UCL

Advanced Machine Learning in Finance

Paolo Barucca
Tomaso Aste

LECTURE 3

- K-Nearest Neighbors (KNN)
- Random Forest
- Support Vector Machines (SVM)
- Perceptron
- Multilayer Perceptron
- Case study: Credit Crisis Prediction

Supervised Learning

Let us consider a set of samples, a paired list of features x and labels y

x_{11}					x_{1N}
x_{P1}					x_{PN}

y_1
y_P

K-Nearest Neighbors

K-Nearest Neighbors is a supervised algorithm which - quite literally - takes a test sample and looks at its **k nearest neighbors** (e.g. defined with a Euclidean norm in the space of features) in the training set.

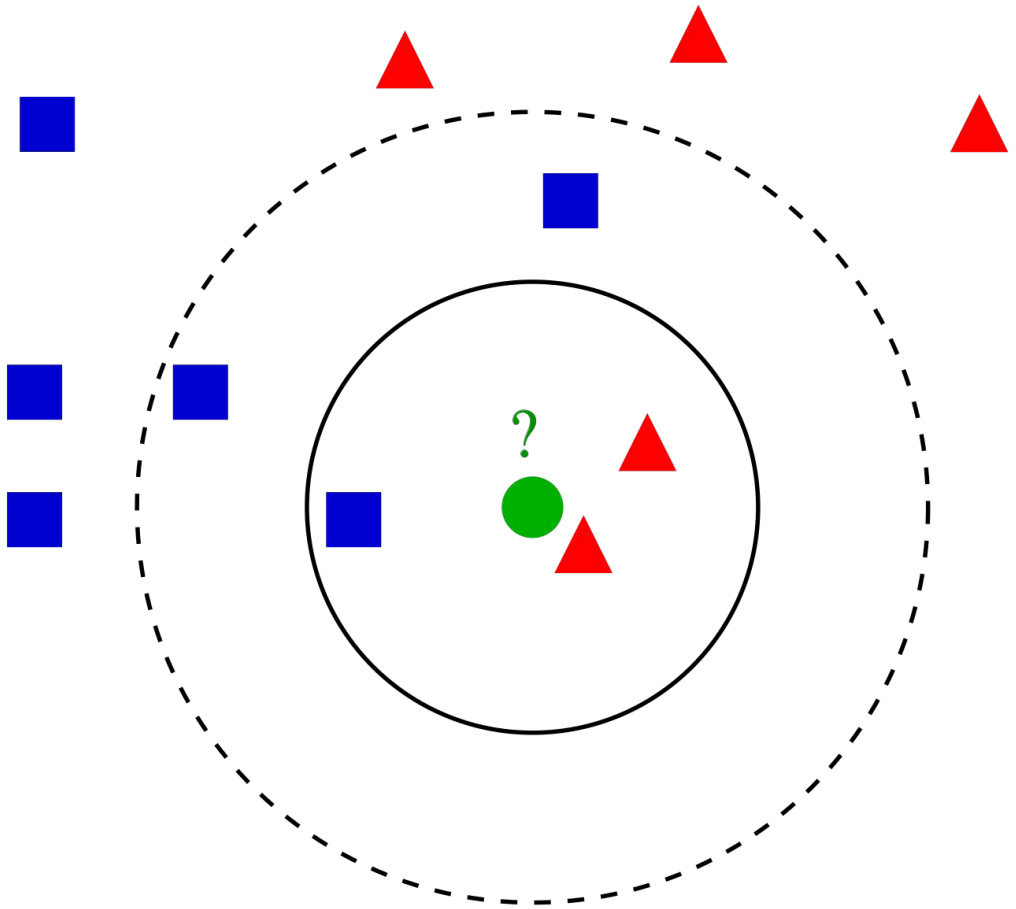
K-Nearest Neighbors

K-NN

- for classification, it assigns the test sample to the class (y label) of the majority of its neighbors
- for regression, it takes the average value of the y values of its neighbors

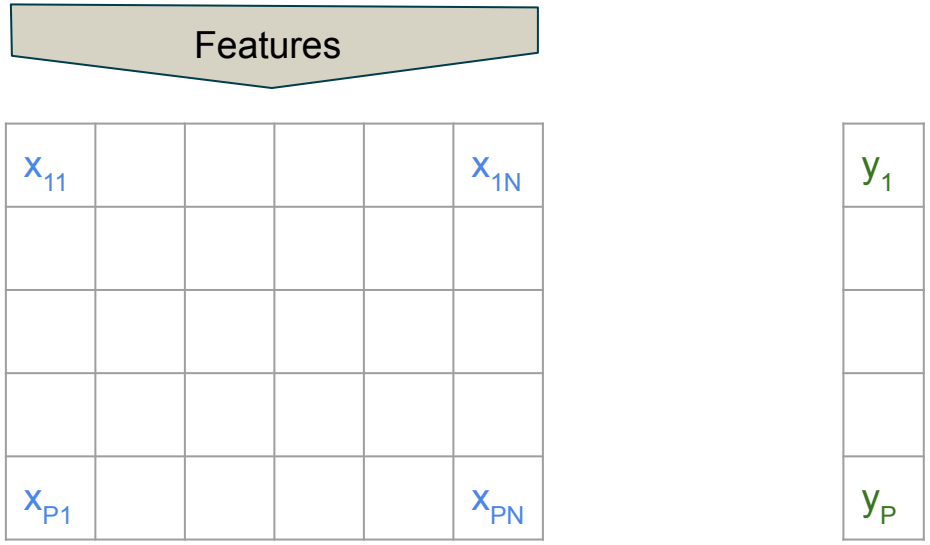
k is an hyper-parameters which can be optimised, e.g. using a validation set.

K-Nearest Neighbors



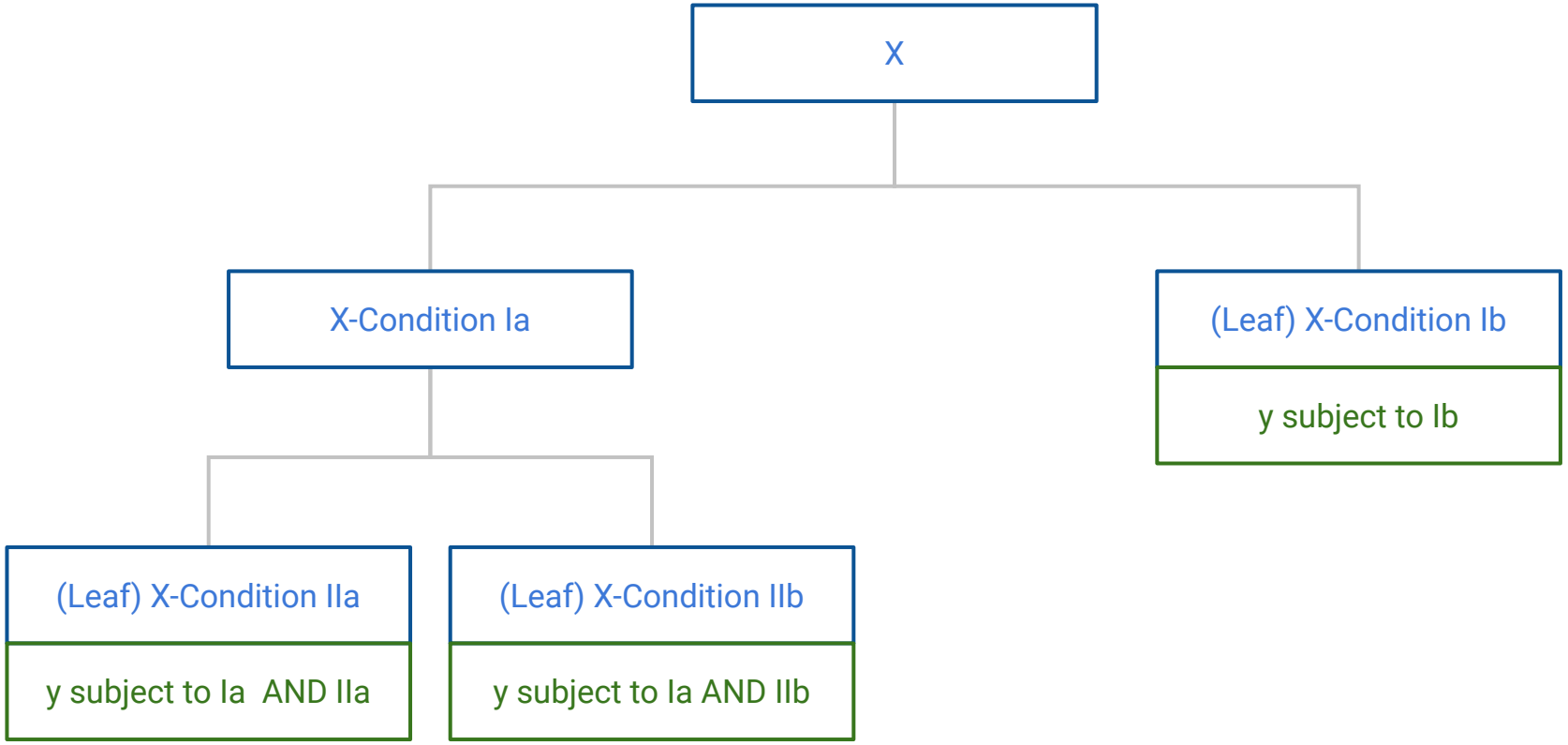
Random (Decision) Forest

A forest of random decision trees, i.e. ensemble learning with decision trees.

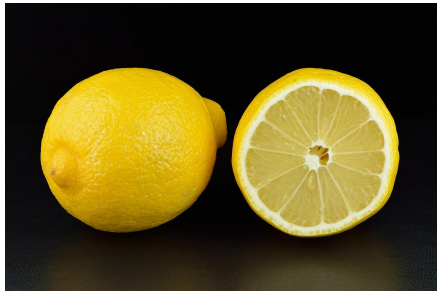


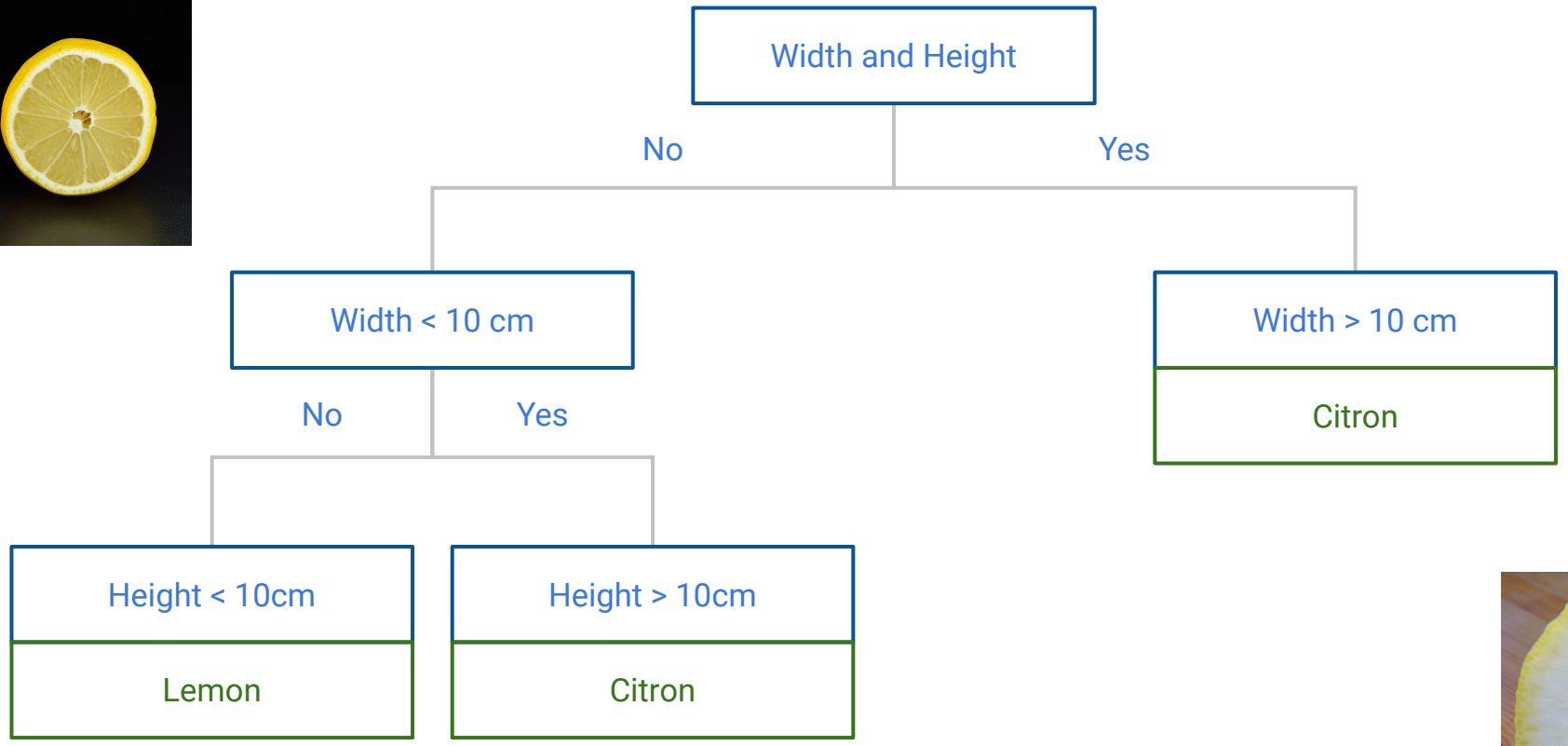
NOT a forest of random decisions!

A decision tree

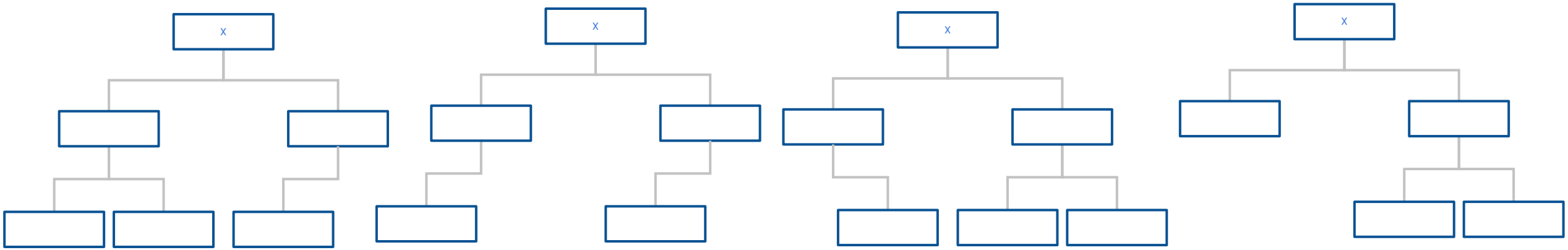


A decision tree (example)





A forest of decision trees



A forest of decision trees

Each tree is NOT completely random,

so exactly what is random about the tree?

A forest of decision trees

Each tree is NOT completely random,

so exactly what is random about the tree?

The subset of features that is used for learning each decision tree is random, i.e. each tree considers a random subset of features (a random subset of columns in the feature matrix) so that the decision problem for each tree takes little time.

A forest of decision trees

This is because one needs to compare a few features at a time and each decision (branch) is made based on computing the information gain given by each feature in the dataset, e.g. the conditional entropies.

$$
\begin{aligned}
H(Y|X) &= \sum_{x \in X} p(x) H(Y|X = x) \\
&= -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(y|x)
\end{aligned}
$$

A random subset of features

A random forest algorithm

Algorithm 1 Random Forest

Precondition: A training set $S := (x_1, y_1), \ldots, (x_n, y_n)$, features F, and number of trees in forest B.

```
function RandomForest(S, F)
    H ← ∅
    for i ∈ 1, ..., B do
        S^(i) ← A bootstrap sample from S
        h_i ← RandomizedTreeLearn(S^(i), F)
        H ← H ∪ {h_i}
    end for
    return H
end function
function RandomizedTreeLearn(S, F)
    At each node:
        f ← very small subset of F
        Split on best feature in f
    return The learned tree
end function
```

http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/ensembles/RandomForests.pdf

Ensemble learning (a committee of trees)

Each random tree provides a prediction for the label given the (subset of) features and we can take the ensemble prediction, that is:

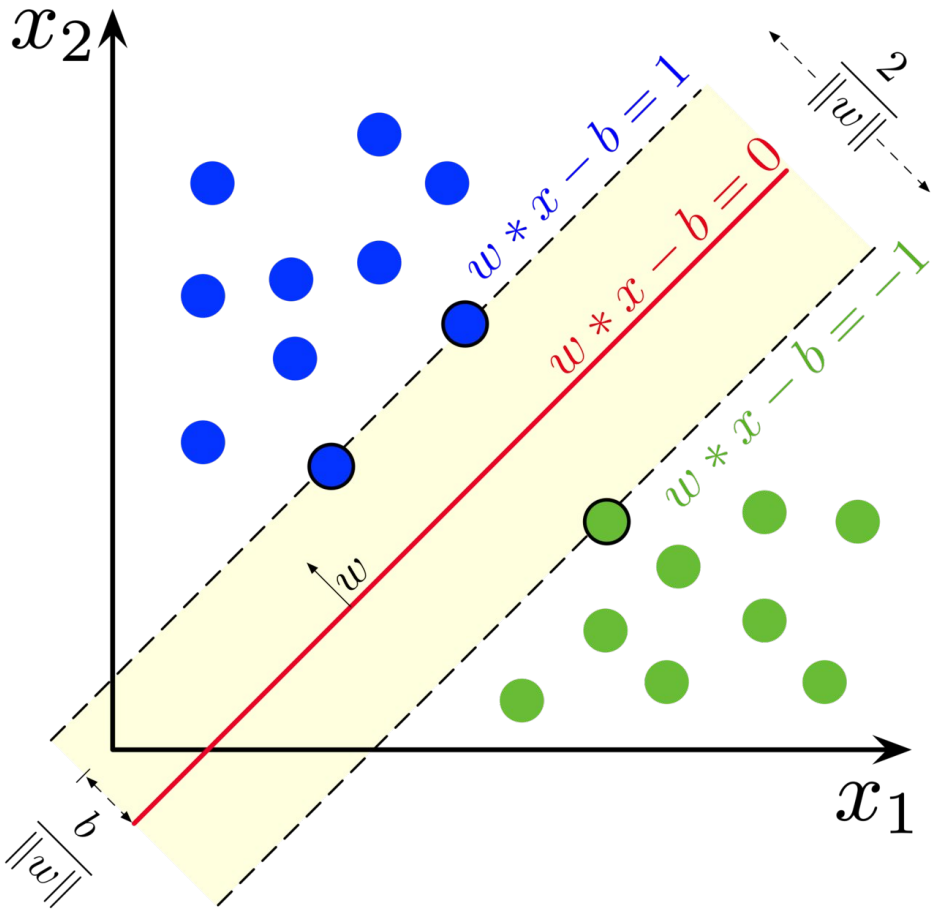
- the value of y which is predicted by the majority of trees for discrete labels (classification)
- y averaged over the trees for continuous labels (regression)

Support Vector Machines

Support vector machines identify a decision boundary,
by introducing a **support vector**.

The support vector defines an hyperplane in the space of features or a
in a transformed space of features.

Support Vector Machines



Support Vector Machines

The original linear version is the support vector classifier, which solves the following problem for the support vector w,

- Hard-margin. Minimise norm(w) subject to $y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1$ for $i = 1, \ldots, n$.

- Soft-margin. Minimise $$\lambda\|\mathbf{w}\|^2 + \left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i - b)\right)\right]$$

Support Vector Machines

Support vector machines also consider a kernel function which allows a non-linear transformation of the original space of features.

In order to introduce the non-linear transformation the optimisation problem is

(1) mapped into its Lagrangian dual and then

(2) the feature space vector is transformed through a function $\varphi(\mathbf{x}_i)$ the kernel

Lagrangian Dual

$$\text{maximize } f(c_1 \dots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (\mathbf{x}_i^T \mathbf{x}_j) y_j c_j,$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$

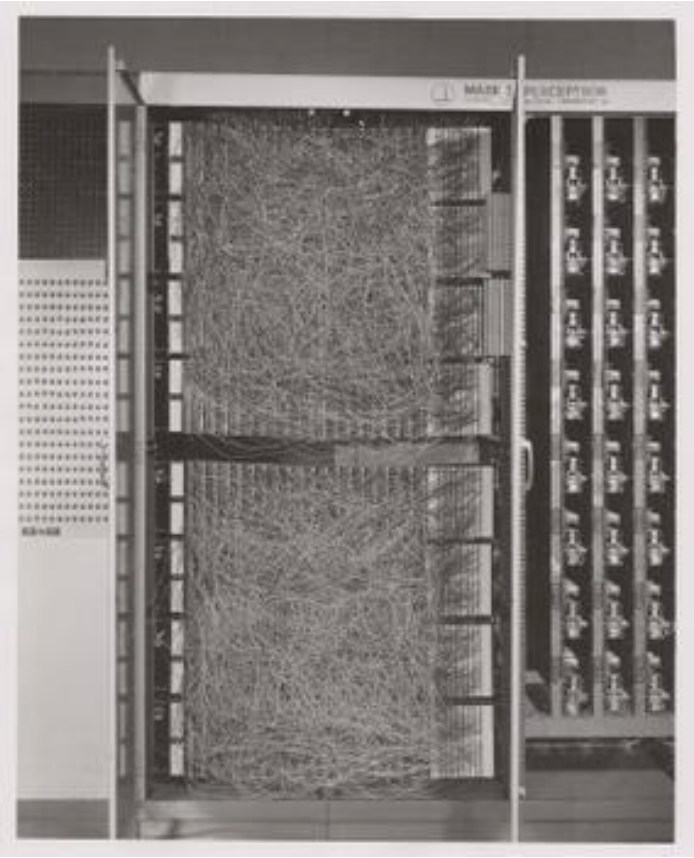
Lagrangian Dual with Kernel

$$\text{maximize } f(c_1 \dots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (\varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)) y_j c_j$$

$$= \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i k(\mathbf{x}_i, \mathbf{x}_j) y_j c_j$$

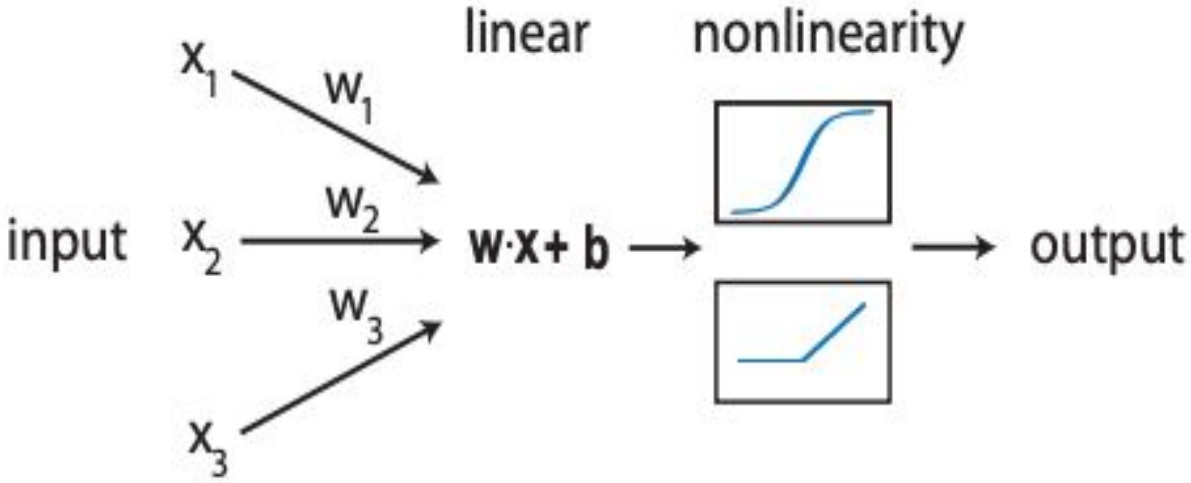
$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$

Perceptron



First perceptron Mark I machine for image recognition, it was connected to a 20x20 pixel camera, based on the adaptive algorithm by Frank Rosenblatt 1958.

Perceptron



$$\hat{y} = \Theta(\sum_{j=1}^{N} w_{1j} x_j + b_1)$$

Figures from A high-bias, low-variance introduction to Machine Learning for physicists. arXiv:1803.08823. Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre Day, Clint Richardson, Charles Fisher, David Schwab.

Perceptron

The perceptron problem

- I have a training set, $training = \{\vec{x}_k, y_k\}_{k=1}^{P}$
- Can I find a set of weights, w, and biases, b, such that when I feed the training input to the perceptron its output coincides with the training output?
- Or equivalently, can we send the cost function, E, to zero?

$$E(w,b) = \sum_{k=1}^{P} (\hat{y}_k(w,b) - y_k)^2$$

Perceptron

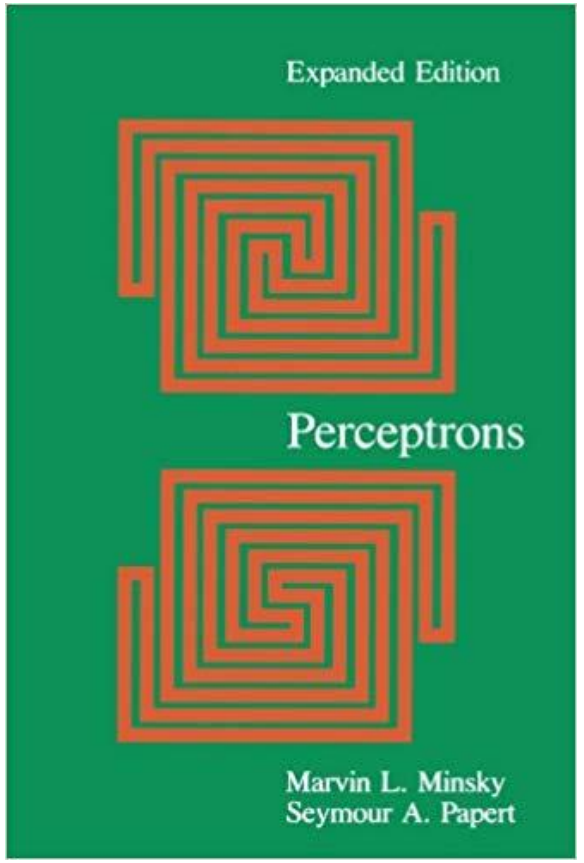
The perceptron problem

- I have a training set, $training = \{\vec{x}_k, y_k\}_{k=1}^{P}$
- Can I find a set of weights, w, and biases, b, such that when I feed the training input to the perceptron its output coincides with the training output?
- Yes, if P not too large with respect to N there exist values w* and b* such E(w*,b*)=0

$$E(w,b) = \sum_{k=1}^{P} (\hat{y}_k(w,b) - y_k)^2$$

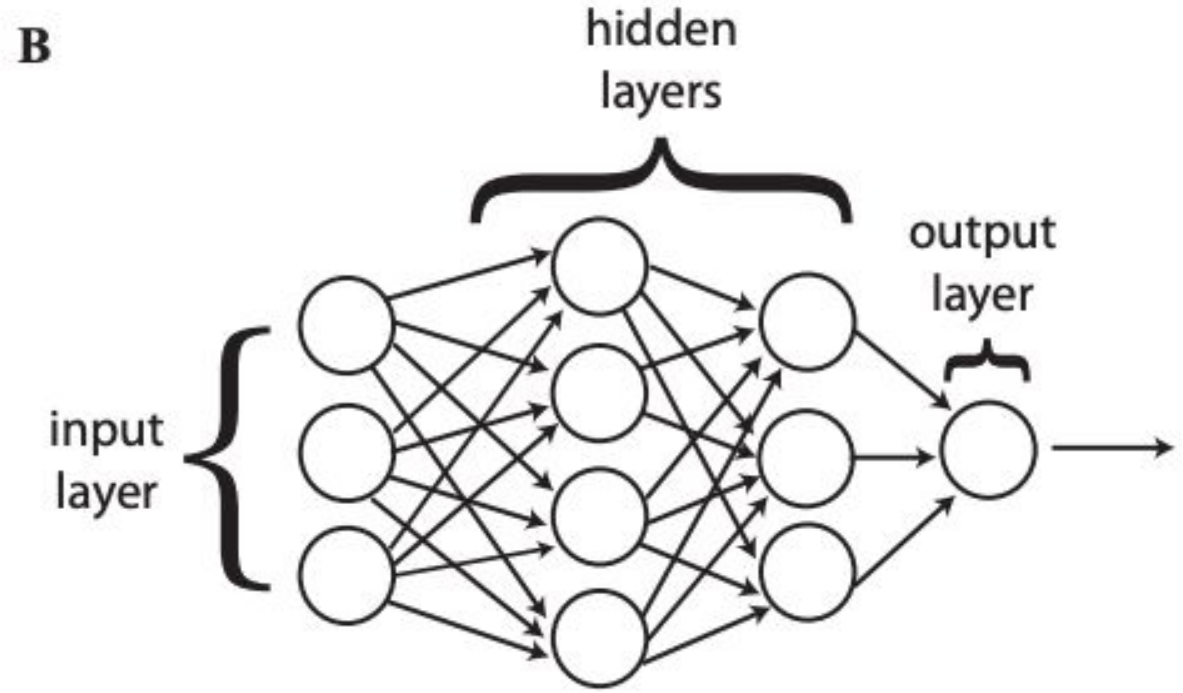
Perceptron

- Long known and studied in computer science and statistical physics literature
- Thought incapable to solve specific tasks in feasible simulations
- Minsky and Papert proved that the perceptron '*cannot compute parity under the condition of conjunctive localness and showed that the size required for a perceptron to compute connectivity grew impractically large'*



Minsky, Marvin; Papert, Seymour (1988). *Perceptrons: An Introduction to Computational Geometry*. MIT Press.

Multilayer Perceptron



$$\hat{y} = f_L\left(\sum_{j=1}^{N_{L-1}} w_{1j}^{(L)} a_j^{(L-1)}(x) + b_1^{(L)}\right)$$

$$z_i^{(l)} = (w^{(l)T} a^{(l-1)} + b^{(l)})_i \qquad a_i^{(l)}(x) = f_l\left(\sum_{j=1}^{N_{l-1}} w_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)}\right) = f_l(z_i^{(l)})$$

Figures from A high-bias, low-variance introduction to Machine Learning for physicists. arXiv:1803.08823. Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre Day, Clint Richardson, Charles Fisher, David Schwab.

Credit Crisis Prediction

Let us consider the paper by Daniel Fricke

Financial Crisis Prediction: A Model Comparison,

Daniel Fricke
Deutsche Bundesbank; University College London; London School of Economics & Political Science (LSE) - Systemic Risk Centre