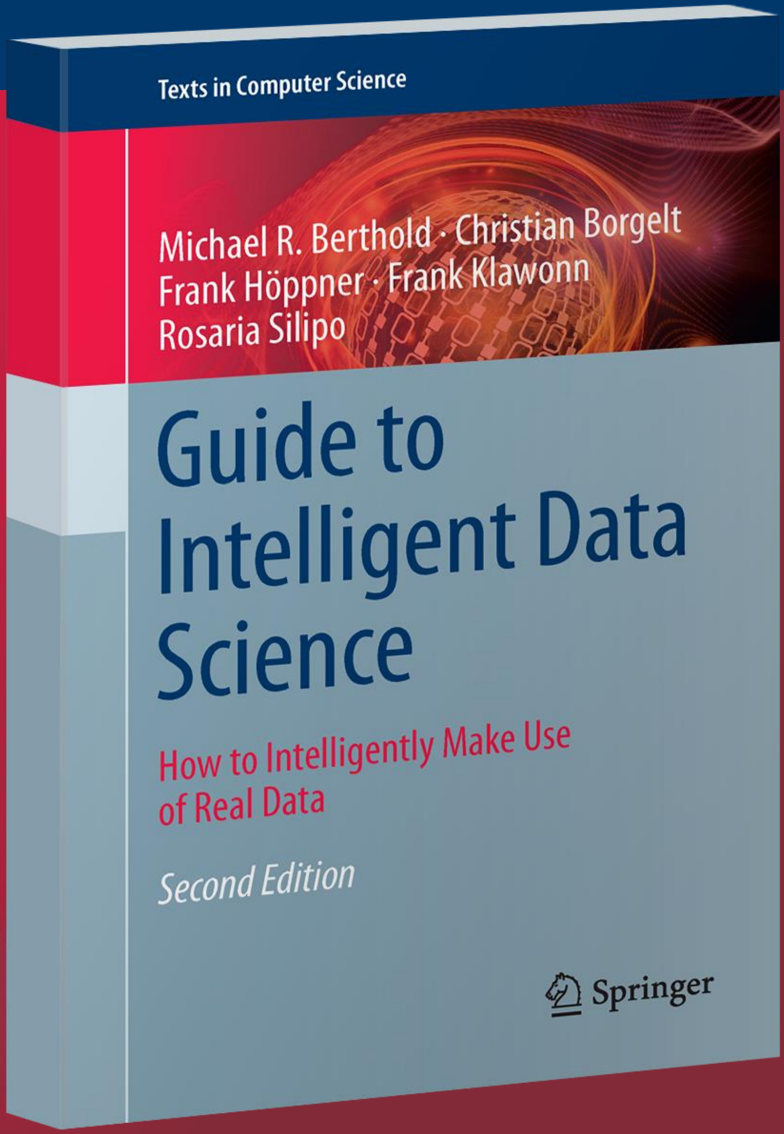
Support Vector Machines (SVM)



Summary of this lesson

"The key to artificial intelligence has always been the representation"
-Jeff Hawkins

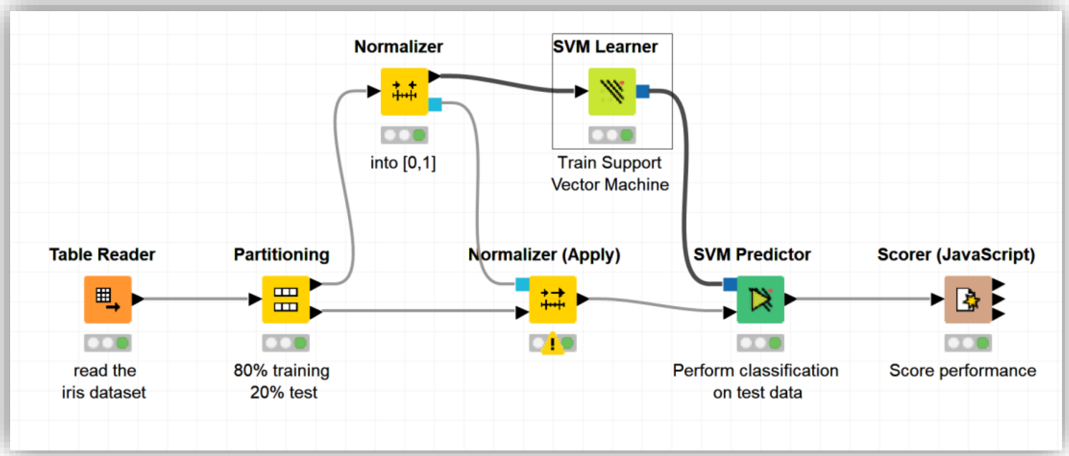
What are Support Vector Machines?

**This lesson refers to chapter 9 of the GIDS book*

Support Vector Machines (more generally – Kernel Machines)

- Motivation
- Linear Classifiers
  - Rosenblatt Learning Rule
- Kernel Methods and Support Vector Machines
  - Dual Representation
  - Maximal Margins
  - Kernels
- Margin of Error and Variations
  - Soft and Hard Margin Classifiers
  - Multi-Class SVM
  - Support Vector Regression

- Datasets used : iris dataset
- Example Workflows:
  - „SVM on iris dataset " https://kni.me/w/DTfbNITUngKQVF8v
    - Normalization
    - SVM



Motivation

- Main idea of Kernel Methods
  - Embed data into suitable vector space
  - Find linear classifier (or other linear pattern of interest) in new space

- Needed: a Mapping

$$\Phi\colon\ x \in X \ \rightarrow \ \Phi(x) \in F$$

- Key Assumptions:
  - Information about relative position is often all that is needed by learning methods
  - The inner products between points in the projected space can be computed in the original space using special functions (kernels).

Linear Classifiers

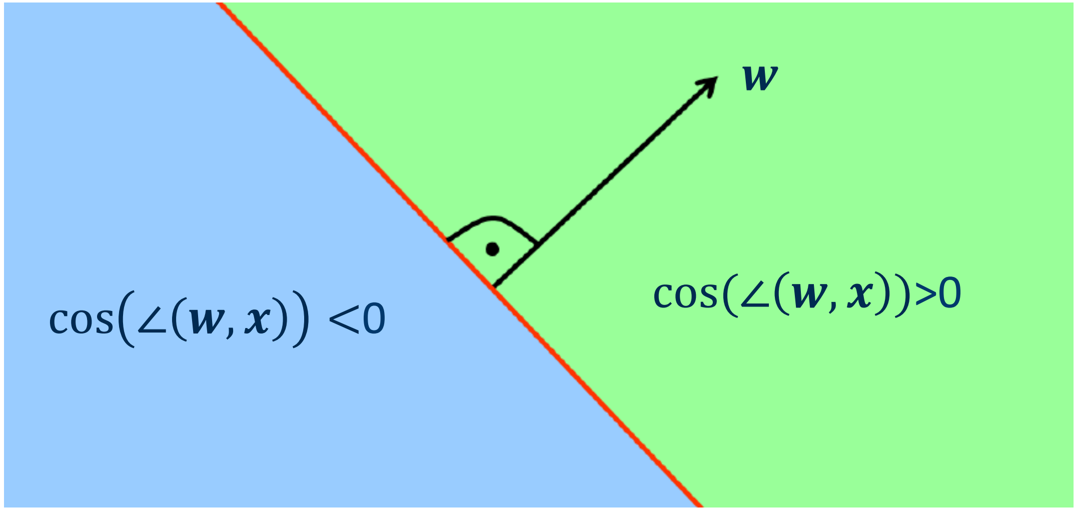
- Simple linear, binary classifier:

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b = \sum_{i=1}^{n} x_i w_i + b = b + \|\boldsymbol{w}\|\|\boldsymbol{x}\|\cos(\angle(\boldsymbol{w}, \boldsymbol{x}))$$

- Class A if $f(\boldsymbol{x})$ positive
- Class B if $f(\boldsymbol{x})$ negative

- e.g. $\mathrm{h}(\boldsymbol{x}) = sgn(f(\boldsymbol{x}))$ is the decision function

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = b + \|\boldsymbol{w}\|\|\boldsymbol{x}\|\cos(\angle(\boldsymbol{w},\boldsymbol{x}))$$



– Linear discriminants represent hyperplanes in feature space

Training a “Perceptron”

- Classification using a Perceptron
  - Represents a (hyper-) plane: $\sum_{i=1}^{n} w_i \cdot x_i = \theta$
  - Left of hyperplane: class 0
  - Right of hyperplane: class 1

- Training a Perceptron
  - Learn the “correct” weights to distinguish the two classes
  - Iterative adaption of weights w_i
  - Rotation of the hyperplane defined by $\boldsymbol{w}$ and θ in small direction of $\boldsymbol{x}$ if $\boldsymbol{x}$ is not yet on the correct side of the hyperplane.

Primal Perceptron

- Rosenblatt (1959) introduced a simple learning algorithm for linear discriminants ("perceptrons"):
- Given a linearly separable training set S

$w_0 \leftarrow \mathbf{0}; b_0 \leftarrow \mathbf{0}; k \leftarrow \mathbf{0}$

$\mathrm{R} \leftarrow \max_{1 \le j \le m} \|\boldsymbol{x}_j\|$

repeat

  for $j = 1$ **to** m

    if $y_j \cdot (\boldsymbol{w}_k^{\boldsymbol{T}} x_j + b) \le 0$ **then**

      $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k + y_j \boldsymbol{x}_j$

      $b_{k+1} \leftarrow b_k + y_j R^2$

      $k \leftarrow k + 1$

    end if

  end for

until no mistakes made within the *for* loop

return $(\boldsymbol{w}_k, b_k)$

Rosenblatt Algorithm

- Algorithm is
  - *On-line* (pattern by pattern approach)
  - *Mistake driven* (updates only in case of wrong classification)
- Algorithm converges guaranteed if a hyperplane exists which classifies all training data correctly (data is linearly separable)
- Learning rule:

$$\boldsymbol{IF}\ y_i \cdot \left(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x}_j + b\right) < 0 \quad \boldsymbol{THEN} \begin{cases} \boldsymbol{w}(t+1) = \boldsymbol{w}(t) + y_i \cdot \boldsymbol{x}_j \\ b(t+1) = b(t) + y_j \cdot R^2 \end{cases}$$

- One observation:
  - Weight vector (if initialized properly) is simply a weighted sum of input vectors (b is even more trivial).

Dual Representation (of discriminant function)

– Weight vector $\boldsymbol{w}$ is a weighted sum of input $\boldsymbol{x_j}$

$$\boldsymbol{w} = \sum_{j=1}^{n} \alpha_j \cdot y_j \cdot \boldsymbol{x}_j$$

Where α_j represents how much $\boldsymbol{x_j}$ contributes to $\boldsymbol{w}$

– Large α_j: $\boldsymbol{x_j}$ is difficult to classify – higher information content

– Small or zero α_j: $\boldsymbol{x_j}$ easy to classify – smaller information content

→ This representation with α_j's – known as ***dual representation***

– We can now represent the discriminant function as

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b = \left(\sum_{j=1}^{n} \alpha_j \cdot y_j \cdot \boldsymbol{x}_j^{\boldsymbol{T}}\boldsymbol{x}\right) + b$$

- Dual Representation of Learning Algorithm:
- Given a training set S

$\boldsymbol{\alpha} \leftarrow \mathbf{0}; b \leftarrow \mathbf{0}$
$\mathrm{R} \leftarrow \max_{1 \le i \le m} \|\boldsymbol{x}_i\|$
repeat
    for $i = 1$ **to** m
        if $y_j \cdot \left(\sum_{j=1}^{m} \alpha_j y_j \boldsymbol{x}_j^{\boldsymbol{T}} \boldsymbol{x}_i + b\right) \le 0$ **then**
            $\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i + 1$
            $b \leftarrow b + y_i R^2$
        end if
    end for
until no mistakes made within the *for* loop
return $(\boldsymbol{\alpha}, b)$

- Both α_j and b can be updated iteratively
- Learning Rule at iteration t:

$$\boldsymbol{IF} \quad y_j \cdot \left(\sum_{j=1}^{n} \alpha_j y_j \boldsymbol{x}_i^{\boldsymbol{T}} \boldsymbol{x}_j + b \right) < 0 \quad \boldsymbol{THEN} \begin{cases} \alpha_i(t+1) = \alpha_i + 1 \\ b(t+1) = b(t) + y_i \cdot R^2 \end{cases}$$

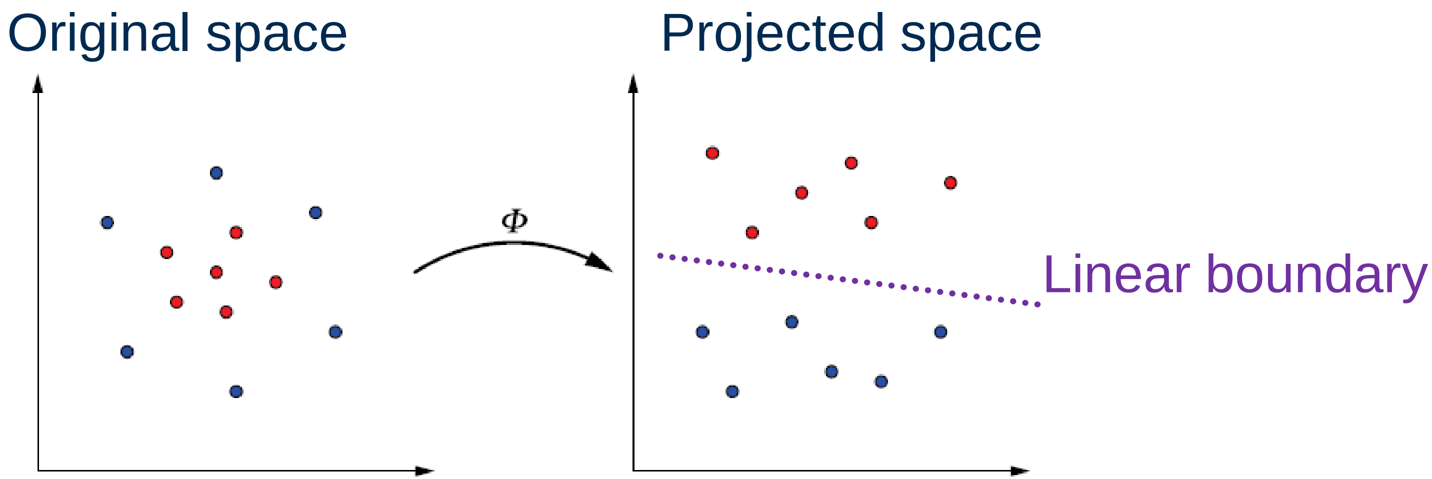
where $R = \max_j \lVert \boldsymbol{x}_j \rVert$

- Harder to learn examples having larger alpha
- The information about training examples enters algorithm only through the inner products (which we could pre-compute)

- So far, we have seen training via computation of inner products

→ Indicating which side of the linear decision boundary $\boldsymbol{x}$ falls into

- Say, it is hard to find a linear boundary in the original space



- **Solution**: project to another space, find the linear boundary in the projected space, classify in the projected space

Kernel Methods and Support Vector Machines

- A **kernel function** is a function K, such that for all $(x, y) \in X$

$$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \Phi(\boldsymbol{x}_1)^T \Phi(\boldsymbol{x}_2)$$

where Φ is a mapping from X to an (inner product) feature space F.

- It is not necessary to transform the original data into the projected space before learning linear SVM
- The kernel K allows us to compute the inner product of two points x and y in the projected space without even entering that space

- The discriminant function in the projected space

$$f(\boldsymbol{x}) = \left(\sum_{j=1}^{n} \alpha_j \cdot y_j \cdot \Phi(\boldsymbol{x})^T \Phi(\boldsymbol{x}_j)\right) + b$$

- Or with the kernel function K

$$f(\boldsymbol{x}) = \left(\sum_{j=1}^{n} \alpha_j \cdot y_j \cdot K(\boldsymbol{x}, \boldsymbol{x}_j)\right) + b$$

Gram Matrix

All data necessary for

- the decision function $h(\boldsymbol{x})$
- the training of the coefficients

can be pre-computed using a Gram matrix K

$$K = \begin{pmatrix} K(\boldsymbol{x}_1, \boldsymbol{x}_1) & K(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & K(\boldsymbol{x}_1, \boldsymbol{x}_m) \\ K(\boldsymbol{x}_2, \boldsymbol{x}_1) & K(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & K(\boldsymbol{x}_2, \boldsymbol{x}_m) \\ \vdots & \vdots & \ddots & \vdots \\ K(\boldsymbol{x}_m, \boldsymbol{x}_1) & K(\boldsymbol{x}_m, \boldsymbol{x}_2) & \cdots & K(\boldsymbol{x}_m, \boldsymbol{x}_m) \end{pmatrix}$$

- Let X be a non empty set. A function is a valid kernel in X if for all n and all $x_1, \ldots, x_n \in X$ it produces a Gram matrix K, which is:

- Symmetric

$$K = K^T$$

- Positive semi-definite

$$\forall \boldsymbol{\alpha} : \boldsymbol{\alpha}^{\boldsymbol{T}} K \boldsymbol{\alpha} \geq 0$$

- Eigenvectors of the matrix correspond to the input vectors

Moreover,

- Every positive definite & symmetric matrix is a Gram matrix

- A simple kernel is

$$K(x,y) = (x_1 y_1 + x_2 y_2)^2$$

- And the corresponding projected space:

$$(x_1, x_2) \mapsto \Phi(\boldsymbol{x}) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2)$$

- Since

$$\langle x, y \rangle^2 = \langle (x_1, x_2), (y_1, y_2) \rangle^2$$
$$= \left\langle \left(x_1^2, x_2^2, \sqrt{2} x_1 x_2 \right), \left(y_1^2, y_2^2, \sqrt{2} y_1 y_2 \right) \right\rangle$$
$$= x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 x_2 y_1 y_2$$
$$= (x_1 y_1 + x_2 y_2)^2$$

- A few less simple kernels are

$$K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y})^d$$

- And the corresponding projected spaces are of dimension

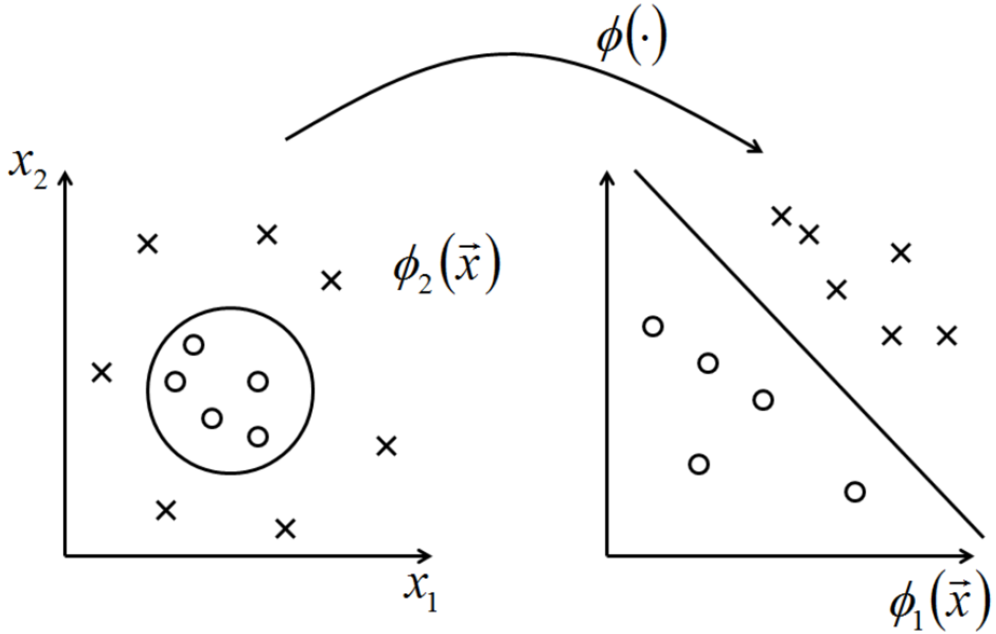
$$\binom{n + d - 1}{d}$$

- But computing the inner products in the projected space can quickly become expensive

More Kernels

- Polynomial kernel of degree d

$$K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y} + c)^d$$

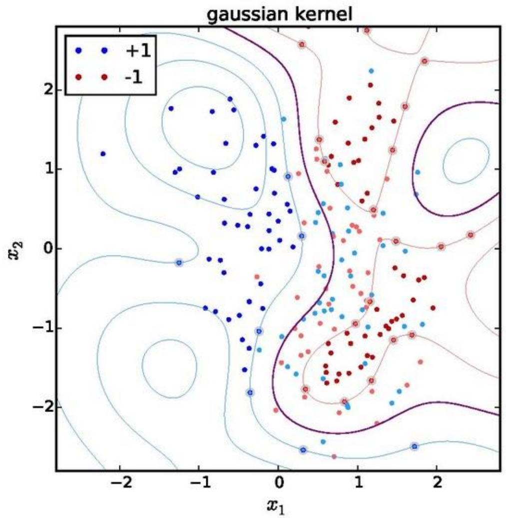


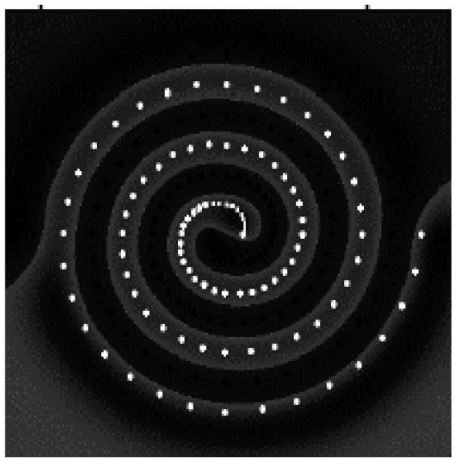
More Kernels

- Gaussian kernel

$$K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\sigma^2}}$$

- Also known as radial basis function (RBF) kernel





- Note that we do not need to know the projection Φ.
- It is sufficient to prove that $K(\cdot)$ is a Kernel.

A few notes:

- Kernels are modular and closed: we can compose new Kernels based on existing ones
- Kernels can be defined over non numerical objects:
  - Text: e.g. string matching kernel
  - Images, trees, graphs…
- A good kernel is crucial
  - Gram Matrix diagonal: classification easy and useless

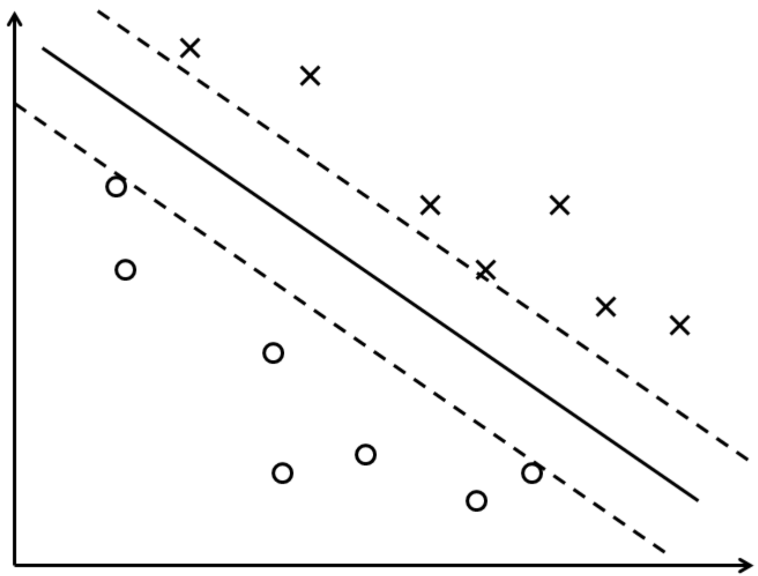
- Finding the hyperplane (in any space) still leaves lots of room for variations
- We can define “margins” of individual training examples:

$$\gamma_i = y_i(\boldsymbol{w}^T \boldsymbol{x} + b)$$

appropriately normalized this is a “geometrical” margin

- The margin of a hyperplane (with respect to a training set): $\min_{i=1\ldots n} \gamma_i$
- And a maximal margin of all training examples indicates the maximum margin over all hyperplanes

(maximum) Margin of a Hyperplane



- The original objective function

$$y_i \cdot (\boldsymbol{w}^T \boldsymbol{x} + b) \geq 0$$

- Is reformulated slightly:

$$y_i \cdot (\boldsymbol{w}^T \boldsymbol{x} + b) \geq 1$$

- The decision line is still defined by

$$\boldsymbol{w}^T \boldsymbol{x} + \mathrm{b} = 0$$

- And in addition the upper and lower margins are defined by

$$\boldsymbol{w}^T \boldsymbol{x} + \mathrm{b} = \pm 1$$

- The distance between those two hyperplanes is $\frac{2}{||\boldsymbol{w}||}$

- Finding the maximum margin now turns into a minimization problem:
  - Minimize (in $\boldsymbol{w}, b$)

    $||\boldsymbol{w}||$
  - subject to (for any $j = 1, \dots, n$)

    $y_i(\boldsymbol{w}^T\boldsymbol{x} - b) \geq 1$

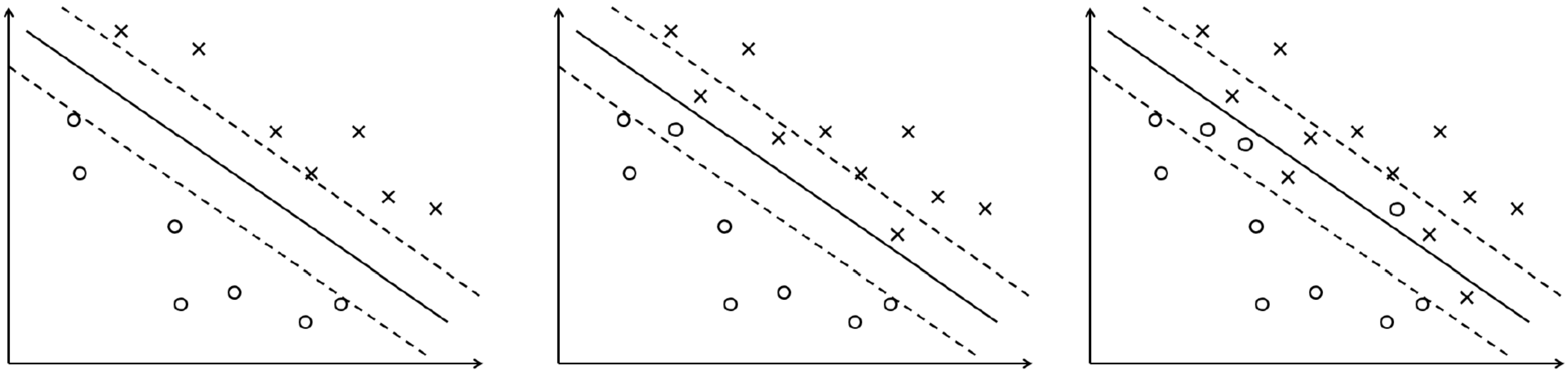
Solution sketch:

- Solutions depend on $||\boldsymbol{w}||$, the norm of $\boldsymbol{w}$ which involves a square root
- Convert into a quadratic form by substituting $||\boldsymbol{w}||$ with $\frac{1}{2}||\boldsymbol{w}||^2$ without changing the solution
- Using Lagrange multipliers this turns into a standard quadratic programming problem

Margin of Error and Variations

- What can we do if no linear separating hyperplane exists?
- Solution: allow minor violations – also known as ***soft margins***

→ In contrast, avoiding any misclassifications ≡ ***hard margins***



Hard margins ⟷ ***Soft margins***

- How do we implement soft margins? → via ***slack variables*** ε_j
- Introducing the slack variables to the minimization constraint

$$\forall j = 1, \ldots, n: \quad y_j \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_j + b\right) \geq 1 - \varepsilon_j$$

- Misclassifications are allowed if slack $\varepsilon_j > 1$ is allowed
- The minimization problem is solved using Lagrange multipliers

$$\arg\min \frac{1}{2} ||\boldsymbol{w}||^2 + C \sum_j \varepsilon_j$$

- Subject to: $y_j \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_j + b\right) \geq 1 - \varepsilon_j$
- The regularization parameter $C > 0$ controls the “hardness” of the margins (large C → hard margins, small C → soft margins)

How do we separate more than two classes?

- Transform the problem into a set of binary classification problems
  - One class vs. all other classes
  - One class vs. another class, for all possible class pairs

- The class with the farthest distance from the hyperplane wins

- The key idea: change the optimization

$$\arg\min \frac{1}{2} ||w||^2$$

- Subject to:

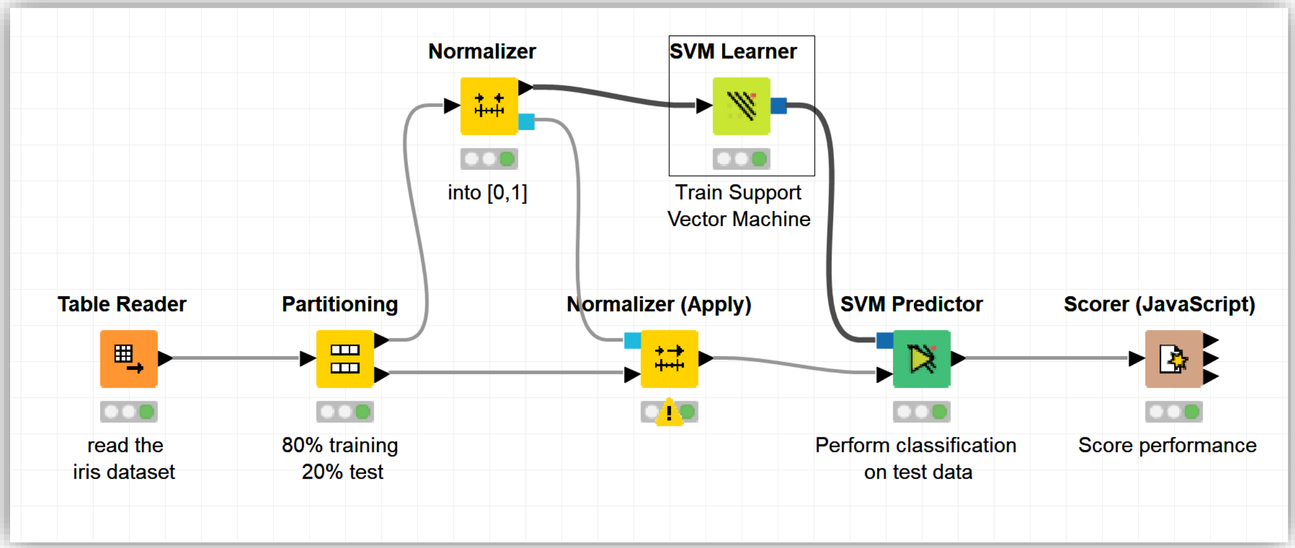
$$y_j - \left(\boldsymbol{w}^T \boldsymbol{x}_j + b\right) \leq \varepsilon \qquad \text{for } 1 \leq j \leq n$$

- This require the prediction error to be within a margin ε
- We can introduce slack variables to tolerate larger errors

- Support Vector Machine
  - Classifier as weighted sum over inner products of training pattern (or only support vectors) and the new pattern.
  - Training analog
- Kernel-Induced feature space
  - Transformation into higher-dimensional space (where we will hopefully be able to find a linear separation plane).
  - Representation of solution through few support vectors ($\alpha > 0$).
- Maximum Margin Classifier
  - Reduction of Capacity (Bias) via maximization of margin (and not via reduction of degrees of freedom).
  - Efficient parameter estimation.
- Relaxations
  - Soft Margin for non separable problems.

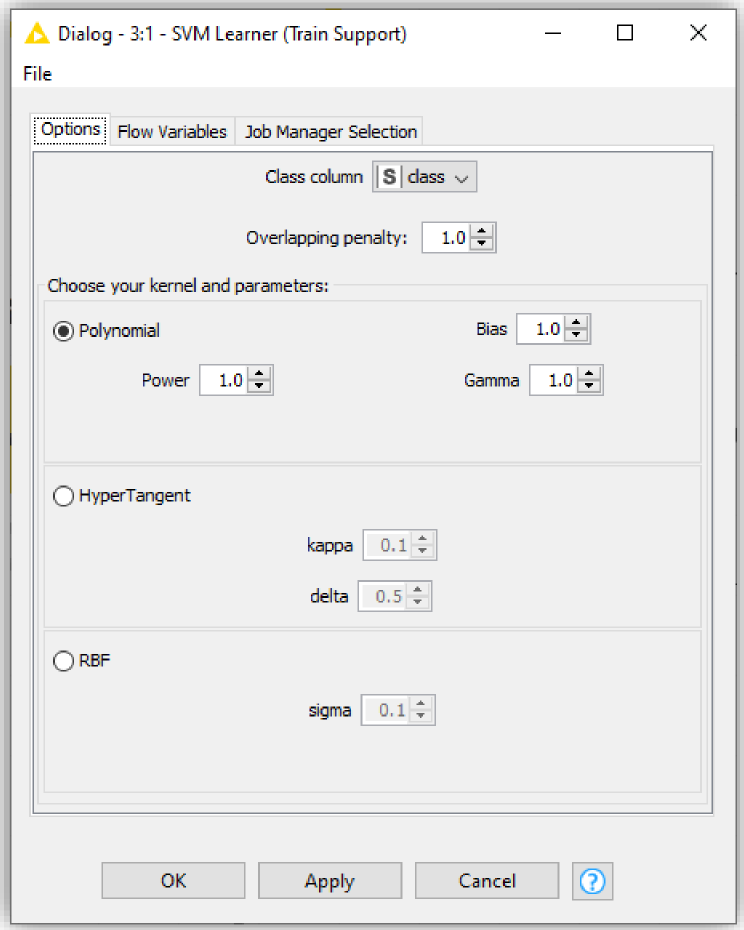
Practical Examples with KNIME Analytics Platform

SVM on the Iris Data



– Workflow training an SVM model to classify the iris data set

- The configuration window of the SVM Learner node
- Allows a selection of a kernel and the associated parameters
- Overlapping penalty controls the margin hardness



Thank you

For any questions please contact: education@knime.com