Machine Learning (CE 40717)

Fall 2025

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

October 26, 2025

How Machines Learn to Decide

- Classification is a **decision-making process**. A model learns from past examples how to assign new inputs to categories.

- Examples from daily life:
  - A doctor examines an X-ray and determines if it shows pneumonia.
  - A bank reviews a transaction and flags it as legitimate or fraudulent.
  - Your phone sorts messages into *Primary*, *Promotions*, or *Spam*.

- In all cases:
  - The input is described by measurable features.
  - The output is a predicted class.
  - The model learns patterns or rules to separate classes.

- Key idea: Classification is about learning general patterns, not memorizing examples.

How Classification Works

- We start with a training dataset:

$$D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$$

  where $\mathbf{x}^{(i)}$ is a feature vector and $y^{(i)}$ is the class label.

- The model learns a function that maps features to classes:

$$f : \mathbb{R}^n \to \{1, 2, \ldots, K\}$$

- During training, the algorithm identifies patterns that separate one class from another.
- After training, the model can predict the class of new, unseen inputs.

Case Study: Predicting Diabetes (Concept)

- Goal: Predict whether a patient has diabetes based on medical measurements.
- Dataset: Pima Indians Diabetes Dataset.
- Input features: Glucose level, blood pressure, BMI, age, etc.
- Output label:

$$y = \begin{cases} 1, & \text{Diabetic (Positive)} \\ 0, & \text{Non-diabetic (Negative)} \end{cases}$$

- Importance: Early prediction supports prevention and treatment.

Case Study: Diabetes Dataset Table

#	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	Pedigree	Age	BMI	Label
1	6	148	72	35	0	0.627	50	33.6	Positive
2	1	85	66	29	0	0.351	31	26.6	Negative
3	0	137	40	35	168	2.288	33	43.1	Positive
4	1	89	66	23	94	0.167	21	28.1	Negative
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Classification vs. Regression: Comparison Table

- Both are *supervised learning* tasks — learn a mapping from inputs to outputs.
- **Regression:** models a *continuous relationship* — how inputs influence a numeric outcome.
- **Classification:** models *decision boundaries or class probabilities* — how inputs determine category membership.

Aspect	**Regression**	**Classification**
Output Type	Continuous value ($\mathbb{R}$)	Discrete class label
Examples	House price, temperature	Spam detection, sentiment analysis
Evaluation Metrics	MSE, MAE	Accuracy, Precision, Recall

Discriminant Functions in Machine Learning

- **Conceptual Overview:**
  A discriminant function constitutes a mapping from the feature space to a real-valued score that quantifies the likelihood or confidence of a sample belonging to a specific class.

- **Formal Definition:**
  Let $\mathbf{x} \in \mathbb{R}^d$ denote a feature vector. A discriminant function is a function $g(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$ such that larger values of $g(\mathbf{x})$ correspond to stronger evidence for a particular class.

- **Objective:**
  Design $g(\mathbf{x})$ to maximize correct classification over a given dataset.

Classification Using Discriminant Functions

- **Binary Classification:**
  - Consider two discriminant functions $g_1(\mathbf{x})$ and $g_2(\mathbf{x})$ corresponding to classes C_1 and C_2, respectively.
  - The predicted class $\hat{y}$ is determined by the criterion:

$$\hat{y} = \begin{cases} C_1 & \text{if } g_1(\mathbf{x}) > g_2(\mathbf{x}) \\ C_2 & \text{otherwise.} \end{cases}$$

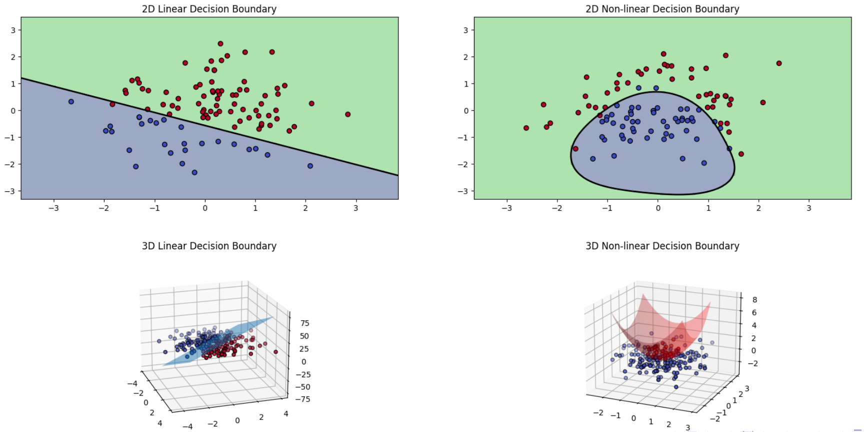
- **Multi-Class Classification:**
  - For k classes, compute $g_i(\mathbf{x})$ for each class C_i, $i = 1, \ldots, k$.
  - Assign $\mathbf{x}$ to the class corresponding to the maximal discriminant value:

$$\hat{y} = \arg\max_i g_i(\mathbf{x})$$

- **Interpretation:** The discriminant function serves as a quantitative measure of class membership confidence.

Decision Boundary

- **Definition**: A dividing hyperplane that separates different classes in a feature space, also known as "Decision Surface".



Linear Classifiers

- **Definition:**
  Linear classifiers assign class labels using a decision function that is linear in the feature vector $\mathbf{x} \in \mathbb{R}^d$, or linear in a set of transformed features of $\mathbf{x}$.

- **Linearly separable data:**
  Data points that can be perfectly separated by a linear decision boundary.

- **General form:**

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0,$$

  where $\mathbf{w}$ defines the orientation of the decision surface and w_0 determines its position.

Two-Category Classification

- **Linear discriminant:**

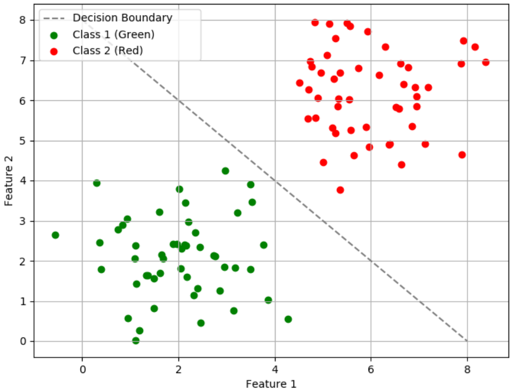
$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$$

- $\mathbf{x} = [x_1, \ldots, x_d]^T$, $\mathbf{w} = [w_1, \ldots, w_d]^T$, w_0 : bias

- **Decision rule:**

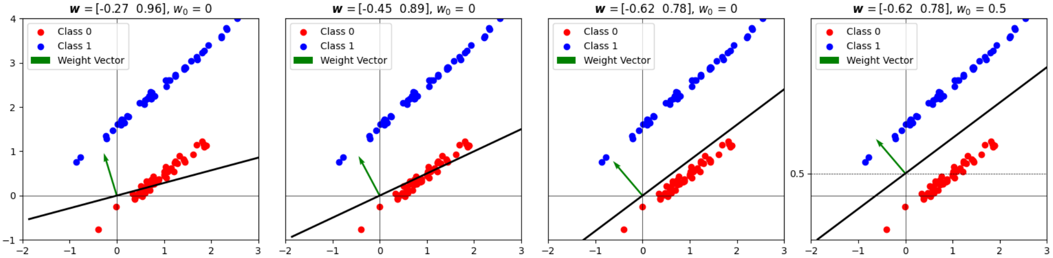
$$\hat{y} = \begin{cases} C_1, & \text{if } g(\mathbf{x}) \geq 0 \\ C_2, & \text{otherwise} \end{cases}$$

- **Decision surface:** $\mathbf{w}^T\mathbf{x} + w_0 = 0$



Geometric Properties of Linear Decision Boundaries

- The decision boundary is a $(d-1)$-dimensional hyperplane in $\mathbb{R}^d$.
- **Properties:**
  - Orientation is determined by the normal vector $\mathbf{w}/\|\mathbf{w}\|$.
  - Bias w_0 controls the displacement along the normal vector.
- Points on opposite sides of the hyperplane are assigned to different classes.



Nonlinear Decision Boundaries

- **Problem:**
  Many datasets cannot be separated by a linear hyperplane.
- **Feature Transformation:**
  Map input vector **x** to a higher-dimensional space $\phi(\mathbf{x})$.
- **Resulting Decision Boundary:**
  Linear in the transformed space, but nonlinear in the original feature space.

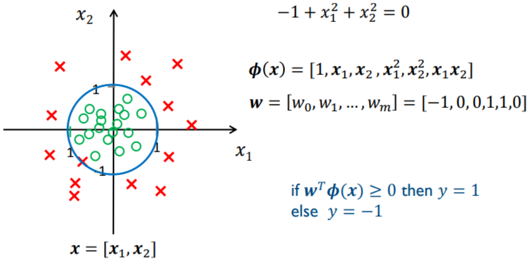


Figure adapted from M. Soleymani, Machine Learning course, Sharif University of Technology.

From Biology to Computation

- **Biological Inspiration:**
  - The human brain consists of interconnected cells called **neurons**, each transmitting signals to others through electrical impulses.
  - Each neuron receives inputs, processes them, and produces an output signal.
  - This biological structure inspired the design of artificial computational models known as **perceptrons**.

Neuron Excitation

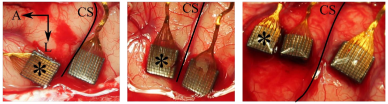


Figure adapted from Nason et al., Nature Biomedical Engineering, 2020.

From Neuron to Perceptron

- **Abstracting a Neuron:**
  - Biological neurons combine multiple inputs, each with a strength (synapse).
  - Similarly, a perceptron multiplies each input by a **weight**, sums them, adds a **bias**, and applies an **activation function**.
  - The activation function determines whether the perceptron "fires" (outputs 1) or stays "inactive" (outputs 0).

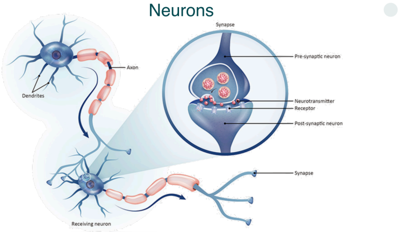


Figure adapted from www.genetex.com

Components of a Perceptron

- **Inputs** $(x_1, x_2, \ldots, x_n)$ – the feature values.
- **Weights** $(w_1, w_2, \ldots, w_n)$ – importance of each feature.
- **Bias** (b) – adjusts the threshold for activation.
- **Weighted Sum:**

$$z = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^T \mathbf{x} + b$$

- **Activation Function** (f) – transforms z into output:

$$y = f(z)$$

Activation Functions — Step Sigmoid

- **Step Function:**

$$f(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

  Classic perceptron; non-differentiable.

- **Sigmoid Function:**

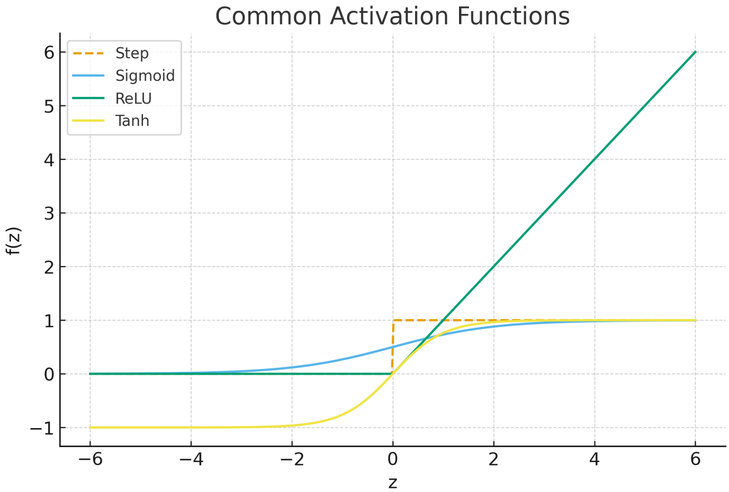
$$f(z) = \frac{1}{1 + e^{-z}}$$

  Smooth output (0–1); differentiable; may saturate for large $|z|$.

Activation Functions — ReLU Variants

- **ReLU:**
$$f(z) = \max(0, z)$$
Passes positives, zeros negatives; fast stable training.
- **Leaky ReLU:**
$$f(z) = \max(0.01z, z)$$
Allows small gradient for negative inputs.
- **Tanh:**
$$f(z) = \tanh(z)$$
Output in $[-1, 1]$; smooth and zero-centered.

Activation Functions



Mathematical Model of a Perceptron

- **Computation Rule:**

$$y = f(\mathbf{w}^T \mathbf{x} + b)$$

- **Explanation:**
  - $\mathbf{x}$: input vector of features.
  - $\mathbf{w}$: weight vector determining importance.
  - b: bias, controlling threshold.
  - f: activation function.

- The perceptron outputs 1 if the weighted sum exceeds the threshold, otherwise 0.

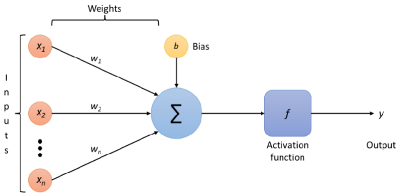


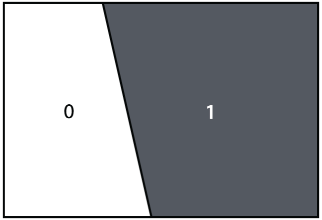
Figure Adapted from Sánchez et al. (2022)

Linear Decision Boundary

- The perceptron defines a **linear boundary**:

$$\mathbf{w}^T\mathbf{x} + b = 0$$

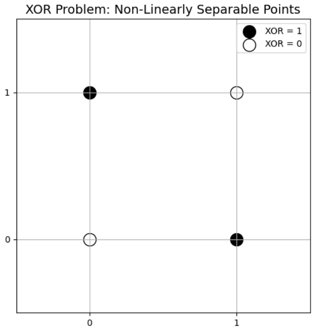
- All points on one side of this line (or hyperplane) belong to class C_1; others to class C_2.
- Example of linearly separable problems:
  - Logical AND
  - Logical OR



Example: linear separation in 2D space.

Limitation of a Single Perceptron

- A single perceptron can only solve **linearly separable** problems.
- It fails on tasks like the **XOR problem**, where no straight line can divide the two classes.
- To handle more complex patterns, we need to move beyond simple linear models.



XOR problem — not linearly separable.

Feature Engineering: Manually Creating New Views of Data

- **Feature engineering** means designing or transforming input variables so that a model can better capture patterns in the data.
- When data isn't linearly separable, we can transform it into a higher-dimensional space.
- Example:

$$(x_1, x_2) \to (x_1, x_2, x_1 x_2)$$

- The new feature $x_1 x_2$ helps separate XOR data using a simple linear classifier.
- In essence, we make the data easier for the model to understand.

Multi-Layer Perceptrons

- **Automatic feature learning:** MLPs extract useful representations from data without manual engineering.
- Layers are stacked to capture **nonlinear relationships**:

  Input Layer → Hidden Layer(s) → Output Layer

- Hidden layers use **nonlinear activations** (e.g., ReLU, Sigmoid) to form flexible decision boundaries.
- Each layer builds on the previous, progressively learning more abstract features.

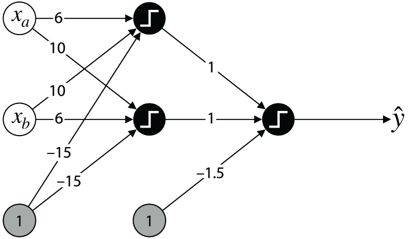


Figure adapted from mriquestions.com.

Key Takeaways

- Perceptrons perform linear classification — simple but limited.
- **Feature engineering** helps models handle nonlinear patterns by transforming inputs.
- Manual feature design is powerful but often impractical for high-dimensional data.
- **Multi-Layer Perceptrons** learn these transformations automatically through hidden layers and nonlinear activations.
- Deep networks are, in many ways, models that learn to do their own feature engineering.

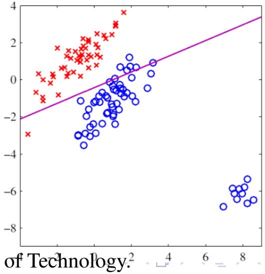
Cost Functions

- **Understanding the Goal**
  - In the perceptron, we use $\mathbf{w}^T\mathbf{x}$ to make predictions.
  - Goal is to find the optimal $\mathbf{w}$ so that the predicted labels match the true labels as much as possible.
  - To achieve this, we define a cost function, which measures the **difference** between **predicted** and **actual** labels.
  - Finding discriminant functions $(\mathbf{w}^T, w_0)$ is framed as minimizing a cost function.
    - Based on training set $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, a cost function $J(\mathbf{w})$ is defined.
    - Problem converts to finding optimal $\hat{g}(\mathbf{x}) = g(\mathbf{x}; \hat{\mathbf{w}})$ where

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w})$$

Sum of Squared Error Cost Function

- **Sum of Squared Error (SSE) Cost Function**
  - **Formula**: $J(\mathbf{w}) = \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2, \quad \hat{y}^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + w_0$
  - SSE minimizes the magnitude of the error, which is ideal for regression but irrelevant for classification.
  - If the model predicts close to the true class but not exactly 0 or 1, SSE still shows positive error, even for correct predictions.
- SSE is also prone to overfitting noisy data, as small variations can cause significant changes in the cost.



Figures adapted from slides of M. Soleymani, Machine Learning course, Sharif University of Technology.

An Alternative for SSE Cost Function

- **Number of Misclassifications**
  - **Definition:** Measures how many samples are misclassified by the model.
  - **Formula:**

$$J(\mathbf{w}) = \sum_{i=1}^{n} \left(\frac{y^{(i)} - \text{sign}(\hat{y}^{(i)})}{2} \right)^2, \quad \hat{y}^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + w_0, \quad y^{(i)} \in \{-1, +1\}$$

    where the **sign function** is defined as:

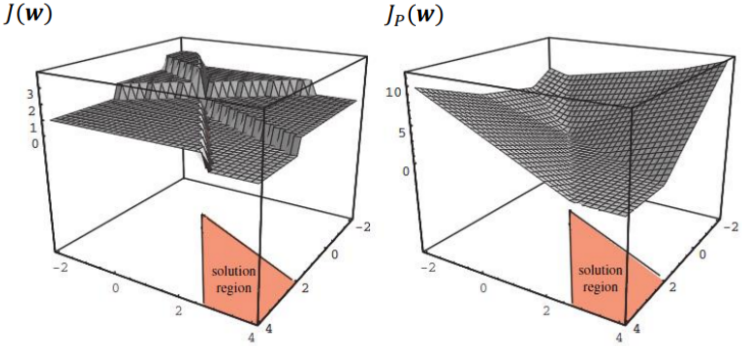
$$\text{sign}(z) = \begin{cases} +1 & \text{if } z \geq 0, \\ -1 & \text{if } z < 0. \end{cases}$$

  - **Limitations:**
    - **Piecewise Constant:** The cost function is non-differentiable, so optimization techniques (like gradient descent) cannot be directly applied.

Figure adapted from Machine Learning and Pattern Recognition, Bishop

Perceptron Algorithm

- **The Perceptron Algorithm**
  - **Purpose**: A simple algorithm for binary classification, separating two classes with a linear boundary.



Figures adapted from Machine Learning and Pattern Recognition, Bishop

Perceptron Criterion

- **Cost Function**: The perceptron criterion focuses on misclassified points:

$$J_p(\mathbf{w}) = -\sum_{i \in M} y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)}, \quad y^{(i)} \in \{-1, +1\}$$

  where M is the set of misclassified points.

- **Goal**: Minimize the loss by correctly classifying all points.

Batch Perceptron

- **Batch Perceptron**: Updates the weight vector using all misclassified points in each iteration.

- **Gradient Descent**: Adjusting weights in the direction that reduces the loss:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} J_p(\mathbf{w})$$

$$\nabla_{\mathbf{w}} J_p(\mathbf{w}) = -\sum_{i \in M} y_i \mathbf{x}_i$$

  - Batch Perceptron converges in finite number of steps for linearly separable data.

Single-sample Perceptron

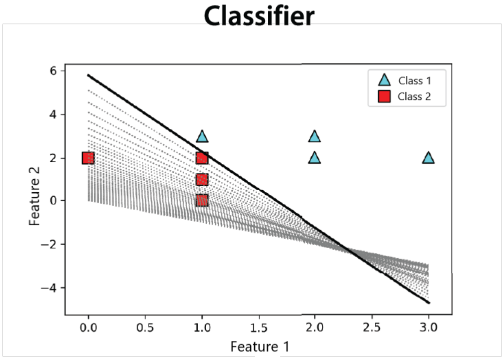
- **Single Sample Perceptron**: Updates the weight vector after each individual point.
- **Stochastic Gradient Descent (SGD) Update Rule:**
  - Using only one misclassified sample at a time:

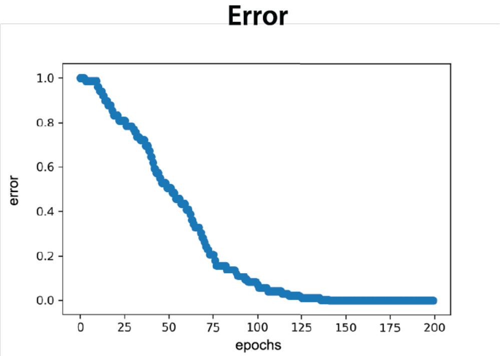
$$\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$$

  - Lower computational cost per iteration, faster convergence.
  - If training data are linearly separable, the single-sample perceptron is also guaranteed to find a solution in a finite number of steps.

Example

- Perceptron changes **w** in a direction that corrects error.





Figures adapted from Grokking Machine Learning, L. G. Serrano.

Convergence of the Perceptron — Theorem

Theorem: For linearly separable data with margin $\gamma > 0$ and $\|x_i\| \leq R$, the Perceptron algorithm makes at most $M \leq \frac{R^2}{\gamma^2}$ updates.

Notation:

- Dataset: $D = \{(x_i, y_i)\}_{i=1}^n$, with $x_i \in \mathbb{R}^d$, $y_i \in \{+1, -1\}$.
- Weight vector at step t: w_t, starting from $w_0 = 0$.
- Update rule (on misclassified sample): $w_{t+1} = w_t + y_t x_t$, where (x_t, y_t) is the misclassified sample at step t.
- Assume there exists w^* with $\|w^*\| = 1$ that correctly classifies all samples.
- Each input is bounded: $\|x_i\| \leq R$ (after scaling, $R = 1$).
- Margin:

$$\gamma = \min_{(x_i, y_i) \in D} y_i (x_i^\top w^*) > 0.$$

Convergence of the Perceptron — Proof (1)

We analyze the Perceptron as a gradient-descent-like algorithm. Each update occurs when a sample is misclassified.

1. Let (x_t, y_t) be the misclassified sample at step t. The update is

$$w_{t+1} = w_t + y_t x_t.$$

2. By induction, after M updates:

$$w_M = \sum_{t=1}^{M} y_t x_t.$$

3. Inner product with w^*:

$$w_M \cdot w^* = \sum_{t=1}^{M} y_t (x_t \cdot w^*) \geq M\gamma.$$

Convergence of the Perceptron — Proof (2)

4. Norm growth:

$$\|w_M\|^2 = \|w_{M-1}\|^2 + 2y_M(w_{M-1} \cdot x_M) + \|x_M\|^2 \leq \|w_{M-1}\|^2 + R^2$$

   because $y_M(w_{M-1} \cdot x_M) \leq 0$ for a misclassified sample.

5. By induction:

$$\|w_M\| \leq R\sqrt{M}.$$

6. Using Cauchy–Schwarz:

$$M\gamma \leq w_M \cdot w^* \leq \|w_M\|\|w^*\| \leq R\sqrt{M} \implies M \leq \frac{R^2}{\gamma^2}.$$

Source: Novikoff (1962), *On Convergence Proofs for Perceptrons*; M. Collins, *Convergence Proof for the Perceptron Algorithm*, Columbia University.

Convergence of Perceptron Cont.

- **Non-Linearly Separable Data**: When no linear decision boundary can perfectly separate the classes, the Perceptron fails to converge.
  - If data is not linearly separable, there will always be some points that the model fails to classify.
  - As a result, the algorithm keeps adjusting the weights to fix the misclassified points, causing it to never converge.
  - For the data that are not linearly separable due to noise, **Pocket Algorithm** keeps in its pocket the best **w** encountered up to now.

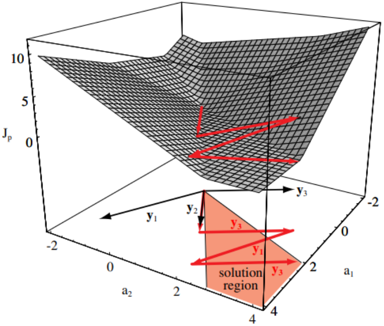


Figure adapted from Machine Learning and Pattern Recognition, Bishop

Pocket Algorithm

Algorithm 1 Pocket Algorithm

Initialize w
for $t = 1$ to T **do**
  $i \leftarrow t \bmod N$
  if $\mathbf{x}^{(i)}$ is misclassified **then**
    $\mathbf{w}^{new} = \mathbf{w} + \eta \mathbf{x}^{(i)} y^{(i)}$
    if $E_{train}(\mathbf{w}^{new}) < E_{train}(\mathbf{w})$ **then** $\triangleright$ $E_{train}(\mathbf{w}) = J_p(\mathbf{w})$
      $\mathbf{w} = \mathbf{w}^{new}$
    end if
  end if
end for

Multi-Category Classification

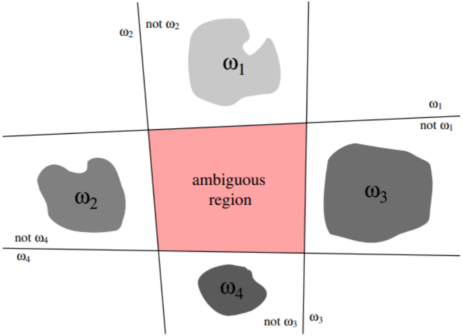
- **Solutions to multi-category classification problem**:
  - Extend the learning algorithm to support multi-class.
    - First, a function g_i for every class C_i is found.
    - Second, $\mathbf{x}$ is assigned to C_i if $g_i(\mathbf{x}) > g_j(\mathbf{x}) \quad \forall i \neq j$

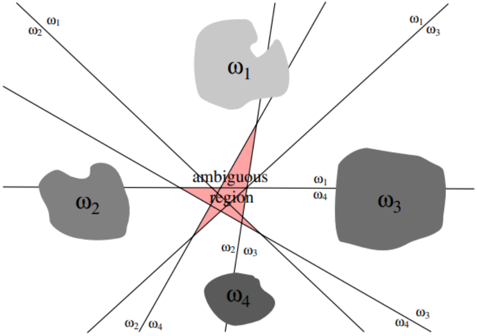
$$\hat{y} = \underset{i=1,\ldots,c}{\text{argmax}}\, g_i(\mathbf{x})$$

  - Convert to a set of two-categorical problems.
    - Methods like **One-vs-Rest** or **One-vs-One**, where each classifier distinguishes between either **one class and the rest**, or **between pairs of classes**.

Multi-Category Classification: Ambiguity

- One-vs-One and One-vs-Rest conversion can lead to regions in which the classification is **undefined**.





Figures adapted from Machine Learning and Pattern Recognition, Bishop

Multi-Category Classification: Linear Machines

- **Linear Machines**: Alternative to One-vs-Rest and One-vs-One methods; Each class is represented by its own discriminant function.

- **Decision Rule:**

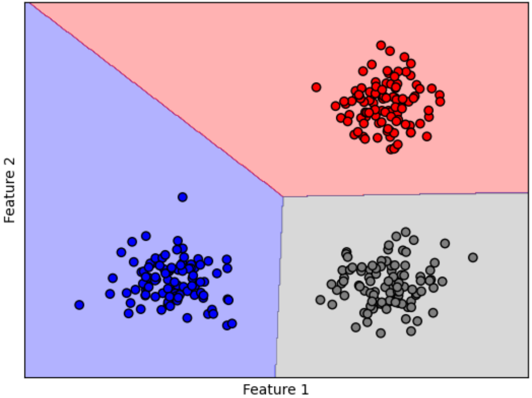
$$\hat{y} = \underset{i=1,\ldots,c}{\operatorname{argmax}}\, g_i(\mathbf{x})$$

  The predicted class is the one with the highest discriminant function value.

- **Decision Boundary**: $g_i(\mathbf{x}) = g_j(\mathbf{x})$

$$(\mathbf{w}_i - \mathbf{w}_j)^T \mathbf{x} + (w_{0i} - w_{0j}) = 0$$

Linear Machines Cont.



- The decision regions of this discriminant are **convex** and **singly connected**. Any point on the line between two points within the same region can be expressed as $\mathbf{x} = \lambda \mathbf{x}_A + (1 - \lambda)\mathbf{x}_B$ where $\mathbf{x}_A, \mathbf{x}_B \in C_k$.

Multi-Class Perceptron Algorithm

- **Weight Vectors**:
  - Maintain a weight matrix $W \in \mathbb{R}^{m \times K}$, where m is the number of features and K is the number of classes.
  - Each column w_k of the matrix corresponds to the weight vector for class k.

$$\hat{y} = \underset{i=1,\ldots,c}{\operatorname{argmax}} \mathbf{w}_i^T \mathbf{x}$$

$$J_p(\mathbf{W}) = -\sum_{i \in M} (\mathbf{w}_{y^{(i)}} - \mathbf{w}_{\hat{y}^{(i)}})^T \mathbf{x}^{(i)}$$

where M is the set of misclassified points.

Multi-Class Perceptron Algorithm

Algorithm 2 Multi-class perceptron

1: **Initialize** $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_c], k \leftarrow 0$
2: **while** A pattern is misclassified **do**
3: $\quad k \leftarrow k + 1 \bmod N$
4: $\quad$ **if** $\mathbf{x}^{(i)}$ is misclassified **then**
5: $\quad\quad \mathbf{w}_{\hat{y}^{(i)}} = \mathbf{w}_{\hat{y}^{(i)}} - \eta \mathbf{x}^{(i)}$
6: $\quad\quad \mathbf{w}_{y^{(i)}} = \mathbf{w}_{y^{(i)}} + \eta \mathbf{x}^{(i)}$
7: $\quad$ **end if**
8: **end while**

Contributions

- **This slide has been prepared thanks to:**
  - Erfan Jafari
  - Aida Jalali

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*.

[2] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. 2001.

[3] M. Soleymani, "Machine learning." Sharif University of Technology.

[4] S. F. S. Salehi, "Machine learning." Sharif University of Technology.

[5] Y. S. Abu-Mostafa, "Machine learning." California Institute of Technology, 2012.

[6] L. G. Serrano, *Grokking Machine Learning*. Manning Publications, 2020.

[7] J. M. Ashfaque, "Introduction to support vector machines and kernel methods." April 2019.