SDSU CS 549 Spring 2024 Supervised Machine Learning Lecture 5: Support Vector Machine (SVM)

IRFAN KHAN

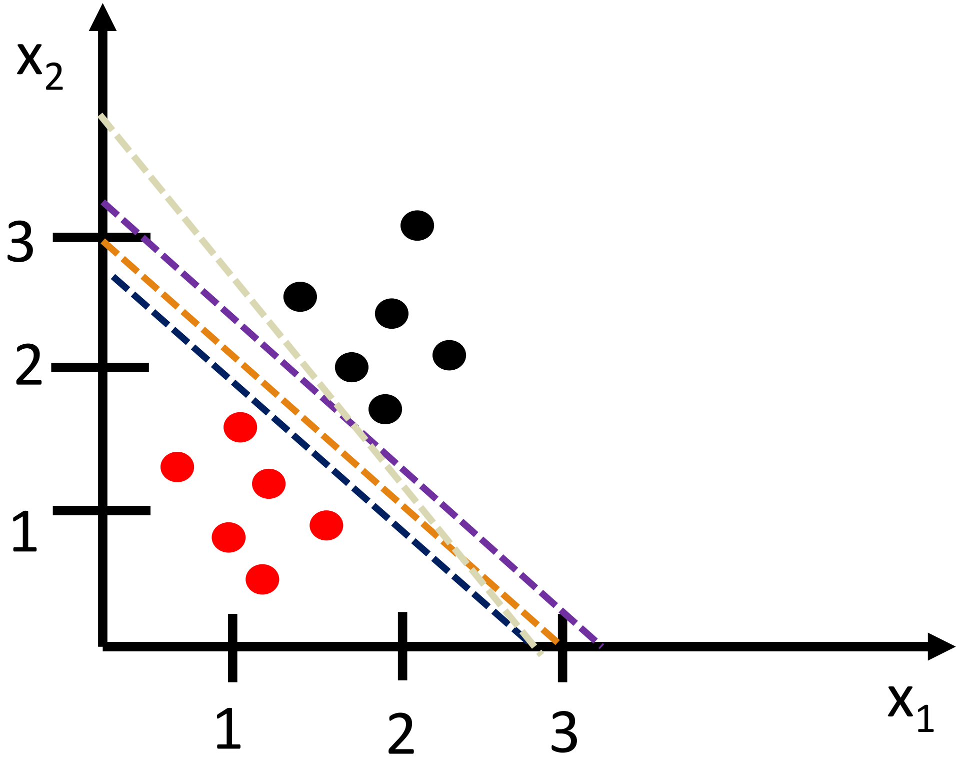
References

SDSU CS549 Lecture Notes by Prof Yang Xu, Spring 2023. Some updated slides used here

Introduction

- Linear regression and logistic regression solve unconstrained optimization problems.
- There are classes of problems that could benefit from constrained optimization, e.g.
  - A machine learning model to recognize handwritten digits (0-9) based on a set of input features extracted from images of the digits. The ***goal is to achieve high accuracy*** in classifying the digits into their corresponding classes
  - Spam vs non-spam email classifier. The ***goal is to achieve high accuracy*** in classifying emails to help prevent spam emails from reaching users' inboxes.
- Support Vector Machine (SVM) can be used here:
  - A supervised machine learning model
  - Developed by Vladimir N. Vapnik and his colleagues in the 1990s.

Recap - A Two-Feature example for Logistic Regression

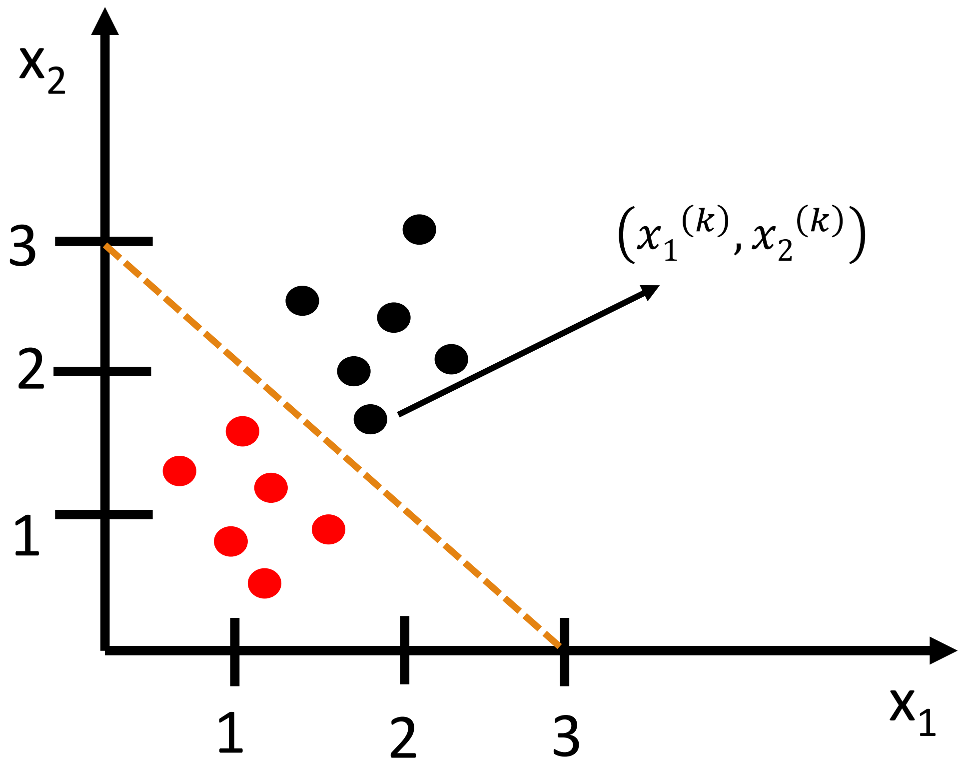


- Intermediate variable $z = w_1x_1 + w_2x_2 + b$
- Pass it through a Sigmoid Function (to make decisions on 0/1 output)
  - $g(x) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}}$
- What is the decision boundary?
  - $z = w_1x_1 + w_2x_2 + b = 0$
- For the example shown on the left, the following straight line seems to be a good decision boundary
  - $x_1 + x_2 = 3$
  - $w_1 = 1, w_2 = 1, b = -3$

- How about these new decision boundaries?
- Which is better?
- Maybe one with a wider margin since that will result in less misclassification.
  - This is the constraint!

Basic SVM

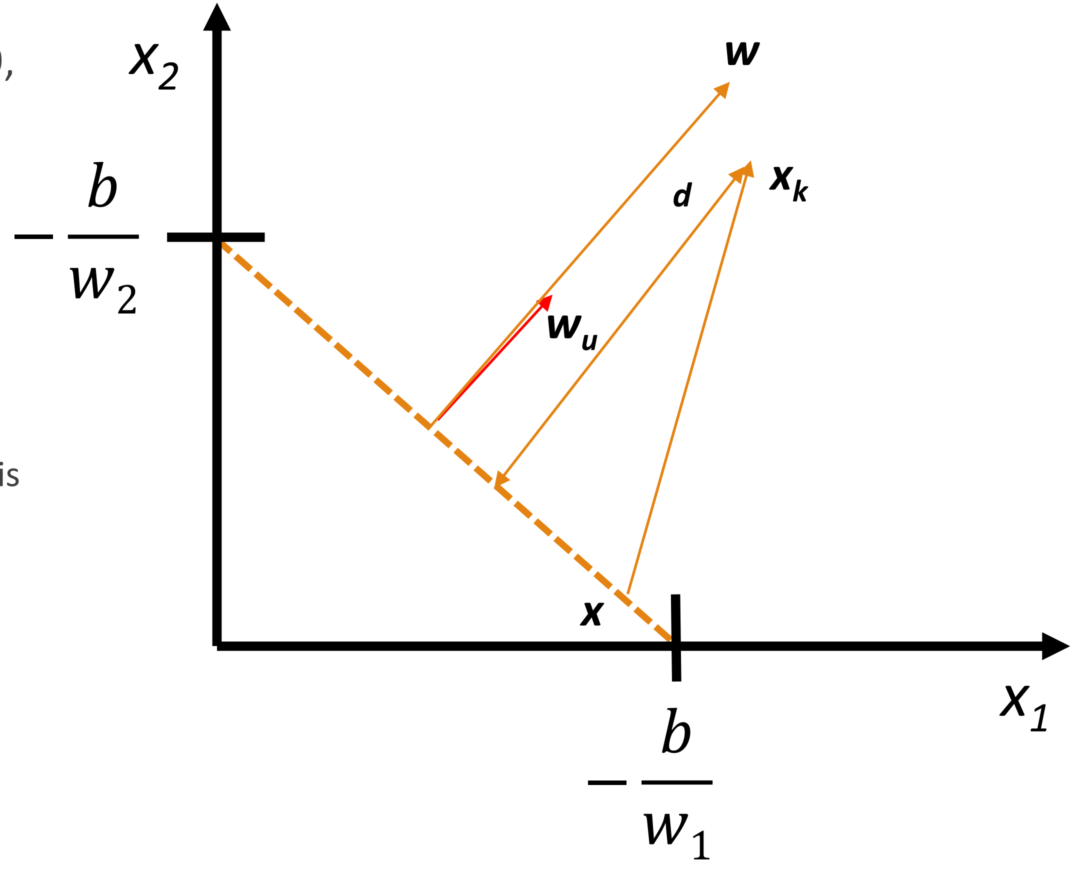
Constructing the Optimization Problem



- Decision Boundary is the straight line: $b + w_1x_1 + w_2x_2 = 0$
- For all data points $\left(x_1^{(i)}, x_2^{(i)}\right), i = 1,2, \ldots m$: $\left|b + w_1x_1^{(i)} + w_2x_2^{(i)}\right| > 0$
- We define the classifier variable, $y^{(i)}$, for each data point $\left(x_1^{(i)}, x_2^{(i)}\right)$ equal to 1 if the datapoint is over the decision line, or equal to -1 if under.
  - Using *(-1,1)* instead of *(0,1)* helps simplify the problem formulation
- Therefore, for all points, $y^{(i)}(b + w_1x_1 + w_2x_2) > 0$
- Consider the nearest point ("support vector") from the sample data points to the decision boundary, $\left(x_1^{(k)}, x_2^{(k)}\right)$
- Since there can be infinite representations of the decision boundary line, let us place the normalization constraint on the decision boundary line: $\left|b + w_1x_1^{(k)} + w_2x_2^{(k)}\right| = 1$
- Our optimization problem finds w_1, w_2, b such that the distance from $\left(x_1^{(k)}, x_2^{(k)}\right)$ to the decision boundary line is maximized.

Distance of Support Vector to the Decision Boundary

- For a straight-line decision boundary represented as $b + [w1 \quad w2]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$, in a 2D space, the vector $\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ is the Normal Vector that is perpendicular to the straight line.
  - Proof: Take any two points on the straight line, say $\boldsymbol{x_a} = \begin{bmatrix} x_{a1} \\ x_{a2} \end{bmatrix}$, and $\boldsymbol{x_b} = \begin{bmatrix} x_{b1} \\ x_{b2} \end{bmatrix}$, then $\boldsymbol{x_a} - \boldsymbol{x_b}$ represents a vector on the straight line. A dot product of this vector with $\boldsymbol{w}$ is always 0.
- The distance d from a point $\boldsymbol{x_k}$ to the straight line decision boundary is the projection of the vector ($\boldsymbol{x_k}$-$\boldsymbol{x}$), for any $\boldsymbol{x}$ on the decision boundary onto $\boldsymbol{w}$
  - $d = \frac{1}{||\boldsymbol{w}||_2} |\boldsymbol{w}^T(\boldsymbol{x_k} - \boldsymbol{x})| = \frac{1}{||\boldsymbol{w}||_2}$



SVM – Optimization Problem

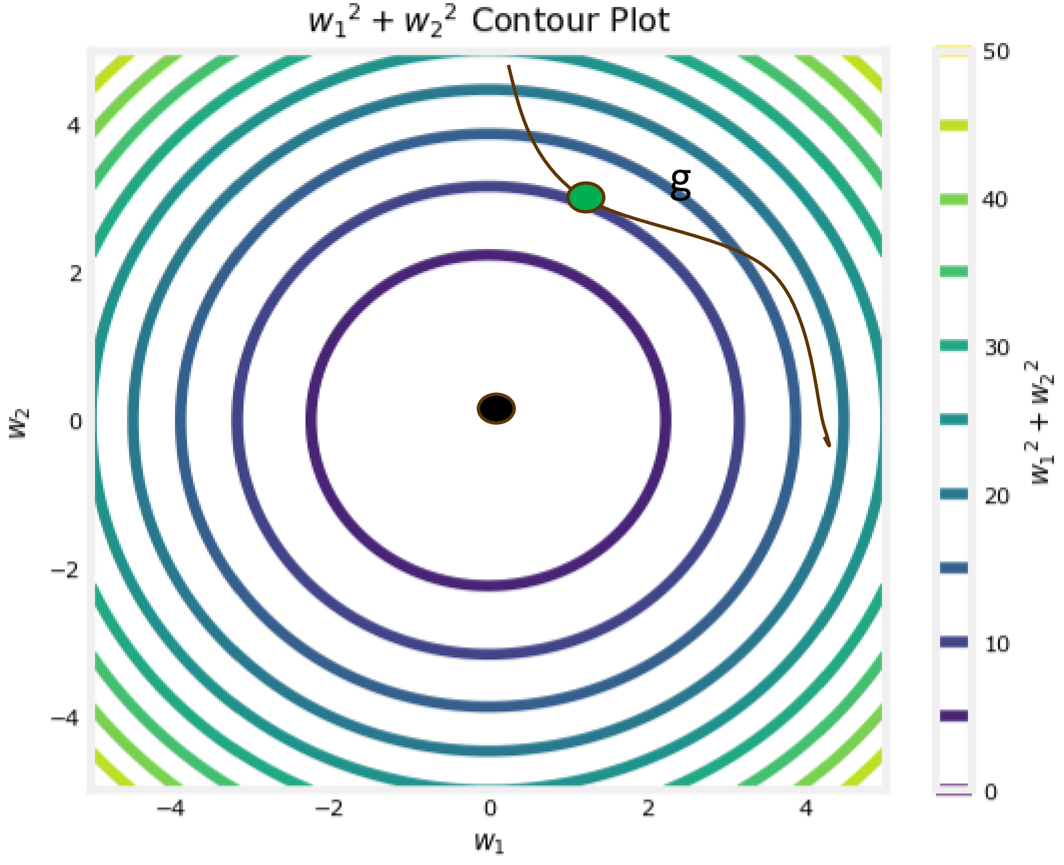
So, our optimization problem can be expressed as:

- Maximize $\frac{1}{||\boldsymbol{w}||_2}$
- Subject to the constraint:
  - $y^{(k)}\left(b + w_1 x_1{}^{(k)} + w_2 x_2{}^{(k)}\right) = 1$ for support vector $\left(x_1{}^{(k)}, x_2{}^{(k)}\right)$ and
  - $y^{(i)}\left(b + w_1 x_1{}^{(i)} + w_2 x_2{}^{(i)}\right) > 0$ for other data points $\left(x_1{}^{(i)}, x_2{}^{(i)}\right)$

$\Longleftrightarrow$

- Minimize $||w||_2$ = Minimize $\sqrt{w_1{}^2 + w_2{}^2} =$
  - Minimize $(w_1{}^2 + w_2{}^2)$ =Minimize $\frac{1}{2}(w_1{}^2 + w_2{}^2)$
  - =Minimize $\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$
- Subject to the constraint:
  - $y^{(k)}\left(b + w_1 x_1{}^{(k)} + w_2 x_2{}^{(k)}\right) = 1$ for support vector $\left(x_1{}^{(k)}, x_2{}^{(k)}\right)$ and
  - $y^{(i)}\left(b + w_1 x_1{}^{(i)} + w_2 x_2{}^{(i)}\right) > 0$ for other data points $\left(x_1{}^{(i)}, x_2{}^{(i)}\right)$

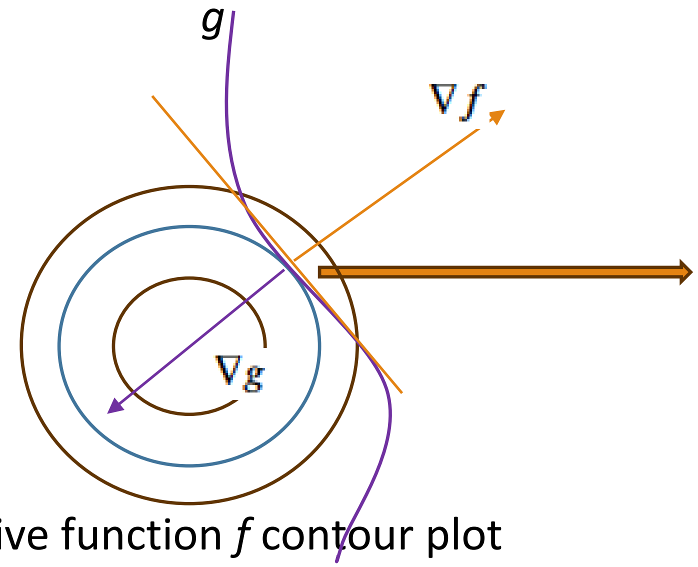
Solving the Constrained Optimization Problem



Optimal Point

- In an unconstrained minimization problem, we would seek a point at the center of the innermost circle
- The constraint set is represented by a function g implementing:
  - $y^{(k)}\left(b + w_1 x_1^{(k)} + w_2 x_2^{(k)}\right) = 1$ for support vector $\left(x_1^{(k)}, x_2^{(k)}\right)$ and
  - $y^{(i)}\left(b + w_1 x_1^{(i)} + w_2 x_2^{(i)}\right) > 0$ for other data points $\left(x_1^{(i)}, x_2^{(i)}\right)$
  - It is a summation of all the constraints
- The optimal point for the constrained minimization problem is as shown

Solving Constrained Optimization Problem (Contd.)



f contour and *g* touch point

Objective function *f* contour plot

- Objective function to be minimized $f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$
- Constraints: $g^{(i)}(\boldsymbol{w}, b) =$
  - $y^{(i)}\left(\boldsymbol{w}^T\,\boldsymbol{x}^{(i)} + b\right) - 1 \geq 0$
- At the *f* contour and *g* touch point, ∇f and ∇g point in opposite direction
- The two are related as:
  - $\nabla f(\boldsymbol{w}) = \sum_{1=1}^{m} \lambda^{(i)} \nabla g^{(i)}(\boldsymbol{w}, b)$
  - $\lambda^{(i)}$ are called Lagrange multipliers

Generalizing to any dimension

- A decision boundary in an *n*-dimensional space is a hyper-plane of *(n-1)* dimensions
  - Previous slides explored *n=2* because it is easier to visualize
- All the concepts carry over to an *n*-dimensional problem
  - Intermediate variable $z = \sum_j w_j {x_j}^{(i)} + b, j = 1,2, \ldots n$
  - Instead of classifying datapoints as 0/1, we classify them as -1/1
  - Hence $y^{(i)} \sum_j w_j {x_j}^{(i)} + b$ is always >0 for all non-support vectors
  - We apply the normalizing constraint for the support vectors $\left| \sum_j w_j \, {x_j}^{(k)} + b \right| = 1$
  - As before, the objective function to be minimized is $\frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}$, subject to $g^{(i)}(\boldsymbol{w}, b) = y^{(i)} \left(\boldsymbol{w}^T \, \boldsymbol{x}^{(i)} + b \right) - 1 \geq 0$

The Lagrangian and its Optimization

- We define the Lagrangjian as:
  - $L(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$ - $\sum_{1=1}^{m} \lambda^{(i)} g^{(i)}(\boldsymbol{w}, b)$, $\lambda^{(i)} \geq 0$
    - Satisfies the gradient intuition we developed earlier
  - $= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$ - $\sum_{1=1}^{m} \lambda^{(i)}(y^{(i)}\left(\boldsymbol{w}^T\, \boldsymbol{x}^{(i)} + b\right) - 1)$
- We minimize *L* wrt ***w*** and *b* and maximize *L* wrt $\lambda^{(i)}$
- Minimize L wrt ***w*** and *b*
  - $\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{1=1}^{m} \lambda^{(i)} y^{(i)} \boldsymbol{x}^{(i)} \quad = 0$
  - $\frac{\partial L}{\partial b} = -\sum_{1=1}^{m} \lambda^{(i)} y^{(i)} = 0$
  - =>$\boldsymbol{\lambda}.\, \boldsymbol{y} = 0$
- In $L(\boldsymbol{w}, b, \boldsymbol{\lambda})$, substituting $\boldsymbol{w} = \sum_{i=1}^{m} \lambda^{(i)} y^{(i)} \boldsymbol{x}^{(i)}$ in L and utilizing $\sum_{i=1}^{m} \lambda^{(i)} y^{(i)} = 0$
- $L(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \lambda^{(i)}\lambda^{(j)} y^{(i)} y^{(j)} \boldsymbol{x}^{(i)} . \boldsymbol{x}^{(j)} - \sum_{i=1}^{m}\sum_{j=1}^{m} \lambda^{(i)}\lambda^{(j)} y^{(i)} y^{(j)} \boldsymbol{x}^{(i)} . \boldsymbol{x}^{(j)} + \sum_{i=1}^{m} \lambda^{(i)}$
  - $-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \lambda^{(i)}\lambda^{(j)} y^{(i)} y^{(j)} \boldsymbol{x}^{(i)} . \boldsymbol{x}^{(j)} + \sum_{i=1}^{m} \lambda^{(i)}$
- *L* thus simplifies to $L(\boldsymbol{\lambda})$ a quadratic in only one variable $\boldsymbol{\lambda}$

The Lagrangian and its Optimization (Contd.)

- $L(\boldsymbol{\lambda}) = -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\lambda^{(i)}\lambda^{(j)}y^{(i)}y^{(j)}\boldsymbol{x}^{(i)}.\boldsymbol{x}^{(j)} + \sum_{i=1}^{m}\lambda^{(i)}$

- Maximizing *L* ⇔ Minimizing *L* given by:
- $L(\boldsymbol{\lambda}) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\lambda^{(i)}\lambda^{(j)}y^{(i)}y^{(j)}\boldsymbol{x}^{(i)}.\boldsymbol{x}^{(j)} - \sum_{i=1}^{m}\lambda^{(i)}$
- $= \frac{1}{2}\boldsymbol{\lambda}^T \begin{bmatrix} y^{(1)}y^{(1)}\boldsymbol{x}^{(1)}.\boldsymbol{x}^{(1)} & y^{(1)}y^{(m)}\boldsymbol{x}^{(1)}.\boldsymbol{x}^{(m)} \\ \dots. & \dots. \\ y^{(m)}y^{(1)}\boldsymbol{x}^{(m)}.\boldsymbol{x}^{(1)} & y^{(m)}y^{(m)}\boldsymbol{x}^{(m)}.\boldsymbol{x}^{(m)} \end{bmatrix}\boldsymbol{\lambda} - (\boldsymbol{1^T})\boldsymbol{\lambda}$
- Denote the matrix as $\boldsymbol{Q}$
- We can write our optimization problem in a compact form as:
- Minimize $\frac{1}{2}\boldsymbol{\lambda}^T\boldsymbol{Q}\,\boldsymbol{\lambda} - (\boldsymbol{1^T})\boldsymbol{\lambda}$ wrt $\boldsymbol{\lambda}$
  - $\boldsymbol{y^T}\boldsymbol{\lambda} = \boldsymbol{0}, \lambda^{\boldsymbol{i}} \geq 0$
  - Known as Quadratic programming optimization problem

Obtaining *w* and *b*

- Once we obtain $\boldsymbol{\lambda} = \begin{bmatrix} \lambda^{(1)} \\ \dots \\ \lambda^{(m)} \end{bmatrix}$
- We can obtain $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \dots \\ w_n \end{bmatrix} = \sum_{i=1}^{m} \lambda^{(i)} y^{(i)} \boldsymbol{x}^{(i)}$
- Majority of $\lambda^{(i)}$ are 0, the few that are >0 correspond to the support vectors
- So, $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \dots \\ w_n \end{bmatrix} = \sum_{\boldsymbol{x}^{(i)} \,\varepsilon\, SV} \lambda^{(i)} y^{(i)} \boldsymbol{x}^{(i)}$
- *b* can be obtained using one of the support vectors $\boldsymbol{x}^{(k)}$ from:
  - $\left| b + w_1 {x_1}^{(k)} + w_2 {x_2}^{(k)} \right| = 1$

Recap of the Steps

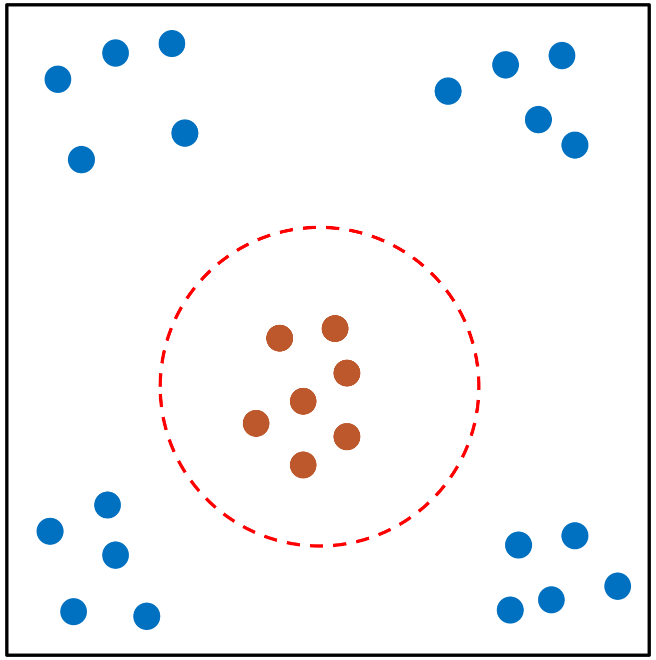
- We want to find the decision boundary $\sum_j w_j \boldsymbol{x}_j^{(i)} + b = 0, j = 1,2, \ldots n$
  - Such that the distance to the support vectors is large
- Instead of 0/1 classifier variable, we use -1/1 and we apply normalization constraint for the support vector $(\boldsymbol{x}^{(k)})$
  - $y^{(k)}(b + \sum_j w_j x_j^{(k)}) = 1$ for support vector $(\boldsymbol{x}^{(k)})$ and
  - $y^{(i)}(b + \sum_j w_j x_j^{(i)}) \geq 0$ for other data points $(\boldsymbol{x}^{(i)})$
- The objective function that maximizes distance of the decision boundary to the support vector(s) simplifies to minimizing $\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$ subject to the constraints $y^{(i)}(\boldsymbol{w}^T\, \boldsymbol{x}^{(i)} + b) - 1 \geq 0$
- Then we construct a Lagranjian $L(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$ - $\sum_{1=1}^{m} \lambda^{(i)} g^{(i)}(\boldsymbol{w}, b)\,, \lambda^{(i)} \geq 0$ to minimize
- Which results in a Quadratic optimization problem which can be solved using off-the-shelf libraries:
  - Minimize $\frac{1}{2}\boldsymbol{\lambda}^T\boldsymbol{Q}\,\boldsymbol{\lambda} - (\boldsymbol{1}^T)\boldsymbol{\lambda}$ wrt $\boldsymbol{\lambda}$
  - $\boldsymbol{y}^T\boldsymbol{\lambda} = \boldsymbol{0}, \lambda^i \geq 0$
- Majority of $\lambda^{(i)}$ are 0, the few that are >0 correspond to the support vectors
- So, $\boldsymbol{w} = \begin{bmatrix} w1 \\ \ldots \\ w_n \end{bmatrix} = \sum_{\boldsymbol{x}^{(i)}\,\varepsilon\, SV} \lambda^{(i)} y^{(i)} \boldsymbol{x}^{(i)}$
- b can be obtained using one of the support vectors $\boldsymbol{x}^{(k)}$ from:
  - $|b + \sum_j w_j x_j^{(k)}| = 1$
- Hence, we have the parameters for the decision boundary

Implementation of QP and SVM

- We use **cvxopt** for the assignment
- **cvxopt** is a Python package for solving optimization problems including quadratic programming problems.
- You can install it using the following command:
  - `conda install -c conda-forge cvxopt`
- **cvxopt** requires the matrices to be **cvxopt matrix**
- From **cvxopt** we use **matrix** to convert our usual numpy matrices to the cvxopt matrices and the quadratic programming optimization function **solvers**
- The scikit-learn library is a widely used machine learning library that includes a built-in implementation of Support Vector Machines (SVM) for classification
  - It can be used directly to solve SVM problems

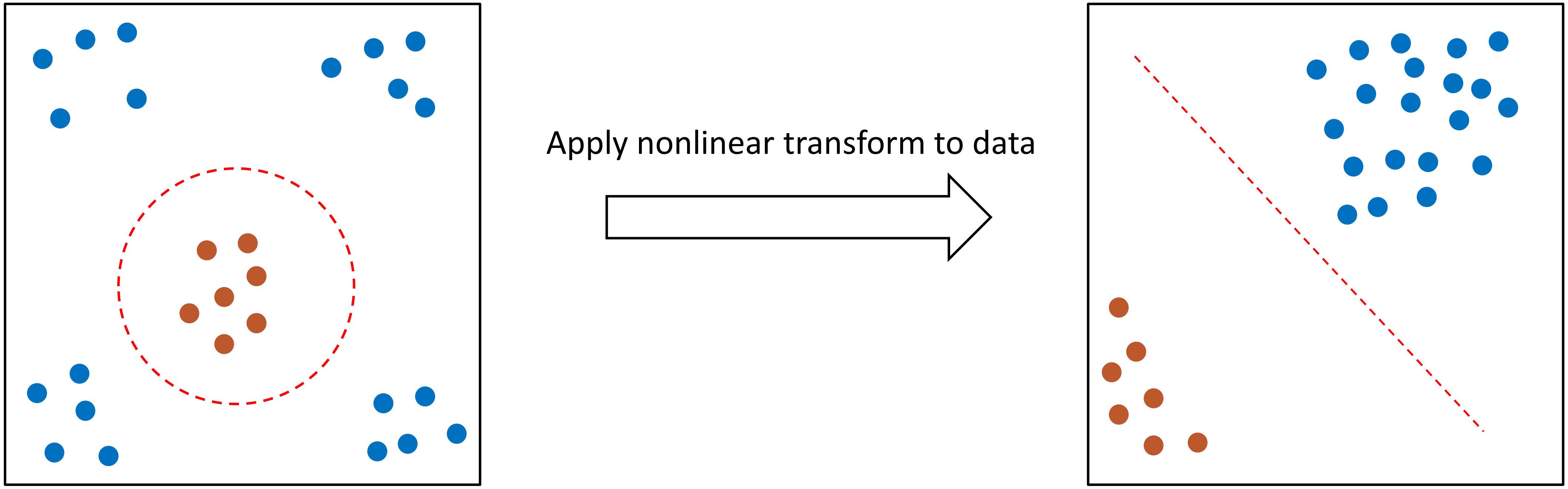
Kernel Methods

Data may not lend itself to a linear boundary for classification



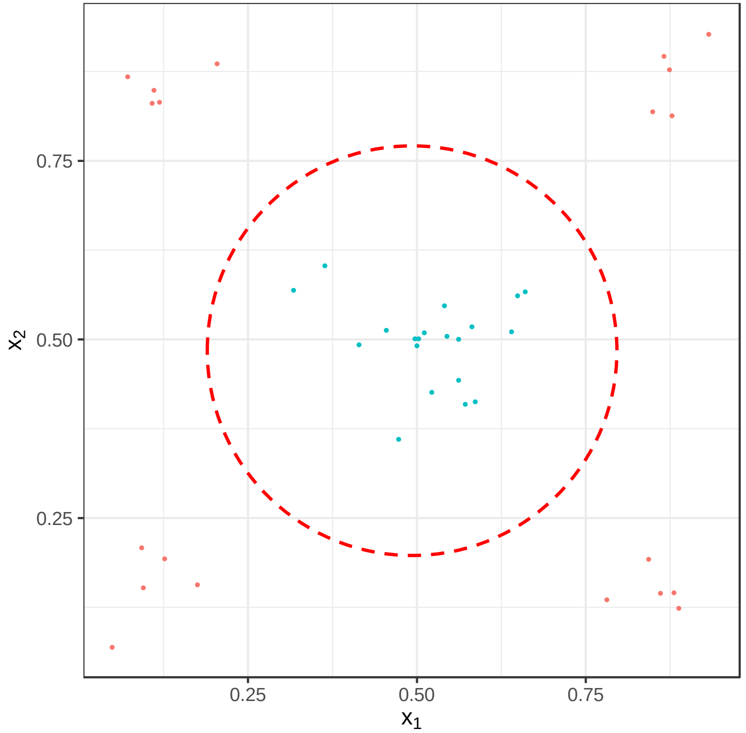
- Image Classification
- Text Classification:
- Bioinformatics and Genomics
- Speech Recognition
- Anomaly Detection
- Chemoinformatics
- Social Network Analysis
- Financial Data Analysis

Nonlinear Transformation to New Space



Example of a Non-Linear Transform

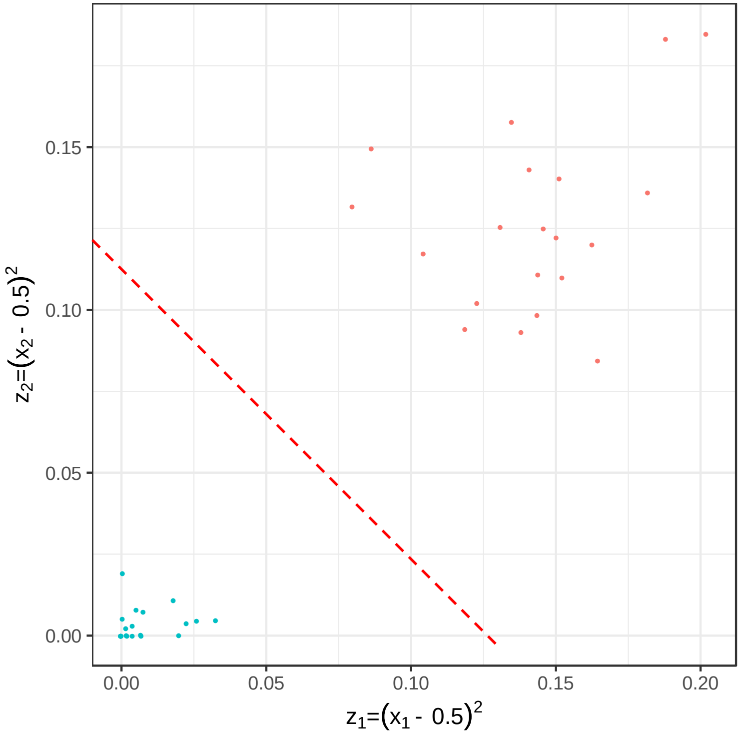
X space: x_1, x_2



Non-linear **transform**

$$z_1 = (x_1 - 0.5)^2$$
$$z_2 = (x_2 - 0.5)^2$$

Z space: z_1, z_2



Modification to the Basic SVM Lagranjian

- $L(\boldsymbol{\lambda}) = -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\lambda^{(i)}\lambda^{(j)}y^{(i)}y^{(j)}\mathbf{z}^{(i)}\,.\,\mathbf{z}^{(j)} + \sum_{i=1}^{m}\lambda^{(i)}$ subject to the constraints:
  - $\boldsymbol{\lambda}^{(i)} \geq \mathbf{0}$
  - $\boldsymbol{\lambda}.\boldsymbol{y} = 0$
- We can solve this in the $\boldsymbol{z}$-space as before
- Dimension of $\boldsymbol{z}$ does not matter
- We just need the inner product term
- Quadratic programming gives us $\lambda^{(i)}$
- Most of these will be 0 except for support vectors where $\lambda^{(k)} > 0$
  - So, $\boldsymbol{w} = \begin{bmatrix} w1 \\ \dots \\ w_n \end{bmatrix} = \sum_{z^{(i)}\,\varepsilon\, SV}\lambda^{(i)}y^{(i)}\boldsymbol{z}^{(i)}$
  - b can be obtained using one of the support vectors $x^{(k)}$ from:
    - $\left|b + \sum_j w_j {z_j}^{(k)}\right| = 1$
  - Hence, we have the parameters for the decision boundary in the $\boldsymbol{z}$-space: $\boldsymbol{w}^T\boldsymbol{z} + b = 0$

What do we need from Z space?

$$\mathcal{L}(\boldsymbol{\lambda}) = \sum_{i=1}^{m} \lambda^{(i)} - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y^{(i)} y^{(j)} \lambda^{(i)} \lambda^{(j)} \boldsymbol{z}^{(i)T} \boldsymbol{z}^{(j)}$$

Constraints:

$$\left.\begin{array}{l} 1.\ \lambda^{(i)} \geq 0 \text{ for } i = 1, \cdots, m \\ 2.\ \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} = 0 \end{array}\right\} \text{No requirement of } z$$

Solve this in $\boldsymbol{Z}$ **space** using quadratic programming

$$\boldsymbol{\lambda} = (\lambda^{(1)}, \lambda^{(2)}, \cdots, \lambda^{(m)})$$

$$w = \sum_{z^{(i)} \in SV} \lambda^{(i)} y^{(i)} z^{(i)}$$

b can be solved by taking any SV $z^{(k)}$: $y^{(k)}\left(w^T z^{(k)} + b\right) = 1$

The decision boundary is: $\boldsymbol{w}^T \boldsymbol{z} + b = 0$

Only need to know $\boldsymbol{z}^{(i)T}\boldsymbol{z}^{(k)}$, i.e., inner product

Do not need to know exactly what the transform $\boldsymbol{x} \rightarrow \boldsymbol{z}$ is

Existence of $\boldsymbol{z}$ space is sufficient:

- We do not to know *what* it is.
- We just need the inner product

Inner product in $\boldsymbol{z}$ space

Given any two points in X space: $\boldsymbol{x}$ and $\boldsymbol{x}'$

We need $\boldsymbol{z}^{\boldsymbol{T}}\boldsymbol{z}'$

Let $\boldsymbol{z}^{\boldsymbol{T}}\boldsymbol{z}' = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}') \longrightarrow$ **Kernel**

a **kernel** is a function that computes the dot product between the transformed data points in a higher-dimensional space

Example:

2nd order polynomial transformation Φ

$$\boldsymbol{x} = (x_1, x_2) \longrightarrow \boldsymbol{z} = \Phi(\boldsymbol{x}) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

$$\boldsymbol{x}' = (x'_1, x'_2) \qquad \boldsymbol{z}' = \Phi(\boldsymbol{x}') = (1, x'_1, x'_2, x'^2_1, x'^2_2, x'_1 x'_2)$$

$$K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{z}^{\boldsymbol{T}}\boldsymbol{z}' = 1 + x_1 x'_1 + x_2 x'_2 + x_1^2 x'^2_1 + x_2^2 x'^2_2 + x_1 x_2 x'_1 x'_2$$

Can we compute $K(\boldsymbol{x}, \boldsymbol{x}')$ *without* transforming $\boldsymbol{x}$ and $\boldsymbol{x}'$, i.e., without explicit knowledge of Φ?

Inner product in Z space

Consider the form $K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{x}')^2$ $\left\{ \begin{array}{l} \text{It's just a function, not inner product in X space} \\ \text{We don't know if it is the inner product in any other space} \end{array} \right.$

$$= (1 + x_1 x'_1 + x_2 x'_2)^2$$

$$= 1 + x_1^2 x'^2_1 + x_2^2 x'^2_2 + 2x_1 x'_1 + 2x_2 x'_2 + 2x_1 x_2 x'_1 x'_2$$

This is still an inner product

If we use such a $\mathbf{x} \rightarrow \boldsymbol{z}$ transformation:

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

$$\boldsymbol{z}' = (1, x'^2_1, x'^2_2, \sqrt{2}x'_1, \sqrt{2}x'_2, \sqrt{2}x'_1 x'_2)$$

Kernel becomes handy when raising to higher order space

Example: Polynomial kernel

$\boldsymbol{x} = \mathbb{R}^d$ $\Phi: \boldsymbol{x} \to \boldsymbol{z}$ is a polynomial transformation of order Q

The equivalent kernel is: $\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{1} + \boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{x}')^{\boldsymbol{Q}}$

$$= (1 + x_1 x'_1 + x_2 x'_2 + \cdots + x_d x'_d)^Q$$

$$\Downarrow$$

This is a rather simple computation:

sum of d products, and raise to the power of Q

$$\Updownarrow$$

Don’t need to actually expand the polynomial

Scales can be adjusted: $\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{a}\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{x}' + \boldsymbol{b})^{\boldsymbol{Q}}$

Example: RBF kernel

We only need $\boldsymbol{z}$ to exist. Don't need to know what $\boldsymbol{z}$ is.

Radius basis function (**RBF**) kernel:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{e}^{-\eta\|\boldsymbol{x}-\boldsymbol{x}'\|^2}$$

Is this an inner product in *some* space? YES!

An easier example: $x \in \mathbb{R},\ \eta = 1$

$$K(x, x') = \mathrm{e}^{-(x-x')^2}$$

$$= \mathrm{e}^{-\mathrm{x}^2}\,\mathrm{e}^{\mathrm{x}'^2}\,\boxed{\mathrm{e}^{2\mathrm{xx}'}}$$

Taylor expansion:

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + \cdots$$

$$= \exp(-x^2)\exp(-x'^2)\sum_{k=0}^{\infty}\frac{2^k x^k x'^k}{k!}$$

This is an inner product within a $\boldsymbol{z}$ space of **infinite dimensions!**

$$= \exp(-x^2)\exp\left(-x'^2\right)\left(1 + 2xx' + \frac{2^2}{2!}x^2x'^2 + \frac{2^3}{3!}x^3x'^3 + \cdots\right)$$

Quadratic programming with kernel function

$$\max_{\boldsymbol{\lambda}}\left(-\frac{1}{2}\boldsymbol{\lambda}^T\boldsymbol{Q}\boldsymbol{\lambda}+(\mathbf{1}^T)\boldsymbol{\lambda}\right)$$ Subject to: $\boldsymbol{y}^T\boldsymbol{\lambda}=0; \boldsymbol{\lambda}>0$

$$Q=\begin{bmatrix} y^{(1)}y^{(1)}\boldsymbol{K}(\boldsymbol{x}^{(1)}.\boldsymbol{x}^{(1)})\ \dots & y^{(1)}y^{(m)}K(\boldsymbol{x}^{(1)}.\boldsymbol{x}^{(m)}) \\ \dots. & \dots. \\ y^{(m)}y^{(1)}\boldsymbol{K}(\boldsymbol{x}^{(m)}.\boldsymbol{x}^{(1)})\ \dots & y^{(m)}y^{(m)}\boldsymbol{K}(\boldsymbol{x}^{(m)}.\boldsymbol{x}^{(m)}) \end{bmatrix}$$

Replace $\boldsymbol{x}^{(\boldsymbol{i})\boldsymbol{T}}\boldsymbol{x}^{(\boldsymbol{j})}$ with $K(\boldsymbol{x}^{(\boldsymbol{i})},\boldsymbol{x}^{(\boldsymbol{j})})$

Everything else is the same!

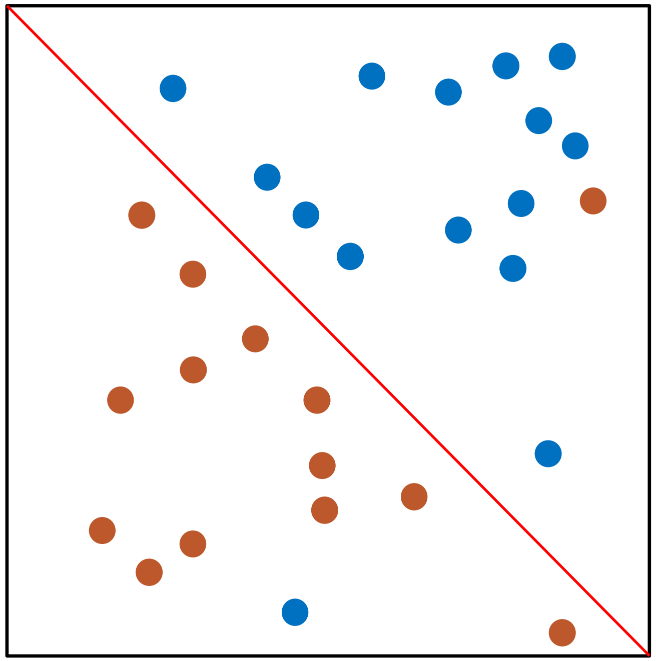
Soft Margin

Introduction

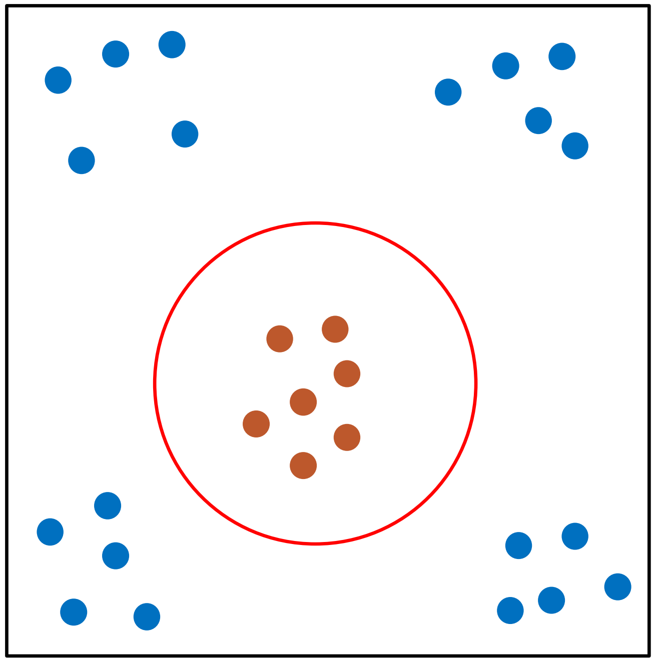
- The soft margin in Support Vector Machines (SVM) is beneficial in scenarios where the data may not be perfectly separable, or when there is a need to tolerate some level of misclassification. Here are some example use cases where the soft margin in SVM could be advantageous:
  - In medical diagnosis, patient data may not have clear boundaries between different classes.
  - Image classification tasks may involve noisy data, and perfect separation between classes may not be possible.
  - In text classification, features may not provide a clear separation between different categories.
  - Financial data which is often noisy and subject to market fluctuations.
  - Biological data, such as gene expression profiles, may exhibit overlapping patterns.
  - Fraud detection may involve imbalanced datasets and ambiguous patterns
  - Handwriting recognition tasks may encounter variations in writing styles.
  - Speech signals often include background noise, making perfect separation difficult.
  - Customer behavior data may not always lead to clear distinctions between market segments.
  - Network security data often contains anomalies that may not be clearly separated from normal behavior.
- In these use cases, the soft margin in SVM provides a balance between achieving a wider margin and allowing for misclassifications, making the model more adaptable to real-world scenarios with inherent uncertainty and complexity.

Use of Soft Margin vs Kernel Method

Linear Separation May be Tolerable

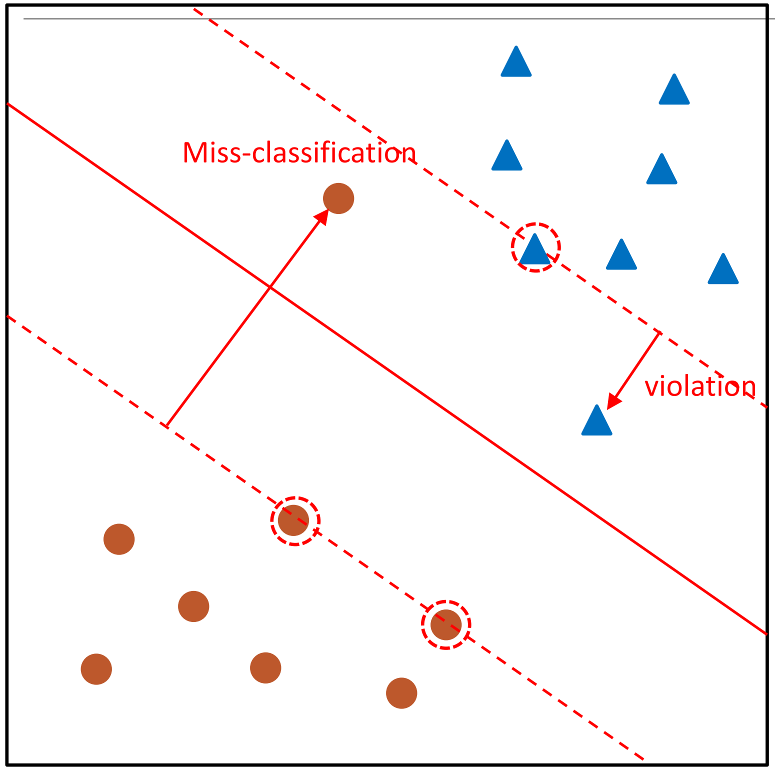


Linear Separation Won’t Work



Soft-margin deals with this → Combine ← Kernels deal with this

Violation of margin



For SVs: $y^{(k)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(k)}+b\right)=1$

For any points outside the margin: $y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)}+b\right)>0$

Allow violations:

$$\begin{cases} 0 < y^{(k)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(k)}+b\right) < 1 & \text{Violate margin} \\ y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)}+b\right) < 0 & \text{Miss-classification} \end{cases}$$

Use a **slack** to measure the violation:

$$y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)}+b\right) \geq 1-\xi^{(i)} \qquad \xi^{(i)} \geq 0$$

Penalize the *total violation*: $=\sum_{i=1}^{m}\xi^{(i)}$

Change the optimization goal by including this error term

New optimization goal

Minimize: $\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{m}\xi^{(i)}$

Subject to $y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b\right) \geq 1 - \xi^{(i)}$, for $i = 1, \cdots, m$

and $\xi^{(i)} \geq 0$, for $i = 1, \cdots, m$

Hyper-parameter C quantifies the relative importance of avoiding violations

or tolerance to violations
Like L2 regularization

If C is big:

w will be learned in a way that only small $\xi^{(i)}$ are allowed

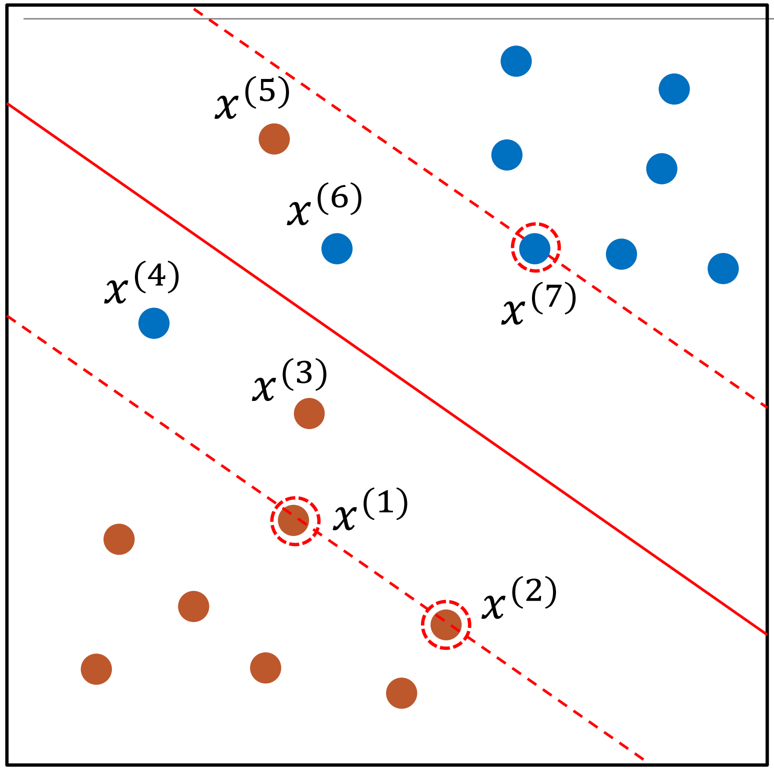
If C is small:

w will be learned in a way that big $\xi^{(i)}$ are allowed

Updated Lagranjian

- We define the Lagranjian as:
  - $L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\gamma}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \boldsymbol{C}\sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \xi^{(\boldsymbol{i})}$ - $\sum_{1=1}^{m} \lambda^{(i)}(y^{(i)}(\boldsymbol{w}^T\, \boldsymbol{x}^{(i)} + b) - 1 + \xi^{(\boldsymbol{i})}) - \sum_{i=1}^{m} \gamma^{(i)}\xi^{(\boldsymbol{i})}$
- We minimize *L* wrt ***w*** and *b* and $\boldsymbol{\xi}$ and maximize *L* wrt $\lambda^{(i)} \geq 0$ and $\gamma^{(i)} \geq 0$
- Minimize L wrt ***w*** and *b*
  - $\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{1=1}^{m} \lambda^{(i)} y^{(i)} \boldsymbol{x}^{(i)} \quad = 0$
  - $\frac{\partial L}{\partial b} = -\sum_{1=1}^{m} \lambda^{(i)} y^{(i)} = 0$
  - $=> \boldsymbol{\lambda} . \boldsymbol{y} = 0$
  - $\frac{\partial L}{\partial \boldsymbol{\xi}^{(i)}} = C - \lambda^{(i)} - \gamma^{(i)} = 0$
- In $L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\gamma})$, substituting $\boldsymbol{w} = \sum_{i=1}^{m} \lambda^{(i)} y^{(i)} \boldsymbol{x}^{(i)}$ in L and utilizing $\sum_{i=1}^{m} \lambda^{(i)} y^{(i)} = 0$ and utilizing $C - \lambda^{(i)} - \gamma^{(i)} = 0$
- $L(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \lambda^{(i)}\lambda^{(j)} y^{(i)} y^{(j)} \boldsymbol{x}^{(i)} . \boldsymbol{x}^{(j)} - \sum_{i=1}^{m}\sum_{j=1}^{m} \lambda^{(i)}\lambda^{(j)} y^{(i)} y^{(j)} \boldsymbol{x}^{(i)} . \boldsymbol{x}^{(j)} + \sum_{i=1}^{m} \lambda^{(i)}$
- Same Lagranjian as before with one additional constraint, $C \geq \lambda^{(i)} \geq 0$ because $\gamma^{(i)} \geq 0$
- We can utilize Quadratic programming with the additional constraint to solve for $\lambda^{(i)}$ and then for ***w*** and *b*

Types of support vectors



Strictly greater than

Margin support vectors: $C > \lambda^{(i)} > 0$

$$y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b\right) = 1 \qquad \xi^{(i)} = 0 \qquad \gamma^{(i)} > 0$$

Non-margin support vectors: $\lambda^{(i)} = C$

$$y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b\right) < 1 \qquad \xi^{(i)} > 0 \qquad \gamma^{(i)} = 0$$

Violate the margin, but correctly classified

$$0 < y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b\right) < 1$$

Violate the margin, and misclassified

$$y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b\right) < 0$$

Choose proper hyperparameters

Soft margin is usually the default

Choose $C \in \{2^{-4}, 2^{-3}, \cdots, 2^{4}\}$ Best C can be determined by cross-validation

Combined with kernel methods

E.g., RBF kernel $K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\eta \|\boldsymbol{x} - \boldsymbol{x}'\|^2)$

Choose best C and γ using grid search: $C, \eta \in \{2^{-4}, 2^{-3}, \cdots, 2^{4}\}$