ell} \text{ makes an error}") \leq r(k-1)$$

- Thus if the pairwise classifiers are accurate enough, the risk of the multiclass classifier can be kept relatively low

## Error-correcting codes

---

# Error-correcting codes (ECOC)

- Error-correcting output codes (ECOC) is a general methods for reducing multi-class problems to binary classification
- In the ECOC approach, each class  $\ell$  is allocated a codeword  $m_\ell$  of length  $c > 1$
- In the simplest case a binary vector can be used  $m_\ell \in \{-1, +1\}^c$
- The code words of all  $k$  classes together form a matrix  
 $M \in \{-1, +1\}^{k \times c}$

		codes					
		1	2	3	4	5	6
classes	1	-1	-1	-1	+1	-1	-1
	2	+1	-1	-1	-1	-1	-1
	3	-1	+1	+1	-1	+1	-1
	4	+1	+1	-1	-1	-1	-1
	5	+1	+1	-1	-1	+1	-1
	6	-1	-1	+1	+1	-1	+1
	7	-1	-1	+1	-1	-1	-1
	8	-1	+1	-1	+1	-1	-1

$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$
-1	+1	+1	-1	+1	+1

new example  $x$

## Error-correcting codes

- Given the codeword matrix, a binary classifier  $f_j : X \mapsto \{-1, +1\}$  is learned for each column  $j = 1, \dots, c$  of the codeword matrix
- For the classifier of column  $j$ , the training input  $x_i$  is relabeled with surrogate label  $\tilde{y}_i^{(j)} = m_{y_i j}$ , read from the row of the codeword matrix corresponding to the codeword of class  $y_i$ .
- The prediction of the ECOC model is taken as the class  $\ell$  with the fewest wrongly predicted columns of the keyword:

$$h(\mathbf{x}) = \operatorname{argmin}_{\ell=1}^k \sum_{j=1}^c \mathbf{1}_{f_j(\mathbf{x}) \neq m_{\ell j}}$$

classes	1	2	3	4	5	6
1	-1	-1	-1	+1	-1	-1
2	+1	-1	-1	-1	-1	-1
3	-1	+1	+1	-1	+1	-1
4	+1	+1	-1	-1	-1	-1
5	+1	+1	-1	-1	+1	-1
6	-1	-1	+1	+1	-1	+1
7	-1	-1	+1	-1	-1	-1
8	-1	+1	-1	+1	-1	-1

$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$
-1	+1	+1	-1	+1	+1

new example  $x$

# How to generate the codewords?

## How to generate the codewords

- Deterministic code: decide on the length  $c$  and choose binary vectors for each class so that the between class Hamming distance is as large as possible
- Random code: draw code words randomly
- Use domain knowledge: each column could be a feature describing the class

		codes					
		1	2	3	4	5	6
classes	1	-1	-1	-1	+1	-1	-1
	2	+1	-1	-1	-1	-1	-1
	3	-1	+1	+1	-1	+1	-1
	4	+1	+1	-1	-1	-1	-1
	5	+1	+1	-1	-1	+1	-1
	6	-1	-1	+1	+1	-1	+1
	7	-1	-1	+1	-1	-1	-1
	8	-1	+1	-1	+1	-1	-1

$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$
-1	+1	+1	-1	+1	+1

new example  $x$

# Why does ECOC work?

- The prediction of the ECOC model can be seen as correcting incorrectly predicted bits of the codeword
- The corrected codeword is then the one in the codebook (matrix  $M$ ) that has the smallest Hamming distance to the predicted codeword
- If the between class Hamming distance of the codewords is at least  $d$ , the upto  $\lfloor \frac{d-1}{2} \rfloor$  one bit errors can be corrected
- Another explanation comes from ensemble learning: model averaging between diverse classifiers  $f_j$  happens by minimizing the Hamming distance between codewords

		codes					
		1	2	3	4	5	6
classes	1	-1	-1	-1	+1	-1	-1
	2	+1	-1	-1	-1	-1	-1
	3	-1	+1	+1	-1	+1	-1
	4	+1	+1	-1	-1	-1	-1
	5	+1	+1	-1	-1	+1	-1
	6	-1	-1	+1	+1	-1	+1
	7	-1	-1	+1	-1	-1	-1
	8	-1	+1	-1	+1	-1	-1
		$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$
		-1	+1	+1	-1	+1	+1
		new example $x$					

## **Standalone multi-class classifiers**

---

## Standalone models

---

- Models that directly aim to minimize a multi-class loss function may give better predictive performance than the approaches based on aggregating binary classifiers
- Defining a combined model may be more efficient to train
- Multiclass models
  - Multiclass SVM
  - Multiclass boosting

## Multi-class SVM

---

## Multi-class SVM

---

- Multi-class SVM learns  $k$  hyperplanes  $f_\ell(\mathbf{x}) = \mathbf{w}_\ell^T \mathbf{x} = 0$  simultaneously
- The predicted class is the class with the highest score

$$h(\mathbf{x}) = \operatorname{argmax}_\ell f_\ell(\mathbf{x})$$

- The ideal objective would be to minimize the zero-one loss

$$L(h(\mathbf{x}), y_i) = \frac{1}{m} \sum_{i=1}^m \mathbf{1}_{h(\mathbf{x}) \neq y_i}$$

but like in binary classification, this is non-convex and NP-hard to optimize

## Multi-class SVM

- Instead, multi-class SVM focuses on the score differences between pairs of classes

$$f_\ell(\mathbf{x}) - f_{\ell'}(\mathbf{x}) = \mathbf{w}_\ell^T \mathbf{x}_i - \mathbf{w}_{\ell'}^T \mathbf{x}_i$$

- In particular, the margins between the correct class  $y_i$  and all the incorrect classes  $\ell \neq y_i$  are optimized
- We aim the score of the correct class to be higher than all the other classes by a margin (of 1)

$$\mathbf{w}_{y_i}^T \mathbf{x}_i - \mathbf{w}_\ell^T \mathbf{x}_i \geq 1 - \xi_i, \text{ for all } \ell \neq y_i$$

- Above, slack  $\xi_i \geq 0$  is used in the analogous way to binary SVMs to allow some examples to not have the required margin

## Multi-class SVM

- Multi-class SVM has  $k$  weight vectors  $\mathbf{w}_1, \dots, \mathbf{w}_k$  to control
- This is achieved by a regularizer that computes the sum of norms:  
$$\sum_{\ell=1}^k \|\mathbf{w}_\ell\|_2^2$$
- The regularizer is motivated by controlling the empirical Rademacher complexity  $\hat{\mathcal{R}}(H)$  of the hypothesis class  $H$  of multi-class SVMs:

$$\hat{\mathcal{R}}(H) \leq \sqrt{\frac{r^2 \Lambda^2}{m}},$$

where  $\sum_{\ell=1}^k \|\mathbf{w}_\ell\|_2^2 \leq \Lambda^2$  and  $\|\mathbf{x}_i\|_2^2 \leq r^2$  for all  $i = 1, \dots, m$

- Thus, minimizing the sum of norms aids achieving good generalization

## Multi-class SVM

The Multi-class SVM optimization problem can be written as follows:

$$\begin{aligned} \min_{\mathbf{w}, \xi} \quad & \frac{1}{2} \sum_{\ell=1}^k \|\mathbf{w}_\ell\|^2 + C \sum_{i=1}^m \xi_i \\ \text{s.t. } \quad & \mathbf{w}_{y_i}^T \mathbf{x}_i - \mathbf{w}_\ell^T \mathbf{x}_i \geq 1 - \xi_i, \\ & \text{for all } \ell \neq y_i \\ & \text{and for all } i = 1, \dots, m \\ & \xi_i \geq 0, i = 1, \dots, m \end{aligned}$$

- Above, we have denoted by  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_k]$  a matrix that contains the weight vectors as columns
- The problem has quadratic objective and linear constraints
- Thus with small to medium sized data, it can be solved by Quadratic Programming (QP) solvers
- For large data, stochastic gradient approaches can be used

## Multi-class Hinge loss

- Rewrite the constraint

$$\begin{aligned}\mathbf{w}_{y_i}^T \mathbf{x}_i - \mathbf{w}_\ell^T \mathbf{x}_i &\geq 1 - \xi_i, \text{ for all } \ell \neq y_i, \xi_i \geq 0 \Leftrightarrow \\ \mathbf{w}_{y_i}^T \mathbf{x}_i - \max_{\ell \neq y_i} \mathbf{w}_\ell^T \mathbf{x}_i &\geq 1 - \xi_i, \xi_i \geq 0 \Leftrightarrow \\ \xi_i &\geq 1 - [\mathbf{w}_{y_i}^T \mathbf{x}_i - \max_{\ell \neq y_i} \mathbf{w}_\ell^T \mathbf{x}_i], \xi_i \geq 0\end{aligned}$$

- Minimizing  $\xi_i$  corresponds to minimizing the **multi-class Hinge loss**

$$L_{MCHinge}(\mathbf{Wx}_i, y_i) = \max\{0, 1 - [\mathbf{w}_{y_i}^T \mathbf{x}_i - \max_{\ell \neq y_i} \mathbf{w}_\ell^T \mathbf{x}_i]\}$$

- Intuitively, it measures by how much the score difference between the correct class  $y_i$  and all the other classes  $\ell$  fails to have the desired margin 1 (margin violation)

## Multi-class SVM as a regularized loss minimization problem

- We can write the Multi-class SVM as regularized loss minimization problem:

$$\min_{\mathbf{w}, \xi} \frac{\lambda}{2} \sum_{\ell=1}^k \|\mathbf{w}_\ell\|_2^2 + \sum_{i=1}^m \max\{0, 1 - [\mathbf{w}_{y_i}^T \mathbf{x}_i - \max_{\ell \neq y_i} \mathbf{w}_\ell^T \mathbf{x}_i]\}$$

- This problem corresponds to the QP formulation by setting  $\lambda = 1/C$
- The problem is convex but the loss is piecewise linear, thus not differentiable everywhere
- A stochastic gradient descent algorithm can be defined through computing the subgradients of the objective function (skipped here)

## Multi-class SVM with kernels

- We can perform non-linear multi-class classification by using a kernel  $\kappa(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$  over the data
- The kernelized version of the multi-class SVM optimizes dual variables  $\alpha = (\alpha_{i,\ell})$ ,  $i = 1, \dots, m$ ,  $\ell = 1, \dots, k$  (one dual variable for each training example  $i$  and possible class  $\ell$ )
- The optimization problem is given by

$$\max \sum_{i=1}^m \alpha_{i,y_i} - \frac{1}{2} \sum_{\ell=1}^k \sum_{i,i'=1}^m \alpha_{i,\ell} \alpha_{i',\ell} \kappa(\mathbf{x}_i, \mathbf{x}_{i'})$$

$$\text{s.t. } \sum_{\ell} \alpha_{i,\ell} = 0$$

$$\alpha_{i,\ell} \leq 0, \text{ for } \ell \neq y_i, 0 \leq \alpha_{i,y_i} \leq C,$$

- Model's prediction in dual form:

$$\hat{y}(\mathbf{x}) = \operatorname{argmax}_{\ell=1, \dots, k} \sum_{i=1}^m \alpha_{i,\ell} \kappa(\mathbf{x}_i, \mathbf{x})$$

## Multi-class boosting

---

## Adaboost for multi-class problems

- AdaBoost.MH is a variant of AdaBoost designed for multi-class and multi-label problems
- Like Adaboost, it learns a linear combination of base classifiers
$$f_N(\mathbf{x}) = \sum_{j=1}^N \alpha_j h_j(\mathbf{x})$$
- The labels are represented as vectors
$$\mathbf{y}_i = (y_{i1}, \dots, y_{ik})^T \in \{-1, +1\}^k$$
, where multi-class setting  $y_{i\ell} = +1$  for the correct class and  $y_{i\ell'} = -1$  for all incorrect classes  $\ell'$
- The base classifiers also return vectors  $h_j(\mathbf{x}) \in \{-1, +1\}^k$ ,  
 $h_j(\mathbf{x}, \ell) \in \{-1, +1\}$
- Prediction either by taking the sign component-wise  
( $h(\mathbf{x}) = \text{sgn}(f_N(\mathbf{x}))$ ) in multi-label setting or taking  $\text{argmax}_{\ell} f_N(\mathbf{x}, \ell)$  in multi-class setting where  $f_N(\mathbf{x}, \ell)$  is the  $\ell$ 'th component of the predicted vector.

## Adaboost for multi-class problems

- A distribution over the training examples and the possible classes is maintained:  $D_t(i, \ell)$  is the weight of example  $\mathbf{x}_i$  and class  $\ell$  at iteration  $t$
- The updates to the example weights is given by the formula:

$$D_{t+1}(i, \ell) = \frac{D_t(i, \ell) \exp(-\alpha y_{i\ell} h_t(\mathbf{x}_i, \ell))}{Z_t}, \ell = 1, \dots, k$$

- $Z_t$  is a normalization factor
- All weights  $D_t(i, \ell)$  where  $y_{i\ell} \neq h_t(\mathbf{x}_i, \ell)$  are exponentially upweighted
- AdaBoost.MH can be seen to minimize an exponential loss which upper bounds zero-one loss in a multi-class setting

$$\sum_{i=1}^m \sum_{\ell=1}^k \mathbf{1}_{y_{i\ell} \neq h(\mathbf{x}_i, \ell)} \leq \sum_{i=1}^m \sum_{\ell=1}^k \exp(-y_{i\ell} h(\mathbf{x}_i, \ell))$$

# Adaboost.MH pseudo-code

---

ADABoost.MH( $S = ((x_1, y_1), \dots, (x_m, y_m))$ )

```
1  for  $i \leftarrow 1$  to  $m$  do
2      for  $l \leftarrow 1$  to  $k$  do
3           $\mathcal{D}_1(i, l) \leftarrow \frac{1}{mk}$ 
4      for  $j \leftarrow 1$  to  $N$  do
5           $h_j \leftarrow$  base classifier in  $\mathcal{H}$  with small error  $\epsilon_j = \mathbb{P}_{(i,l) \sim \mathcal{D}_j} [h_j(x_i, l) \neq y_i[l]]$ 
6           $\bar{\alpha}_j \leftarrow \frac{1}{2} \log \frac{1-\epsilon_j}{\epsilon_j}$ 
7           $Z_t \leftarrow 2[\epsilon_j(1-\epsilon_j)]^{\frac{1}{2}}$      $\triangleright$  normalization factor
8          for  $i \leftarrow 1$  to  $m$  do
9              for  $l \leftarrow 1$  to  $k$  do
10                  $\mathcal{D}_{j+1}(i, l) \leftarrow \frac{\mathcal{D}_j(i, l) \exp(-\bar{\alpha}_j y_i[l] h_j(x_i, l))}{Z_j}$ 
11      $f_N \leftarrow \sum_{j=1}^N \bar{\alpha}_j h_j$ 
12     return  $h = \text{sgn}(f_N)$ 
```

---

# Summary

- Multi-class classification can be approached as an aggregation of binary classification problems
  - One-versus-All, One-versus-One, and Error-correcting codes
- Standalone models aim to directly minimize a multiclass loss function
  - SVM and Boosting models
- Also other models exist: Multi-class neural networks, Decision trees

# CS-E4710 Machine Learning: Supervised Methods

## Lecture 12: Preference learning

---

Riikka Huusari & Juho Rousu

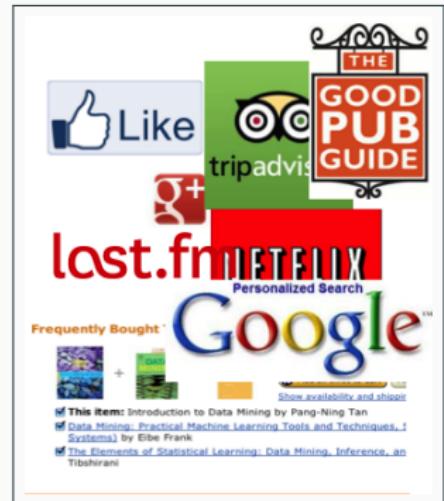
November 29, 2022

Department of Computer Science  
Aalto University

# Preference learning<sup>1</sup>

Preferences play a key role in various fields of application:

- Social networks (facebook, twitter,...)
- Recommender systems  
(Netflix, last.fm,...)
- Review web sites  
(tripadvisor, goodpubguide,...)
- Internet banner advertising
- Electronic commerce (Amazon,...)
- Adaptive retrieval systems (e.g.  
Google personalized search)



---

<sup>1</sup>Huellermeyer & Fuernkranz, Preference Learning: An Introduction, 2010

# Preference learning

- Goal: learn a predictive preference model from observed preference information.
- Notation:  $A$  is preferred over  $B$ :  
 $A \succ B$ , alternatively we can say  $A$  is ranked above  $B$



## Preference learning tasks

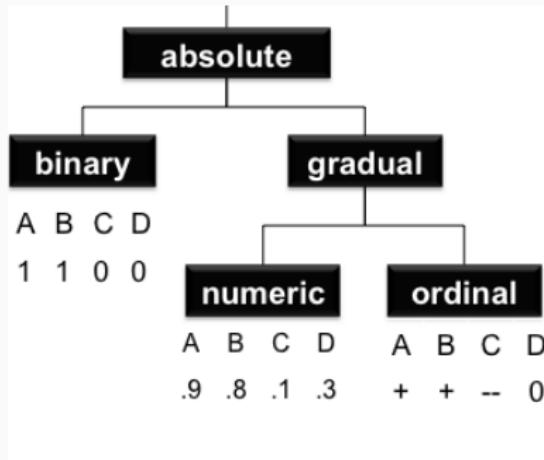
- Object ranking: Given a set of inputs (objects), predict their order.  
Example: web search ranks results based on predicted relevance to a query
- Label ranking: Given an input, and a set of potential labels, predict the (relevance) order of the labels - generalization of multi-class classification
- Rating (also called Instance ranking): Given an input, assign it to one of pre-ordered categories, e.g. (very good, good, neutral, bad, very bad) - this task is otherwise known as ordinal regression

## Representing preferences

---

# Absolute preferences

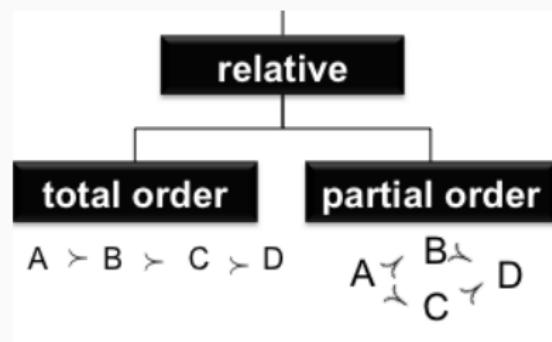
- Absolute preferences: each object has a preference score
- Binary preferences: object is preferred/not preferred (c.f. binary classification)
- Ordinal scale preferences: order of objects is defined ("very satisfied"  $\succ$  "satisfied") but distance is not
- Numeric scale: order and distance is defined



(Source: Huellermeyer & Fuernkrantz, 2010)

# Relative preferences

- Relative preferences: Preference information comes as known pairwise comparisons:  $A \succ B$
- Total order: all objects are ranked from the most preferred to the least preferred (e.g. ranking for all lunch restaurants in Otaniemi)
- Partial order: order is known only for a subset of objects: (e.g. "Fat Lizard"  $\succ$  "Maukas")



(Source: Huellermeyer & Fuernkrantz, 2010)

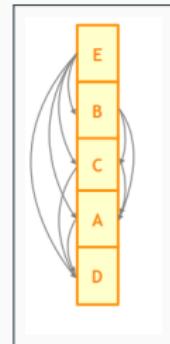
# Representing rankings

- Assume a set of objects (inputs)  $S = \{\mathbf{x}_i\}_{i=1}^m$
- Ranking function for  $S$  is a bijective function

$$\sigma : S \mapsto \{1, \dots, m\}$$

that assigns a unique rank  $1 \leq \sigma(\mathbf{x}) \leq m$  to each object in  $S$

- The inverse mapping  $\sigma^{-1}(j) : \{1, \dots, m\} \mapsto S$  gives the object of  $S$  at given rank  $j$

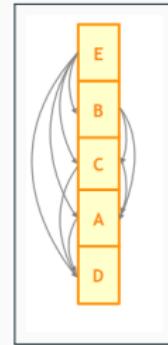


In the Figure:

- $\sigma(A) = 4, \sigma(B) = 2, \sigma(C) = 3, \sigma(D) = 5, \sigma(E) = 1$
- $\sigma^{-1}(1) = E, \sigma^{-1}(2) = B, \sigma^{-1}(3) = C, \sigma^{-1}(4) = A, \sigma^{-1}(5) = D$

# Representing rankings

- A ranking for  $S$  is a permutation of  $S$  sorted in ascending order of  $\sigma$ :  
 $\sigma^{-1}(1), \sigma^{-1}(2), \dots, \sigma^{-1}(m)$
- The ranking corresponds to a sequence of pairwise preferences:  
 $\sigma^{-1}(1) \succ \sigma^{-1}(2) \succ \dots \succ \sigma^{-1}(m)$
- Note: high preference equals low ranking and vice versa; the most preferred object has rank 1, the least preferred rank  $m$



In the Figure:

- $\sigma^{-1}(1) = E, \sigma^{-1}(2) = B, \sigma^{-1}(3) = C, \sigma^{-1}(4) = A, \sigma^{-1}(5) = D$
- $E \succ B \succ C \succ A \succ D$

# Kendall's distance

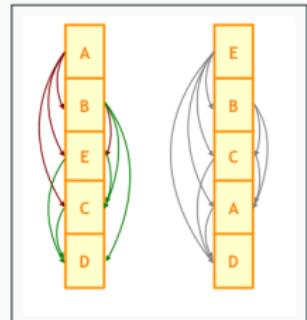
- Kendall's distance compares a predicted ranking  $\sigma'(\mathbf{x})$  to a ground truth ranking  $\sigma(\mathbf{x})$
- It counts the pairs that are inverted in the predicted ranking

$$d_K(\sigma, \sigma') = |\{(j, l) | \sigma(\mathbf{x}_j) > \sigma(\mathbf{x}_l) \text{ and } \sigma'(\mathbf{x}_j) < \sigma'(\mathbf{x}_l)\}|$$

- $d_k$  takes values between  $d_K(\sigma, \sigma') = 0$  and  $d_K(\sigma, \sigma') = m(m - 1)/2$ , where  $m$  is the number of items

- Figure:

- Predicted ranking  $\sigma'$  (left) has four inverted pairs  $(A, B), (A, E), (A, C), (B, E)$  compared to ground truth
- Kendall's distance  $d_K(\sigma, \sigma') = 4$



## Other loss functions for ranking

- Spearman's footrule: sum of absolute distances in ranks

$$d_{SF}(\sigma, \sigma') = \sum_{i=1}^m |\sigma(\mathbf{x}_i) - \sigma'(\mathbf{x}_i)|$$

- Position error: the number of wrong items that are predicted before the target item  $\mathbf{x}_*$ :

$$d_{PE}(\sigma, \sigma') = \sigma'(\mathbf{x}_*) - 1, \text{ where } \sigma(\mathbf{x}_*) = 1$$

- Discounted error: down-weights ranking errors of items with a lower true rank, with some factor  $v_i$ :

$$d_{DE}(\sigma, \sigma') = \sum_{i=1}^m v_i d_{\mathbf{x}_i}(\sigma, \sigma'),$$

where  $d_{\mathbf{x}_i}(\sigma, \sigma')$  is some distance of rankings of single item  $x_i$  in  $\sigma$  and  $\sigma'$

## Object ranking

---

# Object ranking

Given a training set of (input) objects  $\{\mathbf{x}_i\}_{i=1}^m$  and set of pairwise preferences  $\mathcal{P} = \{(i, j) | \mathbf{x}_i \succ \mathbf{x}_j\}$  our aim is to learn a ranking function  $\sigma$  that can order new sets of objects  $\{\mathbf{x}'_j\}_{j=1}^n$

## Training

$$(0.74, 1, 25, 165) \succ (0.45, 0, 35, 155)$$

$$(0.47, 1, 46, 183) \succ (0.57, 1, 61, 177)$$

$$(0.25, 0, 26, 199) \succ (0.73, 0, 46, 185)$$

 $\succ$ 

Pairwise  
preferences  
between objects  
(instances)

## Prediction (ranking a new set of objects)

$$\mathcal{Q} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9, \mathbf{x}_{10}, \mathbf{x}_{11}, \mathbf{x}_{12}, \mathbf{x}_{13}\}$$

$$\mathbf{x}_{10} \succ \mathbf{x}_4 \succ \mathbf{x}_7 \succ \mathbf{x}_1 \succ \mathbf{x}_{11} \succ \mathbf{x}_2 \succ \mathbf{x}_8 \succ \mathbf{x}_{13} \succ \mathbf{x}_9 \succ \mathbf{x}_3 \succ \mathbf{x}_{12} \succ \mathbf{x}_5 \succ \mathbf{x}_6$$

## Two-step scheme for object ranking

We can approach object ranking through a two-step process:

1. Learn a model that assigns preference score  $f(\mathbf{x}, \mathbf{x}')$  for the preferences  $\mathbf{x} \succ \mathbf{x}'$  for any pair of inputs  $(\mathbf{x}, \mathbf{x}')$
2. For a set of new points to be ranked  $\{\mathbf{x}_i\}_{i=1}^n$  find the ranking  $\sigma$  that maximizes the agreement between the ranking and the predicted preference score:

$$AGREE(\sigma, f) = \sum_{\sigma(\mathbf{x}_i) < \sigma(\mathbf{x}_j)} f(\mathbf{x}_i, \mathbf{x}_j),$$

that is, the sum of preference scores consistent with  $\sigma'$

Cohen, W.W., Schapire, R.E. and Singer, Y., 1999. Learning to order things. Journal of artificial intelligence research, 10, pp.243-270.

## First step: Learning to order pairs

- We can convert the problem of ordering pairs into a binary classification problem with input data given by the pairs of objects
- As training data we assume a set of inputs  $\{\mathbf{x}_i\}_{i=1}^m$  and set of preferences  $\mathcal{P} = \{(i, j) | \mathbf{x}_i \succ \mathbf{x}_j\}$ .
- A classifier should predict for a given a pair of inputs  $(\mathbf{x}, \mathbf{x}')$

$$h(\mathbf{x}, \mathbf{x}') = \begin{cases} 1 & \text{if } \mathbf{x} \succ \mathbf{x}' \\ -1 & \text{if } \mathbf{x}' \succ \mathbf{x} \end{cases}$$

- We can use any classification algorithm on the pairwise data to learn the predictor
- If the classifier outputs real valued scores (e.g. probabilities, margins, etc.)  $f(\mathbf{x}_i, \mathbf{x}_j)$ , we can use the scores instead of the predicted binary labels

## Second step: Extracting a ranking

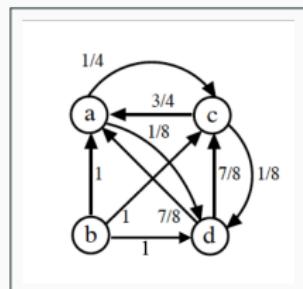
- For a set of new inputs  $\mathbf{x}_1, \dots, \mathbf{x}_n$  we will obtain a pairwise preference  $f(\mathbf{x}_i, \mathbf{x}_j)$  for each pair  $(\mathbf{x}_i, \mathbf{x}_j)$
- These predictions can be contradictory, e.g. we may have cycle  $A \succ B \succ C \succ A$
- To extract a ranking for the objects, pairwise predictions that are not consistent with the chosen order need to be ignored
- The problem is to find a ranking  $\hat{\sigma}$  that maximizes the agreement with  $f$ :  $\hat{\sigma} = \operatorname{argmax}_{\sigma} AGREE(\sigma, f)$
- However: Finding the highest scoring ranking is a NP-hard optimization problem (Cohen et al. 1999)

Cohen, W.W., Schapire, R.E. and Singer, Y., 1999. Learning to order things. Journal of artificial intelligence research, 10, pp.243-270.

## Second step: Extracting a ranking

- An approximate solution can be found by a graph based solution
- In the graph, objects correspond to nodes and pairwise preferences to directed edges
- Edge weights are preference scores  $f(\mathbf{x}_i, \mathbf{x}_j)$  which are scaled to interval  $[0, 1]$  and satisfy  $f(\mathbf{x}_i, \mathbf{x}_j) + f(\mathbf{x}_j, \mathbf{x}_i) = 1$
- Our goal is to maximize the agreement between the preference scores and the chosen ranking

$$AGREE(\sigma', f) = \sum_{\sigma'(\mathbf{x}_i) < \sigma'(\mathbf{x}_j)} f(\mathbf{x}_i, \mathbf{x}_j),$$

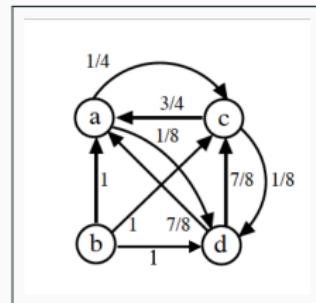


- This amounts to keeping all edges consistent with the chosen order and ignoring the conflicting ones

## Cohen's algorithm

Cohen's algorithm (Cohen et al. 1999) builds a preference graph with nodes corresponding to the input data points (in figure:  $S = \{a, b, c, d\}$ )

- Weighted edges correspond to the predicted preference scores  $f(\mathbf{x}, \mathbf{x}')$  and  $f(\mathbf{x}', \mathbf{x})$
- The algorithm maintains for each node the net preference score  $\pi(\mathbf{x}) = \sum_{\mathbf{x}'} f(\mathbf{x}, \mathbf{x}') - \sum_{\mathbf{x}'} f(\mathbf{x}', \mathbf{x})$  which is the sum of outgoing edge weights (pairwise preferences  $\mathbf{x} \succ \mathbf{x}'$ ) minus the sum of incoming edge weights (pairwise preferences  $\mathbf{x}' \succ \mathbf{x}$ )



Cohen, W.W., Schapire, R.E. and Singer, Y., 1999. Learning to order things. Journal of artificial intelligence research, 10, pp.243-270.

## Cohen's algorithm

- The net preference scores for the full graph are:

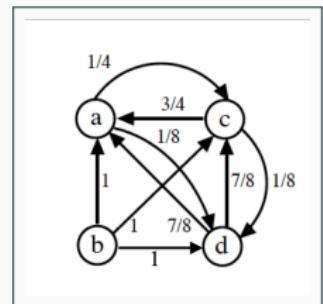
$$\pi(a) = 0 + 1/4 + 1/8 - (1 + 3/4 + 7/8) = -18/8$$

$$\pi(b) = 1 + 1 + 1 - 0 = 3$$

$$\pi(c) = 0 + 3/4 + 1/8 - (1 + 1/4 + 7/8) = -10/8$$

$$\pi(d) = (0 + 7/8 + 7/8) - (1 + 1/8 + 1/8) = 4/8$$

- The most preferred node is computed, it is  $b$
- We set  $\sigma'(b) = 1$



## Cohen's algorithm

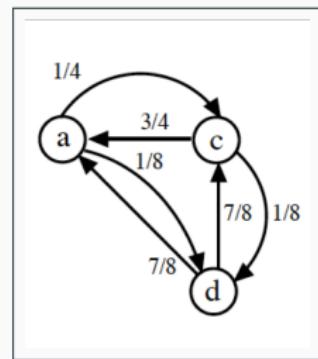
- The most preferred node is deleted and the net preference scores  $\pi(x)$  are updated to reflect the new graph

$$\pi(a) = -18/8 + (1 - 0) = -10/8$$

$$\pi(c) = -10/8 + (1 - 0) = -2/8$$

$$\pi(d) = 4/8 + (1 - 0) = 12/8$$

- The most preferred node is again computed:  $(d)$  and it gets the first available rank:  
 $\sigma(d) = 2$



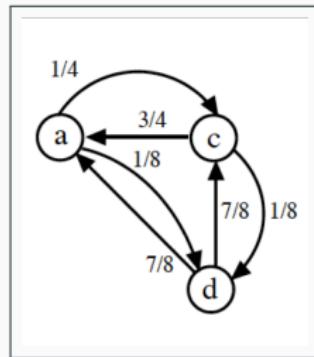
## Cohen's algorithm

- The most preferred node  $d$  is deleted and the net preference scores are updated to reflect the new graph

$$\pi(a) = -10/8 + (7/8 - 1/8) = -2/4$$

$$\pi(c) = -2/8 + (7/8 - 1/8) = 2/4$$

- The most preferred node is  $c$ , we set  $\sigma(c) = 3$

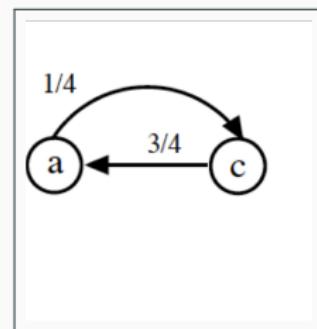


## Cohen's algorithm

- One node  $a$  remains in the graph with net preference score

$$\pi(a) = -2/4 + (3/4 - 1/4) = 0$$

- We set  $\sigma(a) = 4$ , and terminate the algorithm
- The extracted total order is then  $b \succ d \succ c \succ a$



## Cohen's algorithm: pseudocode

**Input:** A set of objects  $S = \{x_i\}_{i=1^n}$ , preference function  $f(x, x')$

$t=1$

Set  $\pi(x)$  as the net preference score of for all  $x \in S$ :

$$\pi(x) = \sum_{x' \in S} f(x, x') - \sum_{x' \in S} f(x', x)$$

**while**  $S \neq \emptyset$  **do**

    Find the object with largest net preference:

$$x^* = \operatorname{argmax}_{x \in S} \pi(x)$$

$$S = S - x^*$$

$$\sigma(x^*) = t;$$

    Remove the contribution of  $x^*$  from the net preference scores:

$$\pi(x) = \pi(x) + (f(x^*, x) - f(x, x^*)) \text{ for all } x \in S$$

$$t = t + 1;$$

**end while**

**Output:**  $(\sigma(x_1), \sigma(x_2), \dots, \sigma(x_n))$

## **Preference learning through ranking loss minimization**

---

## Preference learning through ranking loss minimization

- The above described scheme is two-step preference learning scheme (binary classification and post-processing to extract a ranking)
- Although it simple and can be effective, it does not directly optimize a loss function for ranking
- In the following we examine algorithms that directly to optimize the quality of the ranking

## Preference learning through linear models

- Consider learning a linear model  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  that assigns a preference score  $f(\mathbf{x})$  to each input  $\mathbf{x}$
- As training data we assume a set of inputs  $\{\mathbf{x}_i\}_{i=1}^m$  and set of preferences  $\mathcal{P} = \{(i, j) | \mathbf{x}_i \succ \mathbf{x}_j\}$ .
- The pair  $(\mathbf{x}_i, \mathbf{x}_j)$ ,  $(i, j) \in \mathcal{P}$  is consistently predicted if and only if

$$f(\mathbf{x}_i) \geq f(\mathbf{x}_j)$$

or alternatively if and only if

$$f(\mathbf{x}_i) - f(\mathbf{x}_j) = \mathbf{w}^T (\mathbf{x}_i - \mathbf{x}_j) = \mathbf{w}^T \Delta \mathbf{x}_{ij} \geq 0$$

where  $\Delta \mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$  is the difference vector of  $\mathbf{x}_i$  and  $\mathbf{x}_j$

## Preference learning through linear models

- We can denote the preferences by labels

$$y_{ij} = \begin{cases} +1 & \text{if } (i,j) \in \mathcal{P} \\ -1 & \text{if } (j,i) \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$$

- Then a pair is consistently predicted if it has a non-negative margin

$$y_{ij}\mathbf{w}^T \Delta\mathbf{x}_{ij} \geq 0$$

- This is a hyperplane classifier with difference vectors  $\Delta\mathbf{x}_{ij}$  as inputs and the preferences encoded into the labels  $y_{ij}$
- Data points with  $y_{ij} = 0$  correspond to the pairs with no preferred order. They are always consistently classified.

## Preference learning through linear discrimination

- Recall that finding the hyperplane that minimizes the zero-one loss of training set is NP-hard
- In our case, an error happens when the pair has a negative margin

$$y_{ij}\mathbf{w}^T \Delta \mathbf{x}_{ij} < 0$$

in other words when the model puts the pair in inverted order

$$\mathbf{w}^T \mathbf{x}_i < \mathbf{w}^T \mathbf{x}_j, \mathbf{x}_i \succ \mathbf{x}_j$$

- Thus, minimizing the number of inverted pairs - the Kendall distance - is hard as well

# Hinge loss for preference learning

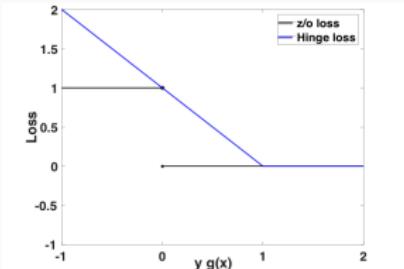
- Similarly to the binary classification, replacing the zero-one loss with a convex upper bound, such as Hinge loss, leads to efficient optimization
- Hinge loss for a pair  $(i, j)$ :

$$\max(0, 1 - y_{ij} \mathbf{w}^T \Delta \mathbf{x}_{ij})$$

- Loss is incurred if the functional margin  $y_{ij} \mathbf{w}^T \Delta \mathbf{x}_{ij} < 1$
- Average Hinge loss over all pairs:

$$\frac{1}{m(m-1)} \sum_{(i,j), i \neq j} \max(0, 1 - y_{ij} \mathbf{w}^T \Delta \mathbf{x}_{ij})$$

- RankSVM minimizes the above loss, while controlling the norm of the weight vector



# RankSVM

- RankSVM (Joachims, 2002) solves the following regularised learning problem:

$$\begin{aligned} \min_{\mathbf{w}, \xi} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + \frac{C}{|P|} \sum_{(i,j) \in \mathcal{P}} \xi_{ij} \\ \text{s.t. } \quad & \mathbf{w}^T \mathbf{x}_i - \mathbf{w}^T \mathbf{x}_j \geq 1 - \xi_{ij}, \text{ for all } (i,j) \in \mathcal{P} \\ & \xi_{ij} \geq 0, \text{ for all } (i,j) \in \mathcal{P} \end{aligned}$$

- The objective is to minimize the combination of the norm of the weight vector (regularizer) and the loss (given by  $\xi_{ij}$ )
- Note that only the preferred order  $(i,j) \in \mathcal{P}$  is considered, not the opposite order  $(j,i)$ . This is ok, since:

$$y_{ij} \mathbf{w}^T \Delta \mathbf{x}_{ij} = -y_{ij} \mathbf{w}^T \Delta \mathbf{x}_{ji} = y_{ji} \mathbf{w}^T \Delta \mathbf{x}_{ji}$$

- That is, satisfying the constraints for  $(i,j) \in \mathcal{P}$ , the constraints for  $(j,i)$  are automatically satisfied

## RankSVM with kernels

- We can use kernel functions to perform non-linear ranking
- This is solved by the dual RankSVM problem:

$$\begin{aligned} \max_{\alpha} g(\alpha) &= \sum_{(i,j) \in \mathcal{P}} \alpha_{ij} - \frac{1}{2} \sum_{(i,j) \in \mathcal{P}} \sum_{(r,s) \in \mathcal{P}} \alpha_{ij} \Delta \mathbf{x}_{ij}^T \Delta \mathbf{x}_{rs} \alpha_{rs} \\ \text{s.t. } &0 \leq \alpha_{ij} \leq \frac{C}{|P|}, \text{ for all } i \succ j \end{aligned}$$

- It is a constrained Quadratic Programme
- The inner product  $\Delta \mathbf{x}_{ij}^T \Delta \mathbf{x}_{rs}$  can be replaced with any kernel  $\kappa(\Delta \mathbf{x}_{ij}, \Delta \mathbf{x}_{rs})$  acting on the difference vectors  $\Delta \mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$
- The number of dual variables is proportional to the set of pairwise preferences, at worst quadratic in number of objects

## Boosting for ranking

---

## RankBoost algorithm

- RankBoost is an algorithm that applies the AdaBoost framework to the ranking problem
- It gets as input a training sample  $S = \{(x_i, x'_i, y_i)\}$  where

$$y_i = \begin{cases} +1 & \text{if } x'_i \succ x_i \\ 0 & \text{if } x'_i, x_i \text{ have the same preference or are incomparable} \\ -1 & \text{if } x_i \succ x'_i \end{cases}$$

- It learns a linear combination

$$f(x) = \sum_{t=1}^T \alpha_t h_t(x)$$

of base rankers or weak rankers  $h_t$

- Base rankers are assumed to output a binary preference (preferred/not preferred):  $h_t(x) \in \{0, 1\}$  learned by minimizing the weighted ranking errors  $D_t(i)\mathbf{1}_{y_i(h_t(x'_i) - h_t(x_i)) < 0}$  in the training set

## Weak ranker?

- The weak learning assumption that the base rankers are assumed to satisfy is that they rank correctly more pairs than incorrectly
- Denote by

$$\epsilon_t^+ = \sum_{i=1}^m D_t(i) \mathbf{1}_{y_i(h_t(x'_i) - h_t(x_i)) \geq 0}$$

the proportion of correctly ranked pairs, by

$$\epsilon_t^- = \sum_{i=1}^m D_t(i) \mathbf{1}_{y_i(h_t(x'_i) - h_t(x_i)) < 0}$$

the proportion of the incorrectly ranked pairs and by

$$\epsilon_t^0 = \sum_{i=1}^m D_t(i) \mathbf{1}_{y_i(h_t(x'_i) - h_t(x_i)) = 0}$$

the proportion of the non-ranked pairs

- A weak ranker is thus required to satisfy:  $\epsilon_t^+ - \epsilon_t^- > 0$

## Weights of the weak rankers

- The weights of the weak learner is given by  $\alpha_t = \frac{1}{2} \log \frac{\epsilon_t^+}{\epsilon_t^-}$  which represents the log-odds ratio between the weak learner being correct or incorrect on the training sample
- When the weak ranking assumption  $\epsilon_t^+ - \epsilon_t^- > 0$  is satisfied, we have  $\frac{\epsilon_t^+}{\epsilon_t^-} > 1$
- Thus  $\alpha_t > 0$  in this case

## Re-weighting of examples

- The weight distribution of examples is updated by

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i (h_t(x'_i) - h_t(x_i))}}{Z_t}$$

- The exponent will be positive when the weak ranker (with  $\alpha_t > 0$ ) makes a ranking mistake ( $y_i(h_t(x'_i) - h_t(x_i)) < 0$ ) on the pair  $\Rightarrow$  up-weighting the example for the next iteration
- Correctly classified pairs result in down-weighting
- For pairs for which the weak ranker cannot decide on ranking ( $h(x_i) - h(x'_i) = 0$ ), weights are unchanged
- $Z_t = \sum_{i=1}^m D_t(i) e^{-\alpha_t y_i (h_t(x'_i) - h_t(x_i))} = \epsilon_t^0 + 2(\epsilon_t^+ \epsilon_t^-)^{1/2}$  is a normalization factor

# RankBoost pseudocode

```
RANKBOOST( $S = ((x_1, x'_1, y_1), \dots, (x_m, x'_m, y_m))$ )
```

```
1  for  $i \leftarrow 1$  to  $m$  do
2       $\mathcal{D}_1(i) \leftarrow \frac{1}{m}$ 
3  for  $t \leftarrow 1$  to  $T$  do
4       $h_t \leftarrow$  base ranker in  $\mathcal{H}$  with smallest  $\epsilon_t^- - \epsilon_t^+ = -\mathbb{E}_{i \sim \mathcal{D}_t} [y_i(h_t(x'_i) - h_t(x_i))]$ 
5       $\alpha_t \leftarrow \frac{1}{2} \log \frac{\epsilon_t^+}{\epsilon_t^-}$ 
6       $Z_t \leftarrow \epsilon_t^0 + 2[\epsilon_t^+ \epsilon_t^-]^{\frac{1}{2}}$      $\triangleright$  normalization factor
7      for  $i \leftarrow 1$  to  $m$  do
8           $\mathcal{D}_{t+1}(i) \leftarrow \frac{\mathcal{D}_t(i) \exp [-\alpha_t y_i (h_t(x'_i) - h_t(x_i))]}{Z_t}$ 
9   $f \leftarrow \sum_{t=1}^T \alpha_t h_t$ 
10 return  $f$ 
```

## RankBoost error

- RankBoost can be shown to minimize the loss function

$$\sum_{i=1}^m e^{-y_i(f_N(x'_i) - f_N(x_i))}$$

- It is a convex upper bound of the empirical risk defined as the number of inverted pairs  $\hat{R}(h) = \sum_{i=1}^m \mathbf{1}_{f_N(x'_i) - f_N(x_i) \leq 0}$  i.e. Kendall's distance
- If all weak rankers satisfy  $\frac{\epsilon_t^+ - \epsilon_t^-}{2} \geq \gamma \geq 0$  then  $\hat{R}(h) \leq \exp(-2\gamma^2 T)$
- The empirical risk goes exponentially down in the boosting iterations  $T$ ,
- A larger edge  $\epsilon_t^+ - \epsilon_t^-$  - how many more pairs are correct than incorrect - gives faster decrease of the risk

# Summary

- Preference learning covers a number of machine learning tasks where the aim is to order, rank or rate objects
- In object ranking the goal is to rank new objects with a ranking function learned from existing preference data
- Two-stage approach for object ranking consists of using a binary classifier to order pairs, followed by a phase where the best consistent order for the whole dataset is extracted
- RankSVM and RankBoost are examples of models that aim to directly minimize a ranking loss function