Machine Learning (CE 40717)

Fall 2025

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

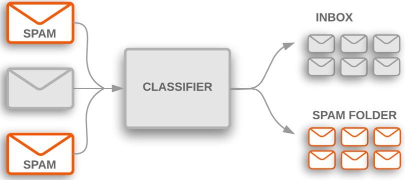
October 24, 2025

Definition

- **Given:** A training dataset

$$D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}, \quad y^{(i)} \in \{1, 2, \dots, K\}$$

- **Goal:** Learn a function that maps any new input $\mathbf{x}$ to one of K classes.
- **Examples:**
  - Email → Spam or Not Spam
  - Image → Cat, Dog, or Bird
  - Transaction → Fraudulent or Legitimate



Real-World Example of Classification

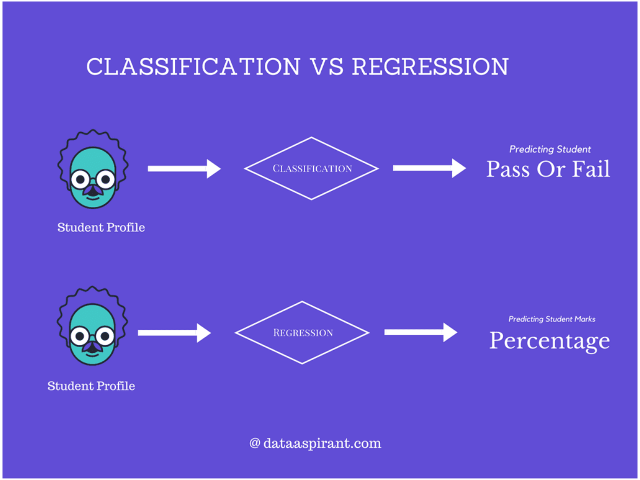
- **Pima Indians Diabetes Dataset:**
  - **Problem**: Predict whether a patient has diabetes based on medical diagnostics.
  - **Context**: Early detection of diabetes is critical for treatment and management.

	Number of times pregnant	Glucose	Blood Pressure	Skin Thickness	Insulin	Diabetes pedigree function	Age	BMI	Label
Patient 1	6	148	72	35	0	0.627	50	33.6	Positive
Patient 2	1	85	66	29	0	0.351	31	26.6	Negative
Patient 3	0	137	40	35	168	2.288	33	43.1	Positive
Patient 4	1	89	66	23	94	0.167	21	28.1	Negative
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Classification vs. Regression: Comparison Table

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

Classification vs. Regression: Figure



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:

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

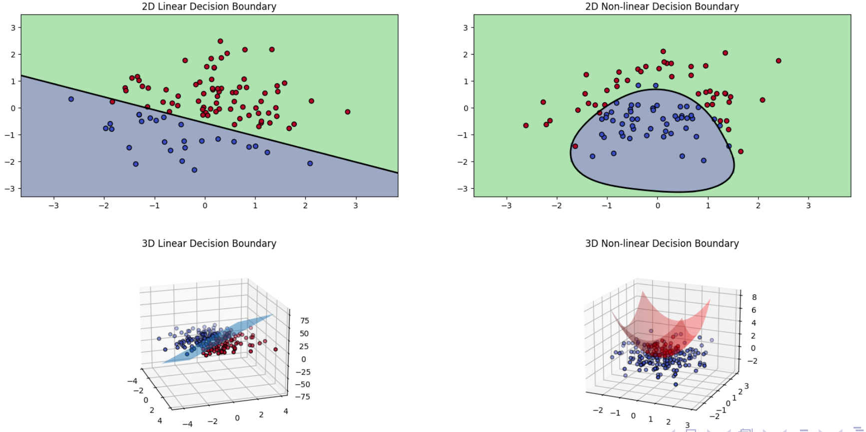
- **Multi-Class Classification:**
  - For k classes, compute $g_i(\mathbf{x})$ for each class C_i, $i = 1, \ldots, k$.
  - Assign $\mathbf{x}$ to the class corresponding to the maximal discriminant value:

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

- **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".



Two-Class Discriminant Function

- In the case of a binary classification problem, a single discriminant function suffices:

$$g : \mathbb{R}^d \to \mathbb{R}.$$

- The two class-specific discriminants can be represented as:

$$g_1(\mathbf{x}) = g(\mathbf{x}), \quad g_2(\mathbf{x}) = -g(\mathbf{x}).$$

- The associated decision rule is:

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

- The decision boundary corresponds to the set $\{\mathbf{x} \mid g(\mathbf{x}) = 0\}$.
- This formulation provides a foundation for subsequent extension to multi-class discriminant analysis.

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:**

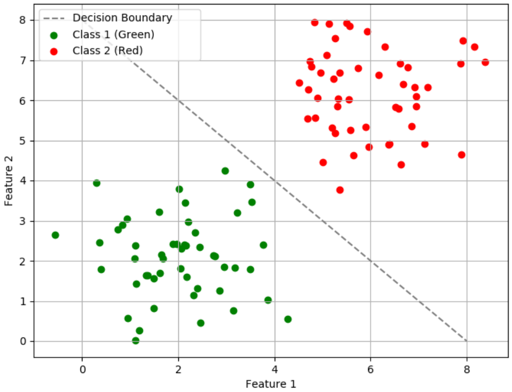
$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$$

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

- **Decision rule:**

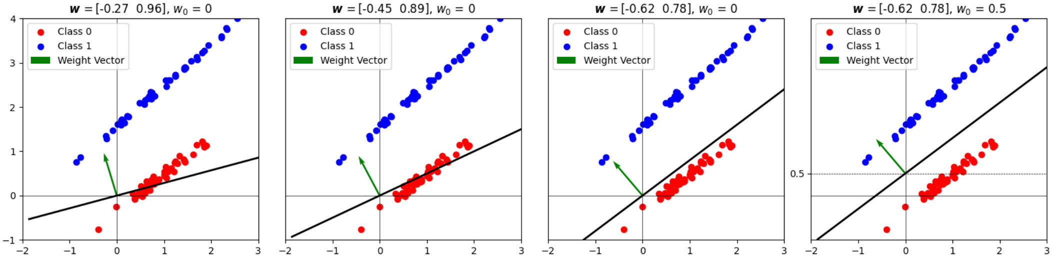
$$\hat{y} = \begin{cases} C_1, & \text{if } g(\mathbf{x}) \geq 0 \\ C_2, & \text{otherwise} \end{cases}$$

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



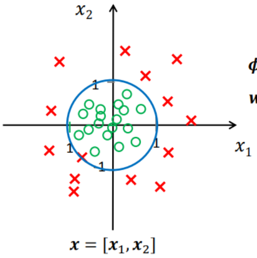
Geometric Properties of Linear Decision Boundaries

- 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.



$-1 + x_1^2 + x_2^2 = 0$

$\boldsymbol{\phi}(\boldsymbol{x}) = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2]$

$\boldsymbol{w} = [w_0, w_1, \ldots, w_m] = [-1, 0, 0, 1, 1, 0]$

if $\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \geq 0$ then $y = 1$
else $y = -1$

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

What Is Perceptron?

- **Perceptron Unit:**
  - **Basic Building Block:** The perceptron is the simplest form of an artificial neuron used for classification tasks.
  - **Linear Classifier:** It computes a linear combination of input features followed by a threshold-based decision.
  - **Binary Decision:** Outputs 1 if the weighted sum of inputs exceeds a threshold; otherwise outputs 0.
  - **Components:** Consists of input features, associated weights, a bias term, and an activation function (commonly a step or sigmoid function).

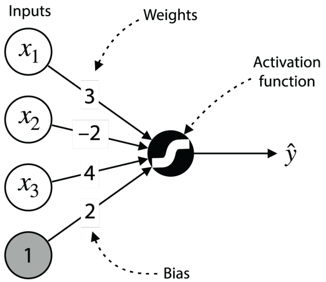


Figure adapted from *Grokking Machine Learning*, L. G. Serrano.

Inspired by Biology

- **Biological Motivation Behind Perceptron:**
  - **Inspired by Neurons**: Perceptron mimics the basic function of biological neurons in the brain.
    - Input and Output, Activation Function.

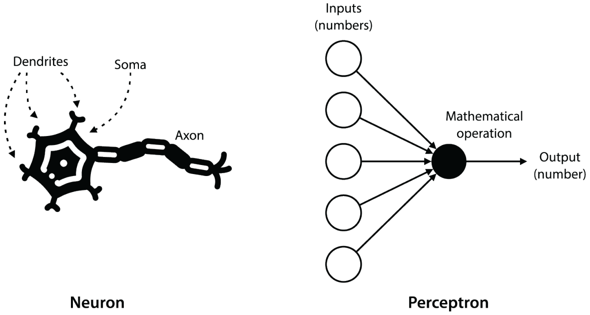


Figure adapted from Grokking Machine Learning, L. G. Serrano.

Single Neuron

- **Single Neuron as a Linear Decision Boundary**
  - **Mathematical Form**: The output of a single neuron is computed as:
  $$y = f(\mathbf{w}^T\mathbf{x} + w_0)$$
  where:
    - $\mathbf{x}$ is the input vector.
    - $\mathbf{w}$ is the weight vector.
    - w_0 is the bias term.
    - f is an activation function (e.g., step function).
  - **Linear Separation**: A neuron defines a linear decision boundary:
  $\mathbf{w}^T\mathbf{x} + w_0 = \textit{threshold}$ (0 for step, 0.5 for sigmoid)
  - **Decision Rule**: C_1 if $\mathbf{w}^T\mathbf{x} + w_0 \geq \textit{threshold}$, otherwise C_2.

$$\text{Class} = f(\mathbf{w}^T\mathbf{x} + w_0)$$

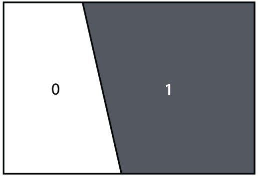
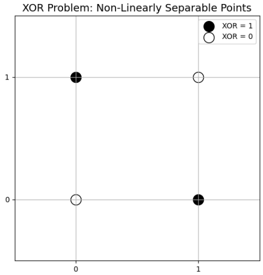


Figure adapted from Grokking Machine Learning, L. G. Serrano.

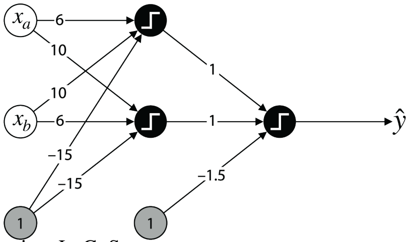
Limitations of a Single Perceptron

- **What a Single Perceptron Can and Can't Do:**
  - **Performs Linear Separations:** A perceptron can handle linearly separable problems such as:
    - AND operation
    - OR operation
  - **Fails on Non-Linear Problems:** A single perceptron fails to solve non-linear problems like XOR, as the data points cannot be separated by a straight line.



Towards Complex Decision Boundaries

- **Multi-Layer Perceptron (MLP):**
  - **Adding Layers for More Complexity**: An MLP consists of multiple layers of neurons that allow us to model more complex functions than a single neuron.
    - Each layer introduces new decision boundaries, making it possible to separate non-linear data.
  - **Two-Layer Example:**
    - Input Layer → Hidden Layer → Output Layer
    - Hidden layer introduces non-linear transformations that enable complex decision regions.



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

Refining the Decision Boundary

- **New Neurons for Better Separation**: By adding more neurons to a layer, we can further refine the decision boundary to better separate complex data.
- Each additional neuron introduces new features that help the model make more accurate decisions.

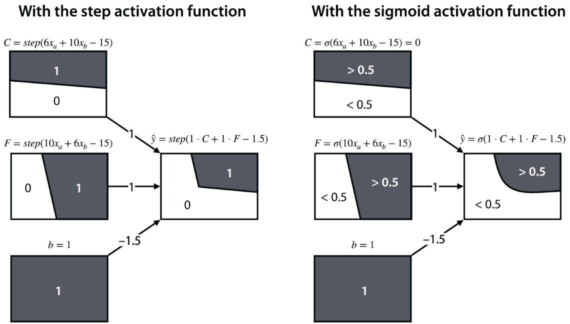


Figure adapted from Grokking Machine Learning, L. G. Serrano.

Cost Functions

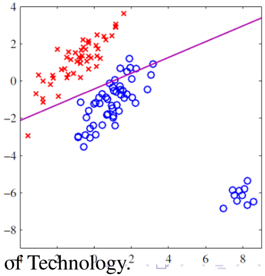
- **Understanding the Goal**
  - In the perceptron, we use $\mathbf{w}^T\mathbf{x}$ to make predictions.
  - Goal is to find the optimal $\mathbf{w}$ so that the predicted labels match the true labels as much as possible.
  - To achieve this, we define a cost function, which measures the **difference** 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**:

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

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

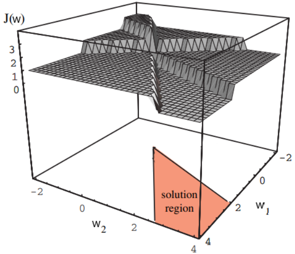
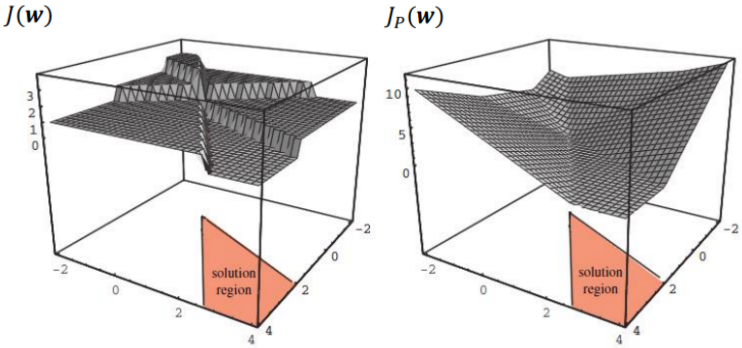


Figure adapted from Machine Learning and Pattern Recognition, Bishop

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

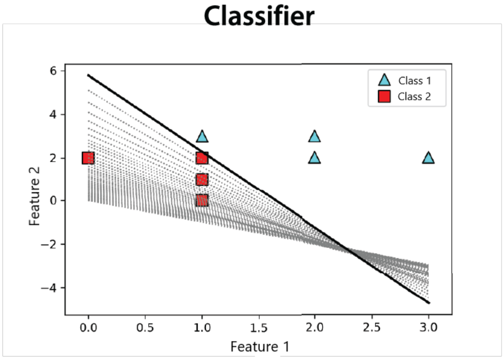
- **Single Sample Perceptron**: Updates the weight vector after each individual point.
- **Stochastic Gradient Descent (SGD) Update Rule:**
  - Using only one misclassified sample at a time:

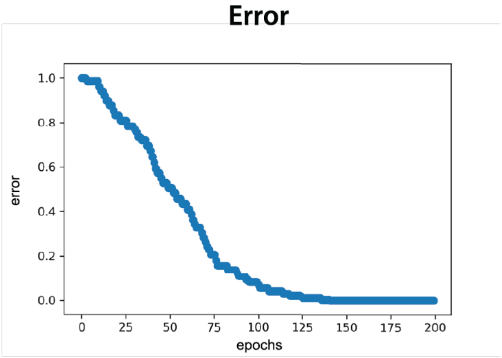
$$\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$$

  - 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 $\|x_i\| \le R$, the Perceptron algorithm makes at most $M \le \frac{R^2}{\gamma^2}$ 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 mistake): $w_{t+1} = w_t + y_i x_i$.
- Assume there exists w^* with $\|w^*\| = 1$ that correctly classifies all samples.
- Each input is bounded: $\|x_i\| \le R$ (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)

Let the algorithm make M mistakes.

1. Each mistake on (x_i, y_i) updates

$$w_{t+1} = w_t + y_i x_i.$$

2. By induction, $w_M = \sum_{t=1}^{M} y_t x_t$.
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:

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

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

5. Combine inequalities:

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

6. Therefore,

$$M \leq \frac{R^2}{\gamma^2}.$$

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.

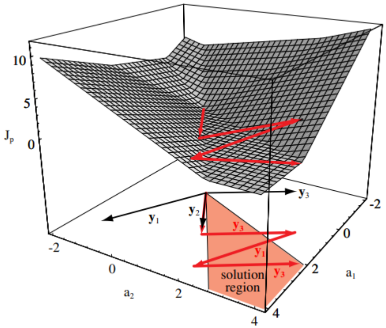


Figure adapted from Machine Learning and Pattern Recognition, Bishop

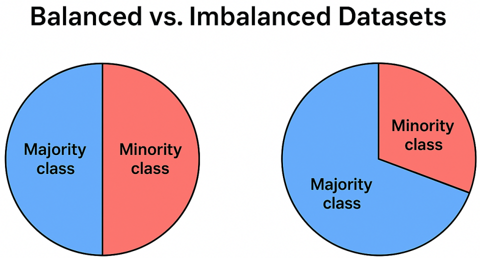
Pocket Algorithm

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

The Problem: Imbalanced Data

- Real-world datasets often contain classes with unequal representation.
- **Examples:**
  - Fraud detection: 0.1% Fraud, 99.9% Non-Fraud
  - Medical diagnosis: 2% Disease, 98% Healthy
- **Issue:** High accuracy may hide poor performance on the minority class.



Solution 1: Resampling Techniques

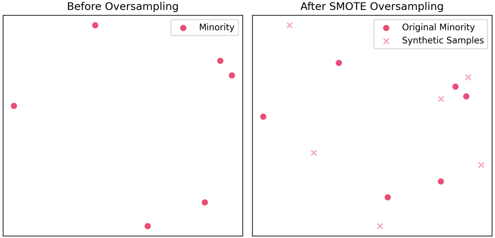
Undersampling

- Remove samples from majority class.
- + Reduces training time.
- – Risk of losing information.

Oversampling (e.g., SMOTE)

- Generate synthetic minority samples.
- + Preserves information.
- – May cause overfitting.

Majority class reduced.



New minority points added via interpolation.

Solution 2: Algorithmic Approaches

Weighted Loss Function

- Assign a higher penalty to errors on the minority class.
- Encourages the model to focus on rare but important cases.

$$J(w) = -\sum_i w_{y^{(i)}} \, y^{(i)} \log(\hat{y}^{(i)})$$

where $w_{y^{(i)}}$ is larger for the minority class.

Hybrid Approaches

Combining Sampling Strategies

- Integrates both oversampling and undersampling to achieve better class balance.

Typical Pipeline:

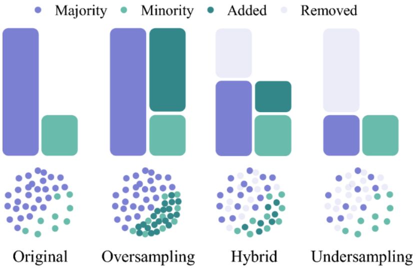
1. Apply **SMOTE** to synthetically augment minority samples.
2. Randomly undersample the majority class to reduce imbalance.
3. Train the model on the newly balanced dataset.

- Provides a practical trade-off between **bias** and **variance**.

Comparison of Methods

Summary

- **Sampling** — modifies data distribution
  (+ Simple, improves balance)
  (– Risk of over/underfitting)
- **Algorithmic** — adjusts model focus
  (+ No data change, interpretable)
  (– Needs careful weight tuning)
- **Hybrid** — combines both strategies
  (+ Balanced, strong results)
  (– More complex, slower training)



Visual summary of trade-offs

Model Selection via Cross Validation

- **Cross-Validation**
  - **Purpose**: Technique for evaluating how well a model generalizes to unseen data.
  - **How It Works**: Split data into k folds; train on $k-1$ folds and validate on the remaining fold.
  - **Repeat Process**: Repeat k times, rotating the test fold each time. Average of all scores is the final score of the model.
  - Cross-validation reduces overfitting and provides a more reliable estimation of model performance.

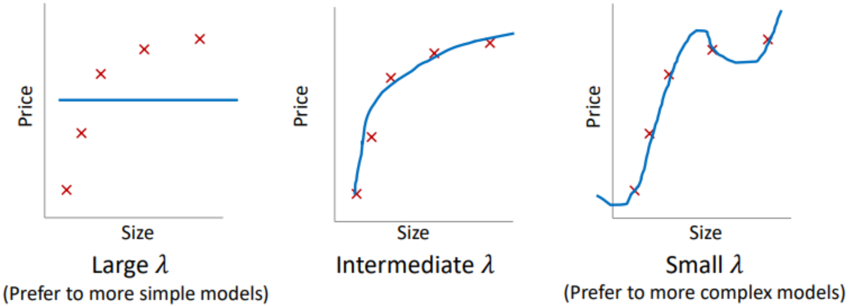
K-Fold Cross Validation

Figure adapted from Introduction to Support Vector Machines and Kernel Methods, J.M. Ashfaque.

Leave-One-Out Cross-Validation (LOOCV)

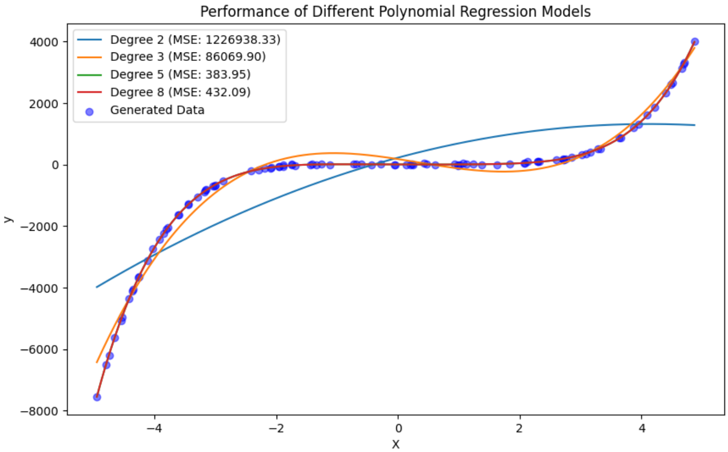
- **Leave-One-Out Cross-Validation (LOOCV)**
  - **How It Works**: Uses a single data point as the validation set ($k = 1$) and the rest as the training set. Repeat for all data points.
  - **Properties:**
    - **No Data Wastage**: Every data point is used for both training and validation.
    - **High Variance, Low Bias**.
    - **Computationally Expensive**: Requires training the model N times for N data points, making it slow for large datasets.
    - **Best for small datasets**.

Cross-Validation for Choosing Regularization Term



Figures adapted from slides of F.Salehi, Machine Learning course, Sharif University of Technology.

Cross-Validation for Choosing Model Complexity



Multi-Category Classification

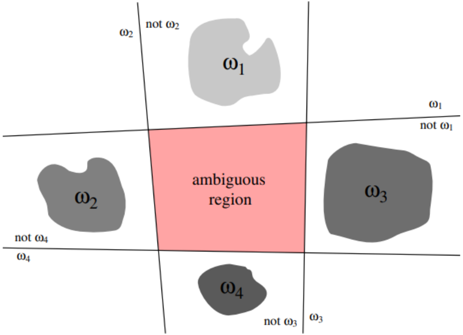
- **Solutions to multi-category classification problem**:
  - Extend the learning algorithm to support multi-class.
    - First, a function g_i for every class C_i is found.
    - Second, $\mathbf{x}$ is assigned to C_i if $g_i(\mathbf{x}) > g_j(\mathbf{x}) \quad \forall i \neq j$

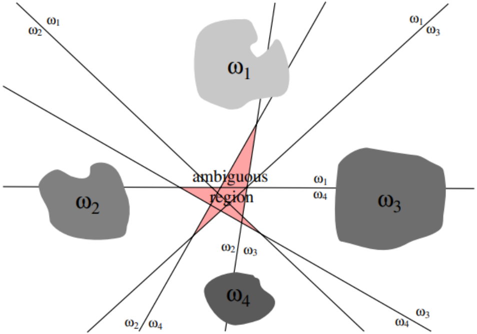
$$\hat{y} = \underset{i=1,\ldots,c}{\operatorname{argmax}}\, g_i(\mathbf{x})$$

  - 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:**

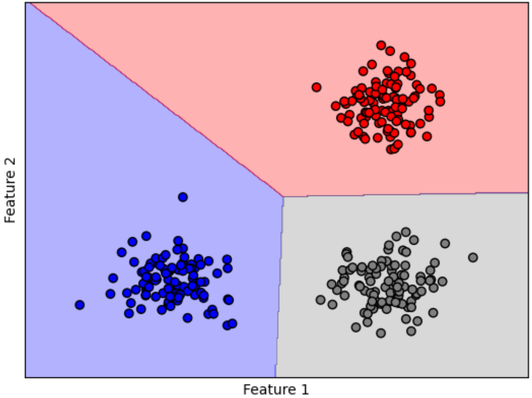
$$\hat{y} = \underset{i=1,\ldots,c}{\operatorname{argmax}}\, g_i(\mathbf{x})$$

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.

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

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

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: $\qquad \mathbf{w}_{\hat{y}^{(i)}} = \mathbf{w}_{\hat{y}^{(i)}} - \eta \mathbf{x}^{(i)}$
6: $\qquad \mathbf{w}_{y^{(i)}} = \mathbf{w}_{y^{(i)}} + \eta \mathbf{x}^{(i)}$
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.