Machine Learning: Supervised Methods
NOTES

Kristian Bonnici

December 19, 2021

Contents

Abstract

These notes cover some of the key concepts in supervised learning. However it's not intended to be a comprehensive introduction into supervised methods. To get most out of the notes, one should have some base knowledge on machine learning and basic algebra.

Summary of Notation

The next list describes several symbols that will be later used within the body of the document.

log Natural logarithm (also commonly written as $\ln, \log_e$)

Part I

Theory

1 Introduction

1.1 Theoretical paradigms

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

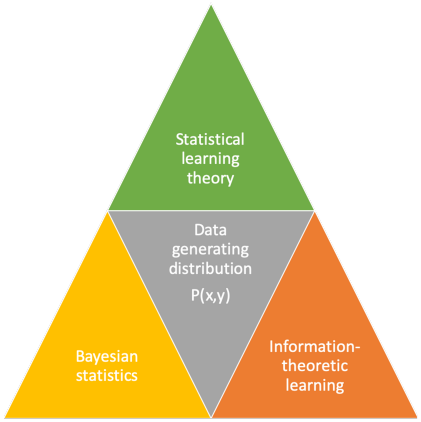


Figure 1: Paradigms for data generation distributions

- **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:** 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)

1.2 Dimensions of a supervised learning algorithm

1. **Training sample:** $S = \{(x_i, y_i)\}_{i=1}^{m}$ the training examples $(x, y) \in X \times Y$ independently drawn from a identical distribution $(i.i.d)D$ defined on $X \times Y, X$ is a space of inputs, Y is the space of outputs.

2. **Model or hypothesis:** $h : X \to Y$ that we use to predict outputs given the inputs x.
3. **Loss function:** $L : Y \times Y \to \mathbb{R}, L(...) \geq 0, L(y, y')$ 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.

1.3 Classification (Task 1/3)

Problem: partitioning the data into pre-defined classes by a *decision boundary* or *decision surface*.

Multi-class classification: more than two classes

- **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)

1.3.1 Version space

Version space: the set of all consistent hypotheses of the hypothesis class

- **Consistent hypothesis:** if correctly classifies all training examples
- **In version space:**
  - **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

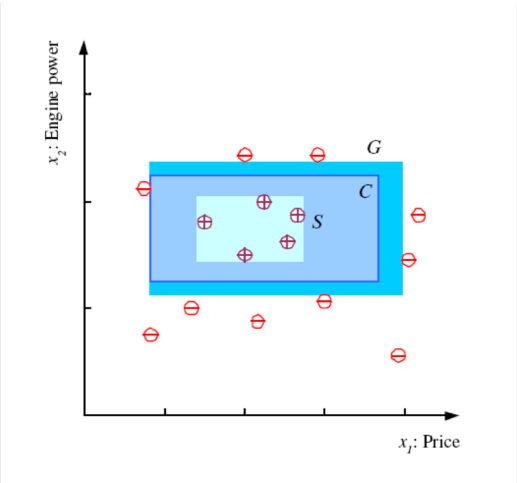


Figure 2: Illustration of a Version Space.

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

1.4 Regression (Task 2/3)

Problem: output variables which are numeric.

1.5 Ranking & preference learning (Task 3/3)

Problem: predict a ordered list of preferred objects.

Training data (typically): pairwise preferences.

- e.g. user x prefers movie y_i over movie y_j

Output: ranked list of elements.

1.6 Generalization

Aim: 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**:

$$R(h) = E_{(x,y)\sim D}[L(h(x), y)] \tag{1}$$

Where:

D : Unknown distribution where from training and future examples are drawn from (i.i.d assumption)

What can we say about $R(h)$ based on training examples and the hypothesis class $\mathcal{H}$ alone? Two possibilities:

- Empirical evaluation by testing (Section 1.6.1)
- Statistical learning theory (Section 2)

1.6.1 Model evaluation by testing

What: estimate the model's ability to generalize on future data

How: approximating true risk by computing the empirical risk on a independent test sample:

$$R_{test}(h) = \sum_{(x_i,y_i)\in S_{test}}^{m} L(h(x_i), y_i)$$

- The expectation of $R_{test}(h)$ is the true risk $R(h)$

1.7 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

---

2 Statistical Learning Theory

What: Statistical learning theory focuses in analyzing the generalization ability of learning algorithms. It's the theoretical background on machine learning.

Goal: Generalization (Section 1.6)

2.1 Probably Approximately Correct (PAC) learning

What: The most studied *theoretical framework* for analyzing the generalization performance of machine learning algorithms. It formalizes the notion of generalization in machine learning. In practice, it's asking for bounding the generaliation error (ϵ) with high probability ($1-\delta$), with arbitrary level of error $\epsilon > 0$ and confidence $\delta > 0$.

Ingredients:

- **input space** X containing all possible **inputs** x
- set of possible **labels** Y (in binary classification $Y = \{0, 1\}$)
- **concept class** $\mathcal{C}$ contains **concepts** $C : X \to Y$ (to be learned), concept C gives a label $C(x)$ for each input x
  - underlying ground truth $\to$ to be learn in the ideal case
  - unknown in practice (or otherwise ML is not needed)
- Unknown (i.i.d) **probability distribution** D for the data
- **training sample** $S = (x_1, C(x_1)), ..., (x_m, C(x_m))$ drawn independently from D
- **hypothesis class** $\mathcal{H}$
  - in the **basic case** $\mathcal{H} = \mathcal{C}$ but this assumption can be relaxed

Goal: to learn a hypothesis with a low generalization error. For 0/1 loss:

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

2.1.1 PAC learnability

What: Question, can we learn a concept class.

PAC-learnable concept class $\mathcal{C}$ if:

- 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

$$Pr(\underbrace{R(\overbrace{h_S}^{\text{chosen hypothesis from } \mathcal{H}}) \leq \epsilon}_{\text{"low generalization error" event}}) \geq \underbrace{1-\delta}_{\text{success rate}} \tag{2}$$

$P(GeneralizationError\ of\ h_S \leq GeneralizationError\ of\ interest) \geq Desired\ SuccessRate$
$P(selection\ hypothesis\ (from\ \mathcal{H})\ with\ GeneralizationError \leq \epsilon) \geq Desired\ SuccessRate$
$Probability\ of\ low\ GeneralizationError \geq Desired\ SuccessRate$

Where:

h_S**:** output hypothesis
S**:** training sample
$m = |S|$**:** sample size that grows polynomially in $1/\epsilon$, $1/\delta$
ϵ**:** generalization error of interest (arbitrary)
$1-\delta$**:** desired success rate / confidence (arbitrary)

Efficiently PAC-learnable concept class $\mathcal{C}$ if:

- **PAC-learnable**
- $\mathcal{A}$ runs in time polynomial in m, $1/\epsilon$, and $1/\delta$
  - We want the requirement for training data and running time **not to explode** when we make ϵ and δ stricter $\rightarrow$ requirement of polynomial growth

Interpretation:

- ϵ: sets the level of generalization error that is of interest to us
  - e.g. say we are satisfied with predicting incorrectly 10% of the new datapoints $\rightarrow \epsilon = 0.10$
- $1 - \delta$: sets a level of confidence that is of interest to us
  - e.g. say we are satisfied of the training algorithm to fail 5% of the time to provide a good hypothesis $\rightarrow \delta = 0.05$
- $\{R(h_S) \leq \epsilon\}$: The event "low generalization error"
  - considered as a *random variable* because we cannot know beforehand which hypothesis $h_S \in \mathcal{H}$ will be selected by the algorithm

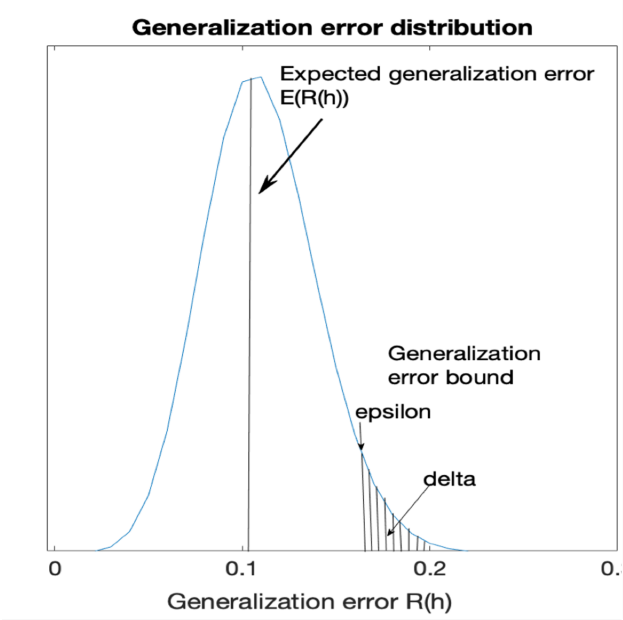


Figure 3: Generalization error distribution and error bound

Generalization error bound vs. expected test error:

- We bound the probability of being in the high error tail of the distribution (not the convergence to the mean or median generalization error) $\rightarrow$ thus **empirically estimated test errors** might be considerably lower than the bounds suggest.

2.2 Learning with finite hypothesis classes

What: Finite concept classes arise when:

- **Input variables** have finite domains or they are converted to such in preprocessing (e.g. discretizing real values)
- The representations of the hypotheses have finite size (e.g. the number of times a single variable can appear)

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 or close to it (relaxed).
- The more hypotheses there are in $\mathcal{H} \rightarrow$ the more training examples are needed

2.2.1 Example: Boolean conjunctions

What: Example of a finity hypothesis class.

Example hypothesis class: Boolean conjunctions

- Dealing with subclasses of Boolean formulae, expressions **binary input variables** (literals) combined with **logical operators** AND & NOT.

Example case: Aldo likes to do sport only when the weather is suitable

- **Training data:** Also has given examples of suitable and not suitable weather

	x^t						$r(x^t)$
t	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
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.

- **Model:** Let us build a classifier (boolean formulae containing AND, and NOT, but not OR operators) for Aldo to decide whether to do sports today
  - e.g. if (Sky=Sunny) AND NOT (Wind=Strong) then (EnjoySport=1)
- **Number of hypotheses $|\mathcal{H}|$:**
  - **Each variable:** "AND", "AND NOT", or can be excluded from the rule $\rightarrow$ 3 possibilities
  - **Total number of hypotheses** is thus 3^d, where d is the number of variables $\rightarrow |\mathcal{H}| = 3^6 = 729$
- **Plotting the bound for Aldo's problem using boolean conjunctions:**
  - **Left plot:** generalization bound ϵ is shown for different values of delta δ, using d = 6 variables.

– **Right plot:** generalization bound ϵ is shown for increasing number of input variables d, using delta $\delta = 0.05$

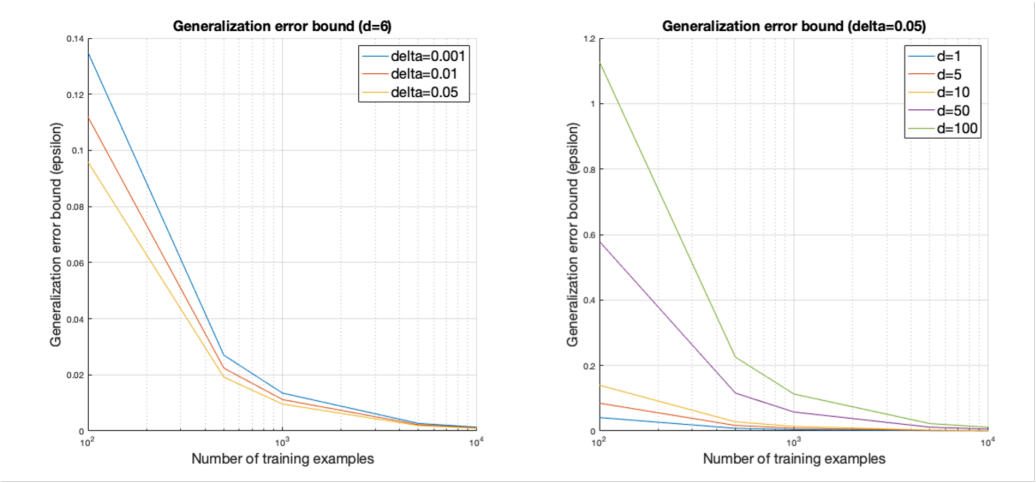


Figure 4: Boolean conjunctions: Generalization error bounds

- **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 δ) for obtaining low error requires more data for the same level of error

2.3 Learning with infinite hypothesis classes

Most models used in practise rely on infinite hypothesis classes, e.g.

- $\mathcal{H}$ = hyperplanes in $\mathbb{R}^d$ (e.g. Support vector machines)
- $\mathcal{H}$ = neural networks with continuous input variables

2.3.1 Vapnik-Chervonenkis (VC) dimension

Purpose: Vapnik-Chervonenkis dimension lets us **study** learnability of infinite hypothesis classes **through** the concept of shattering

What: can be understood as measuring the capacity of a hypothesis class to adapt to different concepts

- $VCdim(\mathcal{H})$ = size of the **largest training set** that we can find a **consistent classifier for all labelings in** Y^m
- Intuitively:
  - low $VCdim \rightarrow$ easy to learn, low sample complexity
  - high $VCdim \rightarrow$ hard to learn, high sample complexity
  - infinite $VCdim \rightarrow$ cannot learn in PAC framework

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): $VCdim(\mathcal{H}) < d+1$

Formally: (for binary labelings)

- Through **growth function**:

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

  - 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

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

2.3.1.1 Shattering

What: underlying concept in VC dimension

Given: a set of points $S = x_1, ..., x_m$ 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 $x \in S_+$

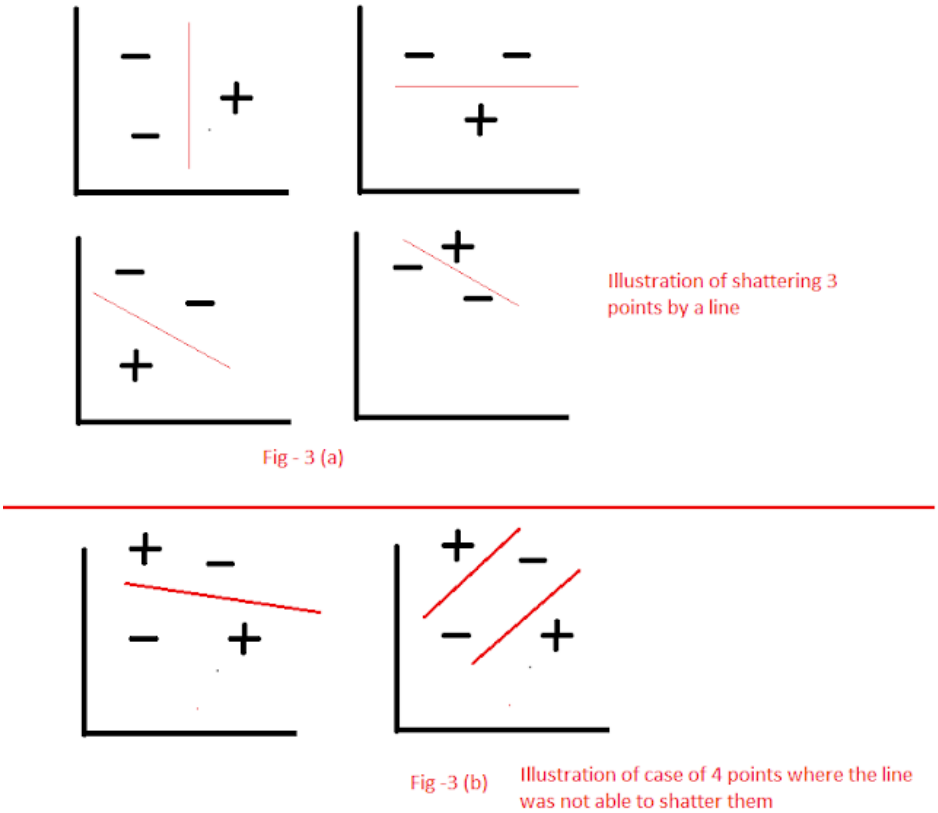


Figure 5: Illustration of shattering

2.3.1.2 Generalization bound with VC-dimension

What: Way of using VC-dimensions to analyze machine learning algorithms.

(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 $h \in \mathcal{H}$:

$$R(h) = \underbrace{\hat{R}(h)}_{\text{empirical risk}} + \sqrt{\frac{2\log(em/d)}{m/d}} + \sqrt{\frac{\log(1/\delta)}{2m}}$$

Where:

d**:** VC-dimension

m**:** amount of training data

e**:** Euler's number, $e \approx 2.71828$

δ**:** failure rate

- m/d shows that changing to a model with higher VC-dimension d makes the same amount of training samples m less effective. $\rightarrow$ the second term grows when VC-dimension grows.
  - For more complex $\mathcal{H}$ class, one needs more data m to guarantee similar kind of generalization.
- Manifestation of the **Occam's razor principle:** to justify an increase in the complexity, we need reciprocally more data

2.3.2 Rademacher complexity

What: Rademacher complexity defines complexity as the capacity of hypothesis class to fit random noise

Why: Rademacher complexity is a practical alternative to VC dimension, giving typically sharper bounds (but requires a lot of simulations to be run).

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** $S_1, ..., S_M$ from S_0 by randomly drawing a new label $\sigma \in Y$ 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

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

- **Compute the average empirical risk over all datasets:**

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

- **Observe the quantity:**

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

  - When $(\hat{\mathcal{R}} = 0, \bar{\epsilon} = 0.5) \rightarrow$ predictions correspond to random coin flips (0.5 probability to predict either class)
  - When $(\hat{\mathcal{R}} = 0.5, \bar{\epsilon} = 0) \rightarrow$ all hypotheses $h_i, i = 1, ..., M$ have zero empirical error (perfect fit to noise, not good!)
  - **Ideally we want:**
    - to be able to **separate noise from signal**
      - Meaning large $\bar{\epsilon}$, so that the model is bad at classifying random noise. $\rightarrow$ low complexity $\hat{\mathcal{R}}$.
    - to have **low empirical error on real data** - otherwise impossible to obtain low generalization error

Rademacher complexity:

- For binary classification with labels $Y = \{-1, +1\}$ **empirical Rademacher complexity** can be defined as

$$\hat{\mathcal{R}}_S(\mathcal{H}) = \frac{1}{2} E_\sigma \underbrace{(\sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{t=1}^{m} \overbrace{\sigma^i}^{\text{random true label}} \overbrace{h(x_i)}^{\text{predicted random label}})}_{\text{hypothesis with highest correlation with random label}}$$

Where:

$\sigma_i \in \{-1, +1\}$**:** are Rademacher random variables, drawn independently from uniform distribution (i.e. $Pr\{\sigma = 1\} = 0.5$)

- We can also rewrite $\hat{\mathcal{R}}_S$ **in terms of empirical error**

$$\hat{\mathcal{R}}_S = \frac{1}{2} - E_\sigma \inf_{h \in \mathcal{H}} \hat{\epsilon}(h)$$

  - Now we have Rademacher complexity in terms of expected minimum error of classifying randomly labeled data

2.3.2.1 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:

$$R(h) \leq \underbrace{\hat{R}_S(h)}_{\text{empirical risk}} + \underbrace{\hat{\mathcal{R}}_S(\mathcal{H})}_{\text{empirical Rademacher complexity}} + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}}$$

2.3.3 Vapnik-Chervonenkis dimension VS. Rademacher complexity

Note the differences between Rademacher complexity and VC dimension

- Dependency on training data

- **VC dimension:** independent → 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

- Focus
  - **VC dimension:** extreme case of realizing all labelings of the data
  - **Rademacher complexity:** measures smoothly the ability to realize random labelings

Example: Rademacher and VC bounds on a real dataset

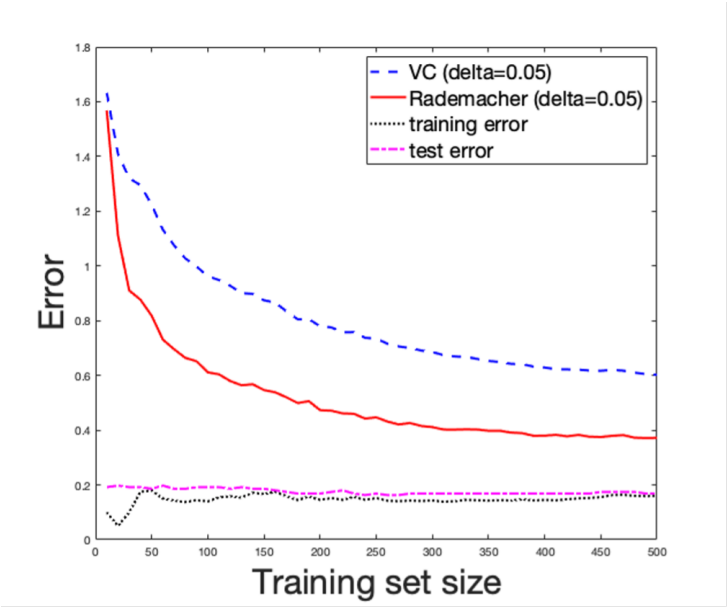


Figure 6: Rademacher and VC bounds on a real dataset

- 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)

3 Model selection

Key principle: Model selection in machine learning can be seen to implement Occam's razor:

- **Occam's razor, (principle of parsimony)** is the *problem-solving principle* that "entities should not be multiplied beyond necessity".
- **Simply:** If there are several equally correct explanations for some phenomenon, the simplest explanation is the most preferred one.

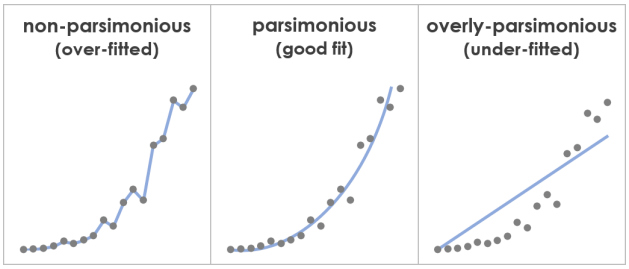


Figure 7: Illustration of parsimony in the context of ML models

- **In model selection:** captures the trade-off between generalization error (bias & variance) and model complexity.

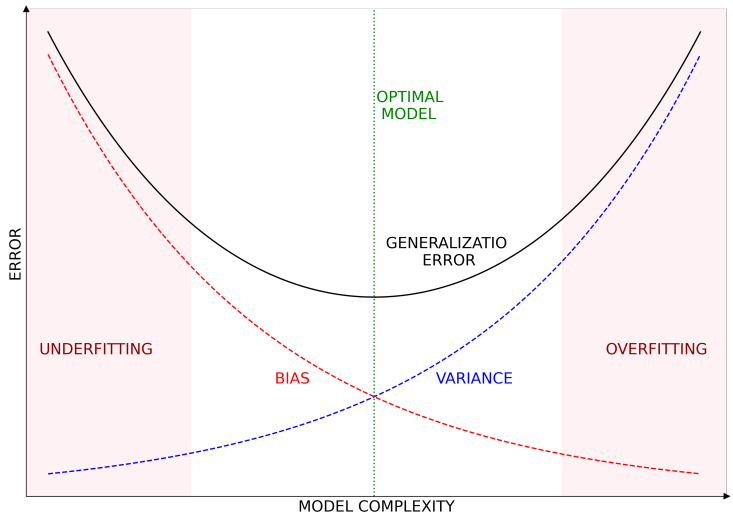


Figure 8: Illustration of parsimony in the context of ML models

Scenario – what we assume from the data:

- **So far- DETERMINISTIC SCENARIO:** The analysis so far assumed that the labels are deterministic functions of the input
- **Now- STOCHASTIC SCENARIO:** Relaxes this assumption by assuming the output is a probabilistic function of the input
  - Inputs X and outputs Y are generated by a joint probability distribution D or $X \times Y$
  - In the stochastic scenario, there **may not always exist a target concept** f that has zero generalization error $R(f) = 0$

Sources of stochasticity: stochastic dependency between input and output can arise from various sources

- Imprecision in recording the input data (e.g. measurement error), shifting our examples
- 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 of stochasticity could be characterized as adding **noise** (or **hiding signal**)

NOISE (All sources of stochasticity) and Complexity:

- **Typical effect:** make the decision boundary more complex (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.
- 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 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

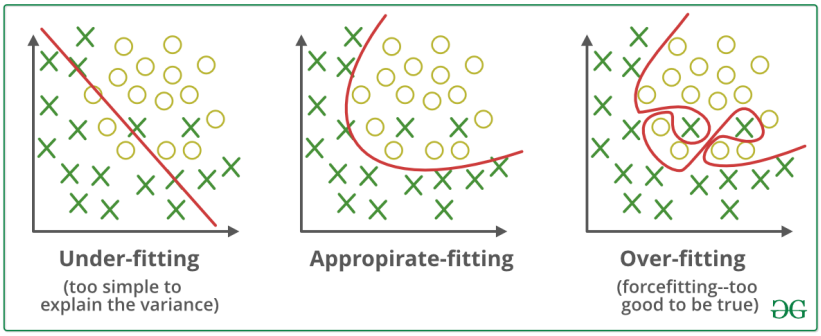


Figure 9: Illustration of under- and overfitting

Controlling complexity (2 general approaches):

1. **Hypothesis Class selection:** e.g. the maximum degree of polynomial to fit the regression model
2. **Regularization:** penalizing the use of too many parameters

Measuring complexity

- We have already looked at some measures:
  - **Number of distinct hypotheses $|\mathcal{H}|$:** works for finite $\mathcal{H}$
  - **Vapnik-Chervonenkis dimension (VCdim) (Section 2.3.1):** the maximum number of examples that can be classified in all possible ways by choosing different hypotheses $h \in \mathcal{H}$
  - **Rademacher complexity (Section 2.3.2):** measures the capability to classify after randomizing the labels
- Lots of other complexity measures and model selection methods exist (not in the scope here)

3.1 Bayes error

Bayes error is the minimum achievable error (non-zero), given a distribution D over $X \times Y$ (*stochastic scenario*), by measurable functions $h : X \to Y$

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

- Note that we cannot actually compute $R^* \to$ serves us as a theoretical measure of best possible performance

Bayes classifier is a hypothesis with $R(h) = R^*$

$$h_{Bayes}(x) = \arg\max_{y \in \{0,1\}} Pr(y|x)$$

- The average error made by the Bayes classifer at $x \in X$ is called the **noise**:
$$noise(x) = \min(Pr(1|x), Pr(0|x))$$
  - Its expectation $E(noise(x)) = R^*$ is the Bayes error
- Similarly to the Bayer error, Bayes classifier is a theoretical tool of best possible classifier, not something we can compute in practice

3.2 Decomposing the error of a hypothesis

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

$$R(h) - R^* = \underbrace{\underbrace{\epsilon_{estimation}}_{\text{variance}} + \underbrace{\epsilon_{approximation}}_{\text{bias}}}_{\text{bias-variance decomposition}}$$

- $\epsilon_{estimation}(variance) = R(h) - R(h^*)$ is the excess generalization error h has over the optimal hypothesis $h^* = \arg\min_{h' \in \mathcal{H}} R(h')$ in the hypothesis class $\mathcal{H}$
- $\epsilon_{approximation}(bias) = R(h^*) - R^* =$ 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)

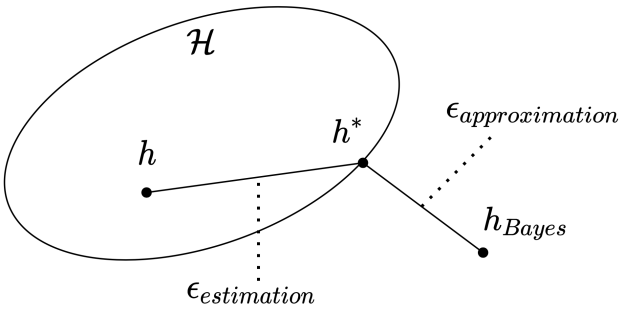


Figure 10: Illustration depicting the concepts

- 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 $R(h^*) - R(h_{Bayes})$
- **Challenge for model selection:** We can bound the estimation error by generalization bounds but we cannot do the same for the approximation error as R^* remains unknown to us.

3.3 Strategies for Model Selection

Initially choose a very complex hypothesis class with zero or very low empirical risk $\mathcal{H}$

Assume the class can be decomposed into a union of increasingly complex hypothesis classes $\mathcal{H} = \bigcup_{\gamma \in \Gamma} \mathcal{H}_\gamma$

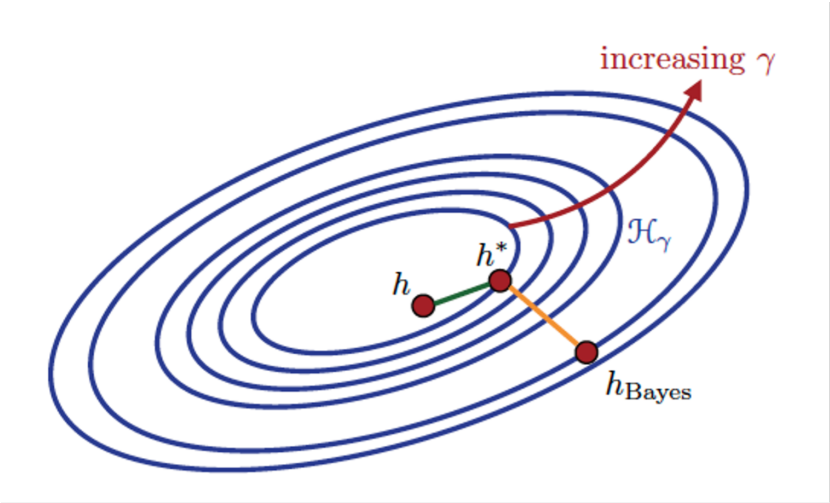


Figure 11: Illustration of hypothesis classes $\mathcal{H}$ being decomposed into a union of increasingly complex hypothesis classes

- The complexity increases by parameter γ e.g.
  - γ = degree of a polynomial function
  - γ = size of a neural network
  - γ = norm of weights of a linear regression model

Strategy:

- **The model selection problem then entails** choosing a parameter value λ^* that gives the best generalization performance

Trade-Off: increasing the complexity of the hypothesis class

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

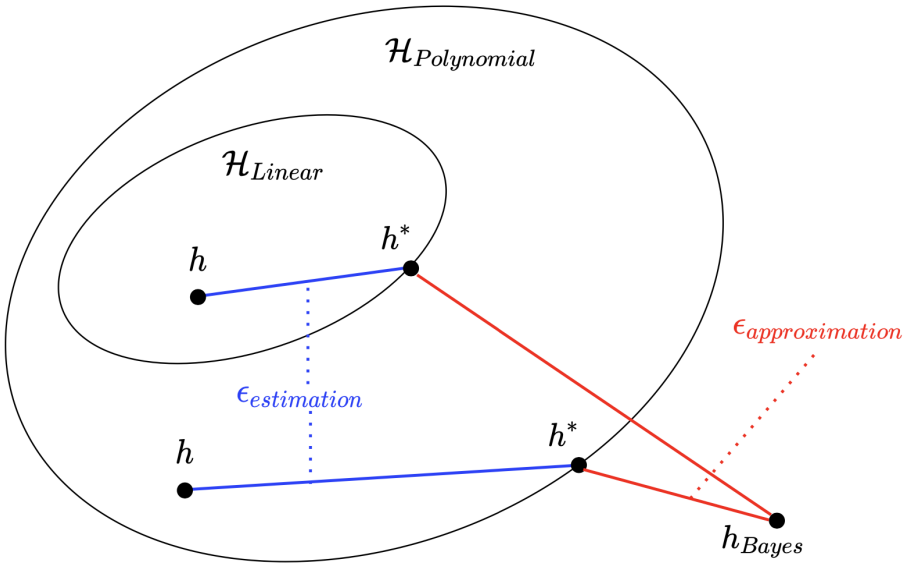


Figure 12: Example case of two hypothesis h from decomposed hypothesis classes $\mathcal{H}$, and their relation to h_{Bayes}

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

3.3.1 Structural Risk Minimization (SRM)

What: An *inductive principle* of use in ML.

- **Assumes** a countable union of hypothesis classes $\mathcal{H} = \bigcup_{k \geq 1} \mathcal{H}_k$, indexed by complexity parameter k
- **Model selection task:** Select the optimal index k^* and the hypothesis $h \in \mathcal{H}_{k^*}$ that gives the best generalization bound

Why: It aims to minimize the excess risk $R(h) - R(h_{Bayes})$.

How: By bounding $R(h)$.

- **Generalization bound for SRM (Mohri et al. 2018) in PAC-framework:** for any $\delta > 0$ with probability at least $1 - \delta$ over the

draw of a sample S of size m, we have for all $k \geq 1$ and $h \in \mathcal{H}_k$

$$R(h) \leq \underbrace{\hat{R}_S(h)}_{\substack{\text{empirical error} \\ \text{on training set}}} + \underbrace{R_m(\mathcal{H}_k(h))}_{\substack{\text{Rademacher complexity of} \\ \text{least complex hypothesis class} \\ (\min k \text{ for } \mathcal{H}) \text{ where } h \text{ belongs}}} + \underbrace{\sqrt{\frac{\log k}{m}}}_{\substack{\text{penalty for not} \\ \text{knowing the correct } \mathcal{H} \\ (\mathcal{H} \text{ selection})}} + \sqrt{\frac{\log \frac{2}{\delta}}{2m}}$$

– See resemblance to 2.3.2.1.

SRM model selection algorithm: Given a training sample S, picks the index k and $h_S^{SRM} \in \mathcal{H}_k$ that minimizes

$$h_S^{SRM} = \underset{k \geq 1, h \in \mathcal{H}_k}{\arg\min}\, \hat{R}_S(h) + R_m(h_k) + \sqrt{\frac{\log k}{m}}$$

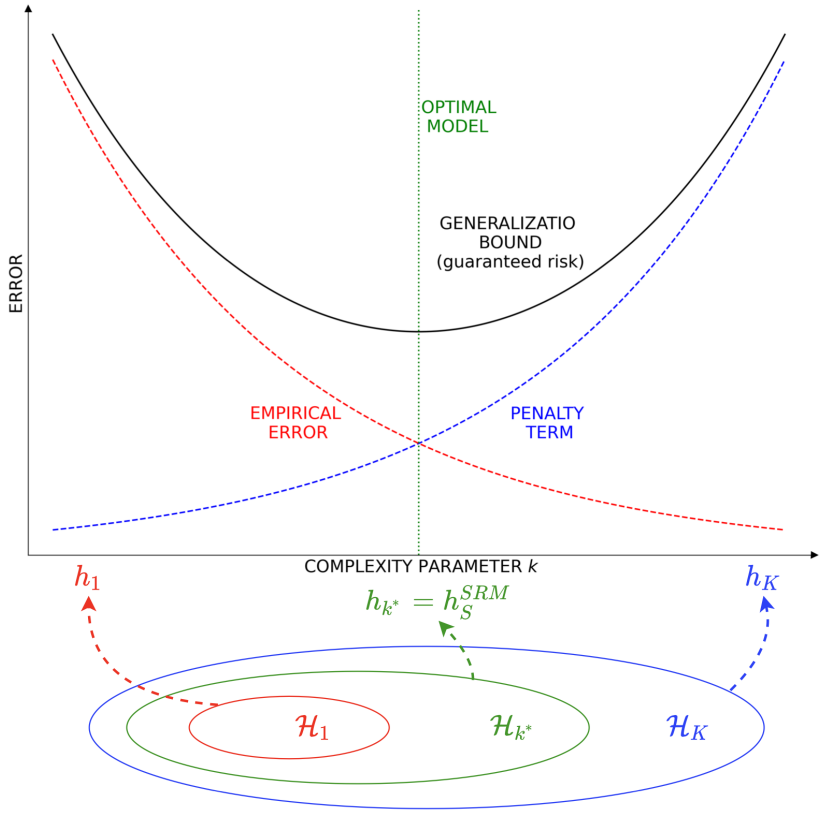


Figure 13: Illustration of Structural Risk Minimization (SRM) with Rademacher complexity

- **Note** that this may be a computationally difficult task:
  - Requires finding the hypothesis that minimizes training error for each hypothesis class separately
  - Obtaining the empirical Rademacher complexity generally requires simulation with multiple datasets with randomized labels for each hypothesis class

Pros & Cons of SRM:

\+ benefits from strong learning guarantees

− restrictive assumption of a countable decomposition of the hypothesis class

− large computational price, especially when a large number of hypothesis classes $\mathcal{H}_k$ has to be processed

3.3.2 Regularization-based algorithms

What: An alternative model selection approach to SRM.

- **Assumes** a very complex family $\mathcal{H} = \bigcup_{\gamma \geq 0} \mathcal{H}_\gamma$ of uncountable union of nested hypothesis classes $\mathcal{H}_\gamma$

How:

This extension to the SRM method would then ask to minimize:

$$= \underset{\gamma \geq 0, h \in \mathcal{H}_\gamma}{\arg\min} \hat{R}_S(h) + R_m(h_\gamma) + \sqrt{\frac{\log \gamma}{m}}$$

- This problem seems to require evaluating the Rademacher complexity of an uncountably infinite number of hypothesis classes $\mathcal{H}_\gamma \rightarrow$ Need efficient algorithms to do this

However this kind of model selection is efficient for the class of **linear functions** $x \rightarrow w^T x$

- The classes as parametrized by the norm $||w||$ of the weight vector bounded by γ:

$$\mathcal{H}_\gamma = \{x \to w^T x : ||w|| \leq \gamma\}$$

  - The norm $||w||$ is typically eiteher:
    - L^2 **norm (Also called Euclidean norm or 2-norm):**
      - used e.g. in support vector machines and ridge regression

$$||w||_2 = \sqrt{\sum_{j=1}^{n} w_j^2}$$

    - L^1 **norm (Also called Manhattan norm or 1-norm):**
      - used e.g. in LASSO regression

$$||w||_1 = \sum_{j=1}^{n} |w_j|$$

Computational shortcut: for L^2-norm: the empirical Rademacher complexity of this class can be bounded analytically!

- Let $S \subset \{x \mid \ ||x|| \leq r\}$ be a sample of size m and let $\mathcal{H}_\gamma = \{x \to w^T x : ||w||_2 \leq \gamma\}$. Then

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

Where:

r : upped bound for length of the input vector $||x||$

γ : upped bound for length of the weight vector $||w||_2$

  - Thus the **Rademacher complexity** depends linearly on the upper bound γ norm of the weight vector, as r and m are constant for any fixed training set
  - So we can use $||w||$ as a efficiently computable upper bound of $hat R_S(\mathcal{H}_\gamma)$

Regularized learning problem: minimize

$$\arg\min_{h \in \mathcal{H}} \hat{R}_S(h) + \lambda \Omega(h)$$

Where:

$\hat{R}_S(h)$: empirical error

$\Omega(h)$: regularization term which increases when the complexity of the hypothesis class increases

λ : regularization parameter, which is usually set by cross-validation

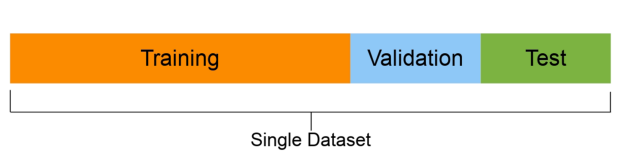
3.3.3 Model selection using a validation set

What: Using the given dataset for *empirical model selection*.

When: *If* the algorihm has input parameters (hyperparameters) that define/affect the model complexity.

How:

1. **Split** the data into training, validation and test sets



2. Use **grid search** to find the hyperparameters combination that gives the best performance on the validation set
   - In its **basic form** it goes through all combinations of parameter values, given a set of candidate values for each parameter
     - **With many hyperparameters** the search becomes computationally hard due to exponentially exploding search space
3. **Retrain a final model** using the *optimal parameter combination*, use both the training and validation data for training
4. **Evaluate the performance** of the final model on the test set

Sets:

- **Training set:** used to fit (optimize) the parameters (weights) of the model.
- **Validation set:** used to avoid overfitting. It provides an unbiased evaluation of a model fit on the training data set while tuning the model's hyperparameters.
- **Test set:** used to obtain reliable performance estimate.

Deciding on set sizes:

- 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: is a process that tries to ensure similar class distributions across the different sets.

- **Motivation:** Class distributions of the training and validation sets should be as similar to each another as possible, otherwise there will be extra unwanted variance
- **Simple approach:** 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.

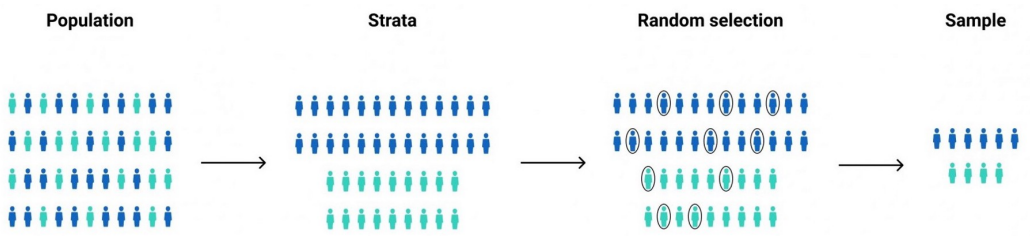


Figure 14: Illustration of stratified sampling

3.3.4 Cross-validation

Why: 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)

How: Fighting the randomnes 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.

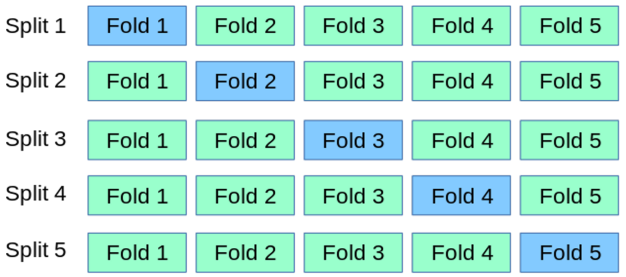
Various cross-validation schemes can be used to tackle the variance of the performance estimates.

n-Fold Cross-Validation

1. First **set a side** a separate test set
2. Then **split the remaining data randomly** into n equal-sized parts (or folds)

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

3. 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



Leave-one-out Cross-Validation (LOO)

- **How:** given a dataset of m examples, only one example is left out as the validation set and training uses the $m - 1$ examples.
- **Has good theoretical properties:** This gives an unbiased estimate of the average generalization error over samples of size $m - 1$ (Mohri, et al. 2018, Theorem 5.4.)
- However, **if** m **is large**, it is comptationally demanding to compute

Nested cross-validation

- **Motivation:** only using a **single test set** may result in unwanted variation → Nested cross-validation solves this problem by using two cross-validation loops
- **Process:**
  1. The dataset is initially divided into n outer folds
  2. **Outer loop** uses **1 fold** at a time as a **test set**,
  3. The **remaining folds** of the data is used in the **inner fold**
     - **Inner loop** splits the remaining exampls into k folds, 1 fold for validation, and $k - 1$ for training

4. The average performance over the n test sets is computed as the **final performance estimate**

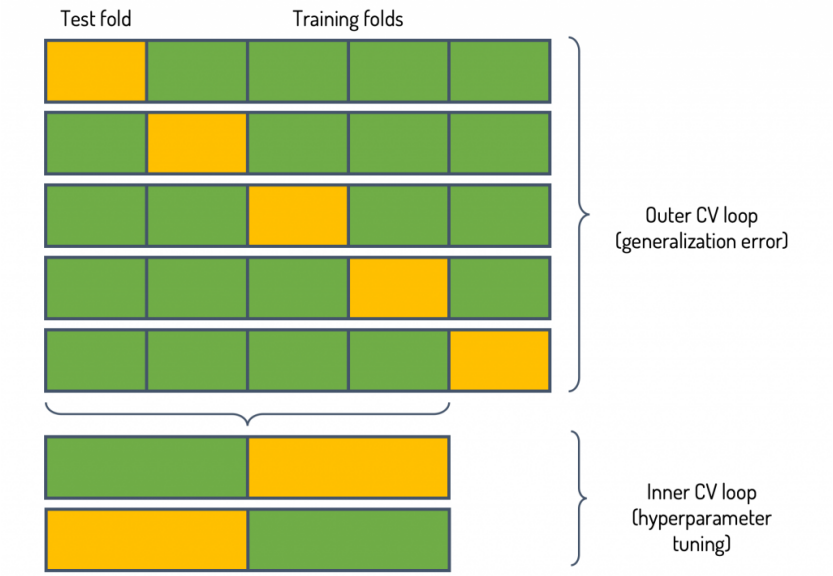


Figure 15: Illustration of 5×2 nested cross-validation

Part II

Algorithms and Models

4 Loss functions

At its core, a loss function is a measure of how good your prediction model does in terms of being able to predict the expected outcome (or value).

Loss functions measure how far an estimated value is from its true value. A loss function maps decisions to their associated costs. Loss functions are not fixed, they change depending on the task in hand and the goal to be met.

Convexity is more crucial than the differentiability. Differentiability is also very usefull.

4.1 Minimizing loss functions

4.1.1 Gradient of a Function (Calculus Refresh)

Derivative is a way to show **instantaneous rate of change (slope of the tangent line at a point on a graph):** that is, the amount by which a function is changing at one given point.

- Common notations:

  $\frac{dy}{dx}$, meaning the difference in y divided by the difference in x

  $f'(x)$, meaning the derivative of function f at point x

- Important in optimization because the **zero derivatives** might indicate a minimum, maximum, or saddle point.

Gradient of a scalar-valued differentiable function f of several variables is the vector field (or vector-valued function) ∇f whose value at a point p is the vector whose components are the partial derivatives of f at p. More generally, it is a **vector that points in the direction in which the function grows the fastest**. value at a point

- Common notations:

  ∇f, meaning the gradient of a function f

  $-\nabla f$, meaning the negative gradient of a function f

When working with gradient descent, you're interested in the direction of the fastest decrease in the cost function. This direction is determined by the negative gradient, $-\nabla f$.

4.2 Loss functions for classification

See more here.

Zero-One loss

- It assigns 0 to loss for a correct classification and 1 for an incorrect classification.
- **Specs:**
  - The main source of difficulty is the "step function" shape of the zero-one loss function
    - It is non-differentiable → cannot optimize using gradient approaches
    - It is non-convex → optimizer susceptible to fall in local minima

A convex loss function makes it easier to find a global optimum and to know when one is reached. There are multiple surrogate losses that are convex and differentiable upper bounds to zero-one loss.

Hinge loss - used in Support vector machines

- **Specs:**
  - It is non-differentiable, but has a subgradient with respect to model parameters w

Exponential loss - used in Boosting

Logistic loss - used in Logistic regression

5 Linear classification

What: Performing classification based on the value of a linear combination of weights and features.

Hypothesis class:

$$\mathcal{H} = \{x \to \text{sgn}\left(\sum_{j=1}^{d} w_j x_j + w_0\right) \mid w \in \mathbb{R}^d, w_0 \in \mathbb{R}\}$$

- Consists of **linear classifiers** that map each example in one of the classes:

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

  - $\text{sgn}(a)$ is the *sign function*. E.g. for binary classification:

$$\text{sgn}(a) = \begin{cases} +1 & a \geq 0 \\ -1 & a < 0 \end{cases}$$

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

5.1 Learning linear classifiers

Change of representation

For presentation is is convenient to subsume term w_0 into the weight vector and augment all inputs with a constant 1:

$$w \Leftarrow \begin{bmatrix} w \\ w_0 \end{bmatrix} \quad x \Leftarrow \begin{bmatrix} x \\ 1 \end{bmatrix}$$

The models have the same value for the discriminant:

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

Checking for prediction errors:

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

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

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

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

Margin: the minimal distance of any training point to the hyperplane (decision boundary).

- **Geometric margin** for an example x is given by $\gamma(x) = yg(x)/||w||$
- **Functional margin** (unnormalized version) for an example x is given by $\gamma(x) = yg(x)$

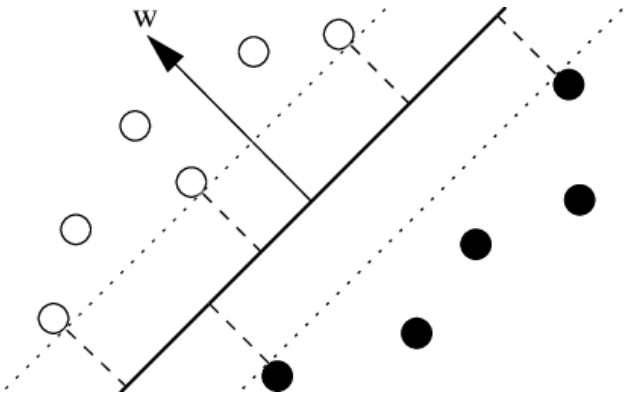


Figure 16: Example where margin is the distance between the dotted lines and the thick line

5.2 Perceptron

What: is a simple algorithm to train linear classifiers on linearly separable data.

How: learns a hyperplane separating two classes:

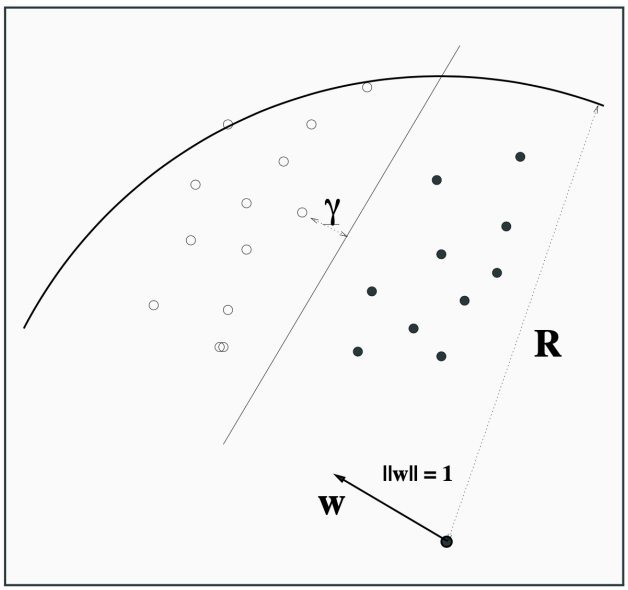
$$g(x) = w^T x$$

- It processes incrementally a set of training examples
  - **At each step**, it finds a training example 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: $w^{(t+1)} \leftarrow w^{(t)} + y_i x_i$
  - This process is **continued until** incorrectly predicted training examples are not found

Convergence:

- Only if the two classes are **linearly separable**
- Doesn't stop if **non-separable training set**
- **Theorem (Novikoff):**

- Will stop after at most $t \leq (\frac{2R}{\gamma})^2$ iterations
  - γ: 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

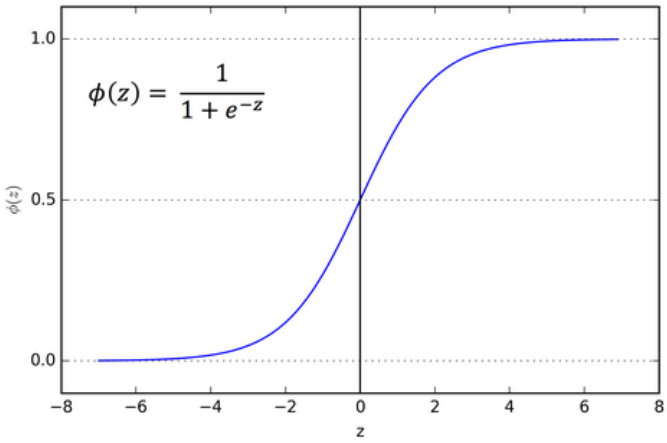


5.3 Logistic Regression

What: is a classification method that can be interpreted as maximizing odds ratios of conditional class probabilities.

- It gets its name from the **logistic function** that maps a real valued input z onto the interval $0 < \phi_{logistic}(z) < 1$

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


$\phi(z) = \frac{1}{1+e^{-z}}$
1.0
0.5
0.0
$\phi(z)$
−8
−6
−4
−2
0
2
4
6
8
z

6 Support Vector Machines (SVMs)

What: classification methods based on the principle of margin maximization. Commonly Performing nonlinear classification with kernel methods.

How: SVMs can be efficiently optimized using Stochastic gradient techniques specially developed for piecewise differentiable functions, such as the Hinge loss.

Main idea:

1. Start with **data** in a relatively low dimension
2. Move the **data** into a higher dimension (using Kernel methods)
3. Find a Support Vector Classifier that separates the higher dimensional data into groups

6.1 Maximum margin hyperplane

What: Hyperplane $\mathbf{w}^T x = 0$ that lies furthest away from the training data (maximizing the minimum margin of the training examples):

$$\begin{gathered} \text{Maximize } \gamma \\ \text{w.r.t. variables } \mathbf{w} \in \mathbb{R}^d \\ \text{Subject to } \frac{y_i \mathbf{w}^T \mathbf{x}_i}{||\mathbf{w}||} \geq \gamma, \text{ for all } i = 1, ..., m \end{gathered}$$

Why: Support vector machines (SVM) are based on this principle.

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

Process: How to maximize the margin:

- Optimizing γ does not give us a unique optimal weight vector $\mathbf{w}^*$
- To get an unique answer, let us multiply the constraint on the geometric margin by $||\mathbf{w}||$ to obtain a an equivalent constraint on the functional margin

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

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

Hard margin Support Vector Machine (SVM) solves the margin maximization as follows:

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$
$$\text{w.r.t. variables } \mathbf{w} \in \mathbb{R}^d$$
$$\text{Subject to } y_i \mathbf{w}^T \mathbf{x}_i \geq 1, \text{ for all } i = 1, ..., m$$

- 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

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

6.2 Soft-Margin SVM

What: relaxed margin constraints

Why: allows non-separable data

How: To allow non-separable data, we allow the functional margin $y_i \mathbf{w}^T \mathbf{x}_i$ of some data points $\mathbf{x}_i$ to be smaller than 1 by a slack variable $\xi \geq 0$.

- Relaxed margin constraint:

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

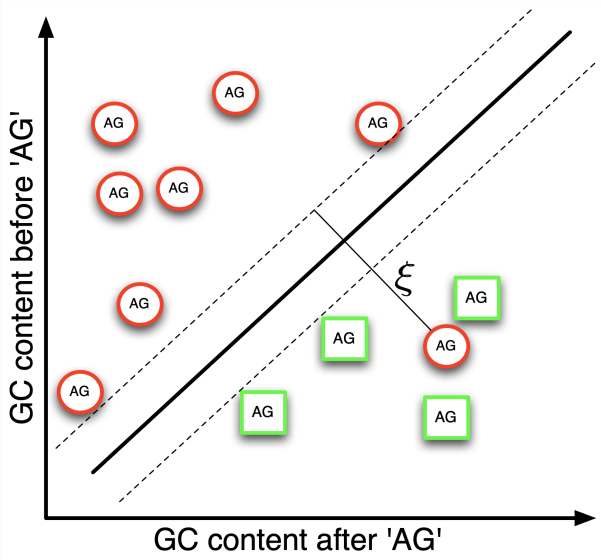


Figure 17: Examples of slack variables for soft margin classifiers.

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

Soft margin Support Vector Machine (SVM) solves the margin maximization as follows:

$$
\begin{array}{ll}
\text{minimize} & \frac{1}{2}||\mathbf{w}||^2 + \overbrace{\frac{C}{m}\sum_{i=1}^{m}\xi_i}^{\text{avg. slack (penalty)}} \\
\text{w.r.t. variables} & \mathbf{w}, \xi \\
\text{subject to} & y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i, \text{ for all } i = 1, ..., m \\
& \xi_i \geq 0, \text{ for all } i = 1, ..., m
\end{array}
$$

- The coefficient $C > 0$ controls the balance between model complexity (low C) and empirical error (high C)

We can rewrite the optimization problem in terms of Hinge loss as

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

- The parameter $\lambda = \frac{1}{C}$ controls the balance between model complexity (low C) and empirical error (high C).

6.2.1 Loss functions: Hinge loss

We can interpret the soft-margin SVM in terms of minimization of a loss function.

- Observe the relaxed margin constraint:

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

- We get so called **Hinge loss** by rearranging the same as:

$$
\begin{aligned}
\xi_i &\geq 1 - y_i \mathbf{w}^T \mathbf{x}_i \ , \quad (\xi \geq 0) \\
\xi_i &\geq \max(1 - y_i \mathbf{w}^T \mathbf{x}_i, 0) \\
L_{Hinge}(y, \mathbf{w}^T, \mathbf{x}) &\geq \max(1 - y_i \mathbf{w}^T \mathbf{x}_i, 0)
\end{aligned}
$$

– For $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$:

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

The soft-margin SVM corresponds to a Quadratic program (QP). A QP is a convex optimization problem (with a unique optimum). When data is small, QP solvers in optimization libraries can be used to solve the soft-margin SVM problem.

On big data, a stochastic gradient descent procedure is a good option. **Rewrite the regularized learning problem as an average:**

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

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

- We can differentiate the linear pieces of the loss separately:

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

- We get:

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

  – At $\mathbf{w}^T\mathbf{x}_i = 1$, the function is not differentiable but we can choose 0 as the value, since the Hinge loss is zero so no update is needed to decrease loss.

- To find the update direction we express $J_i(w)$ as a piecewise differentiable function:

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

- Computing the derivatives piecewise gives the gradient:

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

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

Stochastic gradient descent algorithm for soft-margin SVM

Initialize: weights $\mathbf{w}$ (e.g., to 0)
Repeat:

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

Compute the update direction corresponding to the training example:

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

Determine a stepsize η

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

until stopping criterion satisfied
Output: weights $\mathbf{w}$

6.3 Dual Soft-Margin SVM

What: Dual representation of SVM allows the use of kernel functions.

Why: Enables nonlinear classification.

How: 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 $\alpha_i \geq 0$

- if x_i is a support vector, $\alpha_i > 0$
- else $\alpha_i = 0$

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

$$\begin{array}{ll} \text{maximize} & OBJ(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \\ \text{w.r.t. variables} & \alpha \in \mathbb{R}^m \\ \text{subject to} & 0 \leq \alpha_i \leq C/m, \text{ for all } i = 1, ..., m \end{array}$$

- It is a QP with variables α_i, again with a unique optimum
- The data only appears through the kernel function $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$

---

7 Kernel methods

Key characteristics:

- **Embedding:** Inputs $\mathbf{x} \in X$ from some input space X are embedded into a *feature space* F via a feature map $\phi : X \to F$.
  - ϕ maybe highly non-linear and F potentially very high-dimensional vector space.
- **Linear models:** are built for the 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-steps the efficiency problems of high-dimensionality.
- **Regularized learning:** To avoid overfitting, large feature weights are penalized, separation by large margin is favoured.

---

8 Neural Networks (NNs)

8.1 Perceptron (as building block of NNs)

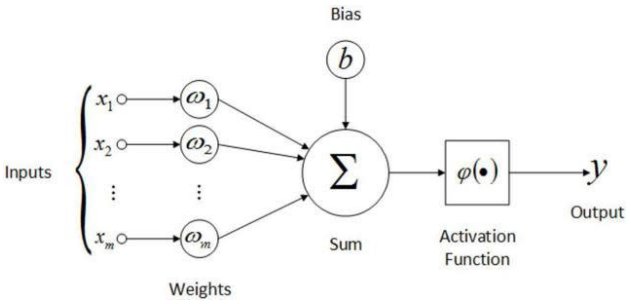


Figure 18: Illustration of a single leayer perceptron.

8.2 Perceptrons as Logical Operators

Motivation: Showcasing the expressive power of neural networks.

AND Perceptron:

- Set the bias $w_0 = -1.5$ and the weights $w_1 = w_2 = 1$
- Now the function $w_1 x_1 + w_2 x_2 + w_0 > 0$ 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

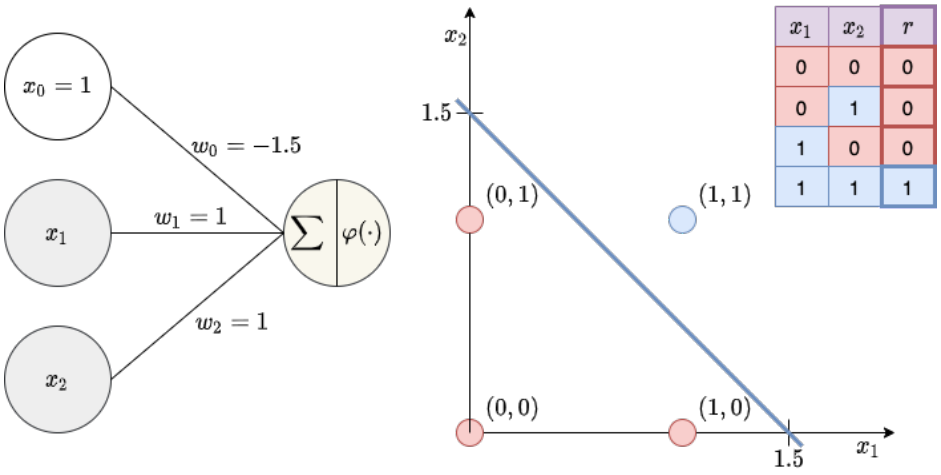


Figure 19: Illustration of AND perceptron.

OR Perceptron:

- 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

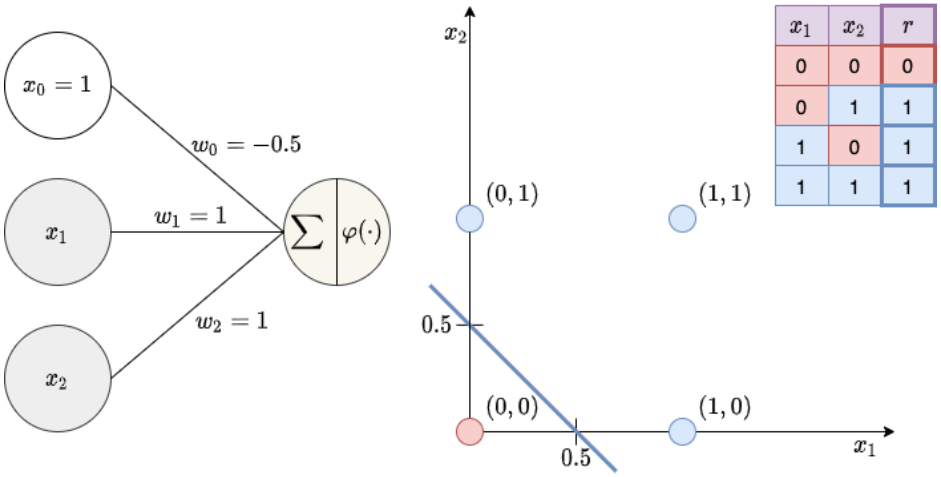


Figure 20: Illustration of OR perceptron.

XOR Multi-Layer Perceptron (MLP): The exclusive OR, or XOR operator cannot be represented by a single-leyer perceptron, as XOR function is not linearly separable. XOR can be computed by a Multi-Layer Perceptron.

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

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

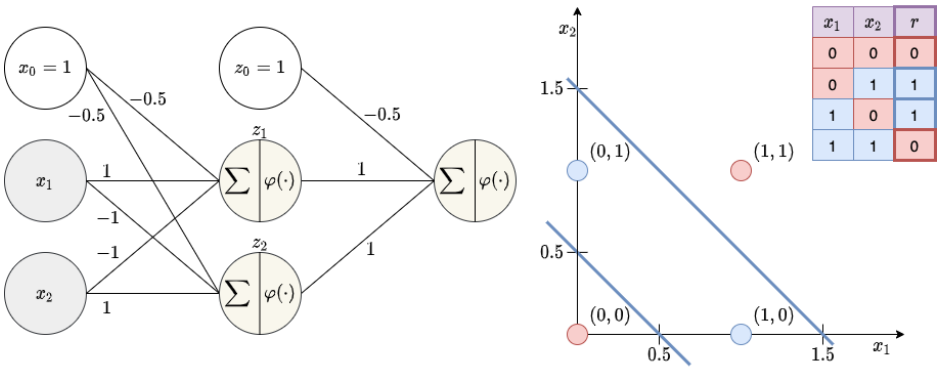


Figure 21: Illustration of XOR perceptron.

Part III

Additional learning models

9 Feature Engineering and Selection

Why: It can be argued that good data representation is actually more important than the choice of the learning algorithm.

How: There are variety of **feature engineering 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

9.1 Feature transformation

Why: 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

Objectives: better error rates **or** faster optimization

How: Some **common feature transformations:** (let $f = (f_1, ..., f_m)$ denote the values of a single variable/feature in the dataset)

- **Centering:**

$$f'_i = f_i - \bar{f}$$

Where: $\bar{f}$**:** mean of variable

  - **Rationale:** learning algorithms generally learn from the variance of the data, not the mean. Centering makes the variance more obvious.

- **Standardization (Z-score Normalization):**

$$f'_i = \frac{f_i - \bar{f}}{\sqrt{\text{var}(f)}}$$

Where: $\text{var}(f)$: 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 (min-max scaling):**

$$f'_i = \frac{f_i - f_{min}}{f_{max} - f_{min}}$$

  - **Rationale:** unit range $[0, 1]$ is useful if the variable's relative position in the observed range is important.

- **Clipping:**

$$f'_i = \text{sgn}(f_i)\min(b, |f_i|), \text{ for some threshold } b > 0$$

Where: $\text{sgn}(f_i)$: sign function

  - **Rationale:** if certain large values are known to be non-informative, clipping may make learning easier (clipping small values: use max instead of min).

- **Logarithmic transformation:**

$$f'_i = \log(b + f_i), \text{ for some constant } b > 0$$

  - **Rationale:** log-transform can be used to emphasize small differences between small values (e.g. 0 vs. 1) if they are more important than small differences between large values (e.g. 1000 vs. 1001).

- **Sigmoid transformation:**

$$f'_i = \frac{1}{1 + \exp(bf_i)}$$

  - **Rationale:** Compresses the high absolute values heavily, "soft version" of clipping.

- **Normalization of feature vectors (Unit-Length Scaling):**

$$x' = \frac{x}{||x||}$$

Where: x: is the feature vector
$||x|| =$ Euclidean length of the feature vector

  - **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

9.2 Feature selection

Why: Potential benefits:

- **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

How: Exhaustive search approach goes through all $2^d - 1$ feature subsets and would give us the optimal feature subset, however it is prohibitively expensive unless d (the set of features) is very small.

In general, **feature selection approaches** aim to avoid the exponential complexity of checking each feature subset:

- **Variable ranking ("filtering" approach):** 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

9.2.1 Variable ranking ("filtering" approach)

Pros and cons:

+ 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.

- limited by two issues:
  - **Features** that are not correlating with the output **(not selected)** 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

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 θ (k or θ generally decided by the user)

9.2.1.1 Regression-based scoring criterion : Pearson correlation

- Assume a dataset $S = \{(x_i, y_i)\}_{i=1}^m$, where $x_i = (x_{i1}, ..., x_{id})^T$, and $y_i \in R$
- (Empirical) Pearson correlation of the j'th feature and the output is given by

$$r_j = \frac{\sum_{i=1}^m (x_{ij} - \bar{x}_j)(y_i - \bar{y})}{\sqrt{\sum_i (x_{ij} - \bar{x}_j)^2} \sqrt{\sum_i (y_i - \bar{y})^2}}$$

Where:

$$\bar{x}_j = \frac{1}{m} \sum_{i=1}^m x_{ij} \text{ denotes the mean of the } j\text{'th feature in the dataset}$$

$$\bar{y} = \frac{1}{m} \sum_{i=1}^m y_i \text{ denotes the mean of the output variable}$$

  - r_j ranges from $+1$ (perfect correlation) to -1 (perfect anti-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

$$r_j = \arg\min_{r \in \mathbb{R}} \frac{1}{m} \sum_{i=1}^m (r x_{ij} + b - y_i)^2$$

- For feature scoring, we use $s_j = r_j^2$ to allow selection of both anti-correlated and correlated features
  - 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

9.2.1.2 Classification-based scoring criterion

- 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

$$[a_j, \theta_j] = \underset{a \in \{-1,+1\}, \theta \in \mathbb{R}}{\arg\min} \sum_{i=1}^{m} 1_{\text{sgn}(ax_{ij}+\theta) \neq y_i}$$

- For feature scoring, we can use the accuracy (or some other evaluation metrics for classification):

$$r_j = 1 - \frac{1}{m} \sum_{i=1}^{m} 1_{\text{sgn}(ax_{ij}+\theta) \neq y_i}$$

9.2.2 Variable subset selection

What (recap): 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.

9.2.2.1 Wrapper approach

How: 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:
  1. Generate a variable subset (by some fixed procedure)
  2. Evaluate the subset by training a model with the subset

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

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.)

9.2.2.1.1 Greedy forward selection

How:

- We **add a 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 $S = \{(x_i, y_i)\}_{i=1}^{m}$
Input: Maximum number of variables K
Initialize: $J \leftarrow \emptyset, s_J$ (score) $\leftarrow 0; k \leftarrow 0$
Repeat:

for $j \in \{1, ..., d\} \setminus J$ (all except the ones that are in J already):

Train a model $f_{J \cup j}$ with input variable $J \cup j$

$s_{J \cup j} \leftarrow$ accuracy of the model $f_{J \cup j}$ (training set or validation set)

Select the best variable: $j^* \leftarrow \arg\max_{j \in \{1,...,d\} \setminus J} s_{J \cup j}$

if $s_{J \cup j^*} > s_J$**:**

$J \leftarrow J \cup j^*$

$k \leftarrow k + 1$

else:

stop $\leftarrow TRUE$

end if:

$k = K$ or stop

Output: A subset of selected variables $J \subset \{1, ..., d\}$

9.2.2.1.2 Backward elimination

How:

- We **starts from the full set of input variables**
- 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

Why: It is argued that backward elimination is more effective in finding good variable subsets than forward selection

Backward elimination pseudo-code

Input: Training set $S = \{(x_i, y_i)\}_{i=1}^m$
Input: Validation set $V = \{(x_i, y_i)\}_{i=1}^n$
Initialize: $J \leftarrow \{1, ..., d\}$, $s_{\max}$ (score) $= 0$; $J_{\max} = J$
Repeat:

for $j \in J$**:**

Train a model $f_{J\setminus j}$ with input variable excluding j

$s_{J\setminus j} \leftarrow$ accuracy of the model $f_{J\setminus j}$ on the validation set

Variable whose elimination gives the best model: $j^* \leftarrow \arg\max_{j\in J} s_{J\setminus j}$

$J \leftarrow J \setminus j^*$

Keep track of the best model:

if $s_J > s_{\max}$**:**

$s_{\max} = s_J$

$J_{\max} = J$

until $J = \emptyset$

Output: A subset of selected variables $J_{\max} \subset \{1, ..., d\}$

9.2.3 Embedded methods

How: 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
- **Sparse modelling methods:** focused on penalizing the weights with sparsity inducing norms

9.2.3.1 Sparse modelling

What: Roughly speaking, a vector x is **sparse** if it contains many zeros.

How: ℓ_1 norm $||x||_1$ promotes sparsity:

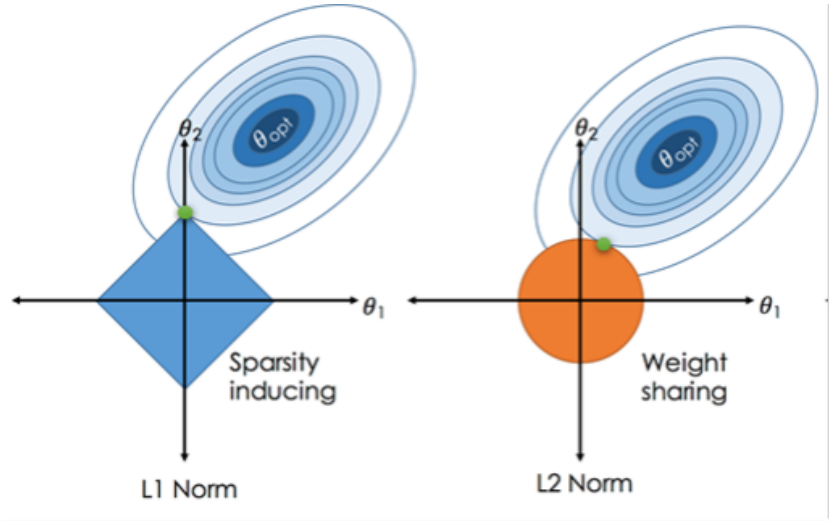


Figure 22: Visualizing the sparsity of L1 norm.

Sparsity-inducing norms: ℓ_p

- But is some other p also possible?
- For general p, the ℓ_p norm is given by

$$||w||_p = (\sum_{j=1}^{d} |w_j|^p)^{1/p}$$

- For $p < 1$ the constraint region $||w||_p \leq 1$ becomes non-convex with "spikes" extending towards the corners: will give sparsity but hard to optimize

- For $p > 1$ the 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

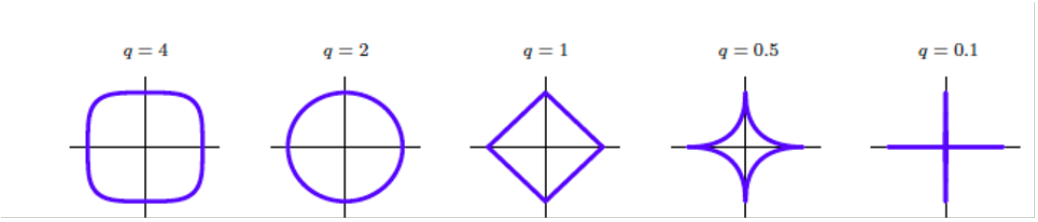


Figure 23: Visualizing different norms.

Sparse learning problems:

- Combining ℓ_1 regularisation with different loss functions gives sparse variants of well-known models
  - Squared loss ⇒ Sparse regression, LASSO:

$$\min_w \frac{1}{m} \sum_{i=1}^{m} L_{sq}(w^T x_i, y_i) + \lambda ||w||_1$$

  - Hinge loss ⇒ Sparse SVM:

$$\min_w \frac{1}{m} \sum_{i=1}^{m} L_{Hinge}(w^T x_i, y_i) + \lambda ||w||_1$$

  - Logistic loss ⇒ Sparse logistic regression:

$$\min_w \frac{1}{m} \sum_{i=1}^{m} L_{Logistic}(w^T x_i, y_i) + \lambda ||w||_1$$

9.2.4 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"

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

---

10 Multi-class Classification

What: the problem of classifying instances into one of three or more classes.

How:

- Given a training dataset $\{(x_i, y_i)\}_{i=1}^m, (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$
- Outputs belongs to a set of possible classes or labels:

$$y_i \in \mathcal{Y} = \{1, 2, ..., k\}$$

- We aim learning a function $f : X \to \mathcal{Y}$

multi-class classification vs. multi-label classification:

- In **multi-class classification**, one of the labels is considered to be the correct label, the other ones incorrect
- 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 y_i is a k-dimensional vector

Two basic strategies to solve the problem:

1. **Aggregated methods using multiple binary classifiers:**
   - One-versus-all (OVA) approach: Separate each class from all the others
   - One-versus-one (OVO) or all-pairs approach: Separate each class pair from each other
   - Error-correcting output codes (ECOC) 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

10.1 One-versus-All (OVA) Classification

Pros and cons:

+ Simple to implement and thus popular
+ Training is relatively efficient with $O(kt)$ time where t is the time to train a single binary classifier, if k (# of classes) is not too large
- May suffer from class imbalance of the training sets for a given class ℓ: there may be a low number of positive examples and a high number of negative examples per class
- In general suffers from a calibration problem: the scores $f_\ell(x)$ returned by the individual classifiers may not be comparable
- Doesn't always produce the optimal empirical error rate for the dataset

How:

- Given a training dataset $\{(x_i, y_i)\}_{i=1}^m, (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$
- If we have $k > 2$ classes we will train k binary hypotheses $h_1, ..., h_k$; $h_\ell : X \to \{+1, -1\}$
- For training the ℓth 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

OVA prediction:

- In general, there may be more than one class ℓ for which $h_\ell(x) = +1$

- Some arbitrary tie-breaking could be used, e.g. predict the class with the smallest index ℓ
- Better results can be obtained if the hypotheses also provide some real-valued score $f_\ell(x) \in \mathbb{R}$ (confidence, margin, etc.) for the label to be ℓ.

- In that case, we can choose the label with the highest score

$$h(x) = \arg\max_\ell f_\ell(x)$$

  - E.g. with linear models $h_\ell(x) = \text{sgn}(w_\ell^T x)$ using the margin of the example is a natural choice $f_\ell(x) = w_\ell^T x$

OVA training pseudo-code

Input: Training set $S = \{(x_i, y_i)\}_{i=1}^m, x_i \in X, y_i \in \mathcal{Y} = 1, ..., k$
for $\ell \in \{1, ..., k\}$**:**

Generate training dataset with surrogate labels: $\{(x_i, \tilde{y}_i^{(\ell)})\}_{i=1}^m$

Train a binary hypothesis $h_\ell : X \to \{-1, +1\}$

Let $f_\ell(x)$ be the score for x given by the model h_ℓ

$h(x) = \arg\max_\ell f_\ell(x)$
Output: Multiclass hypothesis $h : X \to \mathcal{Y}$

10.2 One-versus-One (OVO) (a.k.a all-pairs) Classification

Pros and cons:

- \- 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

+ The OVO training sets are less likely to be imbalanced than in OVA

- Increased chance of overfitting (due to smaller training set)

+ Better theoretical justification (compared to OVA) through the **majority voting ensemble** approach

How:

- 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 $(\ell, \ell'), 1 \leq \ell < \ell' \leq k$
- FThis entails generating a new training set consisting of examples of the pair of classes (ℓ, ℓ') and generating a **surrogate labels**

$$\tilde{y}^{\ell,\ell'} = \begin{cases} +1, & \text{if } y = \ell \\ -1, & \text{if } y = \ell' \end{cases}$$

- For each class pair, a binary hypothesis $h_{\ell,\ell'}(x) : X \to \{-1, +1\}$ is trained using the generated training set

OVO prediction:

- In predicting, for each class ℓ we have $k-1$ pairwise hypotheses, one for each class containing ℓ ($h_{\ell,\ell'}$ and $h_{\ell',\ell}$, for all $\ell' \neq \ell$)
- In the **ideal case**, all of the $k-1$ hypotheses involving class ℓ would predict class
- **In practice** this may not happen, we might have for some classes ℓ', ℓ''
  - $h_{\ell,\ell'} = +1$, predicting class ℓ for x
  - $h_{\ell,\ell''} = -1$, predicting class ℓ'' for x
- We need to resolve these discrepancies. A **voting approach** can be used:

– We count for each input x, how many pairwise hypotheses predict class ℓ (the votes)

$$h(x) = \arg\max_{\ell} \sum_{\ell<\ell'} 1_{\{h_{\ell\ell'}(x)=+1\}} + \sum_{\ell>\ell'} 1_{\{h_{\ell'\ell}(x)=-1\}}$$

– Ties can occur with several classes receiving the same number of votes, we can break them arbitrarily (e.g. predicting the smallest index ℓ)

10.3 Error-correcting codes (ECOC) Classification

What: a general methods for reducing *multi-class problems* **to** binary classification

How:

- Each class ℓ is allocated a codeword m_ℓ 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

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

Figure 24: Illustration of matrix M in the ECOC approach.

- Given the codeword matrix, a **binary classifier** $f_j : X \to \{-1, +1\}$ is learned for each column $j = 1, ..., c$ of the codeword matrix

- The training data for the classifier of column j is relabeled with **surrogate labels**

$$\tilde{y}_i^{(j)} = \begin{cases} m_{\ell,j}, & \text{if } y_i = \ell \\ -m_{\ell,j}, & \text{if } y_i \neq \ell \end{cases}$$

- The **prediction** of the ECOC model is taken as the class ℓ with the fewest wrongly predicted columns of the keyword:

$$h(x) = \arg\min_{\ell=1}^{k} \sum_{j=1}^{c} 1_{f_j(x) \neq m_{\ell j}}$$

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

Why does ECOC work?

- The **prediction of the ECOC model** can be seen as correcting incorrectly predicted bits of the codeword
  - corrected codeword = 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 $\frac{d-1}{2}$ 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

10.4 Standalone multi-class classifiers

What: Models that directly aim to minimize a multi-class loss function.

Why: May give better predictive performance than the approaches based on aggregating binary classifiers. Also defining a combined model may be more efficient to train.

How: Using models like Multiclass SVM, Multiclass boosting, Multiclass neural networks, Multiclass Decision trees and so on.

10.4.1 Multi-class SVM

How:

- Multi-class SVM learns k hyperplanes $f_\ell(x) = w_\ell^T x = 0$ simultaneously
- The **predicted class** is the class with the highest score

$$h(x) = \arg\max_\ell f_\ell(x)$$

- The **objective** is to have the score of the correct class to be higher than all the other classes by a margin (of 1)

$$w_{y_i}^T x_i - w_\ell^T x_i \geq 1 - \xi_i, \text{ for all } \ell \neq y_i$$

  - Above, slack $\xi_i \geq 0$ in used in the analogous way to binary SVMs to allow some examples to not to have the required margin

Multi-class SVM has k weight vectors $w_1, ..., w_k$ to control

- This is achieved by a regularizer that computes the sum of norms:

$$\sum_{\ell=1}^{k} ||w||_2^2$$

- The regularizer is motivated by controlling the empirical Rademacher complexity $\hat{R}(H)$ of the hypothesis class H of multi-class SVMs:

$$\hat{R}(H) \leq \sqrt{\frac{r^2 \wedge^2}{m}}$$

  - Where $\sum_{\ell=1}^{k} ||w_\ell||_2^2 \leq \wedge^2$ and $||x_i||_2^2 \leq r^2$ for all $i = 1, ..., m$

- Thus, minimizing the sum of norms aids achieving good generalization

We can write the Multi-class SVM as regularized loss minimization problem. The Multi-class SVM optimization problem can be written as follows:

$$\min_{w,\xi} \quad \underbrace{\frac{\delta}{2} \sum_{\ell=1}^{k} ||w_\ell||_2^2}_{\text{Regularization}} + \underbrace{\sum_{i=1}^{m} \max\{0, 1 - [w_{y_i}^T x_i - \max_{\ell \neq y_i} w_\ell^T x_i]\}}_{\text{multi-class Hinge loss}}$$

- This problem corresponds to the QP formulation by setting $\lambda = 1/C$
- The problem is convex but but the loss is piecewise linear, thus not differentiable everywhere $\rightarrow$ A **stochastic gradient descent** algorithm can be defined through computing the subgradients of the objective function

Multi-class SVM with kernels:

- We can perform non-linear multi-class classification by using a kernel $\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$ over the data
- The kernelized version of the multi-class SVM optimizes dual variables $\alpha = (\alpha_i, \ell), i = 1, ..., m, \ell = 1, ..., k$ (one dual variable for each training example i and possible class ℓ)
- The optimization problem is given by

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} \alpha_{i,y_i} - \frac{1}{2} \sum_{\ell=1}^{k} \sum_{i,i'=1}^{m} \alpha_{i,\ell}\, \alpha_{i',\ell}\, \kappa(x_i, x_{i'}) \\ \text{s.t.} \quad & \sum_{\ell} \alpha_{i,\ell} = 0 \\ & \alpha_{i,\ell} \leq 0, \text{ for } \ell \neq y_i, 0 \leq \alpha_{i,y_i} \leq C \end{aligned}$$

- Model's prediction in dual form:

$$\hat{y}(x) = \arg\max_{\ell=1,\ldots,k} \sum_{i=1}^{m} \alpha_{i,\ell}\, \kappa(x_i, x)$$