Non-parametric Models and Support Vector Machines

- Recommended reading
  - Textbook: The Elements of Statistical Learning
  - Chapter 2.3: k-nearest neighbour method
  - Chapter 4.5: Linear support vector machines
  - Chapter 12.1-12.3: Nonlinear support vector machines

Curtin University

Outline

- Parametric models vs non-parametric models
- Nearest neighbour methods
- Linear support vector machines: optimal separating hyperplane
- Non-linear support vector machines

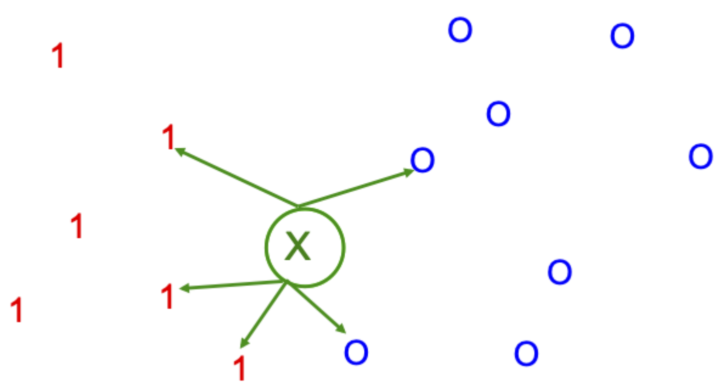
Parametric models vs non-parametric models

- Parametric models
  - Modelled with a fixed-dimensional vector of parameters
  - Linear models: weights and bias
  - Neural networks: weights and biases for each of many layers
  - Parameters are learnt from training data
  - Training data are not used in testing
- Non-parametric models
  - Training data are used in testing
  - Use the similarity/distance between a test input and each of the training inputs.
  - Often called exemplar-based models, instance-based learning or memory-based learning.)
  - Examples: Nearest neighbour methods, kernel methods

Curtin University

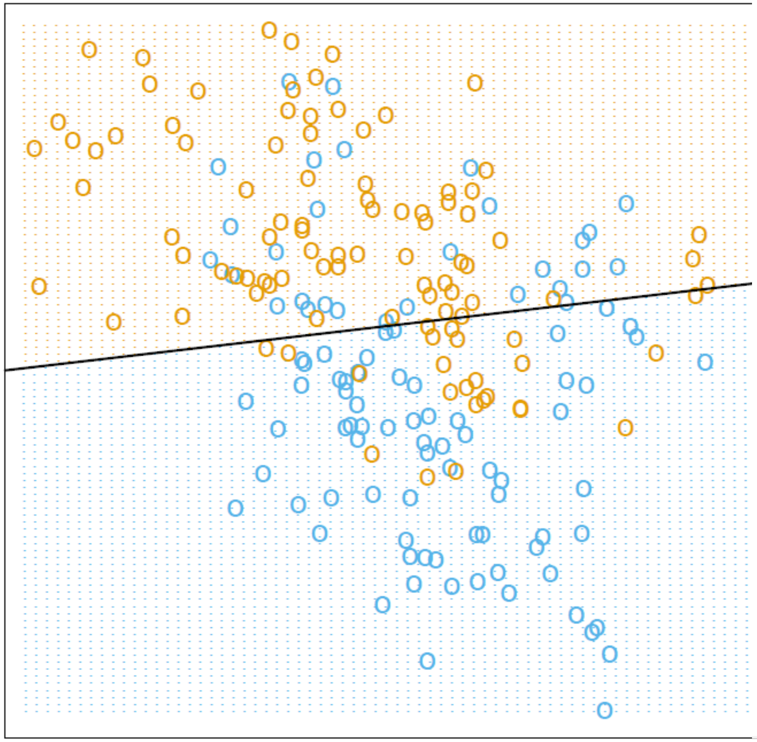
Nearest Neighbour Methods

- KNN: *k*-nearest neighbour classifier: to classify a new input
  - we first find the *k* closest examples to this new input in the training set
  - then use majority voting to classify the new input

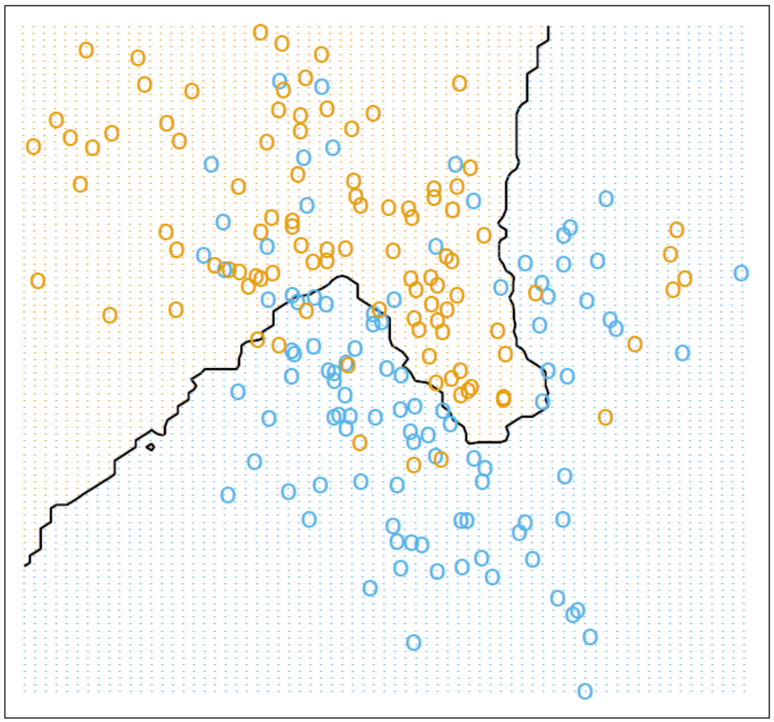


Source: Fig 16.1 from *Probabilistic Machine Learning by* Kevin P. Murphy.

Linear Method vs Nearest Neighbour Methods



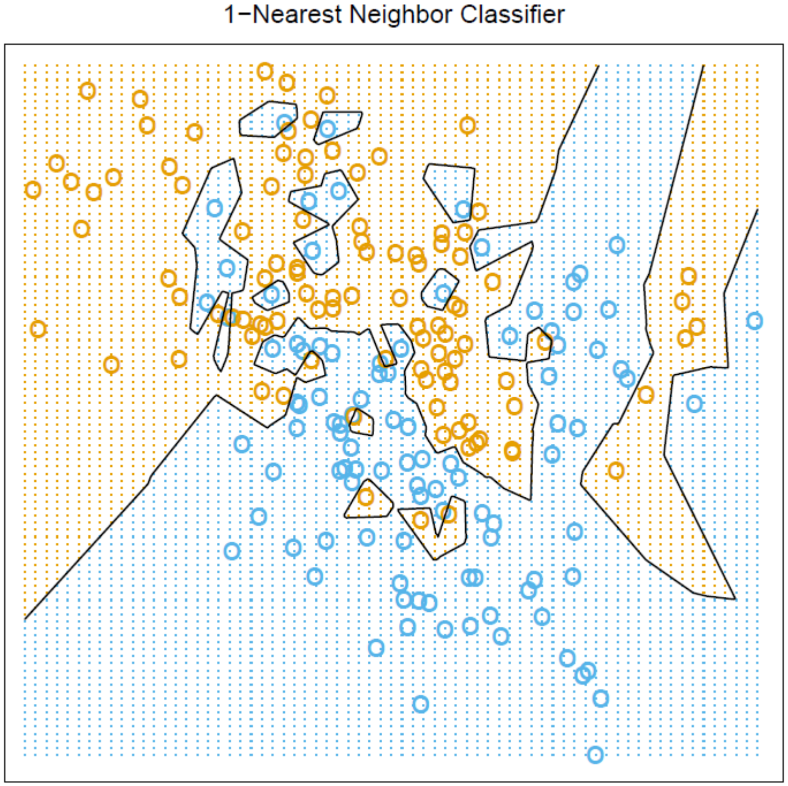
Source: Fig. 2.1 of the textbook



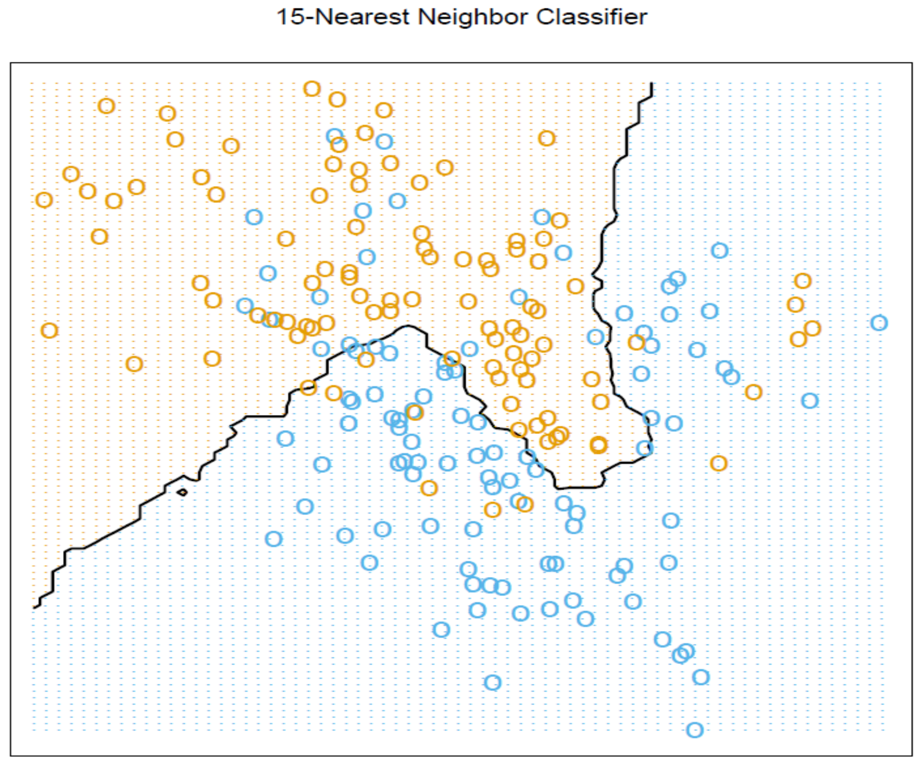
Source: Fig. 2.2 of the textbook

1-NN vs 15-NN

1-NN (Fig 2.3 of the texbook)



15-NN (Fig. 2.2 of the texbook)



Metrics for Nearest Neighbour Methods

- Source: https://scikit-learn.org/0.24/modules/generated/sklearn.neighbors.DistanceMetric.html
- One can also apply KNN with these metrics in feature space such as CNN features

Metrics intended for real-valued vector spaces:

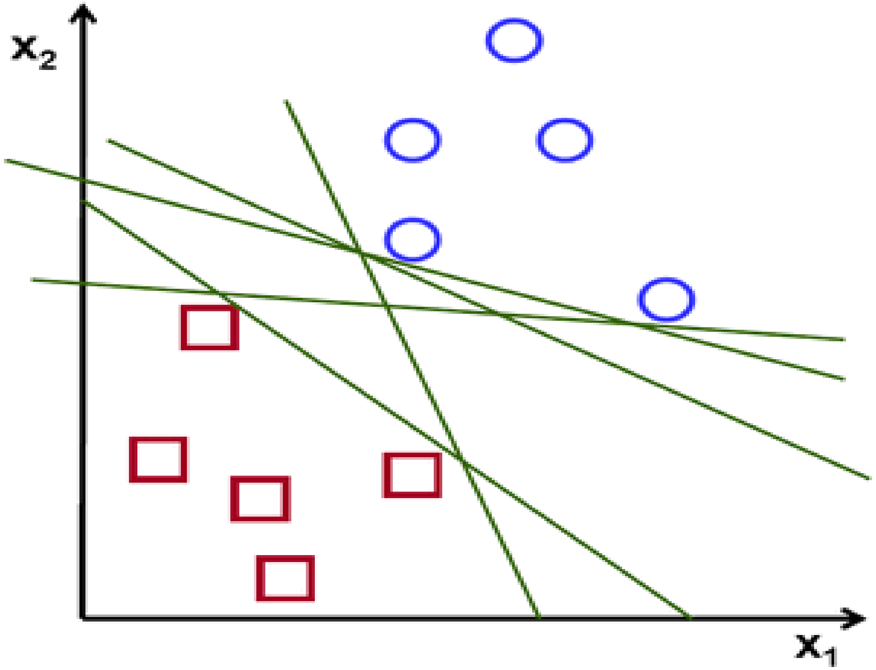
identifier	class name	args	distance function
"euclidean"	EuclideanDistance	•	`sqrt(sum((x - y)^2))`
"manhattan"	ManhattanDistance	•	`sum(|x - y|)`
"chebyshev"	ChebyshevDistance	•	`max(|x - y|)`
"minkowski"	MinkowskiDistance	p	`sum(|x - y|^p)^(1/p)`
"wminkowski"	WMinkowskiDistance	p, w	`sum(|w * (x - y)|^p)^(1/p)`
"seuclidean"	SEuclideanDistance	V	`sqrt(sum((x - y)^2 / V))`
"mahalanobis"	MahalanobisDistance	V or VI	`sqrt((x - y)' V^-1 (x - y))`

Pros and Cons of KNN

- Pros
  - Simple
  - Memory based: no need to train a mode
  - Can be used with the number of training examples is not very large
- Cons
  - Slow when training size is large
  - Sensitive to local noise/outliers
  - Does not work well for high dimensional data due to curse of dimensionality.

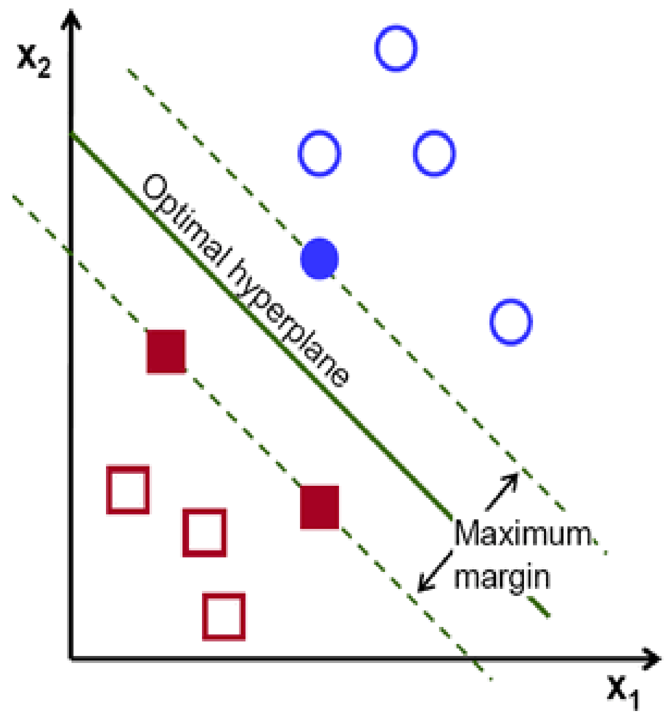
Image adapted from Quasi-dense Object Tracking (Hu et. al. 2021)

Linear SVM for Optimal Separating Boundary



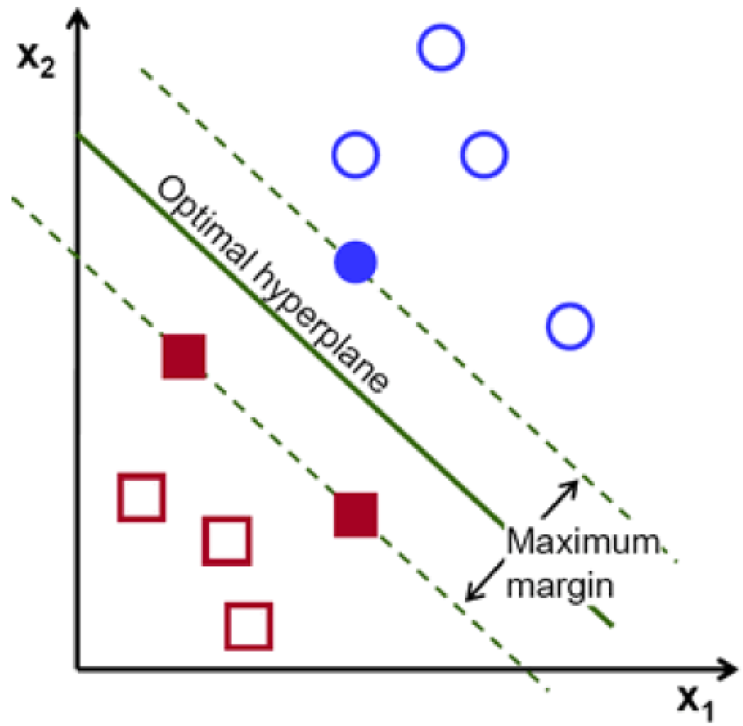
Linear Support Vector Machine

- Well-known non-probabilistic supervised earning method
- Designed for binary classification (multi-class extension available)
- Based on linear decision boundaries
- Non-linear decision boundaries possible via the kernel trick: mapping input space to high-dimensional space where decision boundaries are linear
- Linear boundaries: separating hyperplane
- Maximum margin classifier



Linear SVM

- Some important concepts
  - Margin: degree of separability of two classes
  - Separable case: no training samples found in the maximal margin
  - Non-separable case: allow training errors
  - Signed distance: orthogonal to the separating hyperplane
- Optimization-based method: separating hyperplane found from solving a convex optimization formulation



Source

Margin as a Function of the Weights

- Training examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
- Each input $\mathbf{x}_i$ is a vector of d dimensions (assume real-valued features, attributes, etc.)
- Each output y_i takes either $+1$ or -1 (class labels)
- Hyperplane

$$\mathbf{x} : f(\mathbf{x}) = \boldsymbol{\beta}^T \mathbf{x} + \beta_0$$

  - $\boldsymbol{\beta}$ normal vector of the hyperplane
  - β_0: bias / offset parameter
- Learning goal: find $\boldsymbol{\beta}$, β_0
- What to optimize? maximize the margin (i.e. maximize the separability between two classes)

$$M = \frac{1}{\|\boldsymbol{\beta}\|_2}$$

Margin as a Function of the Weights

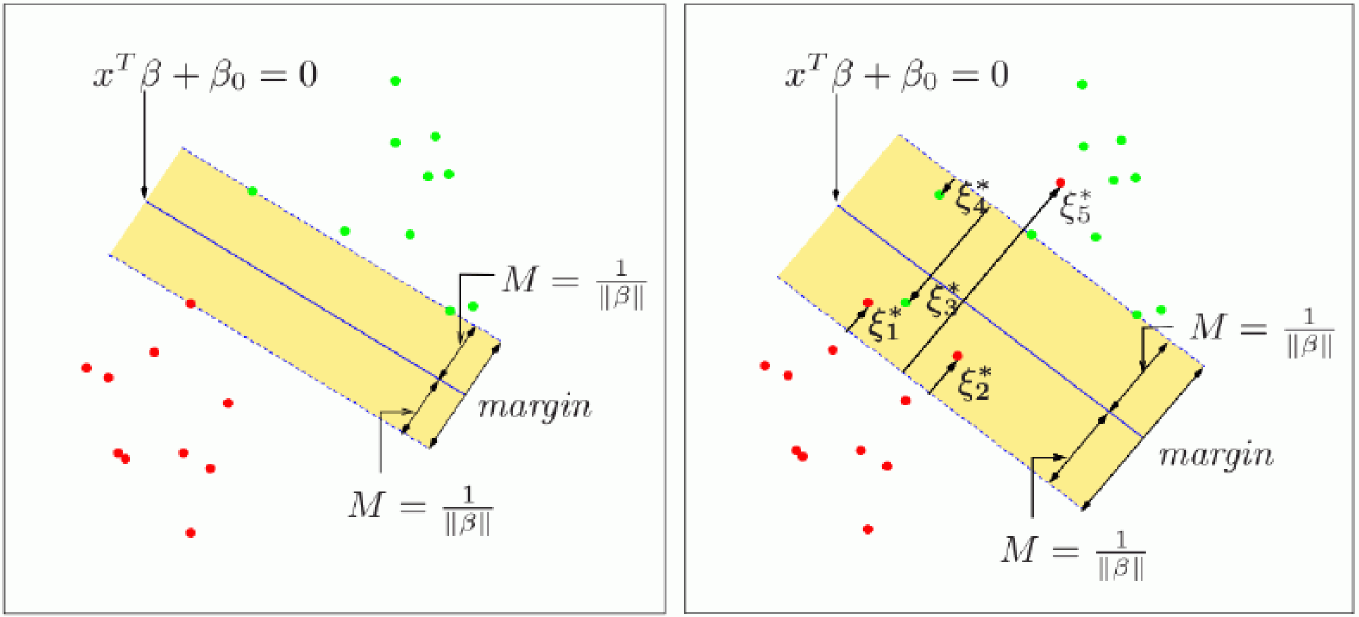


FIGURE 12.1. *Support vector classifiers. The left panel shows the separable case. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width $2M = 2/\|\beta\|$. The right panel shows the nonseparable (overlap) case. The points labeled ξ_j^* are on the wrong side of their margin by an amount $\xi_j^* = M\xi_j$; points on the correct side have $\xi_j^* = 0$. The margin is maximized subject to a total budget $\sum \xi_i \leq$ constant. Hence $\sum \xi_j^*$ is the total distance of points on the wrong side of their margin.*

Linear SVM Formulation

- How to describe training examples?
  - Positive-class examples above the +ve boundary

$$(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) \geq 1, \forall y_i = 1$$

  - Negative-class examples below the -ve boundary

$$(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) \leq -1, \forall y_i = -1$$

Combining

$$y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) \geq 1$$

- How to allow training errors? Introduce slack variables $\xi_i \geq 0$

$$y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) \geq (1 - \xi_i)$$

  - $\xi_i = 0$ perfectly positioned, no training error
  - $\xi_i > 0$ training error

Curtin University

Linear SVM – Hard Margin

Formulation for linearly separable patterns

- Maximize the margin $\iff$ minimize $\|\boldsymbol{\beta}\|_2^2$
- No training errors

Mathematical expressions

$$\begin{aligned} \min_{\boldsymbol{\beta},\beta_0} \quad & \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 \\ \text{subject to} \quad & y_i(\boldsymbol{\beta}^T\mathbf{x}_i + \beta_0) \geq 1 \end{aligned}$$

Observation: quadratic optimization problem with linear inequality constraints.

Linear SVM formulation – Soft Margin

Formulation

- Maximize the margin $\Longleftrightarrow$ minimize $\|\beta\|_2^2$
- Minimize the training errors

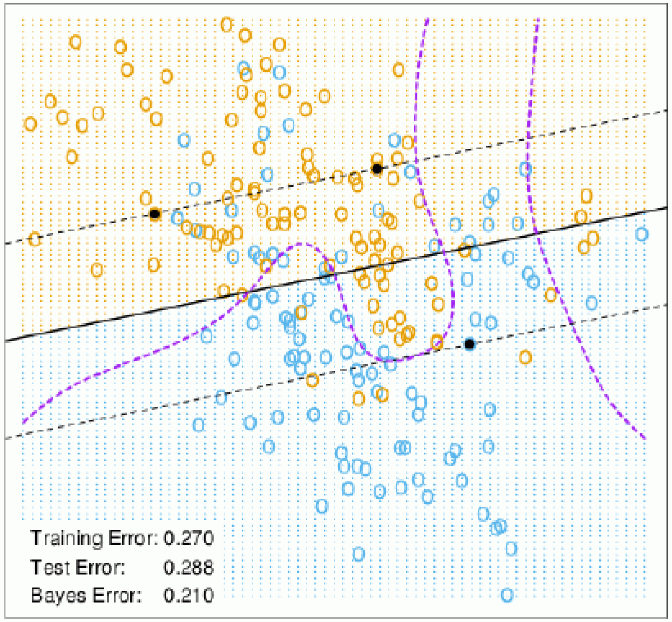
Mathematical expressions (no need to memorize, just to understand)

$$
\begin{aligned}
\min_{\beta,\beta_0} \quad & \frac{1}{2}\|\beta\|_2^2 + C\sum_{i=1}^{N}\xi_i \\
\text{subject to} \quad & \xi_i \geq 0 \\
& y_i(\beta^T \mathbf{x}_i + \beta_0) \geq (1 - \xi_i)
\end{aligned}
$$

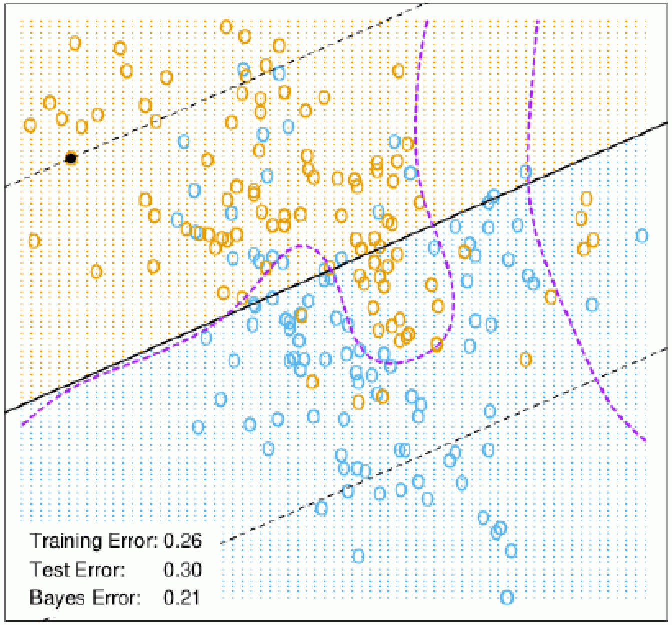
Observation: quadratic optimization problem with inequality constraints $\rightarrow$ specialized algorithms.

Curtin University

SVM - Example



$C = 10000$



$C = 0.01$

Source [4]

Why "support vector" ?

- It can be proved that the $\pm$ve decision boundaries pass through a number of training samples from each class
- These samples are called support vectors

SVM with Kernels

- Extension to obtain nonlinear decision boundaries
- Reformulation of linear classifiers

$$\beta^T \mathbf{x} + \beta_0 = \sum_{i=1}^{N_{\text{train}}} w_i y_i \mathbf{x}_i^T \mathbf{x} + w_0 \tag{1}$$

  - w_i: weight of the i-th training example
  - y_i: label of the i-th training example
  - $\mathbf{x}_i$: the i-th training example
- Observation: The right side formulation is very useful for nonlinear feature mapping $\mathbf{x} \mapsto \Phi(\mathbf{x})$. Let $k(\mathbf{x}, \mathbf{z}) = \Phi(\mathbf{x})^T \Phi(\mathbf{z})$, we have

$$\beta^T \Phi(\mathbf{x}) + \beta_0 = \sum_{i=1}^{N_{\text{train}}} w_i y_i k(\mathbf{x}_i, \mathbf{x}) + w_0 \tag{2}$$

  The linear classifier in the feature space can be formulated without $\Phi(\cdot)$. Only the kernel $k(\cdot, \cdot)$ is required.

The Kernel Trick

- The kernel trick
  - SVM solution only requires inner products to be defined
  - Nonlinear decision boundaries in the input space can be mapped to linear decision boundaries in a transformed space with sufficiently large dimensions

$$\mathbf{x} \mapsto \Phi(\mathbf{x})$$

  - We do not need to know the actual space of $\Phi(\mathbf{x})$
  - However, we need to know the similarity between two input samples $\mathbf{x}_1$ and $\mathbf{x}_2$

$$\textit{similarity}(\mathbf{x}_1, \mathbf{x}_2) = \textit{similarity}(\Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2))$$

  - Similarity in the input space is defined through a kernel function

$$\textit{similarity}(\mathbf{x}_1, \mathbf{x}_2) = k(\mathbf{x}_1, \mathbf{x}_2)$$

  - Similarity in the feature mapping space of Φ is defined through an inner product
  - Mercer condition: requires k to satisfy certain condition in order for an implicit mapping Φ to exist

SVM with Kernels

- Linear classifier in the space of Φ is equivalent to the nonlinear kernelized binary classifier of the form

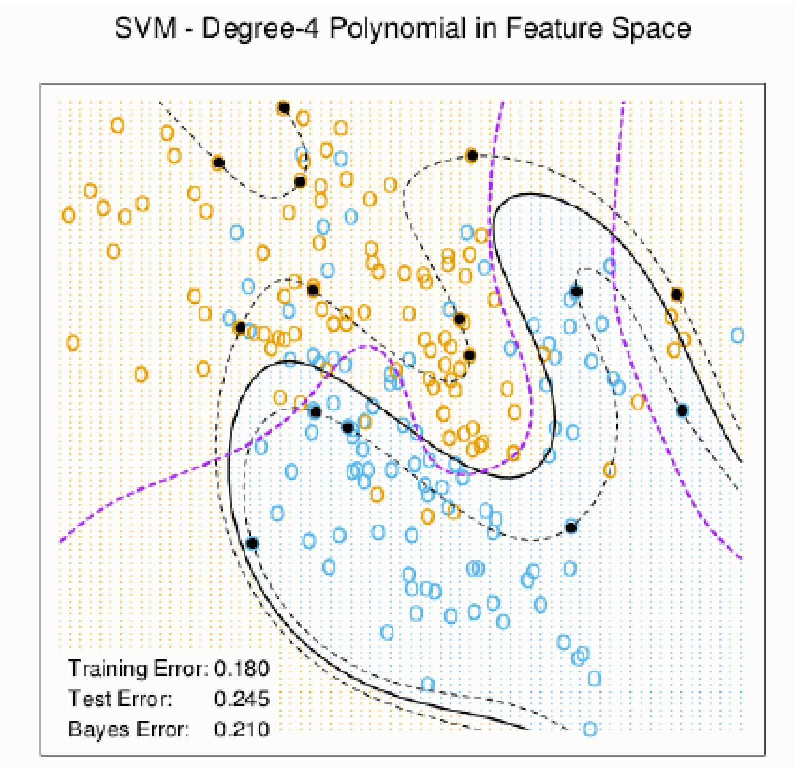
$$\hat{y} = \text{sign}\left(\sum_{i=1}^{N_{\text{train}}} w_i y_i k(\mathbf{x}_i, \mathbf{x}) + w_0 \right)$$

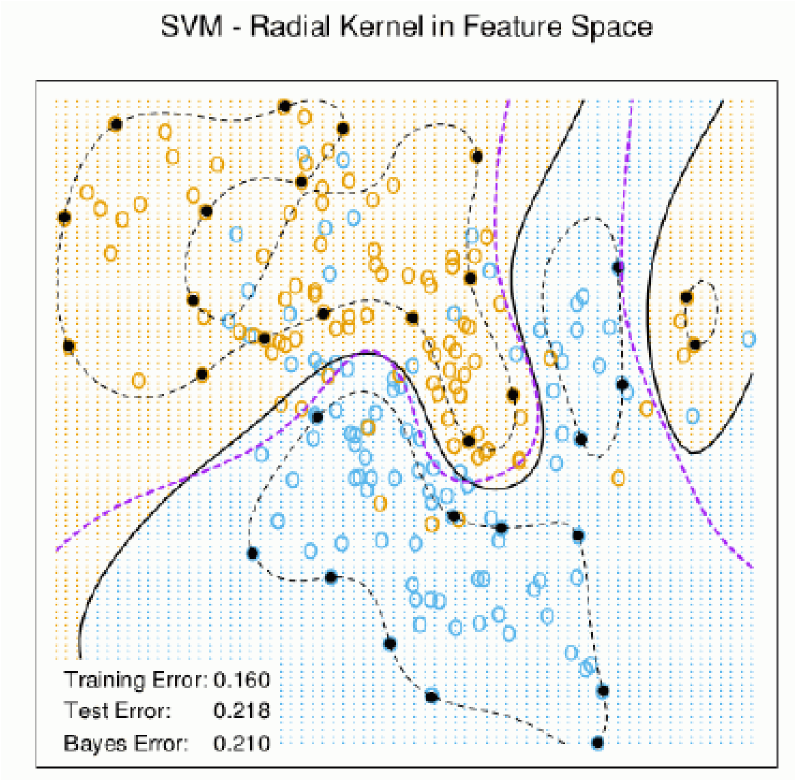
  - w_i: weight of the i-th training example
  - y_i: label of the i-th training example
  - $\mathbf{x}_i$: the i-th training example
- Training the classifier = finding the weights w_i
- It can be shown the equivalent formulation is a convex optimization problem quadratic in terms of $\mathbf{w}$
- Dimension of the weight vector $\mathbf{w}$ is the number of training examples

SVM with Kernels

- Kernel function $k(\mathbf{x}_1, \mathbf{x}_2)$ defines
  - Inner product in the feature space $k(\mathbf{x}_1, \mathbf{x}_2) = \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_1) \rangle$
  - The similarity in the input space between $\mathbf{x}_1$ and $\mathbf{x}_2$
- Similarity:
- Popular kernel choices
  - Radial basis function (RBF) kernel
    $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\gamma \|\mathbf{x}_1 - \mathbf{x}_2\|_2^2\right), \gamma > 0$
  - Polynomial kernel with degree d: $k(\mathbf{x}_1, \mathbf{x}_2) = (1 + \mathbf{x}_1^T \mathbf{x}_2)^d$
  - Linear kernel $k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^T \mathbf{x}_2$
  - Sigmoid/Neural networks kernel $k(\mathbf{x}_1, \mathbf{x}_2) = tanh(\gamma \mathbf{x}_1^T \mathbf{x}_2 + r)$
- Kernel parameters typically selected via cross validation

SVM-Example Revisited





Curtin University

SVM for Multiple Classes

- How to deal with $m > 2$ classes?
  - One-versus-all: design m binary SVM classifiers that classify one class versus the rest. At test time, select the result with largest margin
  - One-versus-one: design $m(m-1)/2$ binary SVM classifier that classify one class versus another. At test time, select the class selected by most of the classifiers (majority voting)
  - All-versus-all: extend SVM formulation to multivariate labels $\rightarrow$ a little more complex
- Pre-processing
  - Categorical attributes $\rightarrow$ binary representation: (red,green,blue) $\rightarrow$ (1,0,0) (0,1,0) (0,0,1)
  - Scaling to the range [-1,+1] or [0,1] to avoid numerical dominance of some attributes
- When to use linear kernel to reduce training cost?
  - Number of features very large (no need for mapping)
  - Number of training examples and number of features are large

SVM for Regression

- Linear regression model

$$f(x) = x^T\beta + \beta_0,$$

- Labels are real numbers instead of 1 or -1

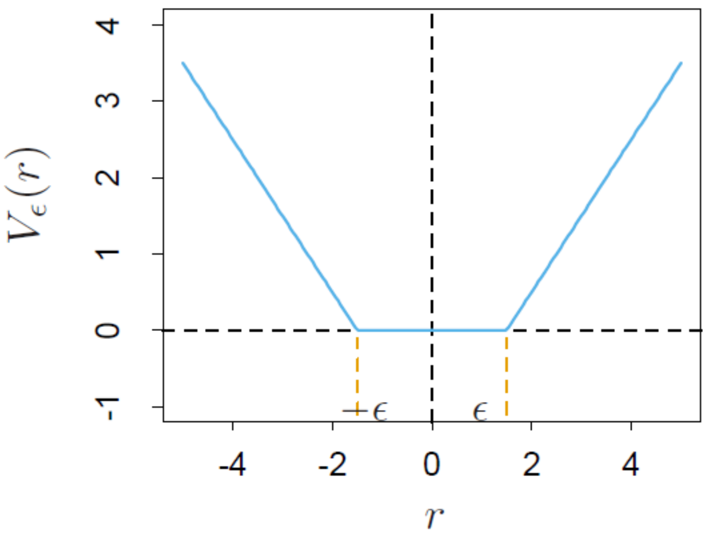
- Loss function

$$H(\beta, \beta_0) = \sum_{i=1}^{N} V(y_i - f(x_i)) + \frac{\lambda}{2}||\beta||^2,$$

SVM for Regression

- epsilon-insensitive loss

$$V_\epsilon(r) = \begin{cases} 0 & \text{if } |r| < \epsilon, \\ |r| - \epsilon, & \text{otherwise.} \end{cases}$$



Pros and Cons of SVM

- Pros
  - Maximum margin for linear separable data
  - Convex optimization: no local minima problem
  - Kernels: efficient ways to design nonlinear features
  - Small number of hyper-parameters: regularization number and kernel parameters
- Cons
  - The capacity for nonlinear feature development is very limited due to few parameters of kernels