Introduction to Machine Learning Notes

Albert Jojo

February 20, 2025

1 Introduction

This section will go over the basic machine learning algorithms. Machine Learning (ML) is a subfield of AI, which focuses on studying the methods that can be used to improve the performance of a model over time.

2 Classification Exercise

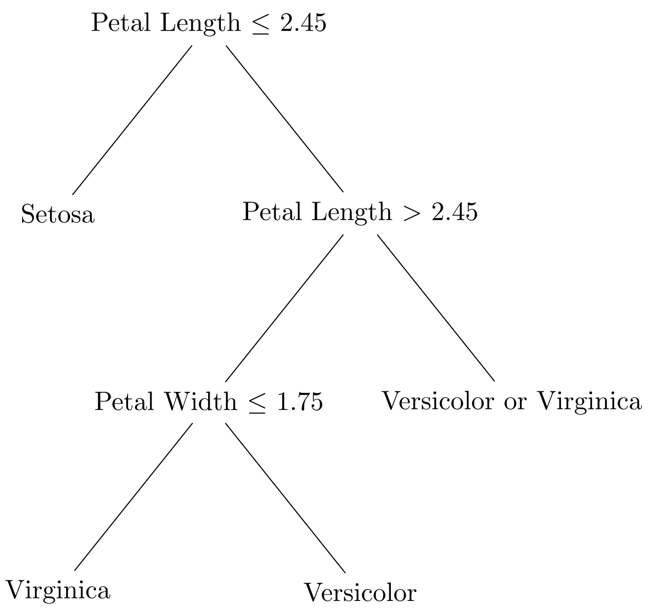
2.1 General Steps to Utilize and Train the Model

1. Load the dataset.
2. Split the dataset into training and testing sets.
3. Train the model.
4. Evaluate the model performance.

In this exercise, we are using the **Decision Tree Classifier** algorithm.

2.2 How it Works

The algorithm tries to find the best way to split the data at each step based on the features. It looks for the splits that maximize the separation based on a feature. The best split is decided by **Entropy**, which is a metric used for information gain. **Note:** A low entropy value indicates that the data in that split is homogeneous, which is desired in a classification task.



3 Definition of AI

- **Artificial Intelligence:** Intelligence exhibited by machines.
- **High-Profile Applications:**
  - Advanced web search engines
  - Recommendation systems
  - Virtual assistants
  - Autonomous driving
  - Generative and creative tools
- **Machine Learning:** A subset of AI that focuses on algorithms that allow computers to learn from data.
- **Deep Learning:** A subset of Machine Learning that utilizes artificial neural networks with multiple layers.

4 ML with Big Data

1. **Observed Input-Output Mapping:** Observe some outputs $y_1, ..., y_T$ corresponding to input feature vectors $x_1, ..., x_T$.
2. **Prediction:** A predictive model that takes in the input vectors and outputs a predictive value.
3. **Supervised Learning:** Learning an input-output mapping based on labeled data.
4. **Unsupervised Learning:** Learning patterns from data without labeled outputs.
5. **Features:** The inputs.
6. **Training:** The act of using data to find the best model.
7. **Loss Function:** A function that assesses model quality, such as Mean Squared Error (MSE):

   $$L = \frac{1}{T}\sum_{i=1}^{T}(y_i - \hat{y}_i)^2 \tag{1}$$

   For classification, a common loss function is Cross-Entropy Loss:

   $$L = -\sum_{i=1}^{T}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] \tag{2}$$

5 Two-Layer Neural Network

A neural network aims to learn an underlying function from input data to output predictions. The basic formulation involves learning a function:

$$y_i = f(x_i) + z_i$$

where z_i represents noise.

For example, fitting a polynomial of degree k:

$$f(x, \theta) = \sum_{|\alpha| \le k} \theta_\alpha x^\alpha$$

where α represents a multi-index over feature dimensions.

To evaluate model quality, we define the expected risk:

$$R(\theta) = \mathbb{E}\left[(f(x) - f(x, \theta))^2\right]$$

and the empirical risk based on observed data:

$$R_n(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i, \theta))^2.$$

5.1 Simple Neural Network Model

A simple neural network approximates the function using multiple neurons:

$$\hat{f}(x, \theta) = \frac{1}{N} \sum_{i=1}^{N} \sigma(x, \theta_i)$$

where:

- N is the number of neurons (hidden units),
- $\sigma(x, \theta_i)$ is the activation function applied to each neuron,
- θ represents the network parameters.

Common activation functions include:

- **Sigmoid:** $\sigma(x) = \frac{1}{1+e^{-x}}$
- **ReLU:** $\sigma(x) = \max(0, x)$

5.2 Stochastic Gradient Descent

To minimize the empirical risk $R_n(\theta)$, we use Stochastic Gradient Descent (SGD):

$$\theta \leftarrow \theta - \eta \nabla_\theta l(y_i, x_i, \theta)$$

where η is the learning rate and $l(y_i, x_i, \theta)$ is the loss function for a single data point. This iterative update gradually optimizes the model parameters.

Excess Risk

:

$$L(\hat{\theta}) - \inf_{\theta \epsilon \theta} L(\theta) \leq \frac{c}{n} + O(\frac{1}{n})$$

where: C: depends on the questions(constant) O(1/n): rate of decreasing

Theorem

- $$\sqrt{n}(\hat{\theta} - \theta_{real}) = O_p(1) \rightarrow \text{bounded in probability} P(\|\sqrt{n}(\hat{\theta} - \theta_{real})\|_2 > M) < \epsilon$$
- $$\sqrt{n}(\hat{\theta} - \theta_{real}) \rightarrow^d N(0,)$$
- $$\sqrt{n}(\hat{\theta} - \theta_{real}) = O_p(1) \rightarrow \text{bounded in probability} P(\|\sqrt{n}(\hat{\theta} - \theta_{real})\|_2 > M) < \epsilon$$

Parts of a Neural Network

1. Layers
2. Activation Functions
3. Loss Functions
4. Neural Network

Layers

Layer: is a collection of neurons(nodes) that process data at a particular stage in the network. Consists of Input Layer, Hidden Layer, and Output Layer.

Abstract Layer

This class serves as an abstract base class for all layers in the network.

- **Forward Pass:** Computes the ouput of the layer given an input
- **Backward Pass:** Computes the gradients with respect to the input and parameters
- **Step:** Updates the layer parameters(weights and biases)

Linear Layers

A layer of neural network, that performs a linear transformation on the input.

$$y = x \cdot W + b$$

- **self.w**: Represents the weight matrix of shape (in_dim, out_dim), initialized using small random values.
- **self.b**: Bias vector of shape (1, out_dim), initialized to zeros.
- **self.dw** & **self.db**: Store the computed gradients of weights and biases during backpropagation.

Forward Pass:

$$\mathbf{out} = \mathbf{inp} \cdot W + b$$

- **inp**: Input matrix of shape $(\texttt{batch\_size}, \texttt{in\_dim})$
- **self.w**: Weight matrix $\mathbf{W} \in \mathbb{R}^{\texttt{in\_dim} \times \texttt{out\_dim}}$
- **self.b**: Bias **vector** of shape $(1, \texttt{out\_dim})$
- **OUTPUT**: Matrix of shape $(\texttt{batch\_size}, \texttt{out\_dim})$

Backward Pass:

Computes the gradients needed for updating the weights and biases. Given the upstream gradient up_grad, we compute:

- **Gradient w.r.t weights** ($\frac{\partial L}{\partial W}$):

  $$\frac{\partial L}{\partial W} = X^T \cdot \text{up\_grad}$$

  where:

  - X represents the input from the previous layer.
  - up_grad is the gradient passed from the next layer.
  - X^T (transpose of X) ensures proper matrix dimensions.

- **Gradient w.r.t bias** ($\frac{\partial L}{\partial b}$):

  $$\frac{\partial L}{\partial b} = \sum \text{up\_grad}$$

  Since the bias does not depend on input values, its gradient is simply the sum of all elements in up_grad.

- **Gradient w.r.t input** ($\frac{\partial L}{\partial X}$) to be propagated to previous layers:

  $$\frac{\partial L}{\partial X} = \text{up\_grad} \cdot W^T$$

These gradients are used to update the model parameters using gradient descent:

$$W = W - \eta \frac{\partial L}{\partial W}$$

where η is the learning rate.

Step Method Updates the weigths and biases using the computed gradients and learning rate(lr).

$$W = W - lr \cdot \frac{\partial L}{\partial W}$$

$$b = b - lr \cdot \frac{\partial L}{\partial W}$$

Activation Functions

Sigmoid

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

- This function squashes the input into the range [0,1] making it useful for binary classification tasks
- Converts any real-values number into probability like input
- It may cause vanishing gradients due to its flat slope for extreme values in deeper networks

ReLU(Rectified Linear Unit)

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

- Ouputs 0 if the input is less than zero, otherwise returns the input itself
- Helps introduce non-linearity into the model, essential for complex learning algorithms
- Helps avoid vanishing gradient problem, common in deep networks with Sigmoid activation
- During backpropagation only the gradients for inputs greater than 0 pass through:

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Softmax

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- Normalizes the input values into probabilities that sum to 1
- Used in the final layer of a neural network for multi-class classification
- Converts raw scores into probabilites where each class has a non-negative probability between 0 and 1
- Subtracting the maximum input value (np.max(inp)) from all inputs before applying np.exp helps prevent overflow errors.

Loss Functions

Cross Entropy Loss

$$L = -\frac{1}{N}\sum_i \sum_c y_{ic} log(p_{ic})$$

Typically used in classification tasks since it measures the dissimilarity between the true distrbution(target) and the predicted probability distribution(prediction).

Mean Squared Error(MSE)

$$L = \frac{1}{N}\sum_i (p_i - y_i)^2$$

Is primarily used for regression tasks, where you need to measure the distance between the predicted, continuous values and true values.