Machine Learning and Intelligent Systems

Kernels (Part 1)

Maria A. Zuluaga

Nov 17, 2023

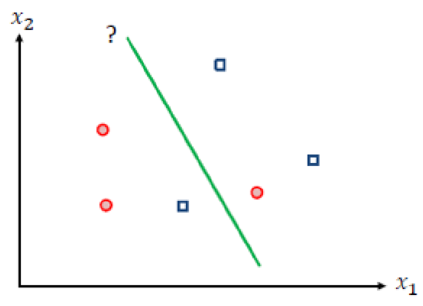
EURECOM - Data Science Department

Table of contents

Recap: Limitations of Hard SVM

Hard Margin SVMs: Limitations

- Real data, most likely, will not meet the of linear separable assumption
- Hard margin loss is too limiting when there is class overlapping
- Hard margin SVM wont be able to deal with it

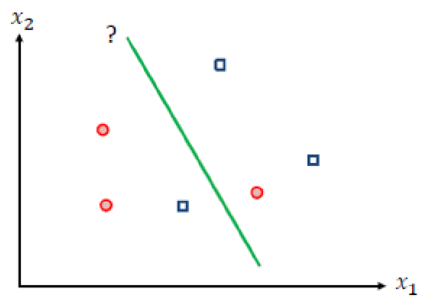


Hard Margin SVMs: Limitations

- Real data, most likely, will not meet the of linear separable assumption
- Hard margin loss is too limiting when there is class overlapping
- Hard margin SVM wont be able to deal with it

Possible solutions:

1. **Transform the data**
2. Relax the constraints
3. Combination of both

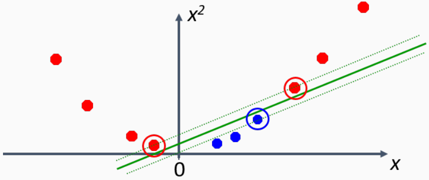


Non-linear Decision Boundary

Idea: move data into a higher dimension space and search for a linear separator

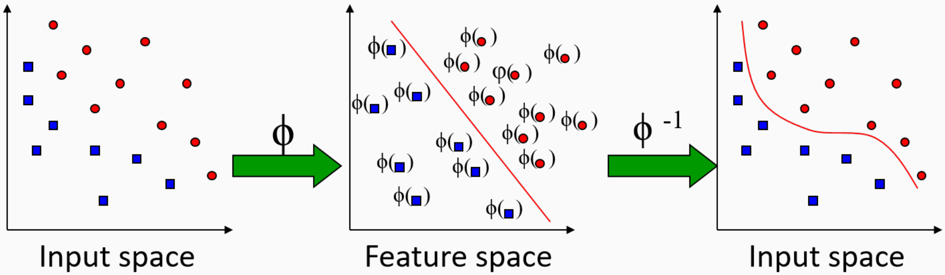
1. 1D problem: Not linearly separable
2. 1D to 2D transformation
3. 2D problem: Linearly separable

$$X \rightarrow (X, X^2)$$



Non-linear Decision Boundary

- **Idea:** Define a transform ϕ from input space to feature space, $\mathbf{x} \to \phi(\mathbf{x})$. It will:
- Solve a linear problem in the feature space
- Result in a non-linear classifier in the input space
- $\phi(\mathbf{x}) \in \mathbb{R}^d$, with typically $d \gg D$



The naïve approach: Hand-crafted features

1. Choose/design a linear model
2. Choose/design a high-dimensional transformation $\phi(\mathbf{x})$
3. After adding many features some of them will make the data linearly separable (fingers crossed)
4. **For each** training example, and **for each** new sample point compute $\phi(\mathbf{x})$
5. Train classifier. Later on, do predictions

The naïve approach: Hand-crafted features

1. Choose/design a linear model
2. Choose/design a high-dimensional transformation $\phi(\mathbf{x})$
3. After adding many features some of them will make the data linearly separable (fingers crossed)
4. **For each** training example, and **for each** new sample point compute $\phi(\mathbf{x})$
5. Train classifier. Later on, do predictions

Problem: Impractical to compute $\phi(\mathbf{x})$ for high-dimensional $\phi(\mathbf{x})$-spaces

Example

Consider $\mathbf{x} = [x^1, \ldots, x^D]^T$ and let us define

$$\phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x^1 \\ \vdots \\ x^D \\ x^1 x^2 \\ \vdots \\ x^1 x^D \\ \vdots \\ x^1 x^2 \ldots x^D \end{pmatrix}$$

What is the dimension of the transformed data?

Kernels

The Kernel Trick

- It is possible to avoid the direct estimation of $\phi(\mathbf{x})$ for every point
- It is even possible to avoid the estimation of $\hat{\mathbf{w}}$

The Kernel Trick

- It is possible to avoid the direct estimation of $\phi(\mathbf{x})$ for every point
- It is even possible to avoid the estimation of $\hat{\mathbf{w}}$
- It is possible to express a linear classifier in term of inner products. This is known as **the kernel trick**

The Kernel Trick

- It is possible to avoid the direct estimation of $\phi(\mathbf{x})$ for every point
- It is even possible to avoid the estimation of $\hat{\mathbf{w}}$
- It is possible to express a linear classifier in term of inner products. This is known as **the kernel trick**
- **Principle:** If we use gradient descent with one of the loss functions we know so far, it is possible to express the gradient as a linear combination of the input samples, $\mathbf{x}$

Sketching the Kernel Trick

- **Claim:** The model parameters can be expressed as a linear combination of all input vectors:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i \tag{1}$$

for a given assignment of α

Sketching the Kernel Trick

- **Claim:** The model parameters can be expressed as a linear combination of all input vectors:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i \tag{1}$$

  for a given assignment of α
- Let us define the inner product between two vectors as $\mathbf{K}_{ij} = \mathbf{x}_i^T \mathbf{x}_j$

Sketching the Kernel Trick

- **Claim:** The model parameters can be expressed as a linear combination of all input vectors:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i \tag{1}$$

  for a given assignment of α
- Let us define the inner product between two vectors as $\mathbf{K}_{ij} = \mathbf{x}_i^T \mathbf{x}_j$
- Let us assume our classifier is trained with the squared loss. We have:

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)^2, \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_{i=1}^{N} 2(\mathbf{w}^T \mathbf{x}_i - y_i)\mathbf{x}_i \tag{2}$$

Sketching the Kernel Trick

- **Claim:** The model parameters can be expressed as a linear combination of all input vectors:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i \tag{1}$$

for a given assignment of α

- Let us define the inner product between two vectors as $\mathbf{K}_{ij} = \mathbf{x}_i^T \mathbf{x}_j$
- Let us assume our classifier is trained with the squared loss. We have:

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)^2, \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_{i=1}^{N} 2(\mathbf{w}^T \mathbf{x}_i - y_i)\mathbf{x}_i \tag{2}$$

- Using Eq. 1, the term $\mathbf{w}^T \mathbf{x}_i$ can be estimated as:

$$\mathbf{w}^T \mathbf{x}_j = \sum_{i=1}^{N} \alpha_i \underbrace{\mathbf{x}_i^T \mathbf{x}_j}_{\mathbf{K}_{ij}}$$

Proof: Setup

Let us recall the gradient descent expression:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \lambda \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \tag{3}$$

Note: We will use λ instead of α for the learning rate in the next slides

Proof: Setup

Let us recall the gradient descent expression:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \lambda \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \tag{3}$$

Note: We will use λ instead of α for the learning rate in the next slides

Given the squared loss we chose in Eq. 2, let us do some reorganization of its derivative

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_{i=1}^{N} \underbrace{2(\mathbf{w}^{T}\mathbf{x}_i - y_i)}_{\gamma_i\text{:function of } \mathbf{w}^T\mathbf{x}_i} \mathbf{x}_i$$

Proof: Setup

Let us recall the gradient descent expression:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \lambda \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \tag{3}$$

Note: We will use λ instead of α for the learning rate in the next slides

Given the squared loss we chose in Eq. 2, let us do some reorganization of its derivative

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \sum_{i=1}^{N} \underbrace{2(\mathbf{w}^T \mathbf{x}_i - y_i)}_{\gamma_i\text{:function of } \mathbf{w}^T\mathbf{x}_i} \mathbf{x}_i \\ &= \sum_{i=1}^{N} \gamma_i \mathbf{x}_i \end{aligned}$$

Proof: Setup

Let us recall the gradient descent expression:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \lambda \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \tag{3}$$

Note: We will use λ instead of α for the learning rate in the next slides

Given the squared loss we chose in Eq. 2, let us do some reorganization of its derivative

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \sum_{i=1}^{N} \underbrace{2(\mathbf{w}^T \mathbf{x}_i - y_i)}_{\gamma_i\text{:function of } \mathbf{w}^T \mathbf{x}_i} \mathbf{x}_i \\ &= \sum_{i=1}^{N} \gamma_i \mathbf{x}_i \end{aligned}$$

Let's move to the proof

Proof by Induction: Sketch

- We will use gradient descent to estimate the model parameters $\mathbf{w}$
- **Important:** Since the squared loss is convex, no matter how we initialize $\mathbf{w}$ the solution is always the same

Proof by Induction: Sketch

- We will use gradient descent to estimate the model parameters $\mathbf{w}$
- **Important:** Since the squared loss is convex, no matter how we initialize $\mathbf{w}$ the solution is always the same
- Let us so choose $\mathbf{w}^{(0)} = \vec{0}$ s
- What is the value of the α's? Recall that

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$$

Proof by Induction: Sketch

- We will use gradient descent to estimate the model parameters $\mathbf{w}$
- **Important:** Since the squared loss is convex, no matter how we initialize $\mathbf{w}$ the solution is always the same
- Let us so choose $\mathbf{w}^{(0)} = \vec{0}$ s
- What is the value of the α's? Recall that

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$$

- Answer: $\alpha_i = 0 \quad \forall\, i$ (base case)

Proof by Induction: Sketch

Let us now see what happens in a given round of gradient descent

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \lambda \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$$

Proof by Induction: Sketch

Let us now see what happens in a given round of gradient descent

$$\begin{aligned}\mathbf{w}^{(\tau)} &= \mathbf{w}^{(\tau-1)} - \lambda \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \\ &= \mathbf{w}^{(\tau-1)} - \lambda \sum_{i=1}^{N} \gamma_i^{(\tau-1)} \mathbf{x}_i\end{aligned}$$

Proof by Induction: Sketch

Let us now see what happens in a given round of gradient descent

$$\begin{aligned}\mathbf{w}^{(\tau)} &= \mathbf{w}^{(\tau-1)} - \lambda \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \\ &= \mathbf{w}^{(\tau-1)} - \lambda \sum_{i=1}^{N} \gamma_i^{(\tau-1)} \mathbf{x}_i\end{aligned}$$

But let us recall that $\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$. Therefore,

Proof by Induction: Sketch

Let us now see what happens in a given round of gradient descent

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \lambda \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$$
$$= \mathbf{w}^{(\tau-1)} - \lambda \sum_{i=1}^{N} \gamma_i^{(\tau-1)} \mathbf{x}_i$$

But let us recall that $\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$. Therefore,

$$\mathbf{w}^{(\tau)} = \sum_{i=1}^{N} \alpha_i^{(\tau-1)} \mathbf{x}_i - \lambda \sum_{i=1}^{N} \gamma_i^{(\tau-1)} \mathbf{x}_i$$

Proof by Induction: Sketch

Let us now see what happens in a given round of gradient descent

$$\begin{aligned}\mathbf{w}^{(\tau)} &= \mathbf{w}^{(\tau-1)} - \lambda\frac{\partial\mathcal{L}}{\partial\mathbf{w}} \\ &= \mathbf{w}^{(\tau-1)} - \lambda\sum_{i=1}^{N}\gamma_i^{(\tau-1)}\mathbf{x}_i\end{aligned}$$

But let us recall that $\mathbf{w} = \sum_{i=1}^{N}\alpha_i\mathbf{x}_i$. Therefore,

$$\begin{aligned}\mathbf{w}^{(\tau)} &= \sum_{i=1}^{N}\alpha_i^{(\tau-1)}\mathbf{x}_i - \lambda\sum_{i=1}^{N}\gamma_i^{(\tau-1)}\mathbf{x}_i \\ &= \sum_{i=1}^{N}\underbrace{\left(\alpha_i^{(\tau-1)} - \lambda\gamma_i^{(\tau-1)}\right)}_{\alpha^{(\tau)}}\mathbf{x}_i\end{aligned}$$

By induction we proof for $\mathbf{w}^{(\tau)}$ and it follows for $\mathbf{w}^{(\tau+1)}$

Proof by Induction: Sketch

Let us now see what happens in a given round of gradient descent

$$\begin{aligned}\mathbf{w}^{(\tau)} &= \mathbf{w}^{(\tau-1)} - \lambda \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \\ &= \mathbf{w}^{(\tau-1)} - \lambda \sum_{i=1}^{N} \gamma_i^{(\tau-1)} \mathbf{x}_i\end{aligned}$$

But let us recall that $\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$. Therefore,

$$\begin{aligned}\mathbf{w}^{(\tau)} &= \sum_{i=1}^{N} \alpha_i^{(\tau-1)} \mathbf{x}_i - \lambda \sum_{i=1}^{N} \gamma_i^{(\tau-1)} \mathbf{x}_i \\ &= \sum_{i=1}^{N} \underbrace{\left(\alpha_i^{(\tau-1)} - \lambda \gamma_i^{(\tau-1)}\right)}_{\alpha^{(\tau)}} \mathbf{x}_i\end{aligned}$$

By induction we proof for $\mathbf{w}^{(\tau)}$ and it follows for $\mathbf{w}^{(\tau+1)}$

It is possible to perform gradient descent without ever expressing $\mathbf{w}$ explicitly

Algorithm

The update rule for $\alpha^{(\tau)}$ is

$$\alpha^{(\tau+1)} = \alpha^{(\tau)} - \lambda\gamma_i^{(\tau)} \tag{4}$$

Training Algorithm:

1. Initialize $\alpha_i = 0 \quad \forall\, i, i = 1, \ldots, N$
2. Estimate γ_i
3. Update α using the update rule (eq. 4)
4. Go to step 2

Prediction: For an unseen point $\mathbf{x}^*$,

$$h(\mathbf{x}^*) = \hat{\mathbf{w}}^T\mathbf{x}^* = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i^T \mathbf{x}^*$$

Kernels

- The previous reasoning applies if we use the transformation $\phi(\mathbf{x})$
- In that case,

$$\mathbf{K}_{ij} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Kernels

- The previous reasoning applies if we use the transformation $\phi(\mathbf{x})$
- In that case,

$$\mathbf{K}_{ij} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

- With the kernel trick we avoid the direct estimation of $\mathbf{w}$. However, we still need to estimate $\phi(\mathbf{x})$
- If d is of very high dimension the estimation of the inner products $\mathbf{K}_{ij}$ can be a very expensive operation. **We have gained nothing!**

Kernels

- **Idea:** Compute the value of **K** without explicitly writing the expensive representation

- Example: Consider $\mathbf{u} = [u_1]$, $\mathbf{v} = [v_1]$, and a transformation $\phi(\mathbf{x}) = [x_1^2, \sqrt{2c}x_1, c]^T$, with c a constant. Estimate:
  - $\phi(\mathbf{u}) =$
  - $\phi(\mathbf{v}) =$
  - $\phi(\mathbf{u})^T\phi(\mathbf{v}) =$
- Alternatively, this can be computed as $(u_1 v_1 + c)^2$

Kernels

- **Idea:** Compute the value of $\mathbf{K}$ without explicitly writing the expensive representation

- Example: Consider $\mathbf{u} = [u_1]$, $\mathbf{v} = [v_1]$, and a transformation $\phi(\mathbf{x}) = [x_1^2, \sqrt{2c}x_1, c]^T$, with c a constant. Estimate:
  - $\phi(\mathbf{u}) =$
  - $\phi(\mathbf{v}) =$
  - $\phi(\mathbf{u})^T\phi(\mathbf{v}) =$
- Alternatively, this can be computed as $(u_1 v_1 + c)^2$

- We denote $k(\mathbf{u}, \mathbf{v}) = (u_1 v_1 + c)^2 = \phi(\mathbf{u})^T\phi(\mathbf{v})$ a **kernel function**

- With a finite training set of N samples, the inner products can be cheaply pre-computed and stored in a **kernel matrix**:

$$\mathbf{K}_{ij} = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$$

Wrap-up

Wrap-up

- We proposed to transform the input space to address the problem of non-linear separability
- We showed that thanks to the kernel trick it is possible to avoid the direct estimation of $\mathbf{w}$ by using inner products
- We showed that certain type of functions, the kernel functions, avoid the need to estimate inner products
- We introduced the kernel matrix
- **What comes next:**
  - We will present different kernel functions
  - We will present the kernel version of some covered methods

Key Concepts

- Feature transformation
- Kernel trick
- Kernel function
- Kernel matrix
- Proof by induction