Deep learning

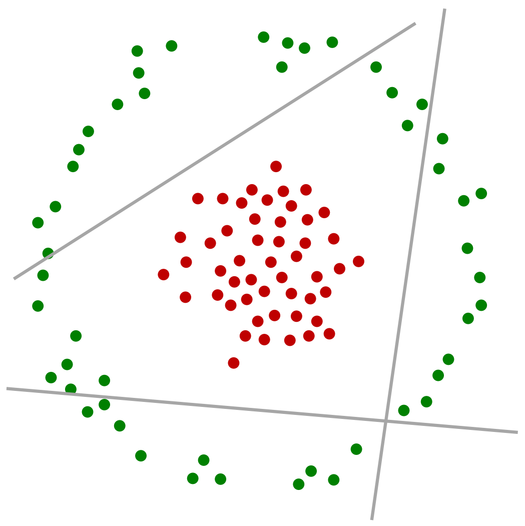
3.3. Linear separability and feature design

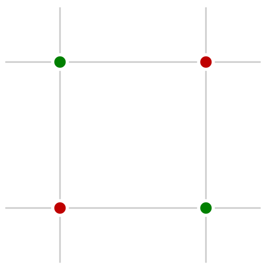
François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ DE GENÈVE

The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.





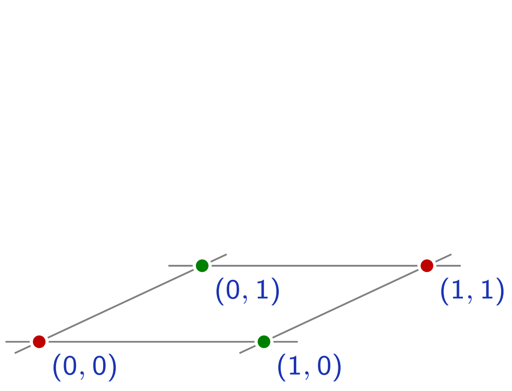
“xor”

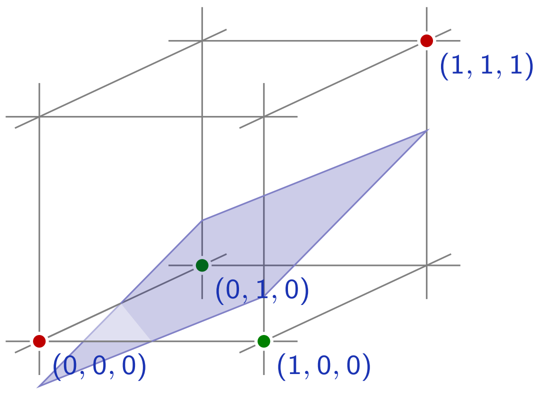
Notes

On the left image, it is clear that it does not exist an hyperplane (i.e. a line) which separates the two populations.
Another example even more vexing it the “xor” example (right image). These four data points are not linearly separable.
As we saw in lecture 2.2. “Over and under fitting”, the capacity of a set of predictors corresponds to its ability to model complex mappings, and linear models have a low capacity.

The xor example can be solved by pre-processing the data to make the two populations linearly separable.

$$\Phi : (x_u, x_v) \mapsto (x_u, x_v, x_u x_v).$$

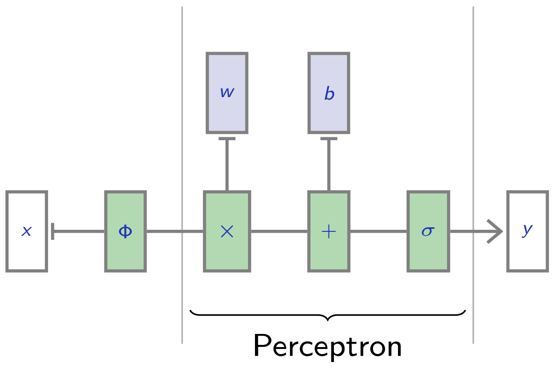




Notes

We can use an *ad-hoc* formula to pre-process the data. This formula is not trained: it is designed from prior knowledge about the problem. This pre-processing is a function of the input vector which aims at creating a new vector which will be used as input by the linear predictor.
In the "xor" problem, the four input points are $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$ and are mapped by Φ to $(0,0,0)$, $(0,1,0)$, $(1,0,0)$, and $(1,1,1)$, which are linearly separable in $\mathbb{R}^3$.
So we can now model the "xor" with:

$$f(x) = \sigma\left(w \cdot \Phi(x) + b\right).$$



Notes

In practice, we pre-process all the training points and then use the perceptron algorithm on it.

This is similar to the polynomial regression. If we have

$$\Phi : x \mapsto (1, x, x^2, \dots, x^D)$$

and

$$\alpha = (\alpha_0, \dots, \alpha_D)$$

then

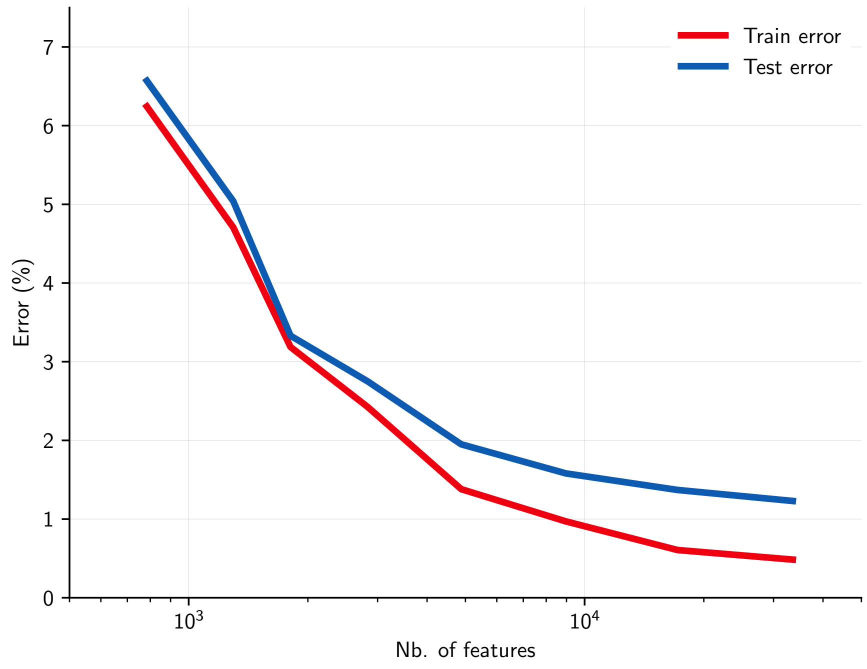
$$\sum_{d=0}^{D} \alpha_d x^d = \alpha \cdot \Phi(x).$$

By increasing D, we can approximate any continuous real function on a compact space (Stone-Weierstrass theorem).

It means that we can make the capacity as high as we want.

We can apply the same to a more realistic binary classification problem: MNIST's "8" vs. the other classes with a perceptron.

The original $28 \times 28 = 784$ features are supplemented with the products of pairs of features taken at random.



Notes

We illustrate the use of a pre-processing which increases the dimension of the data points on MNIST. Class 1 consists of the images of "8" while class 0 consists of all the other digits.
The pre-processing consists of:

- Taking the original 784 pixels of the image. Here, the pixels can be viewed as features.
- Extending these initial features with pairwise product of pixels selected at random among the 784. The pairs are randomly selected prior to learning, and of course remain the same for both training and test.

$$\underbrace{\boxed{x_1}\boxed{x_2}\boxed{\cdots}\boxed{x_{784}}}_{\text{Original pixels}}\underbrace{\boxed{x_4 x_9}\boxed{x_{11} x_8}\boxed{\cdots}\boxed{x_{99} x_{41}}}_{\text{Products of pixels}}$$

The plot shows the error rate as a function of the number of features, which is also the dimension of the input space after pre-processing.

- The curves starts at 784 on the x axis, which corresponds to no added product of pixels. So, we only have the dimension of the original space, the number of pixels in a digit.
- We can see that the more we extend the space with product of pixels, the lower the training and test errors: Adding more features made the problem more separable.
- The gap between train and test errors increases, showing that overfitting gets worse.

Remember the bias-variance tradeoff we saw in 2.3. "Bias-variance dilemma"

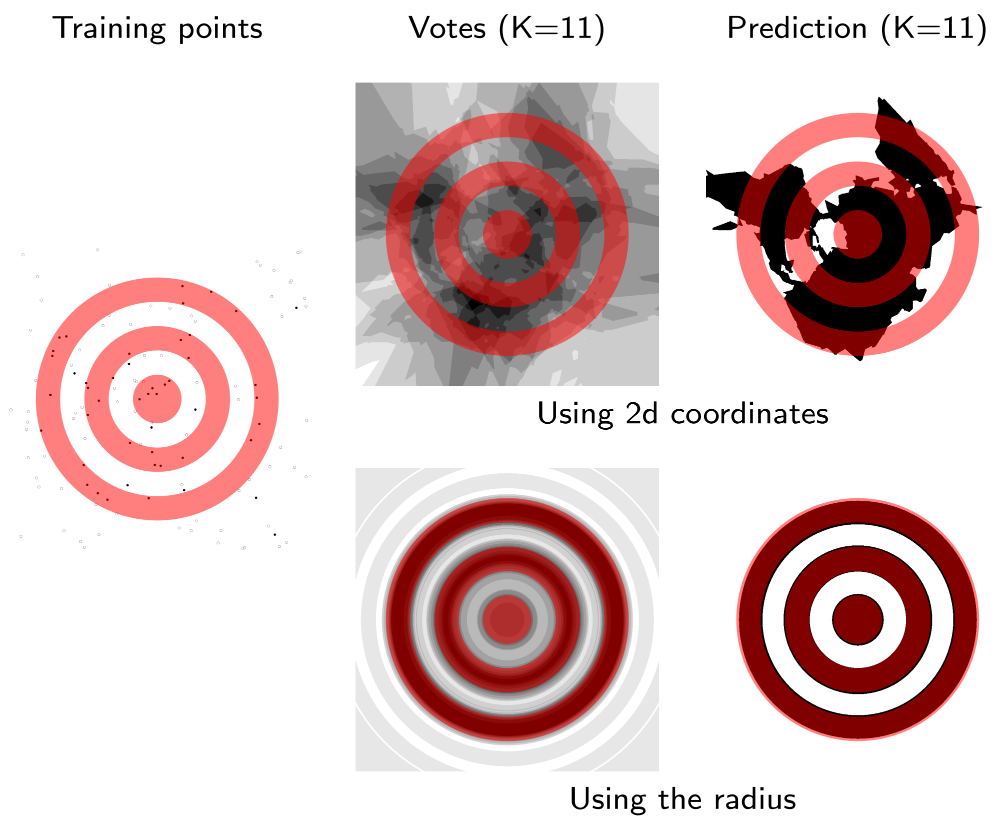
$$\mathbb{E}((Y - y)^2) = \underbrace{(\mathbb{E}(Y) - y)^2}_{\text{Bias}} + \underbrace{\mathbb{V}(Y)}_{\text{Variance}} .$$

The right class of models reduces the bias more and increases the variance less.

Beside increasing capacity to reduce the bias, "feature design" may also be a way of reducing capacity without hurting the bias, or with improving it.

In particular, good features should be invariant to perturbations of the signal known to keep the value to predict unchanged.

We can illustrate the use of features with k-NN on a task with radial symmetry. Using the radius instead of 2d coordinates allows to cope with label noise.



Notes

We illustrate the design of feature with a simple synthetic binary problem:

- The true class of samples is depicted by the red rings: class 1 is inside the red areas, and class 0 outside.
- We generate labeled training points with noise in the labels: some black points (class 1) are actually outside the red areas, and some white points (class 0) are inside.

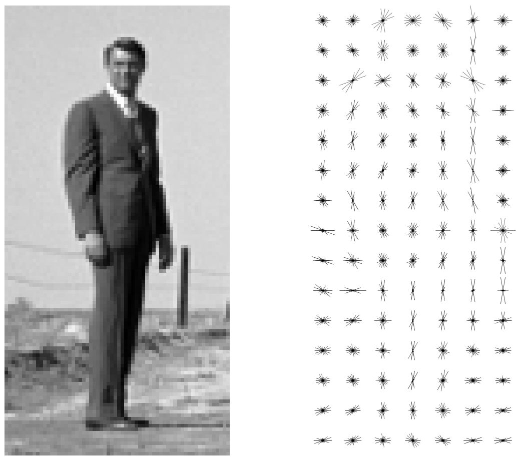
When we apply the K-nearest neighbors algorithm on the original data points the plane, the number of votes is very noisy, which results in a prediction which does not reflect the true structure of the data.
If we have the knowledge that the label of a point is invariant by rotation around a center point, we can pre-process the data to give to the predictor only the distance r to the center point:

$$\begin{aligned}\Phi: \quad & \mathbb{R}^2 \to \mathbb{R} \\ & (x, y) \mapsto \sqrt{(x - x_c)^2 + (y - y_c)^2}\end{aligned}$$

The prediction is now much better, although we have reduce the dimension from $\mathbb{R}^2$ to $\mathbb{R}$.

A classical example is the "Histogram of Oriented Gradient" descriptors (HOG), initially designed for person detection.

Roughly: divide the image in 8×8 blocks, compute in each the distribution of edge orientations over 9 bins.



Dalal and Triggs (2005) combined them with a SVM, and Dollár et al. (2009) extended them with other modalities into the "channel features".

Notes

Prior to deep learning techniques, common pre-processing steps consisted in computing several modalities such as histograms of gradient and to concatenate them in channels, before feeding them to standard algorithms.
In this example here, the task is to predict if an image contains a pedestrian. The gray-level of a pixel is poorly informative, as cloths, skin, or background can be dark or light.
However the orientation of edges has a very specific structure when a person is present. So a linear SVM, which is a linear predictor, could achieve very good performance when fed with the edge orientation statistics of the HOG descriptor.

Many methods (perceptron, SVM, k-means, PCA, etc.) only require to compute $\kappa(x, x') = \Phi(x) \cdot \Phi(x')$ for any (x, x').

So one needs to specify κ alone, and may keep Φ undefined.

This is the **kernel trick**, which we will not talk about in this course.

Training a model composed of manually engineered features and a parametric model such as logistic regression is now referred to as **“shallow learning”**.

The signal goes through a single processing trained from data.

References

N. Dalal and B. Triggs. **Histograms of oriented gradients for human detection**. In Conference on Computer Vision and Pattern Recognition (CVPR), pages 886–893, 2005.

P. Dollár, Z. Tu, P. Perona, and S. Belongie. **Integral channel features**. In British Machine Vision Conference, pages 91.1–91.11, 2009.