Machine Learning

Matteo Alberici

February 2022

How do we construct computer programs that automatically improve with experience?

Contents

1 Introduction **5**
- 1.1 Learning Procedure . . . 5
- 1.2 Models . . . 5
- 1.3 Measure and Measurements . . . 5
  - 1.3.1 Additive Models . . . 6
  - 1.3.2 Multiplicative Models . . . 6
- 1.4 Learning Frameworks . . . 6
  - 1.4.1 Supervised Learning . . . 6
  - 1.4.2 Unsupervised Learning . . . 6
- 1.5 Features . . . 6

2 Regression Problem **7**
- 2.1 Multiple Linear Regression . . . 7
  - 2.1.1 Least Mean Square . . . 7
  - 2.1.2 Estimating Parameters . . . 7
- 2.2 Performance at Task in Regression . . . 8
- 2.3 Properties of Learning Frameworks . . . 8
- 2.4 Ridge Regression . . . 8
- 2.5 Lasso Regression . . . 9
- 2.6 Final Prediction Error . . . 9
- 2.7 Non-linear Regression . . . 9
- 2.8 Structural Risk . . . 9
- 2.9 Feedforward Neural Networks . . . 10
  - 2.9.1 Backpropagation Algorithm . . . 11
- 2.10 Model Complexity . . . 12
  - 2.10.1 Early Stopping . . . 12
  - 2.10.2 Splitting the Data . . . 12

3 Classification Problem **13**
- 3.1 Standard Classifiers . . . 13
  - 3.1.1 Bayes Classifier . . . 13
  - 3.1.2 Binary Classifier . . . 13
  - 3.1.3 K-nearest Neighbors Classifier . . . 14
- 3.2 Linear Discriminant Analysis . . . 14
  - 3.2.1 One-dimensional Setting . . . 15
  - 3.2.2 Vector Space Setting . . . 15
- 3.3 Perceptron Algorithm . . . 15
- 3.4 Feedforward Neural Networks Classification . . . 16
- 3.5 Logistic Regression . . . 16
- 3.6 Loss Functions for Classifiers . . . 16

4 Model Performance **17**
4.1 Quality Assessment . . . 17
4.1.1 Apparent Error Rate . . . 17
4.1.2 Sample Partitioning . . . 17
4.1.3 K-fold Crossvalidation . . . 17
4.1.4 Leave-one-out . . . 17
4.1.5 The Bootstrap Method . . . 17
4.2 Model Validity . . . 17
4.2.1 Regression Problem . . . 17
4.2.2 Is model A better than model B? . . . 18
4.3 Anomaly Detection . . . 18
4.3.1 Confusion Matrices . . . 19
4.3.2 Probabilities . . . 19

5 Deep Learning **21**
5.1 Convolutional Neural Networks . . . 21
5.1.1 Convolutional Layers . . . 21
5.1.2 Pooling Layers . . . 21
5.2 Special Activation Functions . . . 22
5.2.1 Rectified Linear Unit Activation Function . . . 22
5.2.2 Softmax Activation Function . . . 22
5.3 Autoencoders . . . 22

6 Results From Theory **23**
6.1 Vapnik Chervonenkis Dimension . . . 23
6.2 Uniform Convergence of Empirical Mean . . . 23
6.3 Traditional Statistical Approach . . . 23
6.4 Bias-Variance Tradeoff . . . 23

7 Other Classification Families **24**
7.1 Classification and Regression Trees . . . 24
7.2 Bagging . . . 24
7.3 Maximal Margin Classifier . . . 24
7.4 Support Vector Classifier . . . 25
7.5 Kernel and Support Vector Machines . . . 26

8 Unsupervised Learning **27**
8.1 Isolation Forest . . . 27
8.2 Principal Component Analysis . . . 28
8.3 K-means Clustering . . . 29

9 Forecasting **30**
9.1 Time Series . . . 30
9.1.1 Stationarity . . . 30
9.1.2 Time Invariance . . . 30
9.1.3 White Noise Processes . . . 30

9.2 Linear Time Invariance Models 30
9.2.1 AR - Autoregressive Predictive Model 30
9.2.2 MA - Moving Average Predictive Model 31
9.2.3 ARMA - Autoregressive Moving Average Predictive Model 31
9.2.4 ARIMA - Autoregressive Integrated Moving Average Predictive Models 31
9.3 Sliding Window Approach 32
9.4 Exogenous LTI Models 32
9.5 Recurrent Neural Networks 32
9.5.1 RNNs with Long Short-Term Memory 32
9.5.2 Gated Recurrent Units 32
9.5.3 Gated Recurrent Units 32
9.6 Multi-step Prediction Strategies 33
9.6.1 Recursive Strategy 33
9.6.2 Direct Strategy 33
9.6.3 Multiple Input - Multiple Output Strategy 33

10 Appendix A - Mathematical Concepts **34**

1 Introduction

1.1 Learning Procedure

A computer program **learns** from an experience E with respect to some class of tasks T and a performance measure P if its **performance at task** in T improves with E. There exist different types of tasks:

- **Classification**

  Determine the model that partitions measurements in classes

- **Regression**

  Determine the model that explains measurements

- **Prediction**

  Tell the next measurements along with a confidence level

- **Control**

  Generate optimal actions over time to solve a problem

- **Clustering**

  Gather similar instances according to some functions

We must determine the best model from a **family of models** solving a task:

$$f(\theta, x) \implies f(\hat{\theta}, x)$$

1.2 Models

Models are used to satisfy hypotheses. There exist three types of models:

- **White box models**

  Physical laws and structural parameters are known; thus, modeling equations can be derived.

- **Grey box models**

  Physical laws and at least one structural parameter are known; thus, modeling equations can be derived, but parameters must be identified

- **Black box models**

  Physical laws are unknown; thus, modeling equations cannot be derived

1.3 Measure and Measurements

Measuring a quantity x_0 results in a **measurement** x_t at time t with a sensor which could introduce sources of uncertainty; x_t is an approximation:

$$\varepsilon_i = |x_0 - x_t|$$

1.3.1 Additive Models

In **additive models**, measurements depend on an i.i.d. random variable η called the **noise**, depending on the sensors:

$$x = x_0 + \eta \qquad \eta = f_\eta(0, \sigma_\eta^2)$$

1.3.2 Multiplicative Models

In **multiplicative models**, the impact of the noise is $x_0\eta$, but the relative contribution η does not depend on x_0:

$$x = x_0(1 + \eta)$$

1.4 Learning Frameworks

1.4.1 Supervised Learning

In a **supervised learning** framework, there is a concept to learn, a supervisor teaching it, and a program that learns and formulates an educated guess.

1.4.2 Unsupervised Learning

In an **unsupervised learning** framework, we build a representation of data through a machine providing information for decision making given an input.

1.5 Features

We extract **features** from measurements to ease learning tasks. By reducing them to a minimal set called the **feature selection**, they provide a compact representation of the inputs, being advantageous with prior information. Features are usually derived from images or models and each of them is represented by a **parameter vector**:

$$\hat{\theta}_{t1}, \ldots, \hat{\theta}_{tn} \qquad y_1(t) \to f_\theta^{1,2} \to y_2(t)$$

2 Regression Problem

Regression aims at obtaining the optimal hyperplane explaining a dataset. **Linear models** are good with high uncertainty and a linearly-generated small and sparse dataset.

2.1 Multiple Linear Regression

Suppose we have a **training set** of n data couples:

$$\{(x_1, y_1), \ldots, (x_n, y_n)\} \qquad x \in \mathbb{R}^d, \;\; y \in \mathbb{R}$$

Let us assume that the unknown function generating the data is linear and that there is a Gaussian uncertainty affecting measurements in an additive way:

$$y(x) = x^T \theta^\circ + \eta \qquad \theta^{\circ T} = [\theta_1^\circ, \ldots, \theta_d^\circ] \qquad \theta^\circ \in \mathbb{R}^d$$

Optimal parameters and variance of noise are unknown. The goal is to determine the **best estimated parameter vector** $\hat{\theta}$ to generate the best model:

$$f(\hat{\theta}, x) = x^T \hat{\theta}$$

2.1.1 Least Mean Square

The **Least Mean Square** (**LMS**) procedure finds the linear function minimizing the average distance between the dataset and the function itself:

$$V_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y(x_i) - f(\theta, x_i))^2$$

The parameter vector $\hat{\theta}$ minimizing the performance function is:

$$\hat{\theta} = \underset{\theta \in \Theta}{argmin} \, V_n(\theta)$$

2.1.2 Estimating Parameters

By grouping x_i in X and y_i in Y, we obtain:

$$V_n^*(\theta) = (Y - X\theta)^T (Y - X\theta)$$

The function is **convex**, meaning there is one estimate unless the problem degenerates. The minimum is found by computing the **stationary points**:

$$\frac{\partial V_n^*(\theta)}{\partial \theta} = -2X^T Y + 2X^T X\theta = 0$$

The parameter vector $\hat{\theta}$ minimizing the performance function is:

$$\hat{\theta} = (X^T X)^{-1} X^T Y$$

2.2 Performance at Task in Regression

Since the performance at task $V_n(\hat{\theta})$ is biased to the training set, we consider the unseen **test set** to measure it:

$$\{(\bar{x}_1, \bar{y}_1), \ldots, (\bar{x}_l, \bar{y}_l)\}$$

We asses the performance providing a mean through **cross-validation**:

$$V_l(\hat{\theta}) = \frac{1}{l} \sum_{i=1}^{l} (\bar{y}_i - \bar{x}^T \hat{\theta})^2$$

It is fundamental not to bias neither the learning procedure nor the test set.

2.3 Properties of Learning Frameworks

Under a linear framework, it is proved that:

$$\lim_{n \to \infty} \hat{\theta} = \theta^{\circ} \qquad \lim_{l \to \infty} V_l(\hat{\theta}) = \sigma_\eta^2$$

Given n training couples, it can be proved that:

$$Var(\hat{\theta}) = (X^T X)^{-1} \sigma_\eta^2 \qquad \hat{\sigma}_\eta^2 = \frac{1}{n-d} \sum_{i=1}^{n} (y(x_i) - f(\hat{\theta}, x_i))^2$$

The **Occam's razor strategy** states that if a parameter is smaller than twice its standard deviation, then set it to 0, re-evaluate the performance, and decide whether to keep it or not.

2.4 Ridge Regression

Ridge regression aims at pushing as many parameters as possible towards 0 by adding a shrinking penalty to the loss function. Inputs must be centered, meaning that their expected value must be 0. The **Mean Squared Error** (**MSE**) training the performance measure is:

$$V_{Ridge}(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \theta)^2 + \lambda ||\theta||^2$$

The **hyperparameter** λ weighs both accuracy and shrinking: a small λ gives more value to accuracy, while a large one privileges a small number of parameters. We obtain a trade-off by estimating an appropriate λ with the **validation set**. The parameter vector $\hat{\theta}$ minimizing the performance function is:

$$\hat{\theta} = (X^T X + \lambda I)^{-1} X^T Y$$

2.5 Lasso Regression

Lasso regression penalizes the parameter itself, but not in a quadratic way:

$$V_{Lasso}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i^T\theta)^2 + \lambda\sum_{i=2}^{d}|\theta_i|$$

The optimization problem is no more convex; thus, the minimization problem is solved through quadratic programming. Some coefficients are now set exactly to 0 thanks to the dual formulation of the problem.

2.6 Final Prediction Error

We could generate infinite models with infinite n-sized datasets:

$$E_n[E_\eta[V_v]] = \sigma_\eta^2(n+d)$$

The **Final Prediction Error** (**FPE**) assesses performance on unseen data and can be used for model selection on hierarchical families:

$$FPE = \frac{n+d}{n-d}\sum_{i=1}^{n}(y(x_i) - f(\hat{\theta}, x_i))^2$$

2.7 Non-linear Regression

The minimization of a non-linear function is known as **learning procedure**. The **gradient-based optimization** determines the stationary points of a differentiable function through minimization. Given a generic convex and differentiable scalar function, we start from an initial point and move along the gradient in the direction minimizing the loss function:

$$\theta_{i+1} = \theta_i - \varepsilon_L \frac{\partial V_n(\theta)}{\partial\theta}|_{\theta_i}$$

The **identifiability problem** occurs if the function is not convex.

2.8 Structural Risk

A **loss function** measures the distance between the training set and the family of models:

$$L(y(x), f(\theta, x))$$

The **structural risk** of the estimated model is its generalization ability:

$$\bar{V}(\theta) = \int L(y, f(\theta, x))p_x\ dxy$$

The risk can be decomposed as follows:

$$\bar{V}(\hat{\theta}) = (\bar{V}(\hat{\theta}) - \bar{V}(\theta^0)) + (\bar{V}(\theta^0) - V_I) + V_I$$

- **Inherent Risk:** V_I

  Depends on the problem structure and is improved by improving the problem itself, such as by reducing the noise caused by the sensors

- **Approximation Risk:** $\bar{V}(\theta^0) - V_I$

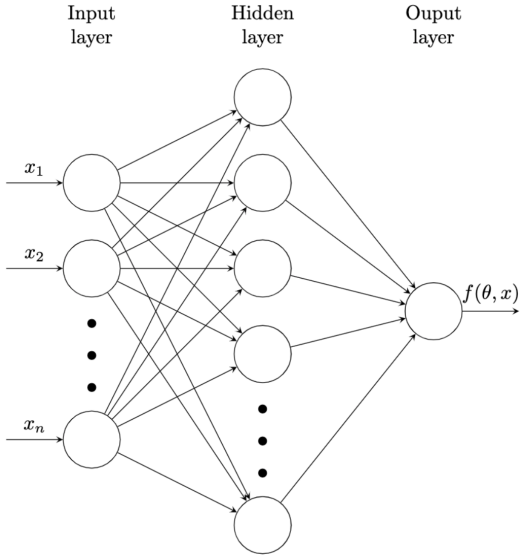
  Depends on how close the approximating family of models is to the function generating the data and is improved with a better family of models

- **Estimation Risk:** $\bar{V}(\hat{\theta}) - \bar{V}(\theta^0)$

  Depends on the learning procedure's effectiveness and can be improved by using a more effective procedure

2.9 Feedforward Neural Networks

A **Feedforward Neural Network** (**FNN**) has the following structure:



The input vector $[x_1, \ldots, x_n]$ must be normalized:

$$\frac{x_i - \mu_i}{\sigma^i}$$

Each neural connection has a weight w_n. The overall weight of a neuron defines its **activation value**:

$$a_t = \sum_{i=1}^{n} x_i w_i$$

Each neuron of the hidden layer has an **activation function** defining how the weighted sum of the input is transformed into an output going to the next layer. There exist different activation functions:

- **Sigmoidal**

  Mainly used by the neurons of the hidden layer:

$$Sig(x) = \frac{1}{1 + e(-x)}$$

- **Hyperbolic Tangent**

  Used along with back-propagation:

$$HT(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- **Linear**

  Mainly used in the output layer

The **universal approximation theorem** states that a feedforward neural network with a single hidden layer containing a finite number of neurons and a linear output neuron approximates any continuous function defined on compact subsets.

2.9.1 Backpropagation Algorithm

The **back-propagation algorithm** computes the gradient to update the weights through the **chain-rule** procedure. Let us consider a quadratic loss function:

$$V_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\theta, x_i))^2$$

By looking at the input space j and the hidden space k, we consider the neuron v_k associated with the weight w_{jk} and obtain the output function:

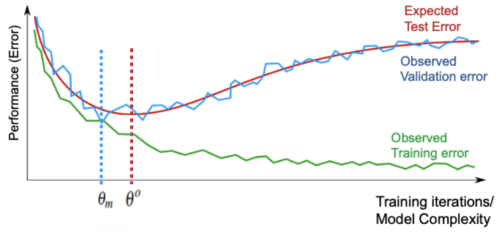
$$f(x_i, \theta) = \sum_k v_k h(\sum_j w_{jk} x_{ij})$$

The **back-propagation algorithm** equation is:

$$\frac{\partial V_n(\theta)}{\partial v_k} = -\frac{2}{n} \sum_i (y_i - f(\theta, x_i)) h(\sum_j w_{jk} x_{ij})$$

2.10 Model Complexity

The goal is to find the optimal **model complexity** of a family of models and a good compromise with performance. **Overfitting** occurs when there are too many parameters in the model, a high classification variance, and a high complexity; in addition, the noise could be learn. In contrast, **underfitting** occurs when there are a few parameters in the model, a high bias, and a low model complexity; thus, training is too simple and the model needs more time and input features.



Too complex models are penalized through the **Tikhonov regularization**, which defines a penalty term controlled by γ:

$$\gamma||\theta||^2$$

2.10.1 Early Stopping

Early stopping is used to avoid overfitting by defining the number of iterations that can be run before the model begins to over-fit: when a minimum is found, the training is stopped after that amount of iteration if a new minimum is not found.

2.10.2 Splitting the Data

In order to split a dataset with length n, we apply a ratio with respect to n and to the model complexity:

$$n = 1000 \rightarrow Tr = 70\%,\ V = 15\%,\ Te = 15\%$$
$$n = 1000000 \rightarrow Tr = 99\%,\ V = 0.5\%,\ Te = 0.5\%$$

3 Classification Problem

Given an input vector x and a set of classes $\{C_1, \ldots, C_K\}$, we compute the probability that x belongs to each class: the one with the maximum probability is chosen. While in regression y has a quantitative value, it assumes a categorical value in classification.

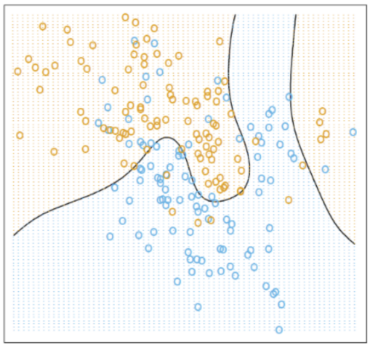
3.1 Standard Classifiers

3.1.1 Bayes Classifier

The **Bayes classifier** assigns a probability to each class, then select the one that maximizes it:

$$P(Y = k \mid X = x) = \frac{P(X = x \mid Y = k) \cdot P(Y = k)}{P(X = x)}$$

- $P(Y = k \mid X = x)$ is the posterior probability
- $P(X = x \mid Y = k)$ is the likelihood
- $P(Y = k)$ is the prior probability
- $P(X = x)$ is the evidence



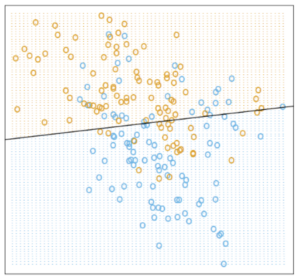
The classifier is "naïve" since it makes strong assumptions on independence among the features: if these assumptions are met and probabilities are known, then the Bayes classifier is the optimal one.

3.1.2 Binary Classifier

Given an input vector of two features, a **binary classifier** splits the input space into two classes through a hyperplane. Given a new x value, we evaluate the **discriminant function**:

$$f(\hat{\theta}, x) = x^T \hat{\theta}$$

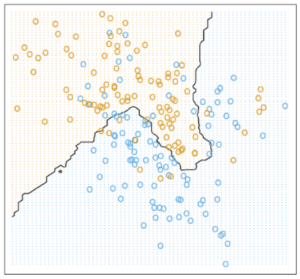
If the value of the discriminant is > 0.5, one class is chosen; otherwise, the output is the other one. This procedure is effective if there is an order making justice at the categorical level, such as "*wavelengths*" for colors.



3.1.3 K-nearest Neighbors Classifier

The **K-nearest neighbors classifier** looks at the k-nearest neighbors of a point x to make decisions:

$$\lim_{n,k\to\infty} \frac{k}{n} = 0$$



3.2 Linear Discriminant Analysis

The **linear discriminant analysis** (**LDA**) finds a separating boundary by reformulating the Bayes theorem in terms of probability density functions:

$$Pr(Y = k \mid X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

- $f_k(x)$ is the likelihood of the k-th class:

$$f_k(x) = Pr(X = x \mid Y = k)$$

- π_k is the prior probability:

$$\pi_k = Pr(Y = k)$$

3.2.1 One-dimensional Setting

Let us consider Gaussian distributed classes with a pdf:

$$f_k(x) = \frac{1}{\sqrt{2\pi} \cdot \sigma_k} e^{-\frac{1}{2}(\frac{x-\mu_k}{\sigma_k})^2}$$

Since we are looking for the maximum of the function, the discriminant function onto which we make the class assignment becomes:

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log(\pi_k)$$

The function is linear in x:

$$\hat{\pi}_k = \frac{n_k}{n}$$

In order to estimate the parameters, we do:

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i \qquad \hat{\sigma}^2 = \sum_{k=1}^{K} \frac{n_k - 1}{n - K} \cdot \hat{\sigma}_k^2$$

The winning class is the k for which $\delta_k(x)$ assumes the maximum value.

3.2.2 Vector Space Setting

If the inputs are vectors instead of scalars, then:

$$f(x) = \frac{1}{(2\pi)^{p/2} |\sum|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T \sum^{-1} (x-\mu)}$$

The linear discriminant function becomes:

$$\delta_k(x) = x^T \sum^{-1} \mu_k - \frac{1}{2} \mu_k^T \sum^{-1} \mu_k + log(\pi_k)$$

With two classes, the LDA coincides with the linear regression classification.

3.3 Perceptron Algorithm

The **perceptron algorithm** solves pattern classification problems. It works well on linearly separable classes and its parameters can be learned. The architecture consists of a single neuron with a Heaviside activation function:

1. Assigns small weights randomly
2. Updates weights by iteratively presenting instances of the training set:

$$w_{t+1}^j = w_t^j - \varepsilon_L (y_i - f(w_t, x_i)) x_i^j$$

If the problem is linearly separable, then the algorithm converges to the optimal parameter configuration.

3.4 Feedforward Neural Networks Classification

The procedure for classifying in feedforward neural networks is the following:

1. Define the performance function:

$$V_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(\theta, x_i))$$

2. Define the learning procedure:

$$\theta_{i+1} = \theta_i - \varepsilon_L \frac{\partial V_n(\theta)}{\partial \theta}|_{\theta=\theta_i}$$

3. Determine the parameter estimate through a minimization problem:

$$\hat{\theta} = \underset{\theta\in\Theta}{argmin}\, V_n(\theta)$$

4. Get the classifier $f(\hat{\theta}, x)$

3.5 Logistic Regression

Since linear classifiers extended from regression methods do not provide a bounded output or a probabilistic interpretation, **logistic regression** aims at training the network's parameters so that a probabilistic framework supports the sigmoidal output. The logistic function is:

$$ln\frac{p}{1-p} = x^T\theta$$

The parameter vector $\hat{\theta}$ is computed as:

$$\hat{\theta} = \underset{\theta\in\Theta}{argmax}\sum_{i=1}^{n}(y_i\theta^T x_i - log(1 + e^{\theta^T x_i}))$$

The estimate can be found with a gradient ascent procedure.

3.6 Loss Functions for Classifiers

The following are some loss functions for classifiers:

- Square error:

$$L(y(x), f(\theta, x)) = (y(x) - f(\theta, x))^2$$

- Binary cross-entropy:

$$L(y(x), f(\theta, x)) = -[y(x)\log f(\theta, x) + (1 - y(x))log(1 - f(\theta, x))]$$

- Multi-classes cross-entropy:

$$L(y(x), f(\theta, x)) = -\sum_{c=1}^{M} y_c(x)\log f_c(\theta, x)$$

4 Model Performance

4.1 Quality Assessment

4.1.1 Apparent Error Rate

The **apparent error rate** (**AER**), or **training error**, estimates the structural risk using the empirical risk: the dataset is used to infer the model and estimate the accuracy performance. AER is optimistically biased unless the dataset is enormous.

4.1.2 Sample Partitioning

The **sample partitioning** (**SP**), or **crossvalidation**, estimates the generalization error $\bar{V}(\hat{\theta})$ on a virgin dataset. Training and test sets are generated by splitting the dataset. SP is an unbiased estimate if it is large.

4.1.3 K-fold Crossvalidation

In **k-fold crossvalidation** (**k-CV**), the dataset is randomly split into k disjoint subsets of equal size: for each subset, the remaining $k-1$ subsets are merged to form the training set, while the reserved one becomes the test set. Finally, the resulting k estimates are averaged.

4.1.4 Leave-one-out

In the **leave-one-out** (**LOO**) method, the test set contains one pattern of the dataset, while the training set contains the remaining $N-1$ patterns. The procedure is iterated N times by holding out each pattern; then, the estimates are averaged. They are highly correlated and there is a large variance.

4.1.5 The Bootstrap Method

The **bootstrap method** performs a the same procedure of the leave-one-out method, but with replacement, meaning that samples can be used more times to estimate. The method underestimates the test error $\bar{V}(\hat{\theta})$.

4.2 Model Validity

4.2.1 Regression Problem

In the regression problems, a quality test requires to inspect the residual errors of the test set S_E:

$$\varepsilon_i = y(x_i) - f(\hat{\theta}, x_i), (x_i, y_i) \in S_E$$

The residual error is characterized by a null expectation and must be unbiased; otherwise, by removing the bias, we improve the performance right away:

$$E[\varepsilon] = 0$$

The verification of the hypothesis requires a test on the sample mean:

$$\textbf{Null hypothesis} \rightarrow H_0 : E[\varepsilon] = 0$$
$$\textbf{Alternate hypothesis} \rightarrow H_1 : E[\varepsilon] \neq 0$$

By designing a statistic, we see which hypothesis holds. Under the null hypothesis, the Central Limit Theorem grants the ***T*-Student** statistic to follow a normal distribution:

$$T = \frac{\bar{\varepsilon}}{\sqrt{s^2/l}} \sim N(0,1) \qquad \bar{\varepsilon} = \frac{1}{l}\sum_{i=1}^{l} \varepsilon_i \qquad s^2 = \frac{1}{l}\sum_{i=1}^{l} \varepsilon_i^2$$

If T is outside the 95% **confidence interval** $[-1.96, 1.96]$, then the null hypothesis is rejected.

4.2.2 Is model A better than model B?

Let us consider two datasets:

$$(x_1^a, y_1^a), \ldots, (x_{la}^a, y_{la}^a) \qquad (x_1^b, y_1^b), \ldots, (x_{lb}^b, y_{lb}^b)$$

Let us assume that both models have no bias:

$$E[\varepsilon_a] = E[\varepsilon_b] = 0$$

Let us design a hypothesis test on the variance. If the models are different, then the model with the smaller variance is preferable:

$$H_0 : Var[\varepsilon_a] = Var[\varepsilon_b]$$
$$H_1 : Var[\varepsilon_a] \neq Var[\varepsilon_b]$$

Let us evaluate the **squared residuals**:

$$e_i^k = (y_i^k - f_k(x_i^k))^2 \qquad i = 1, 2, \ldots, l_k \qquad k = a, b$$

Now we compute the statistic under the Central Limit Theorem:

$$T = \frac{\bar{e}^a - \bar{e}^b}{\sqrt{\frac{s_a^2}{l_a} + \frac{s_b^2}{l_b}}} \sim N(0,1) \qquad \bar{e}^k = \frac{1}{l_k}\sum_{i=1}^{l_k} e_i^k \qquad s_k^2 = \frac{1}{l_k - 1}\sum_{i=1}^{l_k} (e_i^k - \bar{e}^k)^2$$

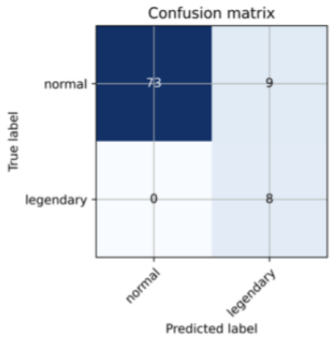
If T is outside the 95% confidence interval, then the hypothesis is rejected, and we select the model with the minor variance.

4.3 Anomaly Detection

Anomaly detectors are classifiers that process data streams looking for anomalies.

4.3.1 Confusion Matrices

The classification performance nature is described in **confusion matrices**.



Each sector of the matrix represents an outcome:

- *normal-normal*: true positives, meaning normal points identified as normal
- *normal-legendary*: false negatives, meaning anomalies identified as normal
- *legendary-normal*: false positives, meaning normal points identified as anomalies
- *legendary-legendary*: true negatives, meaning anomalies identified as anomalies

4.3.2 Probabilities

We can compute some probabilities:

- **True positive rate**: $\mathrm{TPR} = \dfrac{\#\{\text{anomalies correctly detected}\}}{\#\{\text{anomalies}\}}$
- **False positive rate**: $\mathrm{FPR} = \dfrac{\#\{\text{normal samples detected as anomalies}\}}{\#\{\text{normal samples}\}}$
- False negative rate: $\mathrm{FNR} = 1 - \mathrm{TPR}$
- True negative rate: $\mathrm{TNR} = 1 - \mathrm{FPR}$
- **Precision**: $\mathrm{P} = \dfrac{\#\{\text{anomalies correctly detected}\}}{\#\{\text{detections}\}}$
- **Recall**: $\mathrm{R} = \mathrm{TPR}$

We always trade-off TPR and FPR through some hyperparameters. In order to assess the method performance, we must choose at least two indicators among TPR and FPR, or **accuracy** and **F1-score**:

$$acc = \frac{\#\{\text{anomalies detected}\} \ + \ \#\{\text{normal samples not detected}\}}{\#\{\text{samples}\}}$$

$$F1 \ score = \frac{\#2\{\text{anomalies detected}\}}{\#\{\text{detections}\} \ + \ \#\{\text{anomalies}\}}$$

Accuracy and F1-score are $= 1$ with ideal methods having no false positives. Since compared methods must be configured in their best conditions, we design the **Receiver Operating Characteristic curve** (**ROC**): the larger the **area under the curve** (**AUC**) for a method, the better the method. The best hyperparameter is the one closer to the point $(0, 1)$.

5 Deep Learning

In a learning framework, there exist two different approaches: the **traditional approach**, which builds on features hand-crafted by experts, and the **deep learning approach**, wherein features are tasks to be learned. In both the approaches, features are **abstracted hierarchically**.

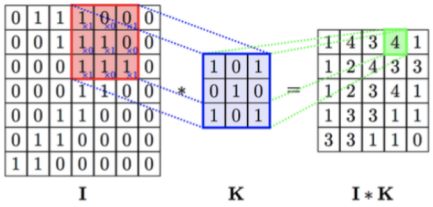
5.1 Convolutional Neural Networks

Given an image as input, **Convolutional neural networks** (**CNNs**) take advantage of space affinities among neighbors through **convolutional** and **pooling layers**, each computing a higher abstract representation making the image shrink at each step. Finally, feature maps are concatenated in a vector and fed to a feedforward neural network.

5.1.1 Convolutional Layers

Convolutional layers evaluate affinities based on the principle of locality by applying a **receptive field** to the image. The kernel K, or filter, contains the parameters to be learned. Many filters can be applied in parallel, each providing a different feature map. Given an original matrix I and a filter K, we obtain the image M:

$$M = I \cdot K$$



5.1.2 Pooling Layers

Pooling layers reduce the image size based on some rules. There are two different operators:

- **Max Pooling**

  The pixel with the maximum values is carried out

- **Average Pooling**

  The average value of the pixels is carried out

5.2 Special Activation Functions

5.2.1 Rectified Linear Unit Activation Function

Stacking many layers may lead to problems when performing backpropagation; the **Rectified Linear Unit** (**ReLU**) mitigates those problems.

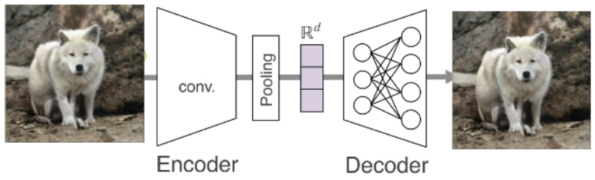
5.2.2 Softmax Activation Function

The **softmax activation function** takes an image as input and outputs a vector representing a probability distribution over the possible classes:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

5.3 Autoencoders

Encoders and **decoders** learn a compact representation of the input space and filter out the noise in input data to generate novel patterns.



6 Results From Theory

6.1 Vapnik Chervonenkis Dimension

Consider a binary classification problem and a family of models $f(\theta, x)$: the **Vapnik Chervonenkis dimension** (**VC**) d_{VC} is the maximum number of points for which there is at least one layout such that the classification makes no errors.

6.2 Uniform Convergence of Empirical Mean

The **Uniform Convergence of Empirical Mean property** (**UCEM**) states that the empirical mean converges to its expectation uniformly as n goes to infinity and for each element of an arbitrarily selected finite sequence of parameter estimates. If M is finite, then the right term goes to 0 as n tends to infinity. The property holds for a family of functions composed of infinite models, provided that the VC dimension is finite:

$$A = \{L(\theta, x), \theta \in \Theta\}$$

6.3 Traditional Statistical Approach

Consider a single hidden layer FNN and a binary output: given d weights, the VC dimension is $O(dlog(d))$. We can extend the results to the regression case by considering the Pollard dimension of family A instead.

6.4 Bias-Variance Tradeoff

We wish to evaluate the **expected test error** for a given x and a quadratic loss function:

$$SE_{PE} = \sigma_\eta^2 + E\left[(g(x) - E[f(\hat{\theta}, x)])^2\right] + E\left[(E[f(\hat{\theta}, x)] - f(\hat{\theta}, x))^2\right]$$

The expectation is taken w.r.t. the realizations of training set and noise. The accuracy performance of the approximating model depends on three terms:

- The variance of the noise, which cannot be canceled out
- The bias, which is the error we should expect on the average; a model with high bias badly approximates the unknown function
- The variance introduced by considering one model, associated with the training set; a model family with high variance is sensitive to the selection of the training set

A complex model has a high variance since it could lead to different results with small changes in the data; on the contrary, a simple model has a low variance since the results are very close to each other. By using more flexible learning methods, the variance increases and the bias decreases. On the contrary, the variance can be decreased by increasing the bias in the parameters.

7 Other Classification Families

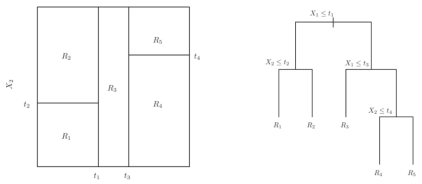
7.1 Classification and Regression Trees

The **Classification and Regression Trees** (**CART**) method partitions the input space in regions assigning a value $\hat{y}_{R_i}$ through a tree of splits. The regions split are learnt from the training set. The **recursive binary splitting** heuristic is:

1. Split along a component d according to a threshold t
2. Identify the branches of a region R:

$$R_{\text{left}}(d,t) = R \cap \{x : x_d \leq t\}$$
$$R_{\text{right}}(d,t) = R \cap \{x : x_d > t\}$$

3. Stop if there are few points in the leaf region



Thresholds are found by minimizing over the estimated value, d, and t. CART can be used for both regression and classification. The key issue consists in the interpretability of results.

7.2 Bagging

Bagging mitigates the high variance obtained by generating different models while training trees on different datasets. Consider taking B independent training sets and building a model on each of them, generating a **forest**; then, we consider the **ensamble** model, meaning the average tree:

$$\hat{f}_{\text{avg}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x)$$

7.3 Maximal Margin Classifier

The **maximal margin classifier** looks at the region of space occupied by the data. Let the separating hyperplane be:

$$w^T x + b = 0$$

We must find the **plus-hyperplane** and the **minus-hyperplane**:

$$w^T x + b = M \geq 0 \;\Rightarrow\; y = 1$$
$$w^T x + b = -M < 0 \;\Rightarrow\; y = -1$$
$$y_i(w^T x_i + b) \geq 1$$

Now we define x_m and x_p as points belonging to the two facing hyperplanes:

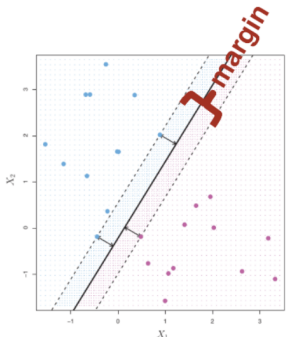
$$w^T(x_p - x_m) = 2$$

The margin is the following projection:

$$\frac{1}{||w||} w^T(x_p - x_m)$$

The margin is evaluated as follows:

$$\frac{2}{||w||}$$



In order to maximize the margin, we solve the relation through quadratic programming:

$$\min_w ||w||_2^2 \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 1 \; \forall i$$

7.4 Support Vector Classifier

When data points are not linearly separable, we use the **Support Vector Classifier** (**SVC**), or soft margin, which tolerates errors by introducing slack variables ε_i:

$$\min_{w,\varepsilon_i} ||w||_2^2 + \lambda \sum_i^n \varepsilon_i$$

The hyperplane is determined by the **support vectors** (**SV**), meaning the points on the wrong side of the plus- and minus-hyperplanes or by those lying on them.

7.5 Kernel and Support Vector Machines

Consider embedding the input data: if the embedded space is rich, then it is easier to solve the problem there. A **Support Vector Machine** (**SVM**) uses a **kernel** to transpose non-linearly separable data in higher dimensions in order to define an hyperplane. There are different types of kernels:

- **Polynomial Kernel**

  Uses a polynomial of degree d:

  $$K(x, x') = (c + x^T x')^d$$

- **RBF Kernel**

  Uses a value γ to indicate the neighborhood tightness and has an infinitely dimensional mapping space:

  $$K(x, x') = e^{-\gamma||x-x'||^2}$$

8 Unsupervised Learning

In unsupervised learning, we aim at discovering properties of a dataset $\{x_1 \dots x_n\}$ with no label y, experiencing dimensionality reduction projecting data on a subspace and clustering data in subgroups.

8.1 Isolation Forest

Isolation Forest (**IFOR**) is a tree-based method that explicitly isolates anomalies: given a random uniform splitting criterion and a k-dimensional tree, it isolates an anomalous point x_0 w.r.t. a genuine point x_i. Anomalies lie in leaves with shallow depth. Given a dataset X_n, for each training set:

1. Select an axis to split
2. Determine a split point
3. Stop splitting if:
   - Depth limit is reached
   - There is one point in the leaf
   - All the points in the leaf have the same value
4. Compute the average path length among all trees given a test point x:

$$E(h(x))$$

5. Point x is anomalous if:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} > T$$

   - $c(n)$ is the average path length in generic binary trees
   - T is the threshold

The average path lengths of x_0 and x_i converge if the number of trees increases.

8.2 Principal Component Analysis

The **Principal Component Analysis** (**PCA**) changes the reference system through rotation: the new axes show the largest data scattering. Given d-dimensional data points $\{x'_1, \ldots, x'_n\}$:

1. Center the data:

$$x_i = x'_i - \frac{1}{n}\sum_{j=1}^{n} x'_j$$

2. Group the data:

$$X = [x_1| \ldots |x_n]$$

3. Compute the sample covariance matrix:

$$\hat{\Sigma} = \frac{1}{n-1} X^T X$$

4. Consider the symmetrical semidefinite positive matrix:

$$H = U \Lambda U^T$$

5. Rotate the axes:

$$U = [U_1| \ldots |U_d]$$

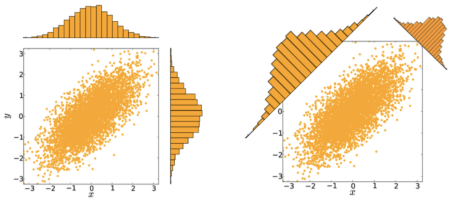
6. Remove the smallest l eigenvalues and group the remaining eigenvectors:

$$\tilde{U} = [U_1| \ldots |U_{l+1}]$$

7. Project the data points reducing the dimension by means of:

$$\tilde{x} = U^T x$$

   Each component of $\tilde{x}$ is a **principal component**



Dimensional reduction is achieved by embedding vectors from d to a d-l dimensional space. The spectral decomposition theorem grants that the information lost in the worst-case is:

$$||H - \tilde{H}||_2 = \lambda_l$$

8.3 K-means Clustering

Clustering groups points so that those in the same clusters are close to each other. Consider a dataset $\{x_1, \ldots, x_n\}$ and a number of clusters K; **K-means clustering** searches for K means and assigns each point to a class:

1. Extract the means μ_K randomly from x_n data points
2. Create K clusters by searching for the minimal Euclidean distance between each μ_K and a point:

$$\hat{k} = \underset{k \in \{1,\ldots,K\}}{argmin} \; ||x_i - \mu_K||_2$$

3. Recompute the means:

$$\mu_K = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$$

4. If clusters change, then repeat from step 2

9 Forecasting

9.1 Time Series

A **time series** X is a sequence of random variable indexed in discrete time order:

$$X = \{X_t \in \mathbb{R}^d \mid t = 0, 1, \dots\}$$

If $d = 1$, then X is **univariate**; otherwise, X is **multivariate**.

9.1.1 Stationarity

A process generating data is **stationary** if the data are realizations of a random variable with a distribution that does not change over time. A process is **weakly-stationary** if only the first two moments of the random variable do not change; its covariance matrix depends on the time difference:

$$T = r - s \quad r, s \in T$$

9.1.2 Time Invariance

A process is **time invariant** if its outputs do not explicitly depend on time (i.e., time does not appear in the equation). On the other hand, a process is **time variant** if its outputs depend on time (i.e. time is in the equation).

9.1.3 White Noise Processes

A **white noise process** is a sequence of independent random variables with mean $\mu_x = 0$ and Var $= \sigma^2$. It is unpredictable since unrelated at different time steps.

9.2 Linear Time Invariance Models

A **Linear Time Invariant model** (**LTI**) predicts linear time series.

9.2.1 AR - Autoregressive Predictive Model

A stationary time series is **autoregressive of order p** (**AR(p)**) if:

$$y(t) = \phi_1 x(t-1) + \cdots + \phi_p x(t-p) + \eta(t) \qquad \eta(t) \sim WN(0, \sigma^2)$$

The AR transfer function is composed of **zeros** over **poles** and represent the solution of A(z) = 0:

$$H(z) = \frac{1}{A(z)} = \frac{z^p}{z^p - \phi_1 z^{p-1} - \cdots - \phi_p}$$

AR models are used if the data has some structure. If each pole has magnitude $\in [0, 1]$, then the system is BIBO stable.

9.2.2 MA - Moving Average Predictive Model

A stationary time series is a **moving average order q** (**MA(q)**) if:

$$y(t) = \theta_1\eta(t-1) + \cdots + \theta_q\eta(t-q) + \eta(t) \quad \eta(t) \sim WN(0, \sigma^2)$$

The MA transfer function is:

$$H(z) = C(z) = \frac{z^q + \theta_1 z^{q-1} + \cdots + \theta_q}{z^q}$$

MA models are used if data show an apparent randomness. MAs are always BIBO stable.

9.2.3 ARMA - Autoregressive Moving Average Predictive Model

An **Autoregressive Moving Average** (**ARMA(p,q)**) model combines AR(p) and MA(q) for stationary time series:

$$y(t) = \eta(t) + \sum_{i=1}^{p} \phi_i x(t-i) + \sum_{j=1}^{q} \theta_j \eta(t-j)$$

The ARMA transfer function is:

$$H(z) = \frac{C(z)}{A(z)}$$

The p + q parameters can be estimated via LMS only if q=0 since lagged errors are unobservable otherwise. They can be estimated through MLE if the process is gaussian, if the noise terms are iid, if data is centered, and if the joint pdf is $f_\Theta = (X_1, \ldots, X_p)$:

$$\frac{\partial \log(L(\Theta|X))}{\partial \Theta} = 0$$

ARMA models are used if data have some structure and there is noise conditioning them. BIBO stability depends on the AR part.

9.2.4 ARIMA - Autoregressive Integrated Moving Average Predictive Models

An **Autoregressive Integrated Moving Average** (**ARIMA(p,d,q)**) model generalizes an ARMA model through differentiation describing non-stationary time series:

$$\hat{y}(t) = \sum_{i=1}^{p} \phi_i g_i(t) + \sum_{j=1}^{q} \theta_j f_j(t)$$

The single optimal step-ahead predictor is:

$$\hat{y}(t) = (1 - H^{-1})y(t)$$

ARIMA models are used when there is a trend that might drift away.

9.3 Sliding Window Approach

In the **sliding window approach**, inputs are regressor vectors of fixed time-lagged data, associated with a window size n_T sliding over time series. A predictor f forecasts the next time series values.

9.4 Exogenous LTI Models

LTI models can include **exogenous variables**, one for each factor that influences data, providing hints for predictions: an AR model becomes an **Autoregressive Exogenous variables** (**ARX**), and an ARMA model becomes an **Autoregressive Moving Average Exogenous variables** (**ARMAX**). Given an exogenous variable $u(t)$, the optimal predictor is:

$$\hat{y}(t) = (1 - H^{-1})y(t) + H^{-1}Gu(t)$$

9.5 Recurrent Neural Networks

Autoregressive non-linear predictive models use past observations to forecast. In **Recurrent Neural Networks** (**RNNs**), hidden states are given their own dynamics and the networks present their own memory. The network state at time t is a summary of past events:

$$h(t) = f_\Theta(h(t-1), x(t))$$

The optimal predictor is:

$$\hat{y}(t) = h(t)V_{RNN}$$

9.5.1 RNNs with Long Short-Term Memory

RNNs with **Long Short-Term Memory** deal with gradient-related problems. The inner state is a linear combination of the old and the new ones.

9.5.2 Gated Recurrent Units

Gated Recurrent Units (**GRUs**) are RNNs with LSTM where the forget gate and the input gate merge. The cell updates $u(t)$ and $r(t)$.

9.5.3 Gated Recurrent Units

Deep RNNs have stacking recurrent layers on top of each other. Given the hidden state $h_l(t)$ at time t for layer l:

$$h_0(t) = x(t) \quad h_l(t) = f_\Theta(h_l(t-1), h_{l-1}(t))$$

9.6 Multi-step Prediction Strategies

9.6.1 Recursive Strategy

In a **recursive strategy**, a single model is trained to perform a one-step ahead forecast given an input sequence. The output is recursively fed back and considered the correct one. The output vector is composed of the predicted n_0 scalars.

9.6.2 Direct Strategy

A **direct strategy** designs n_0 predictors f_k, each forecasting at time $t + k$ and outputting a scalar. The input vector is the same to all the predictors.

9.6.3 Multiple Input - Multiple Output Strategy

In a **Multiple Input - Multiple Output strategy** (**MIMO**), a predictor f is trained to forecast a whole output sequence of length n_0 in one-shot. Differently from the others, the predictor output is a vector.

10 Appendix A - Mathematical Concepts

Independent and Identical Distribution

Data is identically and independently distributed if there is no time dependency among consecutive samples distributed by an identical pdf.

Stationary Points

The stationary point of a function are those points where the derivative is 0 and where the function is neither increasing nor decreasing.

Realization of a Random Variable

The realization of a random variable is the result of applying it to an experiment observed outcome.

Transfer Function

A transfer function of a system models its output for each possible input.

BIBO Stability

A system is BIBO stable if each output of a bounded input is bounded.