CS-E4710 Machine Learning: Supervised Methods

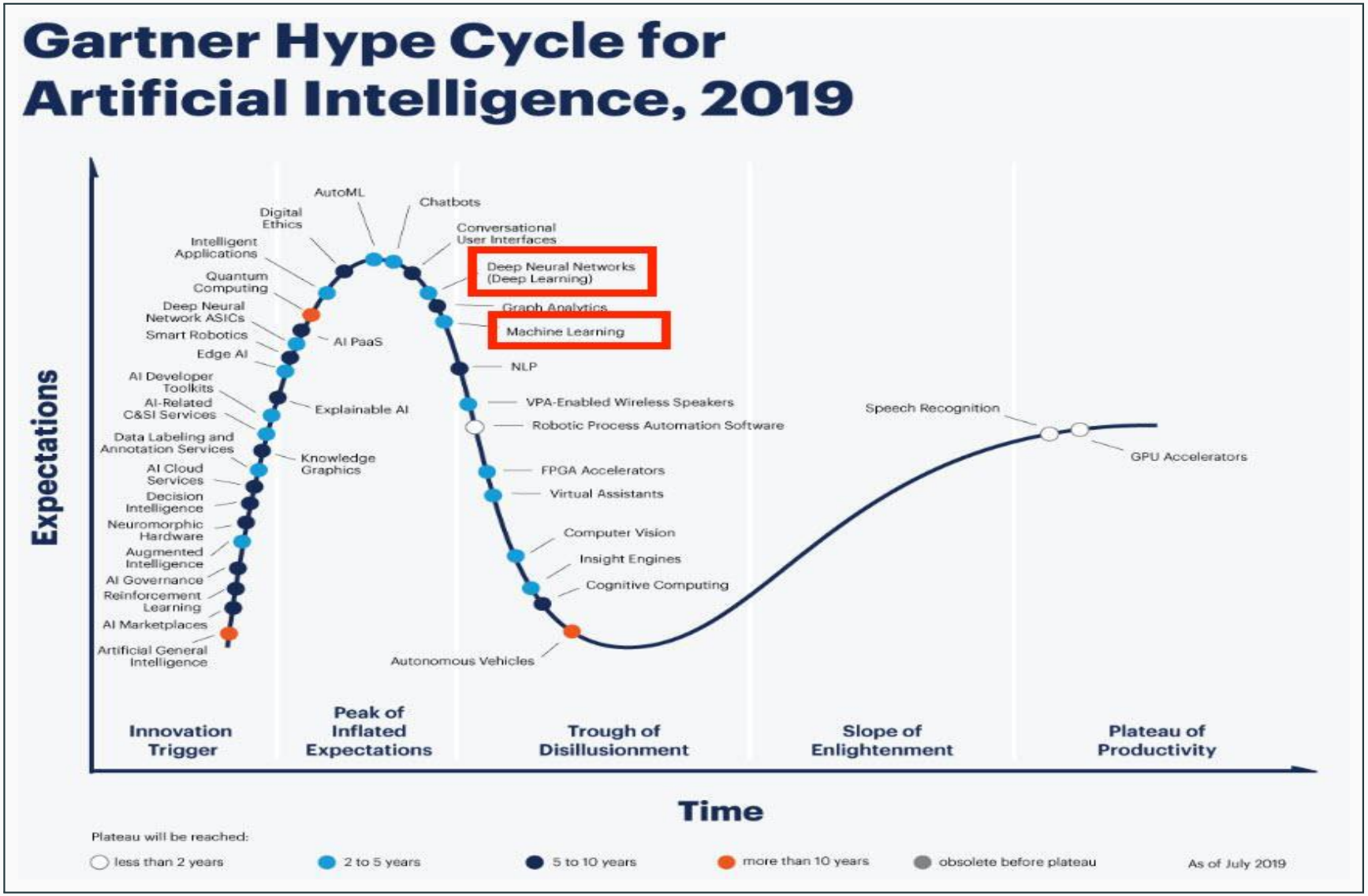
Lecture 1: Introduction

Juho Rousu

September 8, 2020

Department of Computer Science
Aalto University

Mission: going beyond the hype in machine learning

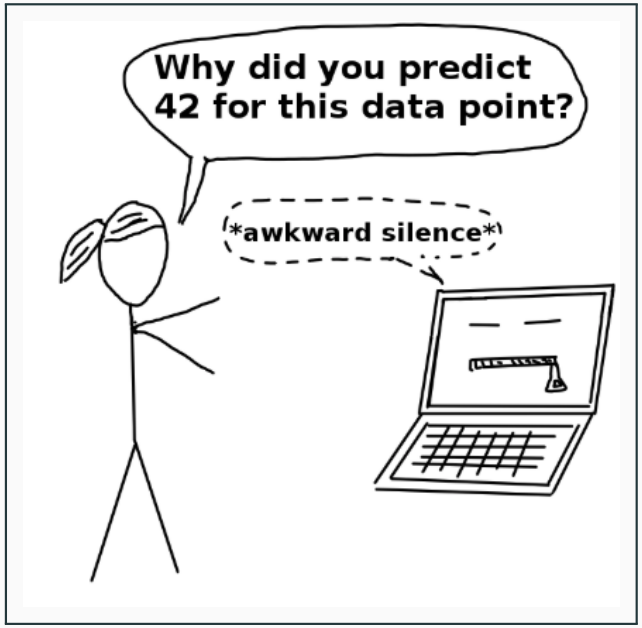


https://www.gartner.com/smarterwithgartner/

Understanding machine learning

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

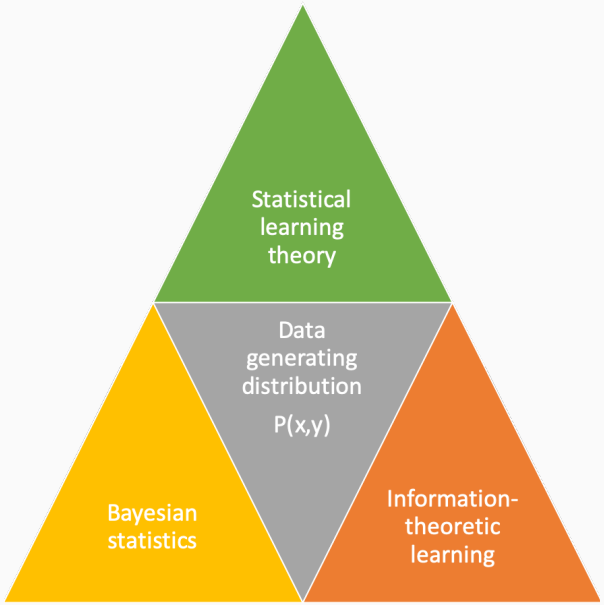
- Machine learning does not always succeed, one needs to be able to understand why and find remedies
- Not possible to follow the scientific advances in the field without understanding the underlying principles
- Competitive advantage: more jobs and better pay for people that have understanding of the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

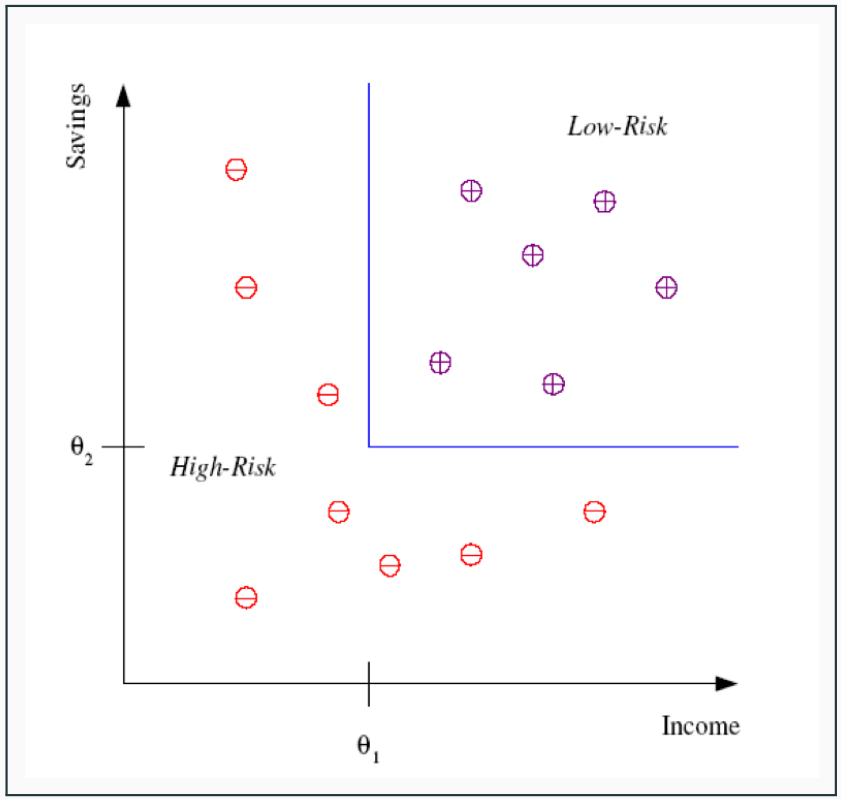
- Part I: Theory
  - Introduction
  - Generalization error analysis & PAC learning
  - Rademacher Complexity & VC dimension
- Part II: Algorithms and models
  - Linear models: perceptron, logistic regession
  - Support vector machines
  - Kernel methods
  - Boosting
  - Neural networks (MLPs)
- Part III: Additional learning models
  - 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)

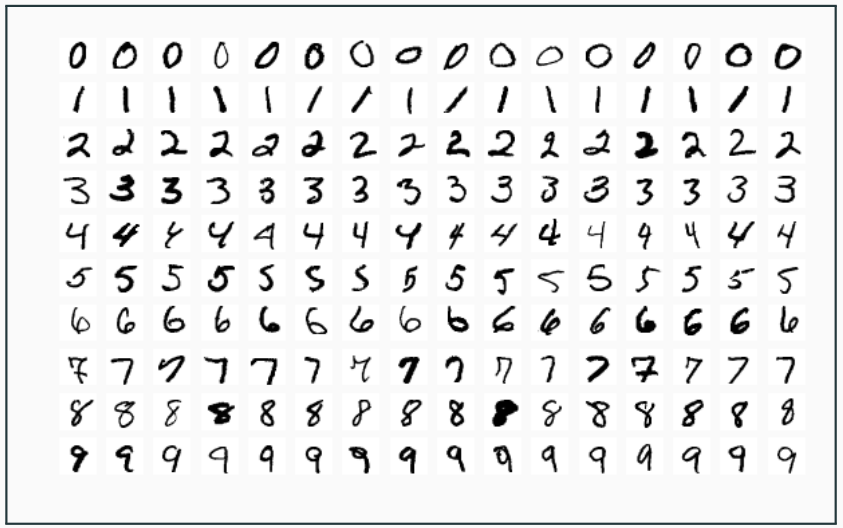
- Example: In credit scoring task, a bank would like to predict is the customer should be given credit or not
- Decision can be based on available input variables: Income, Savings, Employment, Past financial history, etc.
- Output variable is a class label: low-risk (0) or high-risk (1)
- This is called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



Multi-class classification

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

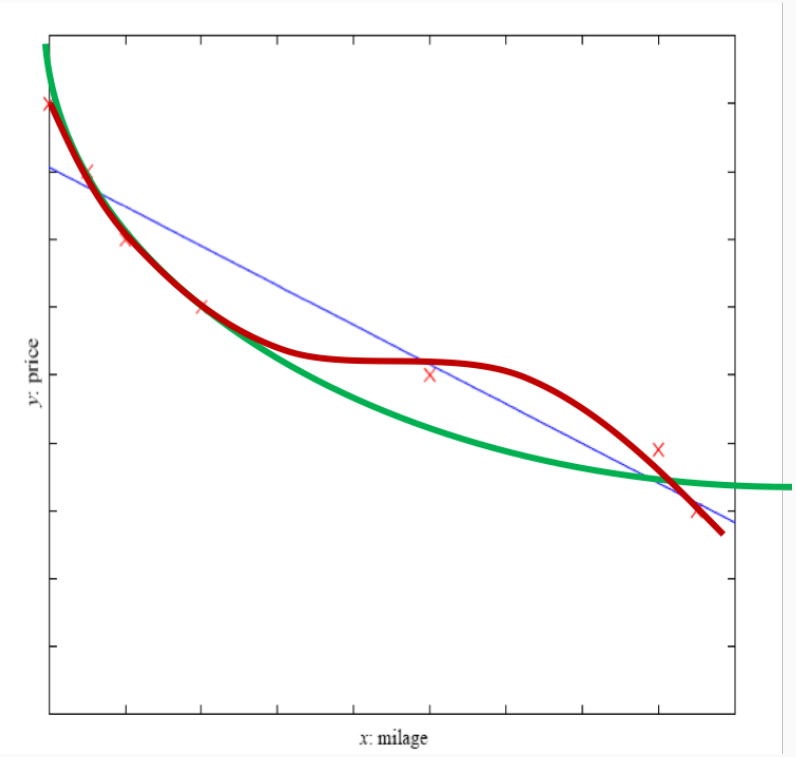
- Example: hand-written character recognition
- Input: images of hand-written characters
- Output: the identity of the character (e.g. Unicode ID)



- **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

Data spaces

- 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: $\mathcal{Y} = \{1, \ldots, K\}$
  - 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: $L_{sq}(y, y') = (y' - y)^2$, $y, y' \in \mathbb{R}$
  - $0/1$ 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**

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

- 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)

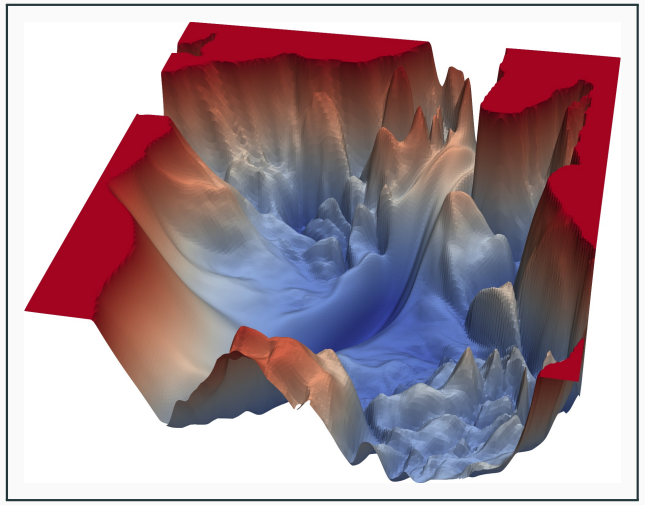
Each have their different pros and cons in different dimensions (accuracy, efficiency, interpretability); No single best hypothesis class exists that would be superior to all others in all circumstances.

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) $\mapsto$ hard to find the global optimum
- NP-hard optimization problems (e.g finding a linear classifier that has the smallest empirical risk) $\mapsto$ 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

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 $L_{sq}(y, y') = (y - y')^2$
- Model family: hyperplanes in $h(\mathbf{x}) = \sum_{j=1}^d \beta_j x_j + \beta_0$
- Model: $y = h(\mathbf{x}) + \epsilon$, where ϵ is random noise corrupting the output. We assume zero-mean normal distributed noise: $\epsilon \sim \mathcal{N}(0, \sigma^2)$, with unknown σ
- Optimization: essentially, inverting a matrix (low polynomial time complexity)

Example: linear regression

Optimization problem

$$minimize \sum_{i=1}^{m}(y - \sum_{j=1}^{d} \beta_j x_{ij} + \beta_0)^2$$

$$w.r.t. \beta_j, j = 0, \ldots, d$$

Write this in matrix form:

$$minimize \ (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

$$w.r.t. \ \boldsymbol{\beta} \in \mathbb{R}^{d+1}$$

where $\mathbf{X} = \begin{bmatrix} 1 & \mathbf{x}_1 \\ \vdots & \vdots \\ 1 & \mathbf{x}_i \\ \vdots & \vdots \\ 1 & \mathbf{x}_m \end{bmatrix}, \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_d \end{bmatrix} \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots y_i \\ \vdots \\ y_m \end{bmatrix}$

Example: linear regression

Minimum of

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

is attained when the partial derivatives are zero

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

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

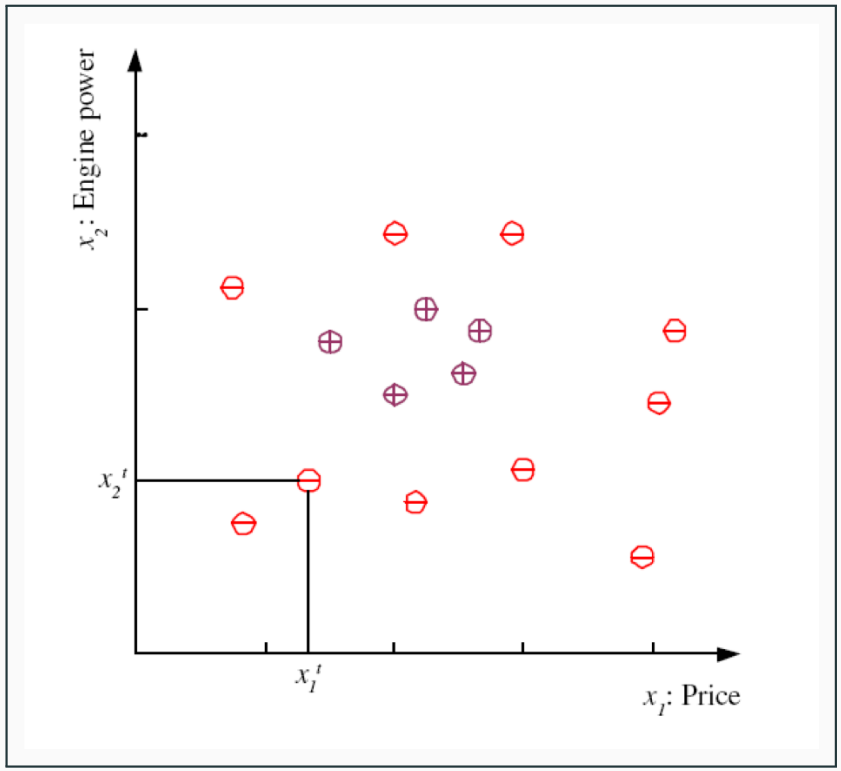
$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

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

Binary classification

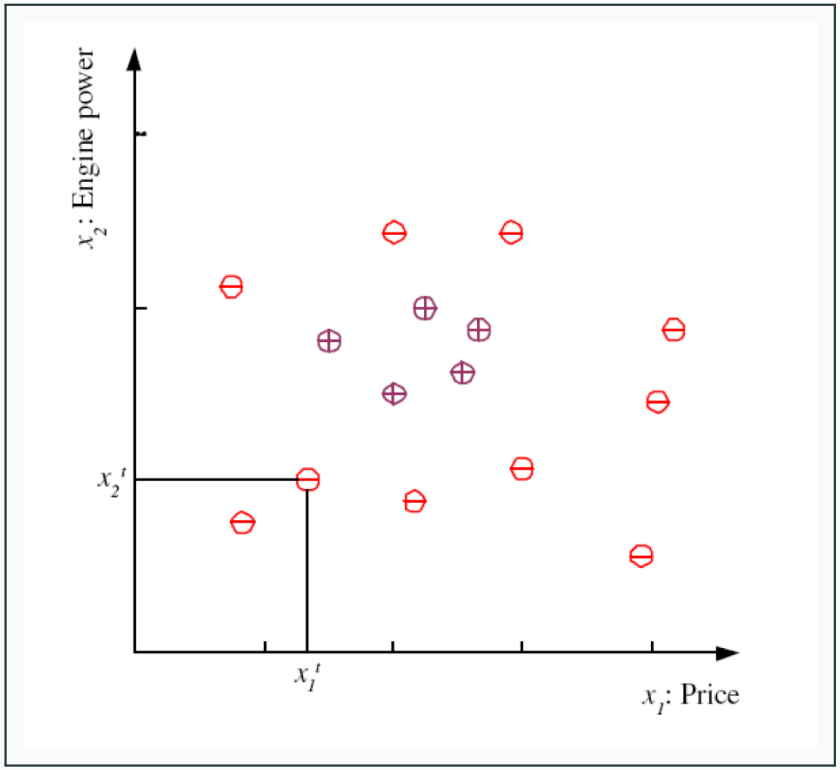
Binary classification

- 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 $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$, where x_1 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

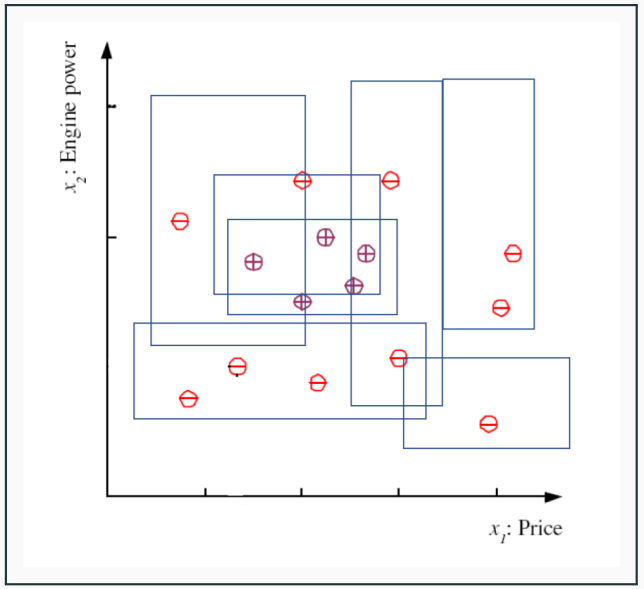


Hypothesis class

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

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

- The classifier will predict a "family car" if both price and engine power are within their respecive 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

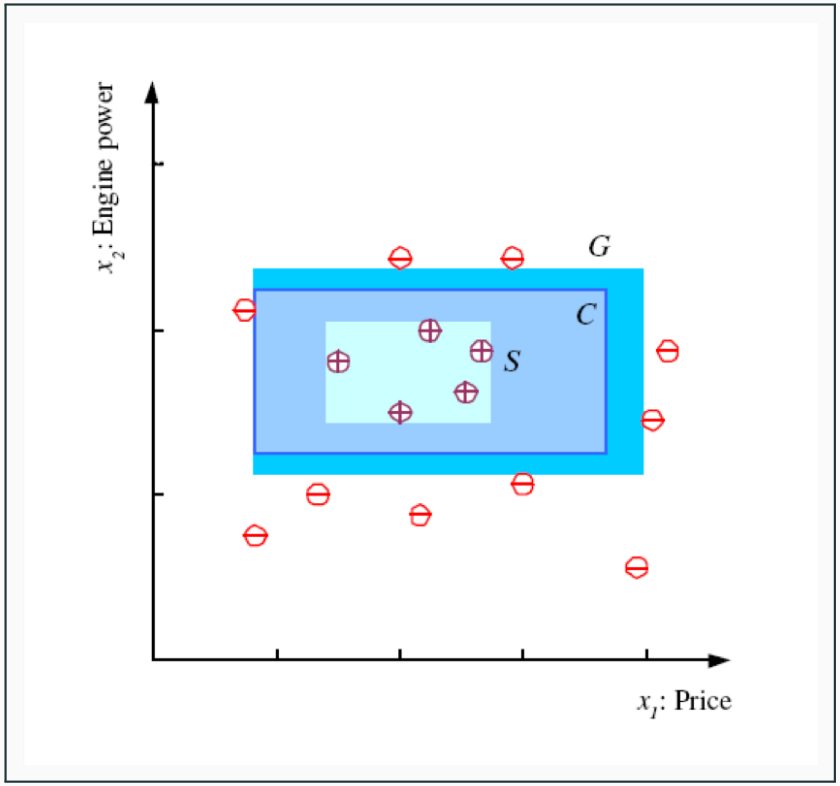


However, we do not know C, so cannot measure exactly how close h is to C!

Version space

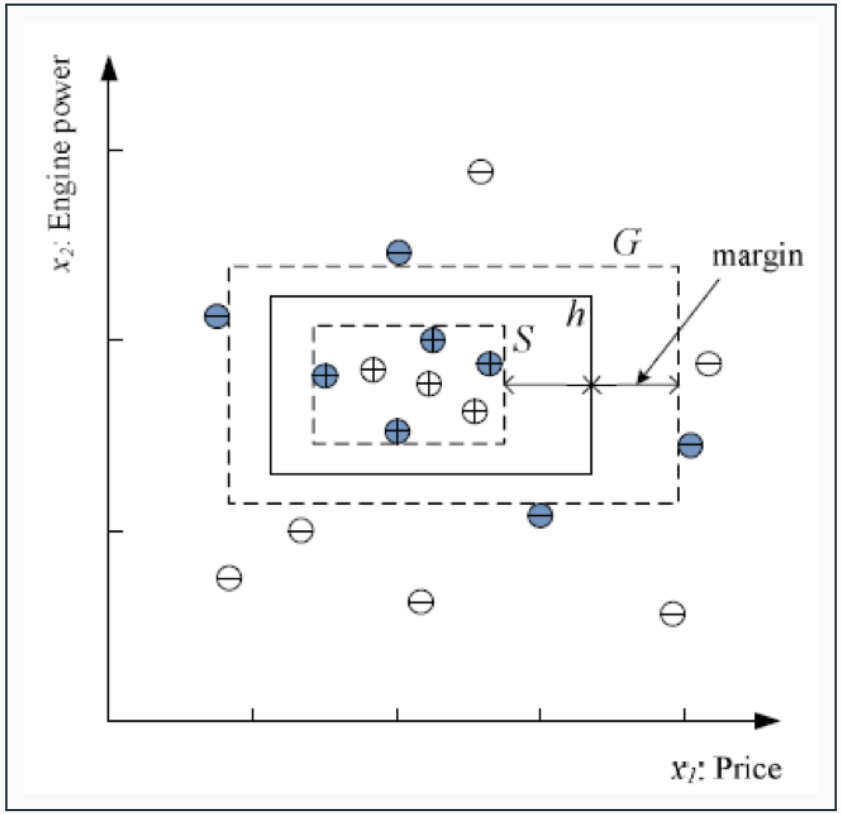
If a hypothesis correctly classifies all training examples we call it a **consistent hypothesis**

- 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

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:

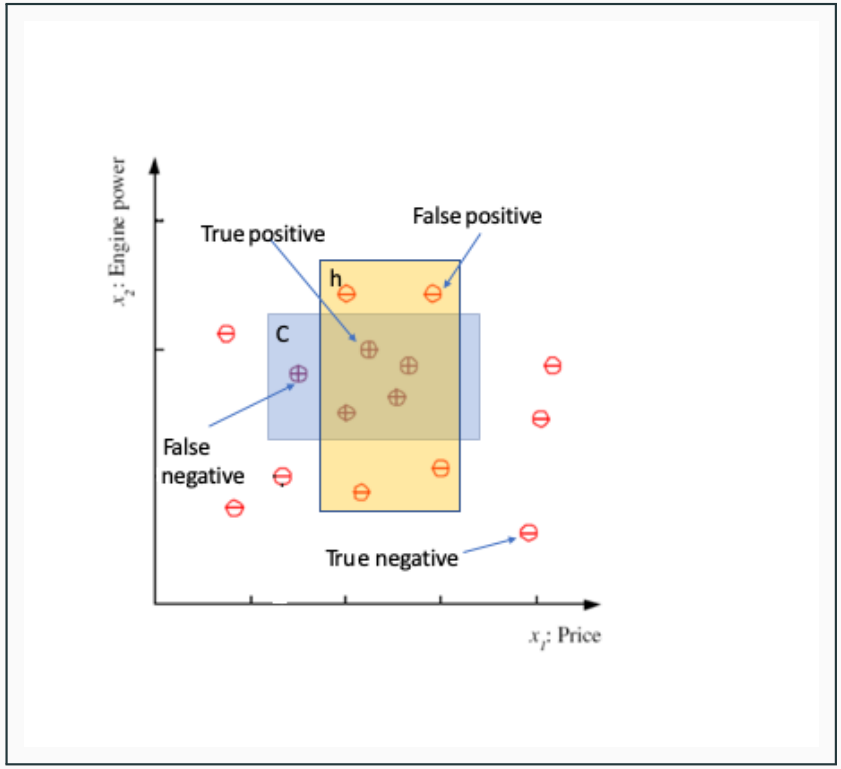
$$\mathbf{1}_A = \begin{cases} 1 & \text{if } A \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

- 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**

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

		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

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:
  $Rec(h) = \frac{m_T P}{m_T P+m_{FN}}$
- 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`

Receiver Operating Characteristics(ROC)

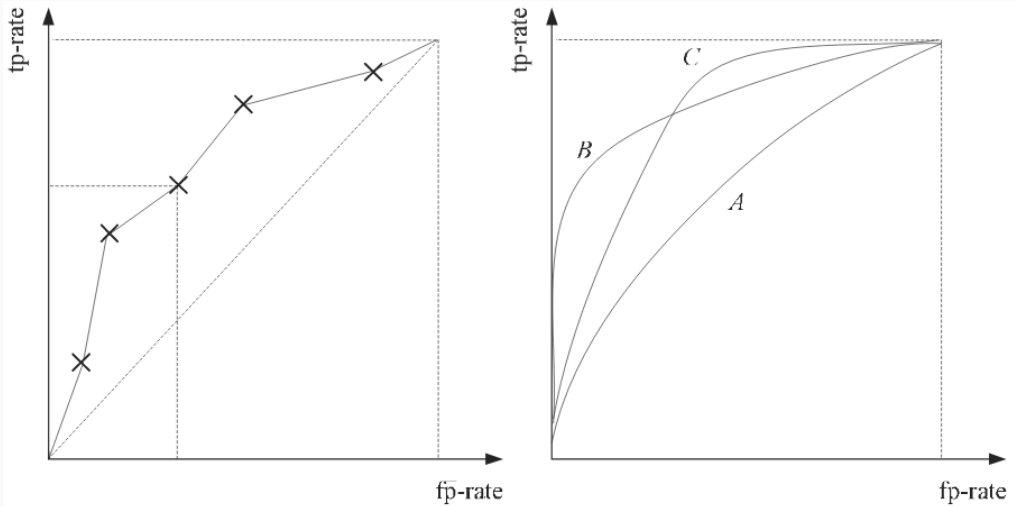
- ROC curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds.
- Consider a system which returns an estimate of the class probability $\hat{P}(y|x)$ or a any score that correlates with it.
- We may choose a threshold θ 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 θ prediction will be 0 for large fraction of the data (and there are likely more false negatives), for low values of θ 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 θ 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/algorithm is globally better
- The curve is sometimes summarized into a single number, the area under the curve (AUC or AUROC)



(a) Example ROC curve

(b) Different ROC curves for different classifiers

Empirical evaluation of supervised learning models

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 a independent test sample:

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

- The expectation of $R_{test}(h)$ 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

Parameter tuning: validation set

- Machine learning models often have parameters that need to be fixed before training
- To tune the parameters we need to set aside part of the training data as a separate validation set
- Train the model using the training data with different parameter values and fix the parameters values that give the best performance on the validation set
- An estimate of the models accuracy (with the chosen parameter value) is obtained by computing the evaluation metric on the test set, not on the validation set

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, test) splits

Dataset splitting

- For replication purposes, our first need is to get a number of training and validation set pairs.
- Given a dataset S, we would like to generate K random splits of $S =$ into training $(\mathcal{T}_k)$ and validation set $(\mathcal{V}_k)$,$\{\mathcal{T}_k, \mathcal{V}_k\}_{k=1}^{K}$
- Generally the training set is kept as large as possible
- Stratification: class distributions of the training and validation sets should be as similar to each another as possible. One may need to split the positive and negative classes separately to ensure the correct class proportions after splitting

K-Fold Cross-Validation

- The dataset X is divided randomly into K equal-sized parts, $X_i, i = 1, \cdots, K$.
- To generate each pair, we keep one of the K parts out as the validation set and combine the remaining $K - 1$ parts to form the training set.

$$\begin{array}{ll} \mathcal{V}_1 = \mathcal{X}_1 & \mathcal{T}_1 = \mathcal{X}_2 \cup \mathcal{X}_3 \cup \cdots \cup \mathcal{X}_K \\ \mathcal{V}_2 = \mathcal{X}_2 & \mathcal{T}_2 = \mathcal{X}_1 \cup \mathcal{X}_3 \cup \cdots \cup \mathcal{X}_K \\ \vdots & \\ \mathcal{V}_K = \mathcal{X}_K & \mathcal{T}_K = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \cdots \cup \mathcal{X}_{K-1} \end{array}$$

- if m (number of the data examples) is small, K should be large to allow large enough training sets
- Extreme case of m-fold 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.