Support Vector Machines

Statistical Machine Learning (ENGG*6600*08)

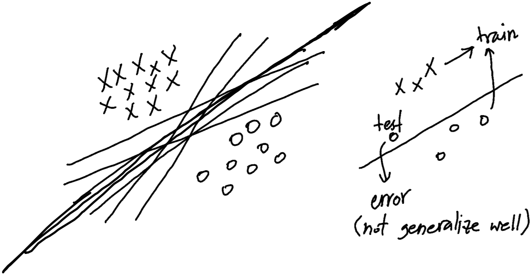
School of Engineering,
University of Guelph, ON, Canada

Course Instructor: Benyamin Ghojogh
Fall 2023

SVM vs. Perceptron

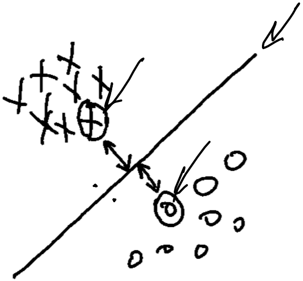
SVM vs. Perceptron

- Perceptron proposed by Rosenblatt (physiologist) in 1958, at Cornell Aeronautical Laboratory [1], was a neuron of neural network for binary classification.
- In later attempts, Hebbian learning, prpoposed in 1949 [2], was used for learning in Perceptron.
- Perceptron cannot generalize well enough because, for linearly separable classes, it finds one of the many possible decision boundaries.



SVM vs. Perceptron

- Linear SVM was proposed by Vladimir Vapnik *et al.* in 1974 [3].
- Liner SVM tries to find the best decision boundary rather than one of the possible decision boundaries. The best decision boundary is the one which has the **largest distance from the closest points of classes to the decision boundary**. This is because those points are also the **closest points of the two classes to each other** and therefore they are **most capable of being confused** between the two classes. These points are called the **support vectors**.



Hard Margin SVM

Hard Margin SVM

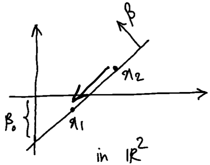
- We want to find a linear decision boundary where one class falls in one side and the other class falls on the other side.
- The equation of a line for linear decision boundary:

$$\boxed{\boldsymbol{\beta}^\top \boldsymbol{x} + \beta_0 = 0,} \tag{1}$$

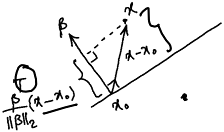
  where $\boldsymbol{x} \in \mathbb{R}^d$ is the data point, $\boldsymbol{\beta} \in \mathbb{R}^d$ is the normal vector of the linear line, and β_0 is the bias (intercept) of the line.
- Consider any two points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ on the decision boundary. As the line passes through each of them, they both satisfy Eq. (1):

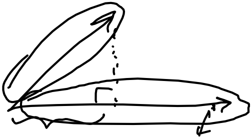
$$\boxed{\boldsymbol{\beta}^\top \boldsymbol{x}_1 + \beta_0 = 0,} \boxed{\boldsymbol{\beta}^\top \boldsymbol{x}_2 + \beta_0 = 0} \implies \boldsymbol{\beta}^\top \boldsymbol{x}_1 + \cancel{\beta_0} = \boldsymbol{\beta}^\top \boldsymbol{x}_2 + \cancel{\beta_0},$$
$$\implies \underbrace{\boldsymbol{\beta}^\top (\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0} \implies \underbrace{\boldsymbol{\beta} \perp (\boldsymbol{x}_1 - \boldsymbol{x}_2),}$$

  which verifies that $\boldsymbol{\beta}$ is the normal vector of the decision boundary.



Hard Margin SVM





- Consider a point $\underline{\boldsymbol{x}_0}$ on the $\underline{\text{decision boundary}}$ and $\underline{\text{a point } \boldsymbol{x}}$ on $\underline{\text{one of the sides of the decision boundary}}$. Therefore:

$$\boxed{\boldsymbol{\beta}^\top \boldsymbol{x}_0 + \beta_0 = 0} \implies \boxed{\beta_0 = -\boldsymbol{\beta}^\top \boldsymbol{x}_0.} \bigstar \tag{2}$$

- The distance of point $\boldsymbol{x}$ from the decision boundary is:

$$\boxed{d} = \underbrace{\frac{\boldsymbol{\beta}^\top}{\|\boldsymbol{\beta}\|_2}(\boldsymbol{x} - \boldsymbol{x}_0)} = \underbrace{\frac{\boldsymbol{\beta}^\top(\boldsymbol{x} - \boldsymbol{x}_0)}{\|\boldsymbol{\beta}\|_2}} = \frac{\boldsymbol{\beta}^\top \boldsymbol{x} \boxed{- \boldsymbol{\beta}^\top \boldsymbol{x}_0}}{\|\boldsymbol{\beta}\|_2} \overset{(2)}{=} \underbrace{\frac{\boldsymbol{\beta}^\top \boldsymbol{x} + \beta_0}{\|\boldsymbol{\beta}\|_2}}. \tag{3}$$

- The distance should be non-negative so we have:

$$d = \left| \frac{\boldsymbol{\beta}^\top \boldsymbol{x} + \beta_0}{\boxed{\|\boldsymbol{\beta}\|_2}} \right| = \boxed{\frac{|\boldsymbol{\beta}^\top \boldsymbol{x} + \beta_0|}{\|\boldsymbol{\beta}\|_2}.}$$

However, the absolute value is non-smooth and non-differentiable. Therefore, we can multiply the distance with the target label to make it always non-negative:

$$\underline{y = +1} \implies \boxed{\boldsymbol{\beta}^\top \boldsymbol{x} + \beta_0} \underline{> 0,} \qquad \underline{y = -1} \implies \underline{\boldsymbol{\beta}^\top \boldsymbol{x} + \beta_0 < 0,} \tag{4}$$

$$\implies \underbrace{d := \frac{y(\boldsymbol{\beta}^\top \boldsymbol{x} + \beta_0)}{\|\boldsymbol{\beta}\|_2}.} \tag{5}$$

Hard Margin SVM

- This distance from decision boundary is either zero or greater than zero:

$$d = \frac{y(\boldsymbol{\beta}^\top \boldsymbol{x} + \beta_0)}{\|\boldsymbol{\beta}\|_2} = \begin{cases} 0 & \text{if } \boldsymbol{x} \text{ on decision boundary} \\ > 0 & \text{if } \boldsymbol{x} \text{ not on decision boundary.} \end{cases} \tag{6}$$

- Suppose there are $\{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$ data points and their class labels $\{y_i\}_{i=1}^n$, where $y_i \in \{-1, 1\}, \forall i \in \{1, \dots, n\}$.
- The closest points to the decision boundary are more at risk of being confused and misclassified. In each class of data, these closest points to the decision boundary are called **support vectors**.
- We want to maximize the margin (gap) between the support vectors and the decision boundary, so the decision boundary becomes as far as possible from data for having least amount of misclassification:

$$\underset{\boxed{\boldsymbol{\beta}, \beta_0}}{\text{maximize}} \quad \underline{d_i} = \underbrace{\frac{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0)}{\|\boldsymbol{\beta}\|_2}}, \quad \forall i \in \{1, \dots, n\}. \tag{7}$$

- This can be converted to a minimization problem:

$$\underset{\boldsymbol{\beta}, \beta_0}{\text{minimize}} \quad \underbrace{\frac{\|\boldsymbol{\beta}\|_2}{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0)}}, \quad \forall i \in \{1, \dots, n\}. \tag{8}$$

Hard Margin SVM

- We had this:

$$\star\star \quad \underset{\boldsymbol{\beta},\beta_0}{\text{minimize}} \quad \frac{\|\boldsymbol{\beta}\|_2}{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0)}, \quad \forall i \in \{1, \dots, n\}.$$

- Assume we desire to find a decision boundary which has some distance from data of classes so the distance of points are positive for support vectors. Let the constant s denote the smallest distance to the decision boundary, which is the distance of support vectors from the decision boundary. Then, all distances are greater then or equal to s:

$$d_i = \frac{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0)}{\|\boldsymbol{\beta}\|_2} \geq s, \quad \forall i \in \{1, \dots, n\}.$$

As this expression is greater than or equal to the constant s, we can assume that its numerator is greater than or equal to some constant c:

$$\star \boxed{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) \geq c,} \quad \forall i \in \{1, \dots, n\}. \tag{9}$$

- This equation appears in the denominator of Eq. (8). We can convert the optimization problem (8) to minimization of its numerator while its denominator satisfies Eq. (9):

$$\begin{aligned} &\underset{\boldsymbol{\beta},\beta_0}{\text{minimize}} && \|\boldsymbol{\beta}\|_2 \\ &\text{subject to} && \underbrace{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) \geq c,} \quad \forall i \in \{1, \dots, n\}. \end{aligned} \tag{10}$$

Hard Margin SVM

- It is simpler to minimize $(1/2)\|\boldsymbol{\beta}\|_2^2$ rather than $\|\boldsymbol{\beta}\|_2$ because $\|\boldsymbol{\beta}\|_2^2$ is quadratic. Therefore, we convert Eq. (10) to:

$$\bigstar \left\{ \begin{array}{ll} \underset{\boldsymbol{\beta},\beta_0}{\text{minimize}} & \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 \\ \text{subject to} & y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) \geq c, \quad \forall i \in \{1,\dots,n\}. \end{array} \right. \tag{11}$$

- The constant c can be any constant which is not important because of not having effect in derivative. Usually, literature uses $c = 1$.
- The constraint can be stated as:

$$y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) \geq c \implies -y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c \leq 0.$$

- The Lagrangian function for this problem is:

$$\bigstar \quad \mathcal{L}(\boldsymbol{\beta}, \beta_0, \{\alpha_i\}_{i=1}^n) = \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + \sum_{i=1}^n \alpha_i\big(-y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c\big),$$

where $\{\alpha_i\}_{i=1}^n$ are the Lagrange multipliers.

Hard Margin SVM

- We had:

$$\star\quad \mathcal{L}(\boldsymbol{\beta}, \beta_0, \{\alpha_i\}_{i=1}^{n}) = \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + \sum_{i=1}^{n} \alpha_i\big(-y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c\big).$$

- Derivative of Lagrangian w.r.t. $\boldsymbol{\beta}$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i \overset{\text{set}}{=} \boldsymbol{0} \Longrightarrow \boxed{\boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i.}\ \star \tag{12}$$

- Derivative of Lagrangian w.r.t. β_0:

$$\frac{\partial \mathcal{L}}{\partial \beta_0} = -\sum_{i=1}^{n} \alpha_i y_i \overset{\text{set}}{=} 0 \Longrightarrow \boxed{\sum_{i=1}^{n} \alpha_i y_i = 0.}\ \star \tag{13}$$

Hard Margin SVM

- We substitute Eqs. (12) and (13) in the Lagrangian:

$$\mathcal{L}(\boldsymbol{\beta}, \beta_0, \{\alpha_i\}_{i=1}^n) = \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + \sum_{i=1}^n \alpha_i \big(-y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c\big).$$

- The first term:

$$\frac{1}{2}\|\boldsymbol{\beta}\|_2^2 = \frac{1}{2}\boldsymbol{\beta}^\top\boldsymbol{\beta} \overset{(12)}{=} \frac{1}{2}\sum_{j=1}^n \alpha_j y_j \boldsymbol{x}_j^\top \sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i = \frac{1}{2}\sum_{j=1}^n\sum_{i=1}^n \alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j.$$

- The second term:

$$\sum_{i=1}^n \alpha_i\big(-y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c\big) = -\sum_{i=1}^n \alpha_i y_i \boldsymbol{\beta}^\top \boldsymbol{x}_i - \sum_{i=1}^n \alpha_i y_i \beta_0 + \sum_{i=1}^n \alpha_i c$$

$$= -\boldsymbol{\beta}^\top \sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i - \beta_0 \cancel{\sum_{i=1}^n \alpha_i y_i} + c\sum_{i=1}^n \alpha_i$$

$$\overset{(a)}{=} -\sum_{j=1}^n \alpha_j y_j \boldsymbol{x}_j^\top \sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i + c\sum_{i=1}^n \alpha_i = -\sum_{j=1}^n\sum_{i=1}^n \alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + c\sum_{i=1}^n \alpha_i,$$

where (a) is because of Eqs. (12) and (13).

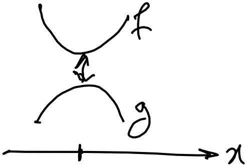
Hard Margin SVM

- Putting the obtained $\boldsymbol{\beta}$ and β_0 from Eqs. (12) and (13), denoted by $\boldsymbol{\beta}^\dagger$ and $\beta_0^\dagger$, in the Lagrangian gives the dual function:

$$\begin{aligned}
\underbrace{g}(\underbrace{\{\alpha_i\}_{i=1}^n}) &= \mathcal{L}(\overset{\frown}{\boldsymbol{\beta}}^\dagger, \overset{\frown}{\beta}_0^\dagger, \{\alpha_i\}_{i=1}^n) \\
&= \underbrace{\frac{1}{2} \overbrace{\sum_{j=1}^n \sum_{i=1}^n \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j} - \overbrace{\sum_{j=1}^n \sum_{i=1}^n \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j} + c \sum_{i=1}^n \alpha_i} \\
&= \underbrace{-\frac{1}{2} \sum_{j=1}^n \sum_{i=1}^n \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j} + c \underbrace{\sum_{i=1}^n \alpha_i}. \qquad (14)
\end{aligned}$$

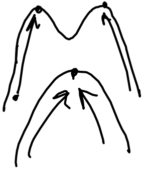
- Also, according to the dual feasibility in Karush-Kuhn-Tucker (KKT) conditions, the dual variable should be non-negative:

$$\underbrace{\alpha_i \geq 0,} \quad \forall i \in \{1, \dots, n\}. \qquad (15)$$



Hard Margin SVM

- Considering Eqs. (13), (14), and (15), the dual optimization problem is [4]:



$$
\begin{aligned}
\underset{\{\alpha_i\}_{i=1}^n}{\text{maximize}} \quad & -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + c\sum_{i=1}^{n}\alpha_i \\
\text{subject to} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0, \\
& \alpha_i \geq 0, \quad \forall i \in \{1,\dots,n\}.
\end{aligned}
\tag{16}
$$

- We define $\boldsymbol{\alpha} := [\alpha_1,\dots,\alpha_n]^\top \in \mathbb{R}^n$, $\boldsymbol{y} := [y_1,\dots,y_n]^\top \in \mathbb{R}^n$, $\boldsymbol{1} := [1,1,\dots,1]^\top \in \mathbb{R}^n$, and a matrix $\boldsymbol{S}$ whose (i,j)-th element is $\boldsymbol{S}(i,j) := (y_i\boldsymbol{x}_i^\top)(y_j\boldsymbol{x}_j)$. Then, Eq. (16) can be stated in vector form as:

Slater's

$$
\begin{aligned}
\underset{\boldsymbol{\alpha}}{\text{maximize}} \quad & -\frac{1}{2}\boldsymbol{\alpha}^\top\boldsymbol{S}\boldsymbol{\alpha} + c\boldsymbol{1}^\top\boldsymbol{\alpha} \\
\text{subject to} \quad & \boldsymbol{y}^\top\boldsymbol{\alpha} = \boldsymbol{0} \xrightarrow{\text{scalar zero}}, \\
& \boldsymbol{\alpha} \succeq \boldsymbol{0}.
\end{aligned}
\tag{17}
$$

- It is a quadratic programming problem. So, it is a concave optimization problem. So, it has one global solution. That is while Perceptron algorithm has various solutions based on its initial random optimization variables.

Hard Margin SVM

KKT conditions:

1. **Stationarity condition**:

$$\underbrace{\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} \overset{\text{set}}{=} \mathbf{0},}_{} \quad \underbrace{\frac{\partial \mathcal{L}}{\partial \beta_0} \overset{\text{set}}{=} 0,}_{} \tag{18}$$

   which resulted in Eqs. (12) and (13).

2. **Primal feasibility**:

$$\underbrace{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) \geq c,}_{} \quad \underbrace{\forall i \in \{1, \dots, n\}.}_{} \tag{19}$$

3. **Dual feasibility**:

$$\rightarrow \underbrace{\alpha_i \geq 0,}_{} \quad \forall i \in \{1, \dots, n\}, \tag{20}$$

   which is already Eq. (15).

$$\overbrace{-y_i(\beta^\top x_i + \beta_0) + c}^{} \leq 0$$

4. **Complementary slackness**:

$$\rightarrow \underbrace{\alpha_i}_{\uparrow}\big(\underbrace{-\,y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c}_{\uparrow}\big) \underline{= 0}, \quad \forall i \in \{1, \dots, n\}. \qquad \boxed{\alpha_i} \tag{21}$$

Hard Margin SVM

KKT conditions:

- We had:

$$\underline{\boxed{\alpha_i}\big(\overbrace{-\, y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c}\big) = 0, \quad \forall i \in \{1, \dots, n\}.}$$

  Also according to Eq. (19), we have:

$$\underline{\overbrace{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0)}\, \overbrace{\geq c}, \quad \forall i \in \{1, \dots, n\}.}$$

  And according to Eq. (20):

$$\underline{\alpha_i \geq 0, \quad \forall i \in \{1, \dots, n\},}$$

- So, there are two cases:

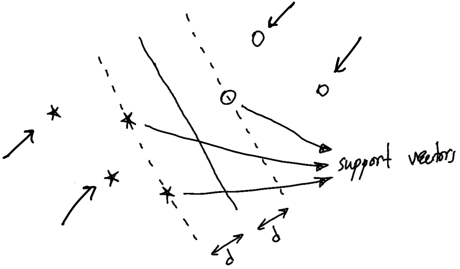
$$\begin{cases} \boxed{\alpha_i = 0} & \text{if } y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) \boxed{> c} \\ \boxed{\alpha_i \geq 0} & \text{if } y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) \boxed{= c.} \end{cases} \tag{22}$$

Hard Margin SVM

- We had:

$$\ast \begin{cases} \boxed{\alpha_i = 0} & \text{if } \boxed{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) > c} \\ \boxed{\alpha_i \geq 0} & \text{if } \boxed{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) = c.} \end{cases}$$

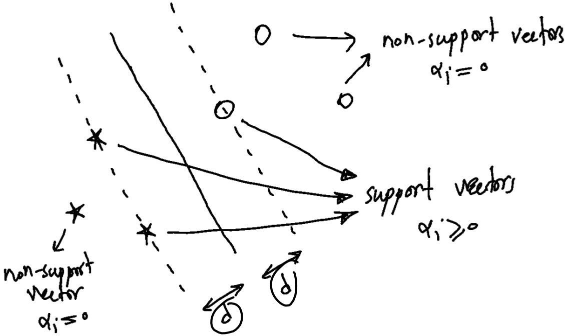
- When $\underline{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) = c}$, it means that the point $\boldsymbol{x}_i$ is on the margin which is the closest distance from the decision boundary. These are called **support vectors** whose α_i values are greater than or equal to zero, according to Eq. (22).
- For points out of the margin, which are farther than the support vectors from the decision boundary, we have $y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) > c$ and therefore $\alpha_i = 0$ based on Eq. (22).



Hard Margin SVM

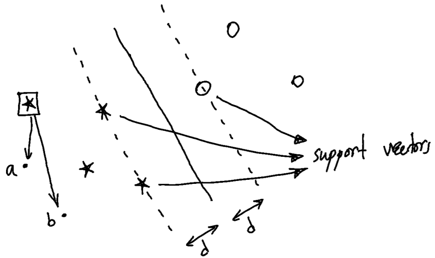
In summary:

- In hard-margin SVM, support vector is defined as the points on the margin boundary.
- According to Eq. (20), the range of α_i, corresponding to the point $\boldsymbol{x}_i$, is $\alpha_i \geq 0$. Various cases happen in this range:
- If the point $\boldsymbol{x}_i$ is a non-support vector, i.e., outside of margins, then $\alpha_i = 0$.
- If the point $\boldsymbol{x}_i$ is a support vector, i.e., on the margin boundary, then $\alpha_i \geq 0$.



Hard Margin SVM

- According to Eq. (16), the cost function is $-\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + c\sum_{i=1}^{n}\alpha_i$. For non-support vectors, we have $\alpha_i = 0$ which makes the cost function zero. Therefore, SVM only cares about support vectors, which are most capable of being misclassified, and the other points are not important. Therefore, SVM is a **sparse** algorithm which is effective for the **"betting on sparsity principle"** [5, 6] and the **Occam's razor** [7].



not important if we move training point ☒ to location a or b.

Hard Margin SVM: Test Phase

- Optimization problem (17) is solved using an optimization toolbox by any optimization algorithm such as the interior-point algorithm. Therefore, we obtain the optimum $\boldsymbol{\alpha}$.

sparse

- The optimum $\boldsymbol{\alpha}$ is put in Eq. (12):

$$\boxed{\boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i,} \longleftarrow \rightarrow \beta$$

to obtain the optimum $\boldsymbol{\beta}$.

- As discussed in KKT conditions, for any support vector $\boldsymbol{x}_i$ (which is on the smallest margin to the decision boundary), we have:

$$\boxed{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) = c.} \longrightarrow \beta_0$$

Putting the obtained $\boldsymbol{\beta}$ in this equation gives β_0. We perform this for all support vectors. Each of them gives a value for β_0. We take the average of these values as the β_0.

- The class label of a point $\boldsymbol{x}$ is estimated as:

$$\widehat{y} = \mathbf{sign}(\underline{\boldsymbol{\beta}^\top \boxed{\boldsymbol{x}} + \beta_0}), \longleftarrow \tag{23}$$

where $\mathbf{sign}(.)$ is the sign function.

$$\begin{cases} \min\limits_{x} \ f(x) \\ \text{s.t. } \boxed{g(x) \le 0} \end{cases} \overset{\approx}{\Longrightarrow} \min_{x} \ f(x) + \underbrace{\boxed{\lambda} g(x)}_{\text{soft penalty}}$$

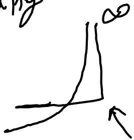
$\underline{0.001}$

$$\min_{x} \ f(x) + \lambda I\big(g(x) \le 0\big) \rightarrow \begin{cases} 0 & g(x) \le 0 \\ \infty & g(x) > 0 \end{cases}$$

hard penalty

Soft Margin SVM

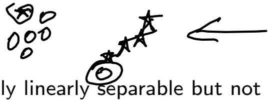
$$\boxed{\min_{x} \ \infty}$$



log-barrier

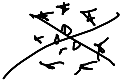
interior-point method

Soft Margin SVM



- When the two classes are roughly linearly separable but not exactly linearly separable, we can use soft margin SVM.
- In this algorithm, we penalize the few points which are misclassified by the decision boundary
- Soft margin SVM adds n additional non-negative scalar variables $\{\zeta_i\}_{i=1}^n$ and changes the optimization problem (11) to:

$$\begin{aligned} & \underset{\boldsymbol{\beta}, \beta_0, \{\zeta_i\}_{i=1}^n}{\text{minimize}} && \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + \gamma \sum_{i=1}^n \zeta_i \\ & \text{subject to} && y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) \geq 1 - \zeta_i, \quad \forall i \in \{1, \dots, n\}, \\ & && \zeta_i \geq 0, \quad \forall i \in \{1, \dots, n\}, \end{aligned} \tag{24}$$



where $\gamma > 0$ is the regularization parameter.

- When $\zeta_i = 0$, the optimization problem reduces to the problem of hard margin SVM. meaning that the point $\boldsymbol{x}_i$ is correctly classified by the decision boundary.
- If a point $\boldsymbol{x}_i$ is misclassified, $\zeta_i > 0$ for that point. Therefore, the less the value of ζ_i, the more accurate the classification is. This is the reason for penalizing the summation of the ζ values.
- Addition of the variables $\{\zeta_i\}_{i=1}^n$ has loosened how hard SVM gets on the classification of the points.

Soft Margin SVM

- We had:

$$\left\{\begin{array}{lll} \underset{\boldsymbol{\beta},\beta_0,\{\zeta_i\}_{i=1}^n}{\text{minimize}} & \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + \gamma \sum_{i=1}^{n} \zeta_i & \\ \text{subject to} & \underline{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) \geq c - \zeta_i}, & \forall i \in \{1, \dots, n\}, \\ & \underline{\zeta_i \geq 0}, \quad \forall i \in \{1, \dots, n\}. & \end{array}\right\}$$

- The constraints can be restated as:

$$y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) \geq c - \zeta_i \implies -y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c - \zeta_i \underline{\leq} 0,$$

$$\underbrace{\zeta_i \geq 0}_{} \implies \underbrace{-\zeta_i \leq 0.}_{}$$

- The Lagrangian:

$$\mathcal{L}(\overbrace{\boldsymbol{\beta}, \beta_0, \{\zeta_i\}_{i=1}^n}^{}, \overbrace{\{\alpha_i\}_{i=1}^n, \{\lambda_i\}_{i=1}^n}^{})$$

$$= \underbrace{\frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + \gamma \sum_{i=1}^{n} \zeta_i}_{} + \underbrace{\sum_{i=1}^{n} \alpha_i\big(-y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c - \zeta_i\big)}_{} + \underbrace{\sum_{i=1}^{n} \lambda_i(-\zeta_i)}_{},$$

where $\underline{\{\alpha_i\}_{i=1}^n \text{ and } \{\lambda_i\}_{i=1}^n \text{ are the Lagrange multipliers}}$.

Soft Margin SVM

$$\xi_1 + \xi_2 + \cdots + \xi_n$$

$$\frac{\partial \mathcal{L}}{\partial \xi_1} = 1$$

- The Lagrangian:

$$★\ \mathcal{L}(\boldsymbol{\beta}, \beta_0, \{\zeta_i\}_{i=1}^n, \{\alpha_i\}_{i=1}^n, \{\lambda_i\}_{i=1}^n)$$

$$= \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + \gamma \sum_{i=1}^n \zeta_i + \sum_{i=1}^n \alpha_i \big(-y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c - \zeta_i\big) + \sum_{i=1}^n \lambda_i(-\zeta_i).$$

- Derivative of Lagrangian w.r.t. $\boldsymbol{\beta}$:

$$★\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta} - \sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i \overset{\text{set}}{=} \boldsymbol{0} \implies \boxed{\boldsymbol{\beta} = \sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i.}\ ★ \tag{25}$$

- Derivative of Lagrangian w.r.t. β_0:

$$\frac{\partial \mathcal{L}}{\partial \beta_0} = -\sum_{i=1}^n \alpha_i y_i \overset{\text{set}}{=} 0 \implies \boxed{\sum_{i=1}^n \alpha_i y_i = 0.}\ ★ \tag{26}$$

- Derivative of Lagrangian w.r.t. ζ_i:

$$\lambda_i = \gamma - \alpha_i.$$

$$\frac{\partial \mathcal{L}}{\partial \zeta_i} = \gamma - \alpha_i - \lambda_i \overset{\text{set}}{=} 0 \implies \boxed{\gamma - \alpha_i - \lambda_i = 0.} \tag{27}$$

Soft Margin SVM

- We substitute Eqs. (25), (26), and (27) in the Lagrangian:

$$\star\quad \mathcal{L}(\boldsymbol{\beta}, \beta_0, \{\zeta_i\}_{i=1}^n, \{\alpha_i\}_{i=1}^n, \{\lambda_i\}_{i=1}^n) = \underbrace{\frac{1}{2}\|\boldsymbol{\beta}\|_2^2} + \gamma \sum_{i=1}^n \zeta_i + \overbrace{\sum_{i=1}^n \alpha_i\big(-y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c - \zeta_i\big)} + \sum_{i=1}^n \lambda_i(-\zeta_i).$$

- The first term:

$$\underbrace{\frac{1}{2}\boxed{\|\boldsymbol{\beta}\|_2^2}} = \frac{1}{2}\boldsymbol{\beta}^\top\boldsymbol{\beta} \overset{(25)}{=} \frac{1}{2}\underbrace{\sum_{j=1}^n \alpha_j y_j \boxed{\boldsymbol{x}_j^\top}}\underbrace{\sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i} = \underbrace{\frac{1}{2}\sum_{j=1}^n\sum_{i=1}^n \alpha_i\alpha_j y_i y_j \underbrace{\boldsymbol{x}_i^\top \boldsymbol{x}_j}}.$$

- The third term:

$$\begin{aligned}
\star\quad \sum_{i=1}^n \alpha_i\big(-y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c - \zeta_i\big) &= \overbrace{-\sum_{i=1}^n \alpha_i y_i \boxed{\boldsymbol{\beta}^\top} \boldsymbol{x}_i} - \overbrace{\sum_{i=1}^n \alpha_i y_i \boxed{\beta_0}} + \overbrace{\sum_{i=1}^n \alpha_i \boxed{c}} - \overbrace{\sum_{i=1}^n \alpha_i \zeta_i} \\
&= -\boxed{\boldsymbol{\beta}^\top}\boxed{\sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i} - \beta_0 \cancel{\sum_{i=1}^n \alpha_i y_i} + c\sum_{i=1}^n \alpha_i - \sum_{i=1}^n \alpha_i \zeta_i \\
&\overset{(a)}{=} -\boxed{\sum_{j=1}^n \alpha_j y_j \boldsymbol{x}_j^\top}\boxed{\sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i} + c\sum_{i=1}^n \alpha_i - \sum_{i=1}^n \alpha_i \zeta_i \\
&= \underbrace{-\sum_{j=1}^n\sum_{i=1}^n \alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j} + \underbrace{c\sum_{i=1}^n \alpha_i - \sum_{i=1}^n \alpha_i \zeta_i},
\end{aligned}$$

where (a) is because of Eqs. (25) and (26).

Soft Margin SVM

- Therefore, the dual function becomes:

$$
\begin{aligned}
g(\{\alpha_i\}_{i=1}^n, \{\lambda_i\}_{i=1}^n) &= \mathcal{L}(\boldsymbol{\beta}^\dagger, \beta_0^\dagger, \{\zeta_i^\dagger\}_{i=1}^n, \{\alpha_i\}_{i=1}^n, \{\lambda_i\}_{i=1}^n) \\
&= \underbrace{\boxed{\frac{1}{2}}\sum_{j=1}^n \sum_{i=1}^n \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j} + \boxed{\gamma \sum_{i=1}^n \zeta_i} \boxed{-} \underbrace{\sum_{j=1}^n \sum_{i=1}^n \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j} \\
&\quad + \underbrace{c \sum_{i=1}^n \alpha_i} \boxed{- \sum_{i=1}^n \alpha_i \zeta_i} + \boxed{\sum_{i=1}^n \lambda_i(-\zeta_i)} \\
&= \boxed{-\frac{1}{2}} \sum_{j=1}^n \sum_{i=1}^n \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + c \underbrace{\sum_{i=1}^n} \alpha_i + \underbrace{\sum_{i=1}^n} \cancel{\overset{= 0}{(\gamma - \alpha_i - \lambda_i)}\zeta_i} \rightarrow 0 \\
&\overset{(27)}{=} \underbrace{-\frac{1}{2} \sum_{j=1}^n \sum_{i=1}^n \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j} + \underbrace{c \sum_{i=1}^n \alpha_i}. \qquad (28)
\end{aligned}
$$

- Also, according to the dual feasibility in Karush-Kuhn-Tucker (KKT) conditions, the dual variable should be non-negative:

$$
\underbrace{\alpha_i \geq 0}, \quad \underbrace{\lambda_i \geq 0}, \quad \underbrace{\forall i \in \{1, \dots, n\}}. \qquad (29)
$$

Soft Margin SVM

- Considering Eqs. (26), (28), and (29), the dual optimization problem is:

$$\left\{\begin{aligned} &\underset{\{\alpha_i\}_{i=1}^n}{\text{maximize}} && -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + c\sum_{i=1}^{n}\alpha_i \\ &\text{subject to} && \sum_{i=1}^{n}\alpha_i y_i = 0, \\ & && \alpha_i \geq 0, \quad \forall i \in \{1,\dots,n\}, \\ & && \boxed{\lambda_i \geq 0, \quad \forall i \in \{1,\dots,n\}.} \end{aligned}\right\} \tag{30}$$

- Comparing Eqs. (16) and (30) shows that the only difference between hard margin and soft margin SVM algorithms is addition of the constraints $\lambda_i \geq 0, \forall i$ in the soft margin SVM.
- According to Eq. (27), the constraint $\lambda_i \geq 0$ can be restated as:

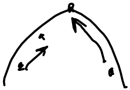
$$\lambda_i \geq 0 \overset{(27)}{\Longrightarrow} \gamma - \alpha_i \geq 0 \Longrightarrow \boxed{\alpha_i \leq \gamma.} \tag{31}$$

Soft Margin SVM

- Combining this equation with the constraint $\alpha_i \geq 0$ can restate the optimization problem as:

$$\left\{\begin{array}{ll} \underset{\{\alpha_i\}_{i=1}^n}{\text{maximize}} & -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + c\sum_{i=1}^{n}\alpha_i \\ \text{subject to} & \sum_{i=1}^{n}\alpha_i y_i = 0, \\ & \boxed{0 \leq \alpha_i \leq \gamma,} \quad \forall i \in \{1, \dots, n\}. \end{array}\right\} \tag{32}$$

- Comparing Eqs. (16) and (32) shows that the only difference between hard margin and soft margin SVM algorithms is the upper bound γ on the optimization variables.
- Defining $\underline{\boldsymbol{\alpha} := [\alpha_1, \dots, \alpha_n]^\top} \in \mathbb{R}^n$, $\underline{\boldsymbol{y} := [y_1, \dots, y_n]^\top} \in \mathbb{R}^n$, $\underline{\mathbf{1} := [1, 1, \dots, 1]^\top} \in \mathbb{R}^n$, and a matrix $\boldsymbol{S}$ whose $\underline{(i,j)\text{-th element is } \boldsymbol{S}(i,j) := (y_i \boldsymbol{x}_i^\top)(y_j \boldsymbol{x}_j)}$, the problem can be stated in vector form as:



$$\left\{\begin{array}{ll} \underset{\boldsymbol{\alpha}}{\text{maximize}} & -\frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{S}\boldsymbol{\alpha} + c\mathbf{1}^\top\boldsymbol{\alpha} \\ \text{subject to} & \boldsymbol{y}^\top\boldsymbol{\alpha} = \mathbf{0}, \\ & \mathbf{0} \preceq \boldsymbol{\alpha} \preceq \gamma\mathbf{1}. \end{array}\right\} \tag{33}$$

- It is a quadratic programming problem. So, it is a concave optimization problem. So, it has one global solution. That is while Perceptron algorithm has various solutions based on its initial random optimization variables.

Soft Margin SVM

- The test phase of the soft margin SVM is the same as the test phase of the hard margin SVM.

KKT conditions:

1. **Stationarity condition**:

$$\underline{\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} \overset{\text{set}}{=} \mathbf{0}, \quad \frac{\partial \mathcal{L}}{\partial \beta_0} \overset{\text{set}}{=} 0, \quad \frac{\partial \mathcal{L}}{\partial \zeta_i} \overset{\text{set}}{=} 0.} \tag{34}$$

which resulted in Eqs. (25), (26), and (27).

2. **Primal feasibility**:

$$\underline{y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) \geq c - \zeta_i, \quad \forall i \in \{1, \dots, n\}}. \tag{35}$$

3. **Dual feasibility**:

$$\underline{\alpha_i \geq 0, \quad \lambda_i \geq 0,} \quad \underline{\forall i \in \{1, \dots, n\}}, \tag{36}$$

which is already Eq. (29).

4. **Complementary slackness**:

$$\boxed{\alpha_i\big(-y_i(\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) + c - \zeta_i\big) = 0,} \quad \forall i \in \{1, \dots, n\}, \tag{37}$$

$$\lambda_i(-\zeta_i) = 0 \implies \boxed{\lambda_i \zeta_i = 0,} \quad \forall i \in \{1, \dots, n\}. \tag{38}$$

Soft Margin SVM

- We found:

$$\begin{cases}\alpha_i\big(-y_i(\boldsymbol{\beta}^\top \mathbf{x}_i+\beta_0)+c-\zeta_i\big)=0, & \forall i\in\{1,\dots,n\},\\ \lambda_i(-\zeta_i)=0 \implies \lambda_i\zeta_i=0, & \forall i\in\{1,\dots,n\}.\end{cases}$$

- According to the primal feasibility, $y_i(\boldsymbol{\beta}^\top \mathbf{x}_i+\beta_0)\geq c-\zeta_i$.
- If $y_i(\boldsymbol{\beta}^\top \mathbf{x}_i+\beta_0)>c-\zeta_i$, then $\alpha_i=0$ according to Eq. (37). It means the point is a non-support vector and outside of the margins. Setting $\alpha_i=0$ in the cost function of problem (30) makes the cost zero; therefore, soft-margin SVM also does not care about the non-support vector points.
- If $y_i(\boldsymbol{\beta}^\top \mathbf{x}_i+\beta_0)=c-\zeta_i$, then $\alpha_i\geq 0$ according to Eq. (37). It means the point is a support vector and either on the margin or inside the margin (violating the margin).
  - Dual feasibility: $\zeta_i\geq 0$.
  - If $\zeta_i>0$ (violate margin, i.e., it passes the margin): According to Eq. (38), $\lambda_i=0$. According to Eq. (27):

$$\gamma-\alpha_i-\lambda_i=0 \implies \lambda_i=\gamma-\alpha_i=0 \implies \alpha_i=\gamma.$$

  - If $\zeta_i=0$ (not violate margin, i.e., exactly on the margin border): According to Eqs. (36) and (38), $\lambda_i\geq 0$. The case $\lambda_i=0$ is like above case and we have $\alpha_i=\gamma$. However, for case $\lambda_i>0$, according to Eq. (27):

$$\gamma-\alpha_i-\lambda_i=0 \implies \lambda_i=\gamma-\alpha_i>0 \implies \alpha_i<\gamma.$$

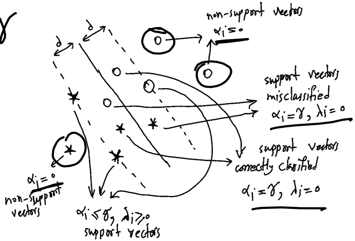
  Overall, in case there is $\zeta_i=0$, i.e., the point is on the margin border, we have $\alpha_i\leq\gamma$.

Soft Margin SVM

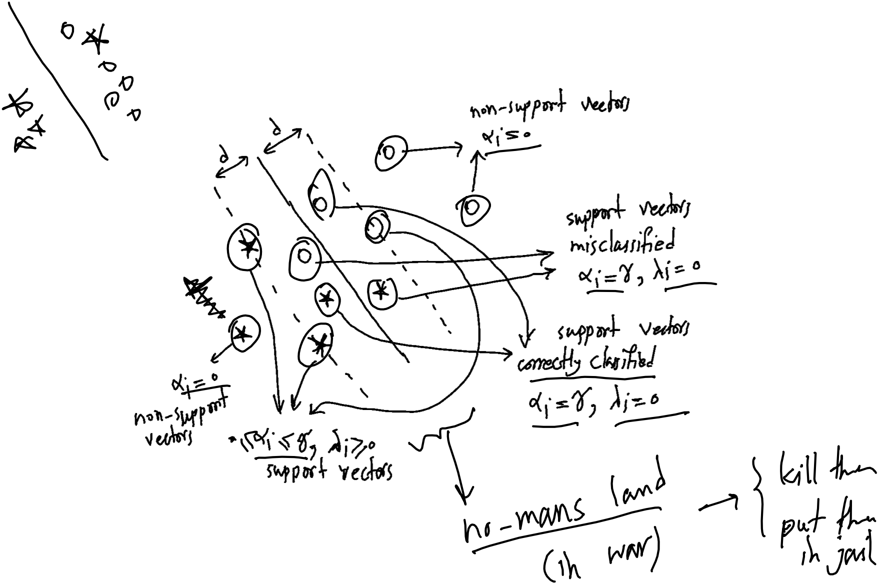
In summary:

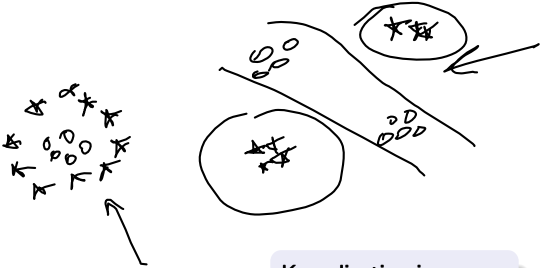
- In soft-margin SVM, support vector is defined as the points inside of margins, either on the margin boundary or in the margin.
- According to the constraint in Eq. (32), the range of α_i, corresponding to the point $\boldsymbol{x}_i$, is $0 \leq \alpha_i \leq \gamma$. Various cases happen in this range:
- If the point $\boldsymbol{x}_i$ is a non-support vector, i.e., outside of margins, then $\alpha_i = 0$.
- If the point $\boldsymbol{x}_i$ is a support vector, i.e., inside of margins (either on the margin boundary or in the margin), then $\alpha_i \geq 0$:
  - If the point violates the margin, i.e., it passes the margin, then $\boxed{\lambda_i = 0}$, $\boxed{\alpha_i = \gamma}$.
    - The point $\boldsymbol{x}_i$ is either in the margin but correctly classified.
    - or the point $\boldsymbol{x}_i$ is in the margin but in the other side of decision boundary, i.e., is misclassified.
  - If the point does not violate the margin, i.e., it is exactly on the margin border, then $\boxed{\lambda_i \geq 0}$, $\boxed{\alpha_i \leq \gamma}$.

$0 \leq \alpha_i \leq \gamma$



Soft Margin SVM

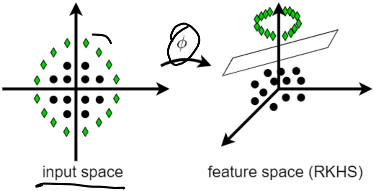




Kernelization in Machine Learning

Kernelization

- Linear algorithms cannot properly handle nonlinear patterns of data obviously.
- When dealing with **nonlinear data**, if the algorithm is linear, two solutions exist to have acceptable performance:
  1. Either the linear method should be modified to become nonlinear or a completely new **nonlinear algorithm** should be proposed to be able to handle nonlinear data. Some examples of this category are nonlinear dimensionality methods such as locally linear embedding [8] and Isomap [9].
  2. Or the **nonlinear data should be modified in a way to become more linear in pattern**. In other words, a transformation should be applied on data so that the pattern of data becomes roughly linear or **easier to process by the linear algorithm**. Some examples of this category are kernel versions of linear methods such as kernel Principal Component Analysis (PCA) [10, 11, 12], kernel Fisher Discriminant Analysis (FDA) [13, 14], and kernel Support Vector Machine (SVM) [15, 16].
- The second approach is called **kernelization** in machine learning which we define in the following.
- This figure shows how kernelization for transforming data can help separate classes for better classification.



Kernel Trick

- Generally, there exist two main approaches for kernelization in machine learning. These two approaches are related in theory but have two ways for kernelization. These approaches are:
  - Kernelization by **Kernel Trick**, e.g., in kernel SVM
  - Kernelization by **Representation Theory**, e.g., in kernel FDA

  Here, we explain the kernel trick used in kernel SVM.

Kernel Trick

Definition (Feature Map or Pulling Function)

We define the mapping:

$$\boxed{\phi : \mathcal{X} \to \mathcal{H},} \tag{39}$$

to transform data from the input space to the feature space, i.e. Hilbert space. In other words, this mapping pulls data to the feature space:

$$\boxed{\boldsymbol{x} \mapsto \phi(\boldsymbol{x}).} \tag{40}$$

The function $\phi(\boldsymbol{x})$ is called the **feature map** or **pulling function**. The feature map is a (possibly infinite-dimensional) vector whose elements are [17]:

$$\boxed{\phi(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \dots]^\top.} \tag{41}$$

- Let t denote the dimensionality of $\phi(\boldsymbol{x})$. The feature map may be infinite or finite dimensional, i.e. t can be infinity; it is usually a very large number (the Hilbert space may have infinite number of dimensions).

Kernel Trick

$$s(x_1, x_2) = \frac{\boxed{x_1^\top x_2}}{\|x_1\|_2 \|x_2\|_2} \longleftarrow$$

- One technique to kernelize an algorithm is **kernel trick**. In this technique, we first try to formulate the algorithm formulas or optimization in a way that **data always appear as inner product of data instances and not a data instance alone**. In other words, the formulation of algorithm should only have $\boldsymbol{x}^\top \boldsymbol{x}$, $\boldsymbol{x}^\top \boldsymbol{X}$, $\boldsymbol{X}^\top \boldsymbol{x}$, or $\boldsymbol{X}^\top \boldsymbol{X}$ and not a lonely $\boldsymbol{x}$ or $\boldsymbol{X}$.
- In this way, kernel trick replaces $\boldsymbol{x}^\top \boldsymbol{x}$ with $\underline{\phi(\boldsymbol{x})^\top \phi(\boldsymbol{x})}$ [4]:

  scalar

$$\boxed{\boldsymbol{x}^\top \boldsymbol{x}} \mapsto \boxed{\boxed{\phi(\boldsymbol{x})}^\top \phi(\boldsymbol{x})} = \boxed{k(\boldsymbol{x}, \boldsymbol{x})}. \tag{42}$$

  because kernel can be computed by inner product between pulled data instances to the RKHS (see our tutorial [18] for more information).
- Therefore, the inner products of points are all replaced with the kernel between points. The matrix form of kernel trick is:

  matrix

$$\boxed{\boldsymbol{X}^\top \boldsymbol{X}} \mapsto \underline{\boldsymbol{\Phi}(\boldsymbol{X})}^\top \underline{\boldsymbol{\Phi}(\boldsymbol{X})} = \boxed{\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})} \in \underline{\mathbb{R}^{n \times n}}. \tag{43}$$

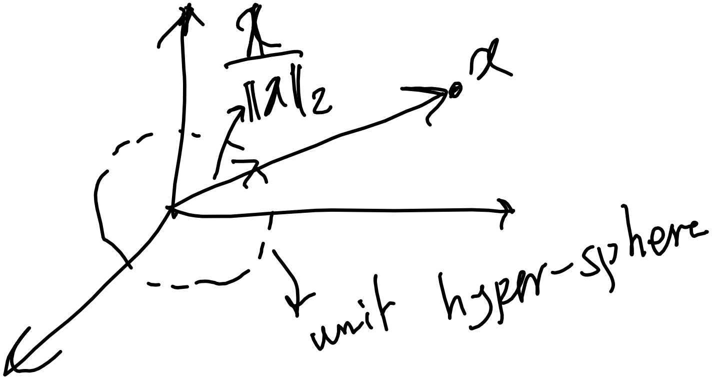
- Most often, kernel matrix is computed over one dataset; hence, its dimensionality is $n \times n$. However, in some cases, the kernel matrix is computed between two sets of data instances with sample sizes n_1 and n_2 for example, i.e. datasets $\boldsymbol{X}_1 := [\boldsymbol{x}_{1,1}, \dots, \boldsymbol{x}_{1,n_1}]$ and $\boldsymbol{X}_2 := [\boldsymbol{x}_{2,1}, \dots, \boldsymbol{x}_{2,n_2}]$. In this case, the kernel matrix has size $n_1 \times n_2$ and the kernel trick is:

$X_1 \in \mathbb{R}^{d \times n_1}$

$X_2 \in \mathbb{R}^{d \times n_2}$

$$\underline{\boldsymbol{x}_{1,i}^\top}\underline{\boldsymbol{x}_{1,j}} \mapsto \underline{\phi(\boldsymbol{x}_{1,i})}^\top \underline{\phi(\boldsymbol{x}_{1,j})} = \underline{k(\boldsymbol{x}_{1,i}, \boldsymbol{x}_{1,j})}, \tag{44}$$

$$\underline{\boldsymbol{X}_1^\top \boldsymbol{X}_2} \mapsto \boldsymbol{\Phi}(\boldsymbol{X}_1)^\top \boldsymbol{\Phi}(\boldsymbol{X}_2) = \underline{\boldsymbol{K}(\boldsymbol{X}_1, \boldsymbol{X}_2)} \in \boxed{\mathbb{R}^{n_1 \times n_2}}. \longleftarrow \tag{45}$$



$$s(x_1, x_2) = \frac{x_1^T x_2}{\|x_1\|_2 \, \|x_2\|_2}$$

$$= \left(\frac{x_1}{\|x_1\|_2}\right)^T \left(\frac{x_2}{\|x_2\|_2}\right)$$

Well-known Kernel Functions

- **Linear Kernel:** Linear kernel is the simplest kernel which is the inner product of points:

$$\boxed{k(\boldsymbol{x}, \boldsymbol{y}) := \boldsymbol{x}^\top \boldsymbol{y}.} \tag{46}$$

  - Comparing this with Eq. (44) shows that in linear kernel we have $\boxed{\phi(\boldsymbol{x}) = \boldsymbol{x}.}$
  - Hence, in this kernel, the feature map is explicitly known.
  - Note that $\phi(\boldsymbol{x}) = \boldsymbol{x}$ shows that data are not pulled to any other space in linear kernel but in the input space, the inner products of points are calculated to obtain the feature space.
  - If we use kernel trick, the kernelized algorithm with a linear kernel is equivalent to the non-kernelized algorithm. This is because in linear kernel, we have $\phi(\boldsymbol{x}) = \boldsymbol{x}$ and $k(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\top \boldsymbol{y}$ according to Eq. (46). So, the kernel trick, which is Eq. (44), maps data as $\boldsymbol{x}^\top \boldsymbol{y} \mapsto \phi(\boldsymbol{x})^\top \phi(\boldsymbol{y}) = \boldsymbol{x}^\top \boldsymbol{y}$ for linear kernel. Therefore, linear kernel does not have any effect when using kernel trick. Examples for this are kernel PCA [10, 11, 12] and kernel SVM [15, 16] which are equivalent to PCA and SVM, respectively, if linear kernel is used.
  - However, linear kernel does have impact when using kernelization by representation theory because it finds the inner products of pulled data points after pulling the solution and representation as a span of bases. Hence, kernelized algorithm using representation theory with linear kernel is not equivalent to non-kernelized algorithm. Examples of this are kernel FDA [13, 14] and kernel supervised PCA [19, 12] which are different from FDA and supervised PCA, respectively, even if linear kernel is used. For more information about kernelization by representation theory, see our tutorial [18].

Well-known Kernel Functions

- **Radial Basis Function (RBF) or Gaussian Kernel:** RBF kernel has a scaled Gaussian (or normal) distribution where the normalization factor of distribution is usually ignored. Hence, it is also called the Gaussian kernel. The RBF kernel is formulated as:

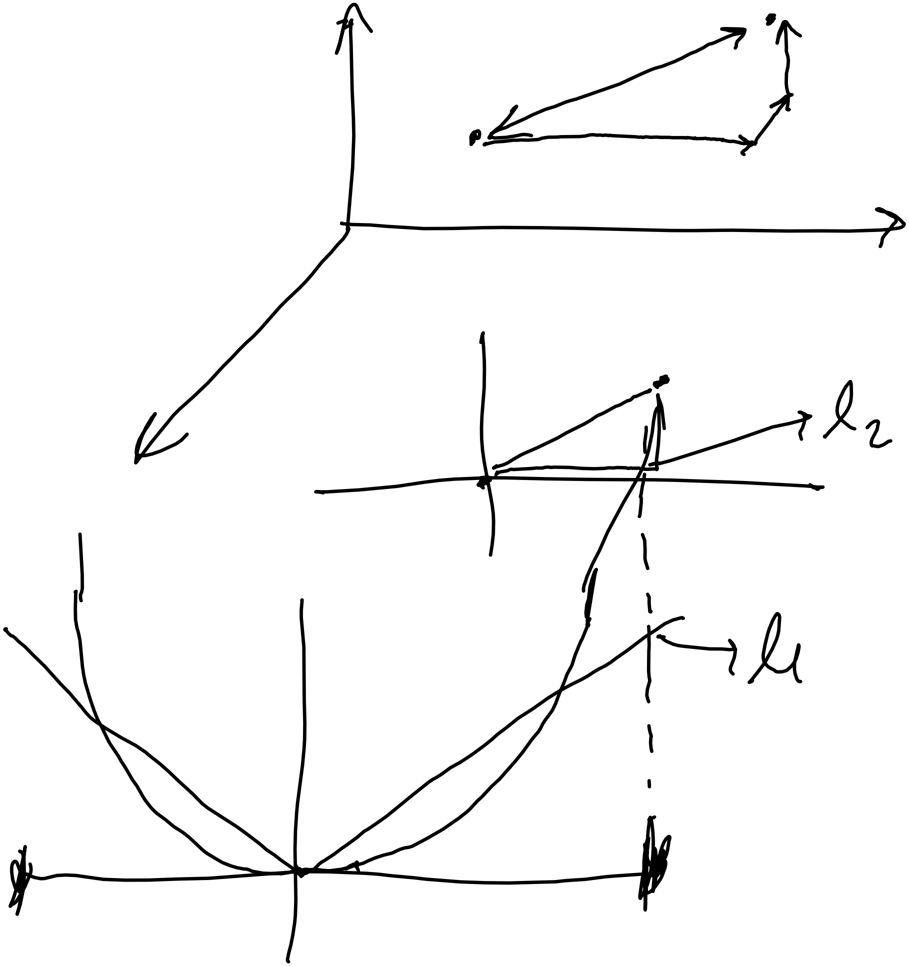
$$\underbrace{k(\boldsymbol{x}, \boldsymbol{y})}_{} := \underbrace{\exp(-\boxed{\gamma}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2)}_{} = \underbrace{\exp(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|_2^2}{\boxed{\sigma^2}})}_{}, \leq \Phi(x)^T \Phi(y) \quad (47)$$

  where $\boxed{\gamma := 1/\sigma^2}$ and σ^2 is the variance of kernel. A proper value for this parameter is $\boxed{\gamma = 1/d}$ where d is the dimensionality of data. Note that RBF kernel has also been widely used in RBF networks [20] and kernel density estimation [21].

- **Laplacian Kernel:** The Laplacian kernel, also called the Laplace kernel, is similar to the RBF kernel but with ℓ_1 norm rather than squared ℓ_2 norm. The Laplacian kernel is:

$$k(\boldsymbol{x}, \boldsymbol{y}) := \underbrace{\exp(-\gamma\, \|\boldsymbol{x} - \boldsymbol{y}\|_1)}_{} = \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|_1}{\sigma^2}), \quad (48)$$

  where $\|\boldsymbol{x} - \boldsymbol{y}\|_1$ is also called the Manhattan distance. A proper value for this parameter is $\gamma = 1/d$ where d is the dimensionality of data. In some specific fields of science, the Laplacian kernel has been found to perform better than Gaussian kernel [22]. This makes sense because of betting on sparsity principal [23] since ℓ_1 norm makes algorithm sparse. Note that ℓ_2 norm in RBF kernel is also more sensitive to noise; however, the computation and derivative of ℓ_1 norm is more difficult than ℓ_2 norm.


ℓ_2
ℓ_1

Well-known Kernel Functions

- **Sigmoid Kernel:** Sigmoid kernel is a hyperbolic tangent function applied on inner product of points. It is formulated as:

$$\boxed{k(\boldsymbol{x}, \boldsymbol{y}) := \tanh(\gamma \boldsymbol{x}^\top \boldsymbol{y} + c),} \tag{49}$$

  where $\gamma > 0$ is the slope and c is the intercept. Some proper values for these parameters are $\gamma = 1/d$ and $c = 1$ where d is the dimensionality of data. Note that the hyperbolic tangent function is also used widely for activation functions in neural networks [24].

- **Polynomial Kernel:** Polynomial kernel applies a polynomial function with degree δ (a positive integer) on inner product of points:

$$k(\boldsymbol{x}, \boldsymbol{y}) := (\gamma \boldsymbol{x}^\top \boldsymbol{y} + c)^d, \tag{50}$$

  where $\gamma > 0$ is the slope and c is the intercept. Some proper values for these parameters are $\gamma = 1/d$ and $c = 1$ where d is the dimensionality of data.

Well-known Kernel Functions

- **Cosine Kernel:** Kernel is a measure of similarity [18] and computes the inner product between points in the feature space. Cosine kernel computes the similarity between points. It is obtained from the formula of cosine and inner product:

$$k(\boldsymbol{x}, \boldsymbol{y}) := \underbrace{\cos(\boldsymbol{x}, \boldsymbol{y})}_{} = \frac{\boldsymbol{x}^\top \boldsymbol{y}}{\underbrace{\|\boldsymbol{x}\|_2 \, \|\boldsymbol{y}\|_2}_{}}. \tag{51}$$

  The normalization in the denominator projects the points onto a unit hyper-sphere so that the inner product measures the similarity of their angles regardless of their lengths. Note that angle-based measures such as cosine are found to work better for face recognition compared to Euclidean distances [25].

- **Chi-squared Kernel:** Assume $\boldsymbol{x}(j)$ denotes the j-th dimension of the d-dimensional point $\boldsymbol{x}$. The Chi-squared (χ^2) kernel is [26]:

$$\underbrace{k(\boldsymbol{x}, \boldsymbol{y}) := \exp\Big(-\gamma \sum_{j=1}^{d} \frac{\big(\boldsymbol{x}(j) - \boldsymbol{y}(j)\big)^2}{\boldsymbol{x}(j) + \boldsymbol{y}(j)}\Big),}_{} \tag{52}$$

  where $\gamma > 0$ is a parameter (a proper value is $\gamma = 1$). Note that the summation term inside exponential (without the minus) is the Chi-squared distance which is related to the Chi-squared test in statistics.

Kernel SVM

Kernel SVM

- Kernel SVM was proposed by Vladimir Vapnik *et al.* in years 1992-1995 [15, 16].
- Recall Eq. (16) which is the dual optimization problem in hard margin SVM:

$$\left\{\begin{array}{ll} \underset{\{\alpha_i\}_{i=1}^n}{\text{maximize}} & -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j \boxed{\boldsymbol{x}_i^\top \boldsymbol{x}_j} + c\sum_{i=1}^{n}\alpha_i \\ \text{subject to} & \sum_{i=1}^{n}\alpha_i y_i = 0, \\ & \alpha_i \geq 0, \quad \forall i \in \{1,\dots,n\}. \end{array}\right.$$

- For kernelization using kernel trick, we can replace $\underline{\boldsymbol{x}_i^\top \boldsymbol{x}_j}$ with $\underline{\phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j)} = \underline{\boldsymbol{k}(\boldsymbol{x}_i, \boldsymbol{x}_j)}$:

$$\left\{\begin{array}{ll} \underset{\{\alpha_i\}_{i=1}^n}{\text{maximize}} & -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j \boxed{\boldsymbol{k}(\boldsymbol{x}_i, \boldsymbol{x}_j)} + c\sum_{i=1}^{n}\alpha_i \\ \text{subject to} & \sum_{i=1}^{n}\alpha_i y_i = 0, \\ & \alpha_i \geq 0, \quad \forall i \in \{1,\dots,n\}. \end{array}\right. \tag{53}$$

- Note that it is possible to use kernel trick in soft-margin SVM also; however, it is not required as soft-margin SVM tried to handle nonlinear classes while kernel trick can do this even when it is used with the hard margin SVM.
- Analysis of KKT conditions of kernel trick is the same as we did for hard-margin SVM.

Kernel SVM vs. Neural Networks

- Because of usage of the kernel function, kernel SVM can handle nonlinearly separable classes.
- Kernel SVM was winning the competition with in 1990's and start of the 21st century. This time period is referred to as the winter of neural networks. The reasons of the overcome of kernel SVM over neural networks were:
  - Kernel SVM is mathematically sound and solid.
  - Kernel SVM was able to perform nonlinear classification.
  - at that time, multilayer Perceptron neural networks could not become deep because of gradient vanishing or gradient explosion problem.
- Since about 2006, neural networks could win the battle back against kernel SVM. The reasons for this winning back were:
  - Kernel SVM required choice of kernel function. It was not obvious which kernel function is best for a dataset. Usually, the RBF kernel function was used by default.
  - Calculation of kernel function is time consuming and thus cannot handle big data with a huge n. This is while neural networks could handle big data. At that time, the world was facing the explosion of data and information and handling big data was crucial.
  - The problems of gradient vanishing and explosion were resolved using various techniques such as initialization by Restricted Boltzmann Machines (RBM) (2006) [27, 28], ReLU activation function (2011) [29] and the dropout technique (2014) [30].

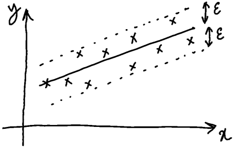
More Notes

Some More Notes

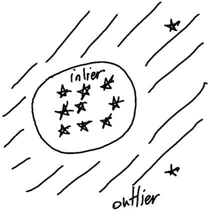
- For multi-class classification, we can consider every pairs of classes as a binary classification. Therefore, a multi-class classification can be considered as a set of binary classification problems.
- SVM can also be used for regression (Support Vector Regression - SVR) [31] and outlier detection (one-class SVM) [32], too. Here, we went through SVM and kernel SVM for binary classification.
- SVR:

$$
\begin{aligned}
& \underset{\boldsymbol{\beta},\beta_0}{\text{minimize}} && \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 \\
& \text{subject to} && y_i - (\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) \leq \varepsilon, \quad \forall i \in \{1, \dots, n\}, \\
& && (\boldsymbol{\beta}^\top \boldsymbol{x}_i + \beta_0) - y_i \leq \varepsilon, \quad \forall i \in \{1, \dots, n\}.
\end{aligned} \tag{54}
$$

- One-class SVM: It considers a class of inliers and a class of outliers. So, it is a binary classification again.



SVR



kernel one-class SVM

SVM and Kernel SVM in Sklearn

- SVM in sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
- Kernel SVM in sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- Linear SVM is much faster than Kernel SVM in sklearn. Sklearn itself recommends using LinearSVC for large datasets.
- Tutorial page for SVM in sklearn: https://scikit-learn.org/stable/modules/svm.html

Acknowledgment

- For more information on KKT conditions, dual problem, method of Lagrange multipliers, etc, refer to my Optimization Techniques course at the University of Guelph, whose lectures are on my youTube channel.
- For more information on KKT conditions, dual problem, method of Lagrange multipliers, etc, see our tutorial paper: Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, Mark Crowley. "KKT Conditions, First-Order and Second-Order Optimization, and Distributed Optimization: Tutorial and Survey." arXiv preprint arXiv:2110.01858 (2021). [33]
- For more information on kernels, see our tutorial paper: Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, Mark Crowley. "Reproducing Kernel Hilbert Space, Mercer's Theorem, Eigenfunctions, Nyström Method, and Use of Kernels in Machine Learning: Tutorial and Survey." arXiv preprint arXiv:2106.08443 (2021). [18]
- Some slides of this slide deck are inspired by teachings of Prof. Ali Ghodsi at University of Waterloo, Department of Statistics.

References

[1] F. Rosenblatt, “The Perceptron – a perceiving and recognizing automaton project para,” tech. rep., Report 85-460-1, Cornell Aeronautical Laboratory., 1957.

[2] D. O. Hebb, *The Organization of Behavior*.
New York: Wiley & Sons, 1949.

[3] V. Vapnik and A. Chervonenkis, *Theory of pattern recognition*.
Nauka, Moscow, 1974.

[4] C. J. Burges, “A tutorial on support vector machines for pattern recognition,” *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[5] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning: Data Mining, Inference, and Prediction*, vol. 2.
Springer series in statistics, New York, NY, USA, 2009.

[6] R. Tibshirani, M. Wainwright, and T. Hastie, *Statistical learning with sparsity: the lasso and generalizations*.
Chapman and Hall/CRC, 2015.

[7] P. Domingos, “The role of Occam's razor in knowledge discovery,” *Data mining and knowledge discovery*, vol. 3, no. 4, pp. 409–425, 1999.

[8] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, “Locally linear embedding and its variants: Tutorial and survey,” *arXiv preprint arXiv:2011.10925*, 2020.

References (cont.)

[9] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Multidimensional scaling, Sammon mapping, and Isomap: Tutorial and survey," *arXiv preprint arXiv:2009.08136*, 2020.

[10] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel principal component analysis," in *International conference on artificial neural networks*, pp. 583–588, Springer, 1997.

[11] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[12] B. Ghojogh and M. Crowley, "Unsupervised and supervised principal component analysis: Tutorial," *arXiv preprint arXiv:1906.03148*, 2019.

[13] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K.-R. Mullers, "Fisher discriminant analysis with kernels," in *Neural networks for signal processing IX: Proceedings of the 1999 IEEE signal processing society workshop*, pp. 41–48, IEEE, 1999.

[14] B. Ghojogh, F. Karray, and M. Crowley, "Fisher and kernel Fisher discriminant analysis: Tutorial," *arXiv preprint arXiv:1906.09436*, 2019.

[15] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, 1992.

[16] V. Vapnik, *The nature of statistical learning theory*.
Springer science & business media, 1995.

References (cont.)

[17] H. Q. Minh, P. Niyogi, and Y. Yao, “Mercer's theorem, feature maps, and smoothing,” in *International Conference on Computational Learning Theory*, pp. 154–168, Springer, 2006.

[18] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, “Reproducing kernel Hilbert space, Mercer's theorem, eigenfunctions, Nystr\" om method, and use of kernels in machine learning: Tutorial and survey,” *arXiv preprint arXiv:2106.08443*, 2021.

[19] E. Barshan, A. Ghodsi, Z. Azimifar, and M. Z. Jahromi, “Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds,” *Pattern Recognition*, vol. 44, no. 7, pp. 1357–1371, 2011.

[20] M. J. L. Orr, “Introduction to radial basis function networks,” tech. rep., Center for Cognitive Science, University of Edinburgh, 1996.

[21] D. W. Scott, *Multivariate density estimation: theory, practice, and visualization*.
John Wiley & Sons, 1992.

[22] M. Rupp, “Machine learning for quantum mechanics in a nutshell,” *International Journal of Quantum Chemistry*, vol. 115, no. 16, pp. 1058–1073, 2015.

[23] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*.
Springer Science & Business Media, 2009.

[24] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1.
MIT press Cambridge, 2016.

References (cont.)

[25] V. Perlibakas, “Distance measures for PCA-based face recognition,” *Pattern recognition letters*, vol. 25, no. 6, pp. 711–724, 2004.

[26] J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid, “Local features and kernels for classification of texture and object categories: A comprehensive study,” *International journal of computer vision*, vol. 73, no. 2, pp. 213–238, 2007.

[27] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[28] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[29] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, JMLR Workshop and Conference Proceedings, 2011.

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[31] A. J. Smola and B. Schölkopf, “A tutorial on support vector regression,” *Statistics and computing*, vol. 14, pp. 199–222, 2004.

References (cont.)

[32] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, “Support vector method for novelty detection,” *Advances in neural information processing systems*, vol. 12, 1999.

[33] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, “KKT conditions, first-order and second-order optimization, and distributed optimization: Tutorial and survey,” *arXiv preprint arXiv:2110.01858*, 2021.