Introduction to *In-Silico* Learning

Machine Learning Algorithms

Giacomo Bergami

Introduction

Machine Learning Algorithms

There are three main models of

- Supervised learning: any method which uses a training set distinguishing positive from negative examples.
- Unsupervised learning: any method guessing the data properties via its data distribution (e.g., `clustering`).
- Reinforcement learning: the agent learns from a series of rewards and punishments, but no direct clue is given.

Given that the majority of unsupervised techniques can be inferred from specific properties of supervised learning approaches, we're only going to study the following supervised learning algorithms for:

- Multilayer Neural Networks
- ν-Support Vector Machines with Kernel Trick
- Decision Trees (CART)

Use Case Scenario: StarCraft II

- **StarCraft II** is a science fiction real-time strategy video game.
- The authors used replay analysis to extract some behaviour predictors
  - E.g., number of workers created, number of actions per second, hours played.
- We want to predict the XP level of each player. All the provided data is numeric.

`http://summit.sfu.ca/item/13328`

Static Report

- It's the first step for decision support
- A software retrieves operational data from data sources, aggregates the pieces of information and outputs a table (e.g., *StarCraft II Replay*)
- Such result is *static* because it snapshots a given database configuration.
- It is useful for analysts that need a regularised and ready to use output.

More info at
`https://drive.google.com/file/d/0B3tBL-tX2EdQV2Q4Znh5UnBrTXM/v`

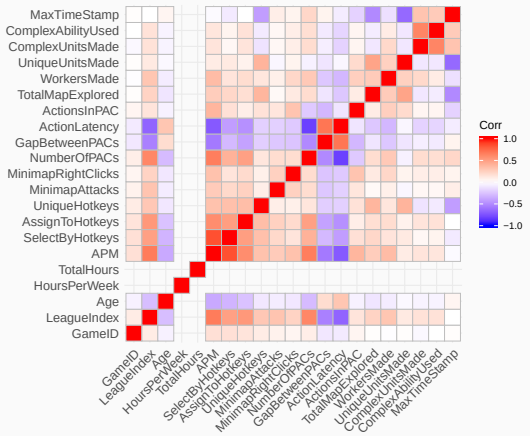
Correlation Matrix (1/2)

- A correlation matrix contains the p-values ρ_{XY} of the correlation between two variables X and Y.
  - ρ_{XY} returns a value in $[-1, 1]$, where a value near 0 means there is almost no correlation, and the sign of the correlation returns whether the correlation is positive or negative.

$$\rho_{XY} := \frac{\sum_{1 \leq i \leq n}(x_i - \mu_X)(y_i - \mu_Y)}{\sqrt{\sum_{1 \leq i \leq n}(x_i - \mu_X)^2}\sqrt{\sum_{1 \leq i \leq n}(y_i - \mu_Y)^2}}$$

  - $\mu_X := \frac{1}{n}\sum_{1 \leq i \leq n} x_i$ is the sample mean for all the values of X.
  - $\sigma_X = \sqrt{\frac{1}{n-1}\sum_{1 \leq i \leq n}(x_i - \mu_X)}$ is the sample standard deviation.

Covariance Matrix (2/2)

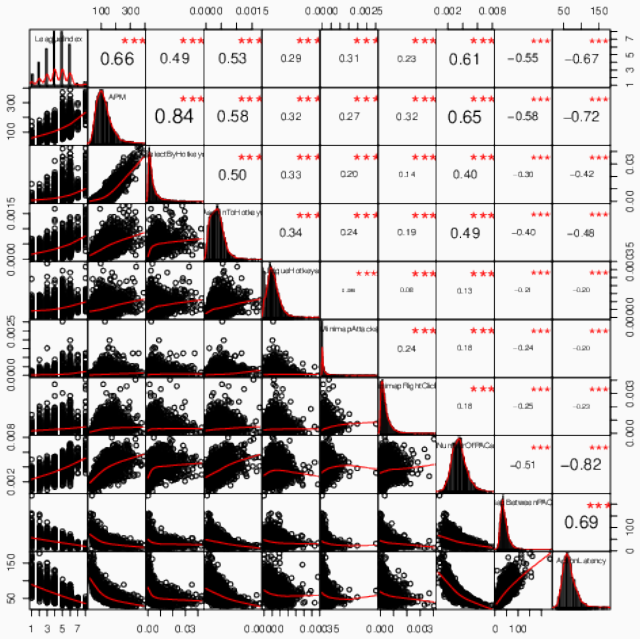


- We are interested in predicting the League Index (XP) from the other parameters: as we might see from the data, all the variables in the range from `APM` to `ActionLatency` are the best predictors.

Scatterplot Matrix (1/2)

- A Scatter-plot Matrix allows the bi-variate profiling of N metric variables.
  - The scatter-plots below the diagonal show possible correlations between the data.
  - The diagonal provides the uni-variate profile of each variable.
  - The numbers above the diagonal are the ones provided by the covariance matrix.
- We might restrict the Scatterplot Matrix plot to the best predictors for the XP level.
  - We might use such restriction as well to reduce the data dimensionality while training our models.

Scatterplot Matrix (2/2)



Multilayer Neural Networks

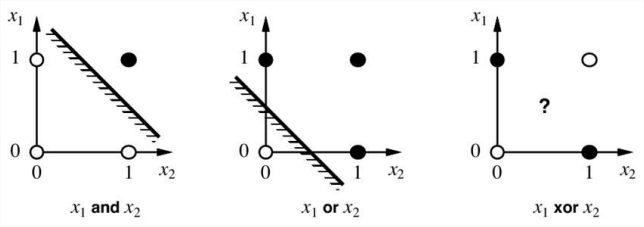
ADALINE (Widrow & Hoff, 1960)

- ADALINE stands for *ADAptive LInear NEuron*
- Each ADALINE neuron calculates the *net* weighted sum of all of its k inputs, and considers a bias value $\theta \in \mathbb{R}$:

$$f_{\vec{W},\theta}(\vec{x}) = \varphi(\vec{W}^t\vec{x} + \theta)$$

- φ is referred as activation function, while $net = \vec{W}^t\vec{x} + \theta$.
  - For ADALINE, $\varphi = \text{sign}$.
- Information moves only from the input to the output (feed-forward network).

Linearly Separable Problems



The *XOR* function is not a linearly separable problem (*Russel & Norvig*))

Two sets of points $E_0, E_1 \subseteq \mathbb{R}^n$ in $(\mathbb{R}^n, \preceq)$ are **linarly separable** if there exist real numbers in W and θ with $|W \cup \{\theta\}| = n + 1$ such that:

- $\forall x \in E_0. \sum_{0 \leq i \leq n} w_i x_i > \theta$
- $\forall x \in E_1. \sum_{0 \leq i \leq n} w_i x_i < \theta$

ADALINE: Problems

As we can see, ADALINE can only act as a binary classificator.

- ADALINE could only solve a small set of problems, those are the linearly separable problems.
- Even though we use the sigmoid/logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$ as an activation φ, we will still obtain a near (hyper)plane separation.
- In order to solve linearly separable problems, we need to combine different linear solvers together.

Backpropagation (Rumelhart, Hinton & Williams, 1986)

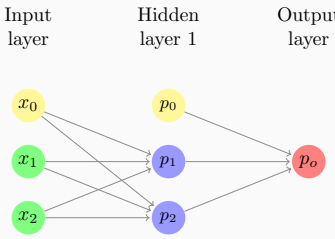
In their 1986 paper, they introuced the following novelites:

1. Hidden Layers, intermediate layers between the input and the output ones.
2. Backpropagation, a learning algorithm for multiple connected layers, used to propagate the *loss* value backwards.

Each perceptron now becomes:

$$f_{\vec{W},\theta}(\vec{x}) = \frac{1}{1 + e^{-(\vec{W}^t\vec{x}+\theta)}}$$

Hidden Layers



- The input layer corresponds to just some input values.
- Each incoming edge for a neuron p in the graph represents a weight $\vec{W}$ for each one of its input.
  - The bias θ is represented as a weight θ for an input (in yellow) which is always supposed to be 1.
- The network computes the following final function:

$$\tilde{h}(x,y) = p_o(p_1(x,y),\ p_2(x,y))$$

Backpropagation (1/2)

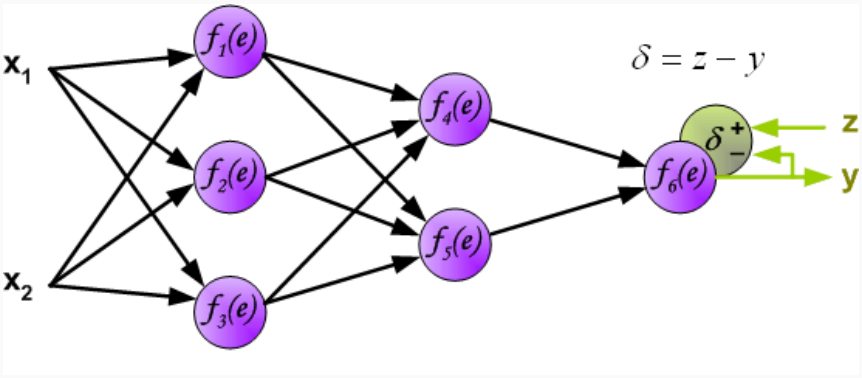
We need to propagate the error signal in the opposite direction:

- Each network with L layers will compute the following function:

$$\tilde{h}(x) = f^L_{W^L,\theta^L}(f^{L-1}_{W^{L-1},\theta^{L-1}}(\cdots f^1_{W^1,\theta^1}(x)\cdots)) \quad (1)$$

- Given a training data sample $(x,y) \in E$, the resulting loss value is $loss(N,x)$.
- We can compute total *loss* between the output neuron and the expected output for each element that was provided as an input:
  - $\mathcal{L} = \frac{1}{2}\sum_{(x,y)\in E}(\tilde{h}(x)-y)^2$
- We want to minimize the total loss that we committed:
  - From *Mathematical Analysis*, we know that E can be minimized when its partial derivatives with respect to the weights W and θ are zero.
  - For non-linear models, we need to use η learning rate because minimum loss functions might not have closed-form solution.

Error computation from the Output Layer



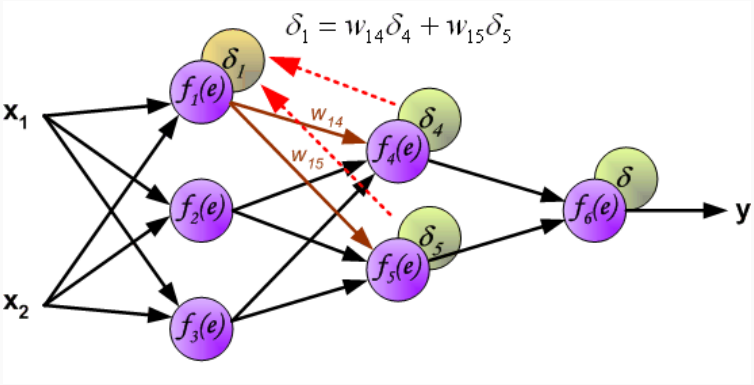
Backpropagation (2/2)

As per last observations and with $\tilde{h}(x) \approx \varphi(net)$ for the output layer, we can then express the weight variation that we need to apply to the perceptron as follows:

$$\begin{aligned}
\Delta w_j = \eta \frac{\partial \mathcal{L}}{\partial w_j} &= \eta \frac{\partial \mathcal{L}}{\partial \varphi(net)} \frac{\partial \varphi(net)}{\partial w_j} = \eta \frac{\partial \mathcal{L}}{\partial \varphi(net)} \left(\frac{\partial \varphi(net)}{\partial net} \frac{\partial net}{w_j} \right) \\
&= \eta \frac{\partial \mathcal{L}}{\partial \varphi(net)} \varphi'(net) \frac{\partial \sum_i w_i x_i + \theta}{\partial w_j} \Bigg|_{\varphi(net)=\tilde{y}} \\
&= \eta \frac{\partial \mathcal{L}}{\partial \tilde{y}} \ \varphi'(net) \ x_j \\
&= \eta \ (\tilde{y} - y) \ \varphi'(net) \ x_j
\end{aligned}$$

The contribution $\delta_j = (\tilde{y} - y) \ \varphi'(net)$ called derivative is the one backpropagated in the network.

Backpropagation: Example (1/2)

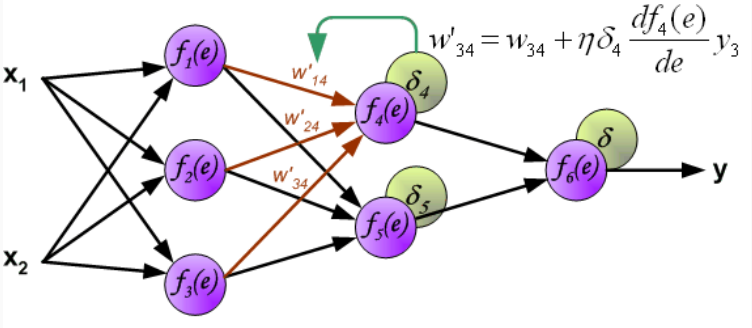


Backpropagation: Example (1/2)

$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



Backpropagation with Momentum

In order to attenuate the oscillations during the gradient descent, we can introduce an α parameter, so to smothen the transition from the current $w_j + \Delta w_j$ to the previous value of Δw_j:

$$\Delta w_j^{(t+1)} = \eta \ (\tilde{y} - y) \ \varphi'(net) \ x_j + \alpha \Delta w_j^{(t)}$$

XOR Trained Network Example (1/4)

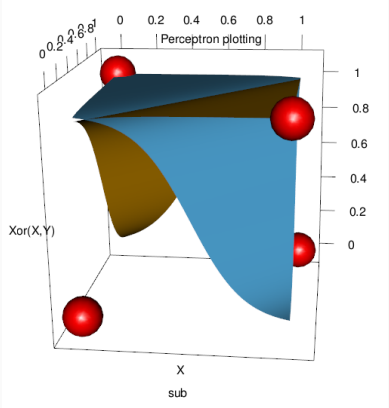
- Each neuron in the network, after the training and the back propagation algorithm, becomes like this.

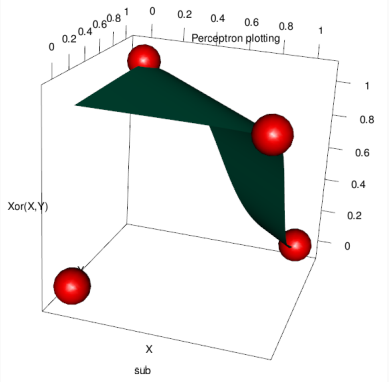
$$p_1(x, y) = \frac{1}{1 + e^{-(7.10607 \cdot x - 6.50904 \cdot y + 3.41603)}}$$

$$p_2(x, y) = \frac{1}{1 + e^{-(-8.05875 \cdot x + 8.086 \cdot y + 4.31492)}}$$

$$p_o(x, y) = \frac{1}{1 + e^{-(-8.65042 \cdot x - 8.48615 \cdot y + 12.8072)}}$$

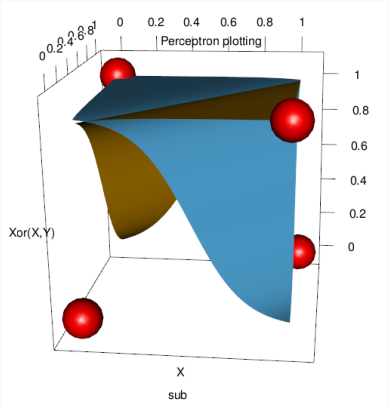
XOR Trained Network Example (2/4)

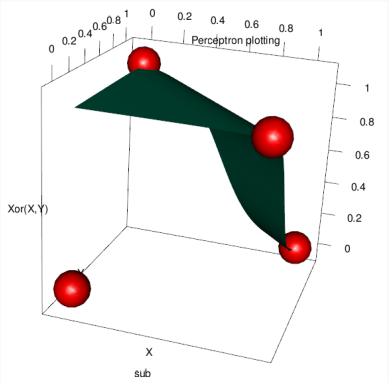




- In the input layer (*left plot*):
  - the first perceptron p_1 is mapping $(1,0) \rightarrow \approx 1$ and $(0,1) \rightarrow \approx 0$.
  - the second perceptron p_2 is mapping $(1,0) \rightarrow \approx 0$ and $(0,1) \rightarrow \approx 1$.
  - both perceptrons map either $(1,1)$ and $(0,0)$ to 1.

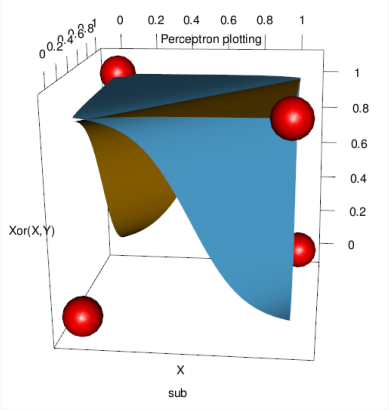
XOR Trained Network Example (3/4)

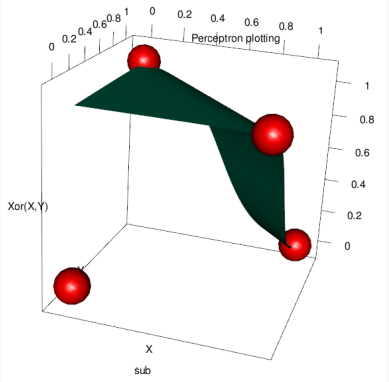




- This implies that:
  - $(1,0) \xrightarrow[p_2]{p_1} (1,0)$
  - $(0,1) \xrightarrow[p_2]{p_1} (0,1)$
  - $(1,1), (0,0) \xrightarrow[p_2]{p_1} (1,1)$

XOR Trained Network Example (4/4)





- In the output layer (*right plot*):
  - $(1,0) \rightarrow \approx 1$
  - $(0,1) \rightarrow \approx 1$
  - $(1,1) \rightarrow \approx 0$

Multilayer Neural Networks for Multiclass Classification

- We can easily extend this simple solution for a multiclass classification scenario if we use as many output neurons as $|\mathcal{Y}|$.
  - As a result, $\tilde{h}\colon [0,1]^n \rightarrow [0,1]^{|\mathcal{Y}|}$, where n is the data dimensionality.
  - The decision function becomes $h(x) = \arg\max_{1 \leq i \in \mathbb{N} \leq |\mathcal{Y}|} \tilde{h}(x)_i$, where $N(x)_i$ is the i-th component of the vector returned by the output layer.
  - The predicted desirability of the class is given by $\tilde{h}(x)_{h(x)}$, that is the greatest neuron output.
- You can easily generalise the case the former definitions if it is possible that a classifier returns multiple possible classes (see the provided code)!

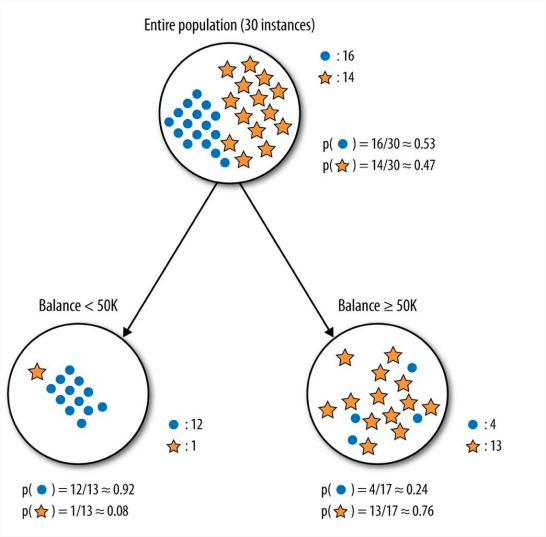
Decision Trees

Binary Decision Tree

Given a table with schema $R(X_1, \ldots, X_n, Y)$ over a dataset E with classification attribute Y with $dom(Y) = \mathcal{Y}$ where $|\mathcal{Y}| = 2$, the binary decision tree t over such dataset is defined as follows:

- Any node $n \in t$ represents a subset $\kappa(n) \subseteq E$ of the overall data.
- A *leaf node* $\ell \in t$ also provides the final classification outcome $y \in Y$ and an associated precision value.
- Any *non-leaf node* n represents a decision predicate P: left (`t.left`) and right (`t.right`) branches will respectively contain the nodes satisfying and not satisfying it.
  - $\kappa(\texttt{t.left}) \cup \kappa(\texttt{t.right}) = \kappa(\texttt{t})$
  - $\kappa(\texttt{t.left}) \cap \kappa(\texttt{t.right}) = \emptyset$

Impurity Function (1/5): intuition



A good split decision allows to generate two sub-trees minimising the impurity (e.g., one class is more represented than the others)

In order to create a binary decision tree, we need an **impurity function** ϕ determining which is the best split for each node n.

Impurity Function (2/5) : preliminary definitions

Let us denote $\mathbf{c}_2(y, S)$ the set of all the objects in S that belong to the class $y \in \mathcal{Y}$ and $p(y, S) = \frac{|\mathbf{c}_2(y,S)|}{|S|}$ a probability function over S.

- The prior probability that an object beongs to a given class y is $\mathbb{P}(y) = p(y, \kappa(n))$.
- The conditional probability that an object is in the class y under the condition that the P test has (not) passed is $\mathbb{P}(y|P) = \frac{p(y,\kappa(n.\texttt{left}))}{|\kappa(n.\texttt{left})|/|E|}$ ($\mathbb{P}(y|\neg P) = \frac{p(y,\kappa(n.\texttt{right}))}{|\kappa(n.\texttt{right})|/|E|}$).

Impurity Function (3/5)

Given a node n which is clear from the context, an **impurity function** ϕ is defined over the tuple $\langle \mathbb{P}(y) \rangle_{y \in \mathcal{Y}}$. Such function shall satisfy the following properties:

1. $\arg\max_k \phi(k) = \langle 1/|\mathcal{Y}| \rangle_{y \in \mathcal{Y}}$
2. $\arg\min_k \phi(k) = t \quad \exists! y \in \mathcal{Y} t_y = 1 \wedge \forall y' \in \mathcal{Y} \backslash \{y\}. t_{y'} = 0$

The **impurity measure** $\iota(n) = \phi(\langle \mathbb{P}(y \mid n) \rangle_{y \in \mathcal{Y}})$ defines an **decrease of impurity** (a.k.a. information gain) for a binary decision tree as:

$$\Delta\iota(n) = \iota(n) - \mathbb{P}(P)\iota(n.\texttt{left}) - \mathbb{P}(\neg P)\iota(n.\texttt{right})$$

Impurity Function (4/5): Examples

Given a node n that is branched into two subtrees via a decision P (and $\neg P$), then we can use the following definitions:

- Shannon Entropy:

$$\iota_{\text{Entropy}}(n) := -\mathbb{P}(P)\log_2(\mathbb{P}(P)) - \mathbb{P}(\neg P)\log_2(\mathbb{P}(\neg P))$$

- Gini Index:

$$\iota_{\text{Gini}}(n) := 1 - \mathbb{P}^2(P) - \mathbb{P}^2(\neg P)$$

Impurity Function (5/5): Examples

Is there any specific preference between these two functions?

- We can prove that these functions only disagree in 2% of all cases: this proves why most empirical results cannot decide which metric performs better.
- Therefore, the Gini index might be slightly more efficient as it doesn't involve to compute log functions.

NP-Complete Training (Hyafil & Rivest, 1976)

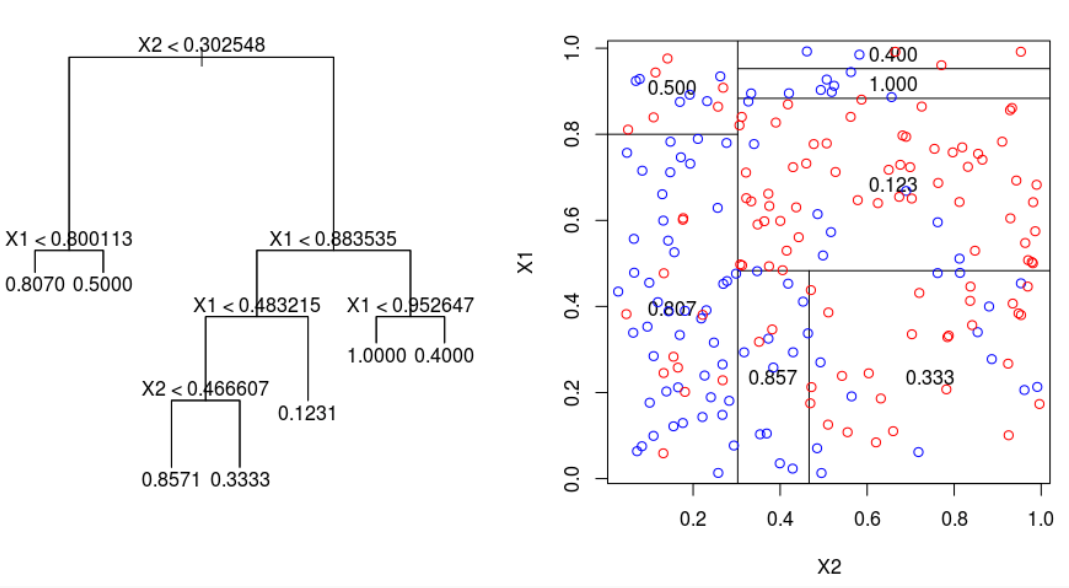
- It was showed that training optimal binary decision trees is an NP-Hard problem.
  - We do not know which is the best number of regions M to split the data
  - Find M regions such that $\min_{\{R_m\}_{1 \leq m \leq M}} \frac{1}{|R_m|} \sum_{i \in R_m} (y_i - \mu_{R_m})^2$
  - All the possible combinations of grouping $|E|$ elements in M groups are $\binom{|E|}{M} = \frac{|N|!}{M!(|N|-M)!}$, thus reducing to analyse a factorial number of combinations for $M \ll |E|$.
- Nevertheless, such exact solution might lead to a overfitted model.

C4.5 – A Training Greedy Algorithm (Quinlan, 1993)

Given the current subtree having a node n:

- If the current node contains only elements of one of the two classes, then halt and create a leaf node.
- If I reached the maximum depth, then set the precision value as the $\mathbb{P}(P)$
- Otherwise, perform the following steps:
  - For each variable X_i, find an optimal cutoff point s_i defining a decision predicate $P(x) \stackrel{def}{=} x.X_i \leq s_i$ or $P(x) \stackrel{def}{=} x.X_i = s_i$. Pick s_i increasing the *decrease of impurity*.
  - Pick the variable X_i having the bigger *decrease of impurity* alongside its decision predicate P.
  - Split the data in two according to P, increase the the depth by 1 and continue in each sub-tree.

Example



Selecting the blue as the candidate $y = 1$ class (e.g., height and weight)

Loss and Decision function

Algorithm 1 Decision function by traversing the decision tree.

1: **function** REGRESSION(t, x, $\pi = \emptyset$)
2: **if** t.isLEAF() **then return** $\langle$t.precision, $\pi\rangle$
3: **else**
4: $n \leftarrow$t.$P(x)$? t.left :t.right;
5: **return** REGRESSION($n, x, \pi \cup \{\texttt{t} \rightarrow n\}$)
6: **end if**
7: **end function**

- While traversing the tree, we build up a decision motivation by remembering the traversed path.
- When we reach a leaf node, then the precision value associated to each node determines the expected distance of x from the target class 1.

Decision Trees for Multiclass classification

- Binary decision trees can be easily generalised to allow multiple possible branches and to solve a multiclass classification problem.
  - See the generalised definition of the *impurity function*.
- In the attached source code, we're going to use $|\mathcal{Y}|$ binary regression trees for the one-versus-all version of the problem.

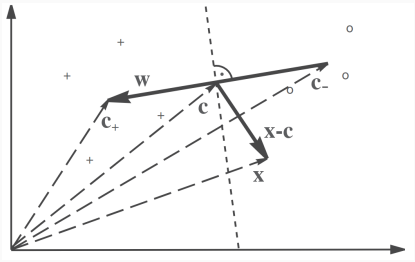
ν-Support Vector Machines with Kernel Trick

Cartesian equation of the (hyper-)plane

Given a (hyper-)plane π, a point $\mathbf{c} \in \pi$, and a non-null vector $\mathbf{w} \perp \pi$, we have that $\mathbf{x} \in \pi$ if and only if $\mathbf{x} - \mathbf{c} \perp \mathbf{w}$.

Given that two perpendicular vectors have a zero dot product, then we can express the *Carteisan equation of the (hyper-)plane* π containing $\mathbf{c}$ and having a norm $\mathbf{w}$ as follows: $\mathbf{w}(\mathbf{x} - \mathbf{c}) = 0$

Support Vector Machines: Intuition



Given a set E of examples with $\mathcal{Y} = \{-1, 1\}$:

- Let be $\mathbf{c}_+$ (and $\mathbf{c}_-$) be the centroid of class 1 (and -1).
- The midpoint $\mathbf{c} := \mathbf{c}_+ - \mathbf{c}_-/2$ ideally separates the two classes.
- The decision plane π should pass through $\mathbf{c}$ and should be perpendicular to the vector $\mathbf{w} = \mathbf{c}_+ - \mathbf{c}_-$ connecting the two classes.

A binary approximation of the decision function for $\mathbf{x}$ becomes:

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}(\mathbf{x} - \mathbf{c}))$$

Expressing the SVM decision function into a similarity function

- We can consider the dot product as a non-normalized similarity function between two vectors.
- Let be n_1 (n_{-1}) the number of the samples in E belonging to class 1 (-1).
- Using the former definitions, we can rewrite the former decision function for a constant b as follows:

$$h(\mathbf{x}) = \text{sign}\left(\frac{1}{n_1}\sum_{\substack{(\mathbf{x}_i, y_i)\in E \\ y=1}}(\mathbf{x}\cdot\mathbf{x}_i) - \frac{1}{n_{-1}}\sum_{\substack{(\mathbf{x}_i, y_i)\in E \\ y=-1}}(\mathbf{x}\cdot\mathbf{x}_i) + b\right)$$

Kernel trick

The *kernel trick* is useful in the current occasions:

- the problem cannot be separated by hyperplanes
- the data is not necessarily represented as numbers and vectors

In these situations, we can first transform the instances $\mathbf{x}$ using a embedding Φ to a specific feature space and then apply the inner product. Thus, a generic kernel can be defined as the following *similarity* metric: $K(x, x') \approx \Phi(x)\Phi(x')$

We can now rewrite the decision function as follows:

$$h(\mathbf{x}) = \text{sign}\left(\frac{1}{n_{\mathbf{1}}} \sum_{\substack{(\mathbf{x}_i, y_i) \in E \\ y=1}} K(\mathbf{x},\ \mathbf{x}_i) - \frac{1}{n_{-\mathbf{1}}} \sum_{\substack{(\mathbf{x}_i, y_i) \in E \\ y=-1}} K(\mathbf{x},\ \mathbf{x}_i) + b\right)$$

Some Kernel Functions for structured data

- Please observe that the composition of kernels is also a kernel: $(K_2 \circ K_1)(\mathbf{x}, \mathbf{x}') = \Phi_2(\Phi_1(\mathbf{x}))\Phi_2(\Phi_1(\mathbf{x}'))$ (this will be useful in the next slides!)
- We can also define kernels for structured and categorical data. In particular, we can also adopt kernel functions for n-ary trees:

$$K_{tree}^{K_\ell, K_\iota}(t,s) = \begin{cases} K_\ell(t,s) & t,s \text{ leaves} \\ K_\iota(t.\text{node}, s.\text{node}) + \sum_j K_{tree}^{K_\ell, K_\iota}(t.\texttt{child}(j), s.\texttt{child}(j)) & t.\text{node} = s.\text{node} \wedge |t.\texttt{child}| = |s.\texttt{child}| \\ K_\iota(t.\text{node}, s.\text{node}) & \text{oth.} \end{cases}$$

From Kernel Tricks to Feature Maps (Wills et al., 2009)

If we have categorical or structured data E, we can generate approximations of a feature map Φ as follows:

- Given that any kernel K provides a similarity function, we can provide a distance d_K which is a proper metric as follows:

$$d_K(x, y) = \sqrt{K(x, x) + K(y, y) - 2 \cdot K(x, y)}$$

- Generate a dissimilarity matrix $D = d_K(e, e')_{(e,e') \in E^2}$
- Use a Multi-Dimensional Scaling technique to learn a matrix $X \in \mathbb{R}^{d \times |E|}$ from $D \in \mathbb{R}^{|E| \times |E|}$.
- $\Phi(i) := X_i$

Training

- We can train a SVM classifier by using the following optimization problem:

$$\underset{\mathbf{w},b}{\text{minimize}}\ \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad \forall(\mathbf{x}_i, y_i) \in E.\ y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1$$

- In order to get better results, we might normalize the samples in E by calculating the component-wise average and standard deviation, thus obtaining the vectors $\vec{\mu}_E$ and $\vec{\sigma}_E$. So, we can define a feature map $\Phi_{\vec{\mu}_E, \vec{\sigma}_E}(\mathbf{x}) := (\mathbf{x} - \vec{\mu}_E) \odot \frac{1}{\vec{\sigma}_E}$.

ν-Support Vector Machines (1/2)

- Due to the noisy nature of real data, there might be potentially no hyperplane separation.
- Therefore, we might introduce $|E|$ *slack variables* $\xi = \{ \xi_i \}_{1 \leq i \leq |E|}$, so that linear separable problem expressed in the former slide can be relaxed as $y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1 - \xi_i$.

ν-Support Vector Machines (2/2)

After some rewriting (e.g., Lagrangian Dual), we can express the final decision function obtained after our training as:

$$h(\mathbf{x}) = \text{sign}\left(\sum_{(\mathbf{x}_i, y_i) \in E} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) + b \right)$$

where $\sum_{1 \leq i \leq |E|} \alpha_i \geq \nu$, and $\alpha = \{ \alpha_i \}_{1 \leq i \leq |E|}$ and b are trained using an optimization problem.

- `dlib` returns $\tilde{\alpha}_i := \alpha_i y_i$

Radial Basis Function Kernel (Cotter, Keshet & Srebro, 2011)

One popular kernel is the Radial Basis Function (a.k.a. Gaussian) Kernel:

$$K_G(x, y) = e^{-\gamma\|x-y\|^2}$$

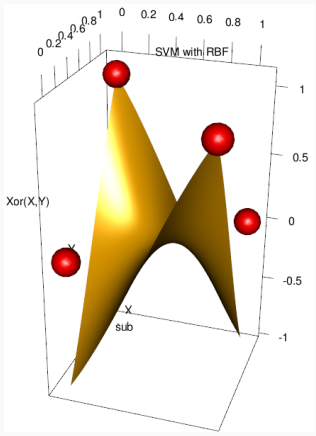
This model has the following advantages:

- It is easy to calibrate (as we only need to set up γ).
- The model is translation invariant: $K_G(x + c, y + c) = K_G(x, y)$ with c constant.
- SVM training methods may take days over millions of examples with dimensions less than 100 for custom SVM kernels. It is possible to efficiently approximate such kernel for k given dimensions by Taylor expansion:

$$\Phi_{G,k}(\mathbf{x}) = e^{-\gamma\|\mathbf{x}\|^2} \frac{1}{(2\gamma)^{k/2}\sqrt{k!}} \prod_{0 \leq i \leq k} \mathbf{x}_j$$

- Models some non-hyperplane separable problems.

XOR Trained ν-SVM with RBF Kernel Example



ν-SVM separation using a RBF Kernel

- Four our training data, we have that $\Phi_{\vec{\mu}_E, \vec{\sigma}_E}(\mathbf{x}) = (\mathbf{x} - (0.5, 0.5)) \odot (\sqrt{3}, \sqrt{3})$
- The best γ parameter for training the RBF kernel is $\gamma = 10 \cdot 0.5^6$
- For the sample $E = \{((0,0), -1), ((1,0), 1), ((1,1), 1), ((0,1), -1)\}$ we

Conclusions

Conclusions (1/3)

- Multilayer Neural Networks
  - Work only on vectorial and numerical representation of data.
  - It is hard to predict the right network configuration for a given random dataset.
  - The model interpretation is defined by the network configuration and the weights W, θ.
  - Main problem: Small data perturbations (noise) might result in completely unexpected results
  - We still need to use MiniMax search problems! (Adversarial training)

Conclusions (2/3)

- Decision Trees
  - They trivially support both numerical and categorical data.
  - They adapt to any possible data distribution.
  - The model interpretation is the returned decision tree itself.
  - Main problem: they can easily go into overfitting because of both their greedy approach and the size of the tree depth.

Conclusions (3/3)

- ν-Support Vector Machines with Kernel Trick
  - They support categorical data via specific kernel tricks.
  - ν smoothing combined with RBF kernels allows to learn specific non "multi"-linearly separable problems over noisy data.
  - Main problem: ν-SVM with RBF kernels provide bad data generalizations.
  - Regularisation parameters can be tuned to avoid overfitting.