Support Vector Machines

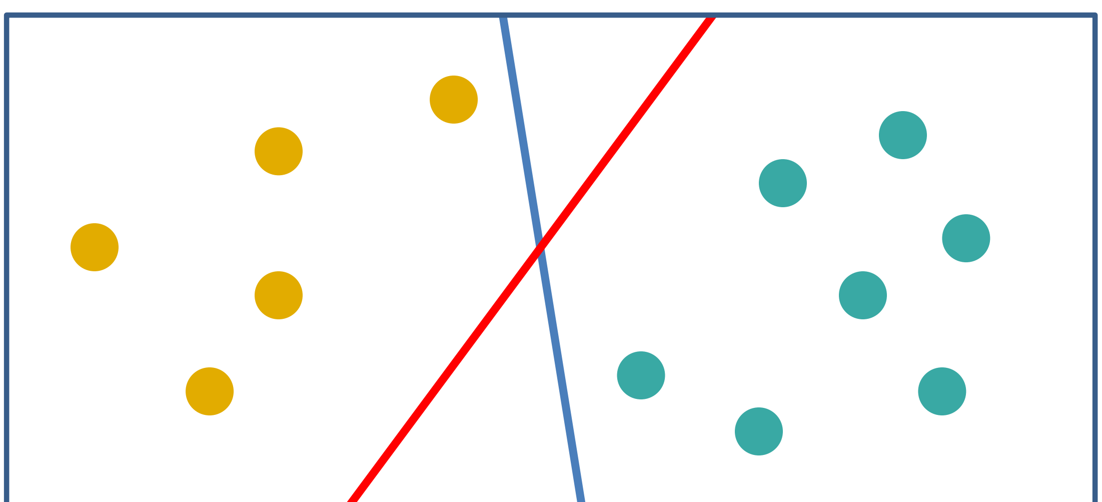
CSE 4309 – Machine Learning
Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

A Linearly Separable Problem

- Consider the binary classification problem on the figure.
  - The blue points belong to one class, with label +1.
  - The orange points belong to the other class, with label -1.
- These two classes are linearly separable.
  - Infinitely many lines separate them.
  - Are any of those infinitely many lines preferable?

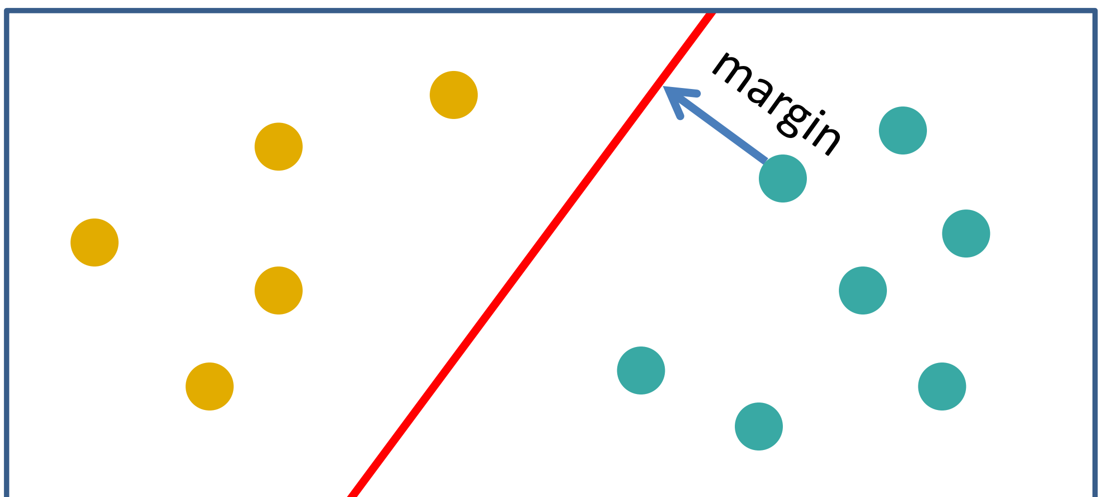
A Linearly Separable Problem

- Do we prefer the blue line or the red line, as decision boundary?
- What criterion can we use?
- Both decision boundaries classify the training data with 100% accuracy.



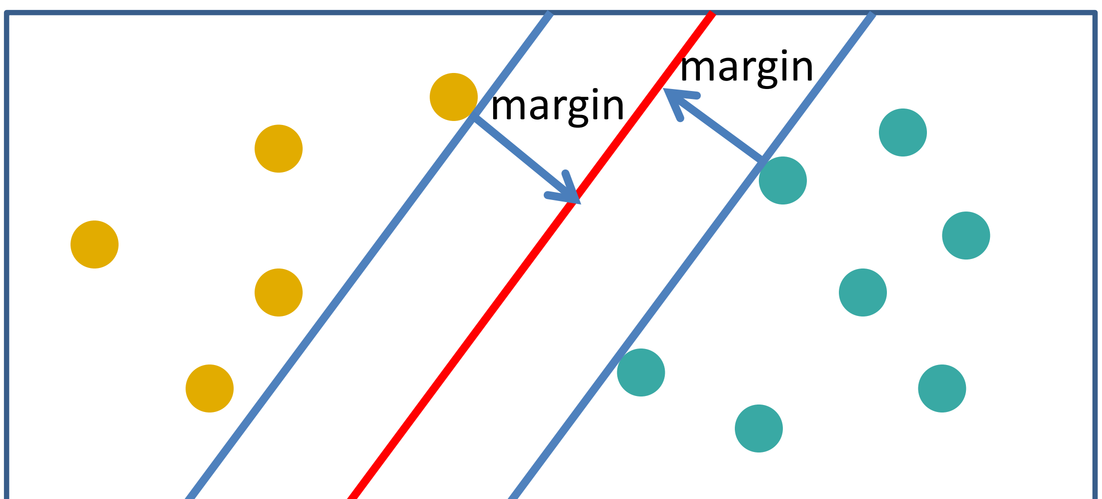
Margin of a Decision Boundary

- The **margin** of a decision boundary is defined as the smallest distance between the boundary and any of the samples.



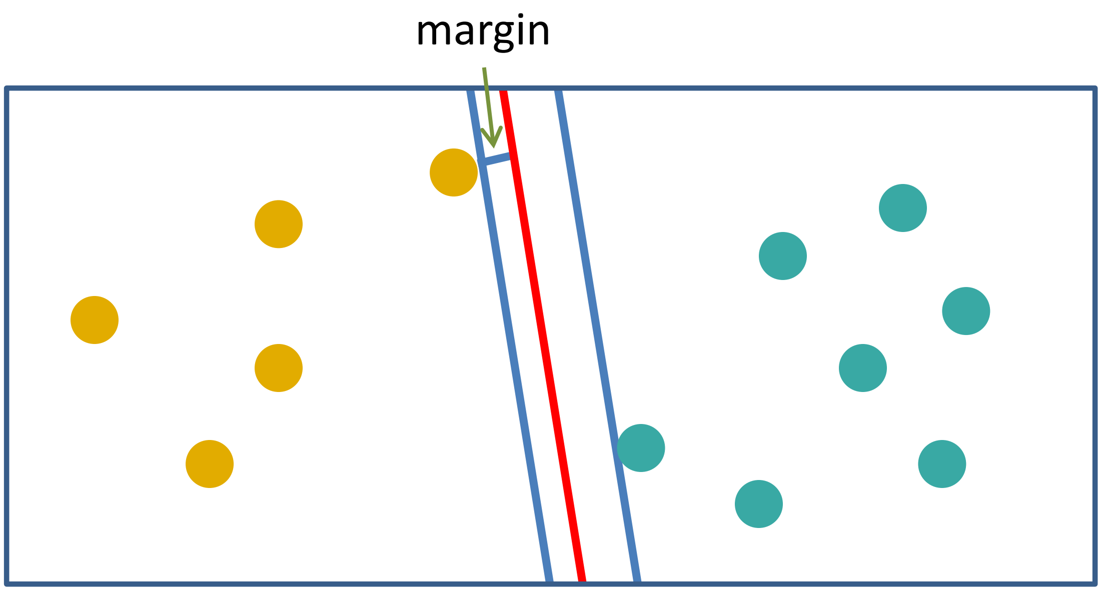
Margin of a Decision Boundary

- One way to visualize the margin is this:
  - For each class, draw a line that:
    - is parallel to the decision boundary.
    - touches the class point that is the closest to the decision boundary.
  - The margin is the smallest distance between the decision boundary and one of those two parallel lines.
    - In this example, the decision boundary is equally far from both lines.



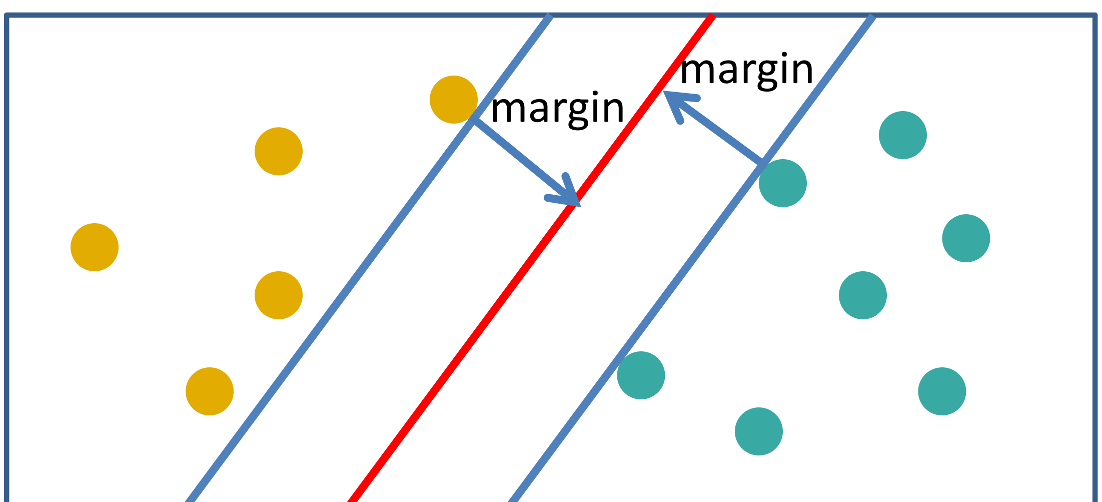
Support Vector Machines

- One way to visualize the margin is this:
  - For each class, draw a line that:
    - is parallel to the decision boundary.
    - touches the class point that is the closest to the decision boundary.
  - The margin is the smallest distance between the decision boundary and one of those two parallel lines.



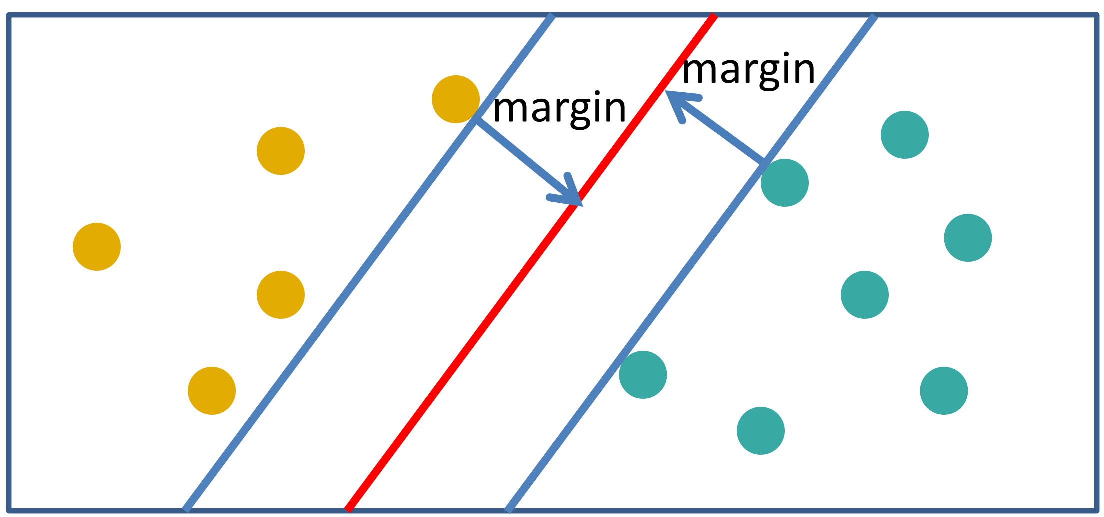
Support Vector Machines

- **Support Vector Machines** (SVMs) are a classification method, whose goal is to find the decision boundary with the **maximum margin**.
  - The idea is that, even if multiple decision boundaries give 100% accuracy on the training data, larger margins lead to less overfitting.
  - Larger margins can tolerate more perturbations of the data.



Support Vector Machines

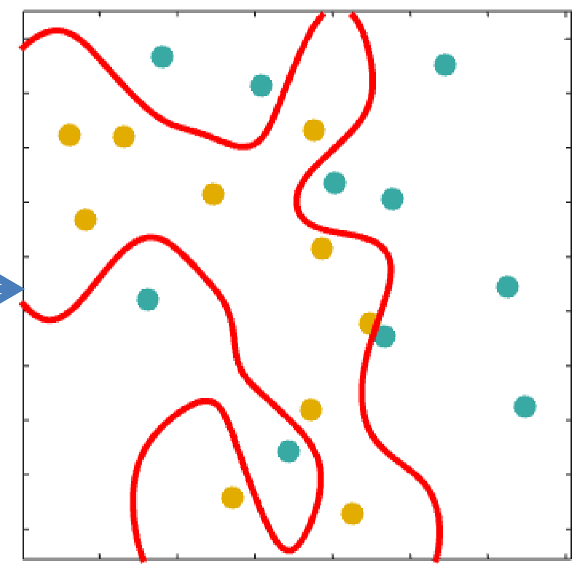
- Note: so far, we are only discussing cases where the training data is linearly separable.
- First, we will see how to maximize the margin for such data.
- Second, we will deal with data that are not linearly separable.
  - We will define SVMs that classify such training data imperfectly.
- Third, we will see how to define nonlinear SVMs, which can define non-linear decision boundaries.



Support Vector Machines

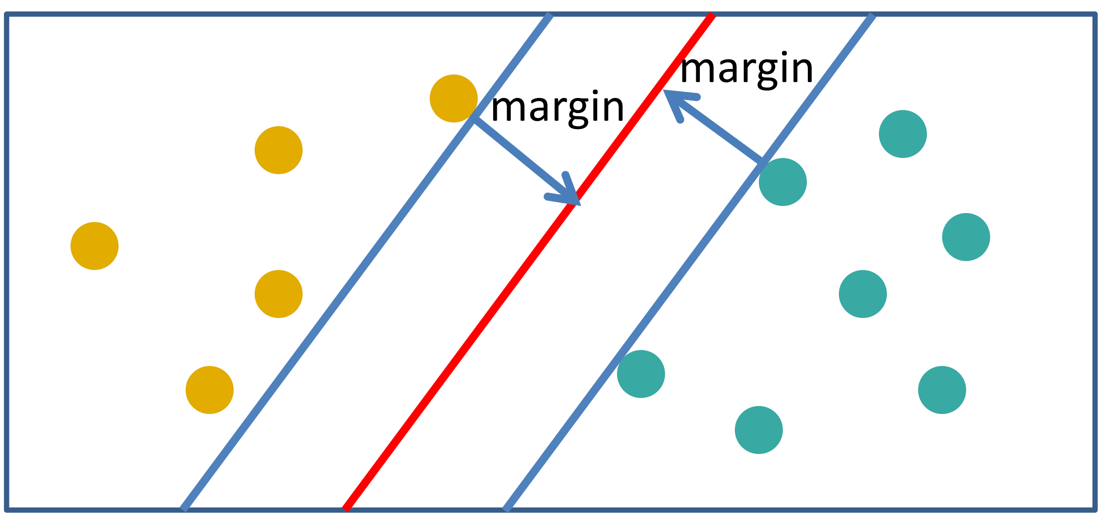
- Note: so far, we are only discussing cases where the training data is linearly separable.
- First, we will see how to maximize the margin for such data.
- Second, we will deal with data that are not linearly separable.
  - We will define SVMs that classify such training data imperfectly.
- Third, we will see how to define nonlinear SVMs, which can define non-linear decision boundaries.

An example of a nonlinear decision boundary produced by a nonlinear SVM.



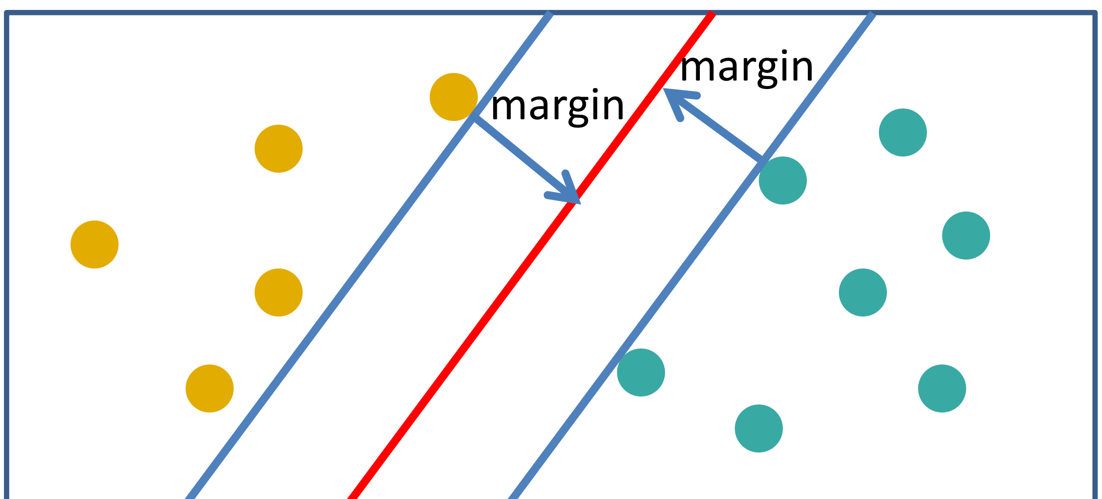
Support Vectors

- In the figure, the red line is the maximum margin decision boundary.
- One of the parallel lines touches a single orange point.
  - If that orange point moves closer to or farther from the red line, the optimal boundary changes.
  - If other orange points move, the optimal boundary does not change, **unless** those points move to the right of the blue line.



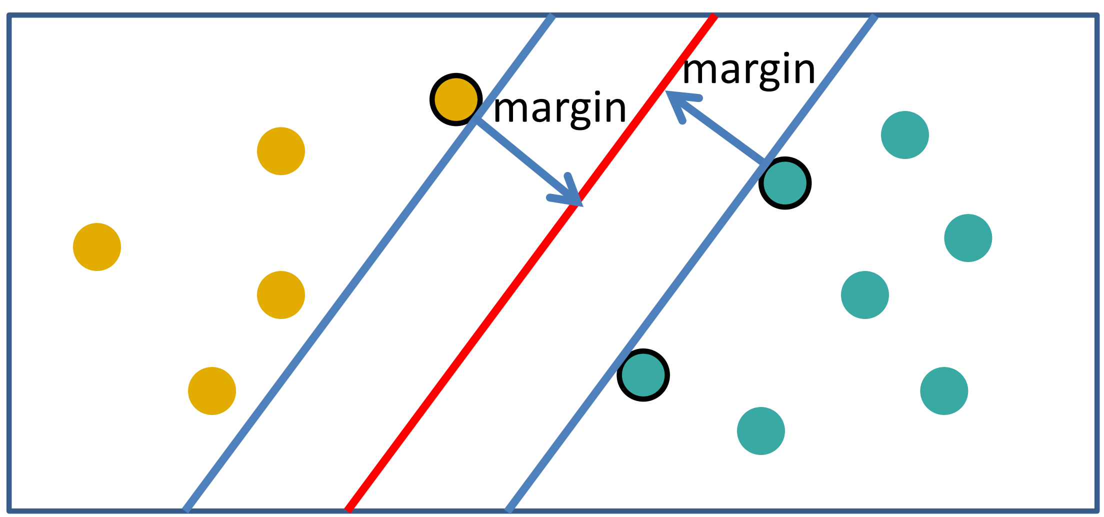
Support Vectors

- In the figure, the red line is the maximum margin decision boundary.
- One of the parallel lines touches two blue points.
  - If either of those points moves closer to or farther from the red line, the optimal boundary changes.
  - If other blue points move, the optimal boundary does not change, **unless** those points move to the left of the blue line.



Support Vectors

- In summary, in this example, the maximum margin is defined by only three points:
  - One orange point.
  - Two blue points.
- These points are called **support vectors**.
  - They are indicated by a black circle around them.



Distances to the Boundary

- The decision boundary consists of all points $\boldsymbol{x}$ that are solutions to equation: $\boldsymbol{w}^T\boldsymbol{x} + b = 0$.
  - $\boldsymbol{w}$ is a column vector of parameters (weights).
  - $\boldsymbol{x}$ is an input vector.
  - b is a scalar value (a real number).
- If $\boldsymbol{x}_n$ is a training point, its distance to the boundary is computed using this equation:

$$D(\boldsymbol{x}_n, \boldsymbol{w}) = \left| \frac{\boldsymbol{w}^T\boldsymbol{x} + b}{||\boldsymbol{w}||} \right|$$

Distances to the Boundary

- If $\boldsymbol{x}_n$ is a training point, its distance to the boundary is computed using this equation:

$$D(\boldsymbol{x}_n, \boldsymbol{w}) = \left| \frac{\boldsymbol{w}^T \boldsymbol{x}_n + b}{\|\boldsymbol{w}\|} \right|$$

- Since the training data are linearly separable, the data from each class should fall on opposite sides of the boundary.
- Suppose that $t_n = -1$ for points of one class, and $t_n = +1$ for points of the other class.
- Then, we can rewrite the distance as:

$$D(\boldsymbol{x}_n, \boldsymbol{w}) = \frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|}$$

Distances to the Boundary

- So, given a decision boundary defined $\boldsymbol{w}$ and $\boldsymbol{b}$, and given a training input $\boldsymbol{x}_n$, the distance of $\boldsymbol{x}_n$ to the boundary is:

$$D(\boldsymbol{x}_n, \boldsymbol{w}) = \frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|}$$

- If $t_n = -1$, then:
  - $\boldsymbol{w}^T \boldsymbol{x}_n + b < 0$.
  - $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) > 0$.
- If $t_n = 1$, then:
  - $\boldsymbol{w}^T \boldsymbol{x}_n + b > 0$.
  - $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) > 0$.
- So, in all cases, $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)$ is positive.

Optimization Criterion

- If $\boldsymbol{x}_n$ is a training point, its distance to the boundary is computed using this equation:

$$D(\boldsymbol{x}_n, \boldsymbol{w}) = \frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|}$$

- Therefore, the optimal boundary $\boldsymbol{w}_{\mathrm{opt}}$ is defined as:

$$(\boldsymbol{w}_{\mathrm{opt}}, b_{\mathrm{opt}}) = \operatorname{argmax}_{\boldsymbol{w},b} \left\{ \min_n \left[\frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|} \right] \right\}$$

  – In words: find the $\boldsymbol{w}$ and b that maximize the minimum distance of any training input from the boundary.

Optimization Criterion

- The optimal boundary $\boldsymbol{w}_{\text{opt}}$ is defined as:

$$(\boldsymbol{w}_{\text{opt}}, b_{\text{opt}}) = \text{argmax}_{\boldsymbol{w},b} \left\{ \min_n \left[\frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|} \right] \right\}$$

- Suppose that, for some values $\boldsymbol{w}$ and b, the decision boundary defined by $\boldsymbol{w}^T \boldsymbol{x}_n + b = 0$ misclassifies some objects.
- Can those values of $\boldsymbol{w}$ and b be selected as $\boldsymbol{w}_{\text{opt}}, b_{\text{opt}}$?

Optimization Criterion

- The optimal boundary $\boldsymbol{w}_{\text{opt}}$ is defined as:

$$(\boldsymbol{w}_{\text{opt}}, b_{\text{opt}}) = \text{argmax}_{\boldsymbol{w},b} \left\{ \min_n \left[\frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|} \right] \right\}$$

- Suppose that, for some values $\boldsymbol{w}$ and b, the decision boundary defined by $\boldsymbol{w}^T \boldsymbol{x}_n + b = 0$ misclassifies some objects.
- Can those values of $\boldsymbol{w}$ and b be selected as $\boldsymbol{w}_{\text{opt}}, b_{\text{opt}}$?
- No.
  - If some objects get misclassified, then, for some $\boldsymbol{x}_n$ it holds that $\frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|} < \mathbf{0}$.
  - Thus, for such $\boldsymbol{w}$ and b, the expression in red will be negative.
  - Since the data is linearly separable, we can find better values for $\boldsymbol{w}$ and b, for which the expression in red will be greater than 0.

Scale of $\boldsymbol{w}$

- The optimal boundary $\boldsymbol{w}_{\text{opt}}$ is defined as:

$$(\boldsymbol{w}_{\text{opt}}, b_{\text{opt}}) = \operatorname{argmax}_{\boldsymbol{w},b} \left\{ \min_n \left[\frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{||\boldsymbol{w}||} \right] \right\}$$

- Suppose that g is a real number, and $c > 0$.
- If $\boldsymbol{w}_{\text{opt}}$ and b_{opt} define an optimal boundary, then $g * \boldsymbol{w}_{\text{opt}}$ and $g * b_{\text{opt}}$ also define an optimal boundary.
- We constrain the scale of $\boldsymbol{w}_{\text{opt}}$ to a single value, by requiring that:

$$\min_n [t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)] = 1$$

Optimization Criterion

- We introduced the requirement that: $\min_n [t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)] = 1$
- Therefore, for any $\boldsymbol{x}_n$, it holds that: $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1$
- The original optimization criterion becomes:

$$(\boldsymbol{w}_{\text{opt}}, b_{\text{opt}}) = \operatorname{argmax}_{\boldsymbol{w},b} \left\{ \min_n \left[\frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|} \right] \right\} \Rightarrow$$

$$\boldsymbol{w}_{\text{opt}} = \operatorname{argmax}_{\boldsymbol{w}} \left\{ \frac{1}{\|\boldsymbol{w}\|} \right\} = \operatorname{argmin}_{\boldsymbol{w}} \{ \|\boldsymbol{w}\| \} \Rightarrow$$

$$\boldsymbol{w}_{\text{opt}} = \operatorname{argmin}_{\boldsymbol{w}} \left\{ \frac{1}{2} \|\boldsymbol{w}\|^2 \right\}$$

These are equivalent formulations. The textbook uses the last one because it simplifies subsequent calculations.

Constrained Optimization

- Summarizing the previous slides, we want to find:

$$\boldsymbol{w}_{\text{opt}} = \operatorname{argmin}_{\boldsymbol{w}} \left\{ \frac{1}{2} \|\boldsymbol{w}\|^{\mathbf{2}} \right\}$$

subject to the following constraints:

$$\forall n \in \{1, \ldots, N\}, t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1$$

- This is a different optimization problem than what we have seen before.
- We need to minimize a quantity **while satisfying a set of inequalities**.
- This type of problem is a <u>**constrained optimization problem**</u>.

Quadratic Programming

- Our constrained optimization problem can be solved using a method called **<u>quadratic programming</u>**.
- Describing **<u>quadratic programming</u>** in depth is outside the scope of this course.
- Our goal is simply to understand how to use quadratic programming as a black box, to solve our optimization problem.
  - This way, you can use any quadratic programming toolkit (Matlab includes one).

Quadratic Programming

- The quadratic programming problem is defined as follows:
- Inputs:
  - $\boldsymbol{s}$: an R-dimensional column vector.
  - $\boldsymbol{Q}$: an $R \times R$-dimensional symmetric matrix.
  - $\boldsymbol{H}$: an $Q \times R$-dimensional symmetric matrix.
  - $\boldsymbol{z}$: an Q-dimensional column vector.
- Output:
  - $\boldsymbol{u}_{\text{opt}}$: an R-dimensional column vector, such that:

$$\boldsymbol{u}_{\text{opt}} = \text{argmin}_u \left\{ \frac{1}{2} \boldsymbol{u}^T \boldsymbol{Q} \boldsymbol{u} + \boldsymbol{s}^T \boldsymbol{u} \right\}$$

subject to constraint: $\boldsymbol{H}\boldsymbol{u} \leq \boldsymbol{z}$

Quadratic Programming for SVMs

- Quadratic Programming:

$\boldsymbol{u}_{\text{opt}} = \text{argmin}_u \left\{\frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T\boldsymbol{u}\right\}$ subject to constraint: $\boldsymbol{H}\boldsymbol{u} \leq \boldsymbol{z}$

- SVM goal: $\boldsymbol{w}_{\text{opt}} = \text{argmin}_{\boldsymbol{w}} \left\{\frac{1}{2}\|\boldsymbol{w}\|^{\mathbf{2}}\right\}$
  subject to constraints: $\forall n \in \{1, \dots, N\}, t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1$

- We need to define appropriate values of $\boldsymbol{Q}$, $\boldsymbol{s}$, $\boldsymbol{H}$, $\boldsymbol{z}$, $\boldsymbol{u}$ so that quadratic programming computes $\boldsymbol{w}_{\text{opt}}$ and b_{opt}.

Quadratic Programming for SVMs

- Quadratic Programming constraint: $\boldsymbol{H}\boldsymbol{u} \leq \boldsymbol{z}$
- SVM constraints: $\forall n \in \{1, \ldots, N\}, t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 \Rightarrow$
  $\forall n \in \{1, \ldots, N\}, -t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \leq -1$

Quadratic Programming for SVMs

- SVM constraints: $\forall n \in \{1, \dots, N\}, -t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \leq -1$
- Define: $\boldsymbol{H} = \begin{bmatrix} -t_1, -t_1 x_{11}, \dots, -t_1 x_{1D} \\ -t_2, -t_2 x_{21}, \dots, -t_1 x_{2D} \\ \dots \\ -t_N, -t_N x_{N1}, \dots, -t_N x_{ND} \end{bmatrix}$, $\boldsymbol{u} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \dots \\ w_D \end{bmatrix}$, $\boldsymbol{z} = \begin{bmatrix} -1 \\ -1 \\ \dots \\ -1 \end{bmatrix}_N$
  - Matrix $\boldsymbol{H}$ is $N \times (D+1)$, vector $\boldsymbol{u}$ has $(D+1)$ rows, vector $\boldsymbol{z}$ has N rows.
- $\boldsymbol{Hu} = \begin{bmatrix} -t_1 b - t_1\, x_{11} w_1 - \cdots - t_1 x_{1D} w_1 \\ \dots \\ -t_N b - t_N\, x_{N1} w_1 - \cdots - t_1 x_{ND} w_1 \end{bmatrix} = \begin{bmatrix} -t_1(b + \boldsymbol{w}^T \boldsymbol{x}_1) \\ \dots \\ -t_N(b + \boldsymbol{w}^T \boldsymbol{x}_N) \end{bmatrix}$
- The n-th row of $\boldsymbol{Hu}$ is $-t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)$, which should be ≤ -1.
- SVM constraint → quadratic programming constraint $\boldsymbol{Hu} \leq \boldsymbol{z}$.

Quadratic Programming for SVMs

- Quadratic programming: $\boldsymbol{u}_{\text{opt}} = \text{argmin}_{u} \left\{ \frac{1}{2} \boldsymbol{u}^T \boldsymbol{Q} \boldsymbol{u} + \boldsymbol{s}^T \boldsymbol{u} \right\}$
- SVM: $\boldsymbol{w}_{\text{opt}} = \text{argmin}_{\boldsymbol{w}} \left\{ \frac{1}{2} \|\boldsymbol{w}\|^{\mathbf{2}} \right\}$

$\boldsymbol{u}$ already defined in the previous slides

$\boldsymbol{s}$: $(D+1)$-dimensional column vector of zeros.

- Define: $\boldsymbol{Q} = \begin{bmatrix} 0,0,0,\ldots,0 \\ 0,1,0,\ldots,0 \\ 0,0,1,\ldots,0 \\ \cdots \\ 0,0,0,\ldots,1 \end{bmatrix}$, $\quad \boldsymbol{u} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \cdots \\ w_D \end{bmatrix}$, $\quad \boldsymbol{s} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}_{D+1}$
  - $\boldsymbol{Q}$ is like the $(D+1) \times (D+1)$ identity matrix, except that $Q_{11} = 0$.
- Then, $\frac{1}{2} \boldsymbol{u}^T \boldsymbol{Q} \boldsymbol{u} + \boldsymbol{s}^T \boldsymbol{u} = \frac{1}{2} \|\boldsymbol{w}\|^{\mathbf{2}}$.

Quadratic Programming for SVMs

- Quadratic programming: $\boldsymbol{u}_{\text{opt}} = \text{argmin}_u \left\{\frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T\boldsymbol{u}\right\}$
- SVM: $\boldsymbol{w}_{\text{opt}} = \text{argmin}_{\boldsymbol{w}} \left\{\frac{1}{2}\|\boldsymbol{w}\|^{\mathbf{2}}\right\}$
- Alternative definitions that would **NOT** work:
- Define: $\boldsymbol{Q} = \begin{bmatrix} 1, 0, \dots, 0 \\ \dots \\ 0, 0, \dots, 1 \end{bmatrix}, \qquad \boldsymbol{u} = \begin{bmatrix} w_1 \\ \dots \\ w_D \end{bmatrix}, \qquad \boldsymbol{s} = \begin{bmatrix} 0 \\ \dots \\ 0 \end{bmatrix}_{\boldsymbol{D}}$
  - $\boldsymbol{Q}$ is the $D \times D$ identity matrix, $\boldsymbol{u} = \boldsymbol{w}$, $\boldsymbol{s}$ is the D-dimensional zero vector.
- It still holds that $\frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T\boldsymbol{u} = \frac{1}{2}\|\boldsymbol{w}\|^{\mathbf{2}}$.
- **Why would these definitions not work?**

Quadratic Programming for SVMs

- Quadratic programming: $\boldsymbol{u}_{\text{opt}} = \operatorname{argmin}_u \left\{\frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T\boldsymbol{u}\right\}$
- SVM: $\boldsymbol{w}_{\text{opt}} = \operatorname{argmin}_{\boldsymbol{w}} \left\{\frac{1}{2}\|\boldsymbol{w}\|^{\mathbf{2}}\right\}$
- Alternative definitions that would **NOT** work:
- Define: $\boldsymbol{Q} = \begin{bmatrix} 1, 0, \dots, 0 \\ \dots \\ 0, 0, \dots, 1 \end{bmatrix}$, $\quad \boldsymbol{u} = \begin{bmatrix} w_1 \\ \dots \\ w_D \end{bmatrix}$, $\quad \boldsymbol{s} = \begin{bmatrix} 0 \\ \dots \\ 0 \end{bmatrix}_{\boldsymbol{D}}$
  - $\boldsymbol{Q}$ is the $D \times D$ identity matrix, $\boldsymbol{u} = \boldsymbol{w}$, $\boldsymbol{s}$ is the D-dimensional zero vector.
- It still holds that $\frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T\boldsymbol{u} = \frac{1}{2}\|\boldsymbol{w}\|^{\mathbf{2}}$.
- **Why would these definitions not work?**
- Vector $\boldsymbol{u}$ must also make $\boldsymbol{H}\boldsymbol{u} \leq \boldsymbol{z}$ match the SVM constraints.
  - With this definition of $\boldsymbol{u}$, no appropriate $\boldsymbol{H}$ and $\boldsymbol{z}$ can be found.

Quadratic Programming for SVMs

- Quadratic programming: $\boldsymbol{u}_{\text{opt}} = \text{argmin}_u \left\{\frac{1}{2}\boldsymbol{u}^T \boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T\boldsymbol{u}\right\}$ subject to constraint: $\boldsymbol{H}\boldsymbol{u} \leq \boldsymbol{z}$
- SVM goal: $\boldsymbol{w}_{\text{opt}} = \text{argmin}_{\boldsymbol{w}} \left\{\frac{1}{2}\|\boldsymbol{w}\|^2\right\}$ subject to constraints: $\forall n \in \{1, \dots, N\}, t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1$
- Task: define $\boldsymbol{Q}, \boldsymbol{s}, \boldsymbol{H}, \boldsymbol{z}, \boldsymbol{u}$ so that quadratic programming computes $\boldsymbol{w}_{\text{opt}}$ and b_{opt}.

$$\boldsymbol{Q} = \begin{bmatrix} 0,0,0,\dots,0 \\ 0,1,0,\dots,0 \\ 0,0,1,\dots,0 \\ \cdots \\ 0,0,0,\dots,1 \end{bmatrix}, \boldsymbol{s} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}, \boldsymbol{H} = \begin{bmatrix} -t_1, -t_1x_{11}, \dots, -t_1x_{1D} \\ -t_2, -t_2x_{21}, \dots, -t_1x_{2D} \\ \cdots \\ -t_N, -t_Nx_{N1}, \dots, -t_Nx_{ND} \end{bmatrix}, \boldsymbol{z} = \begin{bmatrix} -1 \\ -1 \\ \cdots \\ -1 \end{bmatrix}, \boldsymbol{u} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \cdots \\ w_D \end{bmatrix}$$

Like $(D+1) \times (D+1)$ identity matrix, except that $Q_{11} = 0$

$D+1$ rows

N rows, $D+1$ columns

N rows

$D+1$ rows

Quadratic Programming for SVMs

- Quadratic programming: $\boldsymbol{u}_{\text{opt}} = \text{argmin}_u \left\{\frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T\boldsymbol{u}\right\}$ subject to constraint: $\boldsymbol{H}\boldsymbol{u} \leq \boldsymbol{z}$
- SVM goal: $\boldsymbol{w}_{\text{opt}} = \text{argmin}_{\boldsymbol{w}} \left\{\frac{1}{2}\|\boldsymbol{w}\|^{\boldsymbol{2}}\right\}$
subject to constraints: $\forall n \in \{1, \dots, N\}, t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1$
- Task: define $\boldsymbol{Q}, \boldsymbol{s}, \boldsymbol{H}, \boldsymbol{z}, \boldsymbol{u}$ so that quadratic programming computes $\boldsymbol{w}_{\text{opt}}$ and b_{opt}.

$$\boldsymbol{Q} = \begin{bmatrix} 0, 0, 0, \dots, 0 \\ 0, 1, 0, \dots, 0 \\ 0, 0, 1, \dots, 0 \\ \dots \\ 0, 0, 0, \dots, 1 \end{bmatrix}, \boldsymbol{s} = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \end{bmatrix}, \boldsymbol{H} = \begin{bmatrix} -t_1, -t_1 x_{11}, \dots, -t_1 x_{1D} \\ -t_2, -t_2 x_{21}, \dots, -t_1 x_{2D} \\ \dots \\ -t_N, -t_N x_{N1}, \dots, -t_N x_{ND} \end{bmatrix}, \boldsymbol{z} = \begin{bmatrix} -1 \\ -1 \\ \dots \\ -1 \end{bmatrix}, \boldsymbol{u} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \dots \\ w_D \end{bmatrix}$$

- Quadratic programming takes as inputs $\boldsymbol{Q}, \boldsymbol{s}, \boldsymbol{H}, \boldsymbol{z}$, and outputs $\boldsymbol{u}_{\text{opt}}$, from which we get the $\boldsymbol{w}_{\text{opt}}, b_{\text{opt}}$ values for our SVM.

Quadratic Programming for SVMs

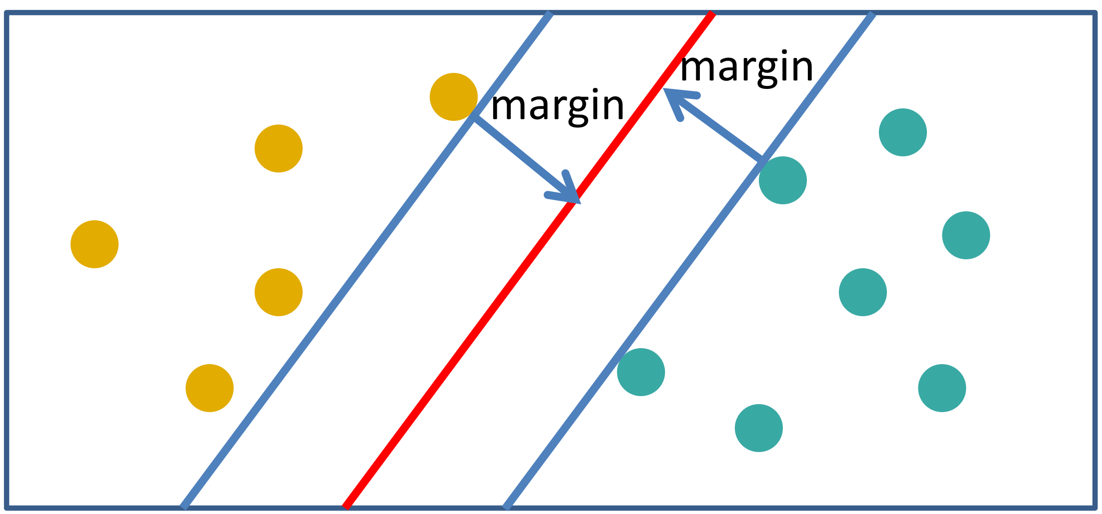
- Quadratic programming: $\boldsymbol{u}_{\text{opt}} = \text{argmin}_u \left\{\frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T\boldsymbol{u}\right\}$ subject to constraint: $\boldsymbol{H}\boldsymbol{u} \leq \boldsymbol{z}$
- SVM goal: $\boldsymbol{w}_{\text{opt}} = \text{argmin}_{\boldsymbol{w}} \left\{\frac{1}{2}\|\boldsymbol{w}\|^{\mathbf{2}}\right\}$
  subject to constraints: $\forall n \in \{1, \dots, N\}, t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1$
- Task: define $\boldsymbol{Q}, \boldsymbol{s}, \boldsymbol{H}, \boldsymbol{z}, \boldsymbol{u}$ so that quadratic programming computes $\boldsymbol{w}_{\text{opt}}$ and b_{opt}.

$$\boldsymbol{Q} = \begin{bmatrix} 0, 0, 0, \dots, 0 \\ 0, 1, 0, \dots, 0 \\ 0, 0, 1, \dots, 0 \\ \cdots \\ 0, 0, 0, \dots, 1 \end{bmatrix}, \boldsymbol{s} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}, \boldsymbol{H} = \begin{bmatrix} -t_1, -t_1x_{11}, \dots, -t_1x_{1D} \\ -t_2, -t_2x_{21}, \dots, -t_1x_{2D} \\ \cdots \\ -t_N, -t_Nx_{N1}, \dots, -t_Nx_{ND} \end{bmatrix}, \boldsymbol{z} = \begin{bmatrix} -1 \\ -1 \\ \cdots \\ -1 \end{bmatrix}, \boldsymbol{u} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \cdots \\ w_D \end{bmatrix}$$

- $\boldsymbol{w}_{\text{opt}}$ is the vector of values at dimensions 2, …, $D + 1$ of $\boldsymbol{u}_{\text{opt}}$.
- b_{opt} is the value at dimension 1 of $\boldsymbol{u}_{\text{opt}}$.

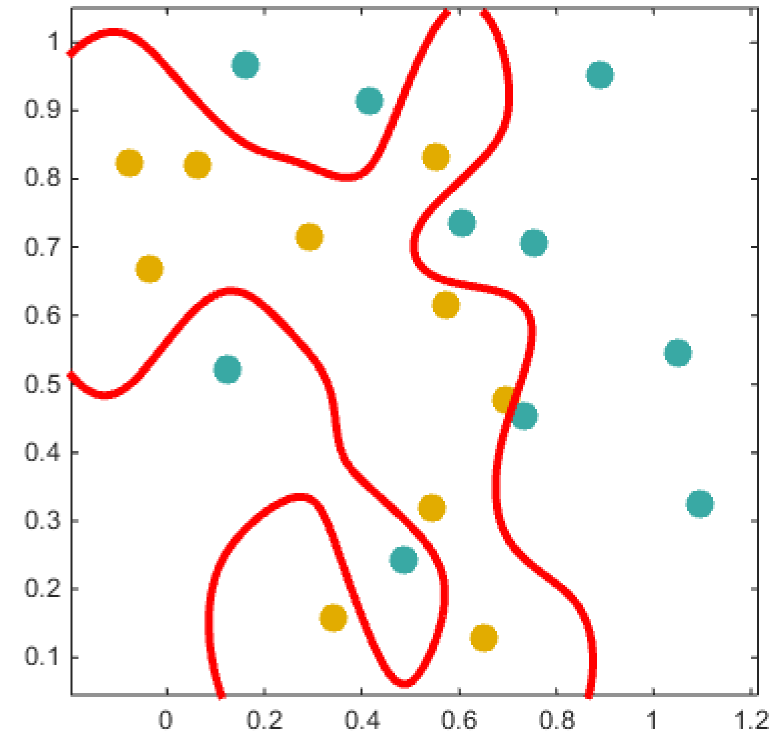
Solving the Same Problem, Again

- So far, we have solved the problem of defining an SVM (i.e., defining $\boldsymbol{w}_{\text{opt}}$ and b_{opt}), so as to maximize the margin between linearly separable data.
- If this were all that SVMs can do, SVMs would not be that important.
  - Linearly separable data are a rare case, and a very easy case to deal with.



Solving the Same Problem, Again

- So far, we have solved the problem of defining an SVM (i.e., defining $\boldsymbol{w}_{\text{opt}}$ and b_{opt}), so as to maximize the margin between linearly separable data.
- We will see two extensions that make SVMs much more powerful.
  - The extensions will allow SVMs to define highly non-linear decision boundaries, as in this figure.
- However, first we need to solve the same problem again.
  - Maximize the margin between linearly separable data.



- We will get a more complicated solution, but that solution will be easier to improve upon.

Lagrange Multipliers

- Our new solutions are derived using **Lagrange multipliers**.
- Here is a quick review from multivariate calculus.
- Let $\boldsymbol{x}$ be a D-dimensional vector.
- Let $f(\boldsymbol{x})$ and $g(\boldsymbol{x})$ be functions from $\mathbb{R}^D$ to $\mathbb{R}$.
  - Functions f and g map D-dimensional vectors to real numbers.
- Suppose that we want to minimize $f(\boldsymbol{x})$, subject to the constraint that $g(\boldsymbol{x}) \geq 0$.
- Then, we can solve this problem using a **Lagrange multiplier** to define a **Lagrangian function**.

Lagrange Multipliers

- To minimize $f(\boldsymbol{x})$, subject to the constraint: $g(\boldsymbol{x}) \geq 0$:
- We define the **Lagrangian function**: $L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) - \lambda g(\boldsymbol{x})$
  - λ is called a **Lagrange multiplier**, $\lambda \geq 0$.
- We find $\boldsymbol{x}_{\text{opt}} = \text{argmin}_{\boldsymbol{x}}\{L(\boldsymbol{x}, \lambda)\}$, and a corresponding value for λ, subject to the following constraints:

  1. $g(\boldsymbol{x}) \geq 0$
  2. $\lambda \geq 0$
  3. $\lambda g(\boldsymbol{x}) = 0$

- If $g(\boldsymbol{x}_{\text{opt}}) > 0$, the third constraint implies that $\lambda = 0$.
  - Then, the constraint $g(\boldsymbol{x}) \geq 0$ is called **inactive**.
- If $g(\boldsymbol{x}_{\text{opt}}) = 0$, then $\lambda > 0$.
  - Then, constraint $g(\boldsymbol{x}) \geq 0$ is called **active**.

Multiple Constraints

- Suppose that we have N constraints:

$$\forall n \in \{1, \dots, N\}, \qquad g_n(\boldsymbol{x}) \geq 0$$

- We want to minimize $f(\boldsymbol{x})$, subject to those N constraints.
- Define vector $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_N)$.
- Define the **Lagrangian function** as:

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \sum_{n=1}^{N} \{\lambda_n g_n(\boldsymbol{x})\}$$

- We find $\boldsymbol{x}_{\text{opt}} = \text{argmin}_{\boldsymbol{x}}\{L(\boldsymbol{x}, \boldsymbol{\lambda})\}$, and a value for $\boldsymbol{\lambda}$, subject to:
  - $\forall n, \;\; g_n(\boldsymbol{x}) \geq 0$
  - $\forall n, \;\; \lambda_n \geq 0$
  - $\forall n, \;\; \lambda_n g_n(\boldsymbol{x}) = 0$

Lagrange Dual Problems

- We have N constraints: $\forall n \in \{1, \dots, N\},\ g_n(\boldsymbol{x}) \geq 0$
- We want to minimize $f(\boldsymbol{x})$, subject to those N constraints.
- Under some conditions (which are satisfied in our SVM problem), we can solve an alternative **dual** problem:
- Define the **Lagrangian function** as before:

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \sum_{n=1}^{N} \{\lambda_n g_n(\boldsymbol{x})\}$$

- We find $\boldsymbol{x}_{\text{opt}}$, and the best value for $\boldsymbol{\lambda}$, denoted as $\boldsymbol{\lambda}_{\text{opt}}$, by solving:

$$\boldsymbol{\lambda}_{\text{opt}} = \underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \left\{ \min_{\boldsymbol{x}} \{L(\boldsymbol{x}, \boldsymbol{\lambda})\} \right\}$$
$$\boldsymbol{x}_{\text{opt}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \{L(\boldsymbol{x}, \boldsymbol{\lambda}_{\text{opt}})\}$$

subject to constraints: $\lambda_n \geq \mathbf{0}$

Lagrange Dual Problems

- Lagrangian **dual** problem: Solve:

$$\boldsymbol{\lambda}_{\text{opt}} = \underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \left\{ \min_{\boldsymbol{x}} \{ L(\boldsymbol{x}, \boldsymbol{\lambda}) \} \right\}$$

$$\boldsymbol{x}_{\text{opt}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \{ L(\boldsymbol{x}, \boldsymbol{\lambda}_{\text{opt}}) \}$$

subject to constraints: $\lambda_n \geq \mathbf{0}$

- This dual problem formulation will be used in training SVMs.
- The key thing to remember is:
  - We minimize the Lagrangian L with respect to $\boldsymbol{x}$.
  - We maximize L with respect to the Lagrange multipliers λ_n.

Lagrange Multipliers and SVMs

- SVM goal: $\boldsymbol{w}_{\text{opt}} = \text{argmin}_{\boldsymbol{w}} \left\{\frac{1}{2}\|\boldsymbol{w}\|^{\mathbf{2}}\right\}$ subject to constraints:

$$\forall n \in \{1, \ldots, N\}, \quad t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1$$

- To make the constraints more amenable to Lagrange multipliers, we rewrite them as:

$$\forall n \in \{1, \ldots, N\}, \quad t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) - 1 \geq 0$$

- Define $\boldsymbol{a} = (a_1, \ldots, a_N)$ to be a vector of N **Lagrange multipliers**.
- Define the **Lagrangian function**:

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{n=1}^{N} \{a_n(t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) - 1)\}$$

- Remember from the previous slides, we minimize L with respect to $\boldsymbol{w}, b$, and maximize L with respect to $\boldsymbol{a}$.

Lagrange Multipliers and SVMs

- The $\boldsymbol{w}$ and b that minimize $L(\boldsymbol{w}, b, \boldsymbol{a})$ must satisfy:

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0, \ \frac{\partial L}{\partial b} = 0.$$

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0 \Rightarrow \frac{\partial \left\{\frac{1}{2}\|\boldsymbol{w}\|^{2} - \sum_{n=1}^{N}\{a_n(t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - 1)\}\right\}}{\partial \boldsymbol{w}} = 0$$

$$\Rightarrow \boldsymbol{w} - \sum_{n=1}^{N}\{a_n t_n \boldsymbol{x}_n\} = 0$$

$$\Rightarrow \boldsymbol{w} = \sum_{n=1}^{N}\{a_n t_n \boldsymbol{x}_n\}$$

Lagrange Multipliers and SVMs

- The $\boldsymbol{w}$ and b that minimize $L(\boldsymbol{w}, b, \boldsymbol{a})$ must satisfy:

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0, \ \frac{\partial L}{\partial b} = 0.$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \frac{\partial \left\{ \frac{1}{2} \|\boldsymbol{w}\|^{\mathbf{2}} - \sum_{n=1}^{N} \{a_n(t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) - 1)\} \right\}}{\partial b} = 0$$

$$\Rightarrow \sum_{n=1}^{N} \{a_n t_n\} = 0$$

Lagrange Multipliers and SVMs

- Our Lagrangian function is:

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{n=1}^{N}\{a_n(t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - 1)\}$$

- We showed that $\boldsymbol{w} = \sum_{n=1}^{N}(a_n t_n \boldsymbol{x}_n)$. Using that, we get:

$$\frac{1}{2}\|\boldsymbol{w}\|^2 = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} = \frac{1}{2}\left(\sum_{n=1}^{N}\left(a_n t_n (\boldsymbol{x}_n)^T\right)\right)\left(\sum_{n=1}^{N}(a_n t_n \boldsymbol{x}_n)\right)$$

$$\Rightarrow \frac{1}{2}\|\boldsymbol{w}\|^2 = \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}\left\{a_n a_m t_n t_m (\boldsymbol{x}_n)^T \boldsymbol{x}_m\right\}$$

Lagrange Multipliers and SVMs

- We showed that:

$$\Rightarrow \frac{1}{2}\|\boldsymbol{w}\|^2 = \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}\{a_n a_m t_n t_m (\boldsymbol{x}_n)^T\, \boldsymbol{x}_m\}$$

- Define an $N \times N$ matrix $\boldsymbol{Q}$ such that $Q_{mn} = t_n t_m (\boldsymbol{x}_n)^T\, \boldsymbol{x}_m$.
- Remember that we have defined $\boldsymbol{a} = (a_1, \ldots, a_N)$.
- Then, it follows that:

$$\frac{1}{2}\|\boldsymbol{w}\|^2 = \frac{1}{2}\boldsymbol{a}^T\, \boldsymbol{Q}\boldsymbol{a}$$

Lagrange Multipliers and SVMs

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\,\boldsymbol{Q}\boldsymbol{a} - \sum_{n=1}^{N}\{a_n(t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - 1)\}$$

- We showed that $\sum_{n=1}^{N}\{a_n t_n\} = 0$. Using that, we get:

$$\sum_{n=1}^{N}\{a_n(t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - 1)\}$$

We have shown before that the red part equals 0.

$$= \sum_{n=1}^{N}\{a_n t_n \boldsymbol{w}^T\boldsymbol{x}_n\} + b\sum_{n=1}^{N}\{a_n t_n\} - \sum_{n=1}^{N} a_n$$

$$= \sum_{n=1}^{N}\{a_n t_n \boldsymbol{w}^T\boldsymbol{x}_n\} - \sum_{n=1}^{N} a_n$$

Lagrange Multipliers and SVMs

$$L(\boldsymbol{w}, \boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\,\boldsymbol{Q}\boldsymbol{a} - \sum_{n=1}^{N}\{a_n t_n \boldsymbol{w}^T \boldsymbol{x}_n\} + \sum_{n=1}^{N} a_n$$

- Function L now does not depend on b.
- We simplify more, using again the fact that $\boldsymbol{w} = \sum_{n=1}^{N}(a_n t_n \boldsymbol{x}_n)$:

$$\sum_{n=1}^{N}\{a_n t_n \boldsymbol{w}^T \boldsymbol{x}_n\} = \sum_{n=1}^{N}\left\{a_n t_n \left(\sum_{m=1}^{N}(a_m t_m (\boldsymbol{x}_m)^T)\right)\boldsymbol{x}_n\right\}$$

$$= \sum_{n=1}^{N}\sum_{m=1}^{N}\{a_n a_m t_n t_m (\boldsymbol{x}_m)^T\,\boldsymbol{x}_n\} = \boldsymbol{a}^T\,\boldsymbol{Q}\boldsymbol{a}$$

Lagrange Multipliers and SVMs

$$L(\boldsymbol{w}, \boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T \boldsymbol{Q}\boldsymbol{a} - \boldsymbol{a}^T \boldsymbol{Q}\boldsymbol{a} + \sum_{n=1}^{N} a_n = \sum_{n=1}^{N} a_n - \frac{1}{2}\boldsymbol{a}^T \boldsymbol{Q}\boldsymbol{a}$$

- Function L now does not depend on w anymore.
- We can rewrite L as a function whose only input is $\boldsymbol{a}$:

$$L(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T \boldsymbol{Q}\boldsymbol{a} - \boldsymbol{a}^T \boldsymbol{Q}\boldsymbol{a} + \sum_{n=1}^{N} a_n = \sum_{n=1}^{N} a_n - \frac{1}{2}\boldsymbol{a}^T \boldsymbol{Q}\boldsymbol{a}$$

- Remember, we want to **maximize** $L(\boldsymbol{a})$ with respect to $\boldsymbol{a}$.

Lagrange Multipliers and SVMs

- By combining the results from the last few slides, our optimization problem becomes:

  Maximize

$$L(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \boldsymbol{a}^T \boldsymbol{Q} \boldsymbol{a}$$

  subject to these constraints:

$$a_n \geq 0$$

$$\sum_{n=1}^{N} \{a_n t_n\} = 0$$

Lagrange Multipliers and SVMs

$$L(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \boldsymbol{a}^T \boldsymbol{Q} \boldsymbol{a}$$

- We want to maximize $L(\boldsymbol{a})$ subject to some constraints.
- Therefore, we want to find an $\boldsymbol{a}_{\text{opt}}$ such that:

$$\boldsymbol{a}_{\text{opt}} = \underset{\boldsymbol{a}}{\operatorname{argmax}} \left\{ \sum_{n=1}^{N} a_n - \frac{1}{2} \boldsymbol{a}^T \boldsymbol{Q} \boldsymbol{a} \right\} = \underset{\boldsymbol{a}}{\operatorname{argmin}} \left\{ \frac{1}{2} \boldsymbol{a}^T \boldsymbol{Q} \boldsymbol{a} - \sum_{n=1}^{N} a_n \right\}$$

subject to those constraints.

SVM Optimization Problem

- Our SVM optimization problem now is to find an $\boldsymbol{a}_{\text{opt}}$ such that:

$$\boldsymbol{a}_{\text{opt}} = \underset{\boldsymbol{a}}{\operatorname{argmin}} \left\{ \frac{1}{2} \boldsymbol{a}^T \, \boldsymbol{Q} \boldsymbol{a} \; - \sum_{n=1}^{N} a_n \right\}$$

subject to these constraints:

$$a_n \geq 0$$

$$\sum_{n=1}^{N} \{a_n t_n\} = 0$$

This problem can be solved again using quadratic programming.

Using Quadratic Programming

- Quadratic programming: $\boldsymbol{u}_{\text{opt}} = \text{argmin}_u \left\{\frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T\boldsymbol{u}\right\}$
  subject to constraint: $\boldsymbol{H}\boldsymbol{u} \leq \boldsymbol{z}$
- SVM problem: find $\boldsymbol{a}_{\text{opt}} = \underset{\boldsymbol{a}}{\text{argmin}} \left\{\frac{1}{2}\boldsymbol{a}^T\ \boldsymbol{Q}\boldsymbol{a}\ - \sum_{n=1}^{N} a_n\right\}$
  subject to constraints:

$$a_n \geq 0, \qquad \sum_{n=1}^{N}\{a_n t_n\} = 0$$

- Again, we must find values for $\boldsymbol{Q}, \boldsymbol{s}, \boldsymbol{H}, \boldsymbol{z}$ that convert the SVM problem into a quadratic programming problem.
- Note that we already have a matrix $\boldsymbol{Q}$ in the Lagrangian. It is an $N \times N$ matrix $\boldsymbol{Q}$ such that $Q_{mn} = t_n t_m (\boldsymbol{x}_n)^T\ \boldsymbol{x}_n$.
- We will use the same $\boldsymbol{Q}$ for quadratic programming.

Using Quadratic Programming

- Quadratic programming: $\boldsymbol{u}_{\text{opt}} = \text{argmin}_u \left\{\frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T\boldsymbol{u}\right\}$
  subject to constraint: $\boldsymbol{H}\boldsymbol{u} \leq \boldsymbol{z}$
- SVM problem: find $\boldsymbol{a}_{\text{opt}} = \underset{\boldsymbol{a}}{\text{argmin}}\left\{\frac{1}{2}\boldsymbol{a}^T\ \boldsymbol{Q}\boldsymbol{a}\ - \sum_{n=1}^{N} a_n\right\}$
  subject to constraints: $a_n \geq 0, \quad \sum_{n=1}^{N}\{a_n t_n\} = 0$
- Define: $\boldsymbol{u} = \boldsymbol{a}$, and define N-dimensional vector $\boldsymbol{s} = \begin{bmatrix} -1 \\ -1 \\ \cdots \\ -1 \end{bmatrix}_N$.
- Then, $\frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T\boldsymbol{u} = \frac{1}{2}\boldsymbol{a}^T\ \boldsymbol{Q}\boldsymbol{a}\ - \sum_{n=1}^{N} a_n$
- We have mapped the SVM minimization goal of finding $\boldsymbol{a}_{\text{opt}}$ to the quadratic programming minimization goal.

Using Quadratic Programming

- Quadratic programming constraint: $\boldsymbol{Hu} \leq \boldsymbol{z}$
- SVM problem constraints: $a_n \geq 0, \quad \sum_{n=1}^{N}\{a_n t_n\} = 0$
- Define: $\boldsymbol{H} = \begin{bmatrix} -1, & 0, & 0, & \ldots, & 0 \\ 0, & -1, & 0, & \ldots, & 0 \\ & & \cdots & & \\ 0, & 0, & 0, & \ldots, & -1 \\ t_1, & t_2, & t_3, & \ldots, & t_n \\ -t_1, & -t_2, & -t_3, & \ldots, & -t_n \end{bmatrix}_{(N+2)\times \boldsymbol{N}}$, $\boldsymbol{z} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}_{N+2}$

$\boldsymbol{z}$: $(N+2)$-dimensional column vector of zeros.

- $\boldsymbol{H}$ has size $(N+2) \times \boldsymbol{N}$. The first N rows of $\boldsymbol{H}$ are the negation of the $N \times N$ identity matrix.
- Row $N+1$ of $\boldsymbol{H}$ is the transpose of vector $\boldsymbol{t}$ of target outputs.
- Row $N+2$ of $\boldsymbol{H}$ is the negation of the previous row.

Using Quadratic Programming

- Quadratic programming constraint: $\boldsymbol{Hu} \leq \boldsymbol{z}$
- SVM problem constraints: $a_n \geq 0, \quad \sum_{n=1}^{N}\{a_n t_n\} = 0$
- Define: $\boldsymbol{H} = \begin{bmatrix} -1, & 0, & 0, & \dots, & 0 \\ 0, & -1, & 0, & \dots, & 0 \\ & & \cdots & & \\ 0, & 0, & 0, & \dots, & -1 \\ t_1, & t_2, & t_3, & \dots, & t_n \\ -t_1, & -t_2, & -t_3, & \dots, & -t_n \end{bmatrix}_{(N+2)\times N}$, $\boldsymbol{z} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}_{N+2}$

$\boldsymbol{z}$: $(N + 2)$-dimensional column vector of zeros.

- Since we defined $\boldsymbol{u} = \boldsymbol{a}$:
  - For $n \leq N$, the n-th row of $\boldsymbol{Hu}$ equals a_n.
  - For $n \leq N$, the n-th row of $\boldsymbol{Hu}$ and $\boldsymbol{z}$ specifies that $-a_n \leq 0 \Rightarrow a_n \geq 0$.
  - The $(n + 1)$-th row of $\boldsymbol{Hu}$ and $\boldsymbol{z}$ specifies that $\sum_{n=1}^{N}\{a_n t_n\} \leq 0$.
  - The $(n + 2)$-th row of $\boldsymbol{Hu}$ and $\boldsymbol{z}$ specifies that $\sum_{n=1}^{N}\{a_n t_n\} \geq 0$.

Using Quadratic Programming

- Quadratic programming constraint: $\boldsymbol{Hu} \leq \boldsymbol{z}$
- SVM problem constraints: $a_n \geq 0, \quad \sum_{n=1}^{N}\{a_n t_n\} = 0$
- Define: $\boldsymbol{H} = \begin{bmatrix} -1, & 0, & 0, & \dots, & 0 \\ 0, & -1, & 0, & \dots, & 0 \\ & & \dots & & \\ 0, & 0, & 0, & \dots, & -1 \\ t_1, & t_2, & t_3, & \dots, & t_n \\ -t_1, & -t_2, & -t_3, & \dots, & -t_n \end{bmatrix}_{(N+2)\times N}$ $, \boldsymbol{z} = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \end{bmatrix}_{N+2}$

$\boldsymbol{z}$: $(N+2)$-dimensional column vector of zeros.

- Since we defined $\boldsymbol{u} = a$:
  - For $n \leq N$, the n-th row of $\boldsymbol{Hu}$ and $\boldsymbol{z}$ specifies that $a_n \geq 0$.
  - The last two rows of $\boldsymbol{Hu}$ and $\boldsymbol{z}$ specify that $\sum_{n=1}^{N}\{a_n t_n\} = 0$.
- We have mapped the SVM constraints to the quadratic programming constraint $\boldsymbol{Hu} \leq \boldsymbol{z}$.

Interpretation of the Solution

- Quadratic programming, given the inputs $\boldsymbol{Q}, \boldsymbol{s}, \boldsymbol{H}, \boldsymbol{z}$ defined in the previous slides, outputs the optimal value for vector $\boldsymbol{a}$.
- We used this **Lagrangian function**:

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{n=1}^{N}\{a_n(t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - 1)\}$$

- Remember, when we find $\boldsymbol{x}_{\text{opt}}$ to minimize any Lagrangian

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \sum_{n=1}^{N}\{\lambda_n g_n(\boldsymbol{x})\}$$

one of the constraints we enforce is that $\forall n,\ \ \lambda_n g_n(\boldsymbol{x}) = 0$.

- In the SVM case, this means: $\forall n,\ \ a_n(t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - 1) = 0$

Interpretation of the Solution

- In the SVM case, it holds that:

$$\forall n, \qquad a_n(t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) - 1) = 0$$

- What does this mean?
- Mathematically, $\forall n$:
  - Either $a_n = 0$,
  - Or $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) = 1$.
- If $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) = 1$, what does that imply for $\boldsymbol{x}_n$?

Interpretation of the Solution

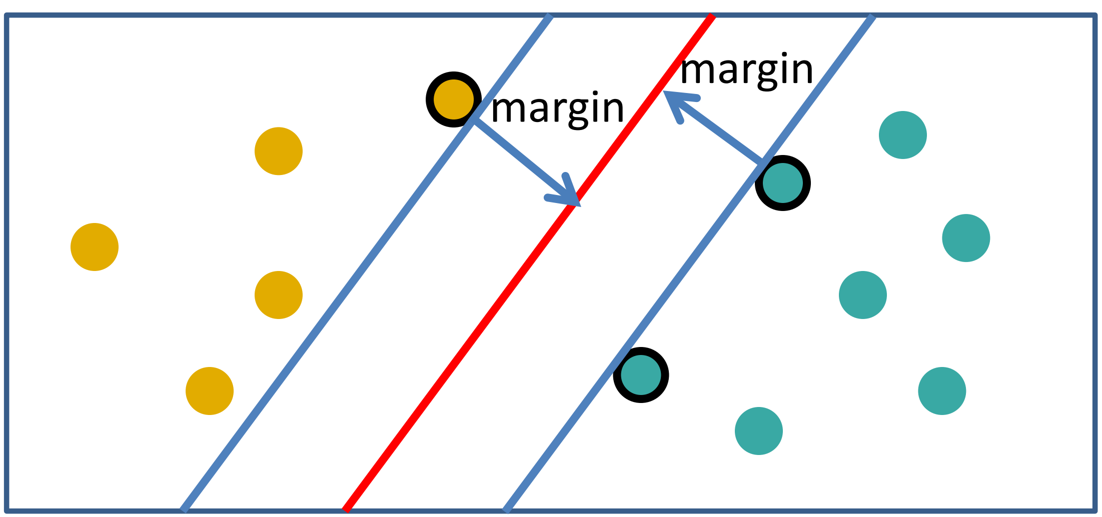
- In the SVM case, it holds that:

$$\forall n, \qquad a_n(t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) - 1) = 0$$

- What does this mean?
- Mathematically, $\forall n$:
  - Either $a_n = 0$,
  - Or $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) = 1$.
- If $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) = 1$, what does that imply for $\boldsymbol{x}_n$?
- Remember, many slides back, we imposed the constraint that $\forall n, \;\; t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1$.
- Equality $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) = 1$ holds only for the support vectors.
- Therefore, $a_n > 0$ **only for the support vectors**.

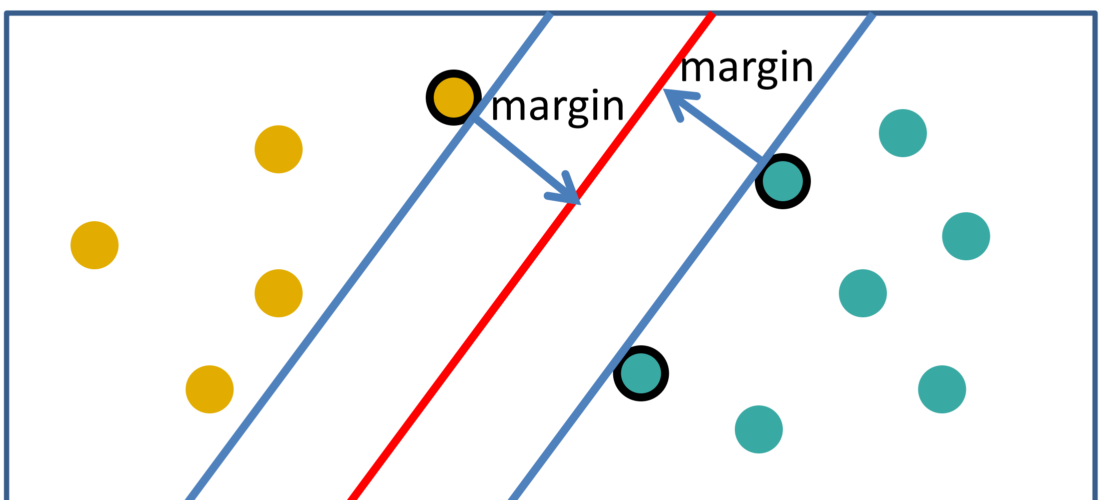
Support Vectors

- This is an example we have seen before.
- The three support vectors have a black circle around them.
- When we find the optimal values $a_1, a_2, \ldots, a_N$ for this problem, only three of those values will be non-zero.
  - If x_n is not a support vector, then $a_n = 0$.



Interpretation of the Solution

- We showed before that: $\boldsymbol{w} = \sum_{n=1}^{N} (a_n t_n \boldsymbol{x}_n)$
- This means that $\boldsymbol{w}$ is a linear combination of the training data.
- However, since $a_n > 0$ only for the support vectors, obviously only the support vectors influence $\boldsymbol{w}$.



Computing b

- $\boldsymbol{w} = \sum_{n=1}^{N} (a_n t_n \boldsymbol{x}_n)$
- Define set $S = \{n \mid \boldsymbol{x}_n$ is a support vector$\}$.
- Since $a_n > 0$ only for the support vectors, we get:

$$\boldsymbol{w} = \sum_{n \in S} (a_n t_n \boldsymbol{x}_n)$$

- If $\boldsymbol{x}_n$ is a support vector, then $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) = 1$.
- Substituting $\boldsymbol{w} = \sum_{m \in S} (a_m t_m \boldsymbol{x}_m)$, if $\boldsymbol{x}_n$ is a support vector:

$$t_n \left(\sum_{m \in S} (a_m t_m (\boldsymbol{x}_m)^T \boldsymbol{x}_n) + b \right) = 1$$

Computing b

$$t_n\left(\sum_{m\in S}(a_m t_m(\boldsymbol{x}_m)^T\boldsymbol{x}_n)+b\right)=1$$

- Remember that t_n can only take values 1 and -1.
- Therefore, $(t_n)^2 = 1$
- Multiplying both sides of the equation with t_n we get:

$$t_n t_n\left(\sum_{m\in S}(a_m t_m(\boldsymbol{x}_m)^T\boldsymbol{x}_n)+b\right)=t_n$$

$$\Rightarrow \sum_{m\in S}(a_m t_m(\boldsymbol{x}_m)^T\boldsymbol{x}_n)+b=t_n \Rightarrow b=t_n-\sum_{m\in S}(a_m t_m(\boldsymbol{x}_m)^T\boldsymbol{x}_n)$$

Computing b

- Thus, if $\boldsymbol{x}_n$ is a support vector, we can compute b with formula:

$$b = t_n - \sum_{m \in S} (a_m t_m (\boldsymbol{x}_m)^T \boldsymbol{x}_n)$$

- To avoid numerical problems, instead of using a single support vector to calculate b, we can use all support vectors (and take the average of the computed values for b).
- If N_S is the number of support vectors, then:

$$b = \frac{1}{N_S} \sum_{n \in S} \left(t_n - \sum_{m \in S} (a_m t_m (\boldsymbol{x}_m)^T \boldsymbol{x}_n) \right)$$

Classification Using $\boldsymbol{a}$

- To classify a test object $\boldsymbol{x}$, we can use the original formula $y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$.
- However, since $\boldsymbol{w} = \sum_{n=1}^{N} (a_n t_n \boldsymbol{x}_n)$, we can substitute that formula for $\boldsymbol{w}$, and classify $\boldsymbol{x}$ using:

$$y(\boldsymbol{x}) = \sum_{n \in S} (a_n t_n (\boldsymbol{x}_n)^T \boldsymbol{x}) + b$$

- This formula will be our only choice when we use SVMs that produce nonlinear boundaries.
  - Details on that will be coming later in this presentation.

Recap of Lagrangian-Based Solution

- We defined the **Lagrangian function**, which became (after simplifications):

$$L(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\boldsymbol{a}^T \boldsymbol{Q}\boldsymbol{a}$$

- Using quadratic programming, we find the value of $\boldsymbol{a}$ that maximizes $L(\boldsymbol{a})$, subject to constraints: $a_n \geq 0, \sum_{n=1}^{N}\{a_n t_n\} = 0$
- Then:

$$\boldsymbol{w} = \sum_{n \in S}(a_n t_n \boldsymbol{x}_n) \qquad b = \frac{1}{N_S}\sum_{n \in S}\left(t_n - \sum_{m \in S}(a_m t_m (\boldsymbol{x}_m)^T \boldsymbol{x}_n)\right)$$

Why Did We Do All This?

- The Lagrangian-based solution solves a problem that we had already solved, in a more complicated way.
- Typically, we prefer simpler solutions.
- However, this more complicated solution can be tweaked relatively easily, to produce more powerful SVMs that:
  - Can be trained even if the training data is not linearly separable.
  - Can produce nonlinear decision boundaries.

The Not Linearly Separable Case

- Our previous formulation required that the training examples are linearly separable.
- This requirement was encoded in the constraint:

$$\forall n \in \{1, \ldots, N\}, t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1$$

- According to that constraint, every $\boldsymbol{x}_n$ has to be on the correct side of the boundary.
- To handle data that is not linearly separable, we introduce N variables $\xi_n \geq 0$, to define a modified constraint:

$$\forall n \in \{1, \ldots, N\}, t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1 - \xi_n$$

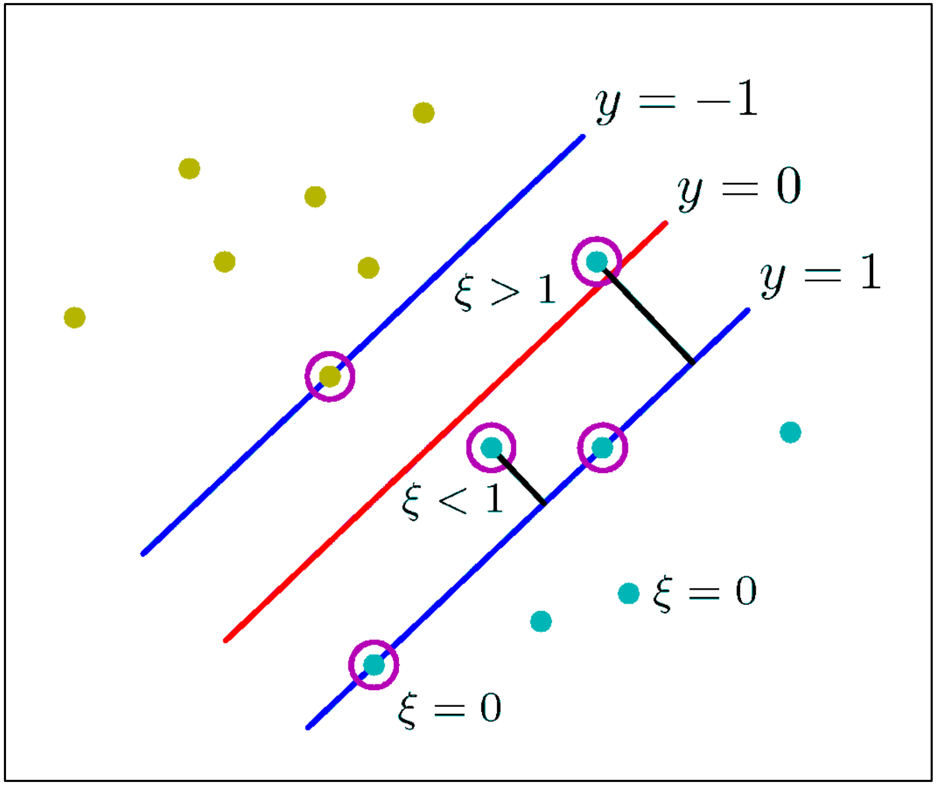
- These variables ξ_n are called **slack variables**.
- The values for ξ_n will be computed during optimization.

The Meaning of Slack Variables

- To handle data that is not linearly separable, we introduce N **slack variables** $\xi_n \geq 0$, to define a modified constraint:

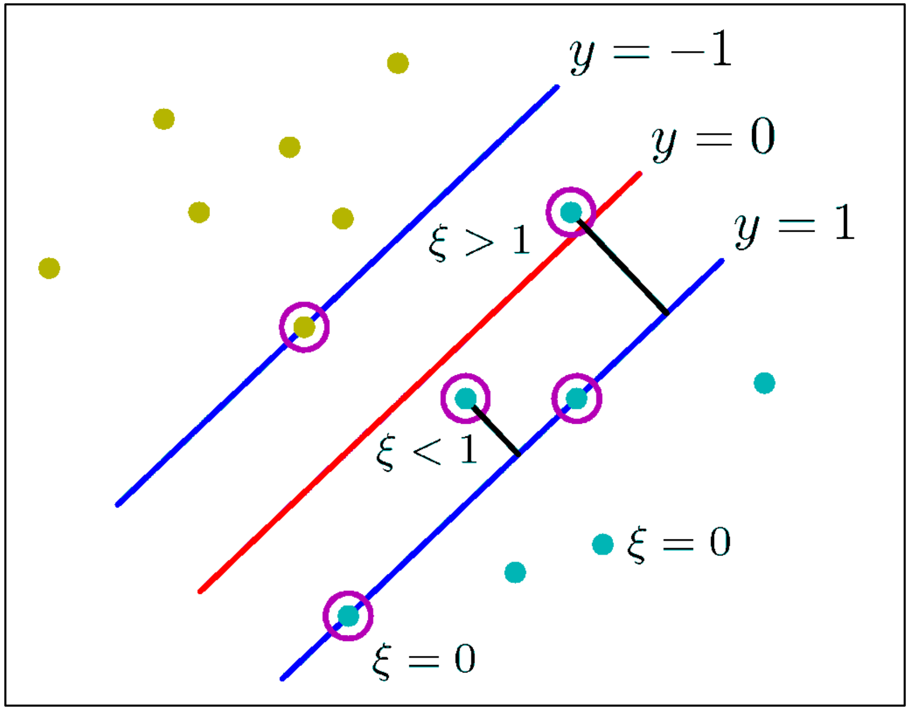
$$\forall n \in \{1, \ldots, N\}, t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1 - \xi_n$$

- If $\xi_n = 0$, then $\boldsymbol{x}_n$ is where it should be:
  - either on the blue line for objects of its class, or on the correct side of that blue line.
- If $0 < \xi_n < 1$, then $\boldsymbol{x}_n$ is too close to the decision boundary.
  - Between the red line and the blue line for objects of its class.
- If $\xi_n = 1$, $\boldsymbol{x}_n$ is on the red line.



- If $\xi_n > 1$, $\boldsymbol{x}_n$ is misclassified (on the wrong side of the red line).

The Meaning of Slack Variables



- If training data is not linearly separable, we introduce N **slack variables** $\xi_n \geq 0$, to define a modified constraint:

$$\forall n \in \{1, \ldots, N\}, t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1 - \xi_n$$

- If $\xi_n = 0$, then $\boldsymbol{x}_n$ is where it should be:
  - either on the blue line for objects of its class, or on the correct side of that blue line.
- If $0 < \xi_n < 1$, then $\boldsymbol{x}_n$ is too close to the decision boundary.
  - Between the red line and the blue line for objects of its class.
- If $\xi_n = 1$, $\boldsymbol{x}_n$ is on the decision boundary (the red line).
- If $\xi_n > 1$, $\boldsymbol{x}_n$ is misclassified (on the wrong side of the red line).

Optimization Criterion

- Before, we minimized $\frac{1}{2}\|\boldsymbol{w}\|^2$ subject to constraints:

$$\forall n \in \{1, \ldots, N\}, t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1$$

- Now, we want to minimize error function: $C\left(\sum_{n=1}^{N} \xi_n\right) + \frac{1}{2}\|\boldsymbol{w}\|^2$ subject to constraints:

$$\forall n \in \{1, \ldots, N\}, t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1 - \xi_n$$
$$\xi_n \geq 0$$

- C is a parameter that we pick manually, $C \geq 0$.
- C controls the trade-off between maximizing the margin, and penalizing training examples that violate the margin.
  - The higher ξ_n is, the farther away $\boldsymbol{x}_n$ is from where it should be.
  - If $\boldsymbol{x}_n$ is on the correct side of the decision boundary, and the distance of $\boldsymbol{x}_n$ to the boundary is greater than or equal to the margin, then $\xi_n = 0$, and $\boldsymbol{x}_n$ does not contribute to the error function.

Lagrangian Function

- To do our constrained optimization, we define the Lagrangian:

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{\mu}) =$$

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_{n=1}^{N} \xi_n\right) - \sum_{n=1}^{N}\{a_n(t_n \boldsymbol{y}(\boldsymbol{x}_n) - 1 + \xi_n)\} - \left(\sum_{n=1}^{N} \mu_n \xi_n\right)$$

- The Lagrange multipliers are now a_n and μ_n.
- The term $a_n(t_n \boldsymbol{y}(\boldsymbol{x}_n) - 1 + \xi_n)$ in the Lagrangian corresponds to constraint $t_n \boldsymbol{y}(\boldsymbol{x}_n) - 1 + \xi_n \geq 0$.
- The term $\mu_n \xi_n$ in the Lagrangian corresponds to constraint $\xi_n \geq 0$.

Lagrangian Function

- Lagrangian $L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{\mu})$:

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_{n=1}^{N} \xi_n\right) - \sum_{n=1}^{N}\{a_n(t_n \boldsymbol{y}(\boldsymbol{x}_n) - 1 + \xi_n)\} - \left(\sum_{n=1}^{N} \mu_n \xi_n\right)$$

- Constraints:

$$\begin{aligned}
&a_n \geq 0 \\
&\mu_n \geq 0 \\
&\xi_n \geq 0 \\
&t_n \boldsymbol{y}(\boldsymbol{x}_n) - 1 + \xi_n \geq 0 \\
&a_n(t_n \boldsymbol{y}(\boldsymbol{x}_n) - 1 + \xi_n) = 0 \\
&\mu_n \xi_n = 0
\end{aligned}$$

Lagrangian Function

- Lagrangian $L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{\mu})$:

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_{n=1}^{N} \xi_n\right) - \sum_{n=1}^{N}\{a_n(t_n \boldsymbol{y}(\boldsymbol{x}_n) - 1 + \xi_n)\} - \left(\sum_{n=1}^{N} \mu_n \xi_n\right)$$

- As before, we can simplify the Lagrangian significantly:

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0 \Rightarrow \boldsymbol{w} = \sum_{n=1}^{N}\{a_n t_n \boldsymbol{x}_n\}$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N}\{a_n t_n\} = 0$$

$$\frac{\partial L}{\partial \xi_n} = 0 \Rightarrow C - a_n - \mu_n = 0 \Rightarrow a_n = C - \mu_n$$

Lagrangian Function

- Lagrangian $L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{\mu})$:

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_{n=1}^{N} \xi_n\right) - \sum_{n=1}^{N}\{a_n(t_n \boldsymbol{y}(\boldsymbol{x}_n) - 1 + \xi_n)\} - \left(\sum_{n=1}^{N} \mu_n \xi_n\right)$$

- In computing $L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{\mu})$, ξ_n contributes the following value:

$$C\xi_n - a_n\, \xi_n - \mu_n \xi_n = \xi_n(C - a_n - \mu_n)$$

- As we saw in the previous slide, $C - a_n - \mu_n = 0$.
- Therefore, we can eliminate all those occurrences of ξ_n.

Lagrangian Function

- Lagrangian $L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{\mu})$:

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_{n=1}^{N} \xi_n\right) - \sum_{n=1}^{N}\{a_n(t_n \boldsymbol{y}(\boldsymbol{x}_n) - 1 + \xi_n)\} - \left(\sum_{n=1}^{N} \mu_n \xi_n\right)$$

- In computing $L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{\mu})$, ξ_n contributes the following value:

$$C\xi_n - a_n\, \xi_n - \mu_n \xi_n = \xi_n(C - a_n - \mu_n)$$

- As we saw in the previous slide, $C - a_n - \mu_n = 0$.
- Therefore, we can eliminate all those occurrences of ξ_n.
- By eliminating ξ_n we also eliminate all appearances of C and μ_n.

Lagrangian Function

- By eliminating ξ_n and μ_n, we get:

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{n=1}^{N}\{a_n(t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - 1)\}$$

- This is **exactly** the Lagrangian we had for the linearly separable case.
- Following the same steps as in the linearly separable case, we can simplify even more to:

$$L(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\boldsymbol{a}^T\,\boldsymbol{Q}\boldsymbol{a}$$

Lagrangian Function

- So, we have ended up with the same Lagrangian as in the linearly separable case:

$$L(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\boldsymbol{a}^T\,\boldsymbol{Q}\boldsymbol{a}$$

- There is a small difference, however, in the constraints.

Linearly separable case:

$0 \leq a_n$

$$\sum_{n=1}^{N}\{a_n t_n\} = 0$$

Linearly inseparable case:

$0 \leq a_n \leq C$

$$\sum_{n=1}^{N}\{a_n t_n\} = 0$$

Lagrangian Function

- So, we have ended up with the same Lagrangian as in the linearly separable case:

$$L(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\boldsymbol{a}^T \boldsymbol{Q}\boldsymbol{a}$$

- Where does constraint $0 \leq a_n \leq C$ come from?

- $0 \leq a_n$ because a_n is a Lagrange multiplier.

- $a_n \leq C$ comes from the fact that we showed earlier, that $a_n = C - \mu_n$.
  - Since μ_n is also a Lagrange multiplier, $\mu_n \geq 0$ and thus $a_n \leq C$.

Using Quadratic Programming

- Quadratic programming: $\boldsymbol{u}_{\text{opt}} = \text{argmin}_u \left\{\frac{1}{2}\boldsymbol{u}^T \boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T \boldsymbol{u}\right\}$
  subject to constraint: $\boldsymbol{H}\boldsymbol{u} \leq \boldsymbol{z}$
- SVM problem: find $\boldsymbol{a}_{\text{opt}} = \underset{\boldsymbol{a}}{\text{argmin}} \left\{\frac{1}{2}\boldsymbol{a}^T \boldsymbol{Q}\boldsymbol{a} - \sum_{n=1}^{N} a_n\right\}$
  subject to constraints:

$$0 \leq a_n \leq C, \qquad \sum_{n=1}^{N}\{a_n t_n\} = 0$$

- Again, we must find values for $\boldsymbol{Q}, \boldsymbol{s}, \boldsymbol{H}, \boldsymbol{z}$ that convert the SVM problem into a quadratic programming problem.
- Values for $\boldsymbol{Q}$ and $\boldsymbol{s}$ are the same as in the linearly separable case, since in both cases we minimize $\frac{1}{2}\boldsymbol{a}^T \boldsymbol{Q}\boldsymbol{a} - \sum_{n=1}^{N} a_n$.

Using Quadratic Programming

- Quadratic programming: $\boldsymbol{u}_{\text{opt}} = \text{argmin}_u \left\{\frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{s}^T\boldsymbol{u}\right\}$
subject to constraint: $\boldsymbol{H}\boldsymbol{u} \leq \boldsymbol{z}$
- SVM problem: find $\boldsymbol{a}_{\text{opt}} = \underset{\boldsymbol{a}}{\text{argmin}} \left\{\frac{1}{2}\boldsymbol{a}^T\,\boldsymbol{Q}\boldsymbol{a} \;-\sum_{n=1}^{N} a_n\right\}$
subject to constraints:

$$0 \leq a_n \leq C, \qquad \sum_{n=1}^{N}\{a_n t_n\} = 0$$

- $\boldsymbol{Q}$ is an $N \times N$ matrix such that $Q_{mn} = t_n t_m (\boldsymbol{x}_n)^T\,\boldsymbol{x}_n$.
- $\boldsymbol{u} = \boldsymbol{a}$, and $\boldsymbol{s} = \begin{bmatrix} -1 \\ -1 \\ \dots \\ -1 \end{bmatrix}_N$.

Using Quadratic Programming

- Quadratic programming constraint: $\boldsymbol{Hu} \leq \boldsymbol{z}$
- SVM problem constraints: $0 \leq a_n \leq C, \quad \sum_{n=1}^{N}\{a_n t_n\} = 0$

Define:

$$\boldsymbol{H} = \begin{bmatrix} -1, & 0, & 0, & \dots, & 0 \\ 0, & -1, & 0, & \dots, & 0 \\ & & \dots & & \\ 0, & 0, & 0, & \dots, & -1 \\ 1, & 0, & 0, & \dots, & 0 \\ 0, & 1, & 0, & \dots, & 0 \\ & & \dots & & \\ 0, & 0, & 0, & \dots, & 1 \\ t_1, & t_2, & t_3, & \dots, & t_n \\ -t_1, & -t_2, & -t_3, & \dots, & -t_n \end{bmatrix}$$

rows 1 to N

rows (N + 1) to $2N$

rows (2N + 1) and (2N + 2)

$$\boldsymbol{z} = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \\ C \\ C \\ \dots \\ C \\ 0 \\ 0 \end{bmatrix}$$

rows 1 to N, set to 0.

rows (N + 1) to $2N$, set to C.

rows 2N + 1, 2N + 2, set to 0.

- The top N rows of $\boldsymbol{H}$ are the negation of the $N \times N$ identity matrix.
- Rows $N + 1$ to 2N of $\boldsymbol{H}$ are the $N \times N$ identity matrix
- Row $N + 1$ of $\boldsymbol{H}$ is the transpose of vector $\boldsymbol{t}$ of target outputs.
- Row $N + 2$ of $\boldsymbol{H}$ is the negation of the previous row.

Using Quadratic Programming

- Quadratic programming constraint: $\boldsymbol{Hu} \leq \boldsymbol{z}$
- SVM problem constraints: $0 \leq a_n \leq C, \quad \sum_{n=1}^{N}\{a_n t_n\} = 0$

Define: $$\boldsymbol{H} = \begin{bmatrix} -1, & 0, & 0, & \dots, & 0 \\ 0, & -1, & 0, & \dots, & 0 \\ & & \dots & & \\ 0, & 0, & 0, & \dots, & -1 \\ 1, & 0, & 0, & \dots, & 0 \\ 0, & 1, & 0, & \dots, & 0 \\ & & \dots & & \\ 0, & 0, & 0, & \dots, & 1 \\ t_1, & t_2, & t_3, & \dots, & t_n \\ -t_1, & -t_2, & -t_3, & \dots, & -t_n \end{bmatrix}$$

rows 1 to N

rows (N + 1) to $2N$

rows (2N + 1) and (2N + 2)

$$\boldsymbol{z} = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \\ C \\ C \\ \dots \\ C \\ 0 \\ 0 \end{bmatrix}$$

rows 1 to N, set to 0.

rows (N + 1) to $2N$, set to C.

rows 2N + 1, 2N + 2, set to 0.

- If $1 \leq n \leq N$, the n-th row of $\boldsymbol{Hu}$ is $-a_n$, and the n-th row of $\boldsymbol{z}$ is 0.
- Thus, the n-th rows of $\boldsymbol{Hu}$ and $\boldsymbol{z}$ capture constraint $-a_n \leq 0 \Rightarrow a_n \geq 0$.

Using Quadratic Programming

- Quadratic programming constraint: $\boldsymbol{Hu} \leq \boldsymbol{z}$
- SVM problem constraints: $0 \leq a_n \leq C, \quad \sum_{n=1}^{N}\{a_n t_n\} = 0$

Define: $$\boldsymbol{H} = \begin{bmatrix} -1, & 0, & 0, & \dots, & 0 \\ 0, & -1, & 0, & \dots, & 0 \\ & & \dots & & \\ 0, & 0, & 0, & \dots, & -1 \\ 1, & 0, & 0, & \dots, & 0 \\ 0, & 1, & 0, & \dots, & 0 \\ & & \dots & & \\ 0, & 0, & 0, & \dots, & 1 \\ t_1, & t_2, & t_3, & \dots, & t_n \\ -t_1, & -t_2, & -t_3, & \dots, & -t_n \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} \text{rows } 1 \text{ to } N \\ \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} \text{rows } (\mathrm{N}+1) \text{ to } 2N \\ \left.\vphantom{\begin{matrix}1\\1\end{matrix}}\right\} \text{rows } (2\mathrm{N}+1) \text{ and } (2\mathrm{N}+2) \end{matrix}$$

$$\boldsymbol{z} = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \\ C \\ C \\ \dots \\ C \\ 0 \\ 0 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} \text{rows } 1 \text{ to } N, \text{ set to } 0. \\ \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} \text{rows } (\mathrm{N}+1) \text{ to } 2N, \text{ set to } C. \\ \left.\vphantom{\begin{matrix}1\\1\end{matrix}}\right\} \text{rows } 2\mathrm{N}+1,\ 2\mathrm{N}+2, \text{ set to } 0. \end{matrix}$$

- If $(\mathrm{N}+1) \leq n \leq 2N$, the n-th row of $\boldsymbol{Hu}$ is a_n, and the n-th row of $\boldsymbol{z}$ is C.
- Thus, the n-th rows of $\boldsymbol{Hu}$ and $\boldsymbol{z}$ capture constraint $a_n \leq 0$.

Using Quadratic Programming

- Quadratic programming constraint: $\boldsymbol{H}\boldsymbol{u} \le \boldsymbol{z}$
- SVM problem constraints: $0 \le a_n \le C, \quad \sum_{n=1}^{N}\{a_n t_n\} = 0$

$$
\text{Define: } \boldsymbol{H} = \begin{bmatrix}
-1, & 0, & 0, & \dots, & 0 \\
0, & -1, & 0, & \dots, & 0 \\
 & & \dots & & \\
0, & 0, & 0, & \dots, & -1 \\
1, & 0, & 0, & \dots, & 0 \\
0, & 1, & 0, & \dots, & 0 \\
 & & \dots & & \\
0, & 0, & 0, & \dots, & 1 \\
t_1, & t_2, & t_3, & \dots, & t_n \\
-t_1, & -t_2, & -t_3, & \dots, & -t_n
\end{bmatrix}
\begin{array}{l}
\left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} \text{rows } 1 \text{ to } N \\
\left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} \text{rows } (\mathrm{N}+1) \text{ to } 2N \\
\left.\begin{array}{l} \\ \\ \end{array}\right\} \text{rows } (2\mathrm{N}+1) \text{ and } (2\mathrm{N}+2)
\end{array}
\qquad
\boldsymbol{z} = \begin{bmatrix}
0 \\ 0 \\ \dots \\ 0 \\ C \\ C \\ \dots \\ C \\ 0 \\ 0
\end{bmatrix}
\begin{array}{l}
\left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} \text{rows } 1 \text{ to } N, \text{ set to } 0. \\
\left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} \text{rows } (\mathrm{N}+1) \text{ to } 2N, \text{ set to } C. \\
\left.\begin{array}{l} \\ \\ \end{array}\right\} \text{rows } 2\mathrm{N}+1,\ 2\mathrm{N}+2, \text{ set to } 0.
\end{array}
$$

- If $n = 2N + 1$, the n-th row of $\boldsymbol{H}\boldsymbol{u}$ is $\boldsymbol{t}$, and the n-th row of $\boldsymbol{z}$ is 0.
- Thus, the n-th rows of $\boldsymbol{H}\boldsymbol{u}$ and $\boldsymbol{z}$ capture constraint $\sum_{n=1}^{N}\{a_n t_n\} \le 0$.

Using Quadratic Programming

- Quadratic programming constraint: $\boldsymbol{Hu} \leq \boldsymbol{z}$
- SVM problem constraints: $0 \leq a_n \leq C, \quad \sum_{n=1}^{N}\{a_n t_n\} = 0$

Define: $$\boldsymbol{H} = \begin{bmatrix} -1, & 0, & 0, & \ldots, & 0 \\ 0, & -1, & 0, & \ldots, & 0 \\ & & \cdots & & \\ 0, & 0, & 0, & \ldots, & -1 \\ 1, & 0, & 0, & \ldots, & 0 \\ 0, & 1, & 0, & \ldots, & 0 \\ & & \cdots & & \\ 0, & 0, & 0, & \ldots, & 1 \\ t_1, & t_2, & t_3, & \ldots, & t_n \\ -t_1, & -t_2, & -t_3, & \ldots, & -t_n \end{bmatrix}$$

rows 1 to N

rows (N + 1) to $2N$

rows (2N + 1) and (2N + 2)

$$\boldsymbol{z} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \\ C \\ C \\ \cdots \\ C \\ 0 \\ 0 \end{bmatrix}$$

rows 1 to N, set to 0.

rows (N + 1) to $2N$, set to C.

rows 2N + 1, 2N + 2, set to 0.

- If $n = 2N + 2$, the n-th row of $\boldsymbol{Hu}$ is $-\boldsymbol{t}$, and the n-th row of $\boldsymbol{z}$ is 0.
- Thus, the n-th rows of $\boldsymbol{Hu}$ and $\boldsymbol{z}$ capture constraint $-\sum_{n=1}^{N}\{a_n t_n\} \leq 0 \Rightarrow \sum_{n=1}^{N}\{a_n t_n\} \geq 0$.

Using Quadratic Programming

- Quadratic programming constraint: $\boldsymbol{Hu} \leq \boldsymbol{z}$
- SVM problem constraints: $0 \leq a_n \leq C, \quad \sum_{n=1}^{N}\{a_n t_n\} = 0$

Define: $\boldsymbol{H} = \begin{bmatrix} -1, & 0, & 0, & \dots, & 0 \\ 0, & -1, & 0, & \dots, & 0 \\ & & \dots & & \\ 0, & 0, & 0, & \dots, & -1 \\ 1, & 0, & 0, & \dots, & 0 \\ 0, & 1, & 0, & \dots, & 0 \\ & & \dots & & \\ 0, & 0, & 0, & \dots, & 1 \\ t_1, & t_2, & t_3, & \dots, & t_n \\ -t_1, & -t_2, & -t_3, & \dots, & -t_n \end{bmatrix}$

rows 1 to N

rows (N + 1) to $2N$

rows (2N + 1) and (2N + 2)

$\boldsymbol{z} = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \\ C \\ C \\ \dots \\ C \\ 0 \\ 0 \end{bmatrix}$

rows 1 to N, set to 0.

rows (N + 1) to $2N$, set to C.

rows 2N + 1, 2N + 2, set to 0.

- Thus, the last two rows of $\boldsymbol{Hu}$ and $\boldsymbol{z}$ in combination capture constraint

$$\sum_{n=1}^{N}\{a_n t_n\} = 0$$

Using the Solution

- Quadratic programming, given the inputs $\boldsymbol{Q}, \boldsymbol{s}, \boldsymbol{H}, \boldsymbol{z}$ defined in the previous slides, outputs the optimal value for vector $\boldsymbol{a}$.
- The value of b is computed as in the linearly separable case:

$$b = \frac{1}{N_S} \sum_{n \in S} \left(t_n - \sum_{m \in S} (a_m t_m (\boldsymbol{x}_m)^T \boldsymbol{x}_n) \right)$$

- Classification of input x is done as in the linearly separable case:

$$y(\boldsymbol{x}) = \sum_{n \in S} (a_n t_n (\boldsymbol{x}_n)^T \boldsymbol{x}) + b$$

The Disappearing $\boldsymbol{w}$

- We have used vector $\boldsymbol{w}$, to define the decision boundary $y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$.
- However, $\boldsymbol{w}$ does not need to be either computed, or used.
- Quadratic programming computes values a_n.
- Using those values a_n, we compute b:

$$b = \frac{1}{N_S} \sum_{n \in S} \left(t_n - \sum_{m \in S} (a_m t_m (\boldsymbol{x}_m)^T \boldsymbol{x}_n) \right)$$

- Using those values for a_n and b, we can classify test objects $\boldsymbol{x}$:

$$y(\boldsymbol{x}) = \sum_{n \in S} (a_n t_n (\boldsymbol{x}_n)^T \boldsymbol{x}) + b$$

- If we know the values for a_n and b, we do not need $\boldsymbol{w}$.

The Role of Training Inputs

- Where, during training, do we use input vectors?
- Where, during classification, do we use input vectors?
- Overall, input vectors are only used in two formulas:

1. During training, we use vectors $\boldsymbol{x}_n$ to define matrix $\boldsymbol{Q}$**,** where:

$$Q_{mn} = t_n t_m (\boldsymbol{x}_n)^T \boldsymbol{x}_m$$

2. During classification, we use training vectors $\boldsymbol{x}_n$ and test input $\boldsymbol{x}$ in this formula:

$$y(\boldsymbol{x}) = \sum_{n \in S} (a_n t_n (\boldsymbol{x}_n)^T \boldsymbol{x}) + b$$

- In both formulas, input vectors are only used by taking their dot products.

The Role of Training Inputs

- To make this more clear, define a **kernel function** $k(\boldsymbol{x}, \boldsymbol{x}')$ as:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \, \boldsymbol{x}'$$

- During training, we use vectors $\boldsymbol{x}_n$ to define matrix $\boldsymbol{Q}$**,** where:

$$Q_{mn} = t_n t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

- During classification, we use training vectors $\boldsymbol{x}_n$ and test input $\boldsymbol{x}$ in this formula:

$$y(\boldsymbol{x}) = \sum_{n \in S} (a_n t_n k(\boldsymbol{x}_n, \boldsymbol{x})) + b$$

- In both formulas, input vectors are used only through function k.

The Kernel Trick

- We have defined **kernel function** $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T\, \boldsymbol{x}'$.
- During training, we use vectors $\boldsymbol{x}_n$ to define matrix $\boldsymbol{Q}$, where:

$$Q_{mn} = t_n t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

- During classification, we use this formula:

$$y(\boldsymbol{x}) = \sum_{n \in S} \left(a_n t_n k(\boldsymbol{x}_n, \boldsymbol{x})\right) + b$$

- What if we defined $k(\boldsymbol{x}, \boldsymbol{x}')$ differently?
- The SVM formulation (both for training and for classification) would <u>**remain exactly the same**</u>.
- In the SVM formulation, the kernel $k(\boldsymbol{x}, \boldsymbol{x}')$ is a black box.
  - You can define $k(\boldsymbol{x}, \boldsymbol{x}')$ any way you like.

A Different Kernel

- Let $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{z} = (z_1, z_2)$ be 2-dimensional vectors.
- Consider this alternative definition for the kernel:

$$k(\boldsymbol{x}, \boldsymbol{z}) = \left(1 + \boldsymbol{x}^T\, \boldsymbol{z}\right)^2$$

- Then:

$$k(\boldsymbol{x}, \boldsymbol{z}) = \left(1 + \boldsymbol{x}^T\, \boldsymbol{z}\right)^2 = (1 + x_1 z_1 + x_2 z_2)^2$$

$$= 1 + 2x_1 z_1 + 2x_2 z_2 + (x_1)^2 (z_1)^2 + 2x_1 z_1 2x_2 z_2 + (x_2)^2 (z_2)^2$$

- Suppose we define a basis function $\varphi(\boldsymbol{x})$ as:

$$\varphi(\boldsymbol{x}) = (1, \sqrt{2} x_1, \sqrt{2} x_2, (x_1)^2, \sqrt{2} x_1 x_2, (x_2)^2)$$

- It is easy to verify that $k(\boldsymbol{x}, \boldsymbol{z}) = \varphi(\boldsymbol{x})^{\boldsymbol{T}} \varphi(\boldsymbol{z})$

Kernels and Basis Functions

- In general, kernels make it easy to incorporate basis functions into SVMs:
  - Define $\varphi(\boldsymbol{x})$ any way you like.
  - Define $k(\boldsymbol{x}, \boldsymbol{z}) = \varphi(\boldsymbol{x})^{\boldsymbol{T}} \varphi(\boldsymbol{z})$.
- The kernel function represents a dot product, but in a (typically) higher-dimensional feature space compared to the original space of $\boldsymbol{x}$ and $\boldsymbol{z}$.

Polynomial Kernels

- Let $\boldsymbol{x}$ and $\boldsymbol{z}$ be D-dimensional vectors.
- A polynomial kernel of degree d is defined as:

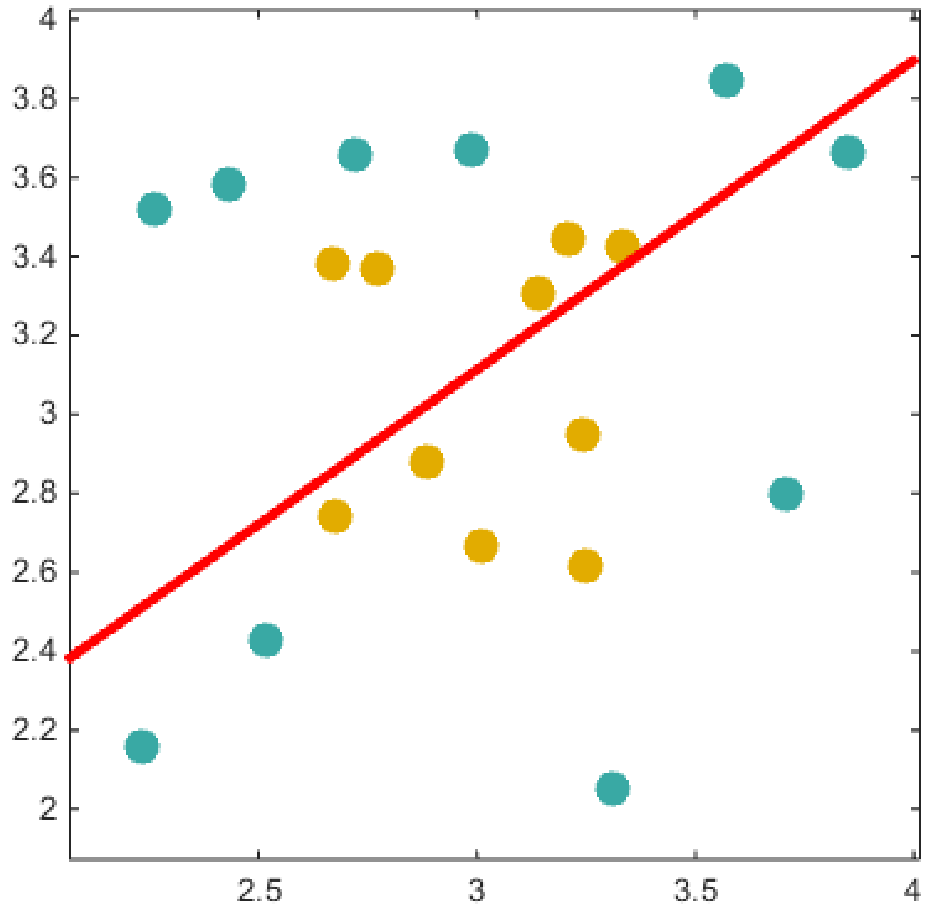
$$k(\boldsymbol{x}, \boldsymbol{z}) = \left(c + \boldsymbol{x}^T \boldsymbol{z}\right)^d$$

- The kernel $k(\boldsymbol{x}, \boldsymbol{z}) = \left(1 + \boldsymbol{x}^T \boldsymbol{z}\right)^2$ that we saw a couple of slides back was a quadratic kernel.
- Parameter c controls the trade-off between influence higher-order and lower-order terms.
  - Increasing values of c give increasing influence to lower-order terms.

Polynomial Kernels – An Easy Case

Decision boundary with polynomial kernel of degree 1.

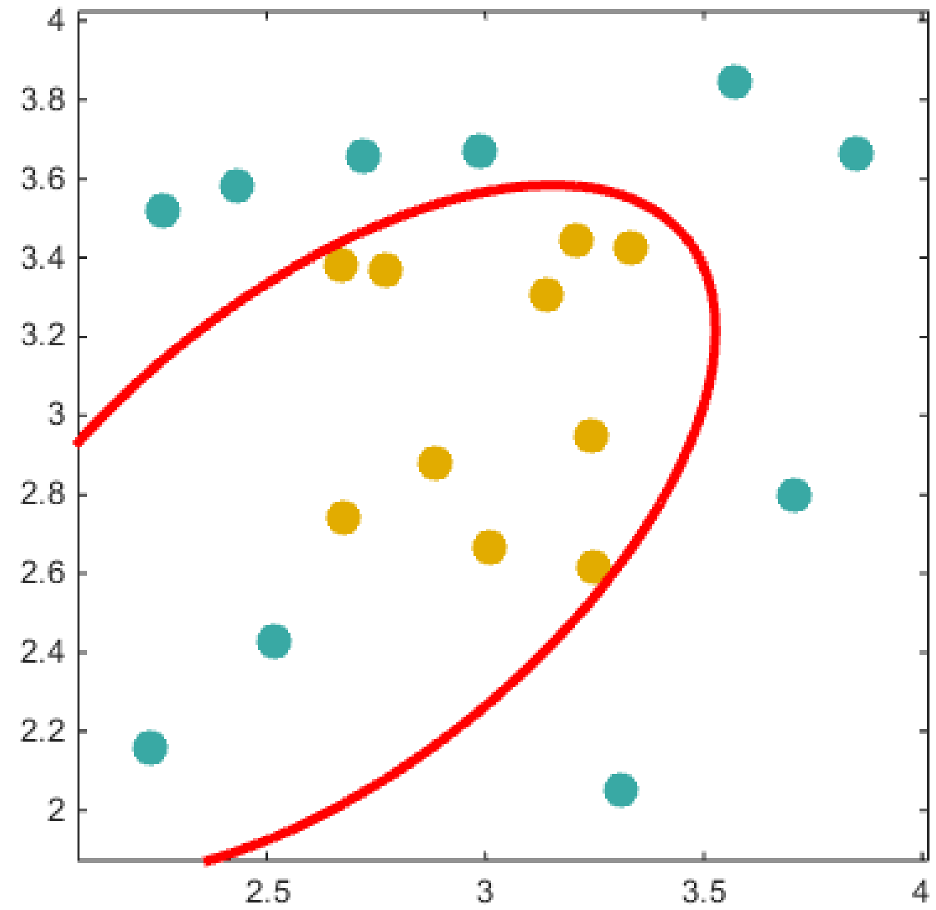
This is identical to the result using the standard dot product as kernel.



Polynomial Kernels – An Easy Case

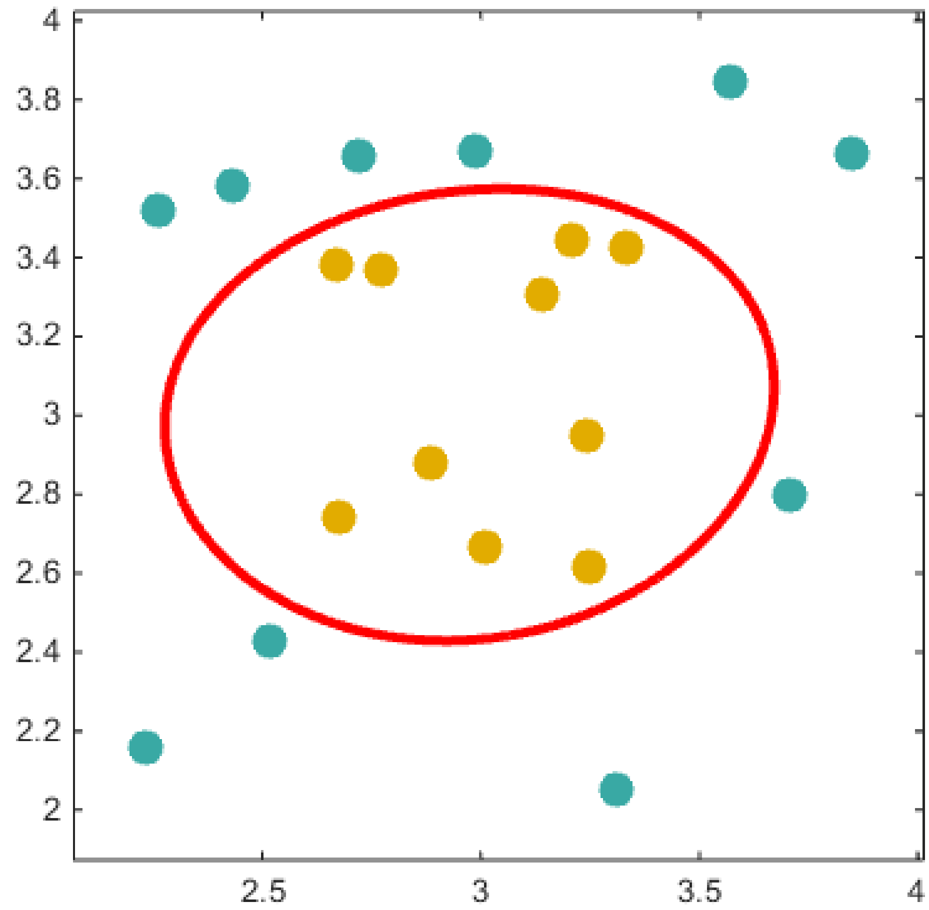
Decision boundary with polynomial kernel of degree 2.

The decision boundary is not linear anymore.



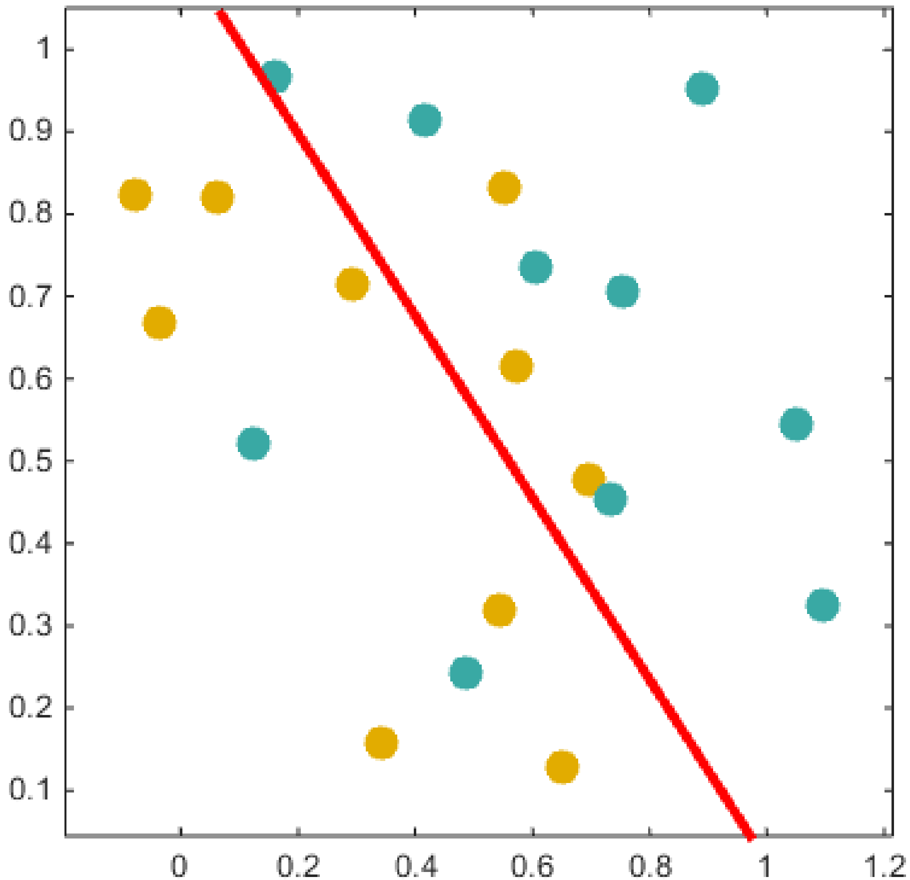
Polynomial Kernels – An Easy Case

Decision boundary with polynomial kernel of degree 3.



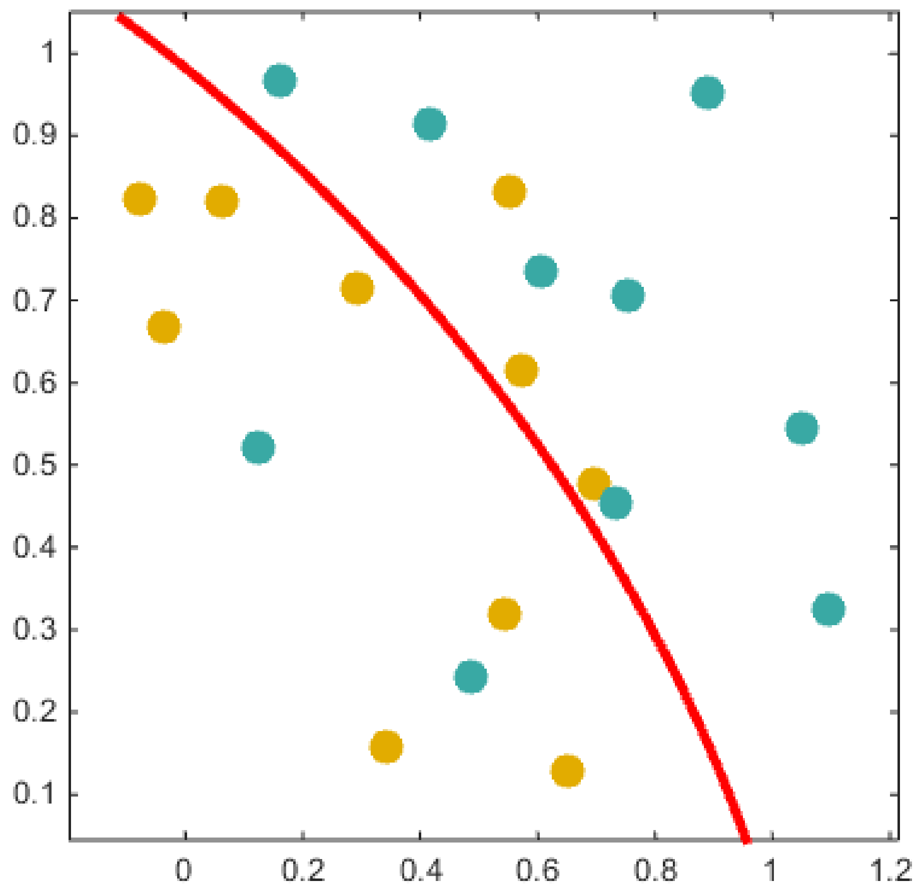
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 1.



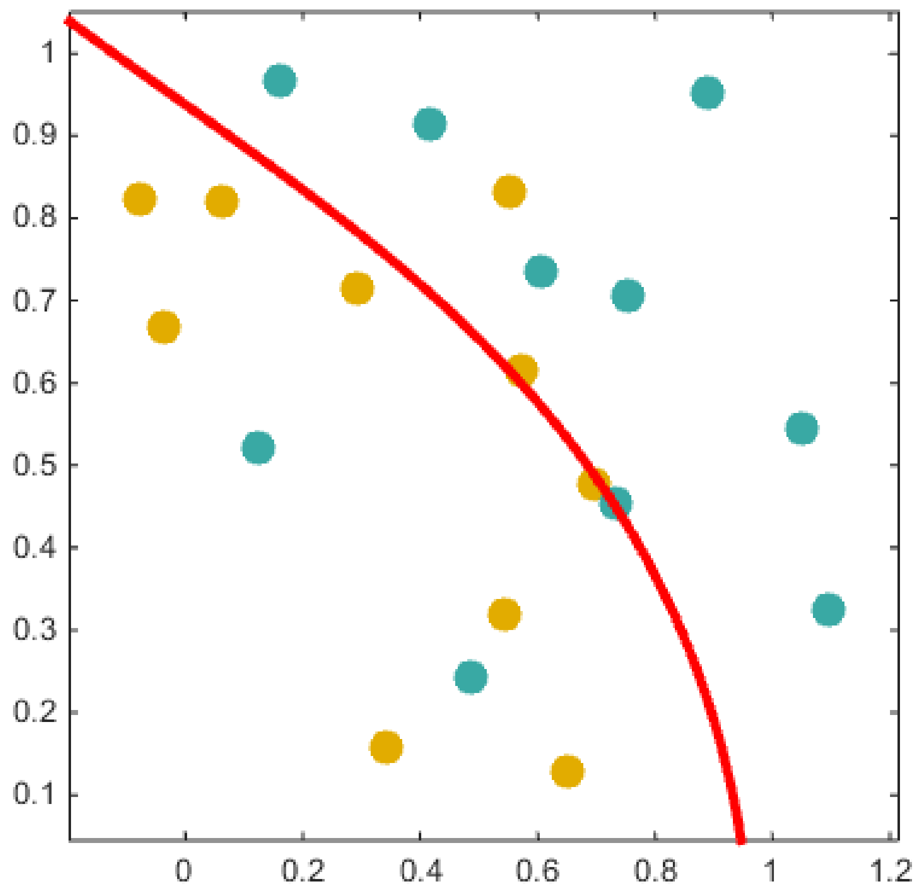
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 2.



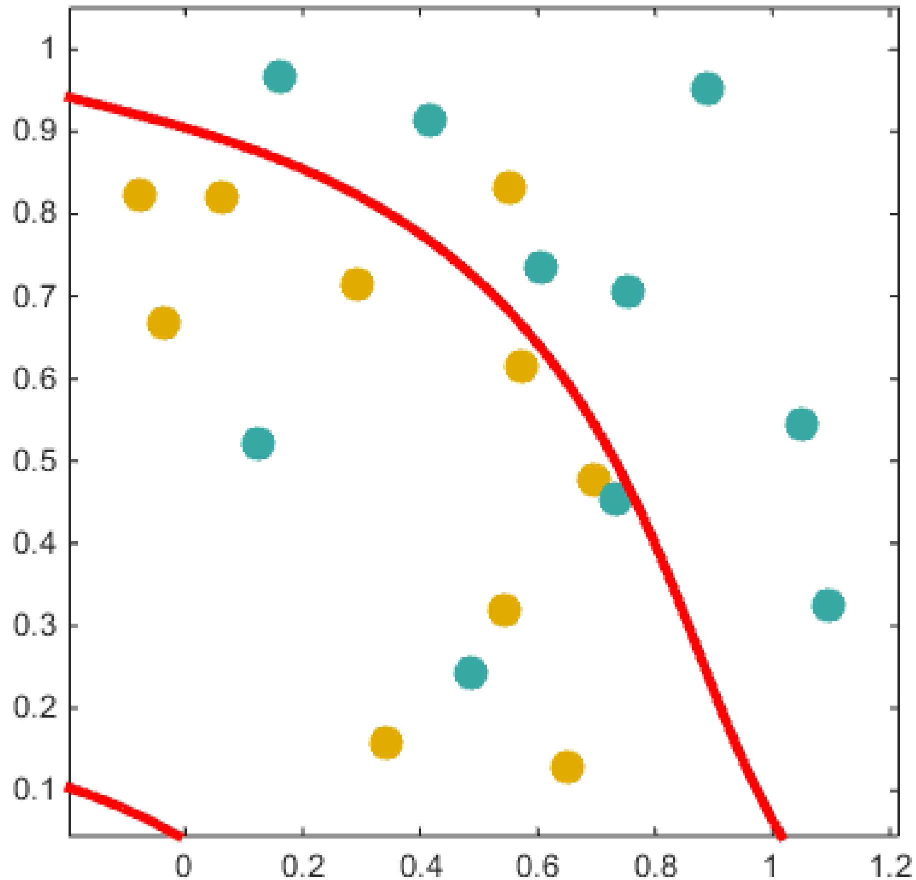
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 3.



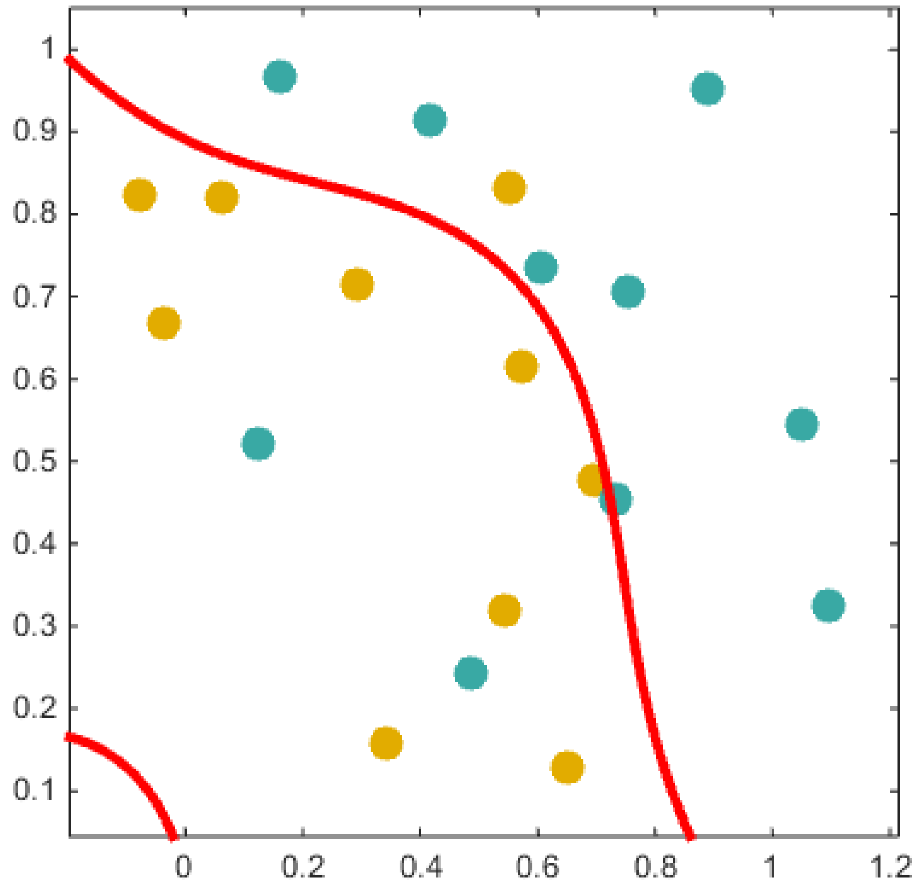
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 4.



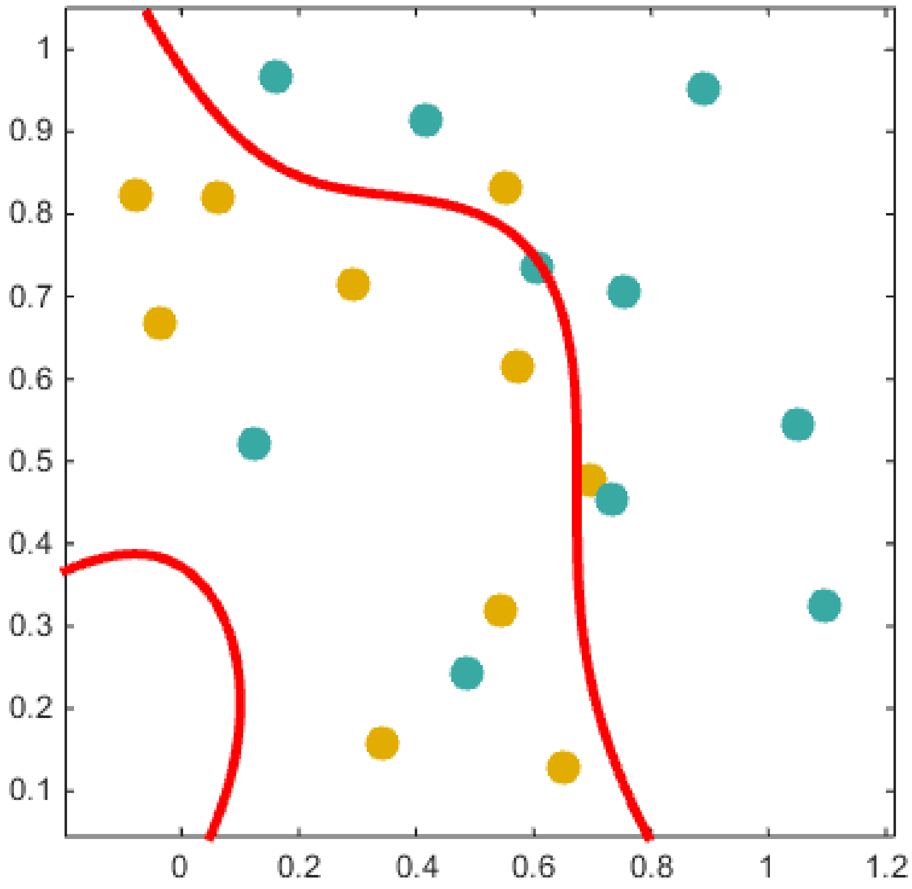
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 5.



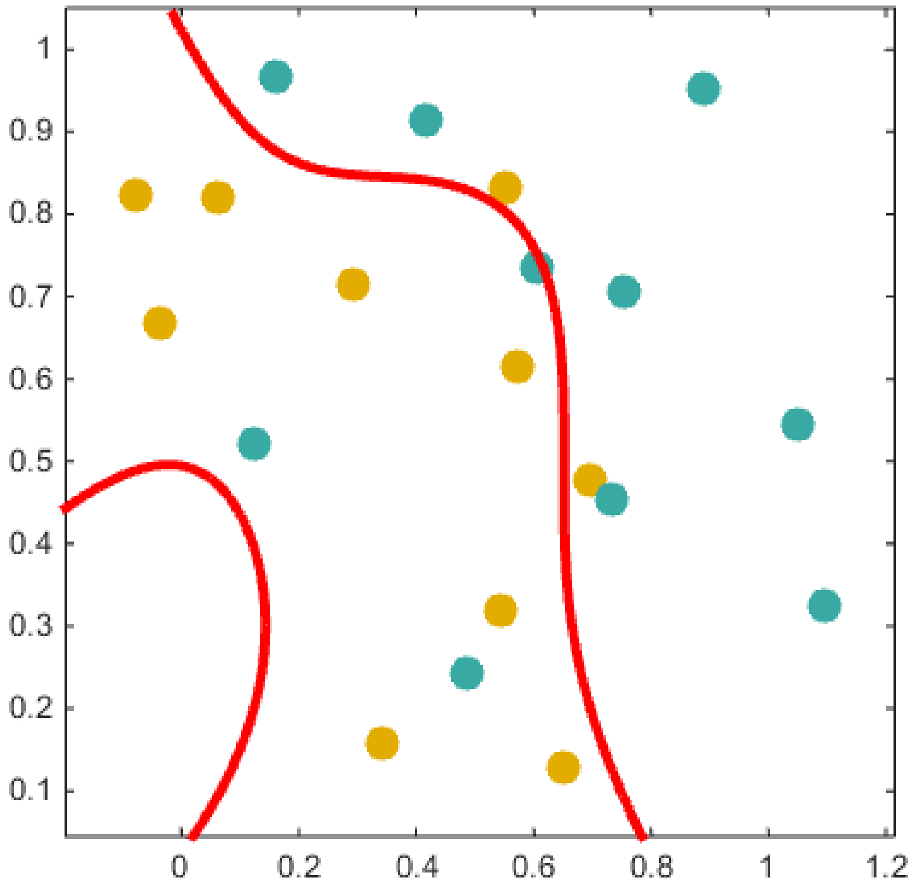
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 6.



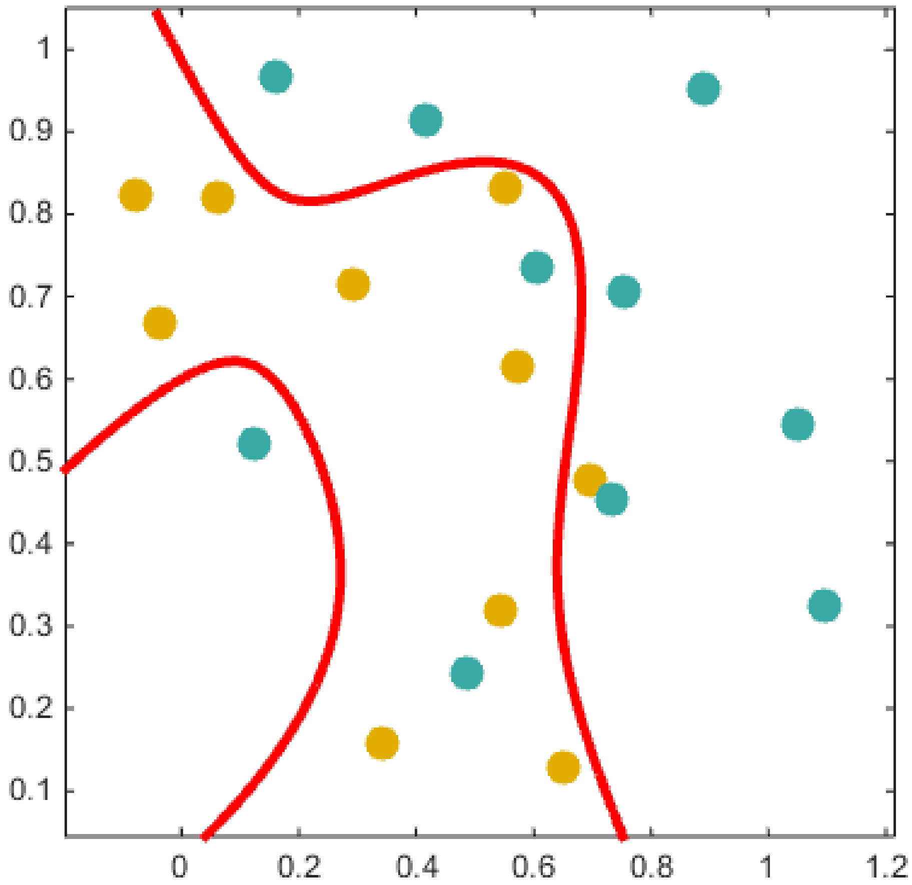
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 7.



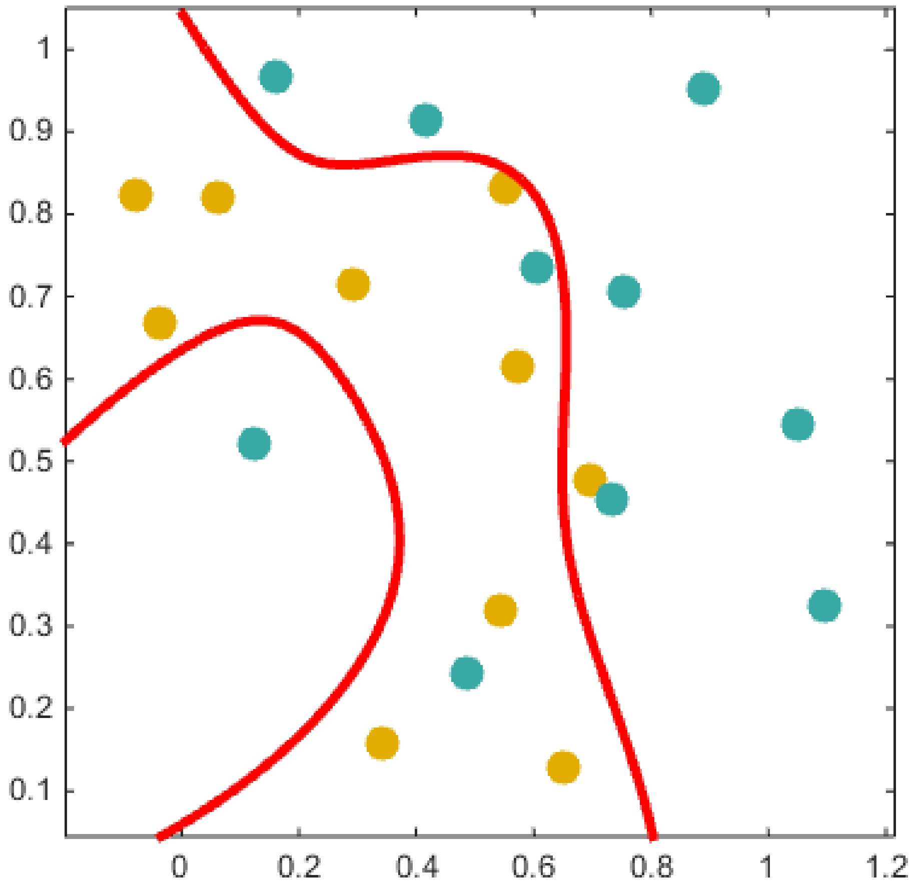
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 8.



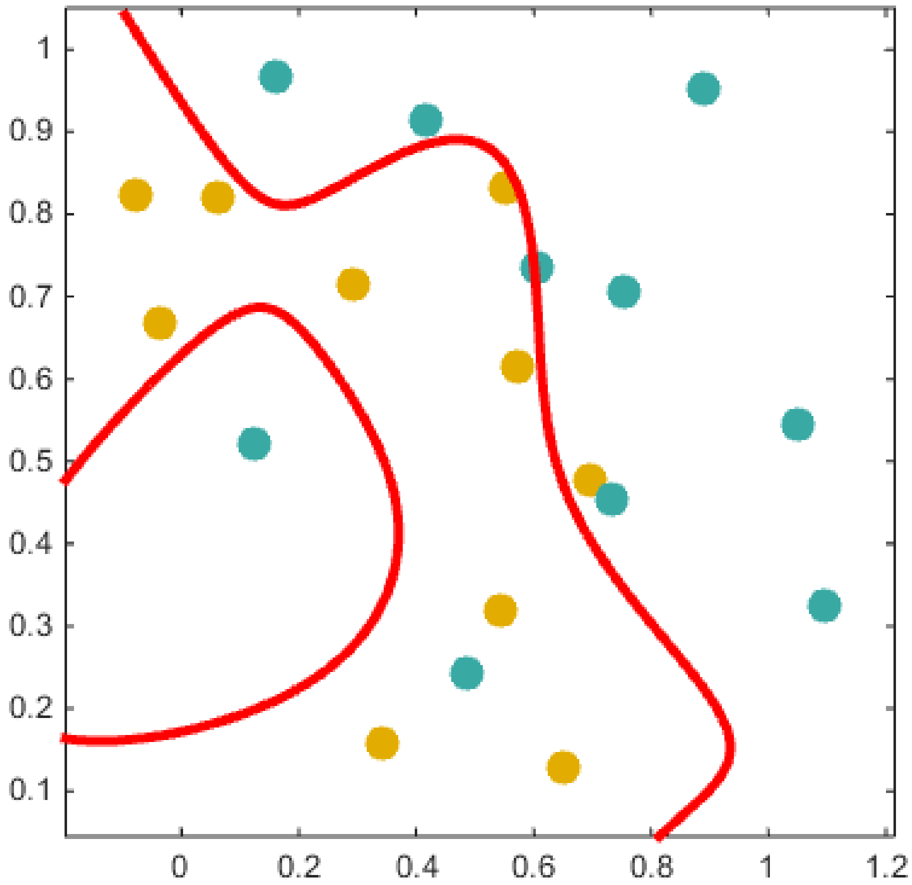
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 9.



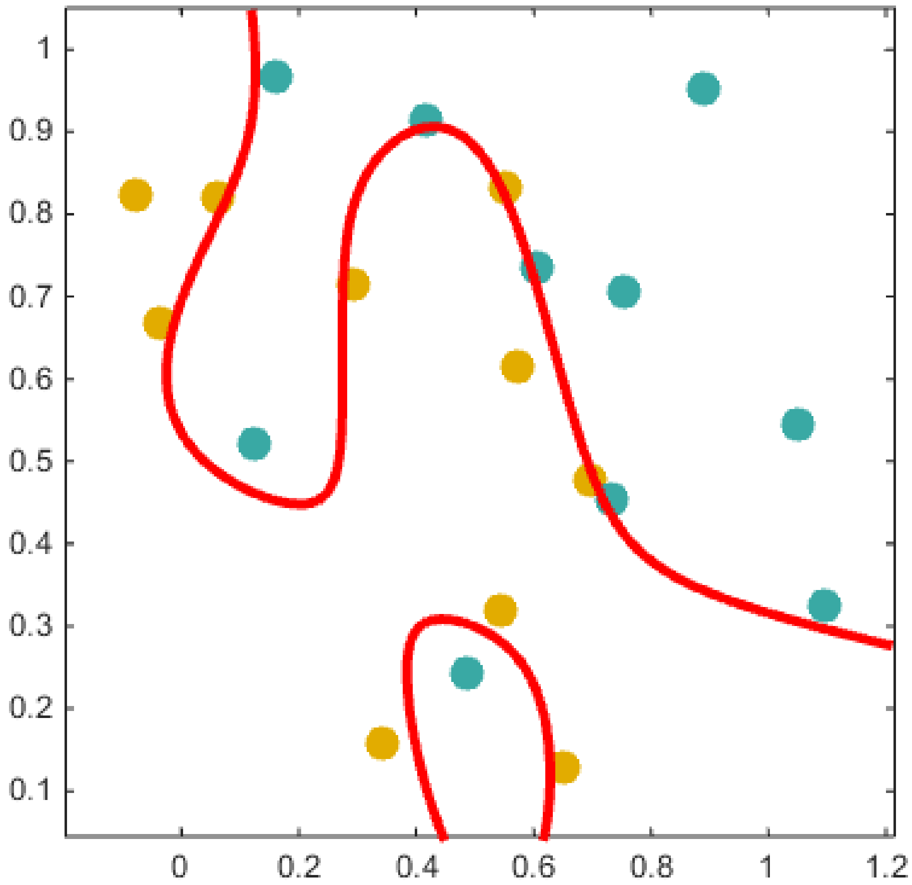
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 10.



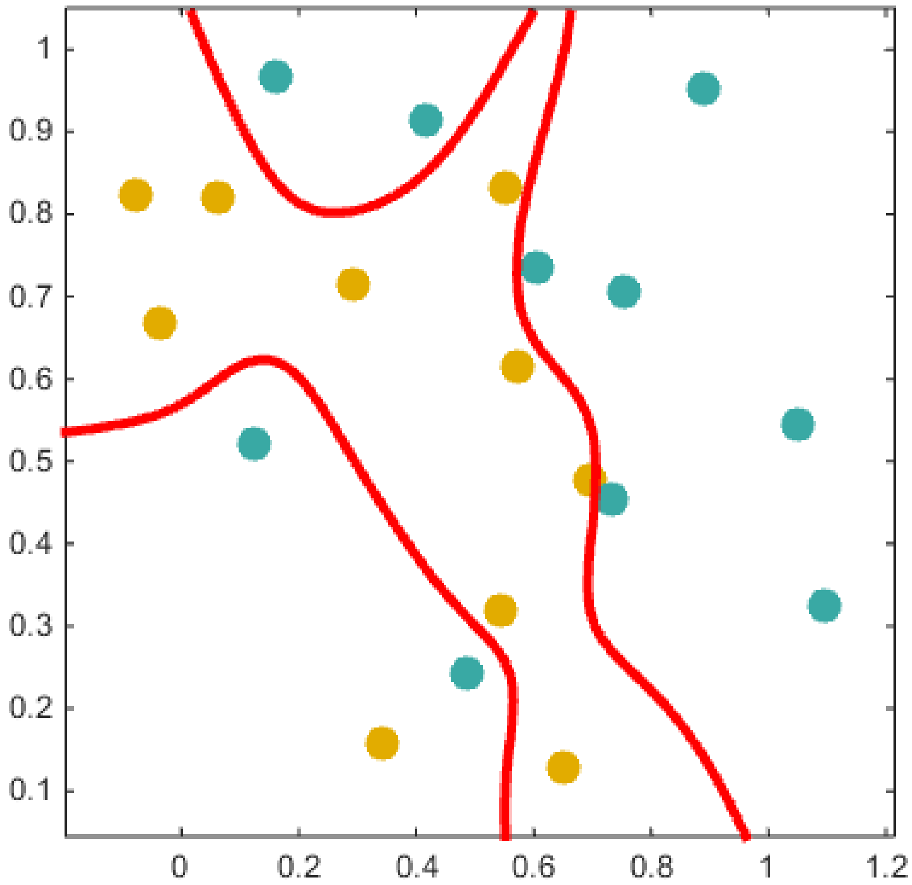
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 20.



Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 100.



RBF/Gaussian Kernels

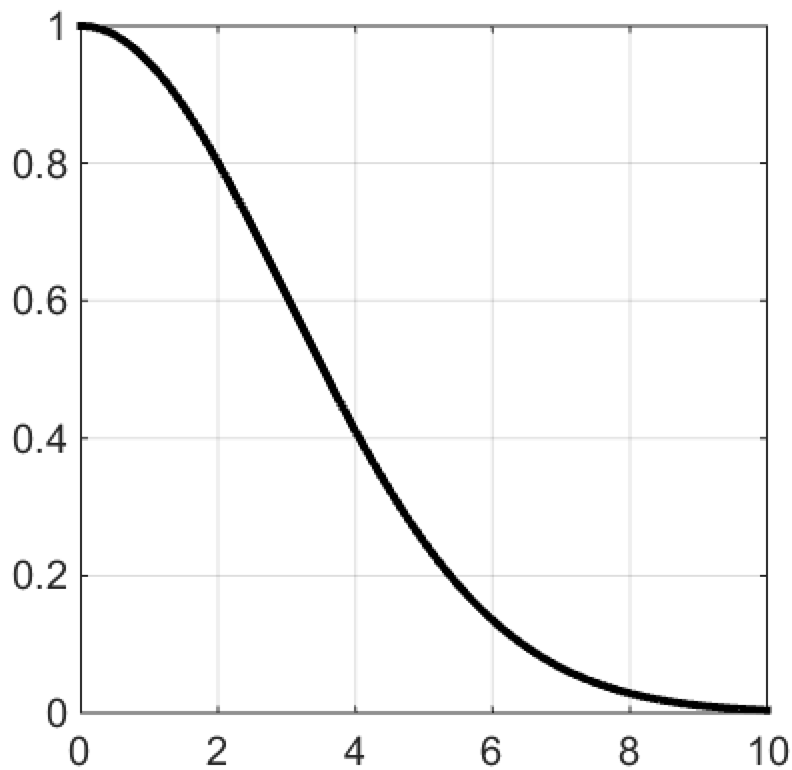
- The **Radial Basis Function (RBF) kernel**, also known as **Gaussian kernel**, is defined as:

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

- Given σ, the value of $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$ only depends on the distance between $\boldsymbol{x}$ and $\boldsymbol{z}$.
  - $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$ decreases exponentially to the distance between $\boldsymbol{x}$ and $\boldsymbol{z}$.
- Parameter σ is chosen manually.
  - Parameter σ specifies how fast $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$ decreases as $\boldsymbol{x}$ moves away from $\boldsymbol{z}$.

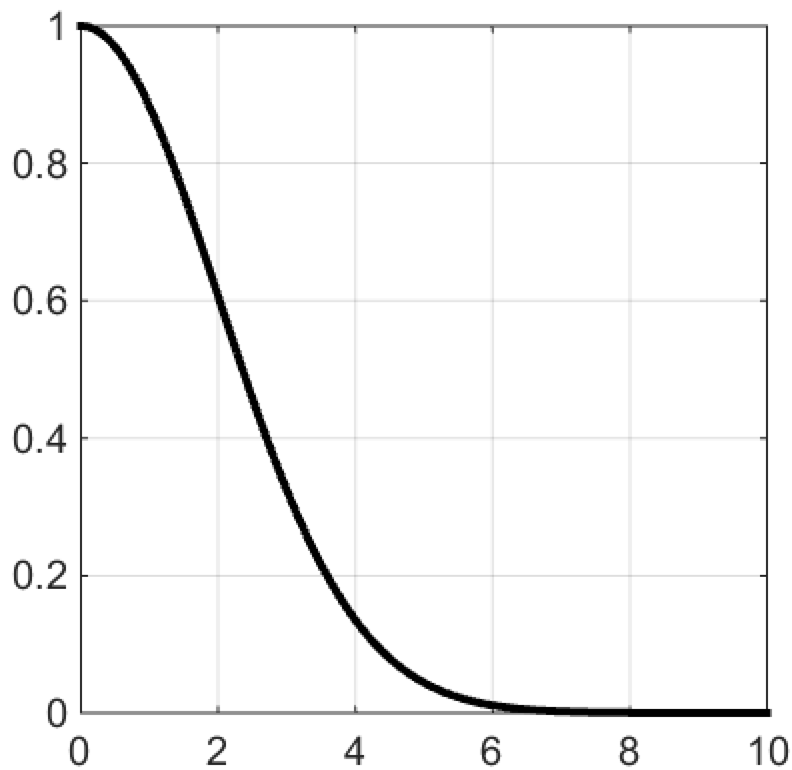
RBF Output Vs. Distance

- X axis: distance between $\boldsymbol{x}$ and $\boldsymbol{z}$.
- Y axis: $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$, with $\sigma = 3$.



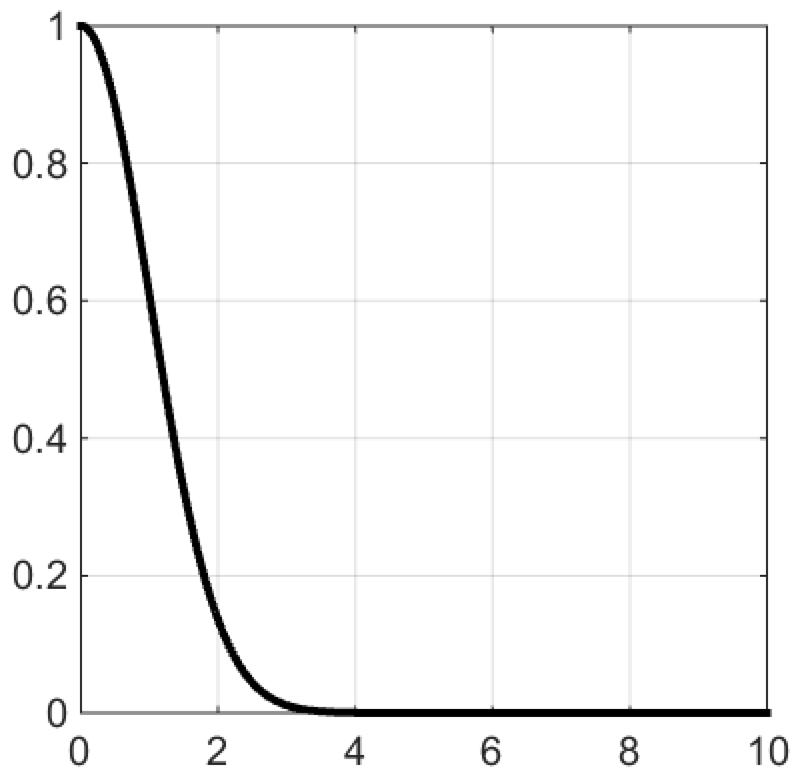
RBF Output Vs. Distance

- X axis: distance between $\boldsymbol{x}$ and $\boldsymbol{z}$.
- Y axis: $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$, with $\sigma = 2$.



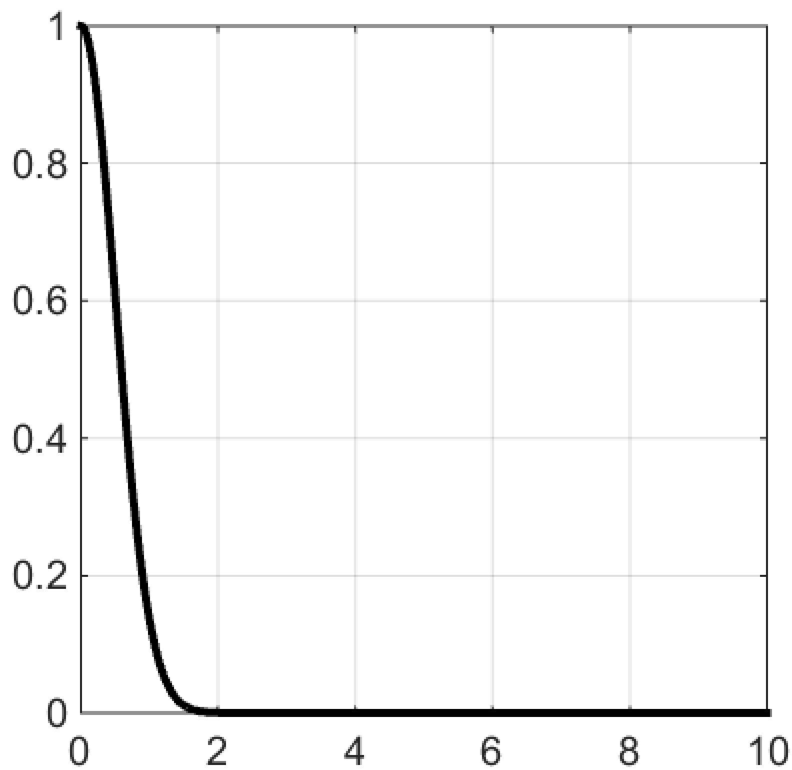
RBF Output Vs. Distance

- X axis: distance between $\boldsymbol{x}$ and $\boldsymbol{z}$.
- Y axis: $k_{\sigma}(\boldsymbol{x}, \boldsymbol{z})$, with σ =1.



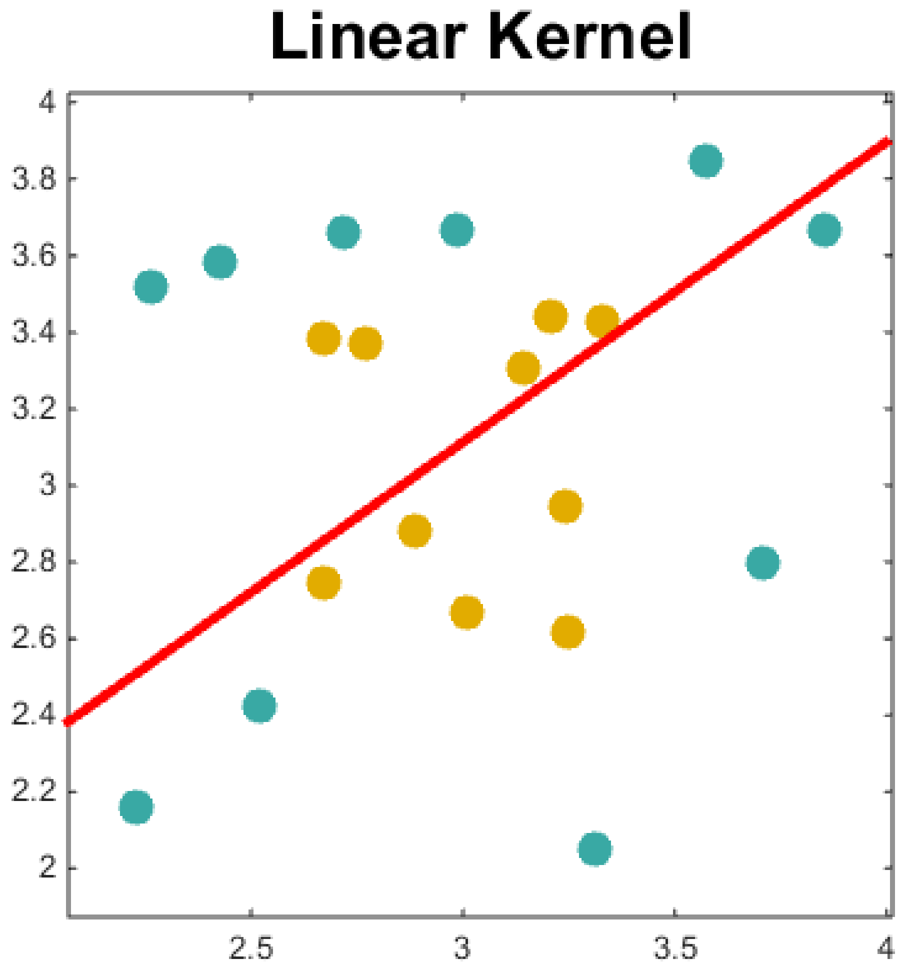
RBF Output Vs. Distance

- X axis: distance between $\boldsymbol{x}$ and $\boldsymbol{z}$.
- Y axis: $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$, with $\sigma = 0.5$.



RBF Kernels – An Easier Dataset

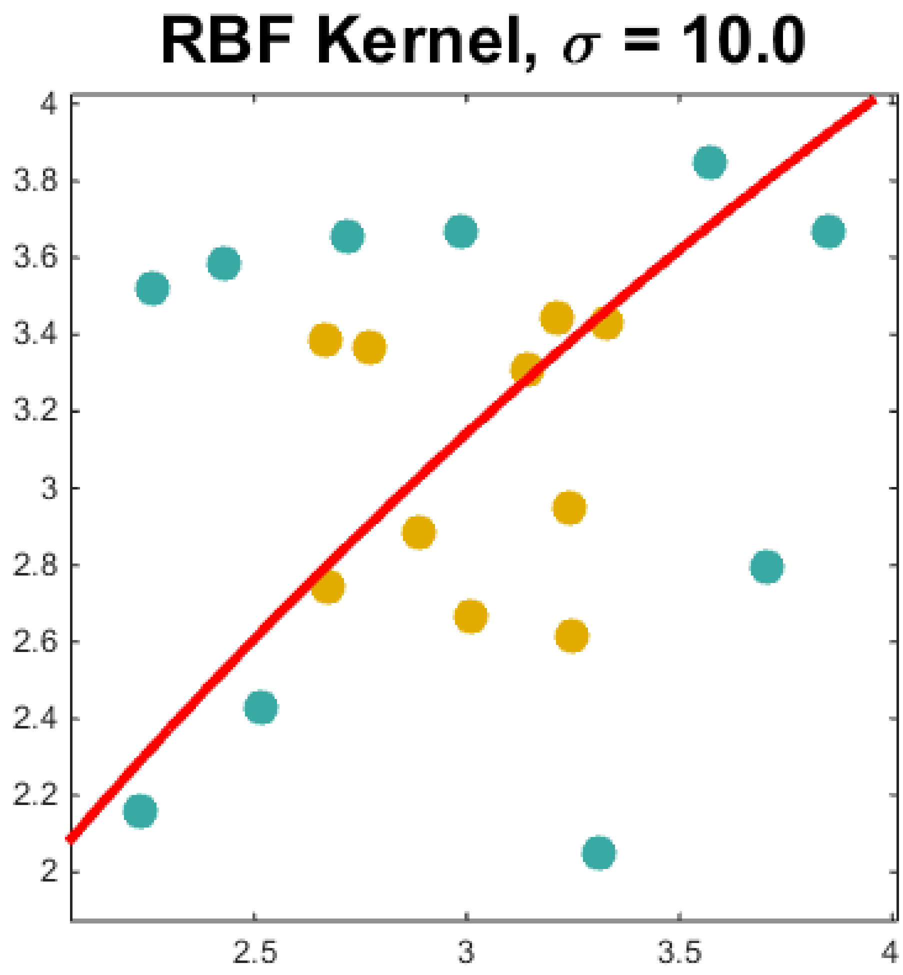
Decision boundary with a linear kernel.



RBF Kernels – An Easier Dataset

Decision boundary with an RBF kernel.

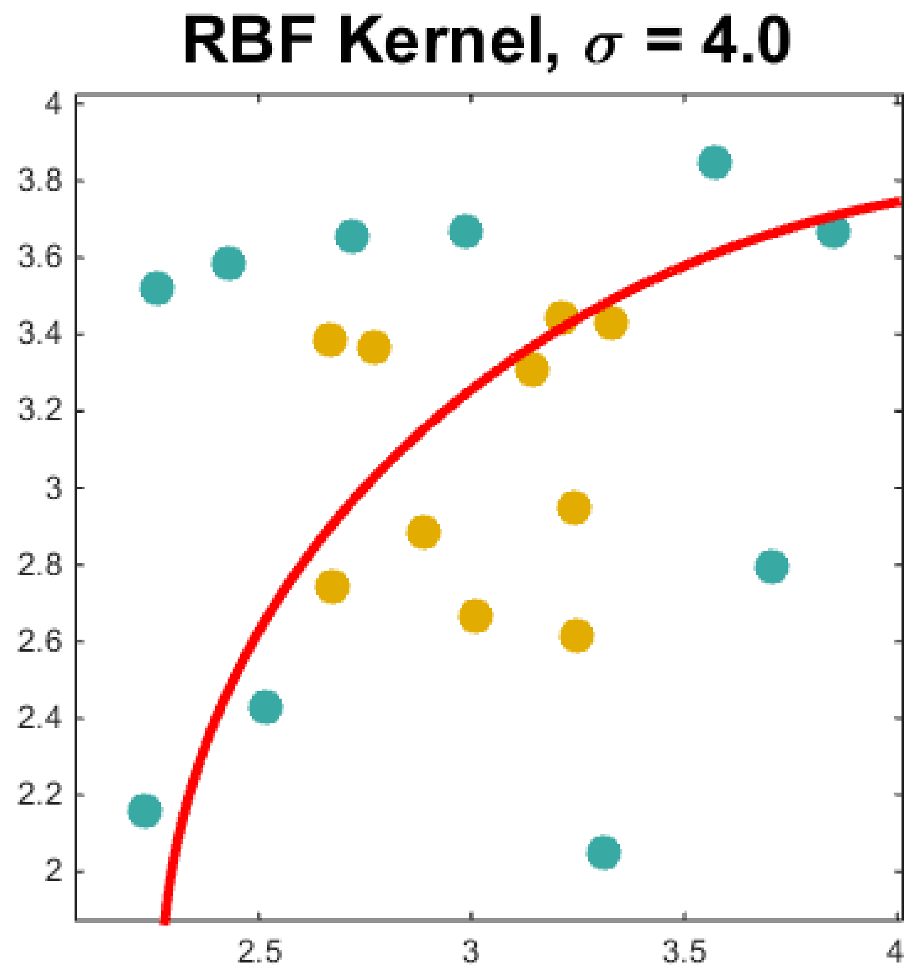
For this dataset, this is a relatively large value for σ, and it produces a boundary that is almost linear.



RBF Kernels – An Easier Dataset

Decision boundary with an RBF kernel.

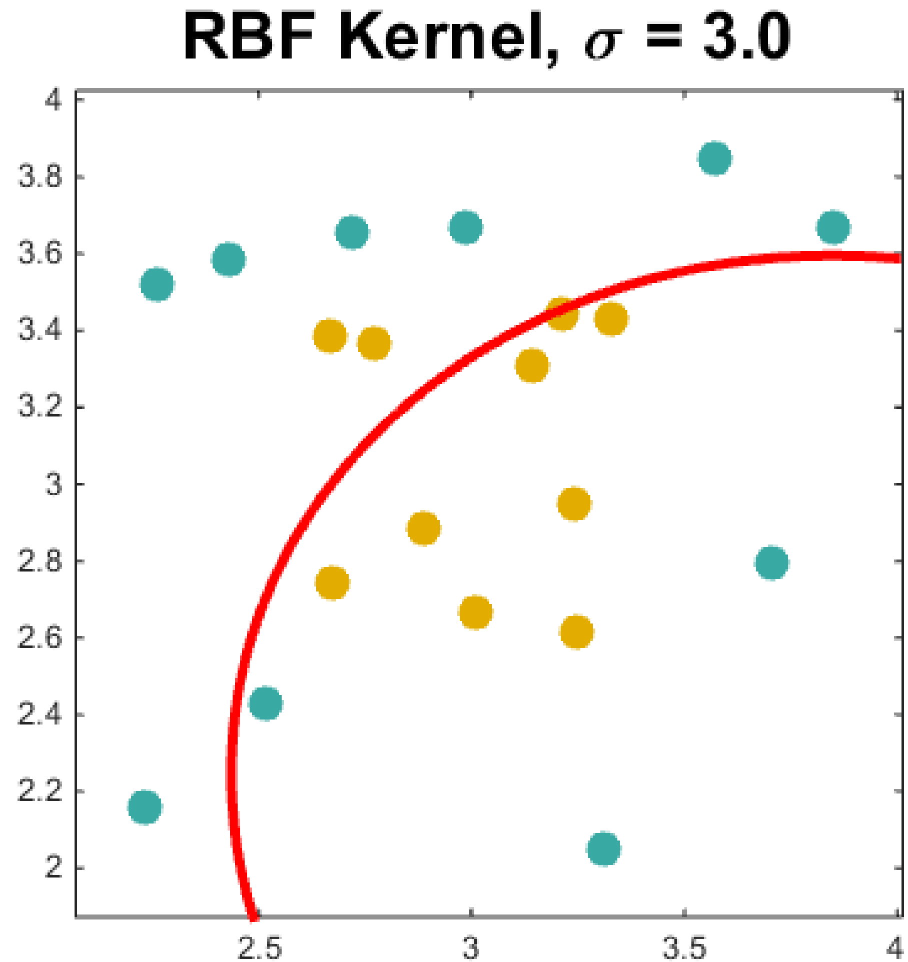
Decreasing the value of σ leads to less linear boundaries.



RBF Kernels – An Easier Dataset

Decision boundary with an RBF kernel.

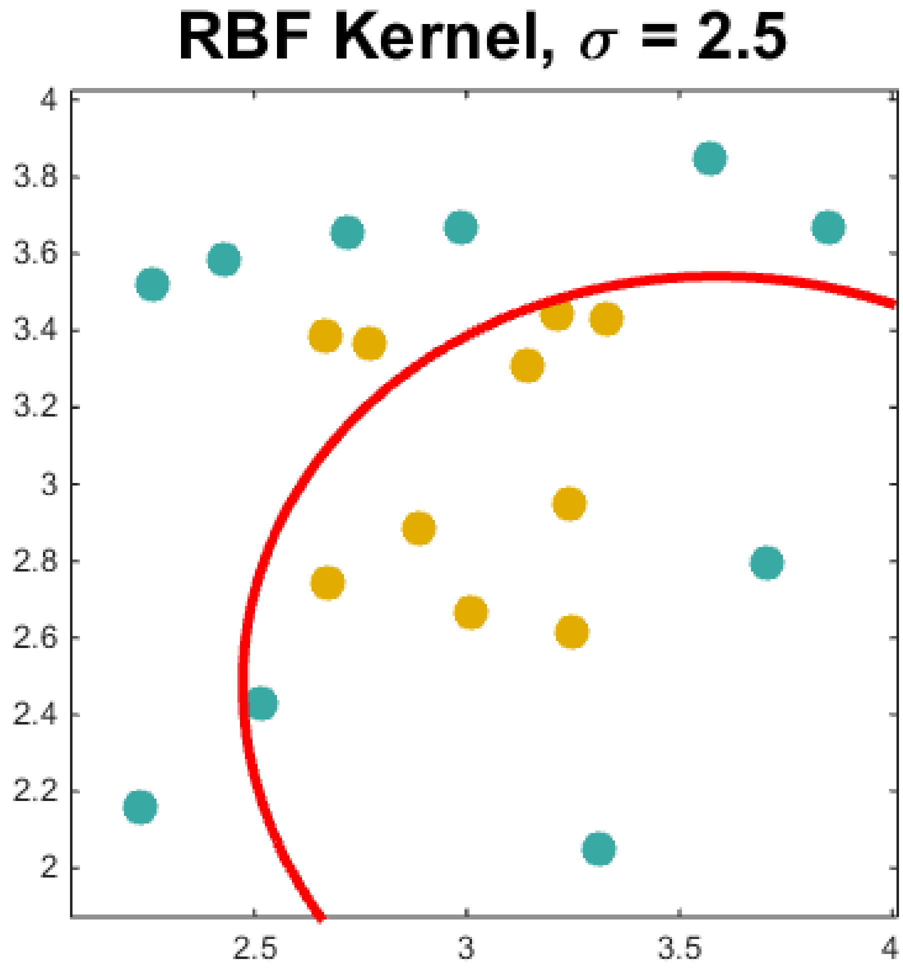
Decreasing the value of σ leads to less linear boundaries.



RBF Kernels – An Easier Dataset

Decision boundary with an RBF kernel.

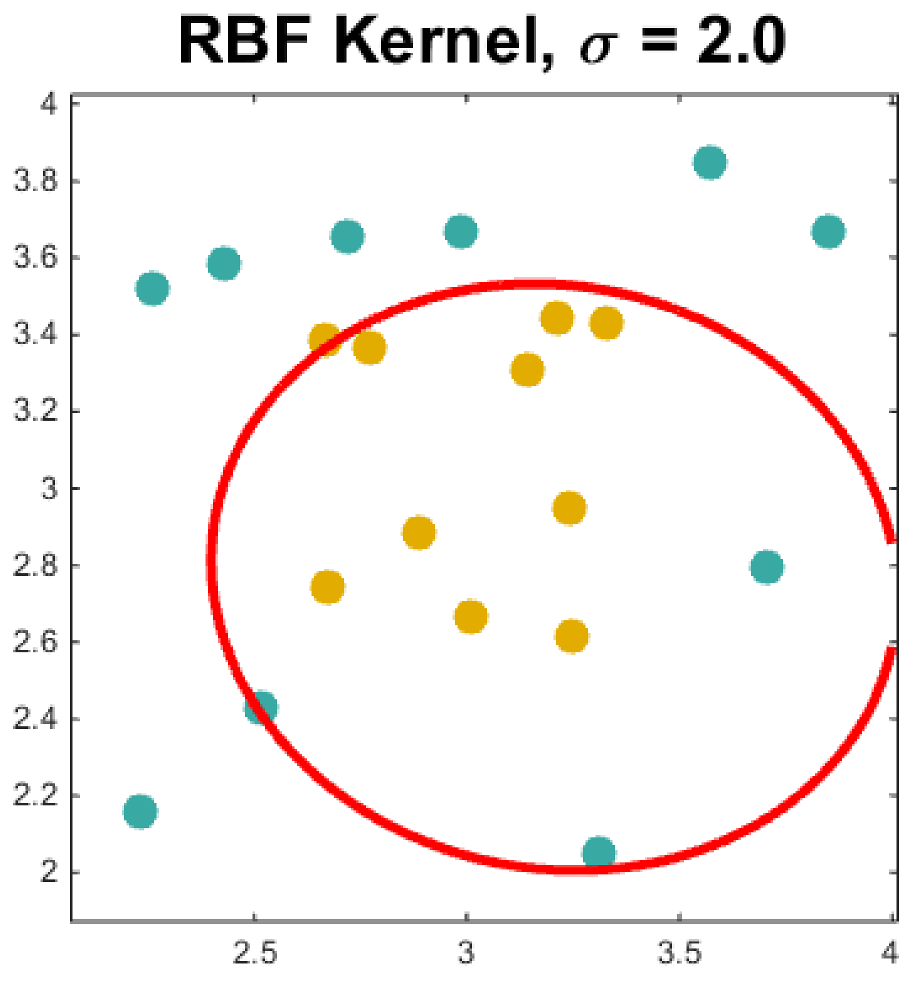
Decreasing the value of σ leads to less linear boundaries.



RBF Kernels – An Easier Dataset

Decision boundary with an RBF kernel.

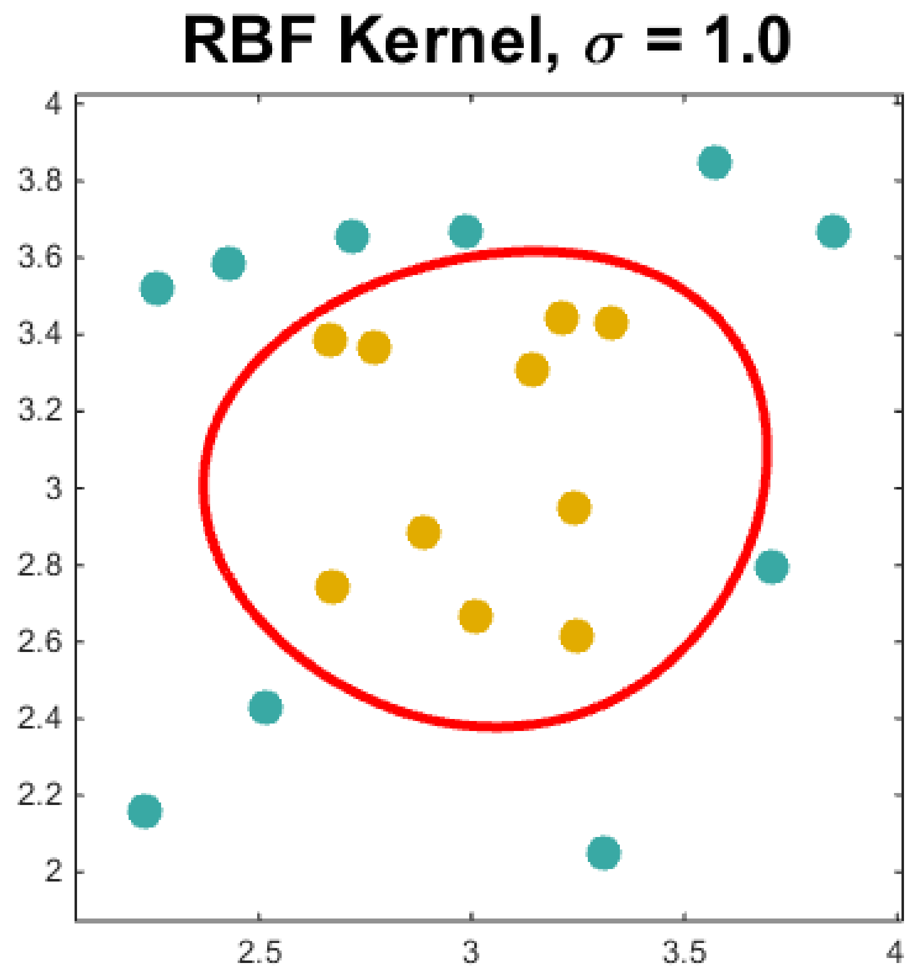
Decreasing the value of σ leads to less linear boundaries.



RBF Kernels – An Easier Dataset

Decision boundary with an RBF kernel.

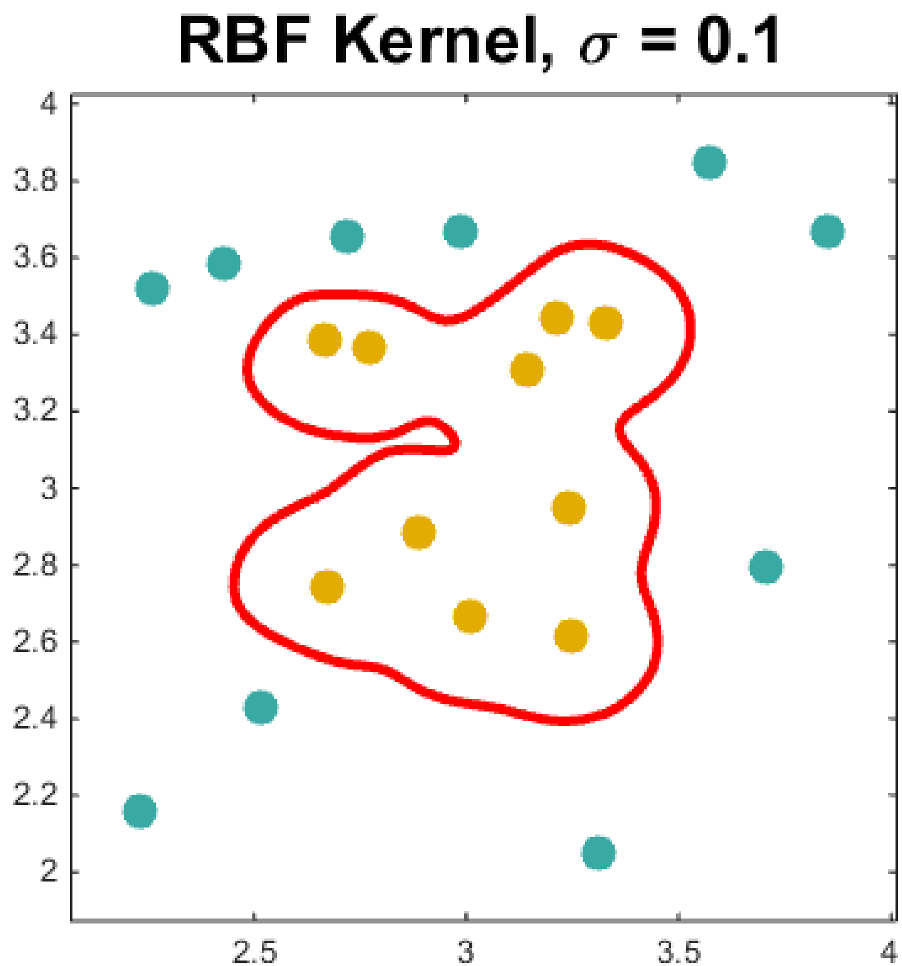
Decreasing the value of σ leads to less linear boundaries.



RBF Kernels – An Easier Dataset

Decision boundary with an RBF kernel.

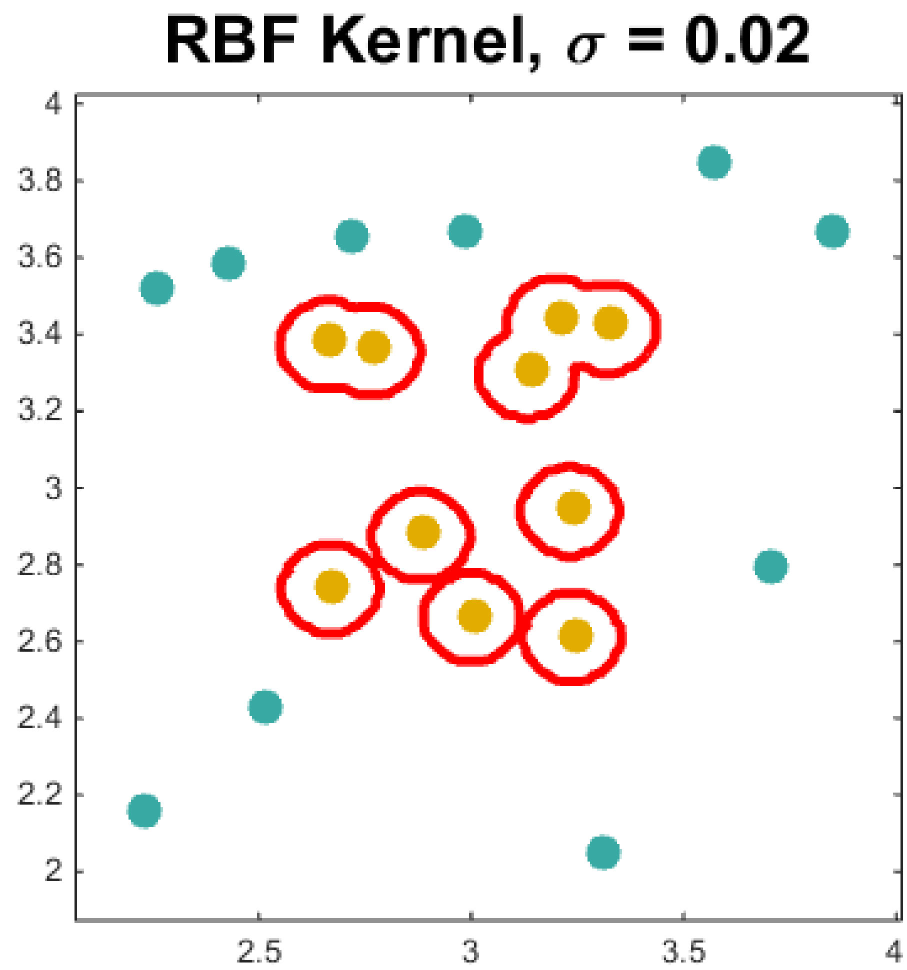
Note that smaller values of σ increase dangers of overfitting.



RBF Kernels – An Easier Dataset

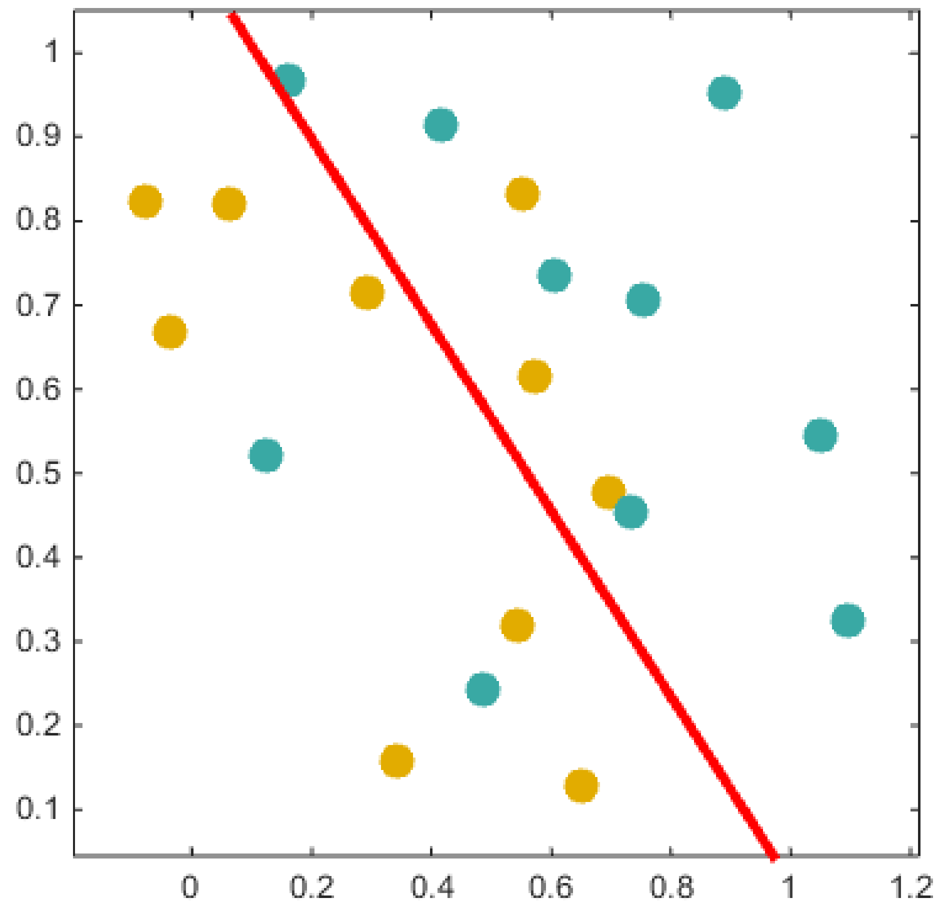
Decision boundary with an RBF kernel.

Note that smaller values of σ increase dangers of overfitting.



RBF Kernels – A Harder Dataset

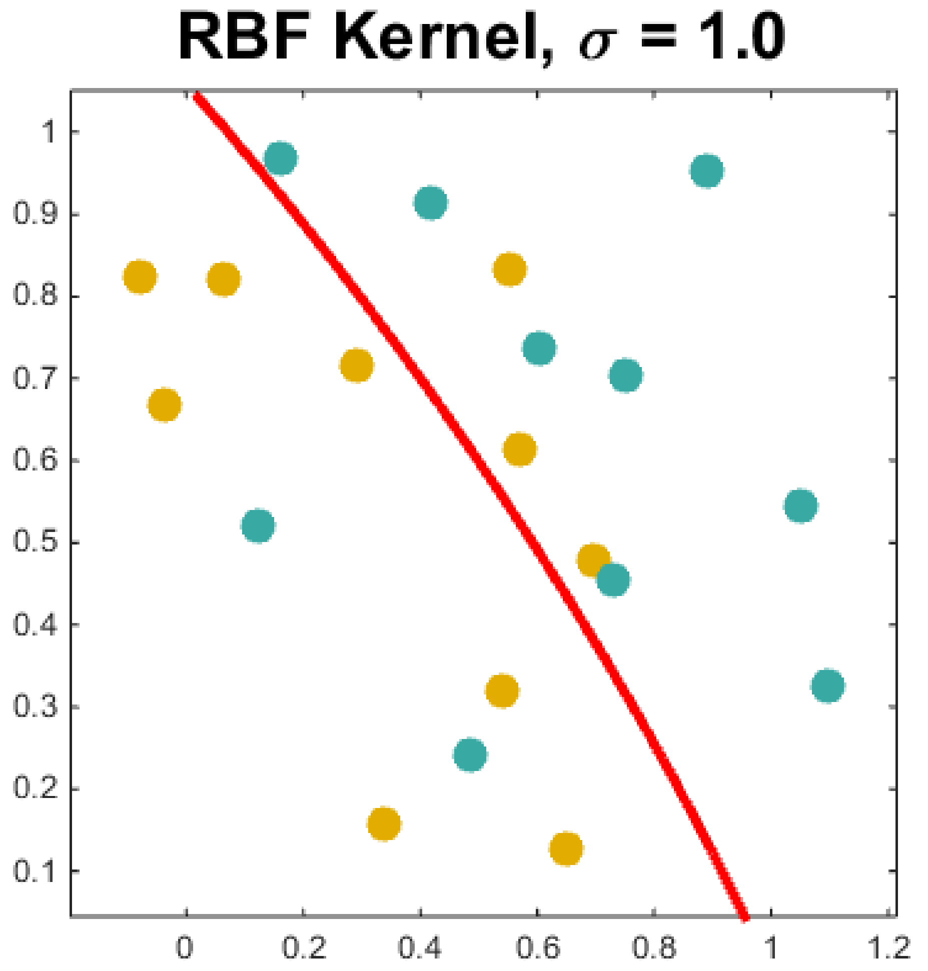
Decision boundary with a linear kernel.



RBF Kernels – A Harder Dataset

Decision boundary with an RBF kernel.

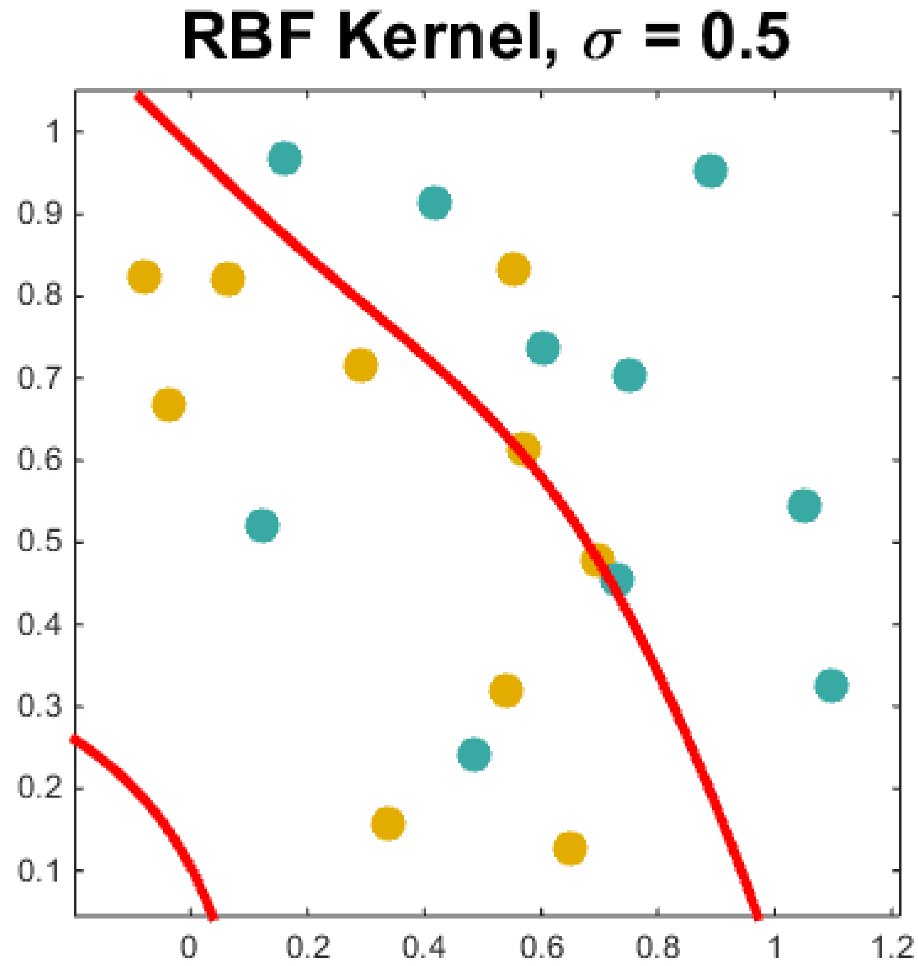
The boundary is almost linear.



RBF Kernels – A Harder Dataset

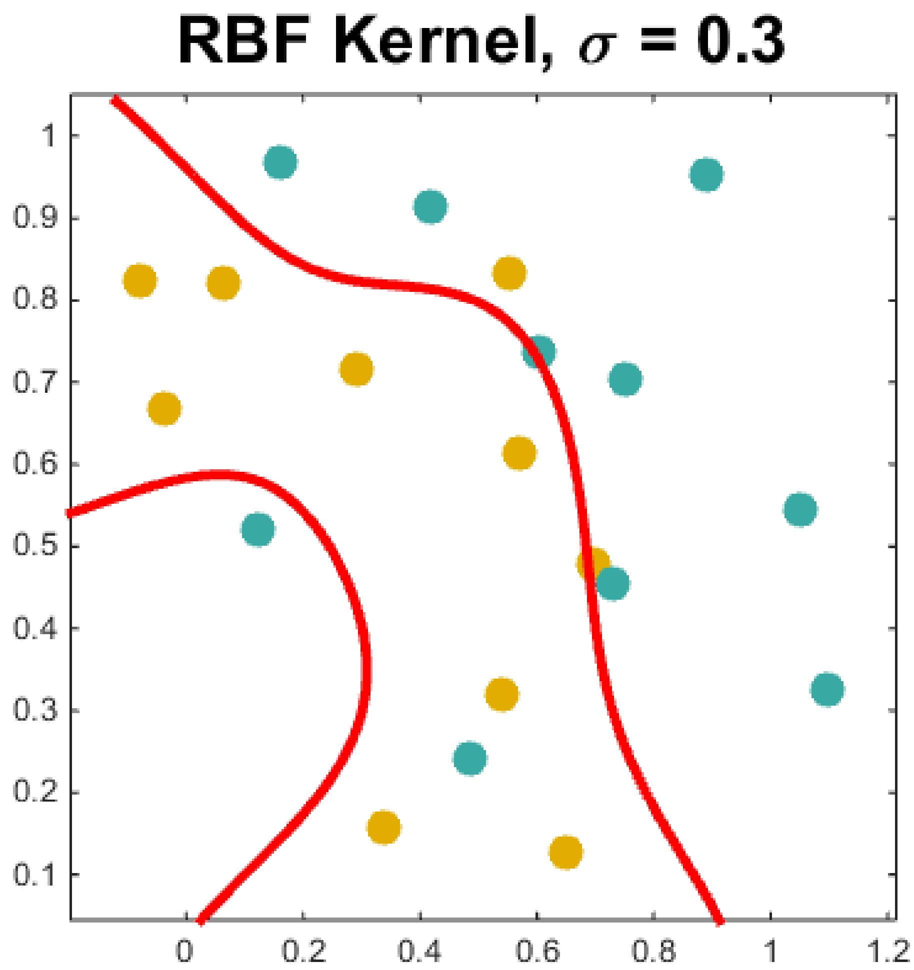
Decision boundary with an RBF kernel.

The boundary now is clearly nonlinear.



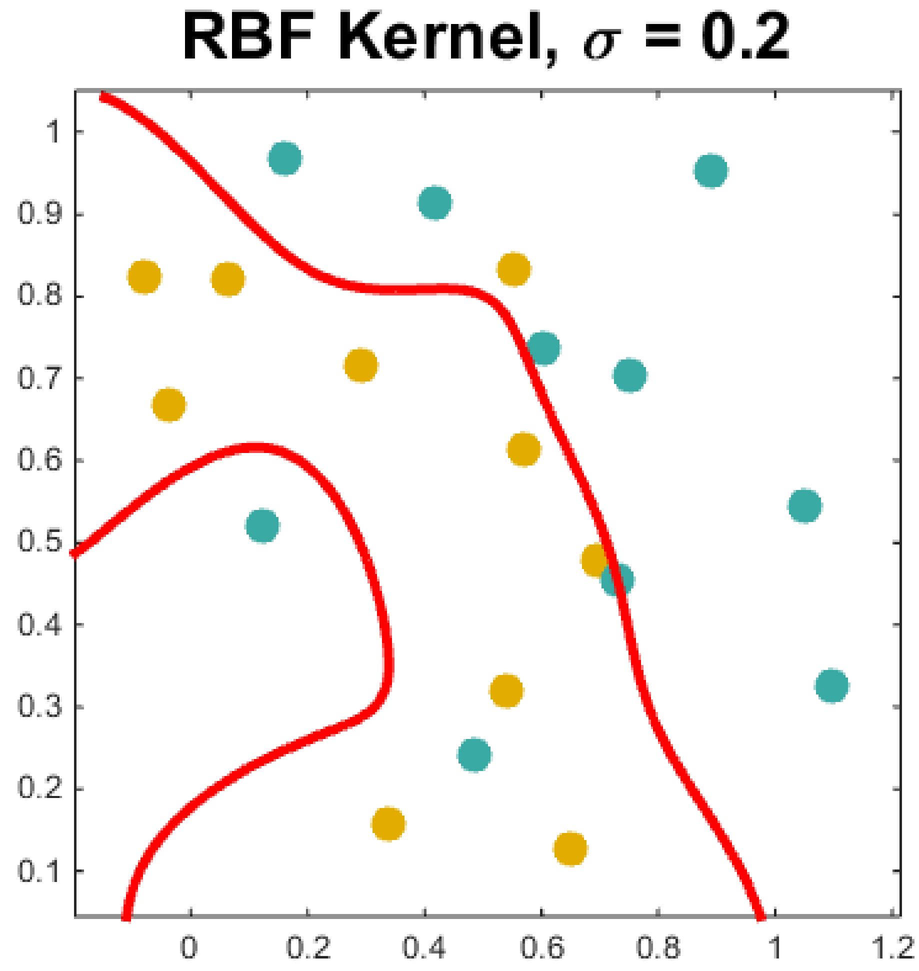
RBF Kernels – A Harder Dataset

Decision boundary with an RBF kernel.



RBF Kernels – A Harder Dataset

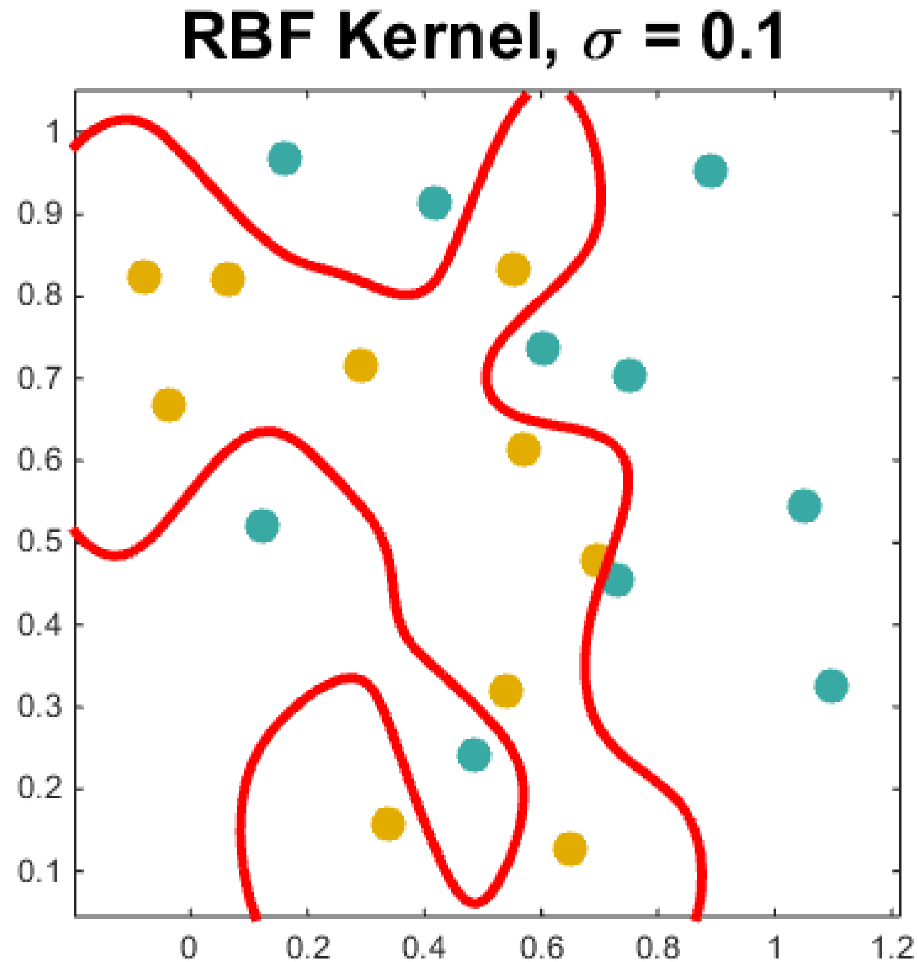
Decision boundary with an RBF kernel.



RBF Kernels – A Harder Dataset

Decision boundary with an RBF kernel.

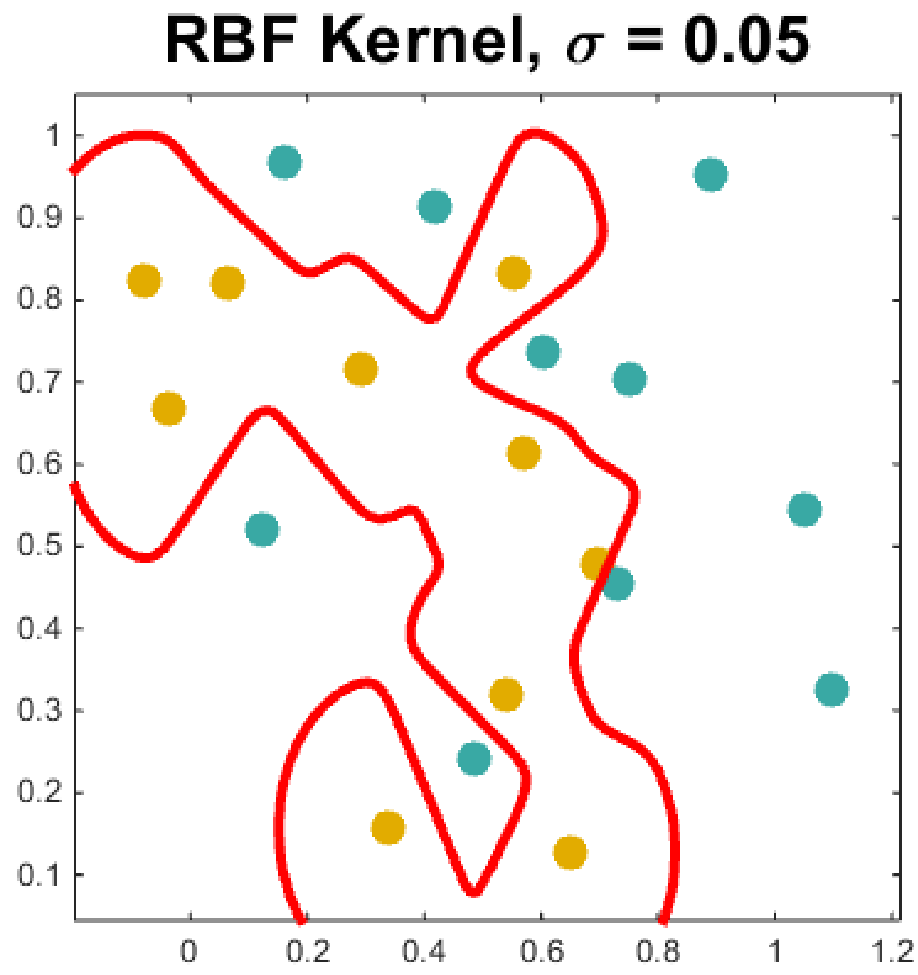
Again, smaller values of σ increase dangers of overfitting.



RBF Kernels – A Harder Dataset

Decision boundary with an RBF kernel.

Again, smaller values of σ increase dangers of overfitting.



RBF Kernels and Basis Functions

- The **RBF kernel** is defined as:

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

- Is the RBF kernel equivalent to taking the dot product in some feature space?
- In other words, is there any basis function φ such that $k_\sigma(\boldsymbol{x}, \boldsymbol{z}) = \varphi(\boldsymbol{x})^T \varphi(\boldsymbol{z})$?
- The answer is "yes":
  - There exists such a function φ, but its output is infinite-dimensional.
  - We will not get into more details here.

Kernels for Non-Vector Data

- So far, all our methods have been taking real-valued vectors as input.
  - The inputs have been elements of $\mathbb{R}^D$, for some $D \geq 1$.
- However, there are many interesting problems where the input is not such real-valued vectors.
- Examples???

Kernels for Non-Vector Data

- So far, all our methods have been taking real-valued vectors as input.
  - The inputs have been elements of $\mathbb{R}^D$, for some $D \geq 1$.
- However, there are many interesting problems where the input is not such real-valued vectors.
- The inputs can be strings, like "cat", "elephant", "dog".
- The inputs can be sets, such as {1, 5, 3}, {5, 1, 2, 7}.
  - Sets are NOT vectors. They have very different properties.
  - As sets, $\{1, 5, 3\} = \{5, 3, 1\}$.
  - As vectors, $(1, 5, 3) \neq (5, 3, 1)$
- There are many other types of non-vector data…
- SVMs can be applied to such data, as long as we define an appropriate kernel function.

Training Time Complexity

- Solving the quadratic programming problem takes $O(d^3)$ time, where d is the the dimensionality of vector $\boldsymbol{u}$.
  - In other words, d is the number of values we want to estimate.
- If we d$\boldsymbol{w}$ and b, then we estimate $D + 1$ values.
  - The time complexity is $O(D^3)$, where D is the dimensionality of input vectors $\boldsymbol{x}$ (or the dimensionality of $\varphi(\boldsymbol{x})$, if we use a basis function).
- If we use quadratic programming to compute vector $\boldsymbol{a} = (a_1, \ldots, a_N)$, then we estimate N values.
  - The time complexity is $O(N^3)$, where N is the number of training inputs.
- Which one is faster?

Training Time Complexity

- If we use quadratic programming to compute directly $\boldsymbol{w}$ and b, then the time complexity is $O(D^3)$.
  - If we use no basis function, D is the dimensionality of input vectors $\boldsymbol{x}$.
  - If we use a basis function φ, D is the dimensionality of $\varphi(\boldsymbol{x})$.
- If we use quadratic programming to compute vector $\boldsymbol{a} = (a_1, \dots, a_N)$, the time complexity is $O(N^3)$.
- For linear SVMs (i.e., SVMs with linear kernels, that use the regular dot product), usually D is much smaller than N.
- If we use RBF kernels, $\varphi(\boldsymbol{x})$ is infinite-dimensional.
  - Computing but computing vector $\boldsymbol{a}$ still takes $O(N^3)$ time.
- If you want to use the kernel trick, then there is no choice, it takes $O(N^3)$ time to do training.

SVMs for Multiclass Problems

- As usual, you can always train one-vs.-all SVMs if there are more than two classes.
- Other, more complicated methods are also available.
- You can also train what is called "all-pairs" classifiers:
  - Each SVM is trained to discriminate between two classes.
  - The number of SVMs is quadratic to the number of classes.
- All-pairs classifiers can be used in different ways to classify an input object.
  - Each pairwise classifier votes for one of the two classes it was trained on. Classification time is quadratic to the number of classes.
  - There is an alternative method, called DAGSVM, where the all-pairs classifiers are organized in a directed acyclic graph, and classification time is linear to the number of classes.

SVMs: Recap

- Advantages:
  - Training finds globally best solution.
    - No need to worry about local optima, or iterations.
  - SVMs can define complex decision boundaries.
- Disadvantages:
  - Training time is cubic to the number of training data. This makes it hard to apply SVMs to large datasets.
  - High-dimensional kernels increase the risk of overfitting.
    - Usually larger training sets help reduce overfitting, but SVMs cannot be applied to large training sets due to cubic time complexity.
  - Some choices must still be made manually.
    - Choice of kernel function.
    - Choice of parameter C in formula $C(\sum_{n=1}^{N} \xi_n) + \frac{1}{2}\|\boldsymbol{w}\|^2$.