Part I – Kernelized Perceptron

TODO: Kernelized perceptron

The **Kernel Trick**

- Inspired by this, we go **beyond** *feature maps*
- In most algorithms, things only depend on the dot product between input vectors:

  → Let's replace everywhere:

$$\vec{x} \cdot \vec{x}' \longrightarrow K(\vec{x}, \vec{x}')$$

- The Kernel K needs to be *positive-semi definite.* (respect some conditions, so that is behaves like a scalar product does)
- See the *Mercer condition*

Part II – Kernels

Kernels

Feature maps are the simplest kind of Kernels:

$$x_i, x_{i'} \mapsto K(x_i, x_{i'}) = \phi(x_i)\phi(x_{i'}) \qquad \text{(factored)}$$

- More advanced: **translation-invariant** Kernels

$$x_i, x_{i'} \mapsto K(x_i, x_{i'}) = \Phi(x_i - x_{i'})$$

e.g. Radial Basis Function (RBF) kernels:

$$K(x, x') = \exp\Big(-\gamma||x - x'||^2\Big)$$

→ no closed-form $\phi(x_i)\phi(x_{i'})$ expression

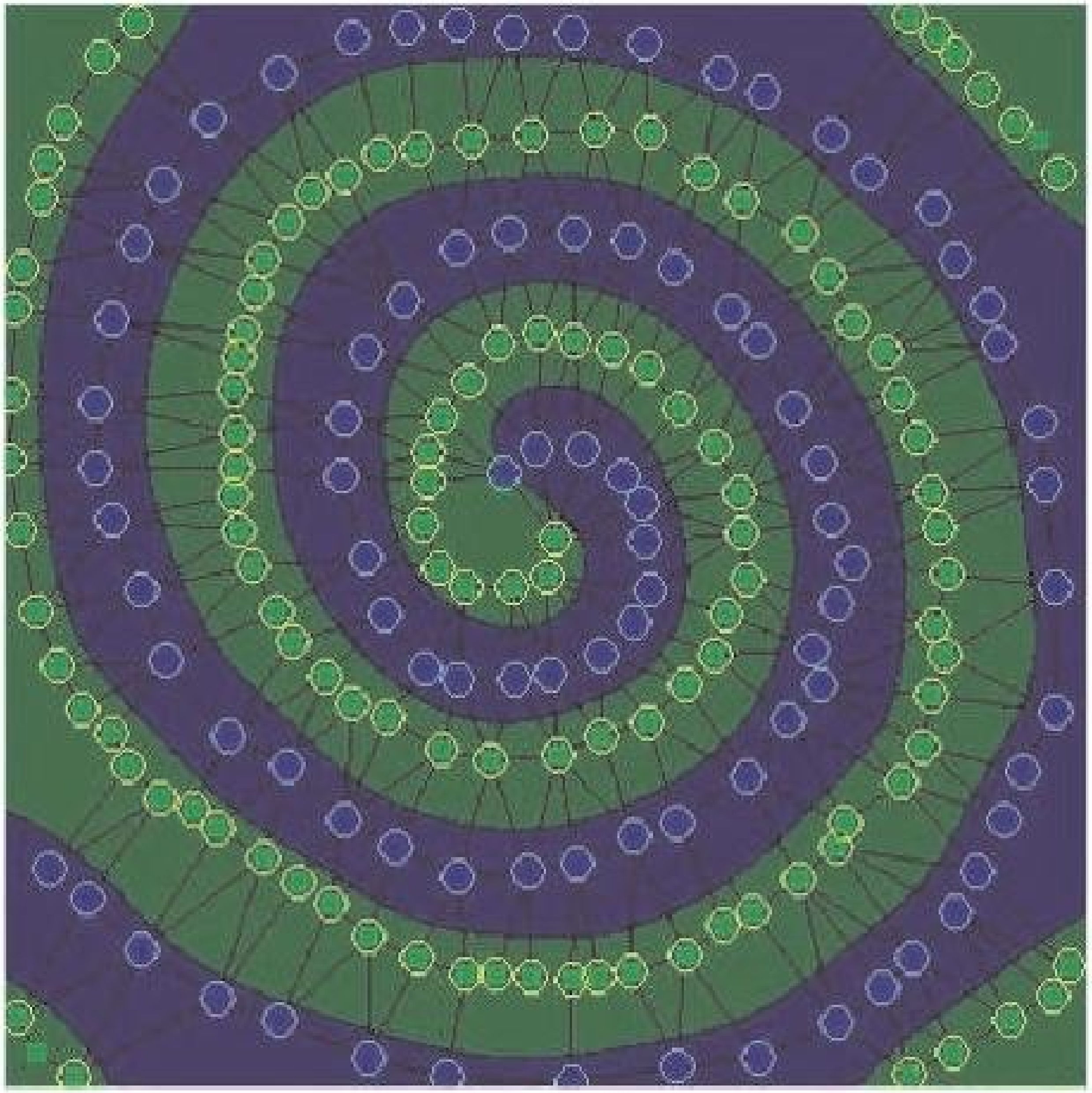
- But **approximate Kernels** exist (we can come back to an ***approximation*** of with *feature maps*) :

$$\phi(x) = exp(-\gamma x^2)\left[1, \sqrt{\frac{2\gamma}{1!}}x, \sqrt{\frac{(2\gamma)^2}{2!}}x^2, \sqrt{\frac{(2\gamma)^3}{3!}}x^3, \ldots\right]$$

- More advanced: (**fully general Kernel**)

$$x_i, x_{i'} \mapsto K(x_i, x_{i'}) = \Phi(x_i, x_{i'})$$

RBF: quite expressive !



Feature maps vs. Kernels

- Feature maps: *Just before* the ML algo,
  we transform the data (**pre-procesing**)

  $$\vec{x}^{(n)} = (x_1, x_2, x_3)^{(n)} \longrightarrow \phi(\vec{x}^{(n)})$$

  The **coefficients** of the map are **fixed**.
  Then we use these **new data features** as input.
  There are **more parameters in the model**.

- Kernels: we transform *on-the-fly,* compute time
  grows with *train set size*, and *feature number*

  → ***True Kernel ≠ pre-processing***

  There are **more parameters in the model**.
  (can become very very expensive actually)

References:

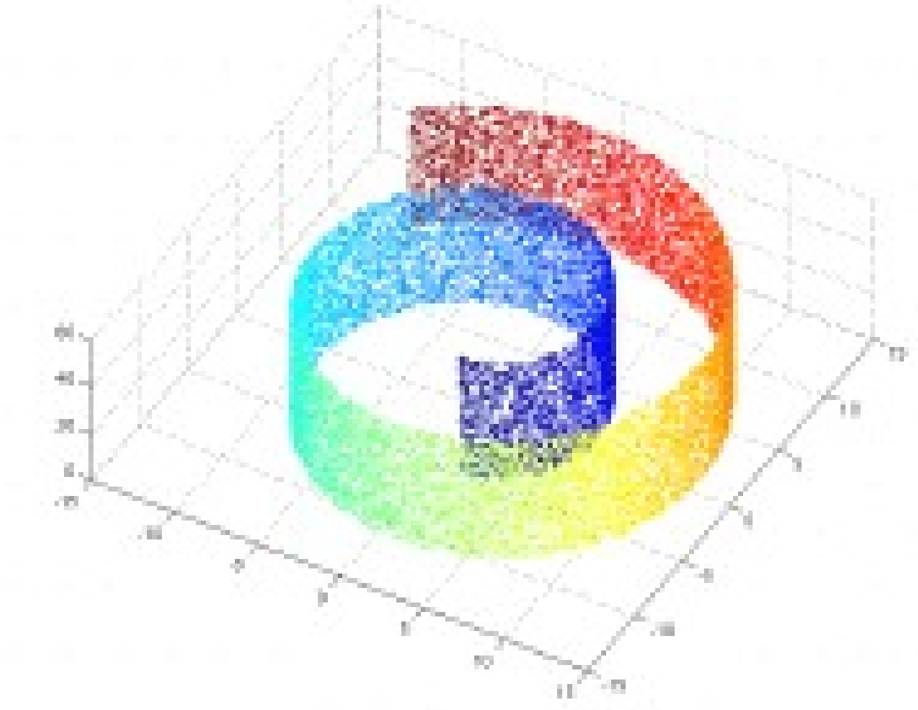
- a very clear explanation for ***feature maps****:*
https://scikit-learn.org/stable/modules/linear_model.html#polynomial-regression

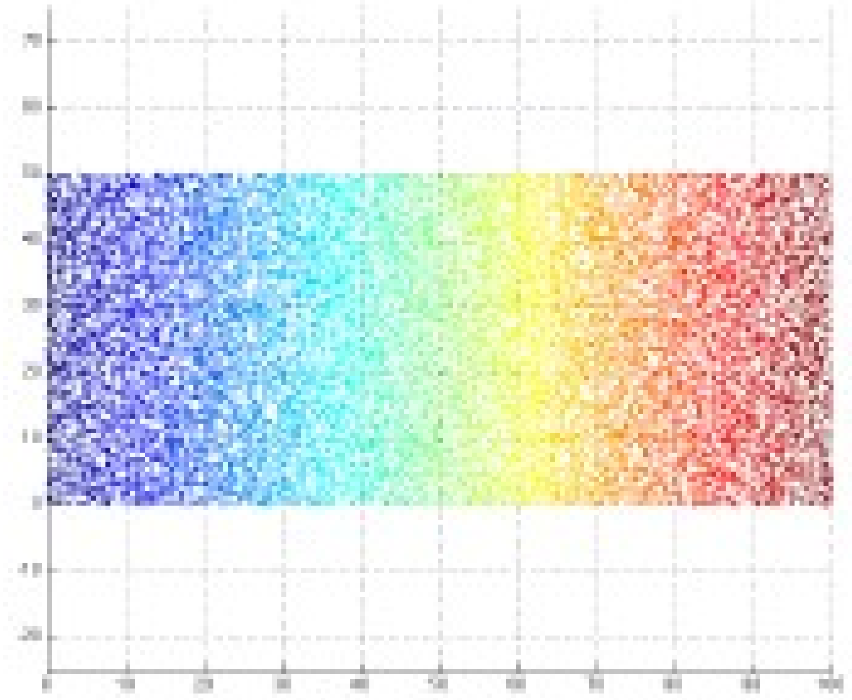
- *Bishop book*, page 291-294 (section 6.1)

- about ***approximate kernels***:
https://scikit-learn.org/stable/modules/kernel_approximation.html

Dimensional games

Example: the *swiss roll*





→ a good algorithm should be able to *unfold* it