Machine Learning

Federico Matteoni

A.A. 2021/22

Index

0.1 Introduction

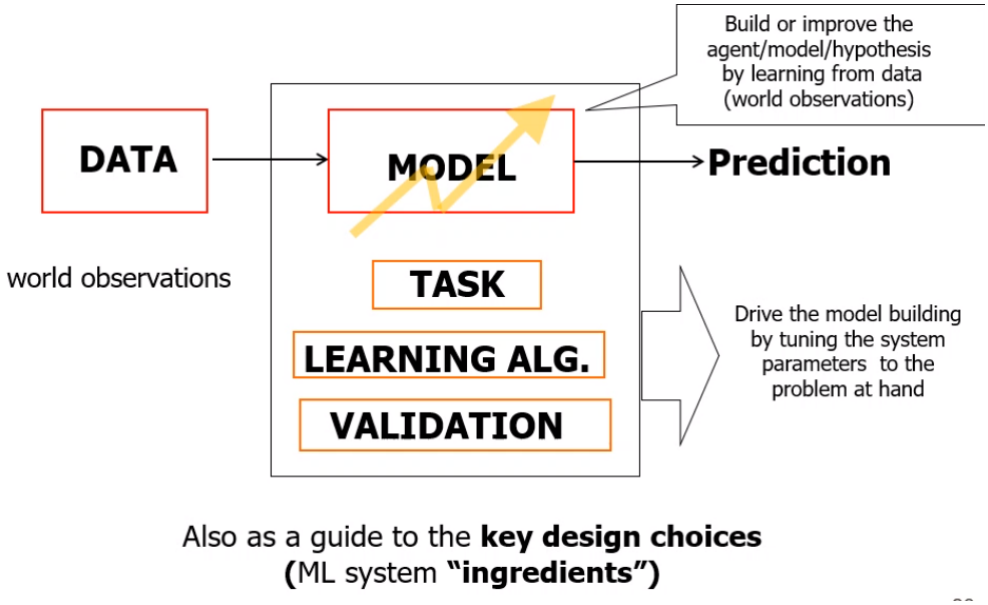
What is ML? Area of research combining aims of creating computers that could learn and powerful and adaptive statistical tools with rigorous foundation in computational science. Luxury or necessity? Growing availability and need for analysis of empirical data and difficult to provide intelligence and adaptivity by programming it. Change of paradigm.
Examples: spam classification, written text recognition... No or poor prior knowledge and rules for solving the problem, but easier to have a source of training experience.
ML is considered the latest general-purpose technology, capable of drastically affect pre-existing economic and social structures. And already has. The ultimate aim is to bring benefits to the people by solving big and small problems, accelerating human progress and empowering humans to add intelligence in any other science field.

Machine Learning We restrict to the computational framework: principles, methods and algorithms for learning and prediction, from experience. Building a model to be used for predictions. Common framework: infer a model or a **function** from a set of examples which allows the generalization (accurate response to new data).
When can we use ML? Be aware of the opportunity and awareness. ML is useful when there's no or poor theory surrounding the phenomenon, or uncertain, noisy or incomplete data which hinders formalization of solutions. The requests are: source of training experience (representative data) and a tolerance on the precision of results. The best examples are models to solve real-world problems that are difficult to be treated with traditional techniques: face and voice recognition (knowledge too difficult to formalize in an algorithm), predicting biding strength of molecules to proteins (not enough human knowledge) and personalized behavior, such as recommendation systems, scoring messages according to user preferences...

Definition The ML studies and proposes methods to build functions/hypothesis from examples of observed data that fits the known examples and able to generalize, with reasonable accuracy, for new data (according to verifiable results and under statistical and computational conditions and criteria.

Data Data **represents the available experience**. Representation problem: capturing the structure of the analyzed objects. Flat (attribute-value), structured..., categorical or continuous, missing data... **preprocessing**: variable scaling, encoding, selection...

Task The task defines the purpose of the application: knowledge that we want to achieve? which is the helpful nature of the result? what information are available?
Predictive task, classification and regression: function approximation
Descriptive task, cluster analysis and association rules: find subsets or groups of unclassified data.



Also as a guide to the **key design choices**
(ML system **"ingredients"**)

Supervised learning Given a set of training examples as ⟨input, output⟩ = $\langle x, d \rangle$ (**labeled examples**) for an unknown function f, find a *good approximation* of f, an hypothesis h that can be used for making predictions on unseen data x'.
The target d can be:

> Discrete value, for **classification tasks**.
> $f(x) \in \{1, 2, \ldots, k\}$
> Patterns, feature vectors, are seen as members of a class and the goal is to assign the new patterns observed to the correct class (or label)
> If the number of possible classes is two, then f is a *boolean function* and the task is called **binary classification** or **concept learning**: true or false, positive or negative, 0 or 1...
> If the number of classes is greater that two then is a **multi-class classification task**.
>
> Real continuous value, for **regression tasks**.
> The patterns are seen as sets of variables (real values), and the task is a curve fitting task. The process aims to estimate a real-value function based of a finite set of noisy samples $\langle x, f(x) +$ random noise$\rangle$

Unsupervised learning No teacher. The training set is a set of unlabeled data $\langle x \rangle$. Examples: clustering, finding natural groupings in a set of data.

Learning algorithm Basing on data, task and model: heuristic search through the hypothesis space H of the **best hypothesis**. I. e. the best approximation of the unknown target function, typically searching for the h with the minimum *error*. H may not coincide with the set of all possible function and the search cannot be exhaustive, we need to make **assumpitons** (**inductive bias**).

Learning Also called:

> Inference, in statistics
>
> Adapting, in biology and systems
>
> Optimizing, in mathematics
>
> Training, in neural networks
>
> Function approximations, in mathematics
>
> ...

After introducing data, task, model and learning algorithm we will focus on: inductive bias, loss and concepts of generalization and validation.

Inductive bias To set up a model we can make assumptions about the nature of the target function, concerning either:

> constraints in the model, **language bias** (in the hypothesis space H, due to the set of hypothesis that we can express or consider
>
> constraints or preferences in learning algorithm/search strategy, **search bias** which is preferred
>
> or both

Such assumptions are needed to obtain an useful model for the ML aims, i.e. a model with generalization capabilities. We can imagine learning a discrete function with discrete inputs assuming **conjunctive rules**, so using a **language bias** to work with a restricted hypothesis space.

Version Space An hypothesis h is consistent with the TR if $h(x) = d(x)$ for each training example $\langle x, d(x) \rangle$.
The **version space** $VS_{H,TR}$ is the subset of H of the hypothesis consistent with all the training examples $\langle x, d(x) \rangle$ in the TR.
It's possible to do an exhaustive search in an efficient way, using clever algorithms. This means finding the set of all the hypothesis h consistent with the TR set.

Unbiased Learner The language bias (ex: using only conjunctive rules, may be too restrictive: if the target concept is not in H it cannot be represented in H. We can use an H that expresses every teachable concept (among propositions), that means that H is the set of all possible subsets of X: the power set $P(X)$. If $n = 10$ binary inputs, then $|X| = 2^{10} = 1024$ and $|P(X)| = 2^{1024} = 10^{308}$ possible concepts, which is much more than the number of the atoms in the universe.
An unbiased learner is unable to generalize: the only examples that are unambiguously classified by an unbiased learner represented with the VS are the training examples themselves. Each unobserved instance will be classified positively by exactly half of the hypothesis in the VS and negative by the other half. Indeed: $\forall\, h$ consistent with x_i, $\exists\, h'$ identical to h except $h'(x_i) \neq h(x_i)$, $h \in \text{VS} \Rightarrow h' \in \text{VS}$ (because they are identical on the TR)

Why prefer the search bias? In ML we use flexible approaches (expressive hypothesis spaces with universal capability of the models, for example neural networks or decision trees. We avoid the language bias, so we do not exclude a priori the unknown target function, but we focus on the search bias (ruled by the learning algorithm).

Loss How to measure the quality of an approximation? We want to measure the distance between $h(x)$ and d, using a loss function/measure $L(h(x), d)$ for a pattern x which has high value in cases of bad approximation. The error (or risk or loss) is an expected value of this L, for example $E(w) = \frac{1}{l}\sum_{p=1}^{l} L(h(x_p), d_p)$. Different L for different tasks. Examples of loss functions:

Regression: $L(h(x_p), d_p) = (d_p - h(x_p))^2$, the squared error. MSE (mean squared error) over the data set

Classification: $L(h(x_p), d_p) = \begin{cases} 0 & h(x_p) = d_p \\ 1 & else \end{cases}$

Learning and generalization Learning: search for a **good function** in a function space from known data (typically minimizing an error/loss). **Good** with respect to generalization error: it measures how accurately the model predicts over novel samples of data (**measured over <u>new</u> data**).
Generalization is the crucial point of ML. Performance in ML is the generalization accuracy or *predictive accuracy* estimated by the error on the test set.

ML issues Inferring general functions from known data is an ill posed problem, which means that in general the solution is not unique because we can't expect the exact solution with finite data. What can we represent? And so, what can we learn?
Learning phase: building the model including training. The prediction phase is evaluating the learned function over new never-seen-before samples (generalization capability). Inductive learning hypothesis: any h that approximates f well on training examples will also approximate f well on new unseen instances x.
Overfitting: a learner overfits data if it outputs an hypothesis $h \in H$ having true/generalization error (risk) R and empirical (training) error E, but there's another $h' \in H$ with $E' > E$ and $R' < R$, which means that h' is the better one despite having a worse fitting.

Statistical Learning Theory Under what mathematical conditions is a model able to generalize? We want to investigate the generalization capability of a model, measured as a risk or test error, the role of the model complexity and the role of the number of data.
Formal Setting: approximate a function $f(x)$, with d target ($d = f(x)+$ noise), minimizing the **risk function**

$$R = \int L(d, h(x))\, dP(x, d)$$

which is the **true error over all the data**, given:

a value d from the teacher and the probability distribution $P(x, d)$

a loss function $L(h(x), d) = (d - h(x))^2$

We search for $h \in H \mid min\, R$, but we only have the finite data set $TR = (x_p, d_p)$ with $p = 1 \ldots l$. Looking for h means minimizing the empirical risk (the training error E), finding the best values for the model free parameters

$$R_{emp} = \frac{1}{l}\sum_{p=1}^{l}(d_p - h(x_p))^2$$

The inductive principle is the **ERM**, Empirical Risk Minimization: can we use R_{emp} to approximate R?

Vapnik-Chervoneniks dim and SLT Given the VC dimension (simply VC), a measure of complexity of H and by that we mean its flexibility to fit data.
The VC-bound states that it holds with probability $\frac{1}{\delta}$ that

$$R \leq R_{emp} + \epsilon\left(\frac{1}{l}, VC, \frac{1}{\delta}\right)$$

ϵ is a function called VC-confidence, that grows with VC and decreases with higher l and δ

R_{emp} decreases using complex models (with high VC)

δ is the confidence, and it rules the probability that the bound holds.
$\delta = 0.01 \Rightarrow$ the bound holds with probability 0.99

Intuitively:

Higher l (data) $\Rightarrow$ lower VC confidence and bound closer to R

A too simple model, meaning with low VC, can be not sufficient due to high R_{emp} (**underfitting**)

An higher VC with fix $l \Rightarrow$ lower R_{emp} but VC and hence R may increase (**overfitting**)

Structural risk minimization Minimize the bound! There are different bounds formulations according to different classes of f, of tasks...
In other words, we can make a good approximation of f from examples, provided that we have a good number of data and the complexity of the model is suitable for the task.

Complexity control The Statistical Learning Theory allows for a formal framing of the problem of generalization and overfitting, providing an analytic upper bound to the risk R for the prediction over all the data, regardless of the type of learning algorithm or the details of the model. So **the machine learning is well founded**, the learning risk can be analytically limited and only a few concepts are fundamental. This leads to new models (such as the Support Vector Machine) and other methods that directly consider the control of the complexity in the construction of the model.

Validation Central role for the applications and the project. Two aims:

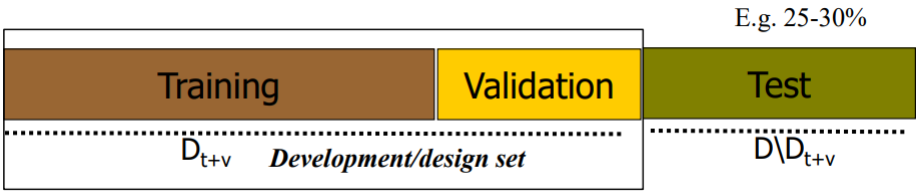
Model Selection: estimating the performance (**generalization error**) of different models in order to choose the best one. This includes searching for the best hyperparameters of the model.
It returns a model.

Model Assessment: with the final model, estimating/evaluating its prediction error/risk (**generalization error**) over new test data.
It returns an estimation.

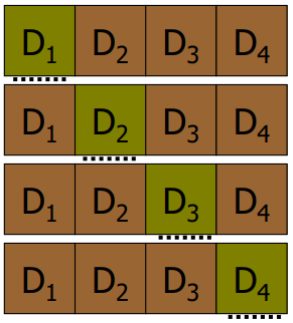
Golden rule: keep the two goals separated and use different datasets for each one.

In an ideal world, we'd have a large training set, a large validation set for model selection and a very large external unseen data test set. With finite and often small data sets we have just an estimation of the generalization performance. We have to use some techniques: hold-out and k-fold cross validation, for example.

Hold-Out: we partition the dataset D into **training set** TR, **validation/selection set** VL and **test set** TS. All three are disjoint: TR is used to run the training algorithm, VL can be used to select the best model (hyperparameters tuning) and the **TS is only used for model assessment**.



K-Fold: this technique can make use of insufficient data. We split the dataset D into k mutually esclusive subsets $D_1, \ldots, D_k$, we train on $D - D_i$ and test it on D_i.
This can be applied to both VL and TS splitting. Can be computationally very expensive and there's the issue of choosing the number of folds k.



Confusion Matrix

Actual/Predicted	**Positive**	**Negative**
Positive	TP	FN
Negative	FP	TN

Specificity $= \frac{TN}{FP+TN}$, and **true negative rate** $= 1 - FPR$
Sensitivity $= \frac{TP}{TP+FN}$, also known as **true positive rate** or **recall**
Precision $= \frac{TP}{TP+FP}$
Accuracy: % of correctly classified patterns $= \frac{TP + TN}{total}$. Note that, for example, a 50% accuracy on a binary classifier is equivalent to a random predictor.

ROC Curve We plot **specificity** on **x-axis** and **sensitivity** on the **y-axis**. The diagonal corresponds to the worst classifier, the random guesser. Better curves have greater Area Under the Curve (AUC)

Linear Models Mainstay of statistics.

Univariate Linear Regression Simple linear regression: we start with 1 input variable x and 1 output variable y. We assume a model $h_w(x)$ expressed as $out = w_1 x + w_0$ where w are real-valued coefficients or **free parameters**, also called **weights**.
Given that the ws are continuous valued, we have an infinite hypothesis space but a nice solution from classical math. We can learn with this basic tool and, although simple, it includes many relevant concepts of modern ML and many methods in the field are based on this.
Least Mean Square: learning means finding w such that it minimizes the error/empirical loss, with best data fitting on the training set with l examples.
So given a set of l training examples (x_p, y_p) with $p = 1, \ldots, l$, we have to find $h_w(x)$ in the form $w_1 x + w_0$ that minimizes the expected loss on the training data. For the loss, we use the square of errors: **least mean square**, find w to **minimize** the residual sum of squares.

$$Loss(h_w) = E(w) = \sum_{p=1}^{l} (y_p - h_w(x_p))^2 = \sum_{p=1}^{l} (y_p - (w_1 x_p + w_0))^2$$

To have the mean, divide by l. How to solve? Local minimum as stationary point, so the gradient $\frac{\partial E(w)}{\partial w_i} = 0$ with $i = 1, \ldots,$ dim_input $+1 = 1, \ldots, n+1$. For the simple linear regression (2 free parameters):

$$\frac{\partial E(w)}{\partial w_0} = 0 \quad \frac{\partial E(w)}{\partial w_1} = 0$$

$$\frac{\partial E(w)}{\partial w_0} = -2(y - h_w(x)) \quad \frac{\partial E(w)}{\partial w_1} = -2(y - h_w(x)) \cdot x$$

Classification The same models used for regression can be used for classification: **categorical targets** y or d, for example 0/1, -1/+1...
We use an hyperplane (wx) assuming negative or positive values. We exploit such models to decide if a point x belongs to the positive or the negative zone of the hyperplane to classify it. So we want to learn w such that we get a good classification accuracy. The decision boundary is $x^T w = w^T x = w_0 + w_1 x_1 + w_2 x_2 = 0$ and we can introduce a threshold function which can be written in many ways:

$$h(x) = \begin{cases} 1 & \text{if } wx + w_0 \geq 0 \\ 0 & \text{else} \end{cases}$$

$$h(x) = sign(wx + w_0)$$

...

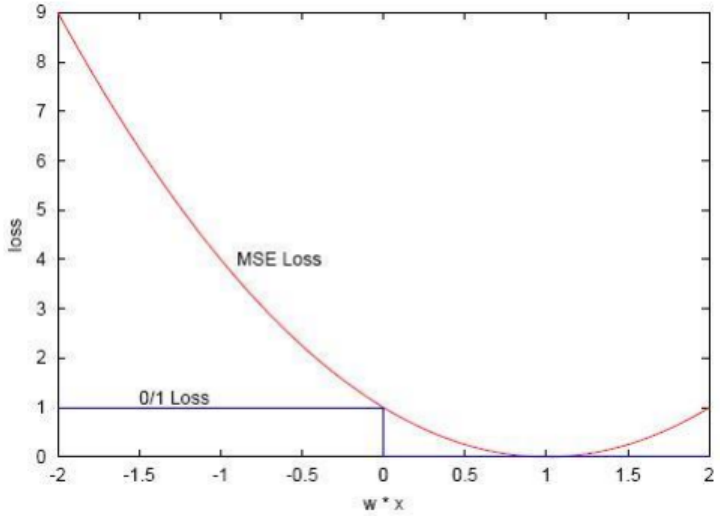
w_0 is called **threshold** or **bias**. $h(x) = w^T x + w_0 \geq 0 \Leftrightarrow w^T x \geq -w_0$

Learning Algorithms Introducing 2 learning algorithms, both based on LSM and used for the linear model on regression and classification tasks. We start redefining the learning problem and the loss for them (in the case of l data and multidimensional inputs).

Learning problem for classification tasks: given a set of l training examples (x_p, y_p) and a loss function L, with $y_p \in \{0, 1\}$ or $y_p \in \{-1, +1\}$, find the weight vector w that minimizes expected loss on the training data

$$R_{emp} = \frac{1}{l} \sum_{p=1}^{l} L(h(x_p), y_p)$$

The expected loss can be approximated by a smooth function. We can make the optimization problem easier by replacing the original objective function L (0/1 loss) with a smooth, differentiable function: for example, the MSE loss (mean squared error).



No classification error minimizing either 0/1 loss or MSE loss. Given l training examples (x_p, y_p), find w that minimizes the residual sum of squares

$$E(w) = \sum_{p=1}^{l} (y_p - x_p^T w)^2 = ||y - Xw||^2$$

We can't use $h(x)$ in $E(w)$, as for regression, because $h(x) = sign(w^T x)$ is non-differentiable. Also, this is a quadratic function so the minimum always exists (but may not be unique). X is a $l \times n$ matrix with a row for each x_p.

Direct Approach with a normal equation Differentiating $E(w)$ with respect to w to get the **normal equation**

$$(X^T X)w = X^T y$$

In the derivation we also find that

$$\frac{\partial E(w)}{\partial w_j} = -2 \sum_{p=1}^{l} (x_p)_j \cdot (y_p - x_p^T w)$$

If $X^T X$ is not singular, then the unique solution is given by

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

with X^+ being the Moor-Penrose pseudoinverse. Else the solutions are infinite, so we can choose the **min norm**(w) solution.
The **Singular Value Decomposition** can be used for computing the pseudoinverse of a matrix (X^+). With $X = U\Sigma V^T \Rightarrow X^+ = V\Sigma^+ U^T$ replacing every non-zero entry by its reciprocal. We can apply SVD directly to compute $w = X^+ y$, obtaining the minimal norm (on w) solution of least squares problem.

$$\frac{\partial E(w)}{\partial w_j} = \frac{\partial \sum_{p=1}^{l}(y_p - x_p^T w)^2}{\partial w_j} = \ldots = -2\sum_{p=1}^{l}(y_p - x_p^T w)(x_p)_j$$

Gradient Descent The derivation suggests an approach based on an iterative algorithm based on $\frac{\partial E(w)}{\partial w_j} = -2\sum_{p=1}^{l}(y_p - x_p^T w)(x_p)_j$. The **gradient** is the **ascent direction**. We can move toward the minimum with a gradient descent $\Delta w = -$ gradient of $E(w)$. **Local search**: we begin with a initial weight vector and modify it iteratively to minimize the error function. The gradient vector is

$$\Delta w = -\frac{\partial E(w)}{\partial w} = \begin{bmatrix} -\frac{\partial E(w)}{\partial w_1} \\ \vdots \\ -\frac{\partial E(w)}{\partial w_n} \end{bmatrix} = \begin{bmatrix} \Delta w_1 \\ \vdots \\ \Delta w_n \end{bmatrix}$$

Allowing us to work in a multi dimensional space without the need to visualize it. Hence, the iterative approach will move using a learning rule based on a "delta" of w proportional to the opposite of the local gradient. The movements will be made according to

$$w_{new} = w + \eta \cdot \Delta w$$

The simple algorithm is as follows:

1. Start with weight vector $w_{initial}$ and fix $0 < \eta < 1$
2. Compute $\Delta w = -$ gradient of $E(w) = -\frac{\partial E(w)}{\partial w}$ (or for each w_i)
3. Compute $w_{new} = w + \eta \cdot \Delta w$ (or for each w_i)
   η is the step size or **learning rate**
4. Repeat from 2 until convergence or $E(w)$ sufficiently small

Batch version The gradient is the sum over all the l patterns. Provides a more precise evaluation of the gradient over l data. We upgrade the weight after the sum

$$\frac{\partial E(w)}{\partial w_j} = -2\sum_{p=1}^{l}(y_p - x_p^T w)(x_p)_j$$

Online/Stochastic version We upgrade the weights with the error that is computed for each pattern. Hence, the second pattern output is based on weights already updated from the first and so on. In makes progress with each example it sees. Can be faster, but needs smaller η

$$\frac{\partial E_p(w)}{\partial w_j} = -2(y_p - x_p^T w)(x_p)_j = -\Delta_p w_j$$

Gradient Descent as error correction delta rule The error correction rule, also called Widrow-Hoff or delta rule, changes w_j proportionally to the error (target y − output)

$$\Delta w_j = 2\sum_{p=1}^{l}(x_p)_j(y_p - x_p^T w)$$

$$w_{new} = w + \eta \cdot \Delta w$$

We improve by learning on previous errors.

Gradient descent is a simple and effective local search approach to a LMS solution. It allows to search through an infinite hypothesis space, can be easily applied for continuous H and differentiable losses and isn't only for linear models (also neural networks and deep learning models).
Many possible improvements (Newton, quasi-Newton methods, conjugate gradients...)

Linear models

Language bias: H is a set of linear functions.

Search Bias: ordered search guided by the least squares minimization goal. For instance, we could prefer a different method to obtain a restriction on the values of parameters, achieving a different solution with other properties.

Shows that even for a simple model there are many possibilities. We still need a principled approach.

Limitations In geometry, two set of points are linearly separable in an n-dimensional space if they can be separated by a $(n-1)$-dimensional hyper-plane. In 2 dimensions, if they can be separated by a line, in 3 dimensions, by a plane...
The linear decision boundary can provide exact solutions only for linearly separable sets of points.

Extending the linear model We can use transformed inputs, such as $x, x^2, x^3 \ldots$ with a non-linear relationship between inputs and output, maintaining the learning machinery used so far.

$$h_w(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Linear basis expansion (LBE)

$$h_w(x) = \sum_{k=0}^{K} w_k \phi_k(x)$$

Augments the input vector with additional variables which are transformations of x according to a function $\phi_k : R^n \rightarrow R$, so number of parameters $K > n$: linear in the parameters, so we can use the same learning algorithms as before.
Which ϕ? Towards the dictionary approaches. Pro: can model more complicated relationships than linear, so it's more expressive. Cons: with large basis of functions, we easily risk overfitting, hence we require controlling the complexity (as in flexibility of the model to fit the data). How to do that? Many approaches:

Ridge Regression (or **Tikhonov Regularization**): smoothed model.
Add constraints to the sum of value of $|w_j|$, penalizing models with high values of $|w|$ (so favoring sparse models, using less terms due to weights $w_j = 0$ or close)

$$Loss(w) = \sum_{p=1}^{l} (y_p - x_p^T w)^2 + \lambda ||w||^2$$

with λ being the regularization hyper-parameter. It implements the control of the model complexity, leading to a model with less VC-dim with a trade-off controlled through a single parameter, λ.
This uses $||\,||_2$

Lasso uses $||\,||_1$

Elastic nets uses both $||\,||_1$ and $||\,||_2$

Learning Timing **Eager**: analyze data and construct an explicit hypothesis
Lazy: store tr data and wait test data point, then construct an ad hoc hypothesis.

K-NN

Voronoi Diagram Each cell consists of point closer to x than any other patterns. The segments are all points in plan equidistant to two patterns. It is implicitly used by K-NN.

Artificial Neuron Input from external source or other units, with weights w as free parameters: can be modified by learning. The unit i computes $f(\sum_j w_{ij} x_j)$ with w_{ij} the weight from input j to unit i. f is called **activation function**: linear, threshold or logistic (sigmoid).

Perceptron A neuron that uses a threshold as activation function. Can be composed and connected to build a network.

Xor $x_1 \oplus x_2 = x_1 \cdot \overline{x_2} + \overline{x_1} \cdot x_2$. Let $h_1 = x_1 \cdot x_2, h_2 x_1 + x_2$ then $x_1 \oplus x_2 = \overline{h_1} \cdot h_2$ with $\wedge = \cdot$ and $\vee = +$.
So two layers are sufficient, but single layer cannot model all functions due to limits of single perceptron and the linear separable problems.

Learning for one unit model

1. **Adaline**, adaptive linear neuron: LMS direct solution and gradient descent solution
2. **Perceptron**, non linear: only classificiation
   Minimize number of misclassified patterns
   (a) initialize weights
   (b) pick learning rate η (between 0 and 1)
   (c) until stopping condition (es weights don't change)

   For each training pattern (x, d) compute output activation $out = sign(w^T x)$. If $out = d$ don't change weights, if $out \neq d$ update weights $w_{new} = w + \eta \cdot d \cdot x$ adding $+\eta x$ if $wx \leq 0$ and $d = +1$ or $-\eta x$ if $wx > 0$ and $d = -1$.
   Different form $w_{new} = w + \frac{1}{2} \cdot \eta \cdot (d - out) \cdot x$

Perceptron convergence theorem

Preliminaries

Proof

Activation functions

Excercise Compute the gradient

Gradient descent algorithm

Neural Network In a MLP architecture: units connected by wighted links, organized in layers:

input layer, source of the input x

hidden layer, projects onto another hidden or an output layer

Can be viewed as network of units o flexible function:

$$h(x) = f_k \left(\sum_j w_{kj} f_j \left(\sum_i w_{ji} x_i \right) \right)$$