Applied Machine Learning
Class 4

Master 2 Physique Fondamentale et Applications

Daniel Förster
daniel.forster@univ-orleans.fr
UFR Sciences et Techniques Université d'Orléans

Chapter 3: Supervised Learning: Classification (and Regression) D: Support Vector Machines

Introduction to Support Vector Machines (SVM)

- SVM is a supervised learning algorithm used for classification tasks.
- The input dataset has feature vectors (x) and labels ($y \in \{-1, +1\}$).
- Goal: Find a decision boundary (hyperplane) to separate positive ($+1$) and negative (-1) examples.

Mathematical Representation:

- Hyperplane equation:

$$\mathrm{w} \cdot \mathrm{x} - b = 0$$

  where w is the weight vector, and b is the bias.

- Prediction:

$$y = \mathrm{sign}(\mathrm{w} \cdot \mathrm{x} - b)$$

Optimization Problem:

://themlbook.com/

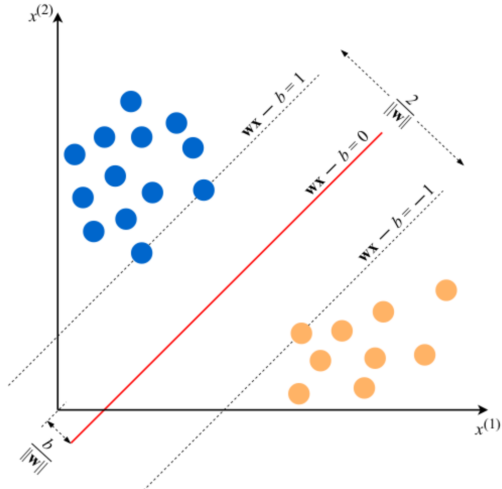
Objective:

- Minimize: $\|\mathrm{w}\|$ (maximize margin)
- Subject to:

$$y_i(\mathrm{w} \cdot \mathrm{x}_i - b) \geq 1, \quad \forall i$$

$\rightarrow$ Maximizing the margin improves generalization to new data.

Image source [1]



Why Does Minimizing $\|w\|$ Maximize the Margin?

The margin is the perpendicular distance between the two supporting hyperplanes:

$$w \cdot x - b = +1 \quad \text{and} \quad w \cdot x - b = -1$$

Formula for the margin:

$$\text{Margin} = \frac{2}{\|w\|}$$

Geometric Intuition:

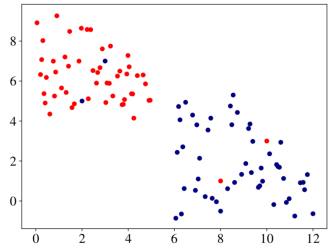
- The distance from a point x to a hyperplane:

$$\text{Distance} = \frac{|w \cdot x - b|}{\|w\|}$$

- For supporting hyperplanes, this simplifies to:

$$\frac{|+1-(-1)|}{\|w\|} = \frac{2}{\|w\|}$$

Dealing with Noise in SVMs: Soft-Margin SVMs



- Data may not be **linearly separable**.
- Add **hinge loss function**:

$$\text{Hinge Loss} = \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i - b))$$

penalizes misclassified points proportionally to the distance to the hyperplane.

Optimization Problem:

$$\min_{\mathbf{w},b} \quad C\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i - b))$$

- **First Term:** Maximizes margin ($\|\mathbf{w}\|^2$).
- **Second Term:** Penalizes hinge loss.

Hyperparameter C:

- Balances the tradeoff between:
  - **Margin size:** Larger margins improve generalization.
  - **Classification errors:** Penalizing misclassifications.

Image source [1]

The Kernel Trick: Transforming Non-Separable Data

- Some datasets are not linearly separable in their original space.
- Solution: Transform data into a higher-dimensional space where it becomes linearly separable.
- Transformation is done using a mapping function $\phi(\mathrm{x})$:

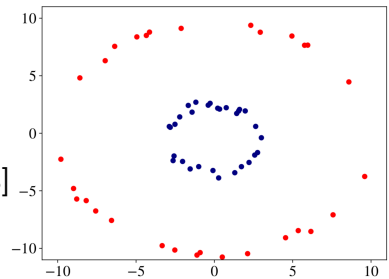
$$\phi(\mathrm{x}) = [\text{transformed features in higher dimensions}]$$

Image source [1]

Example:

- Transform 2D data $[q, p]$ into 3D space:

$$\phi([q, p]) = [q^2, \sqrt{2}qp, p^2]$$



The Kernel Trick

- The SVM optimization problem involves dot products of transformed feature vectors:

$$\phi(\mathrm{x}_i) \cdot \phi(\mathrm{x}_k)$$

- Computing $\phi(\mathrm{x})$ explicitly is costly.
- The **kernel trick** computes the dot product directly in the original space:

$$k(\mathrm{x}_i, \mathrm{x}_k) = \phi(\mathrm{x}_i) \cdot \phi(\mathrm{x}_k)$$

Image source [1]

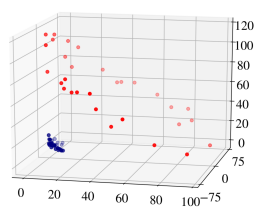
Example: Quadratic Kernel

- Instead of transforming:

$$\phi([q, p]) = [q^2, \sqrt{2}qp, p^2]$$

- Use the kernel directly:

$$k(\mathrm{x}_i, \mathrm{x}_k) = (\mathrm{x}_i \cdot \mathrm{x}_k)^2$$



Chapter 3: Supervised Learning: Classification and Regression
E: k-Nearest Neighbors

k-Nearest Neighbors

- kNN is one of the simplest supervised learning algorithms.
- It is a lazy learning algorithm: no explicit training, computation deferred until prediction.
- Prediction: Find the k nearest neighbors of a query point and use them to classify or predict.
- Commonly used as a baseline in classification and regression tasks.

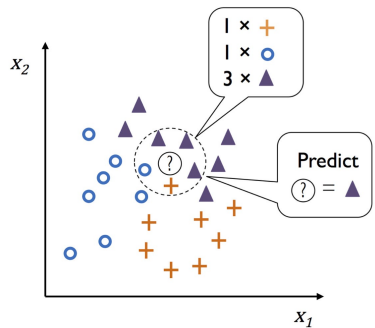


Image source [2]

k-NN Algorithm Overview

Training (there is no real training to speak of):

- Store the training examples.

Prediction:

1. Compute distances between query point and training points.
2. Identify the k nearest neighbors.
3. Aggregate the labels (e.g., majority vote for classification).

Common Distance Metrics:

- Euclidean: $d(x, y) = \sqrt{\sum_{j=1}^{m} (x_j - y_j)^2}$
- Manhattan: $d(x, y) = \sum_{j=1}^{m} |x_j - y_j|$ (useful in higher-dimensional problems)

Cosine Similarity

Definition:

$$s(x_i, x_k) = \frac{\sum_{j=1}^{D} x_i^{(j)} x_k^{(j)}}{\sqrt{\sum_{j=1}^{D} \left(x_i^{(j)}\right)^2} \sqrt{\sum_{j=1}^{D} \left(x_k^{(j)}\right)^2}}$$

- Measures the similarity of directions between two vectors.
- Value ranges from -1 to 1:
  - 1: Vectors point in the same direction.
  - 0: Vectors are orthogonal (no similarity).
  - -1: Vectors point in opposite directions.

In any case, weigh the features appropriately (for your specific problem).

Nearest Neighbor Decision Boundary (k=1)

- The decision boundary between two training points (e.g., *a* and *b*) is a straight line equidistant from both.
- Globally, the boundary forms a set of connected convex polyhedra.
- Each polyhedron encloses points closest to a particular training example.

Voronoi diagram is a geometric representation of such partitions:

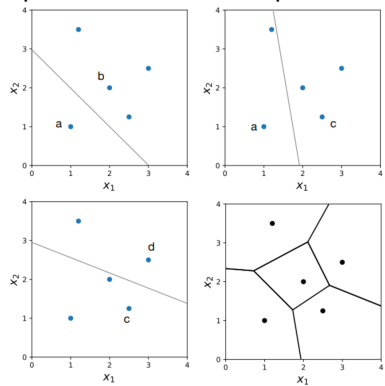
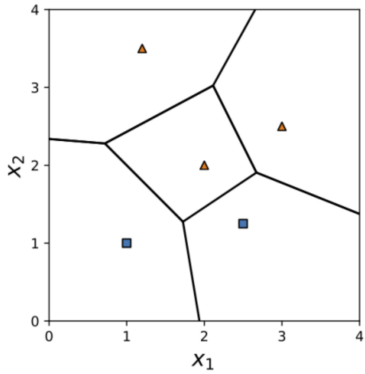


Image source [3]

Visualization of Decision Boundaries

- Partitioning regions in 2D based on proximity to training points.
- Linear segments represent boundaries between two points.
- Nodes are equidistant to three training points.



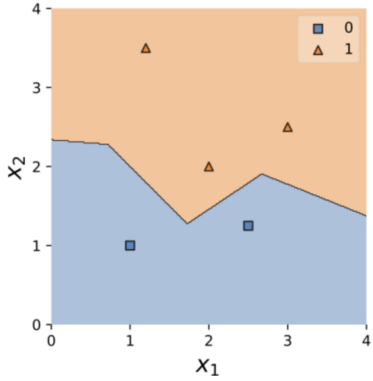


Image source [3]

Classification and Regression with a k-NN model

Classification:

- Predicts the class label based on simple or relative majority vote among the k nearest neighbors
- In case of a tie: choose a default class, choose at random, or **preferably** consider the distance
- Handles binary and multi-class problems

Regression:

- Predicts the continuous target by averaging the values of k nearest neighbors.

Weighting by Distance

$$h(x^{[t]}) = \frac{\sum_{i=1}^{k} w^{[i]} f(x^{[i]})}{\sum_{i=1}^{k} w^{[i]}}$$

with the weights $w^{[i]}$:

$$w^{[i]} = \frac{1}{d(x^{[i]}, x^{[t]})^2}$$

$h(x) = f(x)$ can be used for an exact match or include adding a small constant to the denominator for avoiding zero-division errors.

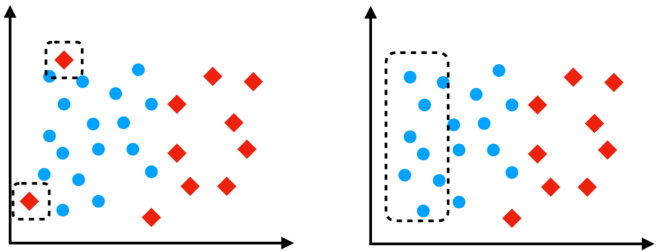
- Assigns higher weights to closer neighbors.
- Helps mitigate the influence of distant neighbors.

Summary Hyperparameters in kNN

- Choosing the value of k.
- Scaling of the feature axes.
- Choice of distance measure.
- Weighting of the distance measure.

Pruning in kNN

- **Editing:** Remove data points that do not affect the decision boundary.
  - Outliers surrounded by points from other classes can be safely removed.
  - Reduces computation by minimizing data storage and distance calculations.



- **Prototype Selection:** Replace dense clusters of points with representative prototypes.
  - Summarizes multiple points into one prototype.
  - Reduces complexity while preserving key patterns in the dataset.

Image source [3]

Advantages and Disadvantages

Advantages:

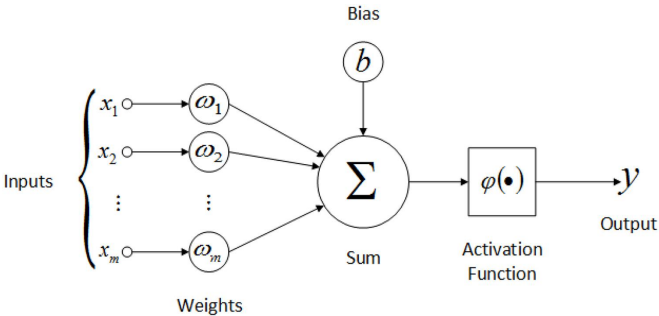
- Simple to implement and understand.
- Flexible, works for both classification and regression.

Disadvantages:

- Computationally expensive at prediction time.
- Sensitive to the choice of distance metric and the value of k.
- Poor performance in high-dimensional spaces (Curse of Dimensionality)
  - Performance degrades in high-dimensional (>10) spaces.
  - Distances in higher dimensions become more similar (all are far away).
  - Possible solution: Feature selection and dimensionality reduction (e.g., PCA).

Chapter 3: Supervised Learning: Classification and Regression
F: Neural Networks

What is an Artificial Neuron?



Output y:

$$y = \phi\left(\sum_{i=1}^{m} x_i w_i + b\right)$$

with m number of inputs, w_i "weights", b "bias", and ϕ "activation function".

Image source [4]

Building a Network: Layers

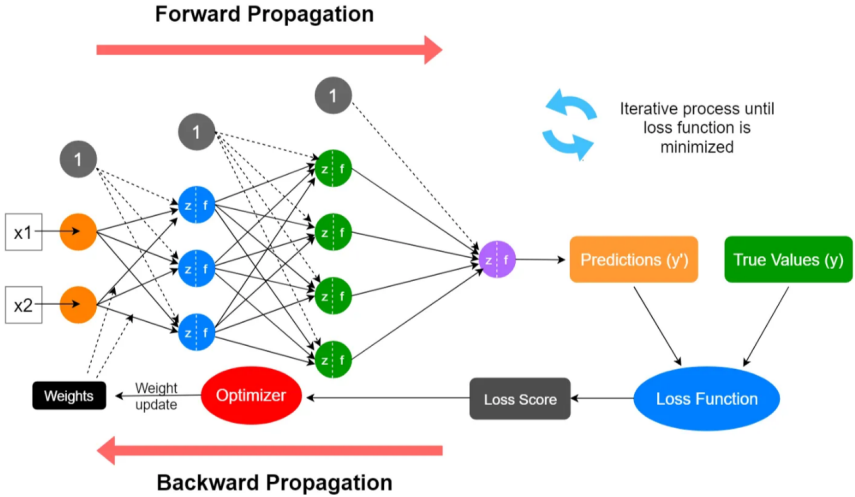


Image source [5]

Activation Functions in Classification

- **Sigmoid Function** (Binary Classification):

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

  Maps any real-valued number into the range (0, 1).

- **Softmax Function** (Multi-class Classification):

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

  Produces a probability distribution over K classes.

- **Properties**:
  - Outputs sum to 1.
  - Each output is between 0 and 1.

Loss Functions in Classification

Binary Cross-Entropy Loss:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

Used for binary classification tasks.

Categorical Cross-Entropy Loss:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_{i,k} \log(\hat{y}_{i,k})$$

Applicable to multi-class classification

- Measures the dissimilarity between true labels and predicted probabilities.
- Encourages the model to assign high probabilities to the correct classes.

Other Activation Functions

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$

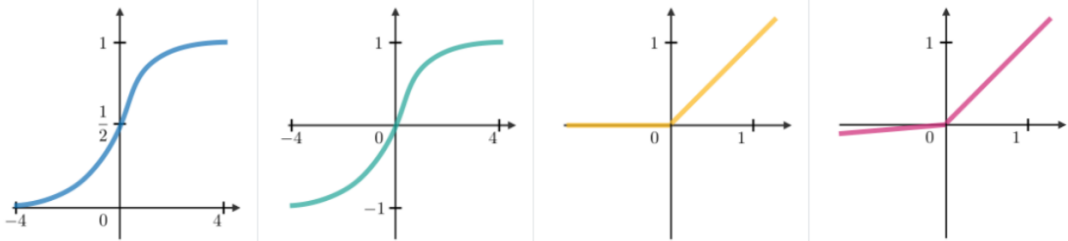


Image source [6]

Backpropagation

Gradient descent applied to weights and biases:

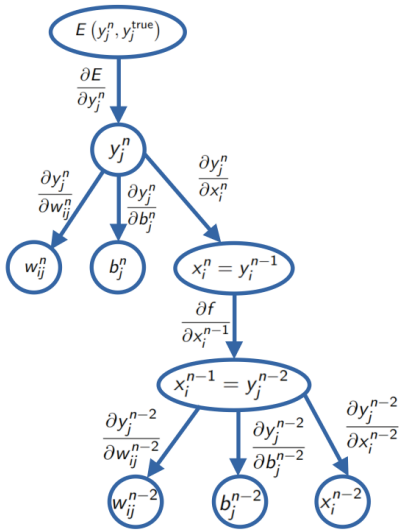
$$w_{ij} \leftarrow -\alpha \frac{\partial L}{\partial w_{ij}}$$

$$b_j \leftarrow -\alpha \frac{\partial L}{\partial b_j}$$

with L, the loss function, and α the learning rate.

We need for each layer: $\frac{\partial L}{\partial w_{ij}}$, $\frac{\partial L}{\partial b_j}$, and $\frac{\partial L}{\partial x_i}$

Chain rule



Loss function

Final dense layer n

$$y_j^n = \sum_i w_{ij}^n x_i^n + b_j^n$$

Activation function, layer $n-1$

$$y_i^{n-1} = f(x_i^{n-1})$$

2^{nd} to last dense layer $n-2$

$$y_j^{n-2} = \sum_i w_{ij}^{n-2} x_i^{n-2} + b_j^{n-2}$$

Individual terms

$$y_j = \sum_i x_i w_{ij} + b_j$$

Concerning $\frac{\partial L}{\partial w_{ij}}$:

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial w_{ij}} = \frac{\partial L}{\partial y_j}x_i$$

Concerning $\frac{\partial L}{\partial b_j}$:

$$\frac{\partial L}{\partial b_j} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial b_j} = \frac{\partial L}{\partial y_j}$$

Concerning $\frac{\partial L}{\partial x_i}$:

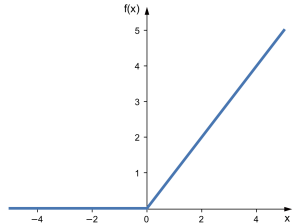
$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial x_i} = \sum_j \frac{\partial L}{\partial y_j}w_{ij}$$

Activation function (another type of layer)

$$y_i = f(x_i)$$

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial x_i} = \frac{\partial L}{\partial y_j} f'(x)$$

for example ReLU:



$$f(x) = \max(x, 0)$$

$$\frac{\partial f(x)}{\partial x} = (x > 0)$$

Image source [7]

Programming design aspects

$$x_i \rightarrow \boxed{\text{layer}} \rightarrow y_j$$

$$\frac{\partial L}{\partial x_i} \leftarrow \boxed{\text{layer}} \leftarrow \frac{\partial L}{\partial y_j}$$

A python class can be used to implement each layer, in our case:

- "fully" connected or "dense"
- layer with the activation function

Programming design aspects

The class may implement the following functions:

- to initialize its parameters ("weights" and "biases"), if any.
- to carry out forward propagation $y(x)$.
- to carry out backpropagation, calculating
  - $\frac{\partial L}{\partial x_i}$ from $\frac{\partial L}{\partial y_j}$.
  - the derivative of L with respect to its parameters. The function needs the layer inputs, so it is useful to save them during the forward pass.
  - the parameter updates according to gradient descent
- to save its weights to a file
- to read weights from a file

Weight Initialization

Importance: avoid vanishing/exploding gradients.

Methods:

- **Zero Initialization:** All weights set to zero $\rightarrow$ not a good idea, gradient is the same for each
- **Random Initialization:** Small positive random numbers $\rightarrow$ not a good idea, on average each layer will add to the signal
- **Xavier/Glorot Initialization:** For sigmoid, tanh, softmax. Zero mean, variance $\frac{2}{n_{in}+n_{out}}$.
- **He Initialization:** For ReLU. Zero mean, variance $\frac{2}{n_{in}}$.

Regularization

- **L1 regularization**: $L' = L + \lambda \sum_i |w_i|$ Penalizes absolute values of weights. Encourages sparsity.
- **L2 regularization**: $L' = L + \lambda \sum_i w_i^2$ Penalizes squared values of weights. Prevents overfitting.
- **Dropout**: Randomly drop units during training to prevent reliance on any single neuron.

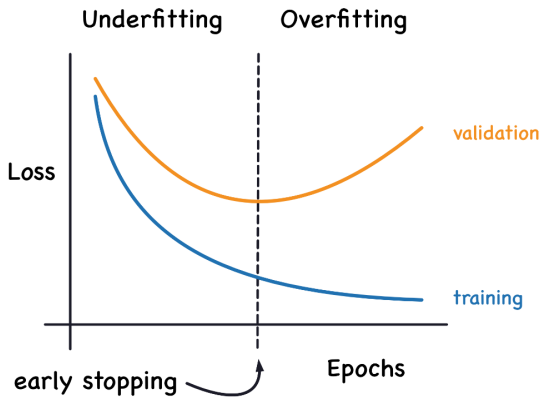
Hyperparameter Tuning

Hyperparameters: everything that's not parameters (i.e. weights and biases)

- **Number of Layers and Units:** Depth and width of the neural network
- **Learning Rate:** Determines the step size during optimization
- **Batch Size:** Number of samples used in each iteration
- **Activation Functions:** Non-linear transformations applied to layers
- **Dropout Rate:** Fraction of neurons to ignore during training
- **Weight Initialization:** Strategy to set initial values of weights
- **Optimizers:** Algorithm to adjust model parameters (e.g., SGD, Adam)

Tuning hyperparameters is as much an art as it is a science. Iterative testing and validation are key.
Use also **ensembling**: Combine multiple models with different hyperparameters?

Optimizing the training process



- Learning rate scheduling: adjust the learning rate during training
- Early stopping: if the model's performance stops improving on the validation dataset for a predetermined number of epochs, training is halted
- Mini batches: updates weights based on a subset of data points

Image source [8]