Deep learning

3.3. Linear separability and feature design

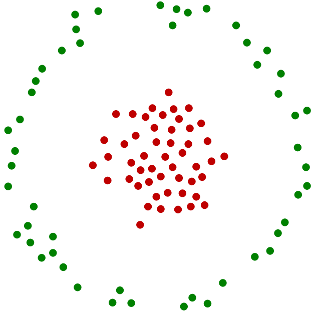
François Fleuret

https://fleuret.org/dlc/

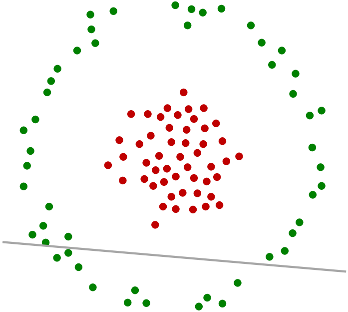
UNIVERSITÉ DE GENÈVE

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

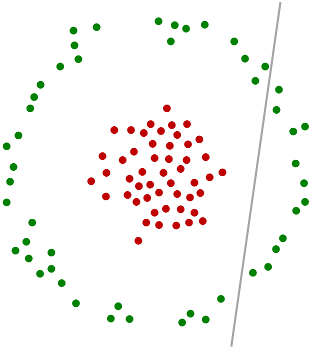
The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.



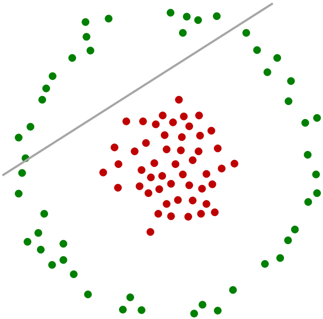
The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.



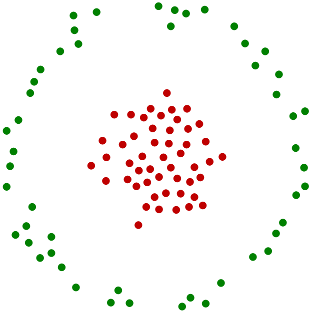
The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.

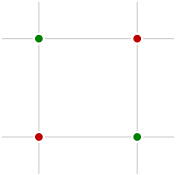


The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.



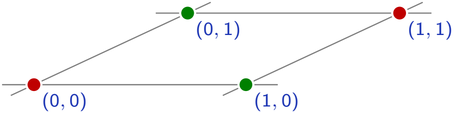
The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.





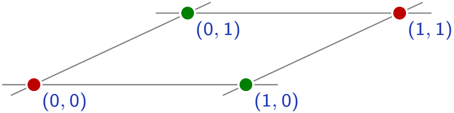
“xor”

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



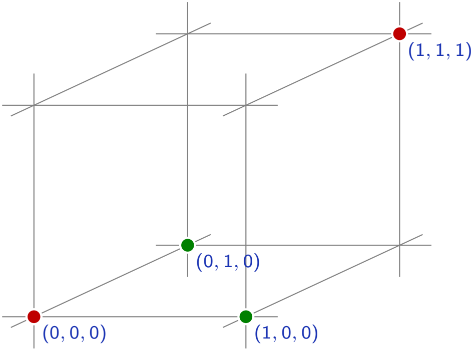
The xor example can be solved by pre-processing the data to make the two populations linearly separable.

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



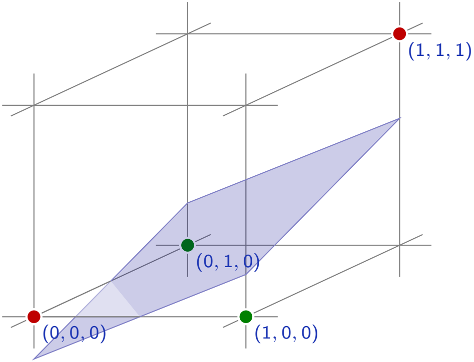
The xor example can be solved by pre-processing the data to make the two populations linearly separable.

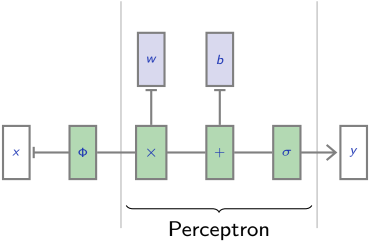
$$\Phi : (x_u, x_v) \mapsto (x_u, x_v, x_u x_v).$$



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

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




w
b
x
Φ
×
+
σ
y
Perceptron

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

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

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).

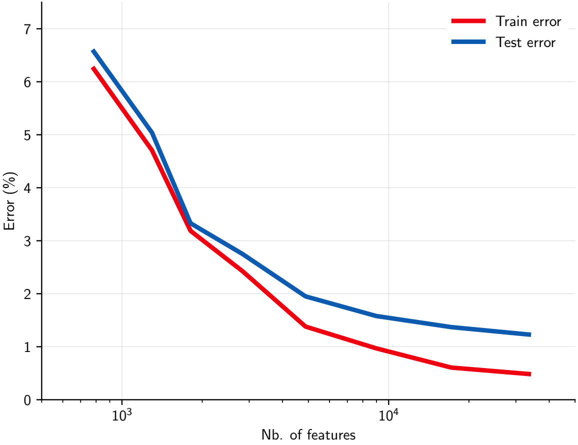
It means that we can make the capacity as high as we want.

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.

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.



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.

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.

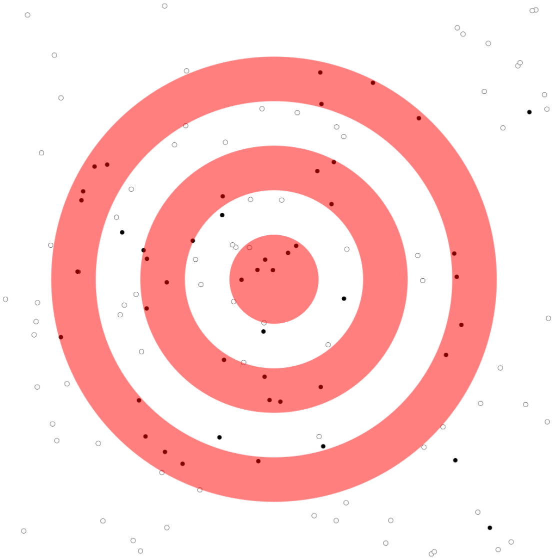
Remember the bias-variance tradeoff we saw in 2.3. “Bias-variance dilemma”

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

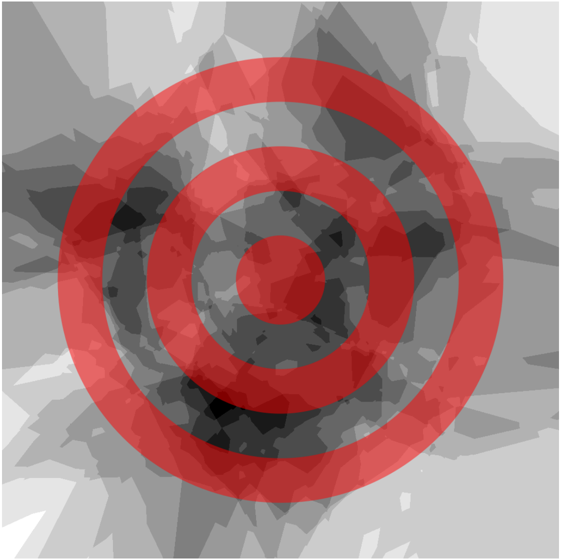
The right class of models reduces the bias more and increases the variance less.

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.

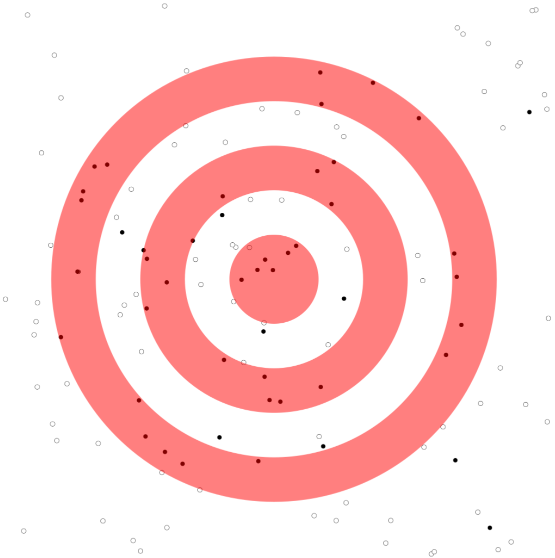


Training points



Votes (K=11)

Prediction (K=11)



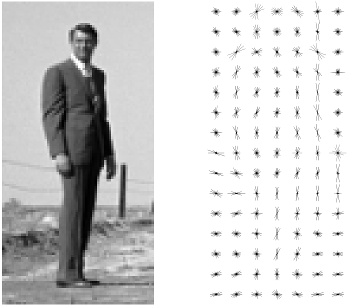
Training points

Votes, radial feature (K=11)

Prediction, radial feature (K=11)

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".

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.

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.

The end

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.